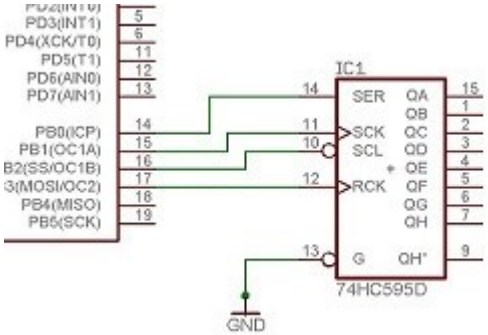
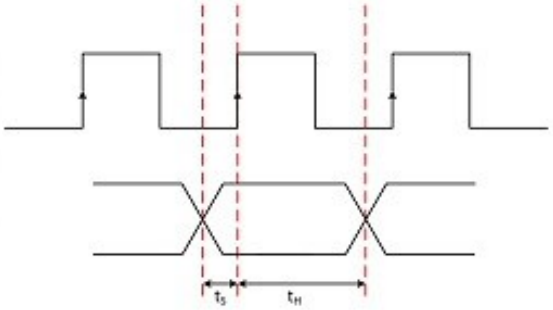
Foliengröße 4:3

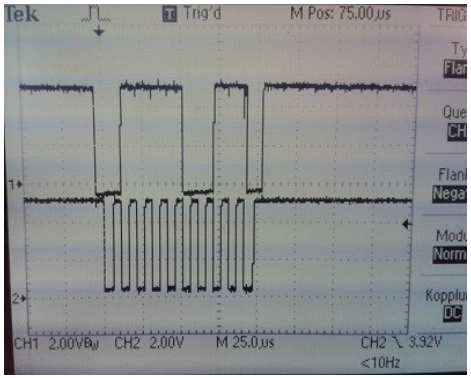
Mikrocontroller Crash Kurs CE

auch als PDF unter „MikroController_Kursskript_CE.pdf“

ANSI-C89 Programmierkonzepte, prof. Softwarestrukturen,
externe Elektronik und In-Out Aktoren,
externer ICs, Sensoren, Servo, Step & Linear Motoren etc....
Skript Version: 2.2f







Erstellt von **Christof Ermer** *in Eigenregie*
Christof.Ermer@ur.de https://homepages.uni-regensburg.de/~erc24492/

Status: Sammlung von, mit der Zeit gewachsenen, Arbeitsblätter.
Brauchbar, kleinere Fehler, Inkonsistenzen möglich.
Ergänzungen und Reihenfolgen-Änderungen jederzeit möglich.
Diese Seiten sind als Lehrwerk mit Erklärung zu verstehen.
Aber auch zum autodidaktischem Selbststudium.

3f.01.24
Seitenzahl: 857

Dieses Skript ist eine historisch gewachsene Themensammlung. Gedacht für einen 14 Tage Gruppen Seminar Unterricht an Universitäten oder Gruppenseminaren und „Quick and dirty“.
Es erklärt den 8 Bit Mikrocontroller der Familie AVR (jetzt Microchip) ATMegaxxx, speziell **ATMega328p,** wie im „Arduino UNO“ verbaut.

Dieses Arbeitsskript ist auch für den Selbstunterricht geeignet.
Vorteil: Beamer taugliche postkartengroße Texte erschlagen einen nicht mit zu viel Informationen auf einmal.
Manche Autoren wollen zeigen, wie toll Sie sind. Diesen Vorwurf kann man mir nicht machen. Daher die nackte Präsentation und nicht immer perfekt. Es soll auch kein lektoriertes Buch sein und es ist keine Designschönheit. Nehmt es wie es ist.

Quell Dateien verwendet im Kurses 2020 - (Ohne Fehlerkorrektur) für die WINAVR Entwicklungsumgebung. Übertragbar in Arduino Sketch .
Download Link: http://gofile.me/4eKLF/cua3nvqz3
000_Kurs2020_ATMega328p_Progs.zip

Christof Ermer – Regensburg - Germany

Anwendung:

Wegen mancher **berechtigter**, dann seltsamer, unsinnigen bis inquisitorischer Kritik:
Es ist primär als **PowerPoint Beamer** Version **mit Erklärung** gedacht.
PDF ist daher oft ungünstig formatiert. (...Kritik von außen)

Es fehlt ein Inhaltsverzeichnis. Warum**?** Weil die Reihenfolge sich immer noch dauernd ändert, neues dazukommt. Und es von der Vorbildung abhängt, was interessiert und was, wann, wo gebraucht wird. Es ist ein dynamisches Werk. Kein Buch!
Also, einfach **mit STRG+F nach Themen suchen**, die einem interessieren.
….oder einfach durchblättern und die vielschichtigen Themen ausprobieren.

Zielgruppe: Es ist für **Neulinge**, **Ungeübt**e oder leicht **Vorgebildete** gedacht.
….nicht für Erbsenzähler, die sowieso alles besser wissen und anders machen.
Merke: ***Mein Kurs, mein Stil, meine Regeln.*** *Punkt.*

Ich verwende die „polnische Notation“ in der Programmierung, weil ich es so will und gut finde. Codebeispiele sind ab und zu inkonsistent formatiert. Das liegt daran das manches sehr alt, anderes neu ist. Oder der Platz pro Seite genutzt werden sollte.

Die vielen Bilder, im Vergleich zu Anderen Werken, sollen die Ödnis mancher Lehrbücher vertreiben helfen. Rechtschreibfehler, dort und da, sind möglich. Der Inhalt zählt.
Meine Poesie, voll Legasthenie.

Dieser Kurs ist immer in Wandlung, Erweiterung, Korrektur usw. Gedacht für einen erklärenden Seminar-Vortrag, auch weitgehend für das Selbststudium geeignet.
Aber weit entfernt, fertig oder perfekt zu sein.
Da mit PowerPoint erstellt, gibt es auch ab und zu Formatierungsfehler, wie Gänsefüßchen oben oder unten. Oder gemeine **Autokorrektur** Patzer und Rechtschreibfehler meinerseits.
Das macht es jedoch nicht falsch.
1. Wer es besser weiß, braucht meinen Kurs dann gar nicht!
2. Mein Kurs, meine Regeln, (Leroy Jethro Gibbs, NCIS)

Nicht verschweige ich, dass ich einen sehr gut gemachten Kurs im Netz gefunden habe: http://stefanfrings.de/index.html. Ein Blick hinein lohnt sich. Googeln hilft.
Christof Ermer – Regensburg 06.07.2021

Es gibt nur 3 Feinde des Programmierers:
Sonnenlicht, Frischluft und das unerträgliche Gebrüll der Vögel.

Theorie ist wenn du alles weißt aber nichts funktioniert.
Praxis ist wenn alles funktioniert aber niemand weiß warum.
Bei Programmieren ist Praxis und Theorie vereint.
Nichts funktioniert und niemand weiß warum.

Folgende Seiten sind für einen mindestens **2 wöchigen** (10Tages-) Ganztages Unterricht/Vortrag mit Erklärung und Beispielen gedacht.

In Eigenregie muss jedoch zusätzlich, gelerntes geübt werden.

„Selber haptisch erarbeitet“, ist niemals durch reine Theorie zu ersetzen

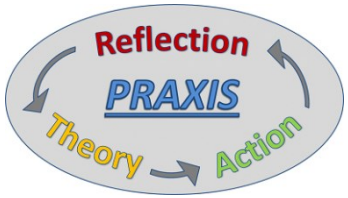
Wegen der Fülle des Stoffes reicht, neben dem Vortrag die Zeit NICHT aus.

µController sind nicht trivial.

Beschrieben wird die ‚Bit nahe‘

Mikrocontroller Programmierung mit ANSI-C(89)

Speziell. Software Konzepte in C + AVR typisches



ANSI-C(89) deshalb, weil es ideale Methoden zur Bitmanipulation bietet und auch ansonsten professionell ausgestattet ist.

Es sollte jedoch jederzeit ein Selbststudium möglich sein.

Auch als Nachschlagwerk mit Begriffssuche „Strg + F“ ist dieses Mikrocontroller Lehrwerk zu verwenden.

Bitte etwas Geduld, denn µController lernen… ist ein Instrument lernen.

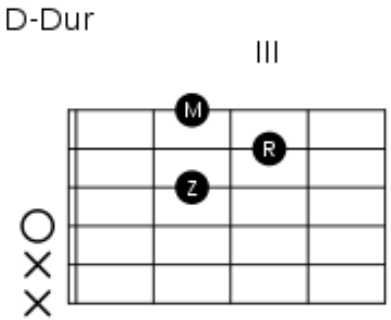
Theorie ohne Praxis ist sinnlos.

Praxis ohne Theorie ist unmöglich.

Ich sagte mal (vor Mathematikern):
„Mathematik ist ganz nett, aber sinnlos….wenn man diese nicht anwendet“

So ist es mit Mikrocontrollern.

Der schnelle Weg, ist der verführerische Weg, der leichte Weg. Er führt zur dunklen Seite der Macht. (Meister Yoda)

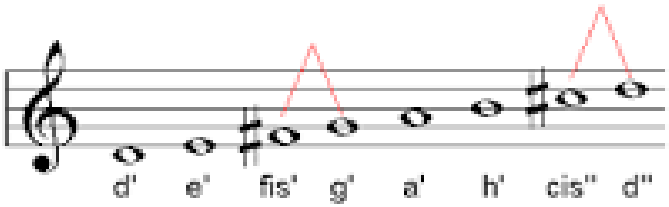


Bsp. Gitarre kann man lernen, in dem man „schnell“ Griffe, genaue Griffpositionen lernt. Also wo die Finger hin gehören. Es klingt. ….man weiß nur nicht, **warum!.**

Lernt man, dass ein Akkord aus Prime, Terz und Quinte usw. besteht, wo die Noten stehen usw., dann kennt man den inneren Aufbau eines Akkordes. Nur, …wenn man diesen Akkord nicht auf einem Instrument spielt. **erklingt er nicht**. Reines Notenlesen ist wie Nachplappern. Wenn auch auf hohem Niveau. **Wir jedoch wollen Komponieren und spielen lernen**. Wenn man nicht Techniken wie ‚Harmonielehre‘ kennt, wie und warum man verschiedene Akkorde aneinanderreiht, wird man auch nicht professionell Komponieren oder eben „P**rogrammieren**“.

Merke:
Wissen macht noch keine Musik, sondern ambitionierte Hingabe und die eigene Erfahrung.

Übrigens: Normal ist…..
zu jedem schön gemalten Bild gibt es anfänglich sehr viele misslungene Bilder.

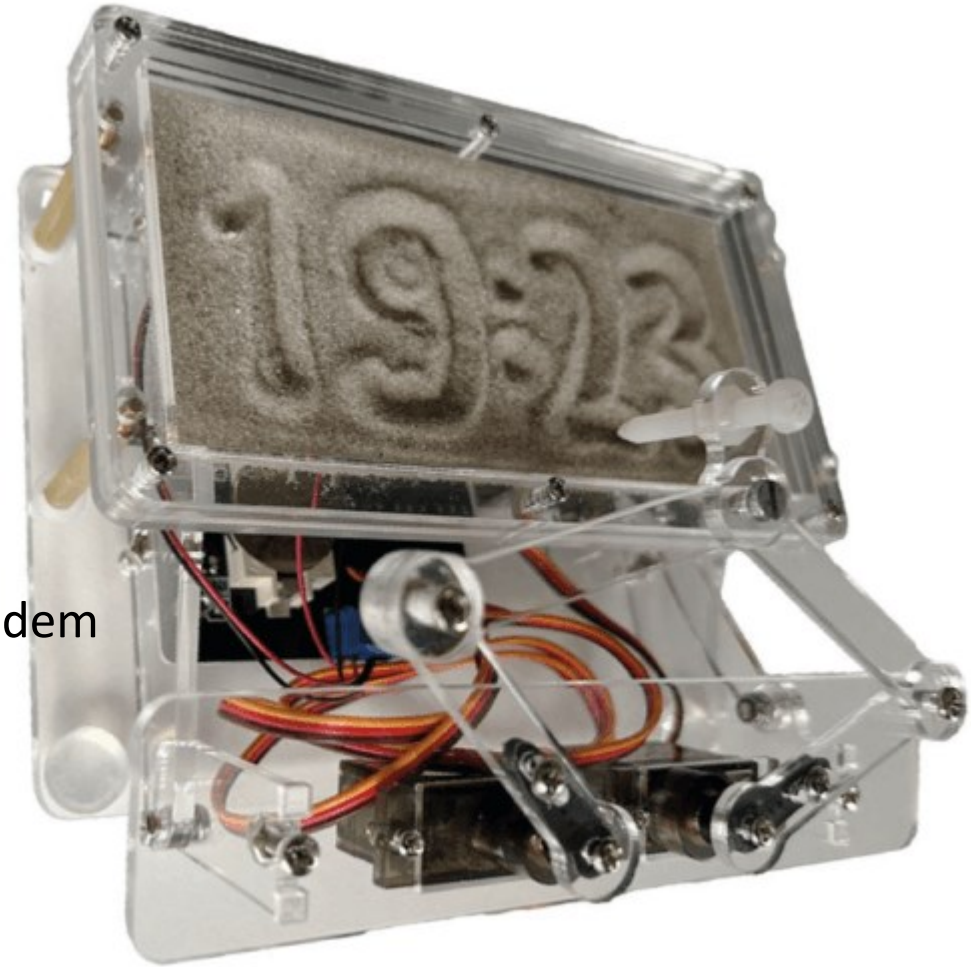


Ziel ist es, zum Bsp. Sowas selbst bauen zu können.

Sand Uhr

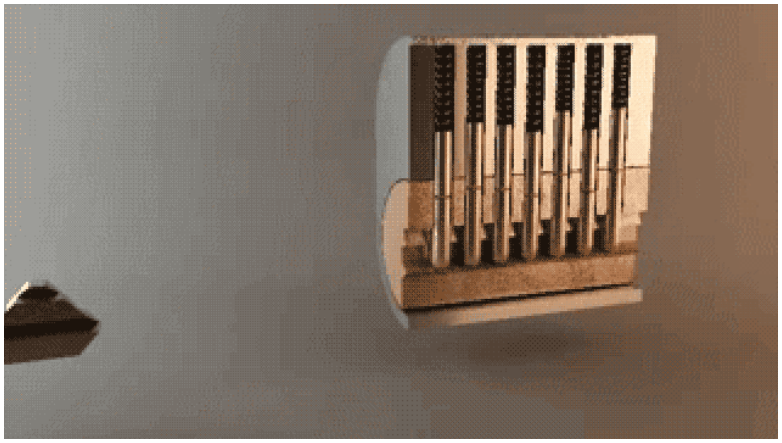
Mechanik
Sägen, Schrauben,
Servo Motoren,
Schritt Motore, Rüttler Motor,
Dazu muss man verstehen, wie
Servomotoren funktionieren

Microkontroller Programme
schreiben, die diese steuern.
Echtzeit Uhr Chip auslesen,
und diesen Chip vorher stellen.
Text Kommunikation via PC mit dem
Mikrokontroller usw.



Mikrocontroller zu Programmieren ist ein kreativer Vorgang.
Zugleich verschmelzen Fachgebiete, wie Elektronik, Mathematik, Physik.

Wenn man es richtig macht,
passt alles gut zueinander.



..und → **Es ist ‚fast schon' eine Berufsausbildung**, eine Qualifikation.

Ich hab es schon öfter erlebt, dass nach dem Studium gut dotierte Stellen gerade deshalb erreicht wurden, weil man Mikrocontroller gut versteht und mit diesem Know How auch sehr bald mit anderen Typen, wie **ARMxx**, arbeiten kann.

If you want to learn anything, don't try to learn everything

Grundlage ist allerdings ein rudimentäres Verständnis für diskrete elektronische Komponenten wie
Widerstände, Kondensatoren , Dioden, **Transistoren**.
Anspruchsvoll, →Ja, aber hier hilft Wikipedia und Googele etc. weiter.
Es gibt zahlreiche Lehrwerke dazu, und es ist nicht trivial.

Hier jedoch geht es um das Innenleben im Mikrocontroller, wie:
Input-Output Ports, Register, Timer, ADC, I2C-Bus usw.

Dazu behandle ich natürlich die bekannten Aktoren und Sensoren.
Die Datenbusse und Methoden, die Schnittstellen
zu externen Bausteinen ermöglichen, möchte ich verstärkt behandeln.

Das schöne ist, es wird überall mit Wasser
gekocht,
oder…..noch eine Weisheit:
„kaum macht man es richtig, schon geht es“.

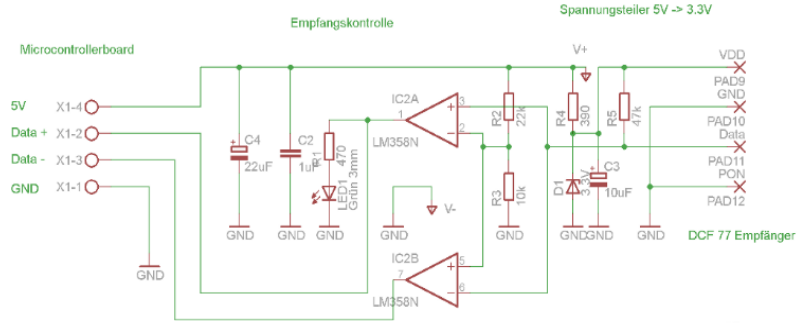
Im Skript sind immer wieder einzelne Seiten mit dem Wort Spickzettel beschrieben.
Diese Seiten sind genau dies. Zusammenfassungen etc..
Es kann sich lohnen sich diese nach Bedarf einzeln auszudrucken oder zu markieren und somit bereit liegen zu haben.
So wird der Anfang beim Programmieren erleichtert, bis man sich an manche ungewohnten Formulierungen gewöhnt hat.

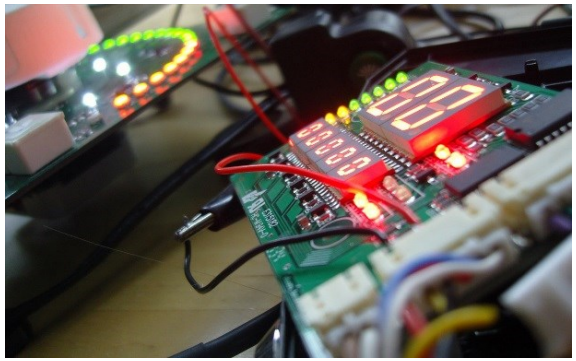
Es ist leider, *nicht nur für Anfänger*, ein anspruchsvoller Wirrwarr aus verschiedenen Fachgebieten
und Themen.

Doch einige Methoden haben sich bewährt und sind eigentlich maschinenunabhängig.
.

Überschneidung der Fachgebiete

Etwas „**Multiexperte**“ zu sein, …. von diversen Fachgebieten zumindest eine ‚Vorstellung‘ zu haben, ist erforderlich und in der Natur der Sache.





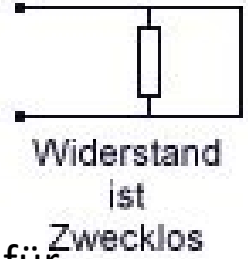
Ziel dieses Skriptes ist es, die Methoden kennenzulernen die man braucht um mit µControllern professionell arbeiten zu können.
Die Grenzen: ***Es ist unmöglich umfassend ALLES zu erklären. Aber es dürfte reichen, die vielen Forenbeiträge und Artikel und Ideen zu verstehen und selbst zu erarbeiten.***

Mit diesem Handwerkszeug können wir in einer universellen Sprache wie „C{++}“ Programme schreiben, die zertifizierungstauglich sind und zuverlässig arbeiten.

Ich möchte euch ein **mächtiges Werkzeug** in die Hand geben:

- um den Programm-Fluss bzw. Ablauf organisieren zu können.
- den Status, sowie die Situation des Programmes zu jeder Zeit zu kennen
- auf Ereignisse (Interrupts etc.) sofort zu reagieren.
- die Außenwelt mit elektrischen Signalen von Ports zu steuern oder abzufragen.
- einzelne Bits gezielt zu manipulieren, zu löschen, setzen, abzufragen….

So kann ein „**einzelnes Bit**“ für jeden gewünschten Zweck genutzt werden
Das zeichnet **effektiv**e, **platzsparend**e und **logisch** in sich **stimmige Software** aus.



Dabei werde ich so „**bit-nahe“** wie möglich bleiben.
Also auf **innerer Mikrocontroller und Registerebene,** und somit recht h**ardwarenahe,** bleiben. → *Was heißt das?*

Das heißt, ich verwende **keine nebulösen *Bibliotheken, Libraries*** etc. die es für **ARDUINO** und andere Plattformen gibt.
Diese „Libs“ machen das anfängliche Arbeiten zwar leichter, aber man lernt **nichts**, versteht nichts vom inneren Geschehen im **µC.**
Man wird zudem „abhängig“ vom Können der Bibliotheken.

Erkenntnis:
Tatsächlich ist der verwendete Mikrocontroller gar nicht so entscheidend.
Es geht vielmehr um Strukturen, und diese gelten immer und bei allen Maschinen.

Der Mikrocontroller-Typ bestimmt ‚nur‘ die **Spezialnamen** für die verwendeten „Register“ und die inneren Komponenten wie Ports, Timer, ADC, Interrupts usw.
Diese heißen evtl. **PORTA DDRA PINA TCNT2 TCCR2 INT0 TIMSK**

An diese Kürzel gewöhnt man sich recht schnell, da man lernt, diese „sprechend“ zu benutzen. So wird aus **TIMSK**: „**T**imer **i**nterrupt **m**a**sk** register“
Die Methoden die ich behandele gelten dagegen universal!
Sie sind eher **Ansi-C** spezifisch.

Zuerst ein kleiner Rundumblick:
(kann übersprungen werden)

~so ab Seite 40..50 geht es langsam los…

µController sind allgegenwärtig

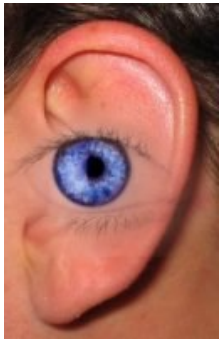
EINGABE → Tastatur,
Folie oder mit Sensoren

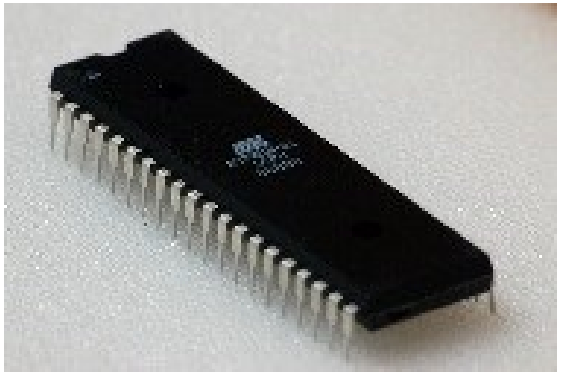
Interne mathematikfähige
Datenverarbeitung.

AUSGABE der Daten via Display, Datenbus
Seriell, I²C, Ethernet.
CAN Bus
eBus (Seriell mit Supply)
Speichermedien,
oder **Aktoren** wie Servos, Motoren,
Signalports etc.
Drucker

Tankstelle

Sensor - Mikrocontroller - Aktor





Mit dem Mikrocontroller kann man billige wie leistungsfähige Geräte bauen. Vom Heizungsregler bis SAT-Receiver usw.

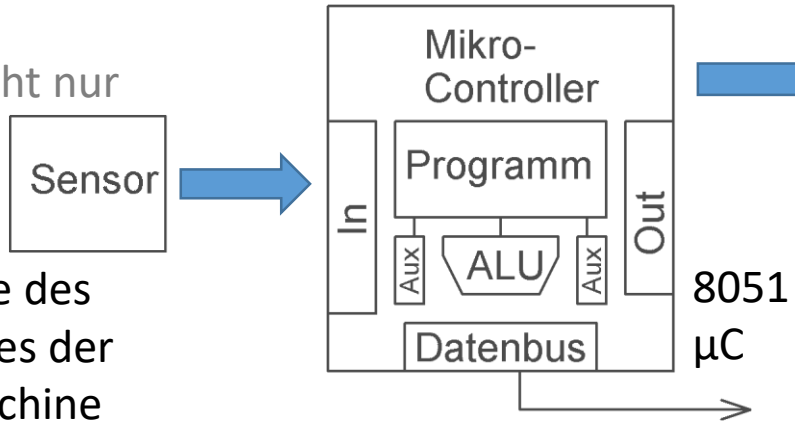
Ein *Heizungsregler* ist ein gutes Beispiel.
Ein **Sensor** liefert einem **mathematisch anspruchsvollen Programm** (**PID**-Regelung) im **µC** die Daten, um einen **Aktor** (Getriebemotor) zu steuern.
Via Tastatur, einem Drehgeber und Temperatur Sensoren wird gesteuert. Über einen Stellmotor wird das Regelergebnis realisiert
Ein LCD-Display gibt Auskunft über den Status.

Bemerkung: Das Ganze kostet heute ~15€

Reale Anwendung in der Industrie

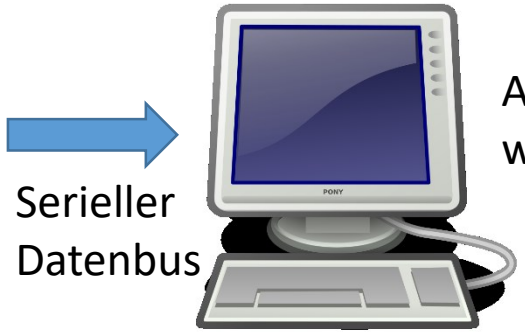
Problem: In der Fertigung war dem Prozessverantwortlichen nicht klar, warum es (auffällig) zu Verzögerungen und wenig effektiven Auslastungen der Produktionsmaschinen kommt . Daher wurde ein Betriebsdaten Erfassungskonzept entwickelt.

Dies diente nicht nur der Kontrolle der Abläufe.



Einfache Eingabe des Betriebszustandes der Produktionsmaschine (von den Arbeitern ungeliebt)

Beispiel eines Betriebsdaten Erfassungsterminals, Was mal gebaut habe, aus der industrie der 80er, worin ein **8051** Mikroprozessor werkelt. Ein **RS485 BUS Netzwerk** lieferte die Daten von der Vorort-Maschine zur Auswertungszentrale.



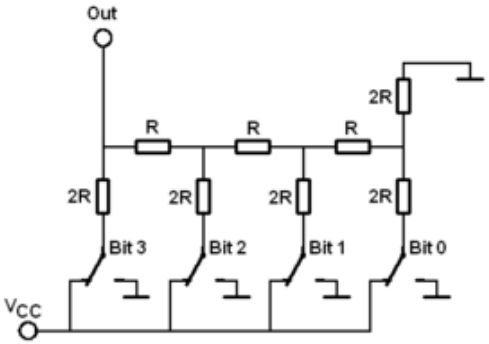
Serieller Datenbus

Aus-
wertung

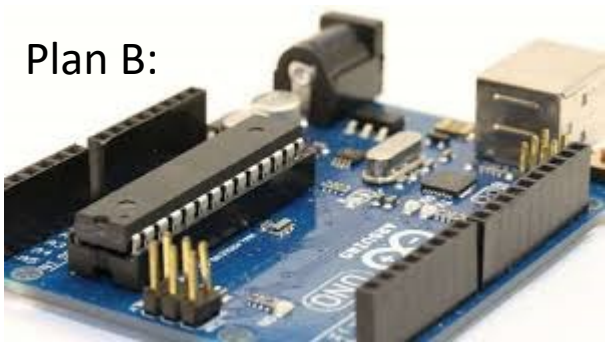
Software ersetzt Hardware

Mikrocontroller erledigen Jobs, die früher **diskrete Hardware** erledigen musste. Mann kann sich also einen Mikrocontroller auch wie eine, **per Software realisierte, austauschbare**, Hardware vorstellen.

Know How: DAC Prinzip mit R2R

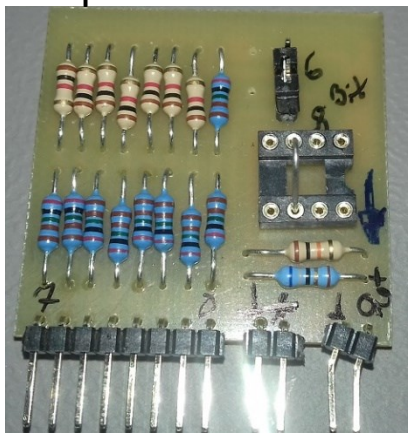


Plan B:

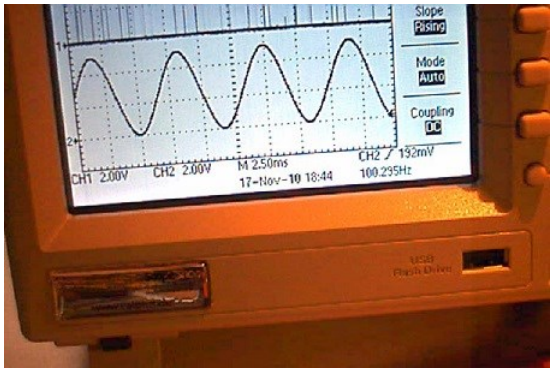


Sinus Software
erzeugt Wertetabelle.
Ein Timer gibt diese am Port aus

Billiges R2R Netzwerk aus simplen Widerständen

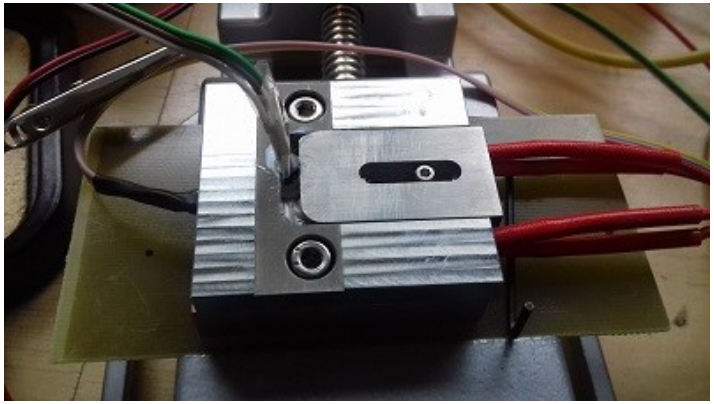


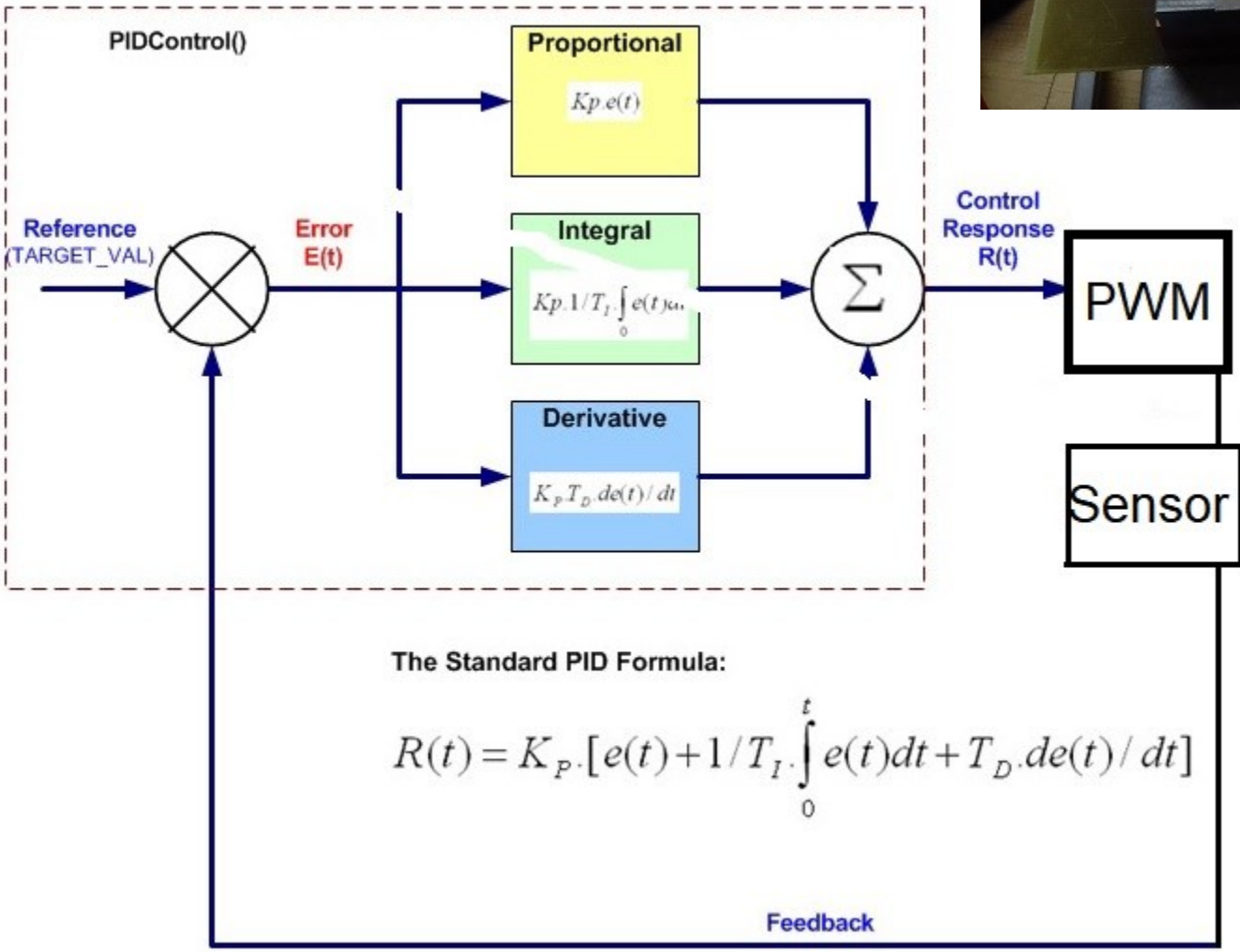
Exzellentes Ergebnis

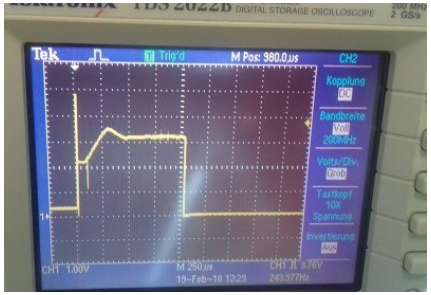


Value = 128+127*sin(2*PI* *{0..255}*/256.0);

Mikrocontroller erlauben mathematische, komplexe Prozesse, wie z.B. eine PID Temperaturregelung





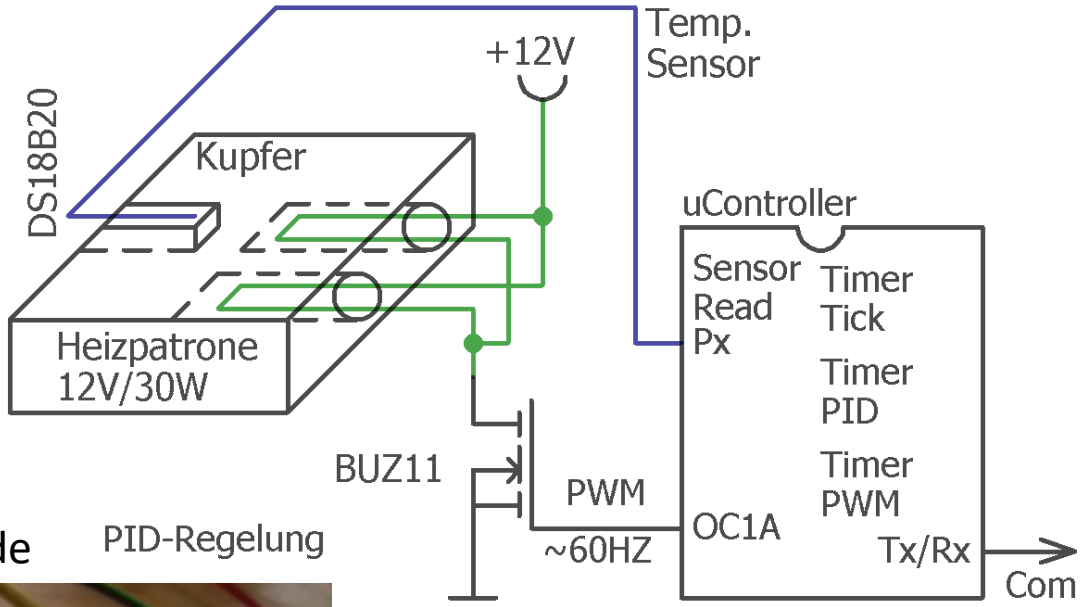


……..dazu muss man sich dann doch etwas einarbeiten. Man lernt es durch ‚machen‘.

Aktive, auf mathematische Konzepte basierende **Temperaturregelung**. Realisierbar durch Mikrocontroller.

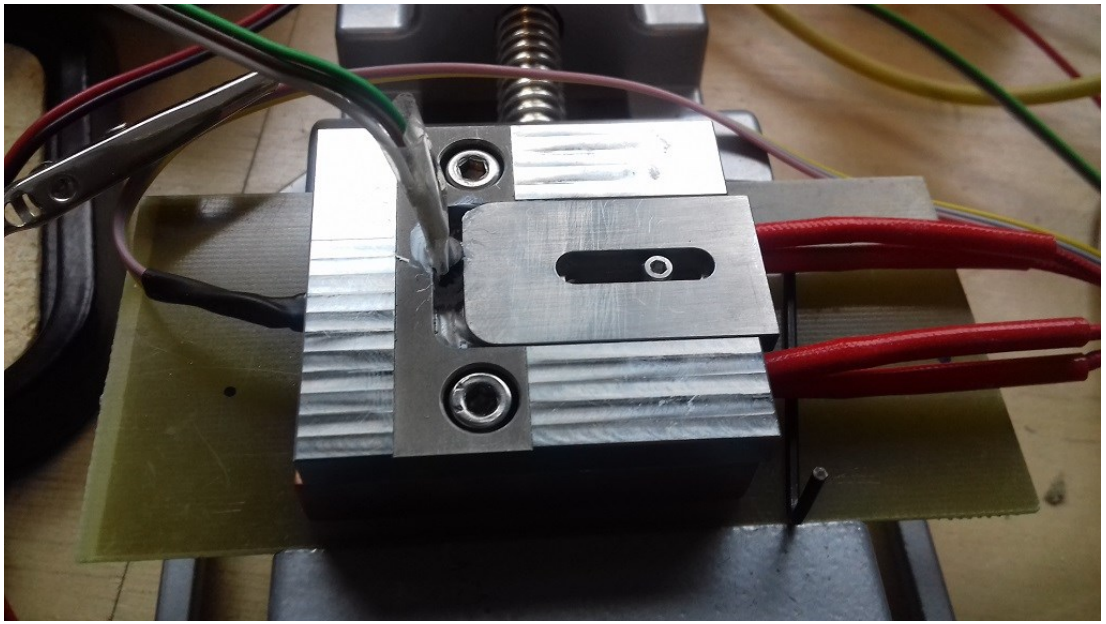
Hier ein sogenannte **PID Regelung** =

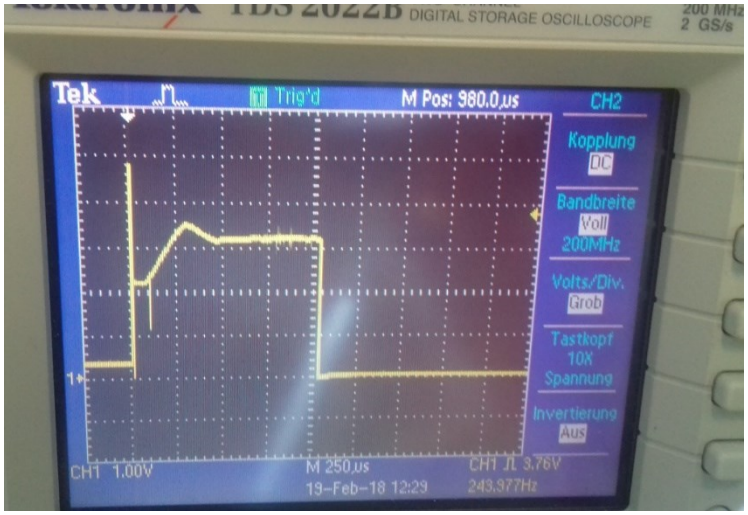
Proportionale
Integrations- und
Differentiations **Regelung**.



Hier: Probenheizung mit 80W Heizelement und Temperaturmesssonde

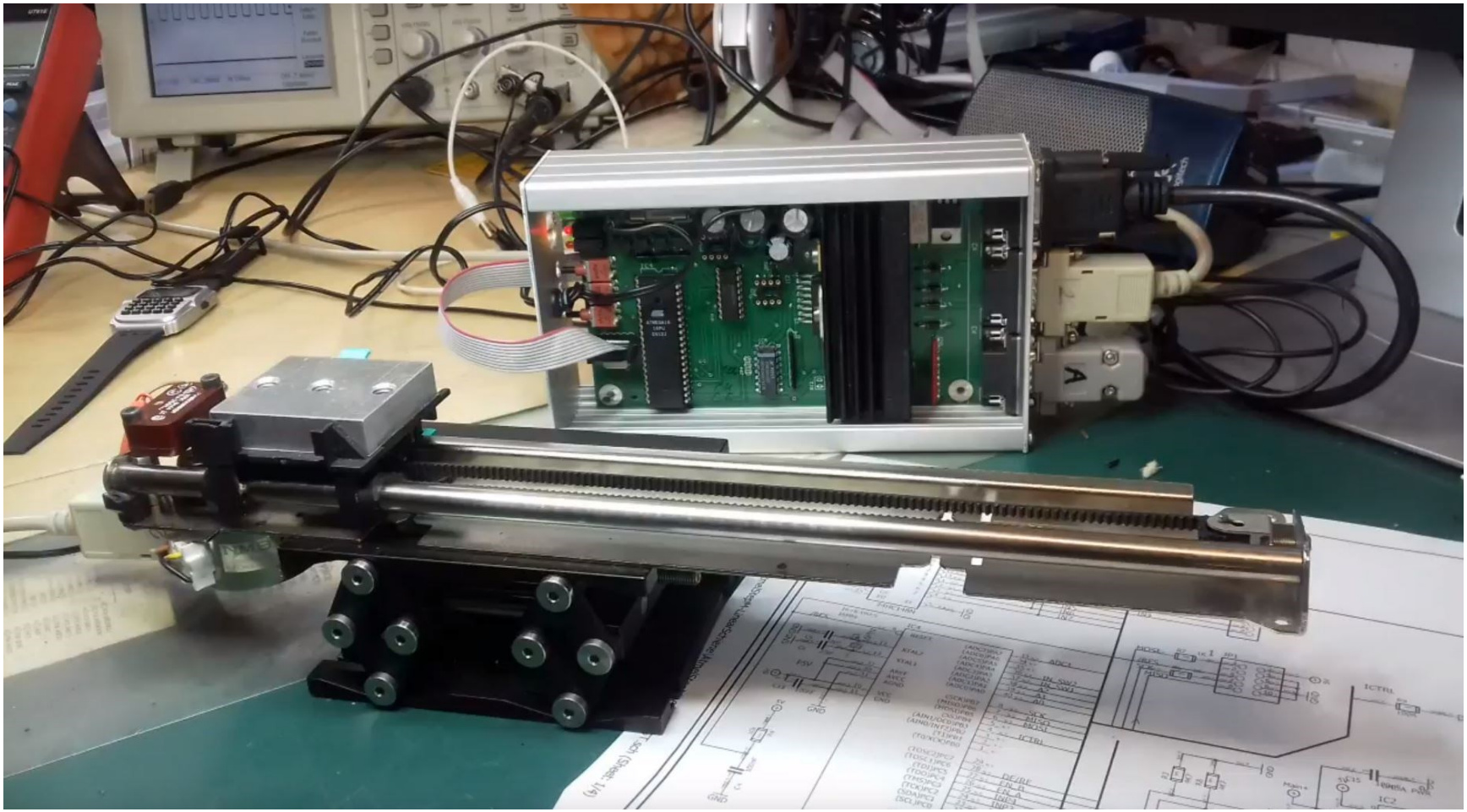
PID-Regelung





Wir basteln:

Aus dem realen Leben: Optiklabor Schrittmotor Linearantrieb, realisiert mit einem alten Nadeldruckerantrieb.

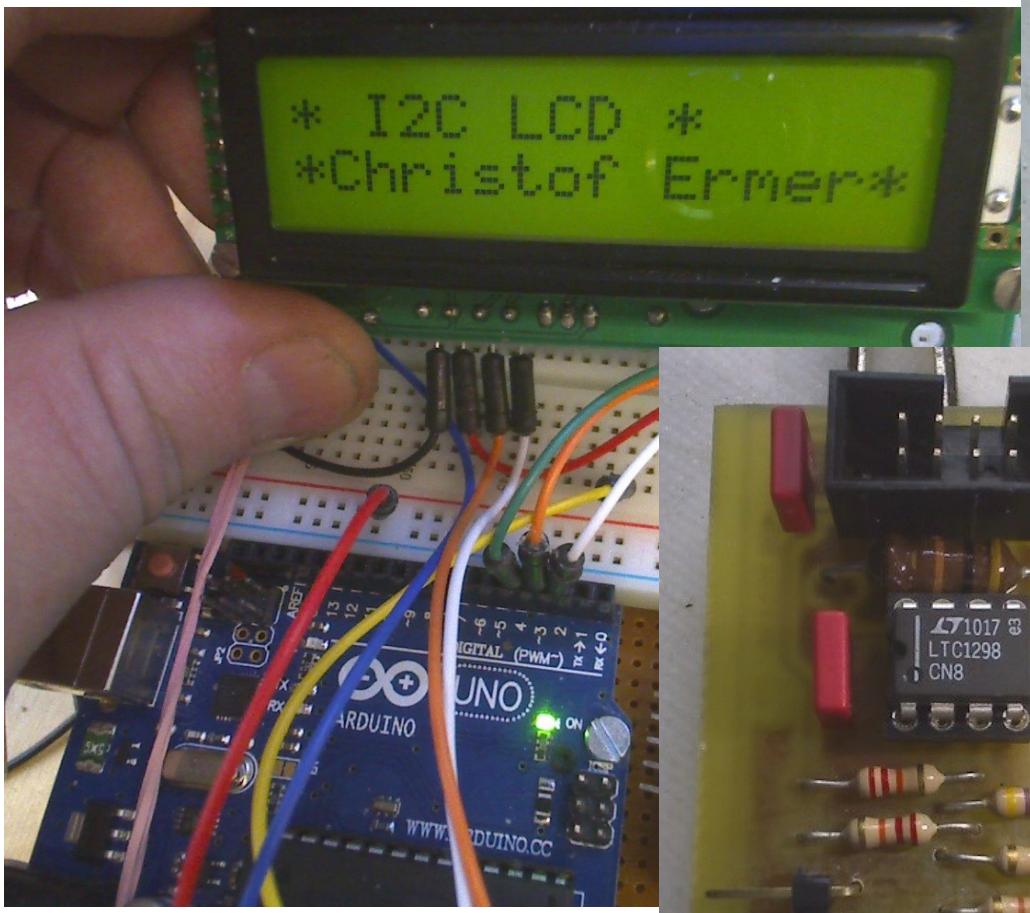


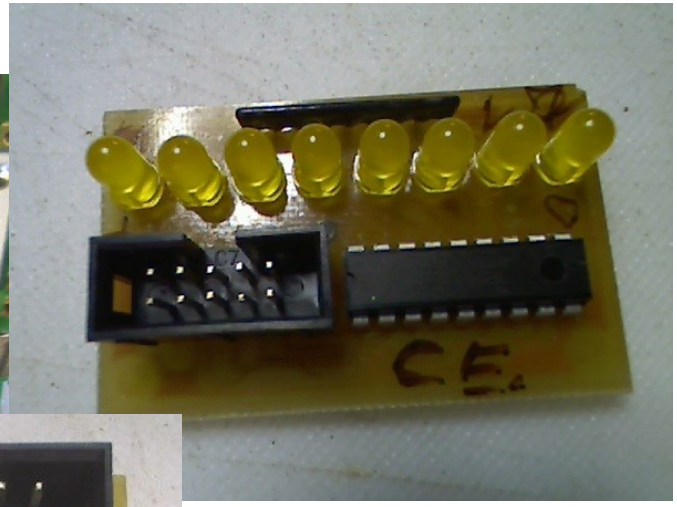
http://homepages.uni-regensburg.de/~erc24492/Labor-Projekte/Labor-Projekte.html

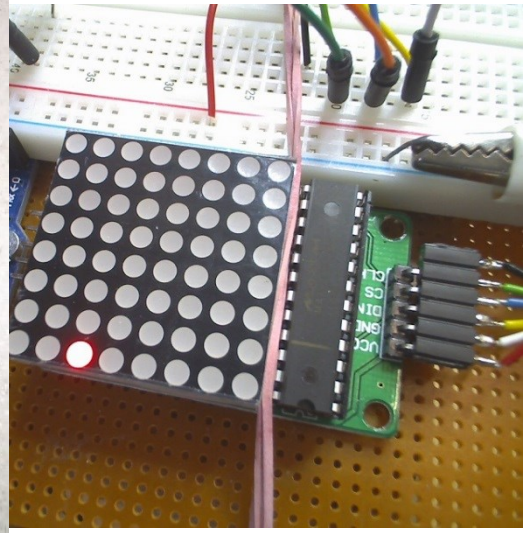
Youtube dazu: https://youtu.be/s_6w92g4vhc

µC steuert unterschiedlichste externe Komponenten

Youtube: http://youtu.be/KoqKPxGFp78

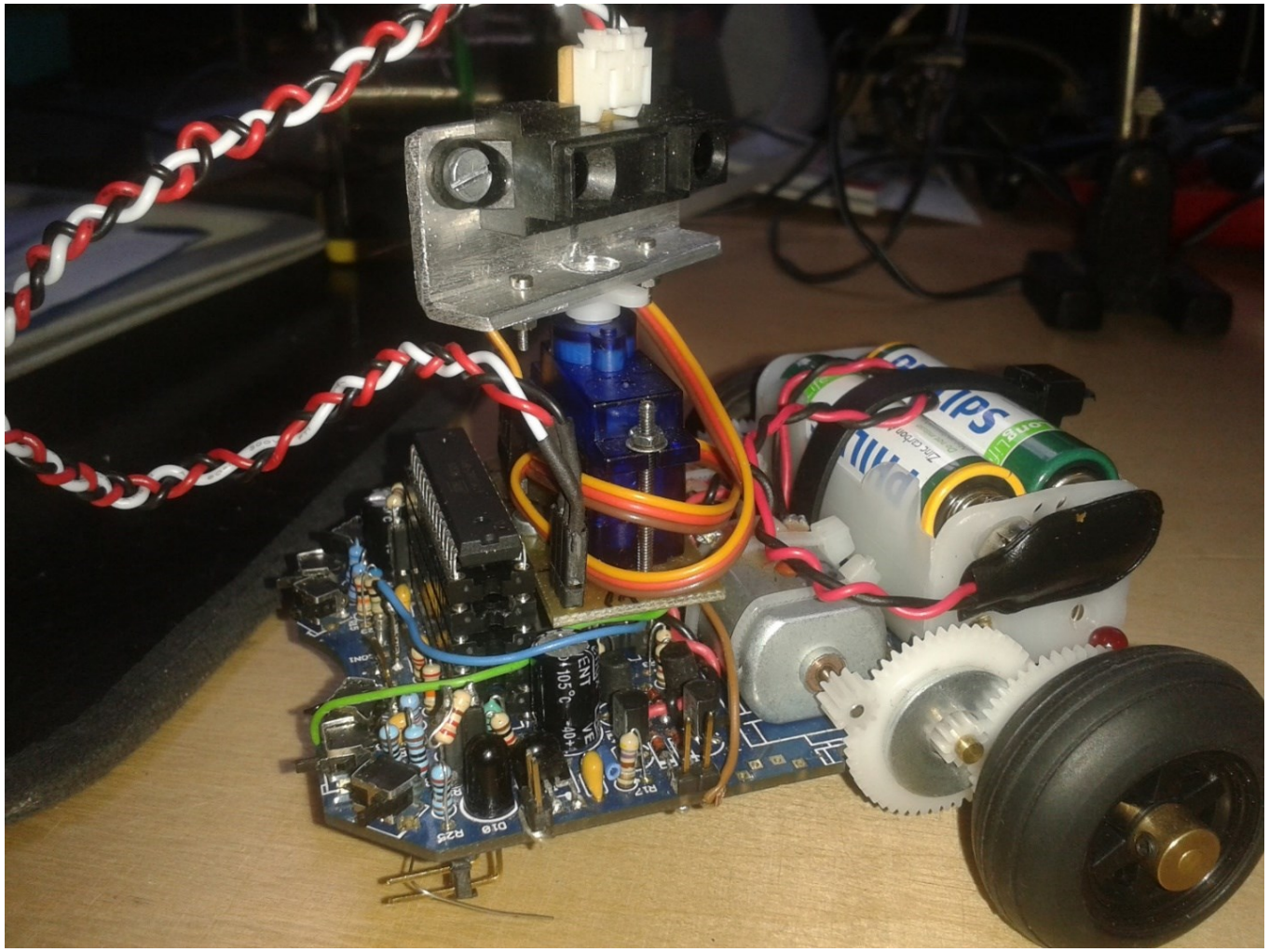






Beispiel aus dem MINT Projekt 2014 DLR Asuro Robot mit ATMega8 **µC**ontroller.
Ein Servo Antrieb (blau) dreht einen IR-Entfernungsmesser wie ein Infrarot Radar.
(Bild aus dem MINT Projekt 2014)

Youtube Link: http://youtu.be/dstx46pDZzQ

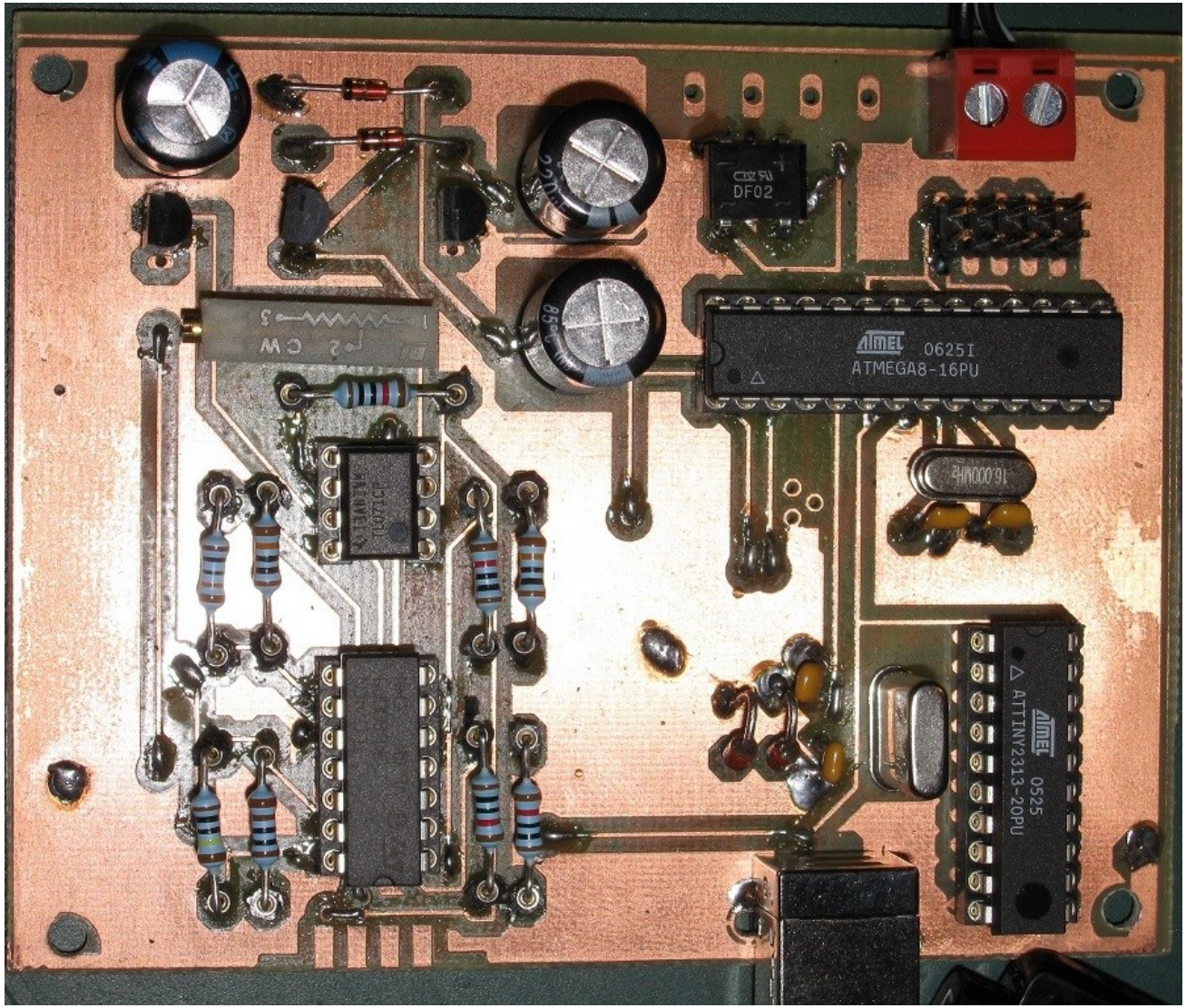


Komponeten:

ATMega8
Servo Motor
Infrarot
Abstands-
senor
2 Linear
Motoren
Mit
Drehsensor
Infrarot ISP
Sensor-Taster
(Schalter)

Typische Selbstbauplatine.

2µC Lösung. 1er Arbeitet, der 2. simuliert eine USB Schnittstelle



USB

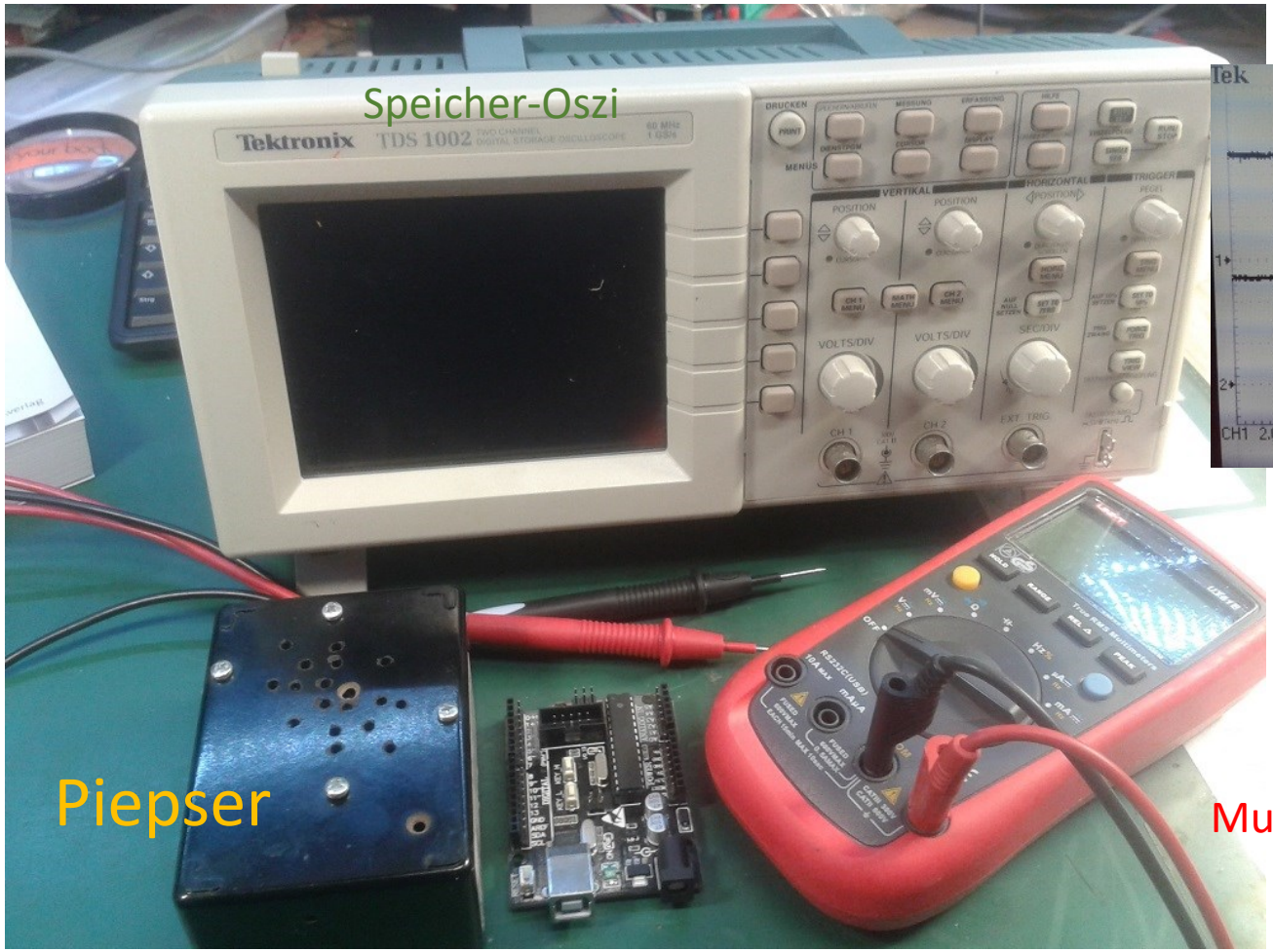
Mit Eagle gezeichnet, belichtet, geätzt, gebohrt. Manuelle Durchkontaktierung.

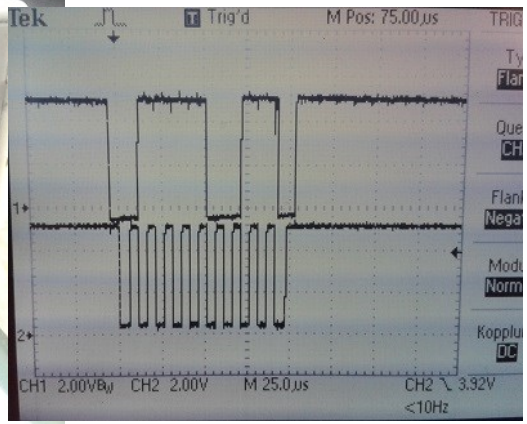
Große Bauteile sind leichter zu verarbeiten. Die Miniaturisierung hat hier seine Grenzen. Meist wird mit 80er Jahre Stil DIL Gehäusen gearbeitet.
SOx Gehäuse und SMD 1206, 0805 geht aber auch sehr gut.

Gut erkennbar sind die Standardkomponenten. µCs, Kondensator, Quarz, Widerstände, Operationsverstärker, Transistoren, usw.

Das Arbeitswerkzeug bleibt übersichtlich:

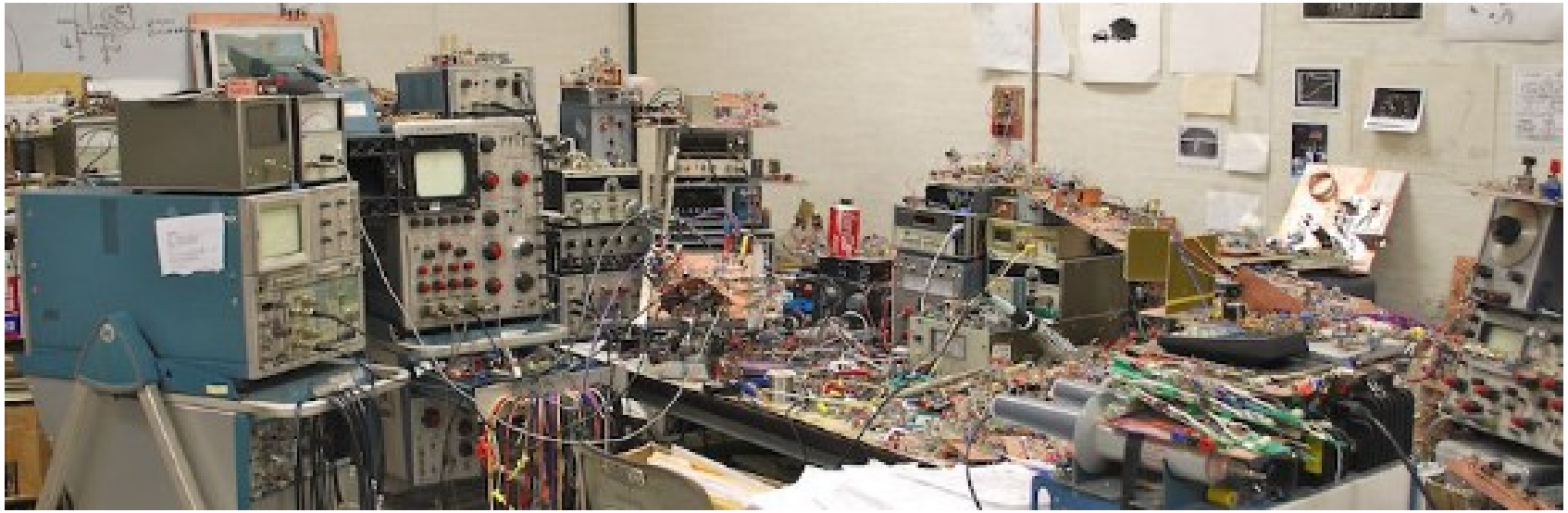
Neben **Zange, Schraubendreher** etc. brauchen wir nur ein billiges Multimeter, ein ‚weniger' billiges Speicher-Oszilloskop (~>300€) *wäre schön* und - fast am wichtigsten - einen Durchgangspiepser mit **+(rot)** / **-(schwarz)** Polung der Messleitungen und mit Tonhöhenänderung, je nach Mess-Situation und Bauteil.





Signal-Diagramm
Bsp. **I2C Bus**
oben: SDA =data
unten: SCL =clock

µController Labor Ausstattung

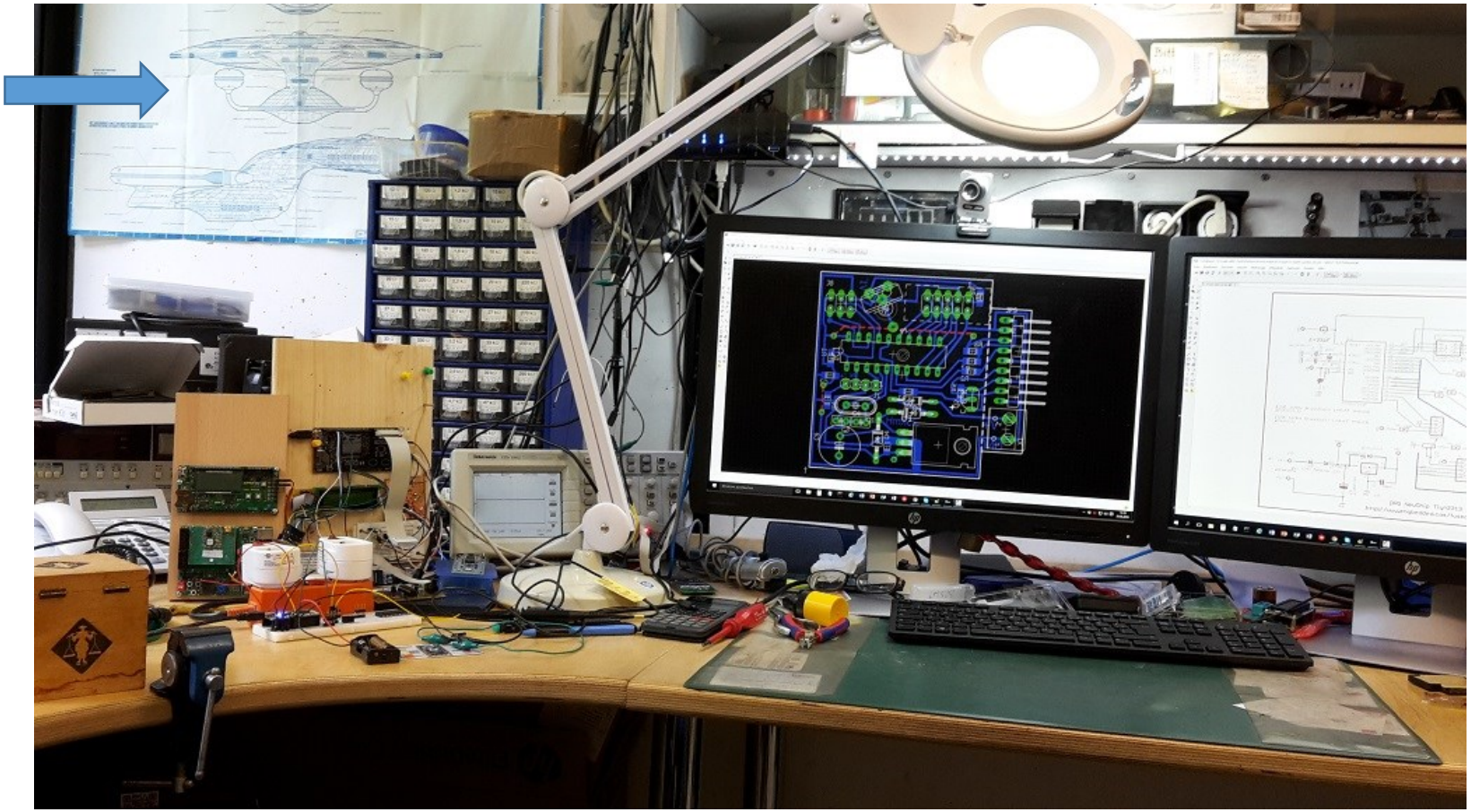


Ein ordentlicher Arbeitsplatz, übersichtliche Anordnung und Planung der Werkzeuge und ein gut sortiertes Teilelager erlauben erfolgreiches Arbeiten.
Ein sauberer Aufbau, sowie eine stringente Verdrahtung ermöglichen eine qualifizierte Fehleranalyse.
Präzise Dokumentationen runden das Ergebnis ab.

Praxisnaher, typischer Elektroniker Arbeitsplatz

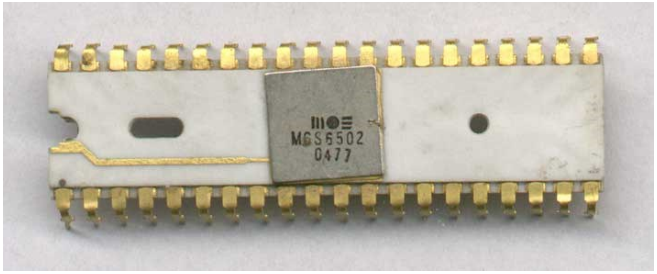
Bauteillager, Messgeräte, Werkzeug, Lötkolben, Stromversorgung, Oszilloskop, PC, Debugger, CAD-Layout Programm, usw. (Der Pirat, braucht auch noch Draht)

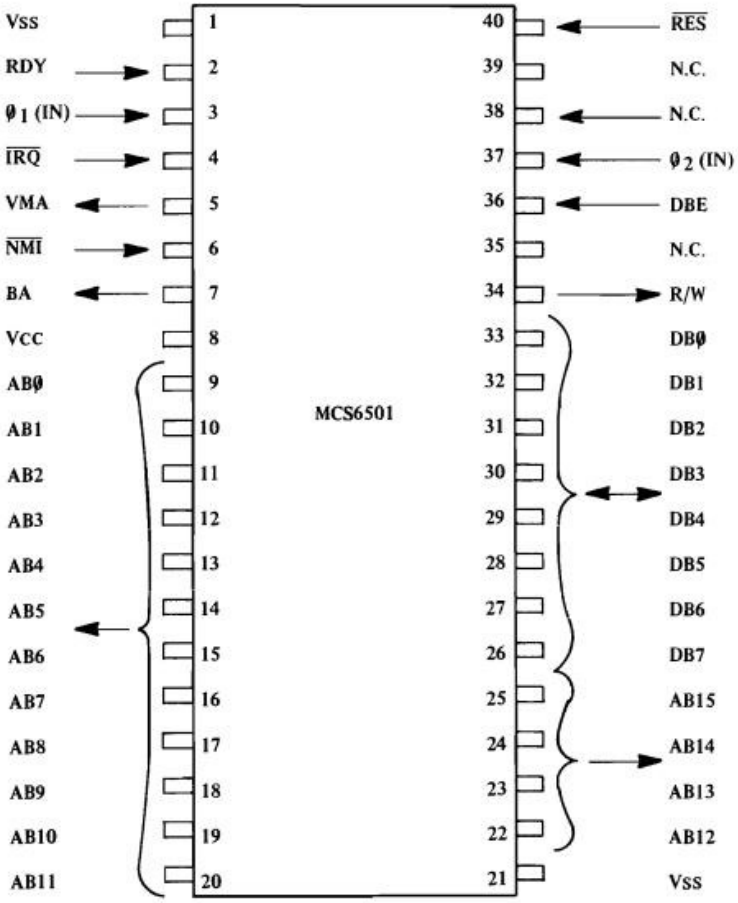
…die Kunst den Tisch dennoch frei zu halten.



Der Unterschied von einer CPU zu einem Mikrocontroller.

Hier hilft der Blick in die Vergangenheit, der 70er





N.C. = NO CONNECTION

Klassische CPUs, wie der **8Bit MOS 6502, Z80A ,** haben ein reduziertes Innenleben.
Die Peripherie Komponenten sind externe Bausteine.
CTC (Counter/Timer) , **RAM**, **ROM**, UART, PIO (Parallel In/Out) wie **8255**, sind *EXTERNE* umgebende, somit erweiterbare, Bausteine
Der µController dagegen beherbergt diese Komponenten direkt auf seinem Chip.
µCs sind 1-Chip Computer.

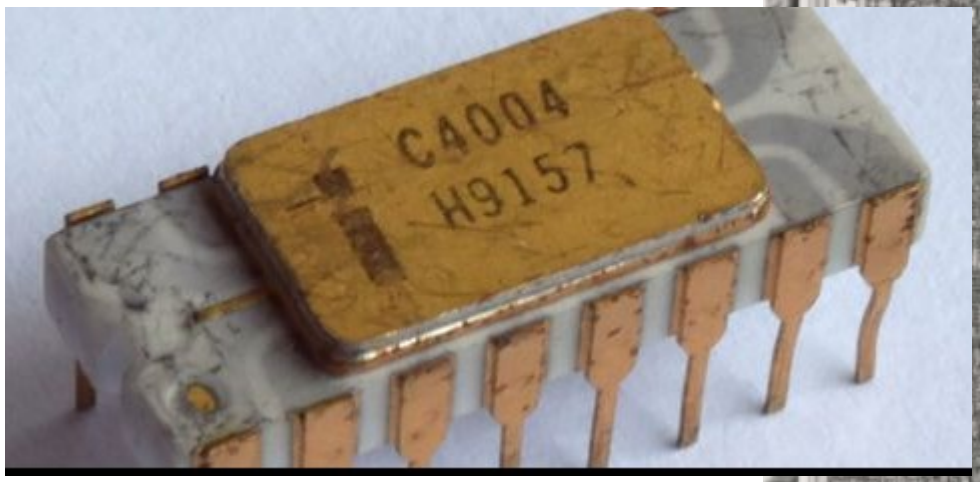
PINs:

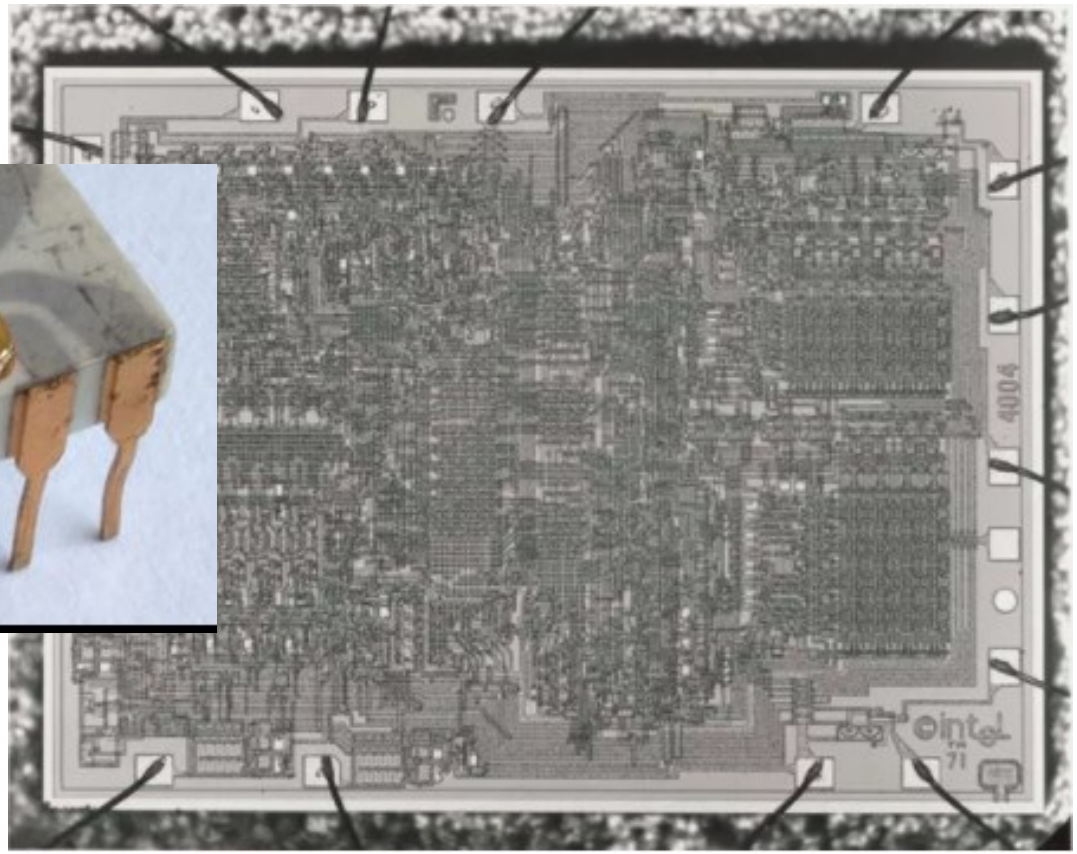
ALU= **A**rithmetik and **L**oigic **U**nit, eine 8/16Bit.
AB0..15 → Adressbus, DB0.8 → Datenbus
Dazu noch Steuerleitungen für das Signal-Bus Management, Controls und den Interrupts
/IRQ → „/“ bedeutet LOW-Aktive
/NMI → No maskable Interrupt,
/RES → Reset, /RDY → Ready,
Und andere PINs, wie VMA, BA, DBE……
(read your fucking manual)

Der 1. Mikroprozessor der Welt: Intel 4004 15.11.1971

4Bit - MOS-Silizium-Gate-Technologie

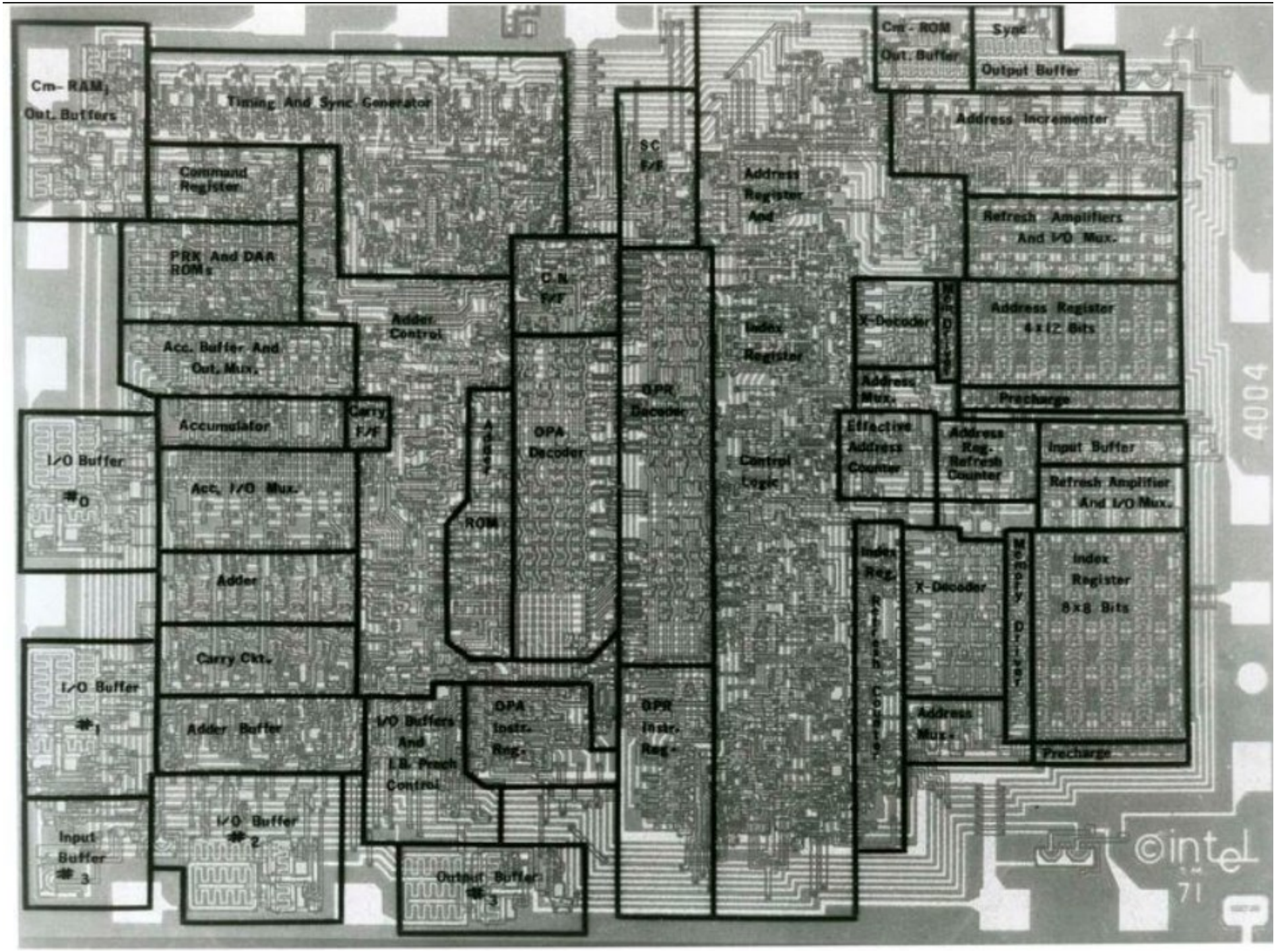
https://www.elektronikpraxis.vogel.de/der-mikroprozessor-feiert-seinen-50-geburtstag-a-1075172/?cmp=nl-95&uuid=



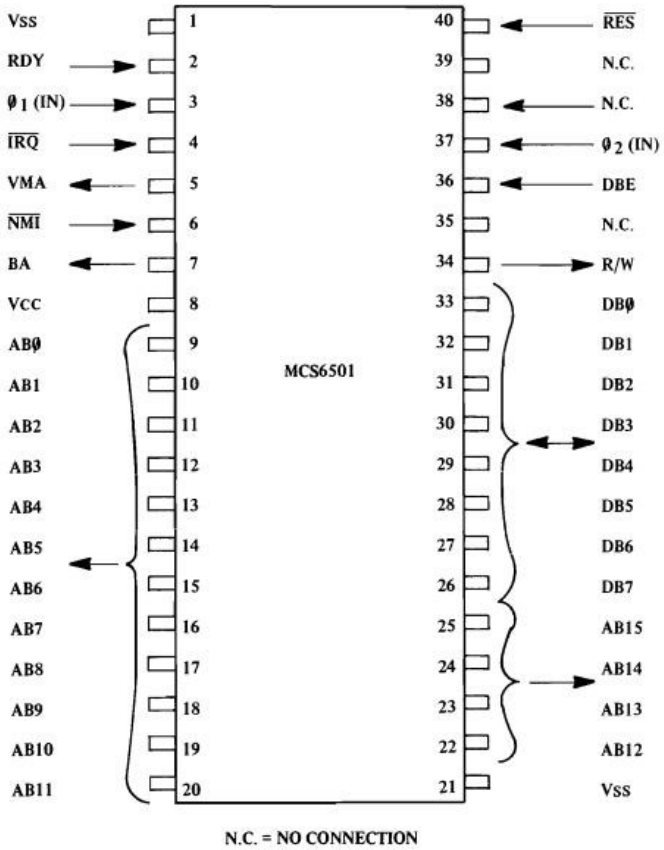


Er benötigte Vdd (V+15V), Vss (V-15V)

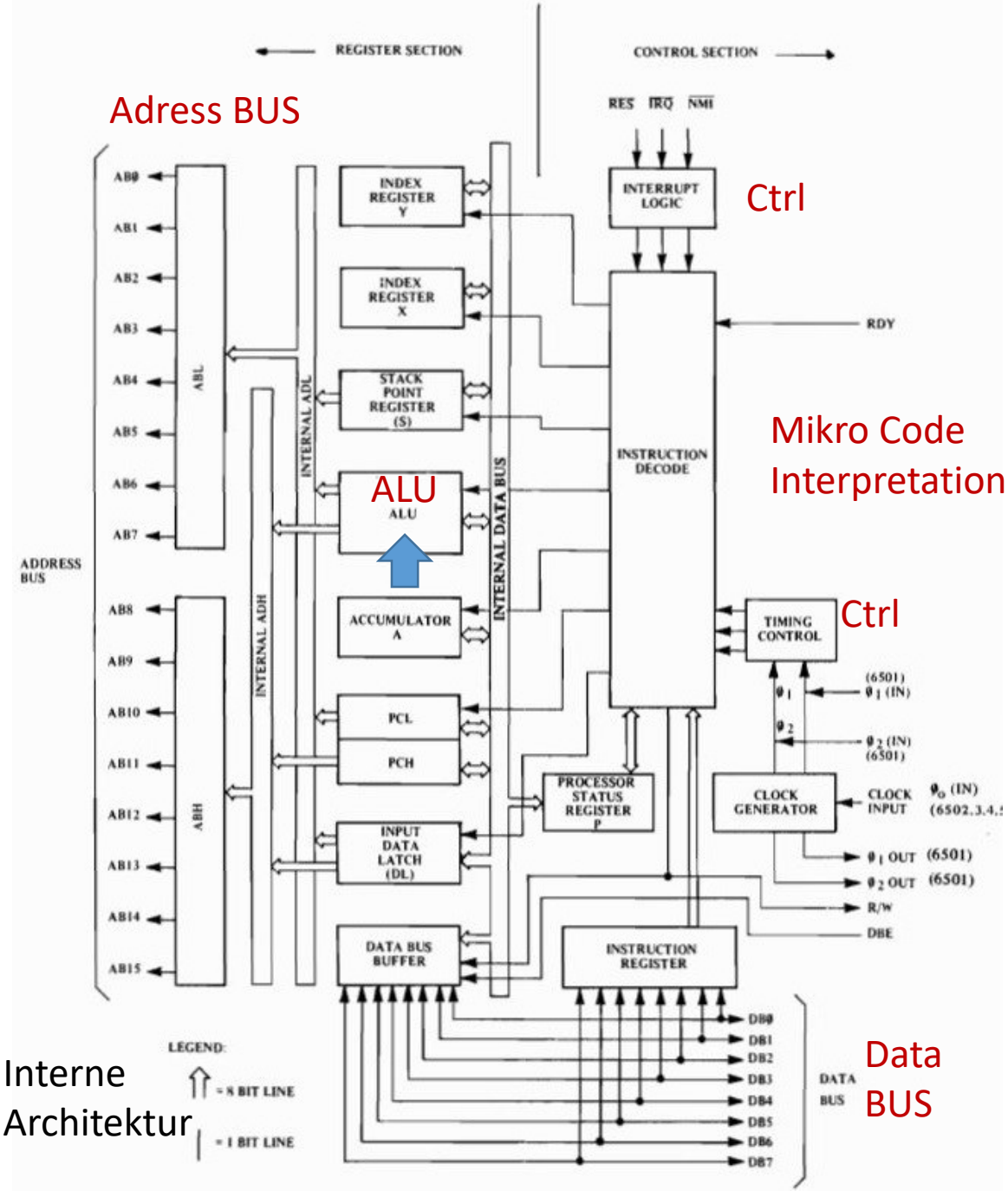
Datenblatt: http://www.applelogic.org/files/4004Data.pdf


Cm-RAM, Out. Buffers
Timing And Sync Generator
Command Register
PRK And DAA ROMs
Acc. Buffer And Out. Mux.
Accumulator
Carry F/F
Adder Control
C.N. F/F
SC F/F
Address Register And
Cm-ROM Out. Buffer
Sync Output Buffer
Address Incrementer
Refresh Amplifiers And I/O Mux.
X-Decoder
Memory Driver
Address Register 4x12 Bits
Address Mux.
Precharge
Index Register
GPR Decoder
Effective Address Counter
Address Reg. Refresh Counter
Input Buffer
Refresh Amplifier And I/O Mux.
I/O Buffer #0
Acc. I/O Mux.
Address ROM
OPA Decoder
Control Logic
Index Reg. Refresh Counter
X-Decoder
Memory Driver
Index Register 8x8 Bits
Adder
Carry Ckt.
I/O Buffer #1
Adder Buffer
I/O Buffers And I.B. Preset Control
OPA Instr. Reg.
GPR Instr. Reg.
Address Mux.
Precharge
Input Buffer #3
I/O Buffer #2
Output Buffer #3
4004
©intel 71

IC-Pinout



CPU → Kontrollsignale - Register-Innenleben. Adressen –Datenbus usw. Aber **ohne** Peripherie wie Speicher (RAM/ROM) etc. ist die CPU geradezu nackt.

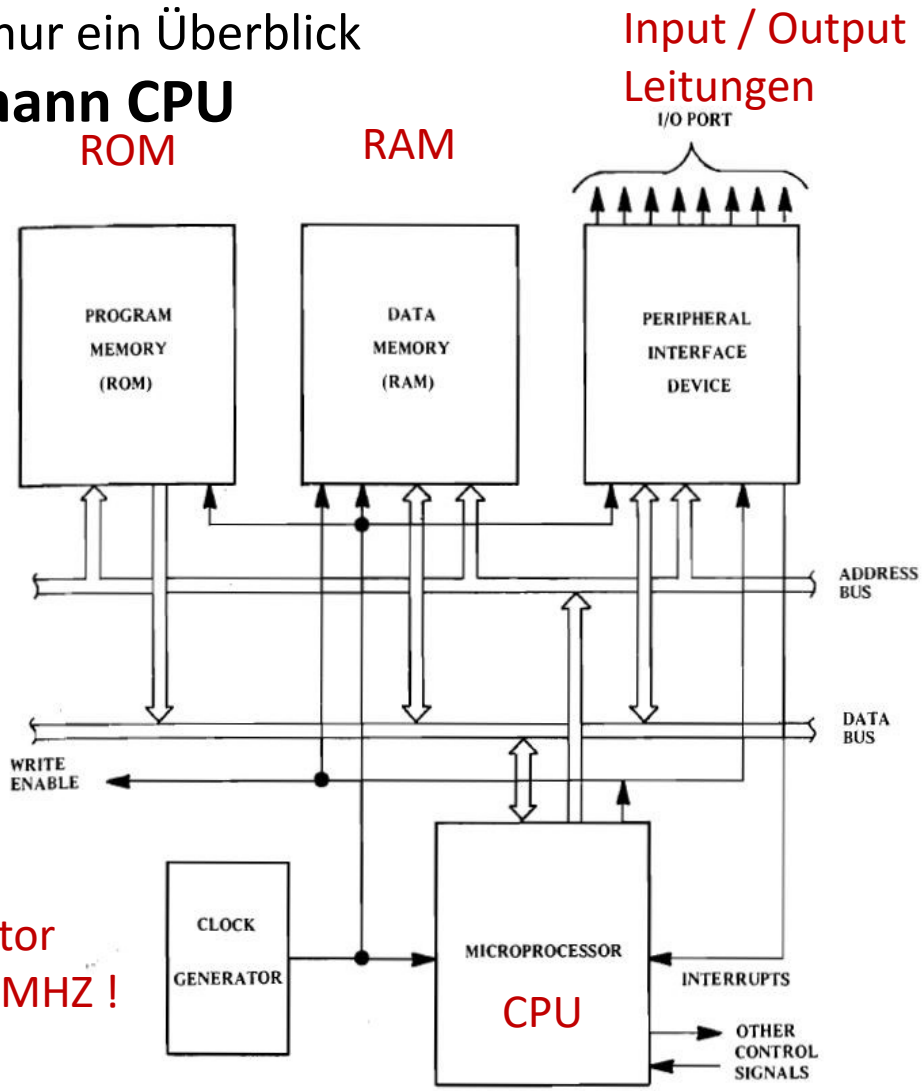


Interne Architektur

Architektur (nur ein Überblick

Von-Neumann CPU

mit typischer externer Umgebung



Adress Bus
16..32..64 Bit

Data Bus
8..16.32..64 Bit

Takt Generator
Quarz 1..20 MHZ !

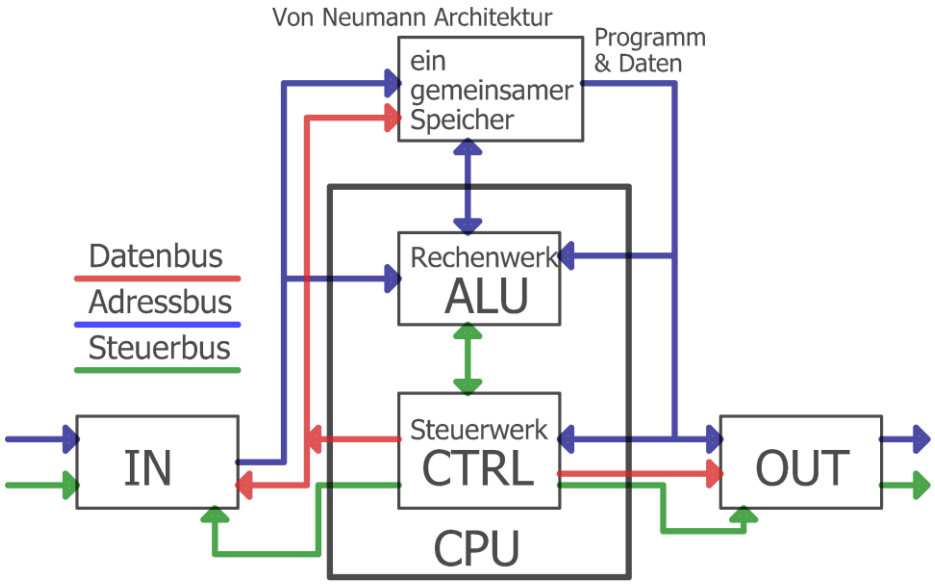
PC CPUs intern 2..>4 GHz

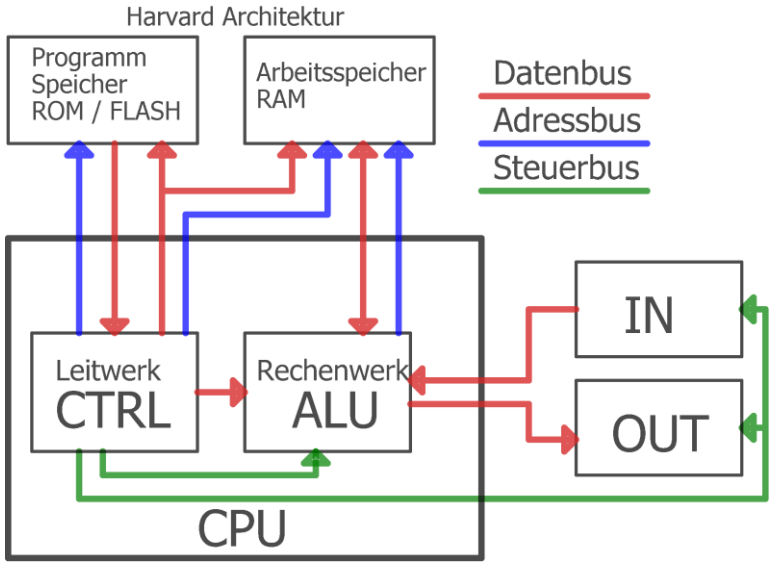
CPU Architektur

Der Unterschied von einer **PC-CPU** zu einem Mikrocontroller -> **MCU**.

Tatsächlich ist die heutige PC Architektur schon ~1945 entstanden. (**Von-Neumann-Rechner, VNR**)

Micro Controller haben strukturell getrennte Buse. Programm & Arbeitsspeicher (**Harvard Architektur**)

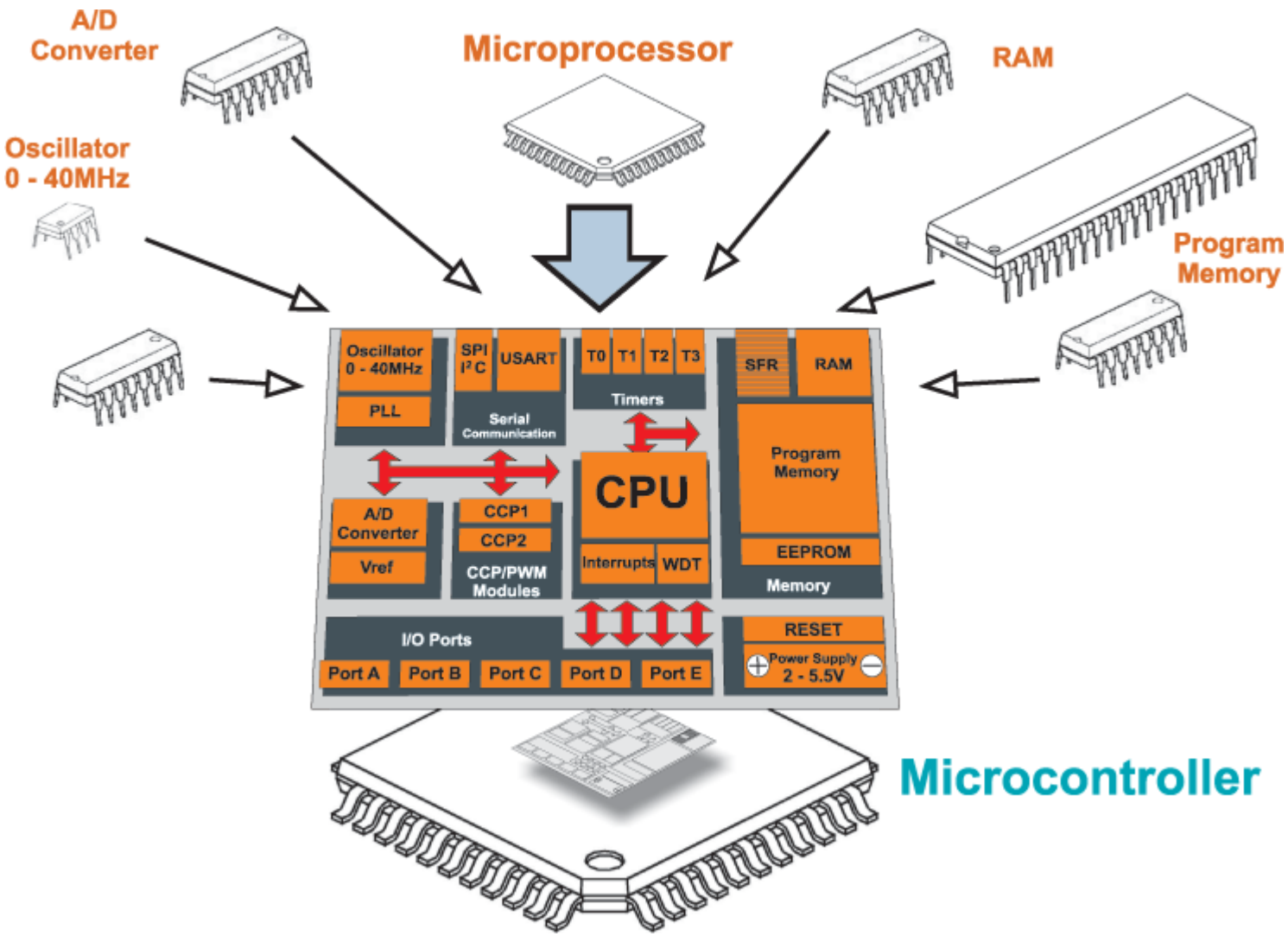




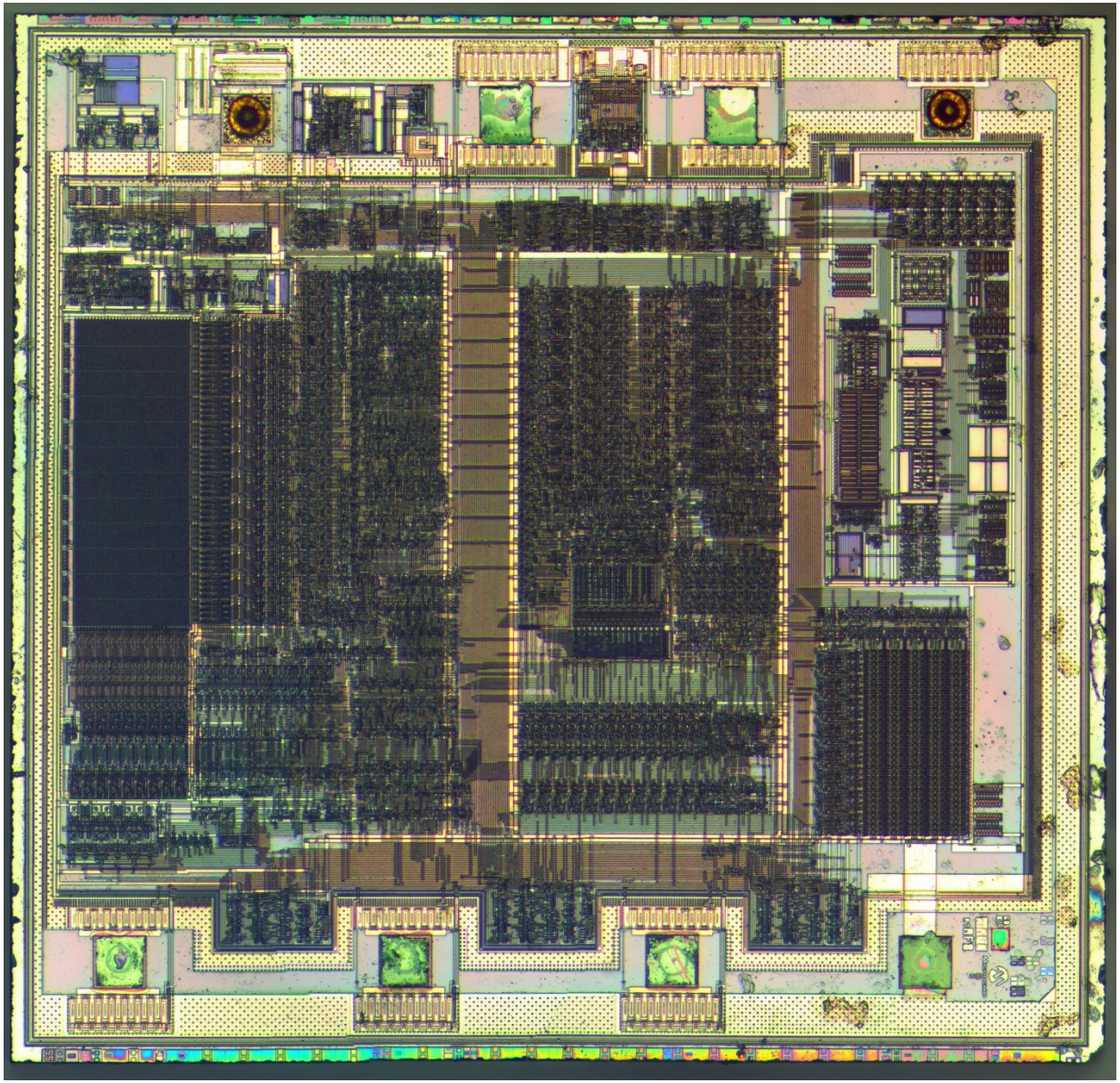
Kein Unterschied zwischen Code & Datenbereich
(RAM-Flaschenhals)
Typen dieser Architektur.
„X86 = PC“, RISC, ARM, 68HC08

Gleichzeitiger Zugriff auf Daten & Programm. Geringer RAM Bedarf.
Zugriff auf ROM (Konstanten) bedarf besonderer Beachtung.
Typen: AVR, PIC, DSPs, Konsequenz: Spez. Zugriff auf Programm Memory Konstanten

Ein **Mikrocontroller** Integriert viele einzelne Computerkomponenten auf einen Chip in einem Gehäuse



Typisches Chipdesign eines alten kleinen Mikrocontrollers: PIC12C508



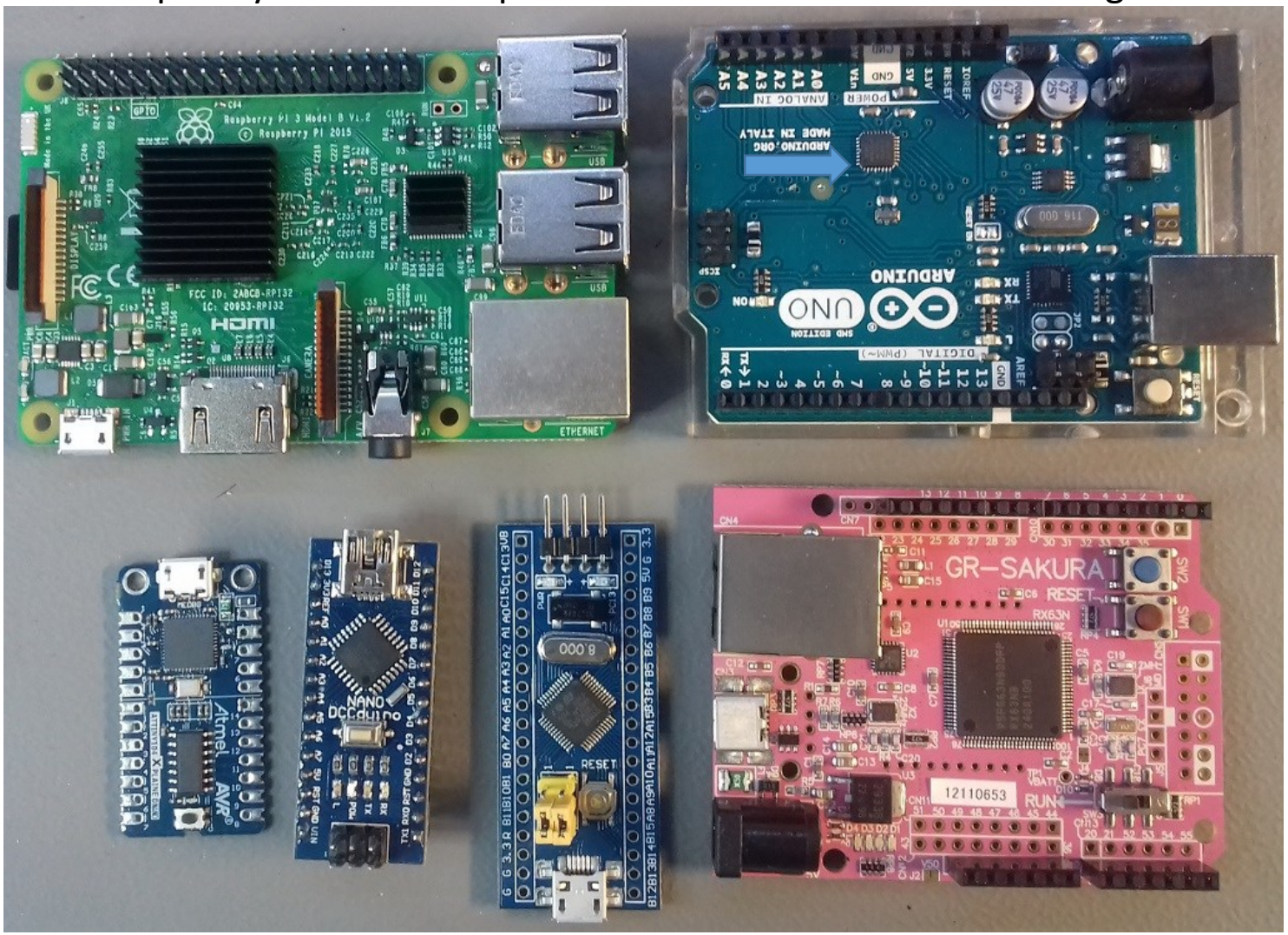
Man kann sogar unterschiedliche Regionen erkennen, auch wenn uns die Bedeutung nicht bekannt ist. Große gleichartige Flächen sind vermutlich jedoch meist Speicher:
RAM, ROM
Flash..

Doch zum Glück gibt es die dazugehörigen Handbuch PDFs. Darin ist alles beschrieben.

Mikrocontroller Allerlei. Der Markt bietet zahlreiche Typen. 8/16/32 Bit, RISC, ARM

Raspberry Pi3 mit ARM µC

Arduino UNO mit ATMega328P C



ATMega32

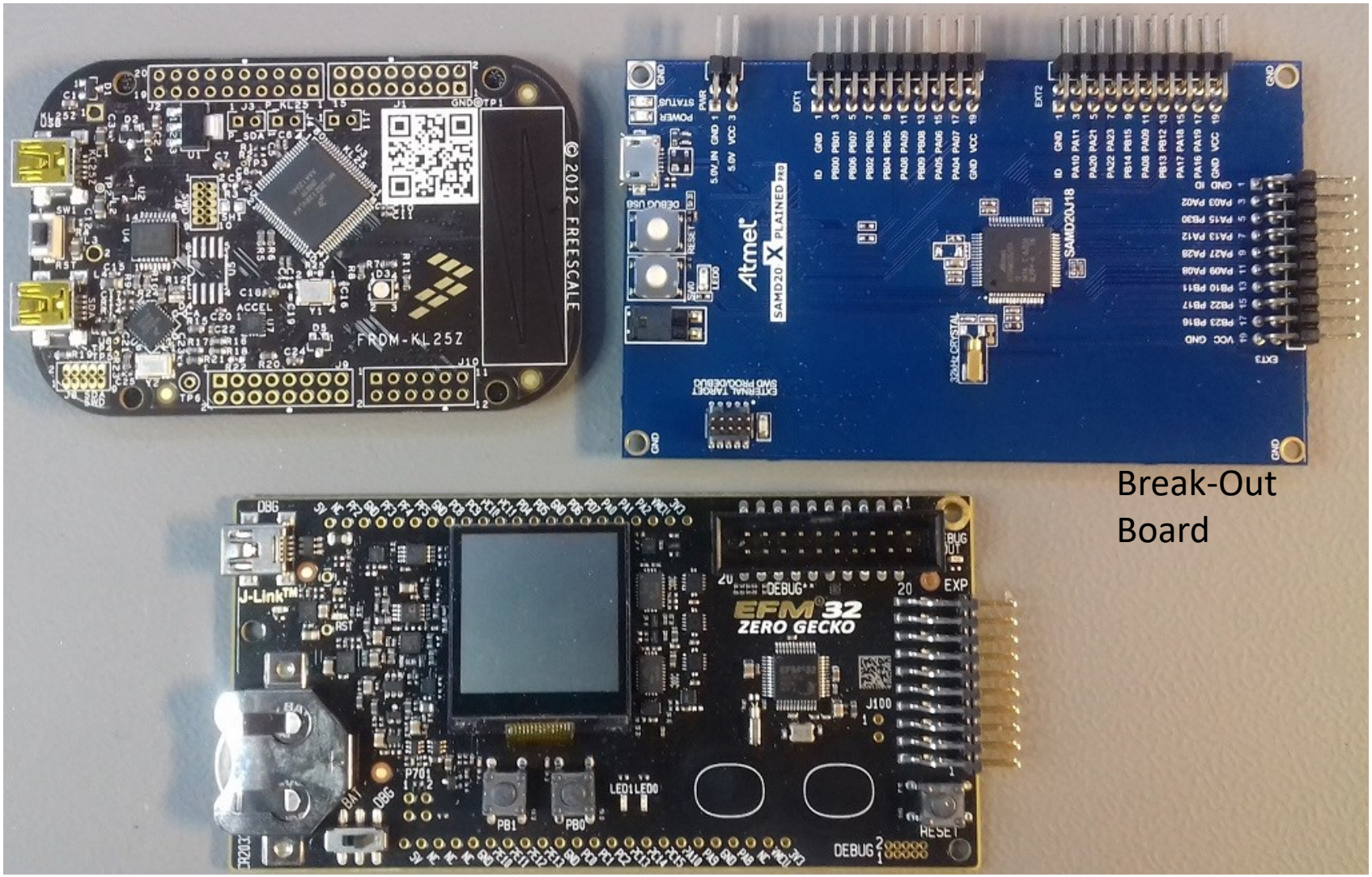
Arduino Nano

STM32 ARM ~1.50€

ARM M3 µC mit Net

Freescale ARM® Cortex™-M0+
Prozessor FRDM-KL25Z 32Bit

SAM Familie: ARM AT**SAM**D20J18A 32Bit



Break-Out
Board

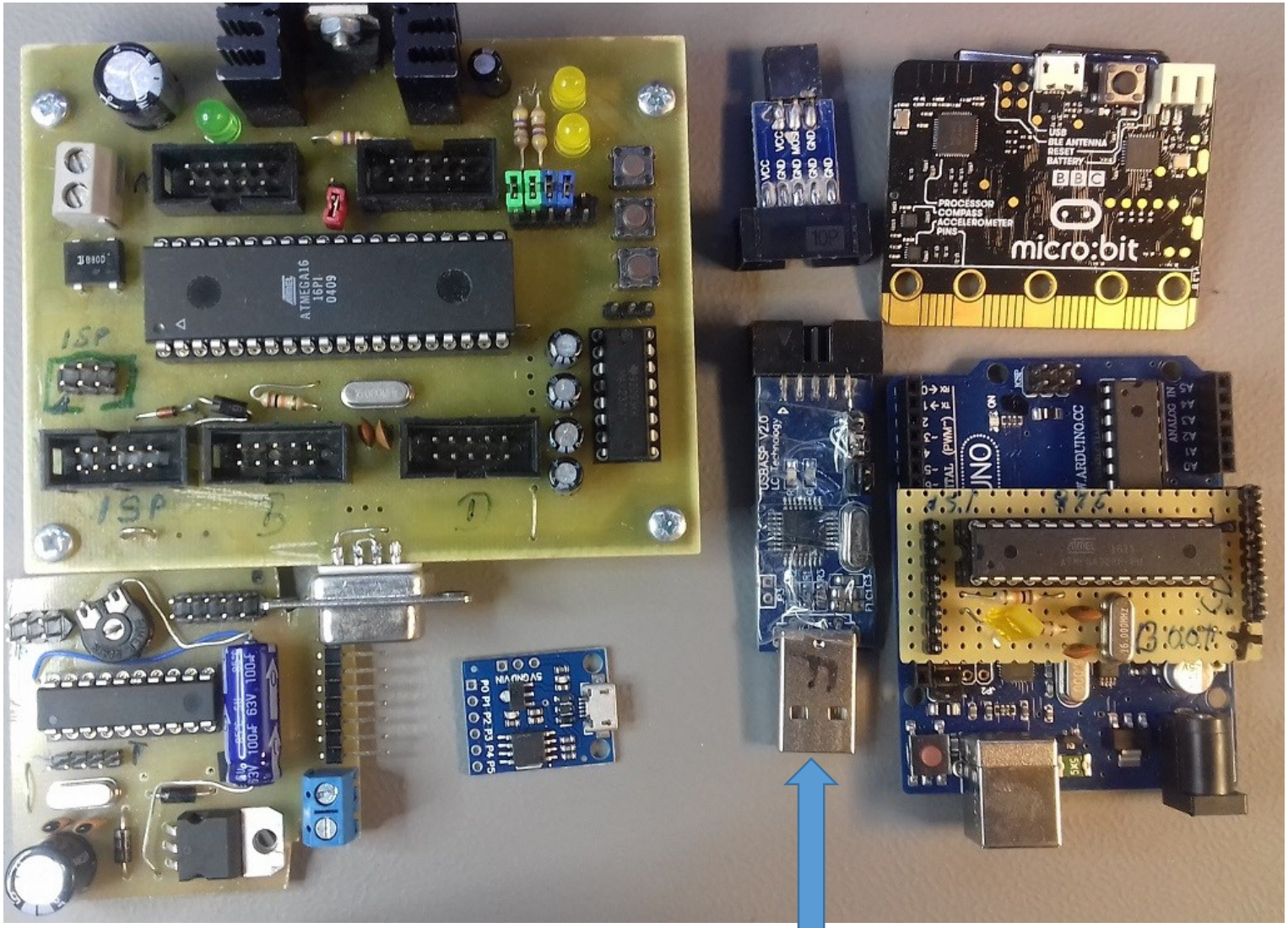
Silicon Labs **EFM32**ZG222F32 32Bit ! sehr Stromsparend

Diverses Kontroller-Klein:

AT16-BreakOut

ISP 6-10 Adapter

BBC-micro:bit



Kleines **ATTiny4313**
E-Bike Steuer Projekt.

Atmel
Tiny85

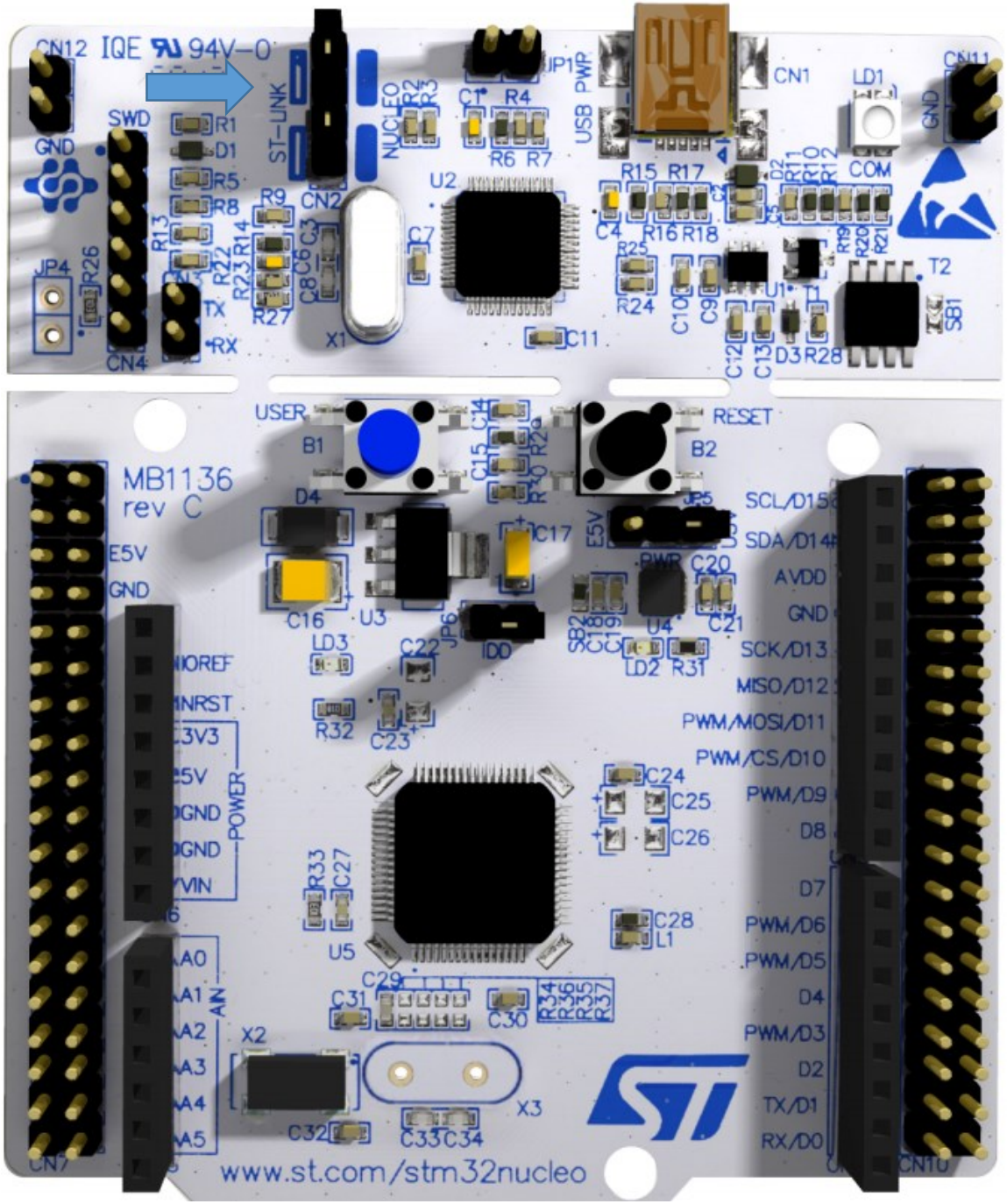
ISPusb
Programmer

Quick and dirty: Bootloader
Download-Adapter für Arduino

Die Zukunft gehört heute (2023) eindeutig den 32 Bit ARM M0(oder x) µControllern

STM32
Silicon Labs **EFM32**ZG222F32
SAM Familie: AT**SAM**D20J18A

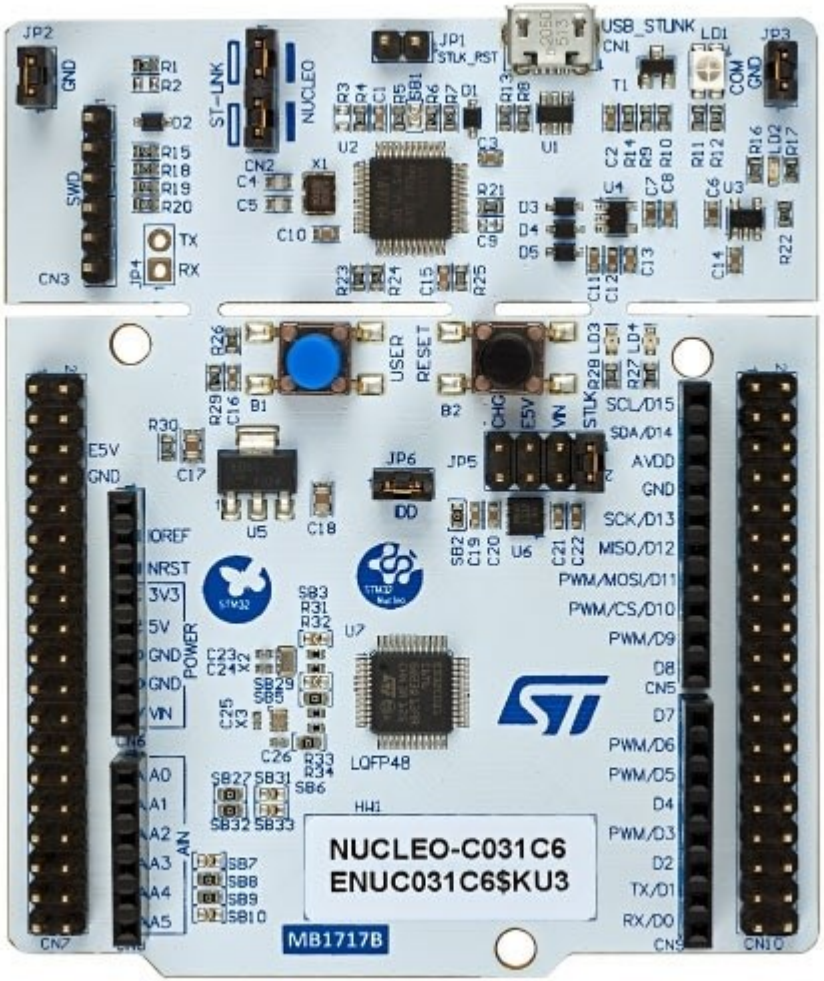
Diese kleinen Kontroller sind enorm leistungsfähig, 40..80..+X MHz, billig, stromsparend, enormen Leitungsvermögen,
12 BIT ADCs, Timer ohne Ende, riesen Flash/RAM,
Debugging fähig via **SWO/SWD**
mit externen **ST-Lin**k oder dessen billigen China Klonen.
So ein ‚Breakout' Board kostet ~20€.



Messebeute: vVariante NUCLEU_C031C6

https://www.st.com/en/evaluation-tools/nucleo-c031c6.html

Inclusive Debugger . (Aktuell 03.2023)



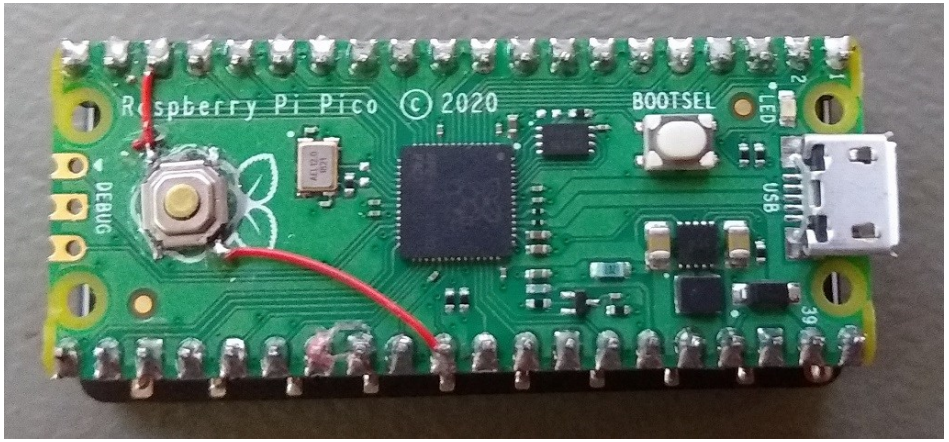
IDE Empfehlung:
STM32Cube MCU Package

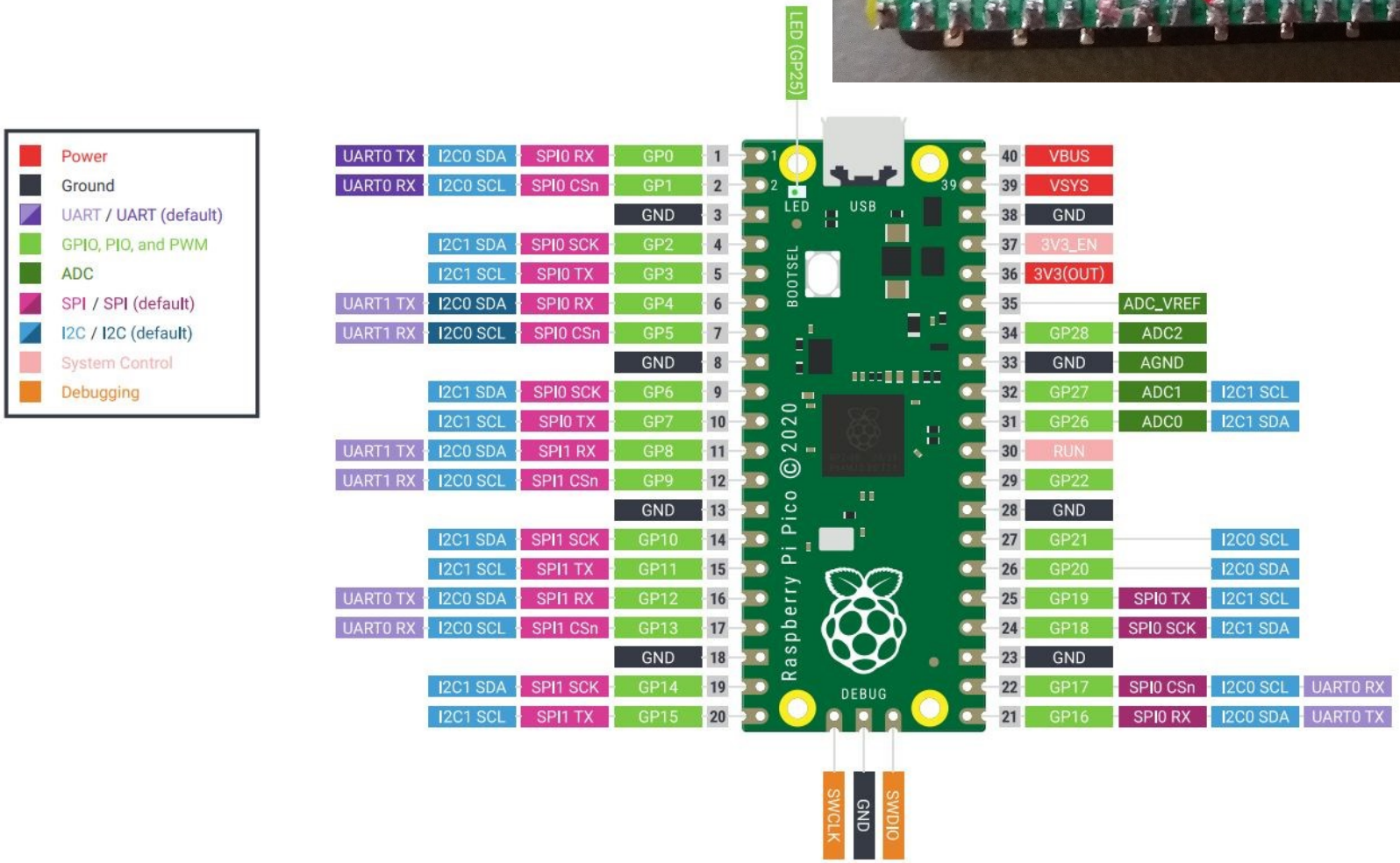
The STM32 Nucleo-64 board comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package.

Raspberry PI Pico.

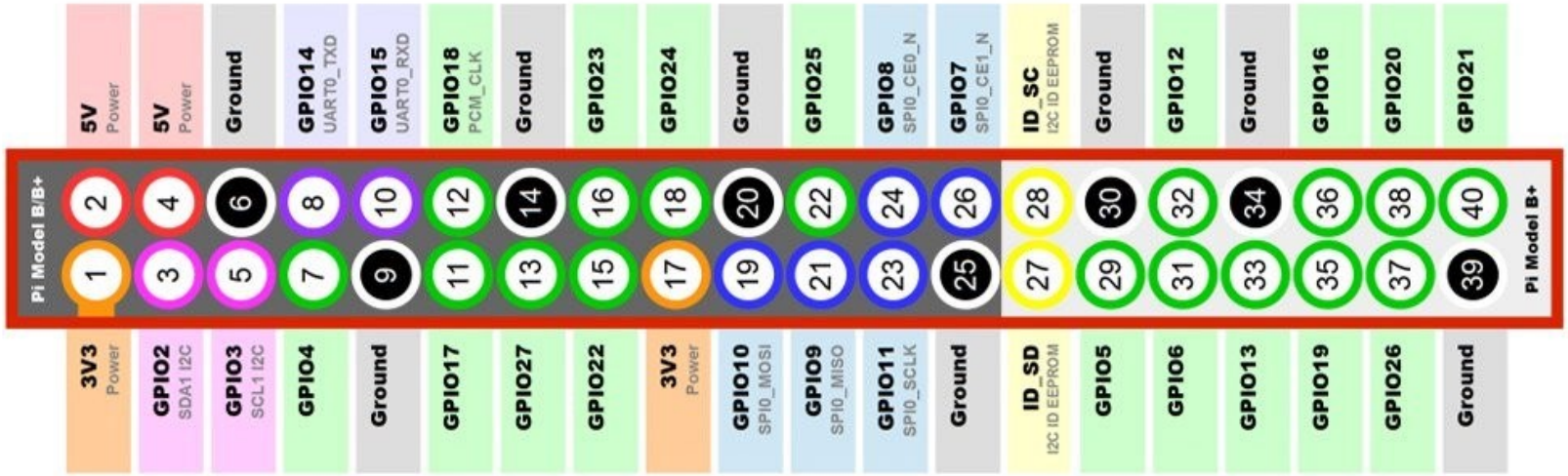
Klein, billig, aber mit hoher Leistung

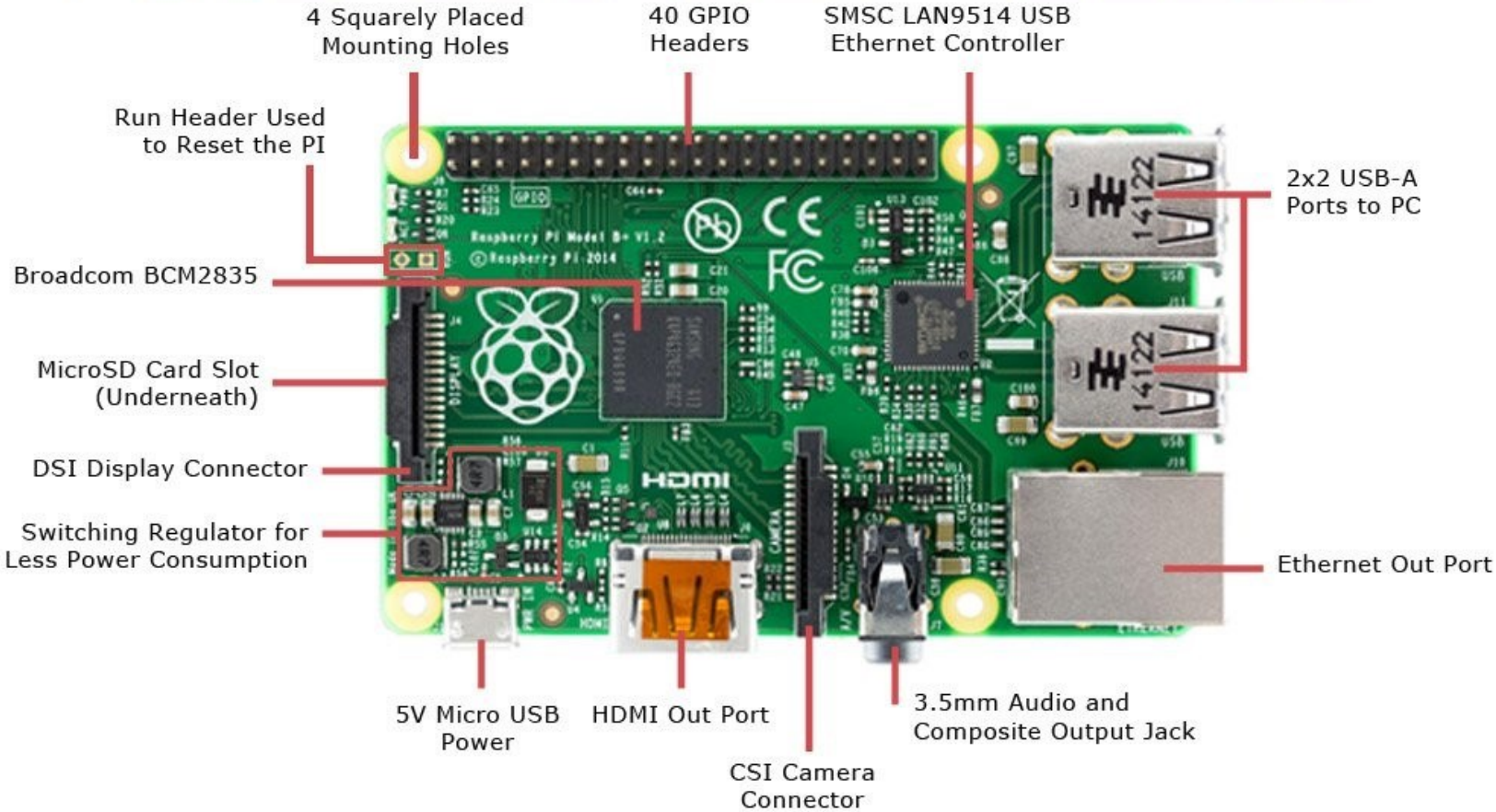
ARM µC





GPIO Pinout Diagram





Warum fange ich dann nicht mit dem 32 Bit ARM M0 an, sondern mit 8-Bitern wir den AVRs ?

Nun: Kennst du einen, kennst du alle !?

NEIN.

Ich halte den **STM für den Einstieg** in die µController Welt zu schwierig!
C?, C++? Python?
Die Konzepte der ARM sind sehr Umfangreich wie Anspruchsvoll.
Die Compiler und die IDEs kompliziert. (**IDE**=Integrated Development Environment).
Die kostenlose IDE "**ECLIPSE"** mit implementierten **Gnu C (GCC) Compiler** ist so lernintensiv, gewöhnungsbedürftig wie mächtig und Umfangreich.
Ohne Handbuch und viel Googlen ein steiniger Anfang
Die Initialisierung der IDE, bis alles läuft, braucht seine Zeit.

Der Kreuzweg sollte kein Holzweg werden.

Die Gefahr, dass der Atem, die Energie, die Zeit *am Anfang* nicht reicht, ist zu groß.
..heißt, um zeitig zu Erfolgen zu kommen, ist es für „Beginner" nicht der richtige Weg.
Die 32 Bit Arm Controller Welt ist selbst noch zu erforschen.

Weitere ‚leistungsfähigere' Mikrocontroller

In letzter Zeit haben sich höchst interessante Entwicklungen gezeigt.
Vor allem die ARM Familie wird immer günstiger und, leitungsfähiger und billiger. Siehe die Bucht. Ich nenne hier einige Beispiele:
Alles auf diesem Modul fertig montiert: für ~5..10€ ! …..tatsächlich

ESP-8266

~**80MHZ**, integriertes WLAN

https://www.heise.de/developer/artikel/ESP32-Neuer-IoT-Chip-von-Espressif-3506140.html

Diesen gibt es mit integriertem WLAN für weniger als 3€-.
Er kann unter SKETCH Programmiert werden: Installation→
http://www.mikrocontroller-elektronik.de/nodemcu-esp8266-tutorial-wlan-board-arduino-ide/
Einfach selbst recherchieren. Es lohnt sich!

Seltsam: keine echten Hardware Timer/Counter inside.
Es muss alles per Software realisiert werden. Aber, diese Controller sind sehr schnell.

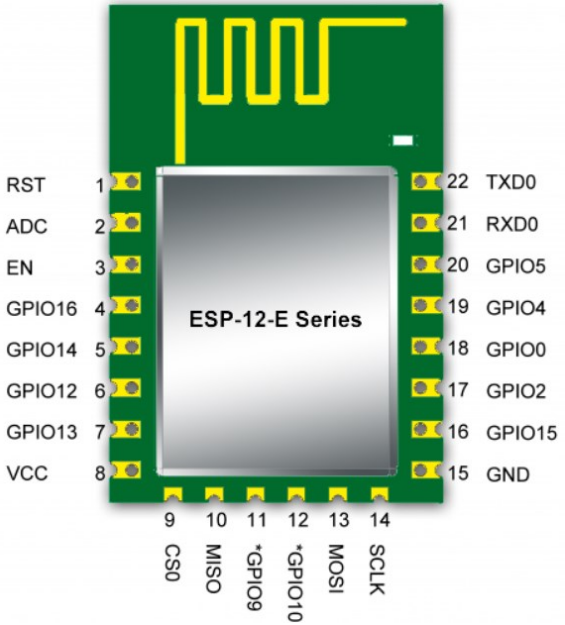
TOUT	ADC0	A0
		Reserve
		Reserve
SDD3	GPIO10	D12
SDD2	GPIO9	D11
	SDD1	INT
	SDCMD	MOSI
	SDD0	MISO
	SDCLK	SCLK
		GND
		3,3V
		EN
		RESET
		GND
		POWER 5,0V

USB

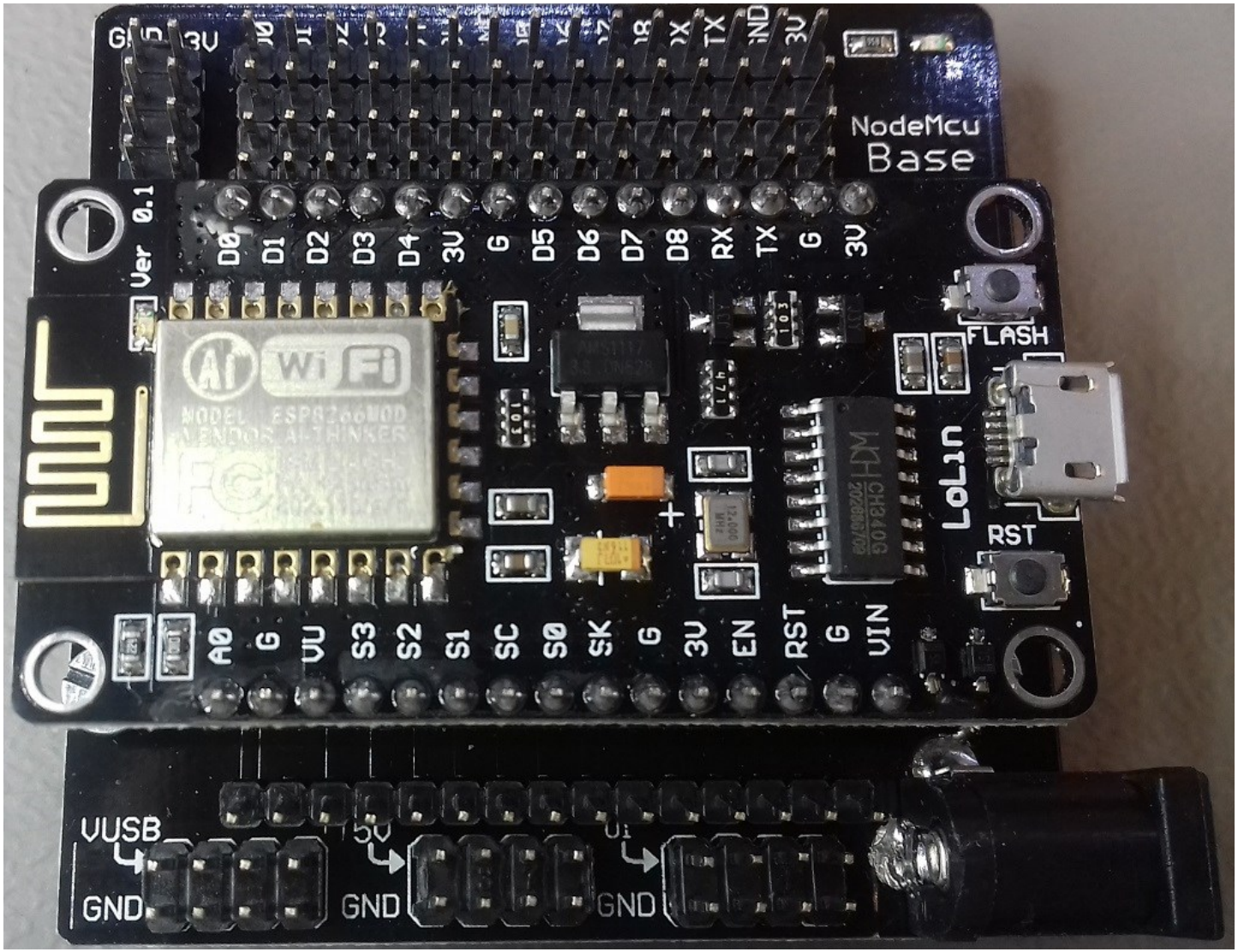
Programmierung und Spannungsversorgung

D0	GPIO16	- USER - WAKE
D1	GPIO5	
D2	GPIO4	
D3	GPIO0	- FLASH
D4	GPIO2	- TXD1
3,3V		
GND		
D5	GPIO14	- HSPICLK
D6	GPIO12	- HSPIQ
D7	GPIO13	- RXD2 - HSPID
D8	GPIO15	- TXD2 - HSPID
D9	GPIO3	- RXD0
D10	GPIO1	- TXD0
GND		
3,3V		

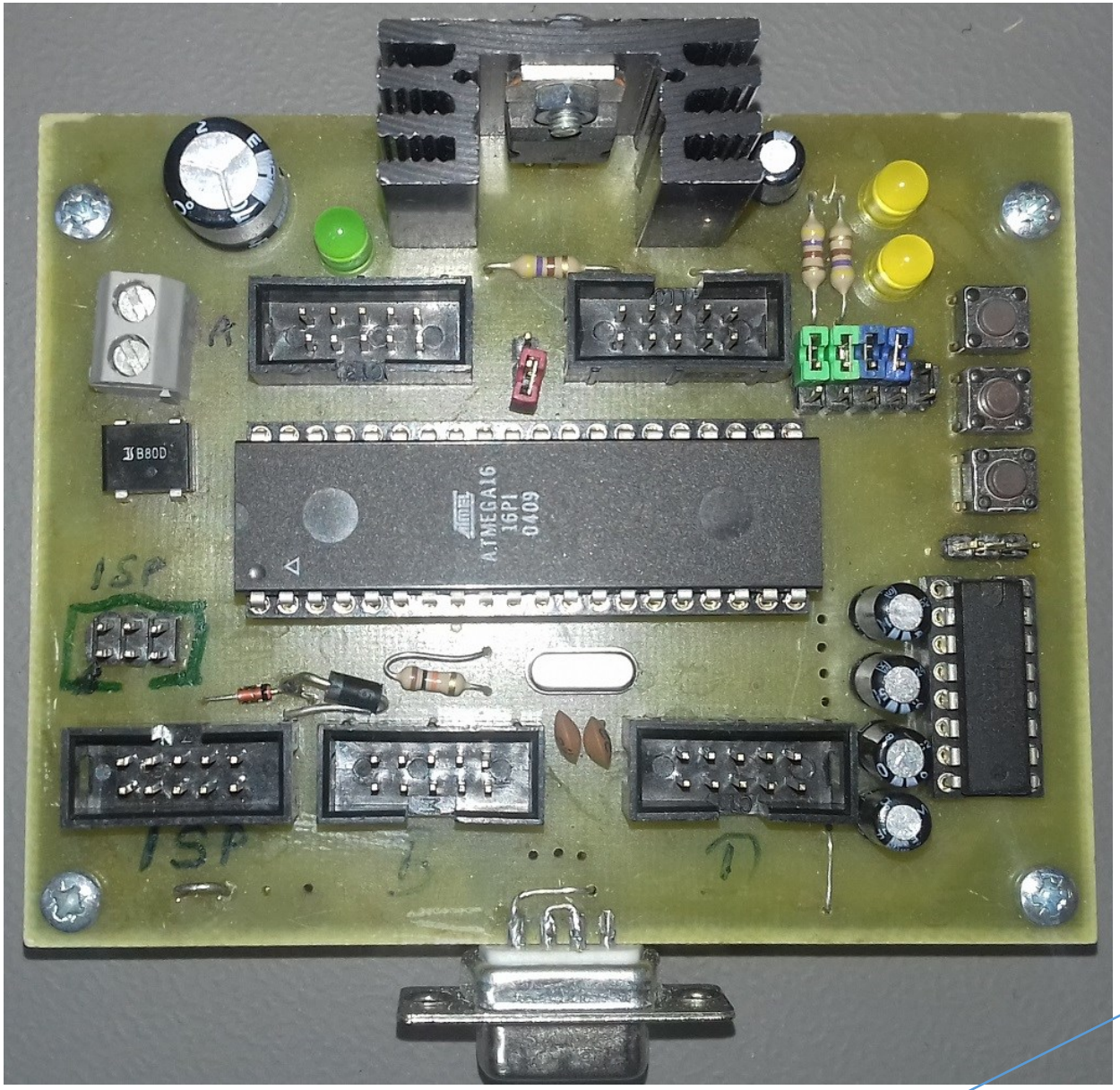
Hier das Original



Hier steckt das **ESP-8266** WLAN Modul in einem Break-Out Adapter
Tipp: wenige € aus der Bucht → hier mit China USB-COM Chip CH340G



Bis vor wenigen Jahren haben ‚wir' dieses **ATMega16** Board im Kurs selbst aufgebaut.



Schritte:

Leere Platine mit Leiterbahnlayout belichten.

Entwickeln, mit **Natriumhydroxid**.

(Rohrfrei)

Ätzen mit **Natriumpersulfa**t oder **Eisen(III)-chlorid**. {bääh}

Aussägen.

Die Löcher bohren, passend zu den Bauteilen.

Durchkontaktierungen und Brücken löten.

Sockel und Bauteile auflöten.

(von klein nach groß)

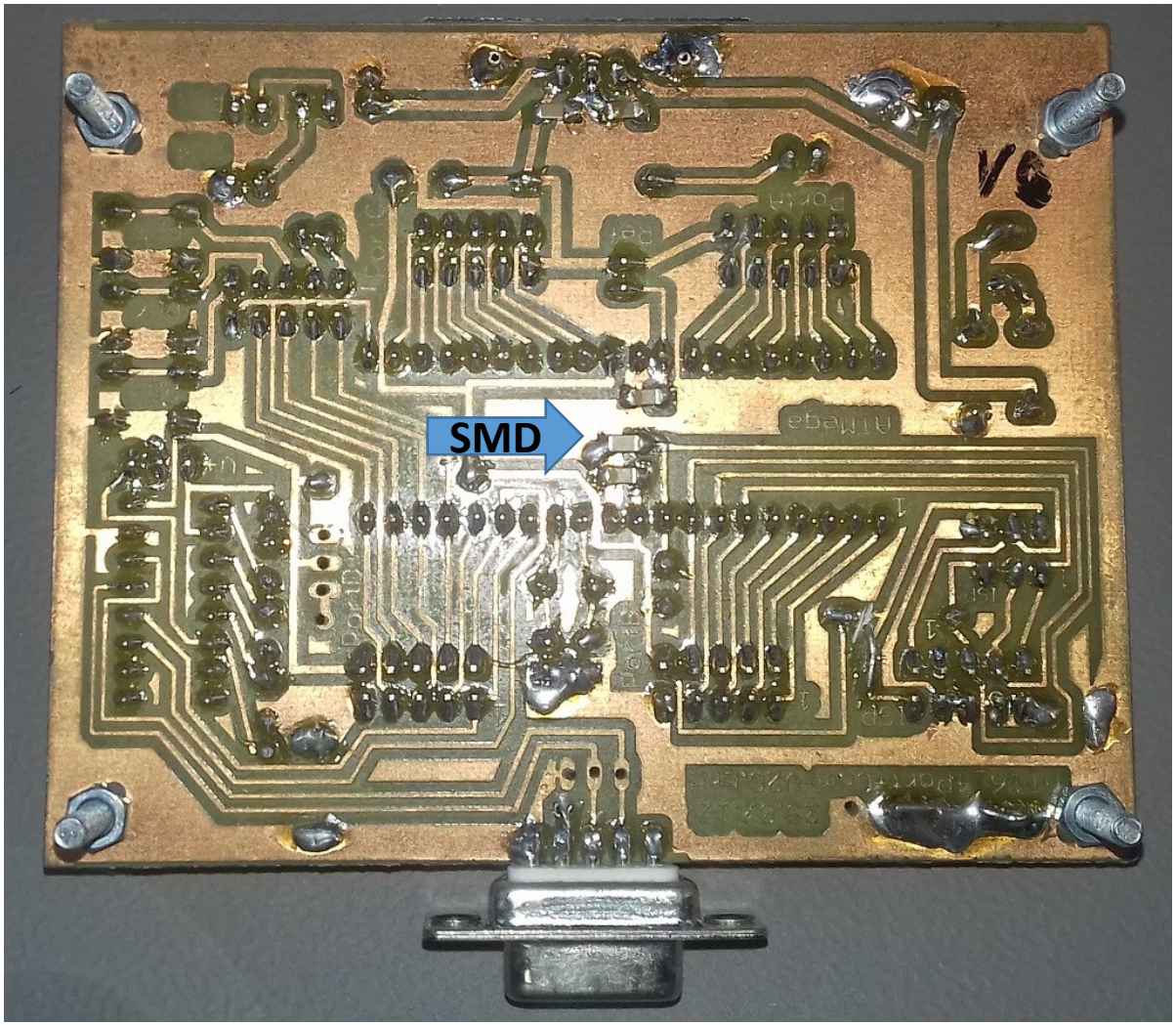
Durchtesten mit Piepser.

ICs bestücken.

Fusebits setzen mit **ISP** Programmer.

Progamm aufladen mit ISP.

http://www.engbedded.com/fusecalc/



Schön war die selbstgemachte Haptik. Dass alles zugänglich und so verständlich wurde. Status: 80er Jahre Stil.
DIL Gehäuse. Standard Bauteile in Riesengröße.

Layout mit vollständiger Herausführung aller vier 8Bit PORTs & +5V Supply + ISP Programmierstecker + SubD9 COM-RS232 Stecker.

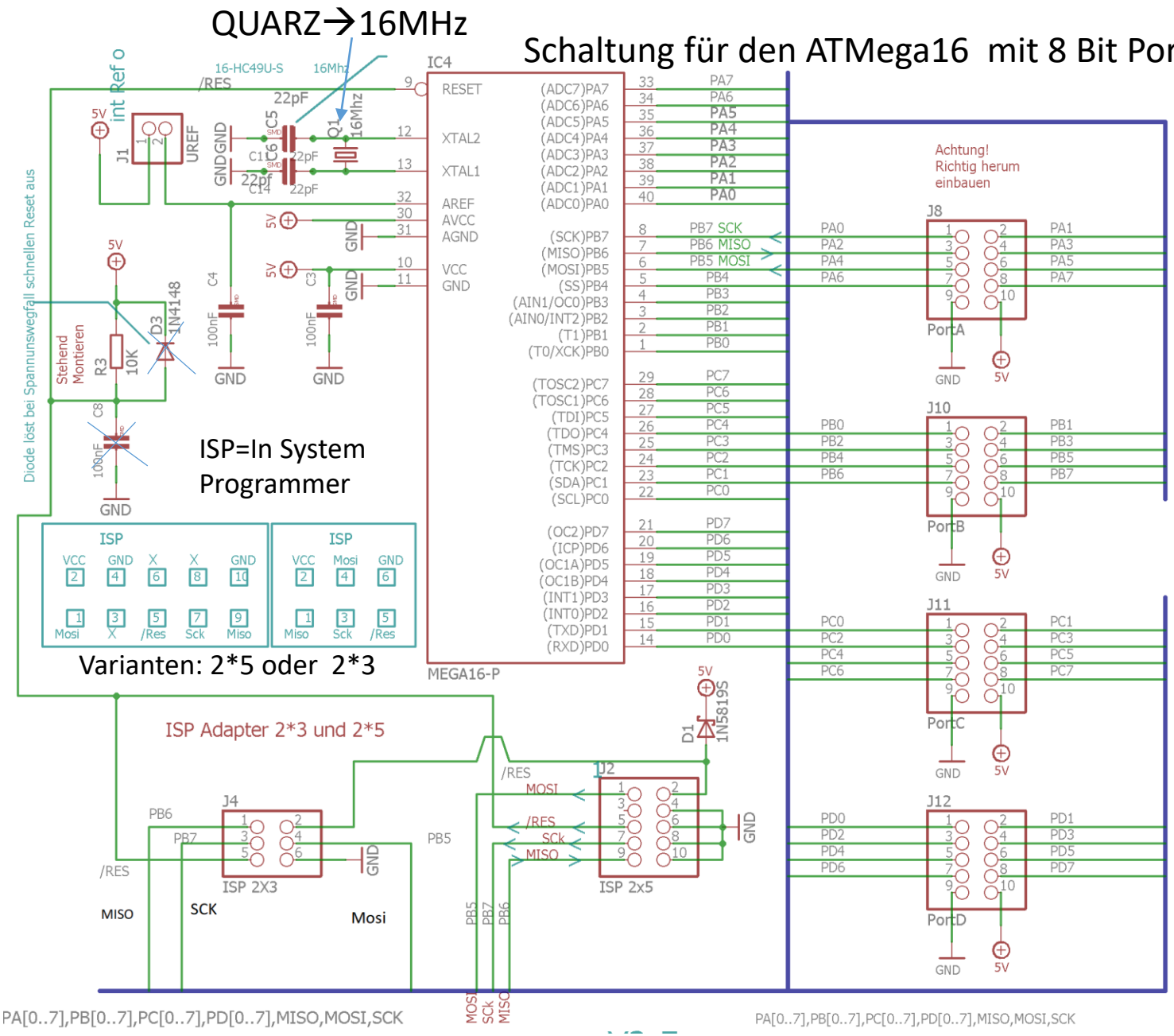
Trick: RS232 Serieller Stecker direkt seitwärts an der Platine befestigt.

Schrauben als Standbeinchen.

Saubere Lötstellen!
(nicht zu viel und, vor allem, sauber rund um die Beinchen)

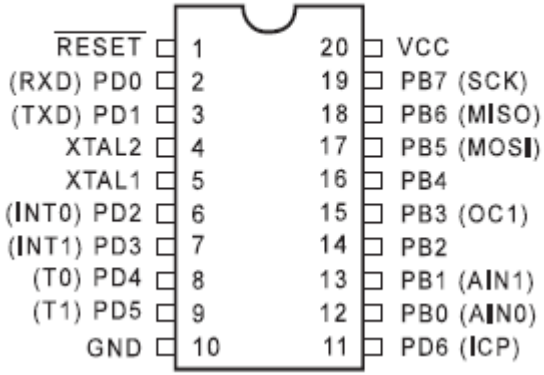
Damals noch ‚wenige' SMD Bauteile (100nF) um das Board nicht unnötig zu verkomplizieren.

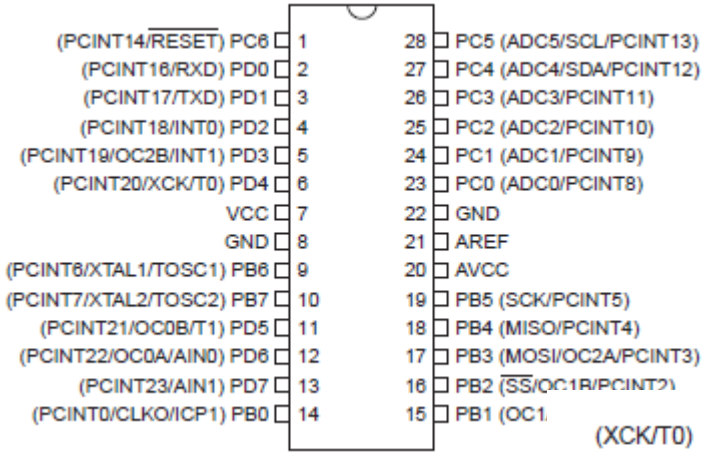
Schaltung für den ATMega16 mit 8 Bit Portstecker



Mit den Kenntnissen des AVR könne ganz unterschiedliche AVR Kontroller dieser Familie ohne große Änderungen genutzt werden.
Vieles bleibt gleich, oder enthalten erweiterte Funktionen oder fehlen.

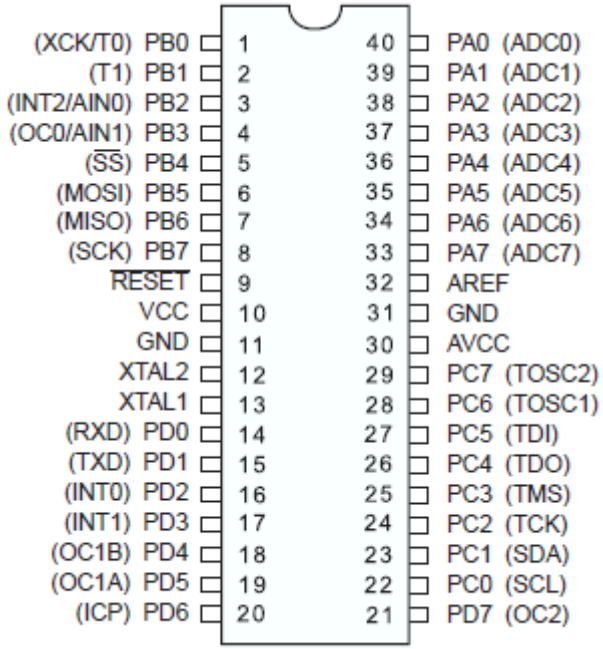
PDIP/SOIC



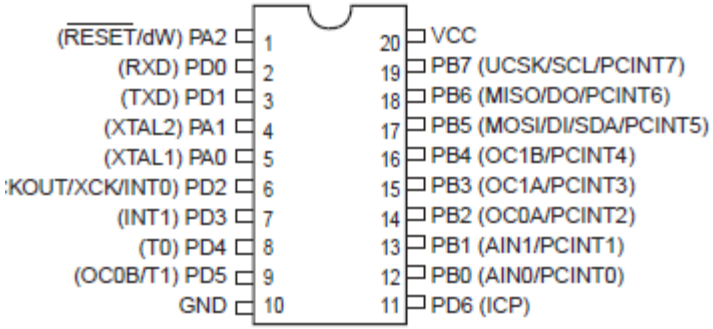


ATMega328p

ATMega16

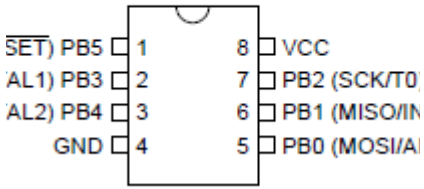


AT90S2313



ATtiny2313 (ohne A)

ATtiny12
PDIP/SOIC



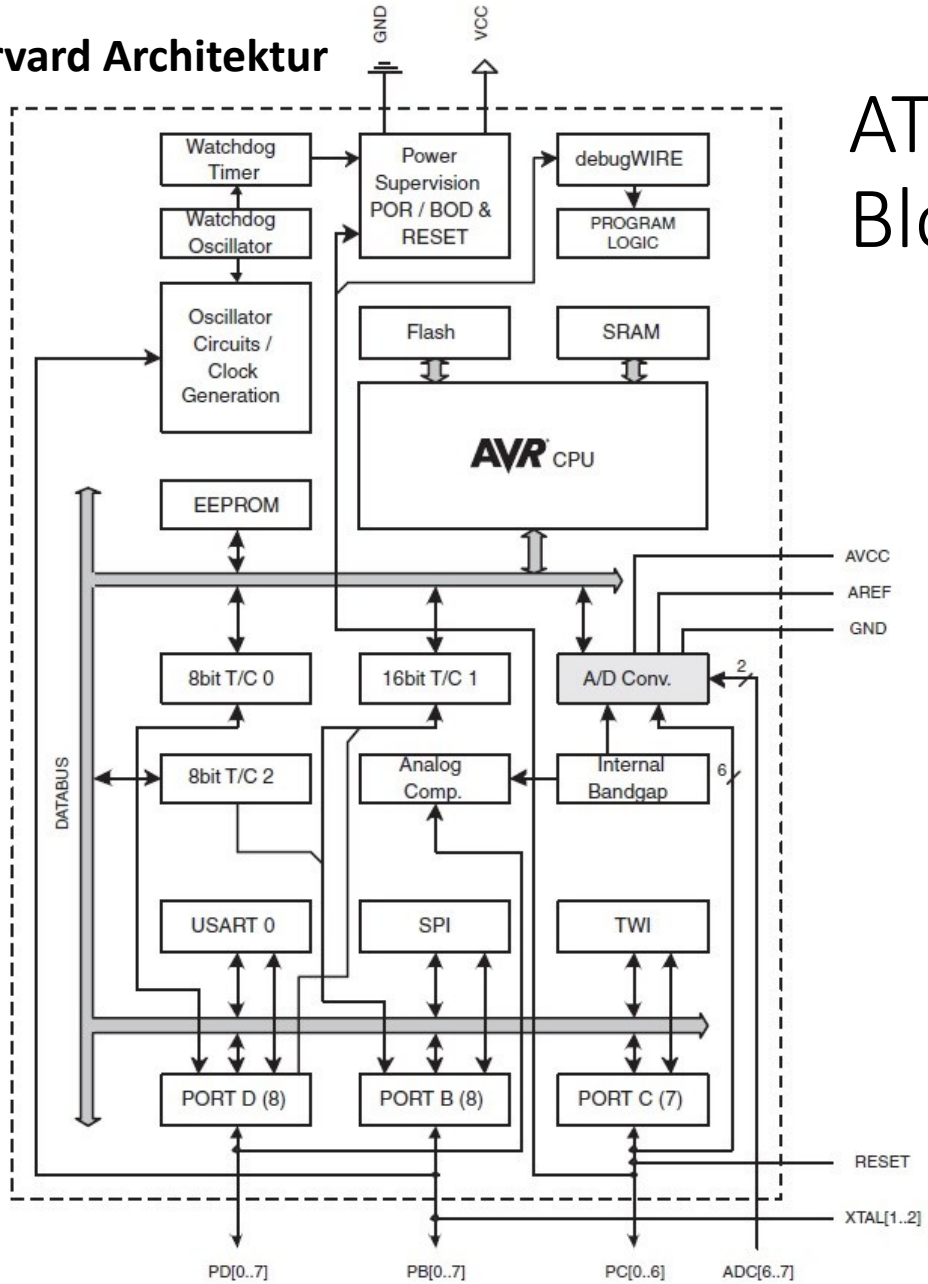
Vorstellung des
AVR ATMega xxx Mikrocontrollers

Da es sehr viele Varianten von Mikrocontrollern gibt
8,16,32 Bit, RISC, ARM
und davon wieder viele historisch gewachsene,
mehr oder weniger gut ausgestattete Untergruppen oder Bautypen gibt, habe ich mich für den bewährten und gut zu verstehenden
‚Atmel' AVR **ATMega** xxxx Mikrocontroller entschieden.
(Atmel wurde von Microchip aufgekauft)
Dieser ist als ATMega328p im Arduino UNO verbaut.
Sein größter Vorteil, außer dass er einfach zu verstehen ist und zuverlässig ist.

Kennst du einen, kennst du fast alle !

Link: ARTE Bericht zum Arduino mit C-Stile Sketch:
http://youtu.be/gwcmN8XoMfE

Harvard Architektur

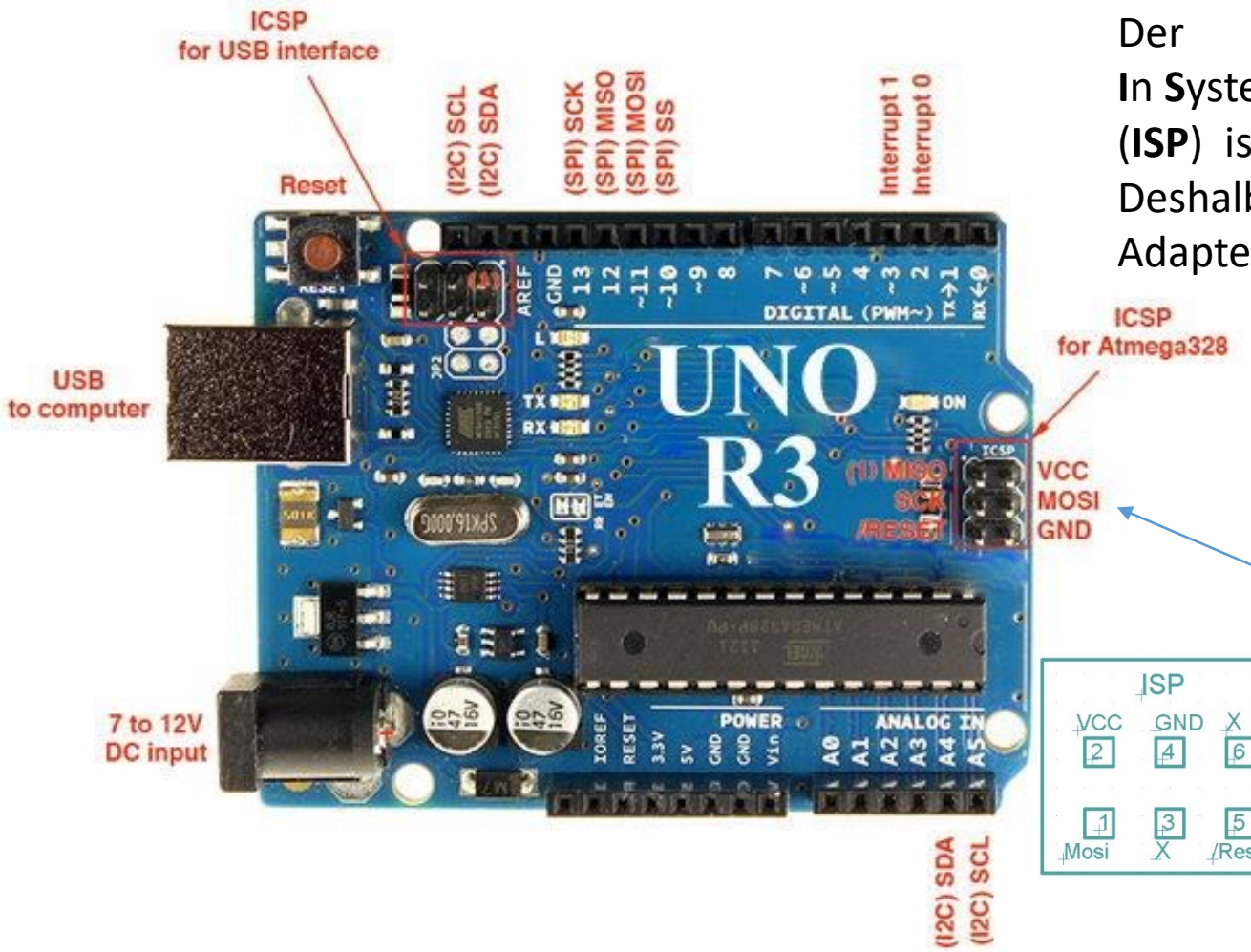


ATmega328p-Core-Block-Diagramm

PORTB (8) 0..6 ISP
PORTC (6) 0..5 6=/Res
PORTD (8) 0..7 0=Rx,1=Tx
TIMER0 8Bit
TIMER1 **16**Bit
TIMER2 8BIT
A/D Converter 10BIT mit Mux
UART = serielle Schnittstelle
SPI = sync. seriell port interface
TWI = I2C Bus
Watchdog
Analog Comparator
FLASH = Programm Speicher
EEPROM
SRAM = Arbeitsspeicher
…..und vieles mehr
PCINTx= pin change interrupt request

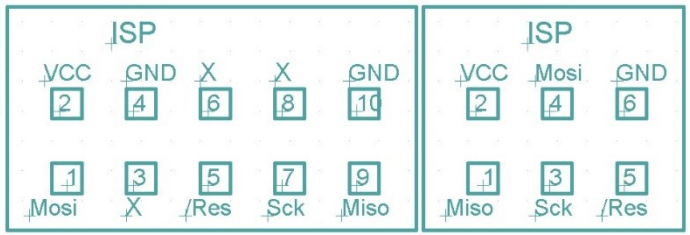
Ein gut zu verstehender Mikrocontroller ist der **ATMega328P** der im **Ardunio Board UNO R3** verbaut ist.

Ältere Boards haben den µC im DIL Gehäuse

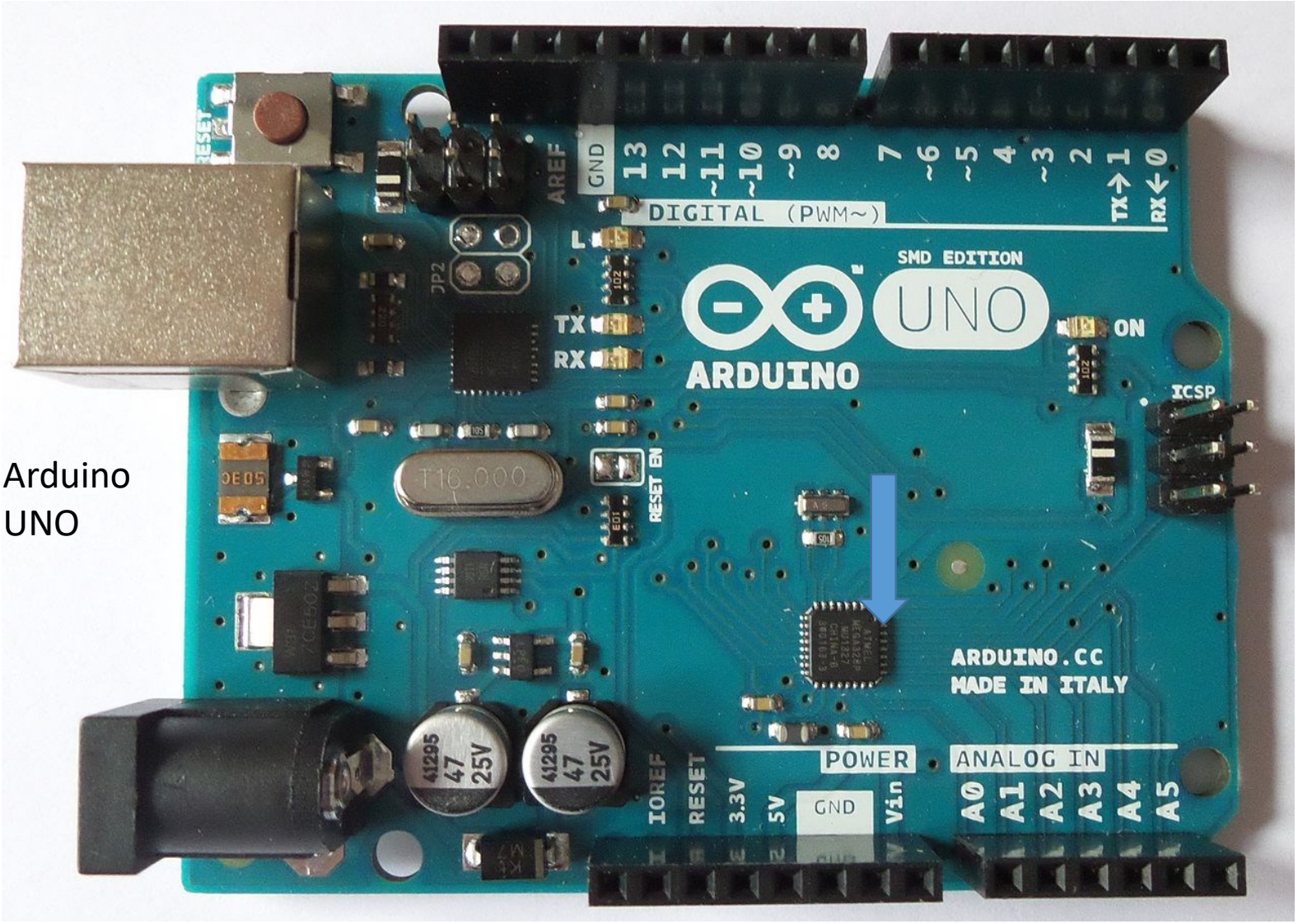


Der
In **S**ystem **P**rogrammer (**ISP**) ist 6 Polig.
Deshalb ist ein 6→10Pol Adapter notwendig.

Der quasi AVR **ISP** Standard ist der USBASP mit 10 Polen



Das ARDUINO UNO Board mit (kleinen) ATMega328P Mikrocontroller



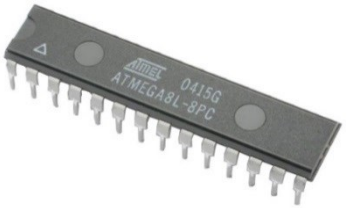
Arduino UNO

Die Nummerierung folgt nicht den **Pinnamen** des µController ICs

Arduino UNO Board mit **ATMega328P**, technische Spezifikation

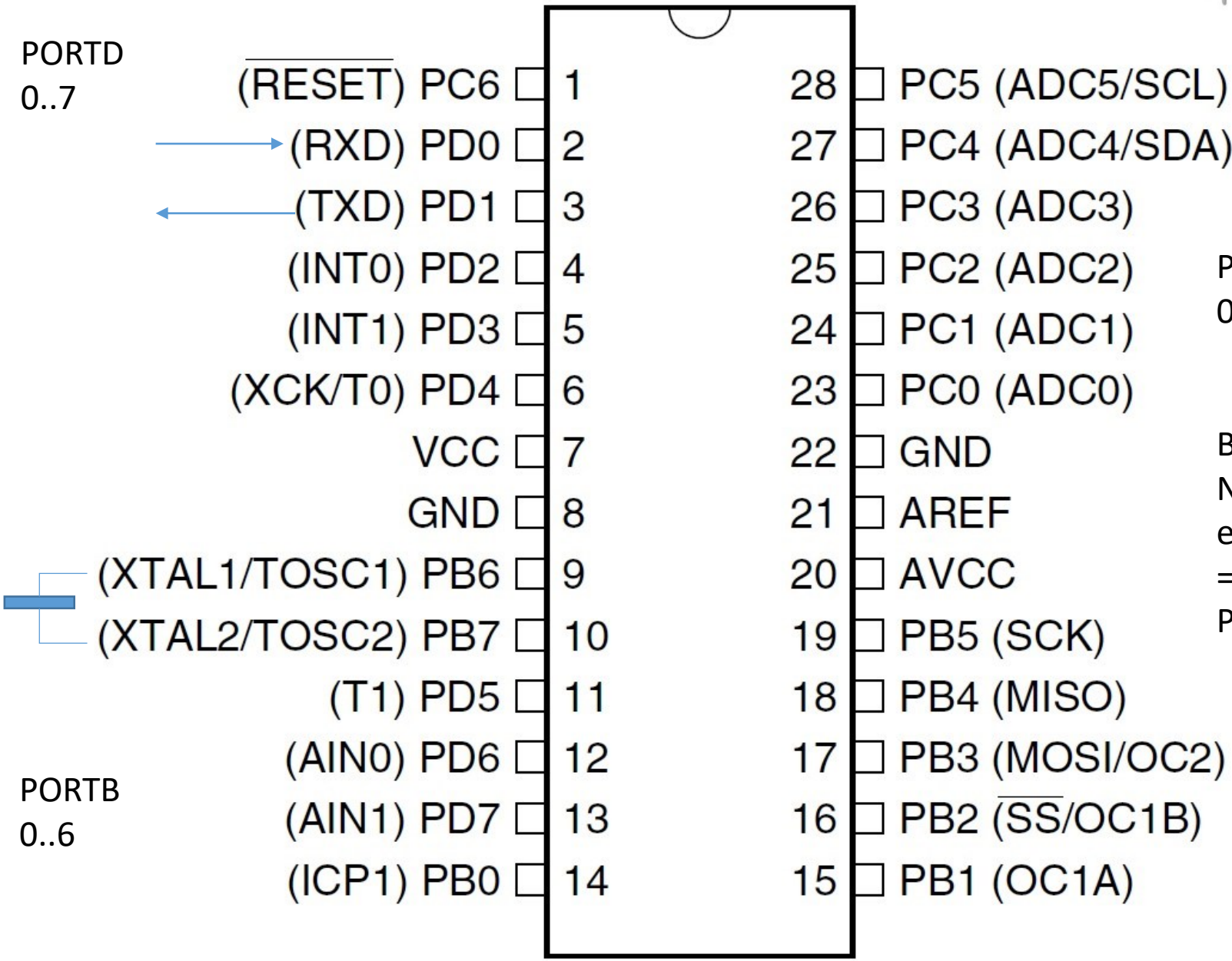
Microcontroller	**ATMega328 P**
Operating Voltage	**5V** (3.3V wären für den µC möglich)
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6
DC Current per I/O Pin	40 mA (aber nur an EINEM Pin)
DC Current for 3.3V Pin	50 mA
Flash Memory	**32 KB** (ATmega328) of which 0.5 KB used by bootloader
SRAM	**2 KB** (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	**16 MHz**

Grundlegende Pins, Bedeutung und Namen des Mikrocontrollers ab AVR **ATMega8** (aufwärts) DIL Gehäuse



PORTD
0..7

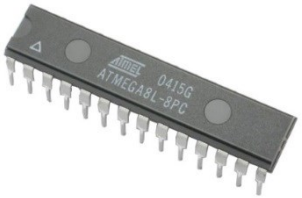
PORTB
0..6

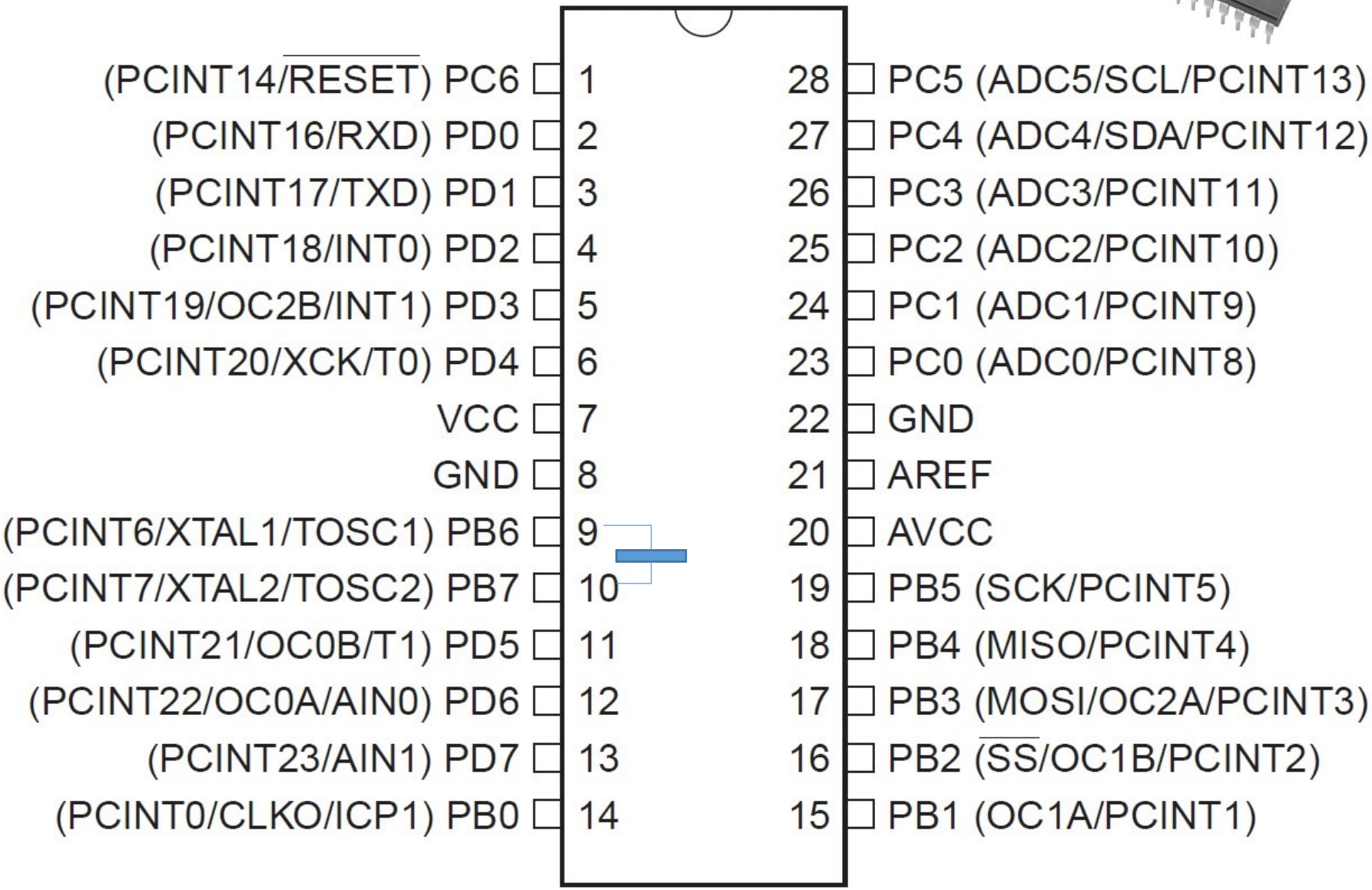


PORTC
0..5

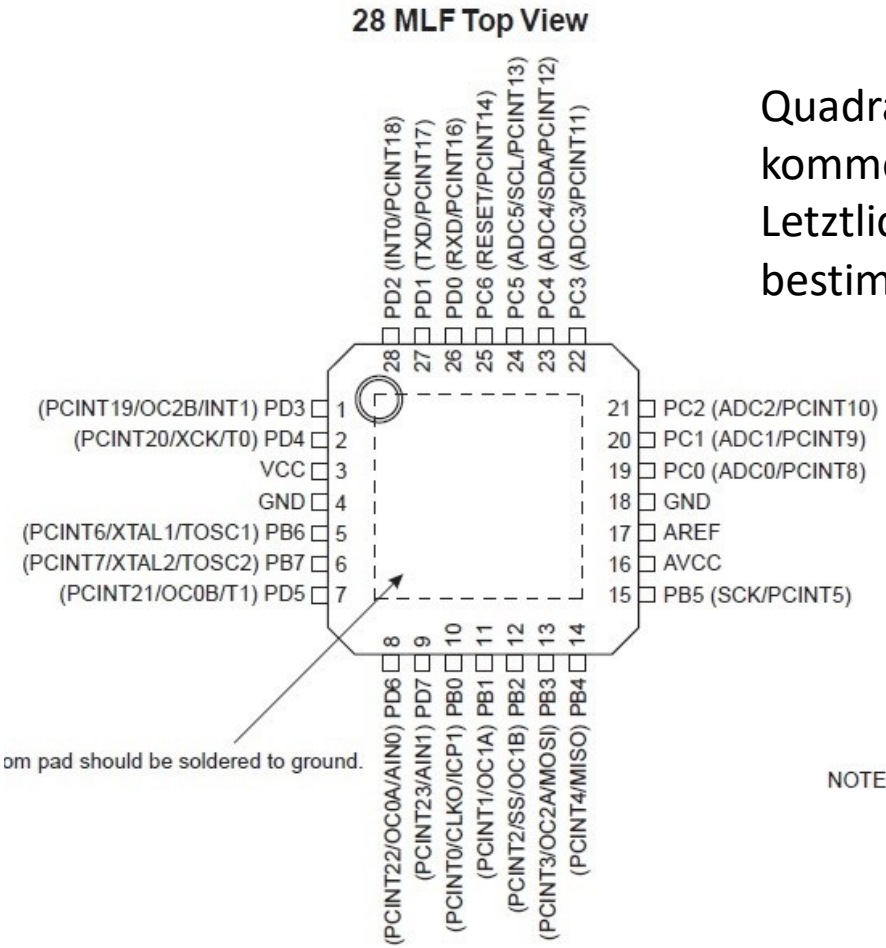
Bsp.
Name für
einen Pin
= PC5
Port C Pin 5

Pin-Out ATmega48PA/88PA/168PA/328P DIL Gehäuse

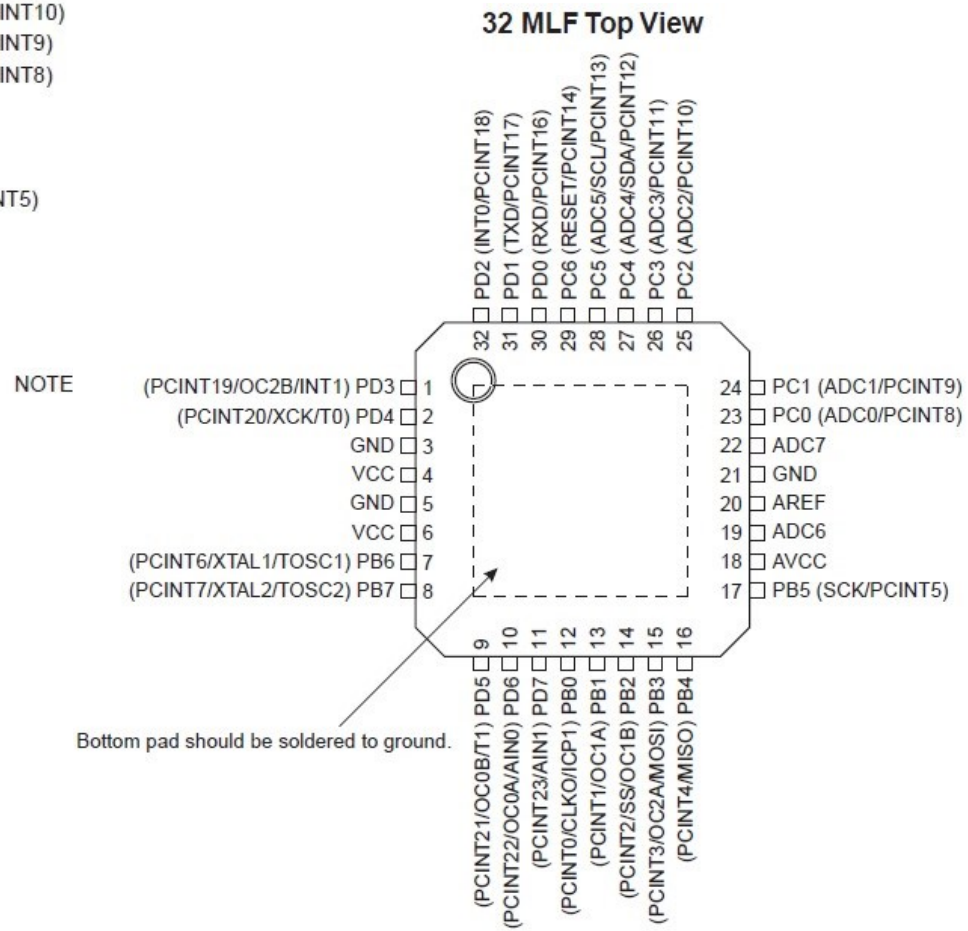




PCINTx= pin change interrupt request



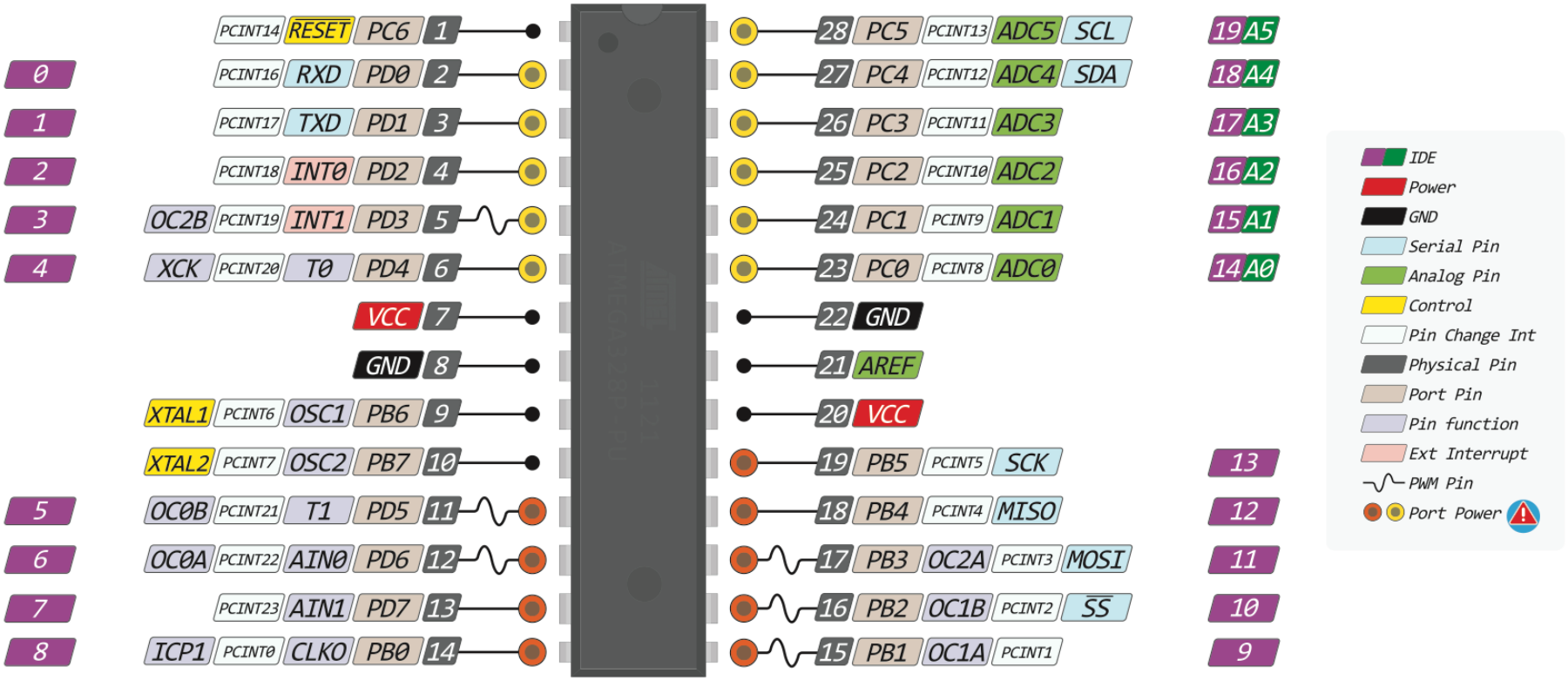
Quadratische Miniatur-Gehäuseformen (QMLF) kommen in der Praxis immer häufiger vor. Letztlich wird alles von der Massenproduktion bestimmt→ Smartphone Technologien etc.



Es ist jedoch leicht im Netz nach den Typen zu suchen und man wird sehr bald fündig.

ATMEGA328 PINOUT

Dies ist der Nachfolger von 2016



Absolute MAX per pin 40mA
recommended 20mA

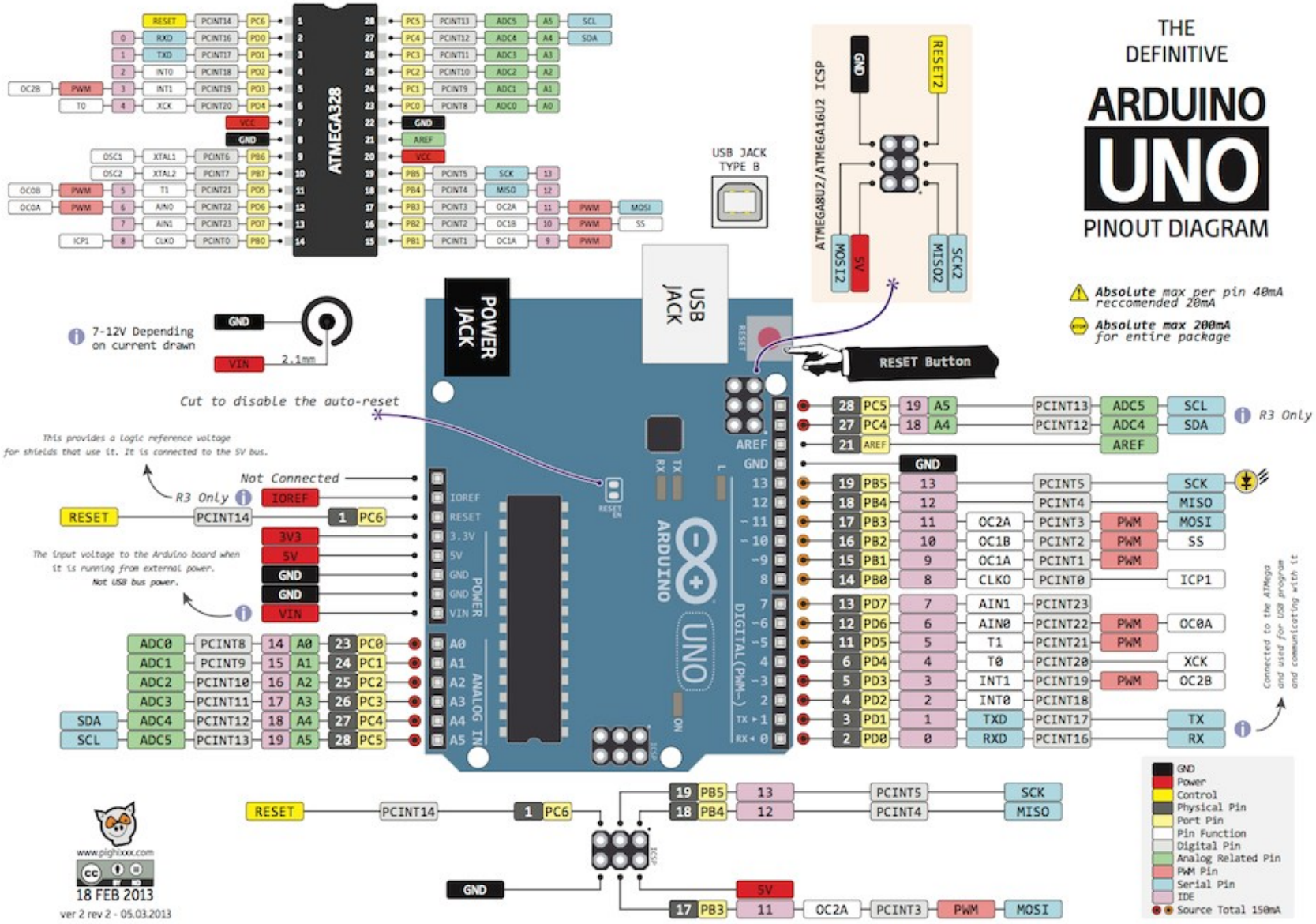
Absolute MAX 200mA
for entire package

The power sum for each pin's group should not exceed 100mA

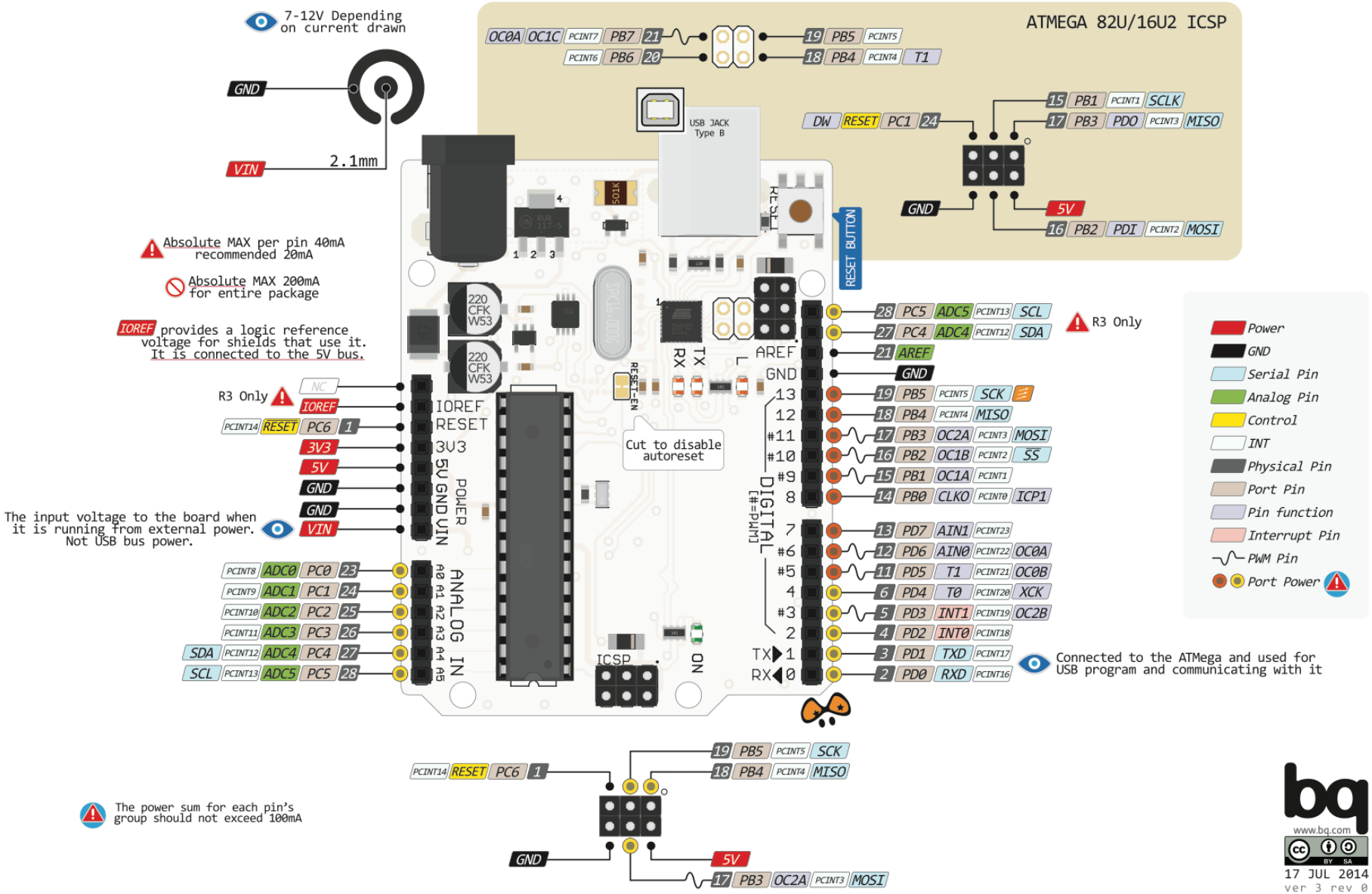
16 JUL 2014
ver 3 rev 0

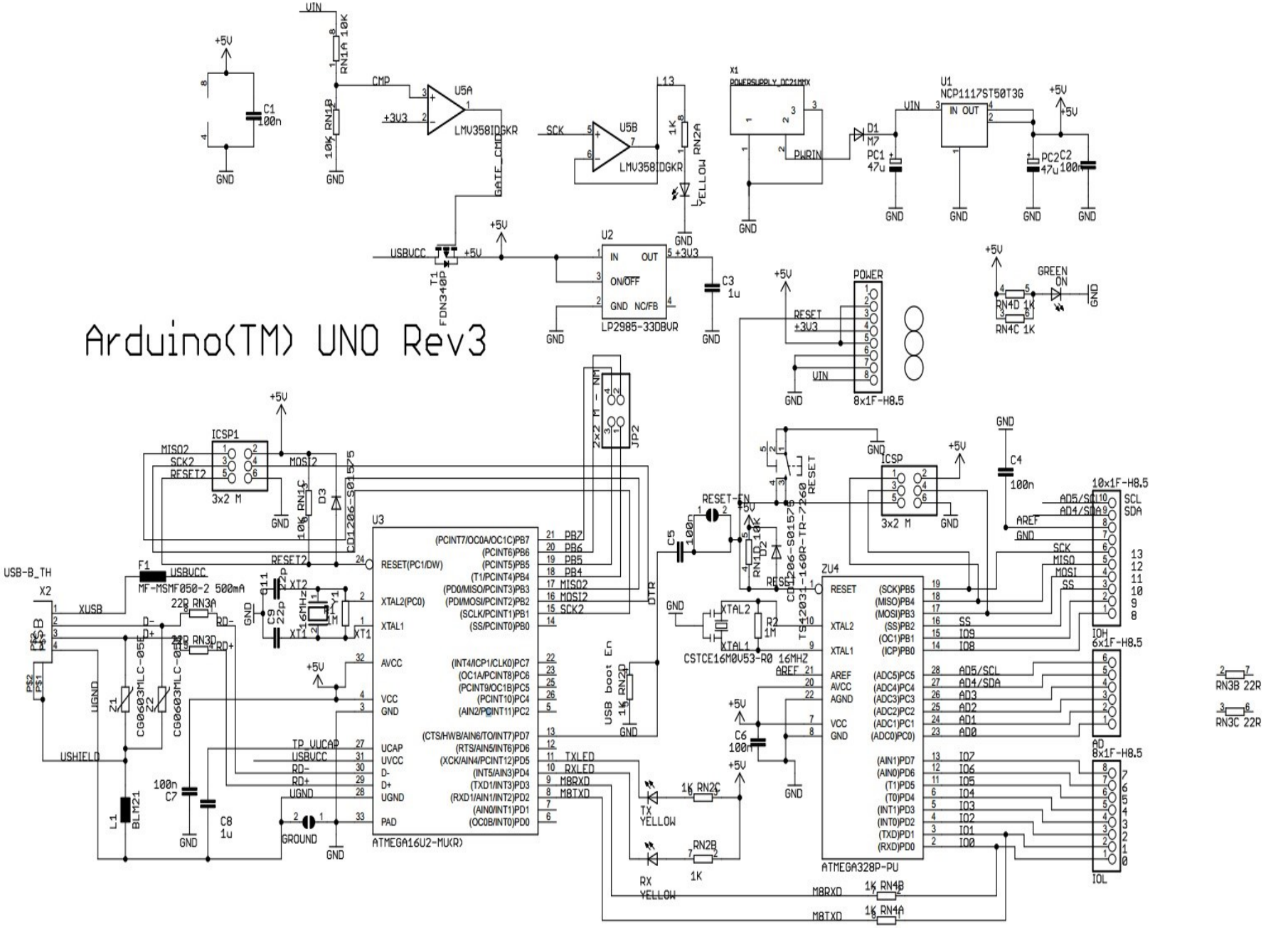
Pin-Out Version 2, V1 war fehlerhaft → Stand: 2013

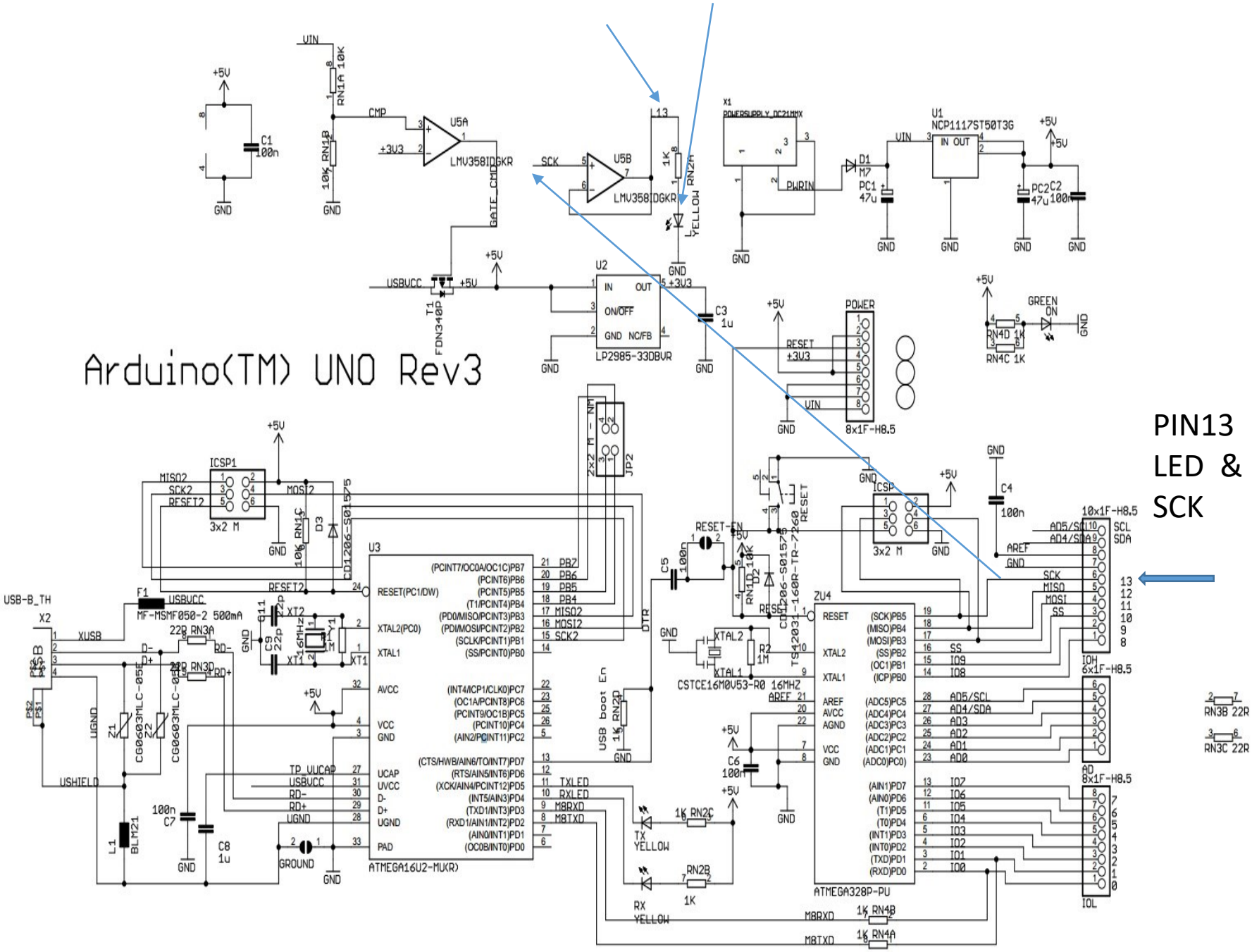
Dies Ausdrucken

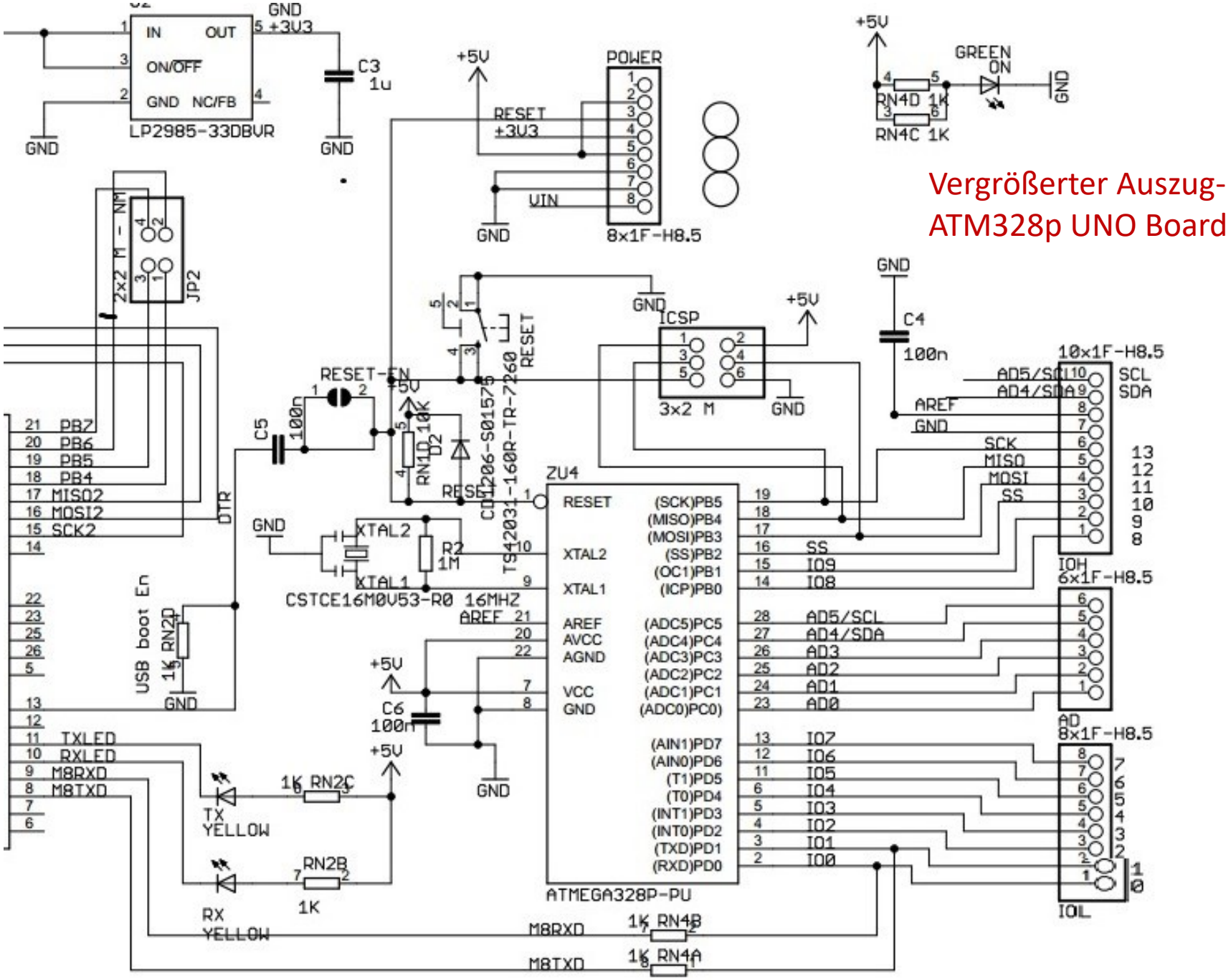


Dies ist von 2016

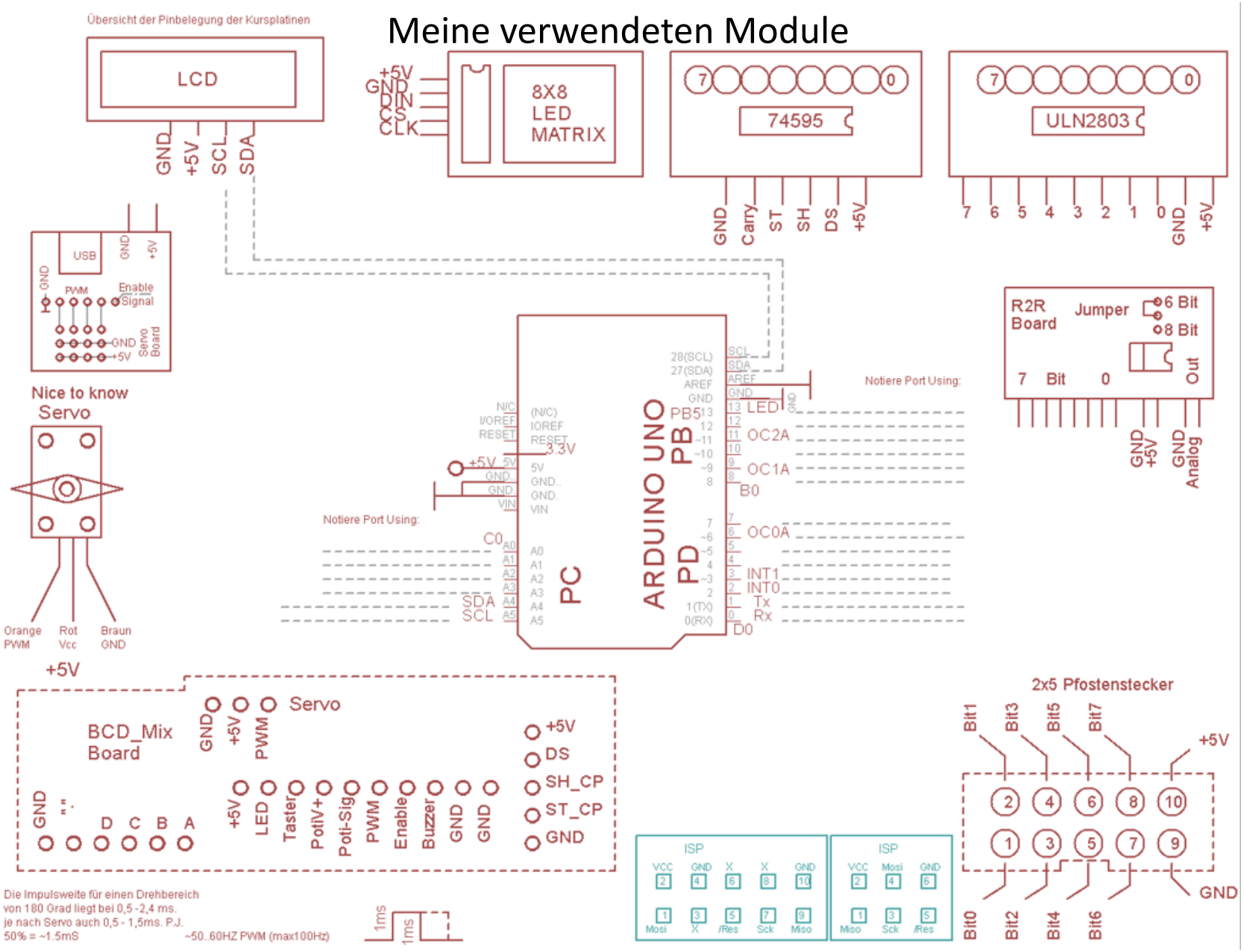



Arduino(TM) UNO Rev3
U5A
LMV358IDGKR
U5B
LMV358IDGKR
U2
LP2985-33DBVR
U1
NCP1117ST50T3G
X1
POWERSUPPLY_DC21MMX
U3
ATMEGA16U2-MUR
ZU4
ATMEGA328P-PU
ICSP1
3x2 M
ICSP
3x2 M
POWER
8x1F-H8.5
10x1F-H8.5
IOH
6x1F-H8.5
AD
8x1F-H8.5
IOL
USB-B_TH
X2
F1
MF-MSMF050-2 500mA
CSTCE16M0V53-R0 16MHZ
TS42031-160R-TR-7260
CD1206-S01575
FDN340P
YELLOW
GREEN
ON
TX
YELLOW
RX
YELLOW
RN3B 22R
RN3C 22R

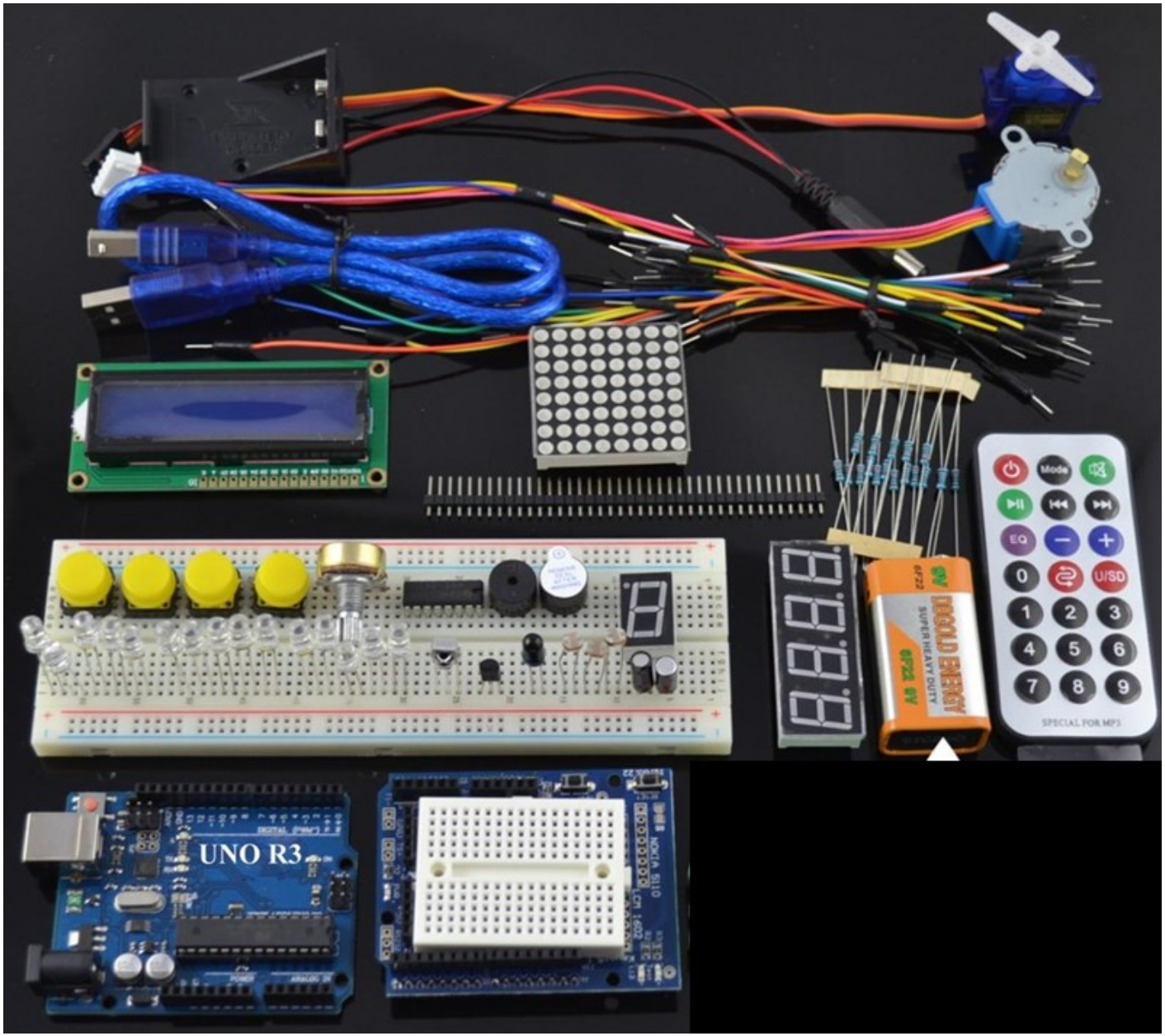

Arduino(TM) UNO Rev3
PIN13
LED &
SCK
U5A
LMV358IDGKR
U5B
LMV358IDGKR
U2
LP2985-33DBVR
U1
NCP1117ST50T3G
X1
POWERSUPPLY_DC21MMX
U3
ATMEGA16U2-MU(R)
ZU4
ATMEGA328P-PU
ICSP1
ICSP
JP2
POWER
8x1F-H8.5
10x1F-H8.5
6x1F-H8.5
8x1F-H8.5
X2
USB-B_TH
F1
MF-MSMF050-2 500mA
CSTCE16M0V53-R0 16MHZ
FDN340P
CD1206-S01575
TS42031-160R-TR-7260
CG0603MLC-05E
BLM21
GROUND
USHIELD
UGND
USBVCC
VIN
GND
+5V
+3V3
RESET
RESET-EN
USB boot En
TX
YELLOW
RX
YELLOW
GREEN
ON
M8RXD
M8TXD
TXLED
RXLED
RN3B 22R
RN3C 22R


Vergrößerter Auszug-
ATM328p UNO Board
LP2985-33DBVR
IN
OUT
ON/OFF
GND
NC/FB
+3V3
C3
1u
POWER
8x1F-H8.5
RESET
+3V3
VIN
+5V
GREEN
ON
RN4D 1K
RN4C 1K
JP2
2x2 M - NM
ICSP
3x2 M
C4
100n
10x1F-H8.5
AD5/SCL
AD4/SDA
AREF
SCK
MISO
MOSI
SS
SCL
SDA
IOH
6x1F-H8.5
AD
8x1F-H8.5
IOL
RESET-EN
C5
100n
RN1D 10K
D2
CD1206-S01575
TS42031-160R-TR-7260
ZU4
XTAL2
XTAL1
R2
1M
CSTCE16M0V53-R0 16MHZ
AREF
AVCC
AGND
VCC
C6
100n
USB boot En
RN2D
1K
DTR
PB7
PB6
PB5
PB4
MISO2
MOSI2
SCK2
TXLED
RXLED
M8RXD
M8TXD
1K RN2C
TX
YELLOW
RN2B
1K
RX
YELLOW
1K RN4B
1K RN4A
ATMEGA328P-PU
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0)
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
IO9
IO8
AD3
AD2
AD1
AD0
IO7
IO6
IO5
IO4
IO3
IO2
IO1
IO0

Meine verwendeten Module



Typisches Starterkit mit Uno R3. Verschiedene Varianten mögl.

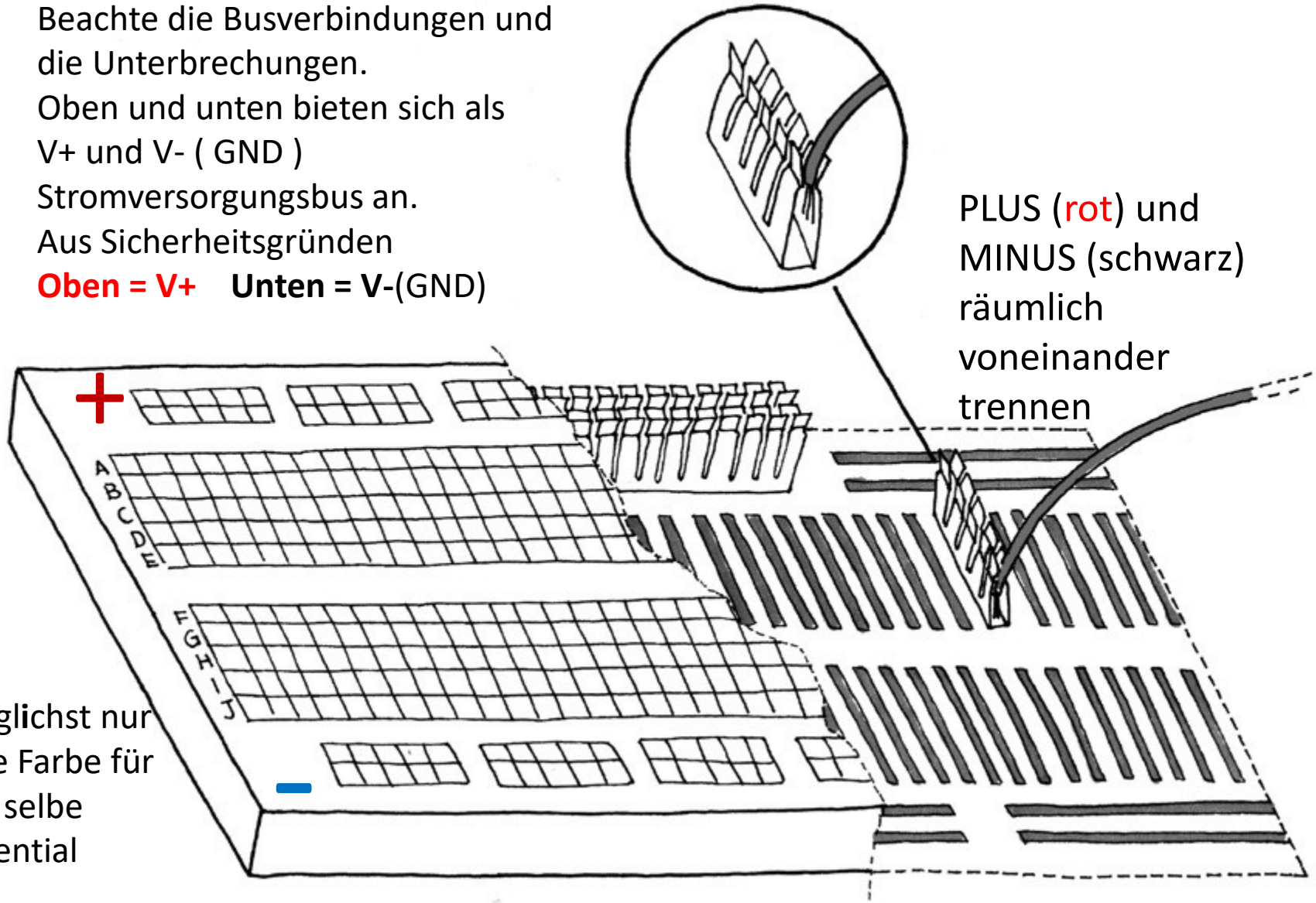


Bread Board (Steckbrett) – fliegender Aufbau

Bild vom ehemaligen Kursteilnehmer Constantin Uhde, München

Steckbrett Pin-Verbindungsansicht

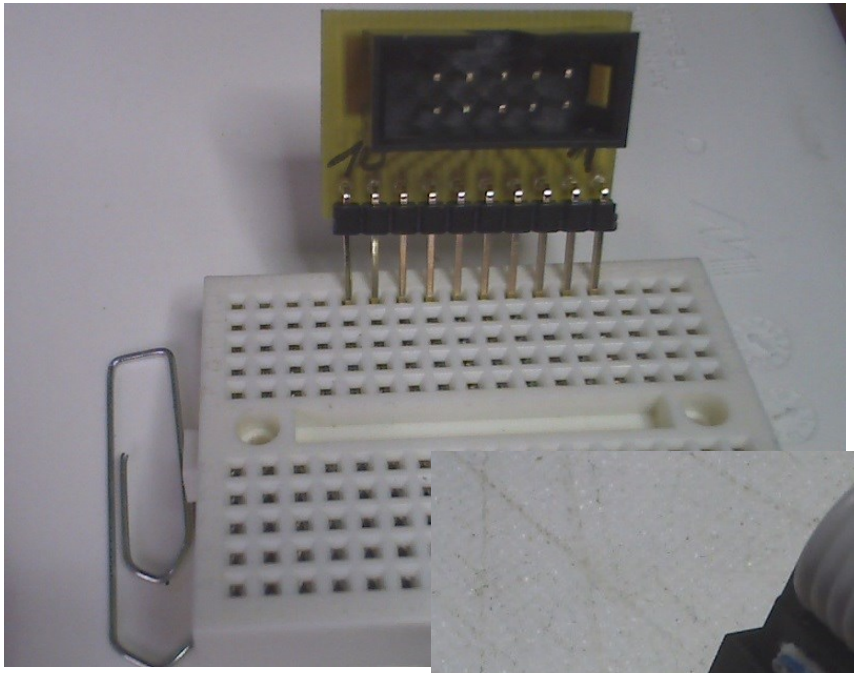
Beachte die Busverbindungen und die Unterbrechungen.
Oben und unten bieten sich als V+ und V- (GND) Stromversorgungsbus an.
Aus Sicherheitsgründen
Oben = V+ **Unten = V-**(GND)



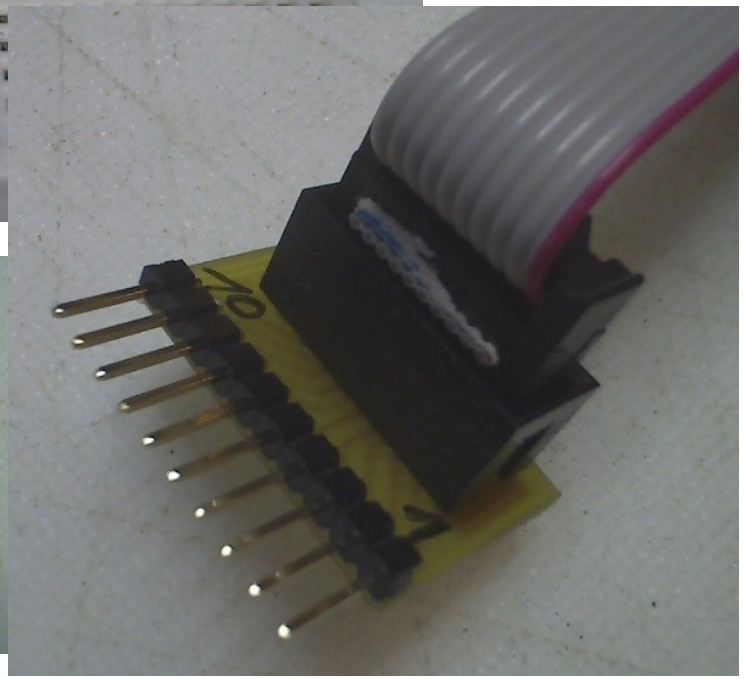
PLUS (rot) und MINUS (schwarz) räumlich voneinander trennen

Möglichst nur eine Farbe für das selbe Potential

Steckbrett Adapter



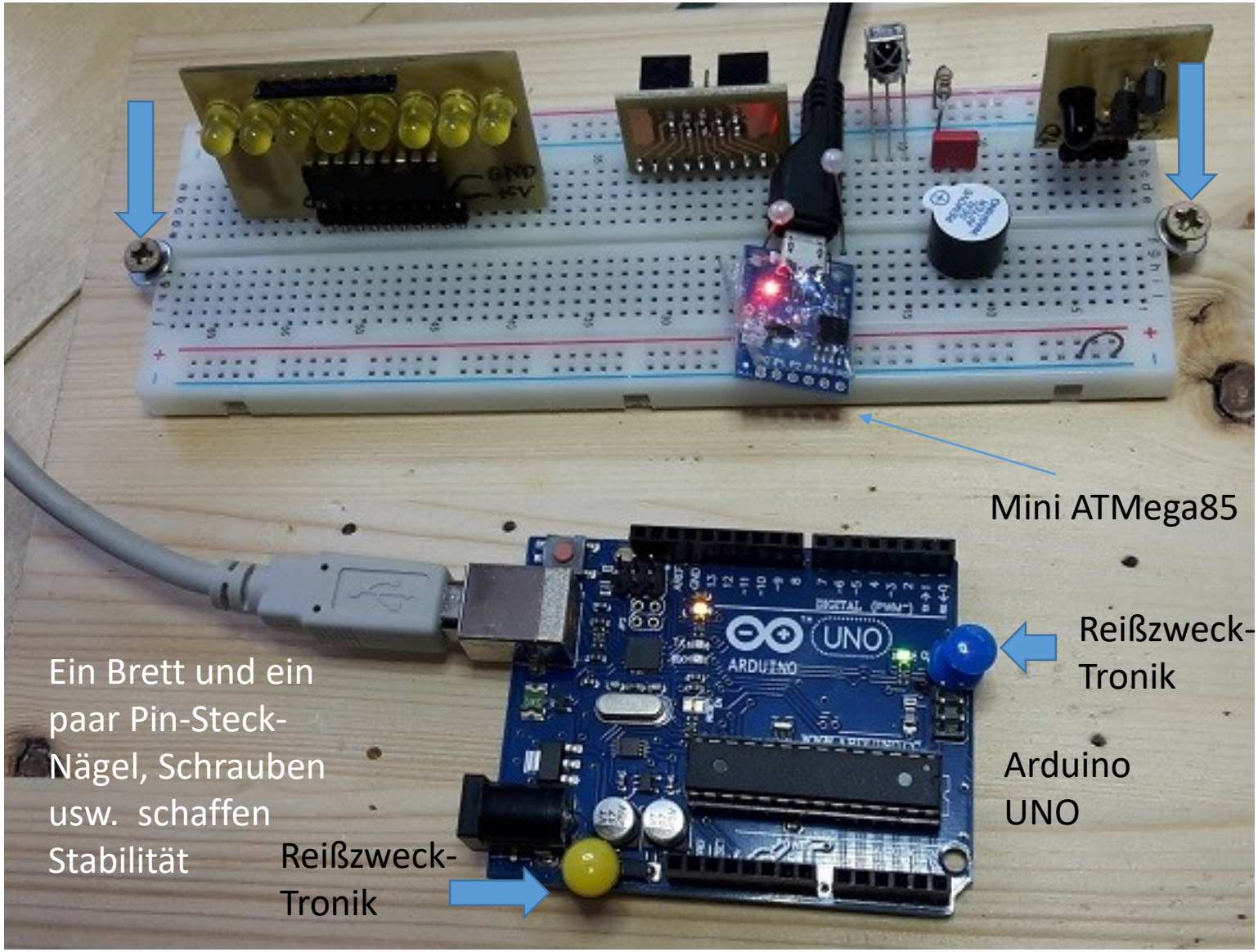
Steckbretter sind zwar ideal für **Quick and Dirty** Aufbauten. Dennoch braucht man gelegentlich etwas bessere Anschlussmöglichkeiten für externe Hardware



Praktisch:
Ein Übergang von Steckbrett auf ein 10er Flachbandkabel ist erwünscht.
Das ist aber leicht selbst zu bauen.
Ein kleines Platinchen hilft da weiter.
Evtl. gibt es das schon in der Bucht.

Steck-Brett-O-Tronik

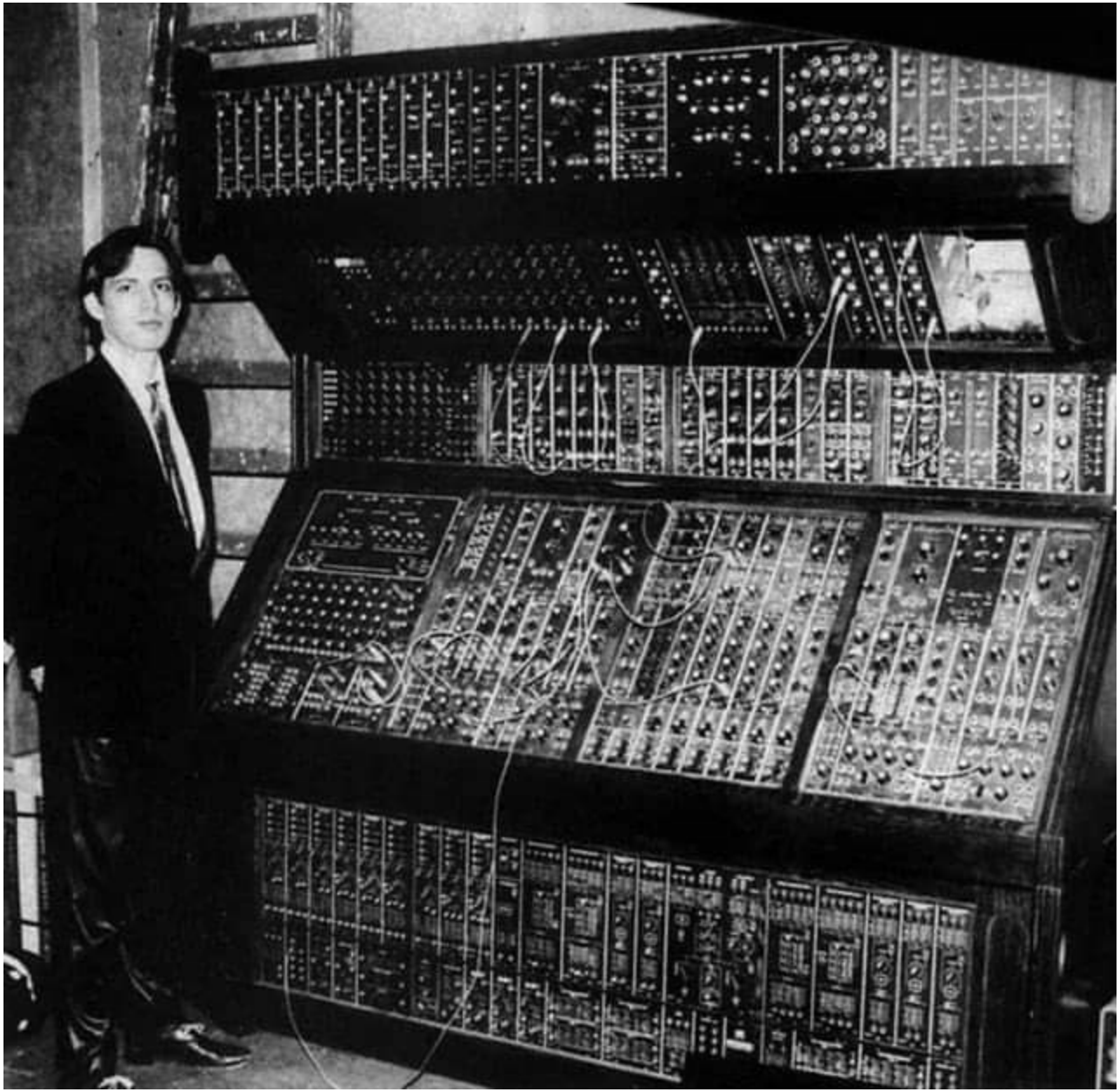
Vorschlag: Einfach, Kurzschluss
sicherer, stabiler
und es fliegt nichts rum.

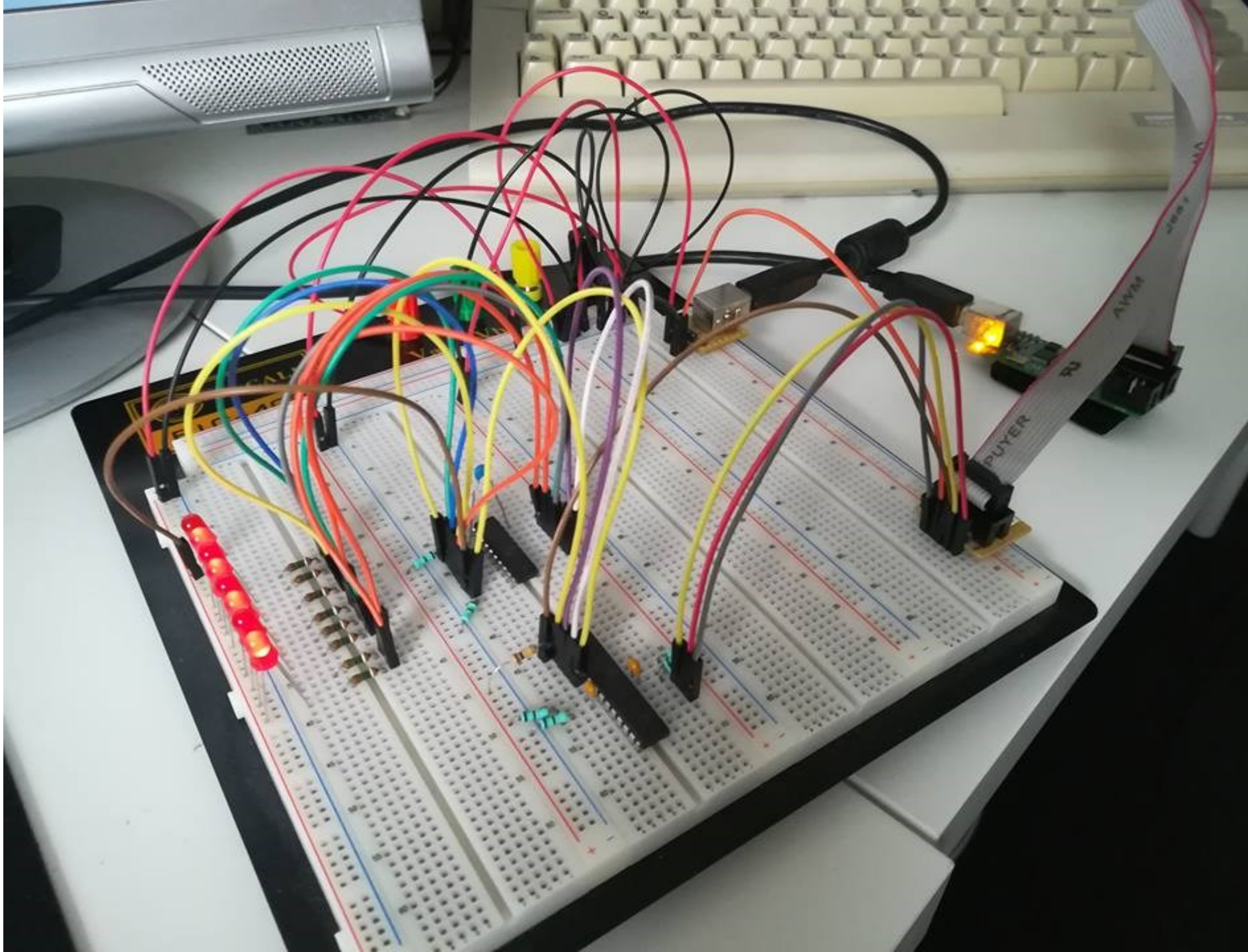


Plan B: Direkte Montage loser Komponenten auf ein Brett.
Hier ein Funkmmodul RFM69HW mit ATmega328p, LCD Modul, LED Array Modul,
Steckbrett, zsw.

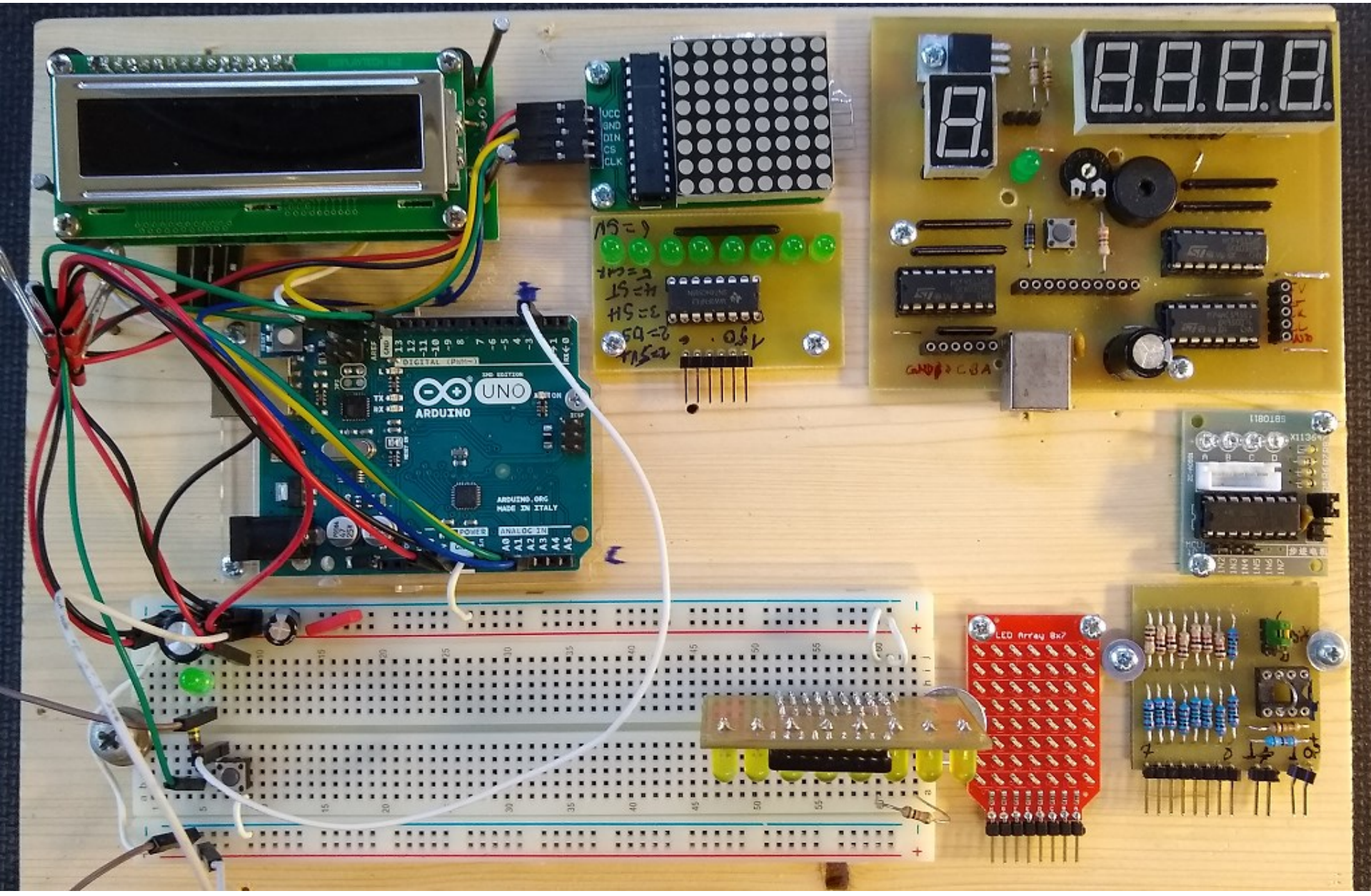
Auch wenn es auf dem ersten Block chaotisch aussieht, es liegt daran, dass es auch etwas „**Quick and Dirty“** mit Schrauben und Nägeln montiert ist.
Aber dennoch fliegen die Teile nicht lose herum!
Die Steckverbindungsdrähte haben eine stabile, wackelfreie Position.

Das ganze Brett steht dazu noch, gut sichtbar, senkrecht auf einem kleinen Schraubstock. Oder man kann es auch an die Wand hängen.




PUYER

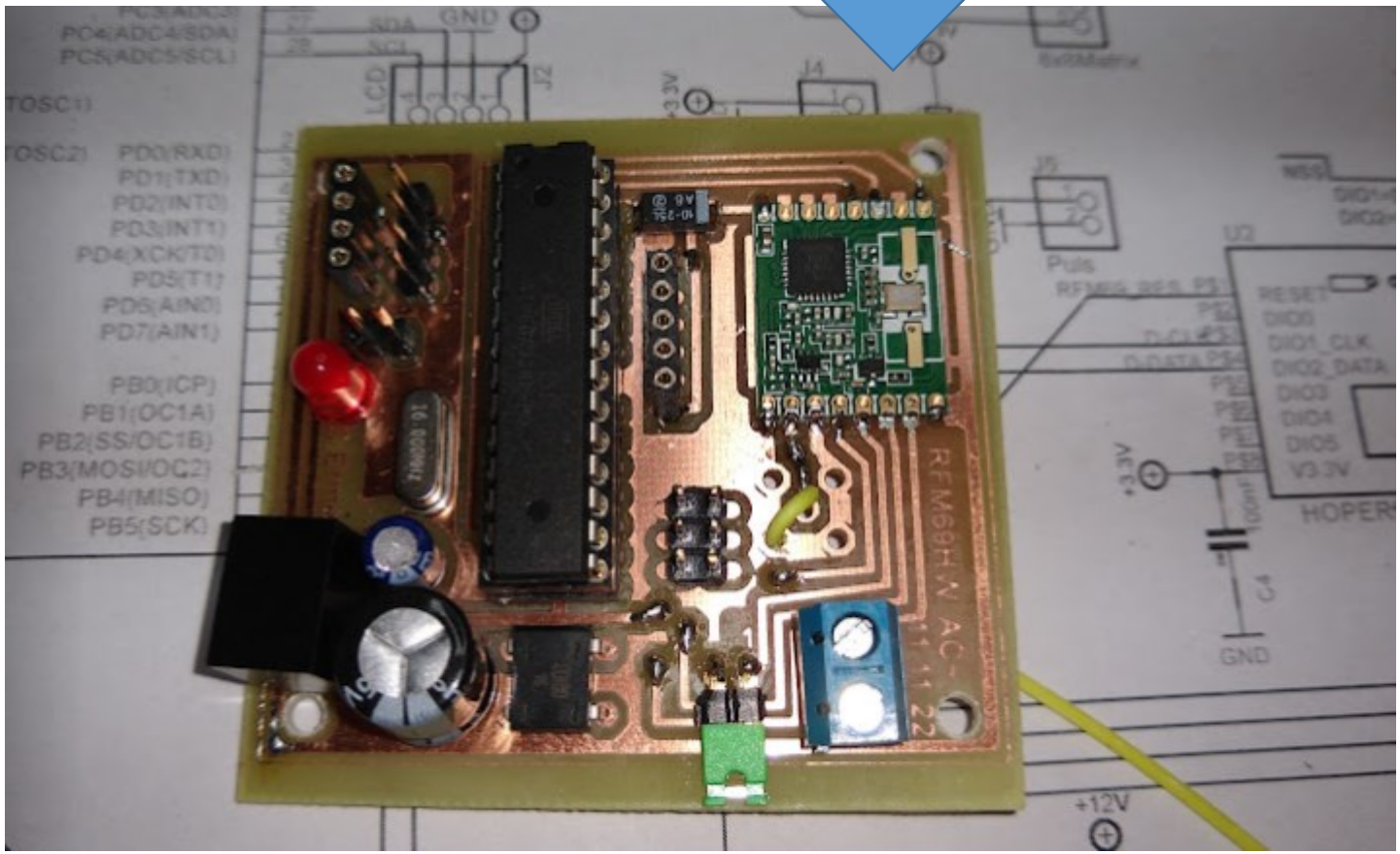
Bretter-Tronic mit einem Modul Allerlei. Aber so ist alles weitgehend, übersichtlich, Kurzschluss und wackelsicher befestigt. 2 Schrauben, Nägel reichen.



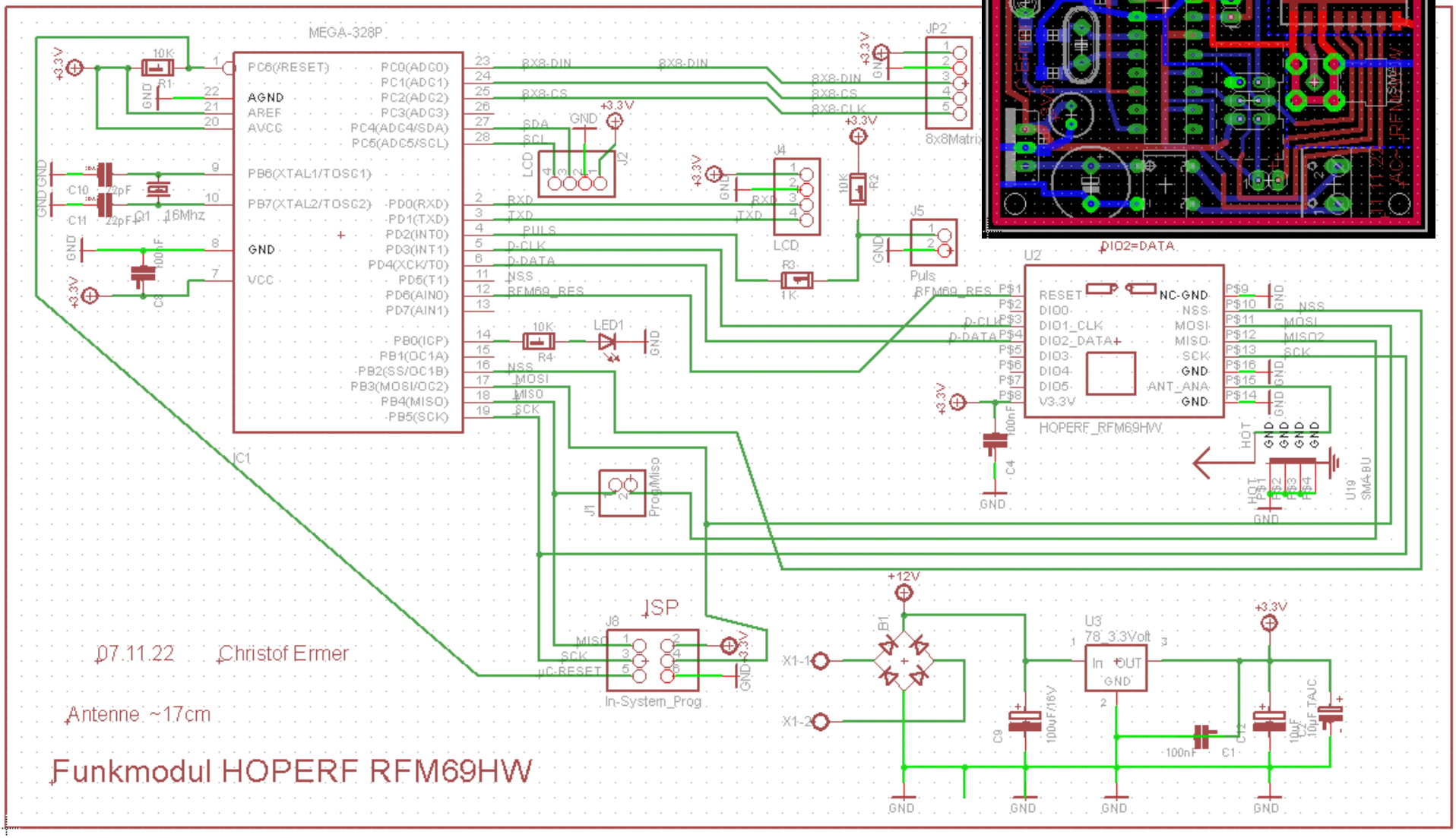
Ein typisches µC Projekt:

Funkmodul RFM69HW mit ATMega328p auf einer selbstentwickelten Platine.

Mit Anschlussbuchsen für I2C LCD Modul und Led Matrix. ISP. Anschlüssen, Stromversorgung 3.3V etc.

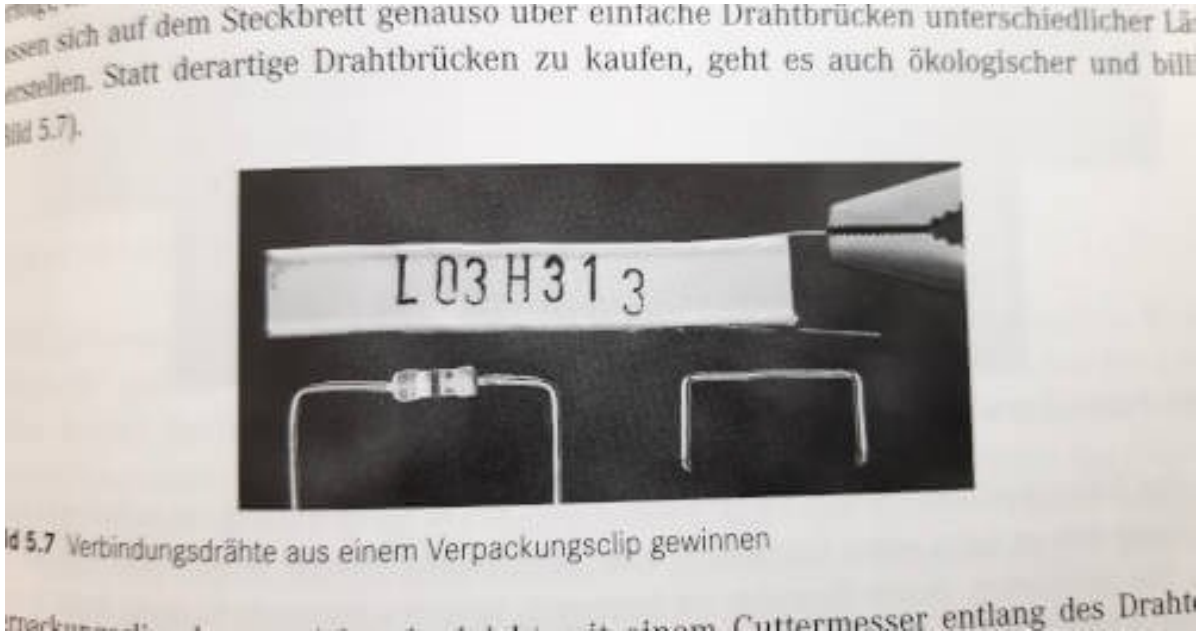


Schaltplan und Board des Funkmoduls mit ATMega328p



Der Pirat, braucht auch noch Draht!

TRICK: kein passender (blank) Draht zur Hand ? ..Hmmmm….



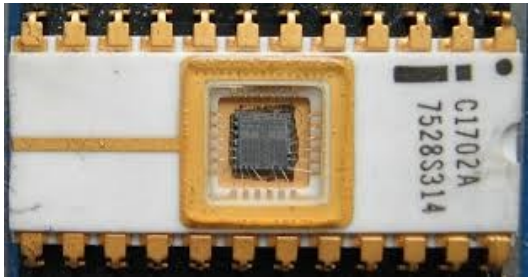
Gerade keine passenden „steifen“ Drahtreste da? Für Brücken etc.

Man kann den Draht von **Verpackungsclips** entfernen, diese passen perfekt ins Steckbrett. Entweder, Rausziehen oder Streifen halbieren.

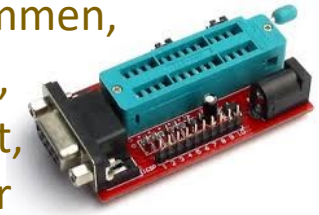
Weiterer Trick. Mini Steineschneider verkehrt herum Ansetzen und als „Klemme , nicht Schneide“ ansetzen, mit einem Ruck ziehen, um ½ bis 1 cm abzuisolieren. Ich habe nie wieder eine Abisolierzange gebraucht.

https://homepages.uni-regensburg.de/~erc24492/Verdrahtung/Verdrahtung.html

Früher musste ein Programm in ein EPROM (erasable programmable read only memory) geladen werden. Das Programm in den EEPROM Chip laden nennt man „**brennen**“.



Nach dem Programmieren musste das EPROM eingesetzt, wieder rausgenommen, mit **UV** gelöscht, neu programmiert, eingesetzt, wieder rausgenommen, usw….
Ein mühsamer Turnaround.

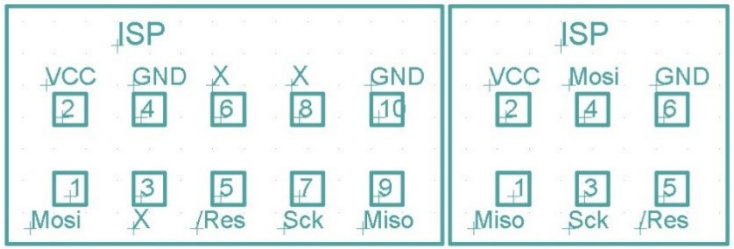


Heute kann man in moderne Mikrocontroller das neue Programm einfach laden (**flashen**). Nur wie?

Variante 1. Ein festes „speicherresistentes“ Programm → der **Bootloader**.
Damit wird mittels eines PC-Ladeprogramms , über die serielle Schnittstelle, das ***.hex** File in den **µC**ontroller geladen. →Langsam, aber kein weiterer Aufwand.

Variante 2. (die Bessere), ein **I**n**S**ystem**P**rogrammer → **ISP**, erfordert jedoch eine kleine Zusatzhardware, die den USB des PC mit der Hardware verbindet.
Dazu bringt die AVR Familie eine besondere Schnittstelle. (mit extra Pins)

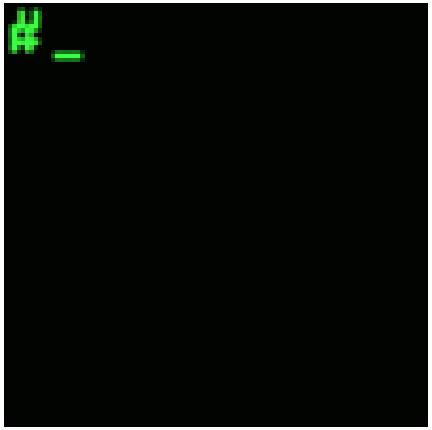
/Reset = was der Name sagt
MOSI = Master Out, Slave IN
MISO = Master In, Slave Out
SCK = System Clock + VCC + GND



Standard ISP Stecker

Der Arduino Bootloader

..ist ein kleines resistentes Programm an der Speicherspitze im Mikrocontroller.
Dieses erlaubt das **Flashen** (Aufladen) der Programmes, als **main.HEX** File, über die integrierte serielle Schnittstelle.
Trickreich, …doch das ist eine recht unsichere Methode.
Aber man braucht keine weitere externe Hardware wie den **USBasp**.
Um ein **.hex** File (hex ist das kompilierte Programm) in den Arduino zu laden kann man das „build-in“ Programm **avrdude.exe** nutzen.
Aktuelle avrdude.conf, avrdude.exe und libusb0.dll.
System wide configuration file is
"C:\WinAVR-20100110\bin\avrdude.conf“

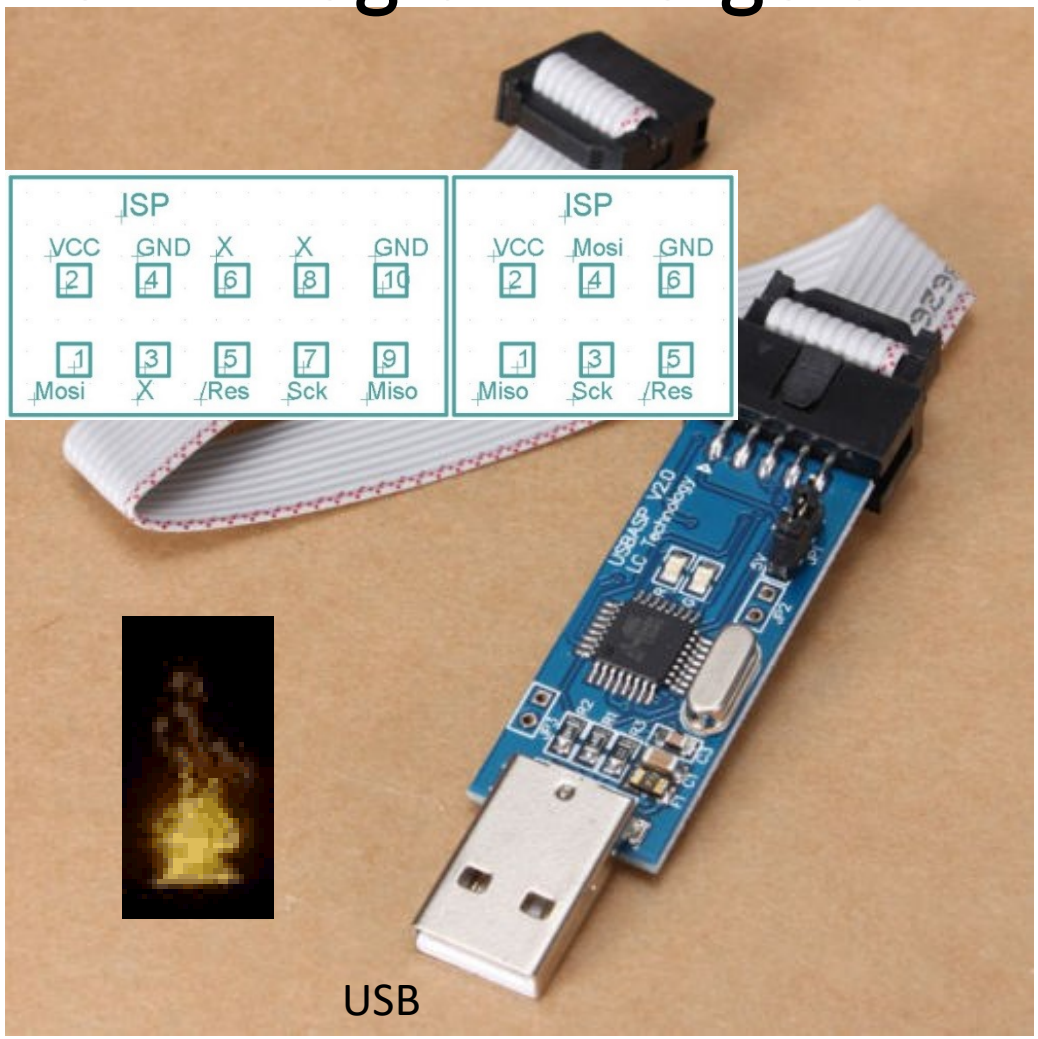


Benötigte Dateien:

So sieht beispielsweise die Auto-Programm-Flash Befehlszeile im „make“-file aus:
WINAVR mit dem GNU-C++

AVRDUDE_FLAGS = -p$(atmega328p) -P$(com8) -b$(115200) -c $(arduino)

ISP - Programmiergerät



Zusätzlich sinnvoll und ohne Bootloader erforderlich:
USBasp In **S**ystem **P**rogrammer (**ISP**).
Mit Arduino UNO R3
ICSP Adapter von 10 auf 6 Pin

Direkt aus WINAVR C-Compiler programmieren!
..mit integriertem **AVRDUDE**

6x10
ISP-Stecker
Adapter

ISP ist eine entschieden stabilere und bessere Lösung als mit einem **Bootloader**.
(Bootloader sind begrenzt, brauchen Platz und sind schnell mal tot=überschrieben, gehen also nur, wenn alles in Ordnung ist).

Arduino Sketch ?

„Der schnelle Weg, ist der verführerische Weg, der leichte Weg.
Er führt zur dunklen Seite der Macht“. (Meister Yoda)

Viel Spricht für das Arduino eigene „Sketch“ genannte „C“ mit Library.
Integrierter Compiler,
viele Bibliotheken,
einheitliche IDE (integrated development environment)
für Einsteiger geeignet.
http://www.arduino-tutorial.de/programmieren/

Was spricht dagegen?
Man bleibt Einsteiger, denn man hat keine Ahnung warum und wie die Bibliotheken funktionieren und keine Kontrolle darüber.

Daher: keine professionelle Software- und Hardwareplattform, auf der man industrietaugliche Hardware und Software entwickelt, die vom Anspruch professionell, kommerziell oder serientauglich ist.
Von juristisch verantwortbarer Sicherheit gar nicht zu reden.
(ABS-Systeme, Motorregel Hard und Software usw.)

Möchte jemand in einem Zug sitzen, dessen Bremsen mit der Aufsteckplatine für 10€ geregelt wird, und die Hardware mit Plastikspannern und Klettband zusammengehalten wird? Möglich ist es…

Arduino Sketch Bibliotheken

Das Netz ist voll von Seiten zum Arduino, Youtubes etc.
Hier kann man selbst stöbern. → etwas googeln…

83https://www.arduino.cc/en/Reference/FunctionDeclaration

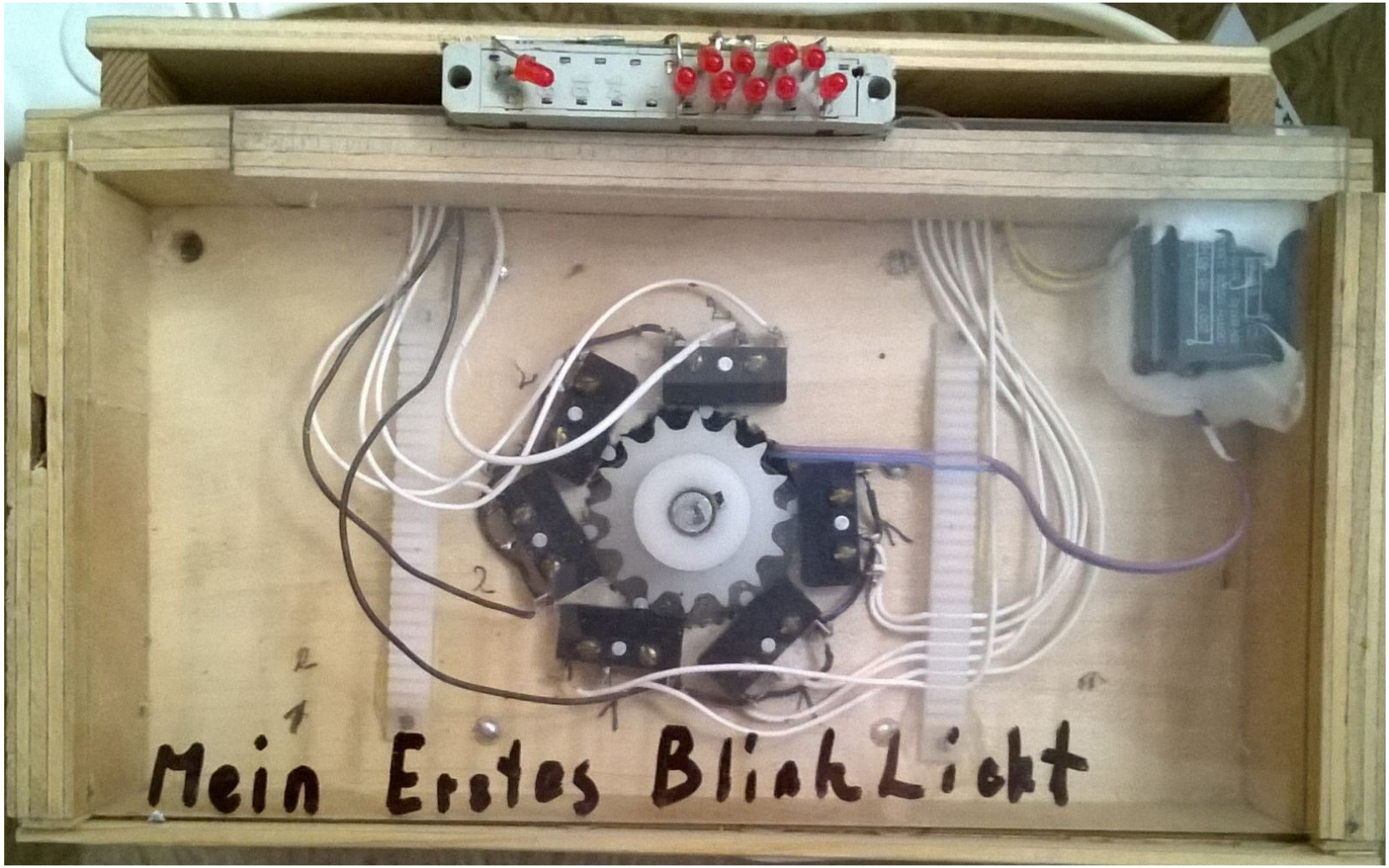
http://www.arduino-tutorial.de

http://www.netzmafia.de/skripten/hardware/Arduino/Arduino_Programmierhandbuch.pdf usw.

In diesem Kurs lernen wir den µKontroller DIREKT zu programmieren.
So werden wir **keine Bibliotheksbediner**, sondern wissen, WARUM etwas funktioniert.

MERKE: ***Wenn man ein Lämpchen blinken lassen kann, kann man alles andere auch!***

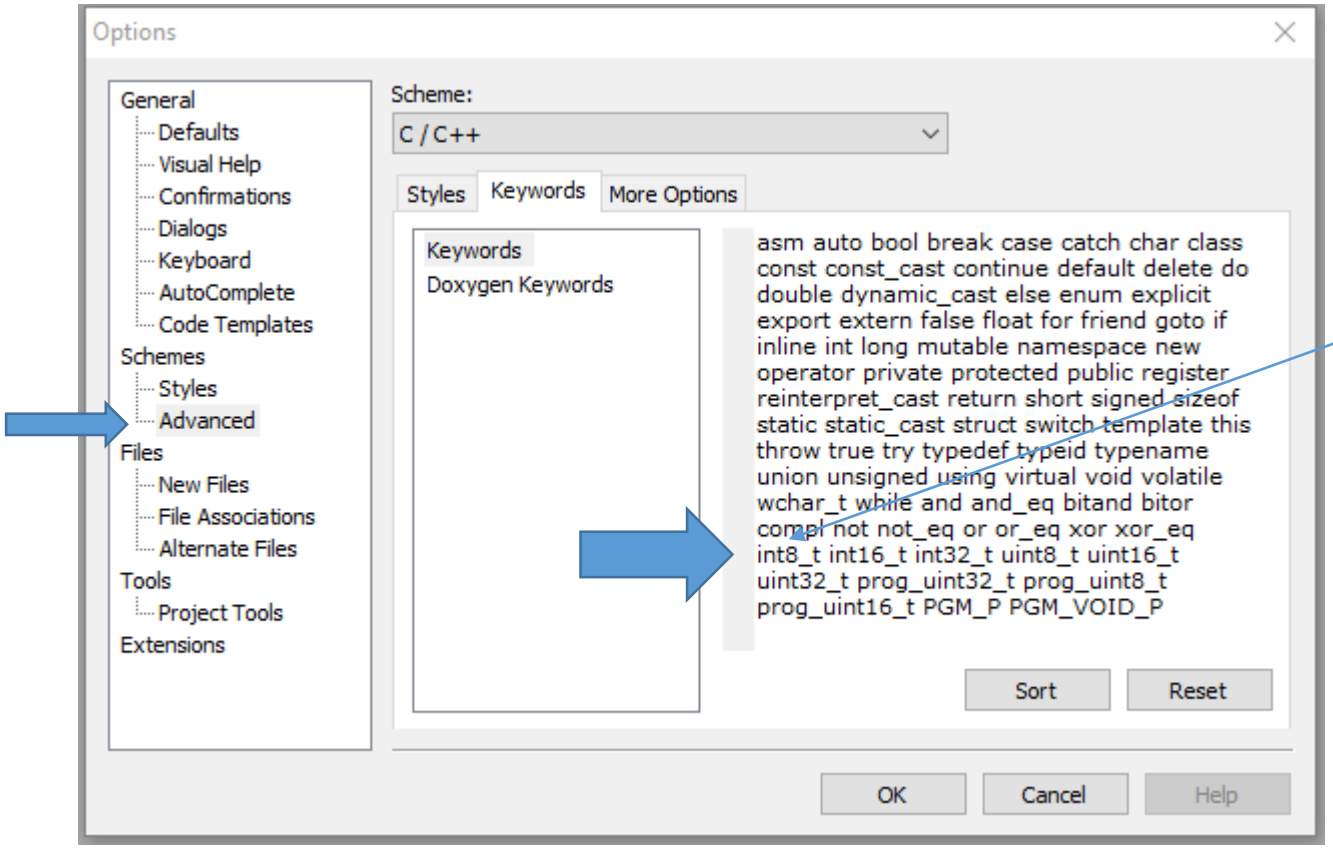
Mein erstes Blinklicht ! → es gibt wohl pfiffigere Wege zum Ziel..



Programmers Notepad (WinAVR) einstellen

Menü: Tools →Options → Advanced --> rechts (keywords) -->Wechsele SCHEME zu C/C++ , Hänge in der Liste rechts folgendes an:

int8_t int16_t int32_t uint8_t uint16_t uint32_t prog_uint32_t prog_uint8_t prog_uint16_t PGM_P PGM_VOID_P



so das es jetzt so aussieht:

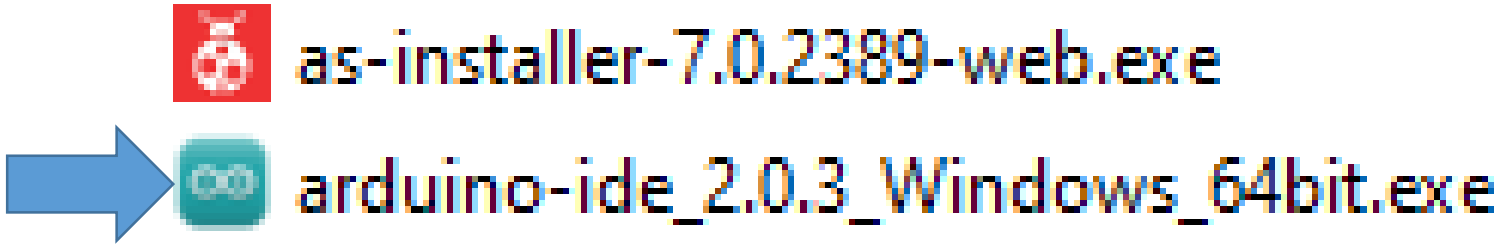
Gehe noch zu Defaults und aktiviere Show Line Numbers

Früher habe ich mit **WINAVR (2010)** gearbeitet.
Das geht immer noch. Doch es muss eine DLL händisch getauscht werden.
Der Compiler ist veraltet. Aber die Bibliotheksbeschreibung ist nach wie vor gut.

Ich wähle also eine neueres, nicht unbedingt besseres, Compiler Werkzeug.

Microchip Studio (früher AVR Studio)
https://www.microchip.com/en-us/tools-resources/archives/avr-sam-mcus
Aber diese ist völlig überladen und für Anfänger verwirrend.

Am einfachsten ist die:
Arduino DIE Stand 02.2023

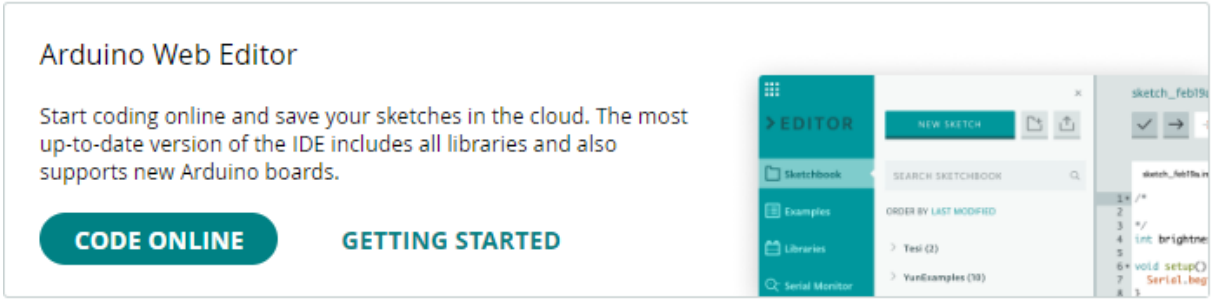


Arduino IDE installieren (Stand 2022)

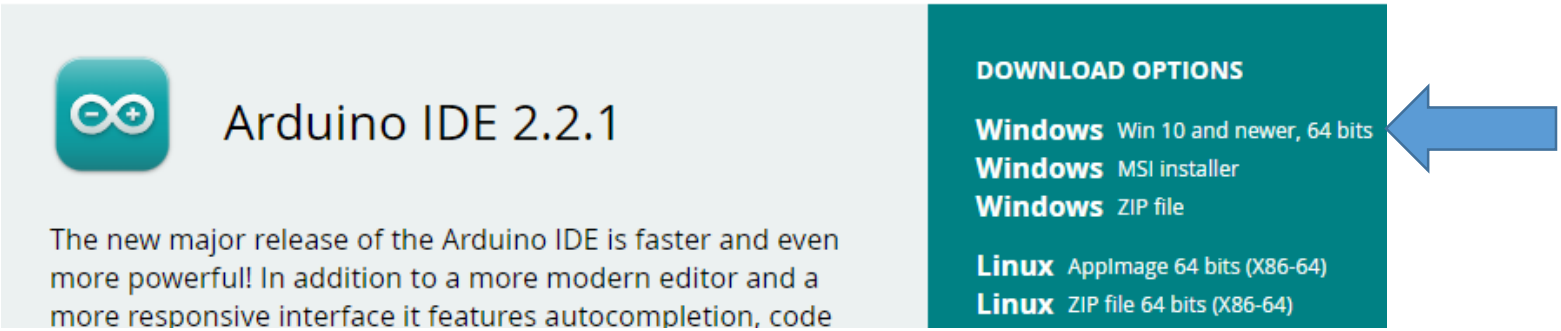
https://www.arduino.cc/en/software

IDE = Integrated Development Environment

Wer nur mit dem Arduino UNO arbeitet, kann die Arduino DIE verwenden. **„Ich“ arbeite allerdings OHNE die Sketch Bibliotheken !!**



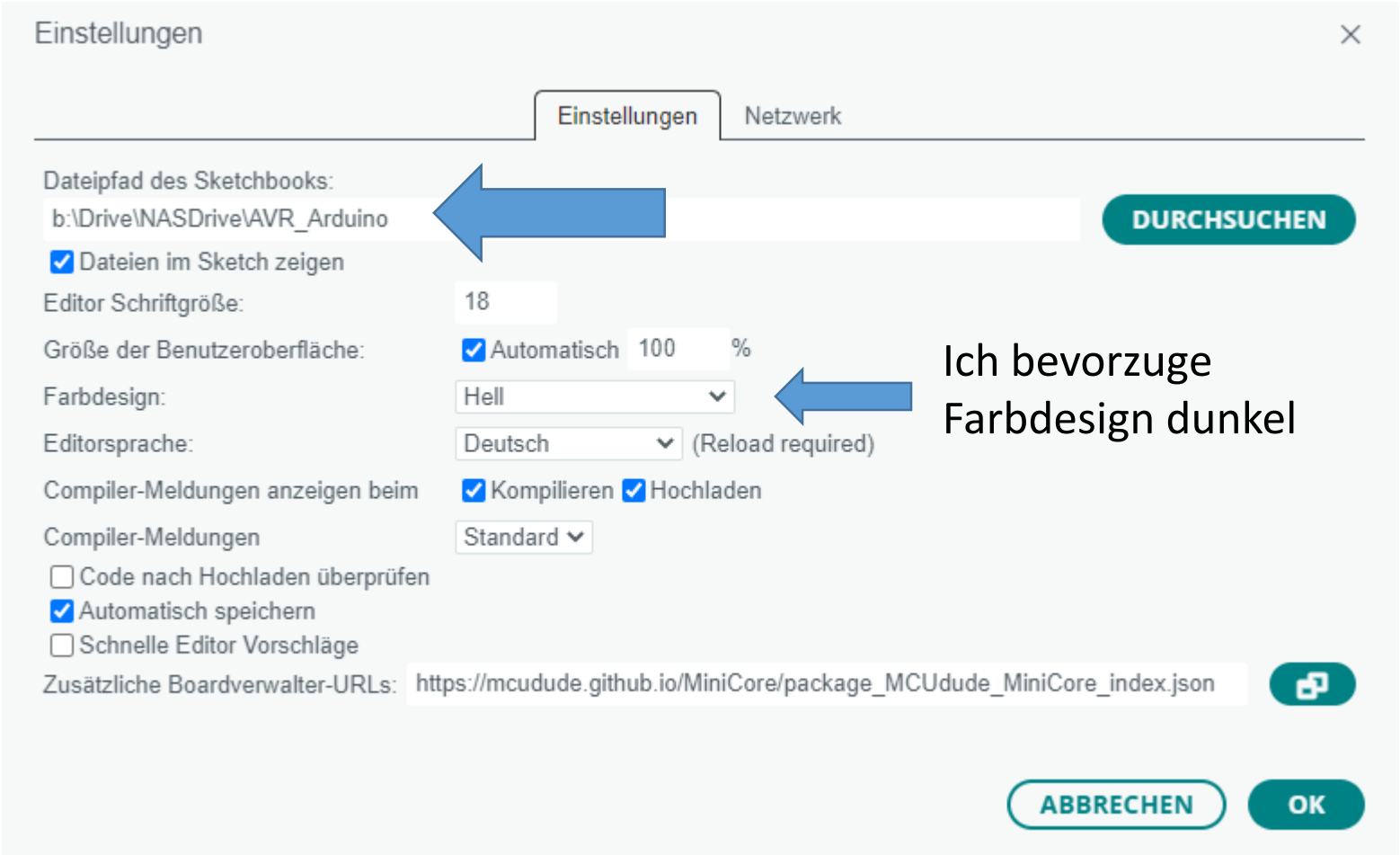
Downloads



https://www.arduino.cc/en/donate/

Vorbereitung des Arduino IDE für einen externen Editor

Datei → Voreinstellungen: Zeilennummer, Externen Editor usw.
Compiler Warnungen würde ich „ALLE“ aktivieren, auch wenn manches unverständlich ist. → **Standard Speicherort wie gewünscht** C:\..\MyAVR



Ich bevorzuge Farbdesign dunkel

SPEICHERPLATZ der Arduino DIE

D:\Program Files\Arduino IDE

..und

Windows: C:\Users\{username}\AppDat\Roaming\arduino-ide\

→ C:\Users\Christof**\AppData\Roaming**arduino-ide\

Und C:\Users\Christof\AppData\Local\Arduino15

Erweiterung der Arduino IDE

Basiswissen AT_Tiny von ***Clemens Andree***.

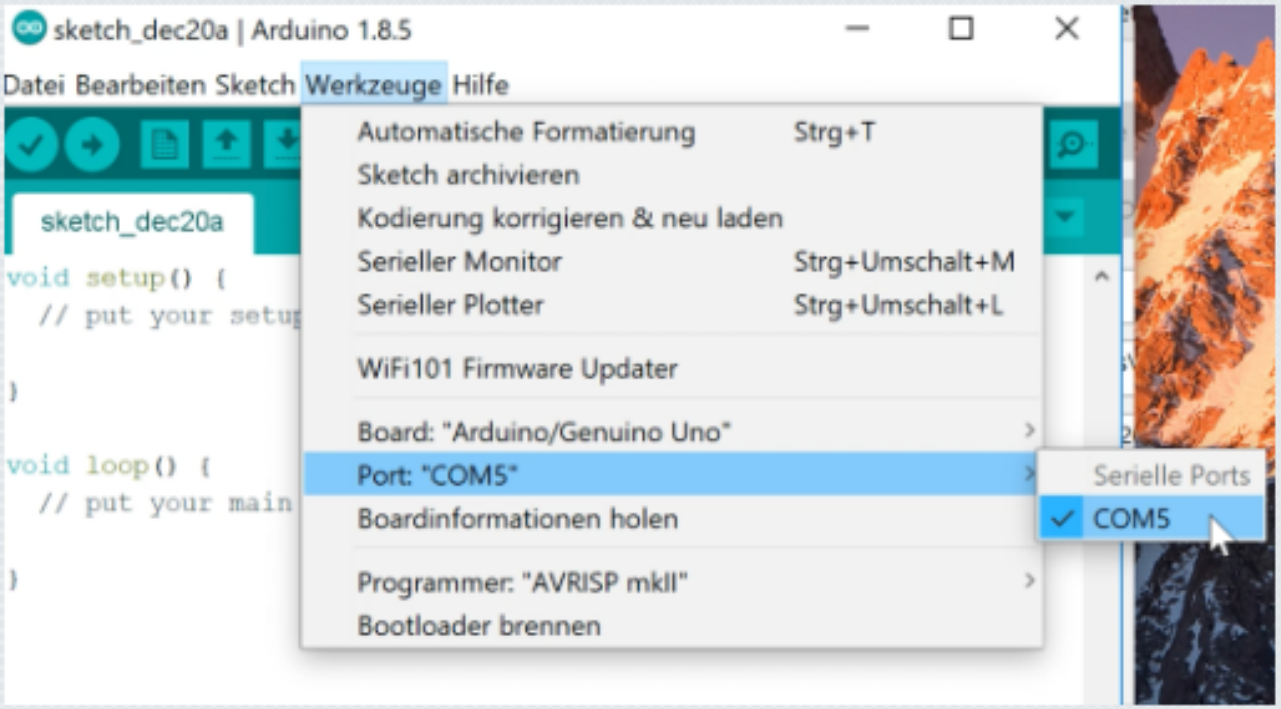
https://www.dropbox.com/sh/k4smdi65q7wm9sd/AACk7DLX16L-PueQzVcvft9Za?dl=0

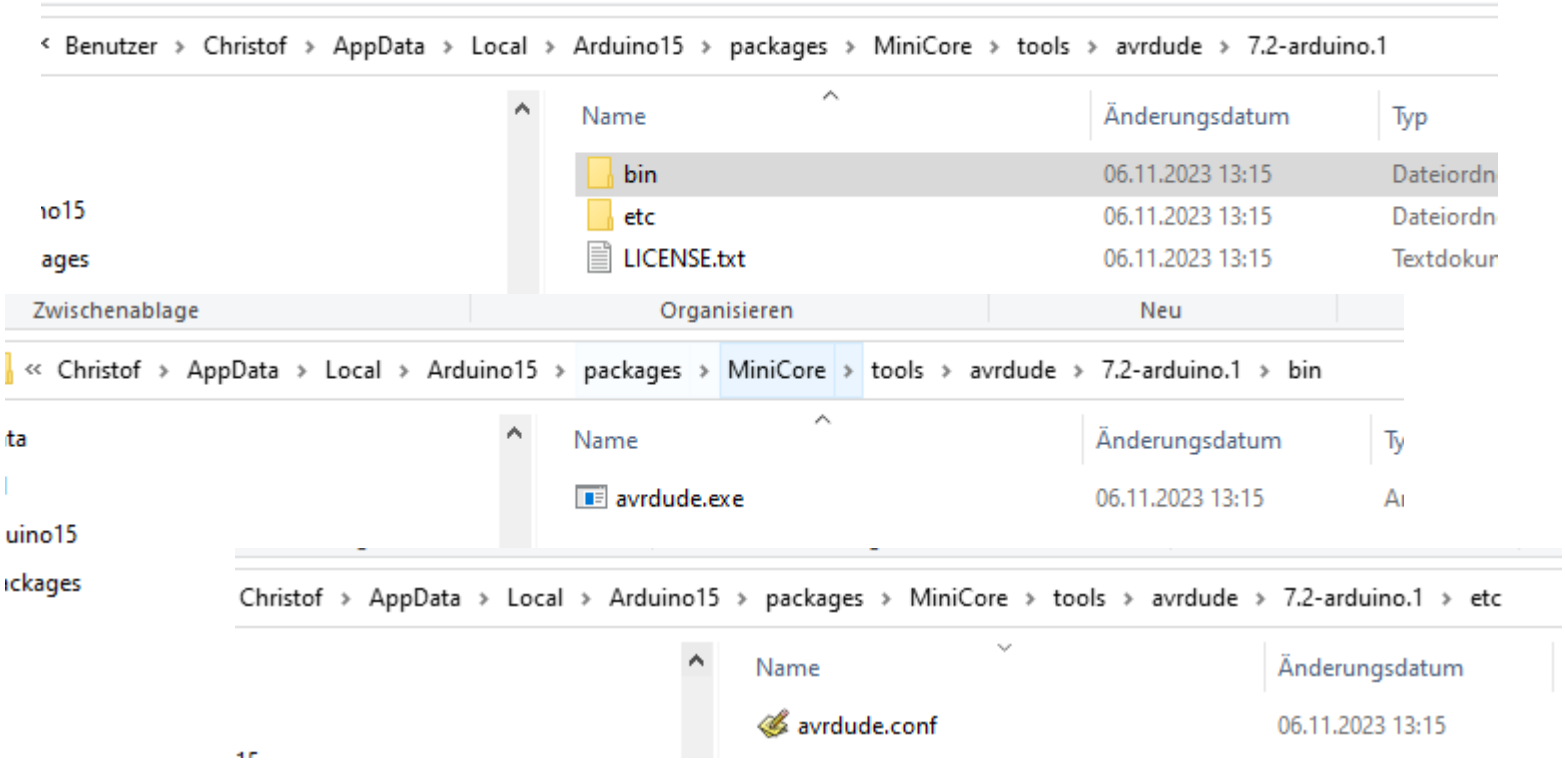
Siehe Youtube Original von **Clemens Andree**:

https://www.youtube.com/watch?v=J6g_X3onNWo&t=774s

Arduino IDE

Voraussetzung um den com-Port herauszufinden ist, dass der Arduino an dem PC via USB angeschlossen ist, Windows den passenden Treiber gefunden und installiert hat. Wenn der Vorgang fehlerfrei durchlaufen wurde, muss Arduino IDE geöffnet werden und klicken in der Toolbar auf die Schaltfläche "Werkzeuge". Unter Werkzeuge finden man weiter unten den Eintrag "Port", sobald man mit der Maus über den Eintrag "Port" geht öffnet sich ein weiterer Reiter und dort können wir dann unseren aktiven com-Port sehen.





C:\Users\Christof\AppData\Local\Arduino15\packages\MiniCore\tools\avrdude\7.2-arduino.1

C:\Users\Christof\AppData\Local\Arduino15\packages\MiniCore\tools\avrdude\7.2-arduino.1\bin

C:\Users\Christof\AppData\Local\Arduino15\packages\MiniCore\tools\avrdude\7.2-arduino.1\etc

Erweiterung der Arduino IDE für µController wie ATTiny, ATMega8

MIT ARDUINO UNO EINEN AVR CONTROLLER PROGRAMMIEREN.

Der Arduino soll in dem Fall als **ISP Programmer** arbeiten. Dafür folgende Schritte durchführen:

Nur bei erstmaliger Verwendung:

Datei / Voreinstellungen / klicken und das Feld

„zusätzliche Bordverwalter-URLs“ suchen und dieses Symbol klicken: 
Dort jeweils in neuen Zeilen eintragen:

https://mcudude.github.io/MiniCore/package_MCUdude_MiniCore_index.json

https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json

http://drazzy.com/package_drazzy.com_index.json

(erste Zeile ist für ATMEGA, zweite und dritte Zeile für ATTINY)

Dann weiter:

Werkzeuge / Board / Boardverwalter klicken und oben im Textfeld dann das Wort MiniCore eintragen, die gefundene Datei wählen und anschließend auf installieren klicken.

Das wiederholt man noch mal mit dem Suchbegriff Attiny und installiert auch **attiny** von David A. Mellis und **ATTinyCore** von Spence Konde.

Arduino Uno

BOARD-VERWALTUNG

ATMega16

Typ: Alle

Atmel AVR Xplained-minis von Atmel University France

Boards included in this package: atmega168pb-xmini, atmega328pb-xmini, atmega328p-xmini

Mehr Information

0.6.0 INSTALLIEREN

MightyCore von MCUdude

Boards included in this package: ATmega1284/P, ATmega644/P/PA/A, ATmega324P/PA/A/PB, ATmega164P/PA/A, ATmega32, ATmega16,...

Mehr Information

3.0.0 INSTALLIEREN

MiniCore von MCUdude

3.0.0 installiert

Boards included in this package: ATmega8, ATmega328, ATmega168, ATmega88, ATmega48

Mehr Information

3.0.0 ENTFERNEN

main.ino | main.h | MultiPWM_T2.c | MultiPWM_T2.h | Befel

```
22 #include <ctype.h>
23 #include <inttypes.h>
24 #include <avr/io.h>
25 #include <string.h>
26 #include <stdlib.h>
27 #include <avr/pgmspace.h>
28 #include <math.h>
29 //#include <stdarg.h>
30 #include <avr/interrupt.h>
31 #include <avr/wdt.h>
32 #include <util/delay.h>
33
34 extern "C" {  // ohne dies, refe
35 #include "main.h"
36 #include "debug.h"
37 #include "uartATM328.h"
38 #include "ServiceFunc.h"
39 #include "TextService.h"
40 #include "Befehle.h"
41 #include "max7219.h"
42 #include "8LED_74-595_Out.h"
43 #include "PCF8574.h"
44 #include "I2cmaster.h"
```

Ausgabe

Ab hier beginnt der Programmiervorgang:

1.) **Datei / Beispiele /** 11.ArduinoISP auswählen
2.) **Werkzeuge / Board /** Arduino Uno auswählen
3.) **Werkzeuge / Port** wie gewohnt den Port auswählen
4.) **Werkzeuge / Programmer /** AVRISP MKII auswählen
5.) **Sketch / Hochladen** auswählen oder alternativ ➲ klicken

Oder USBasp

Damit arbeitet der Arduino jetzt als **ISP Programmer**.

AVR-Chip auf Steckbrett platzieren und Pins mit Arduino verbinden:

Arduino Uno	Funktion	Tiny 13/25/45/85	Tiny 2313	Mega 8
5V	5V	8	20	7
GND	GND	4	10	8
10	Reset	1	1	1
11	MOSI	5	17	17
12	MISO	6	18	18
13	SCK	7	19	19
10µF Elko von Arduino (Reset-Pin) nach GND erst jetzt anschließen!				

main | Arduino IDE 2.2.1

Datei Bearbeiten Sketch Werkzeuge Hilfe

- Neu Strg + N
- Öffnen... Strg + O
- Zuletzt geöffnet
- Sketchbook
- Beispiele
- Schließen Strg + W
- Save Strg + S
- Speichern unter... Strg + Umschalttaste + S
- Einstellungen... Strg + Komma
- Fortgeschritten
- Beenden Strg + Q

main.ino | main.h | MultiPWM_T2.c | MultiPWM_T2.h | Befehle.c | Befehle.h | 8LED74-595_Out.c | 8LED_74-595_Out.h | I2cmaster.c | PCF8574.c

```
22 #include <ctype.h>
23 #include <inttypes.h>
24 #include <avr/io.h>
25 #include <string.h>
26 #include <stdlib.h>
27 #include <avr/pgmspace.h>
28 #include <math.h>
29 //#include
30 #include <
31 #include <
32 #include <
33
34 extern "C"
35 #include "
36 #include "
37 #include "
38 #include "
39 #include "
40 #include "
41 #include "
42 #include "
43 #include "
44 #include "
```

Ausgabe

MightyCore von MCUdude

Boards included in this package: ATmega1284/P, ATmega644/P/PA/A, ATmega324P/PA/A/PB, ATmega164P/PA/A, ATmega32, ATmega16,...
Mehr Information

3.0.0 INSTALLIEREN

MiniCore von MCUdude

3.0.0 installiert

Boards included in this package: ATmega8, ATmega328, ATmega168, ATmega88, ATmega48
Mehr Information

3.0.0 ENTFERNEN

Einstellungen

Einstellungen | Netzwerk

Dateipfad des Sketchbooks: b:\Drive\NASDrive\AVR_Arduino DURCHSUCHEN

☑ Dateien im Sketch zeigen

Editor Schriftgröße: 18

Größe der Benutzeroberfläche: ☑ Automatisch 100 %

Farbdesign: Dunkel

Editorsprache: Deutsch (Reload required)

Compiler-Meldungen anzeigen beim ☑ Kompilieren ☑ Hochladen

Compiler-Meldungen Standard

☐ Code nach Hochladen überprüfen

☑ Automatisch speichern

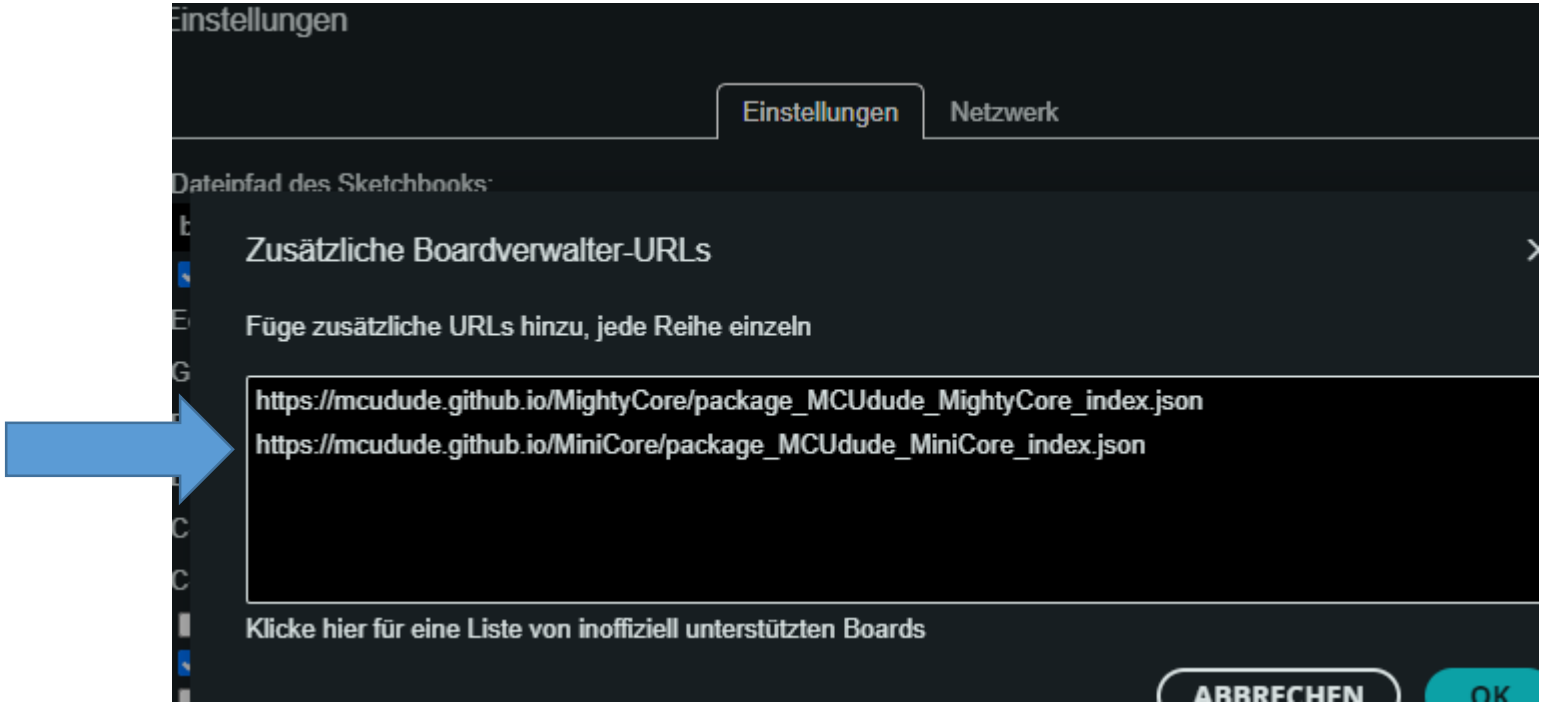
☐ Schnelle Editor Vorschläge

Zusätzliche Boardverwalter-URLs: https://mcudude.github.io/MightyCore/package_MCUdude_MightyCore_index...

ABBRECHEN OK

https://forum.arduino.cc/t/uploading-a-sketch-atmega16-using-arduino-ide/512445
https://github.com/MCUdude/MightyCore#pinout

Um Z.B: den ATMega16 via USBasp Programmer einzubinden, schreibe das in die zusätzliche Datei→ Einstellungen→ zusätzliche Boardverwaltung:
https://mcudude.github.io/MightyCore/package_MCUdude_MightyCore_index.json



ODER SO: (BEISPIEL ATTINY85)

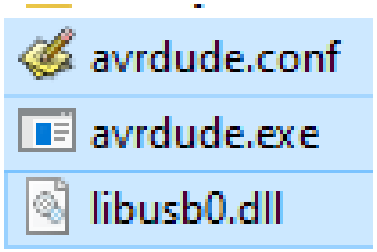
1.) **Werkzeuge / Board /ATTiny Microcontrollers /** ATTiny25/45/85 auswählen
2.) **Werkzeuge / Prozessor /** ATTiny85 auswählen
3.) **Werkzeuge / Clock / intern** 1MHz auswählen (siehe ATMega8)
4.) **Werkzeuge / Programmer /** Arduino **as** ISP auswählen
Nicht verwechseln mit ArduinoISP !!!

Meine 1. Wahl AVR Studio oder AVRStudio

https://www.microchip.com/en-us/tools-resources/archives/avr-sam-mcus

Diese IDE ist sehr umfangreich. Gut geeignet für ATTiny, ältere AV Controller wie ATMega16, aber kompliziert. Projektwechsel ist umständlich!
Die Oberfläche überfrachtet und für Anfänger verwirrend usw.
Der USBasp Programmer ist schwierig einzubinden usw.
Kostenlos.
Die Toolchain für GCC ist recht versteckt. Microchip will ihre eigenen PIC Controller verkaufen
C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain Darin \bin

Es fehlt leider der AVR Programmer
avrdue.exe
avrdude.conf
libusb0.dll
Warum?

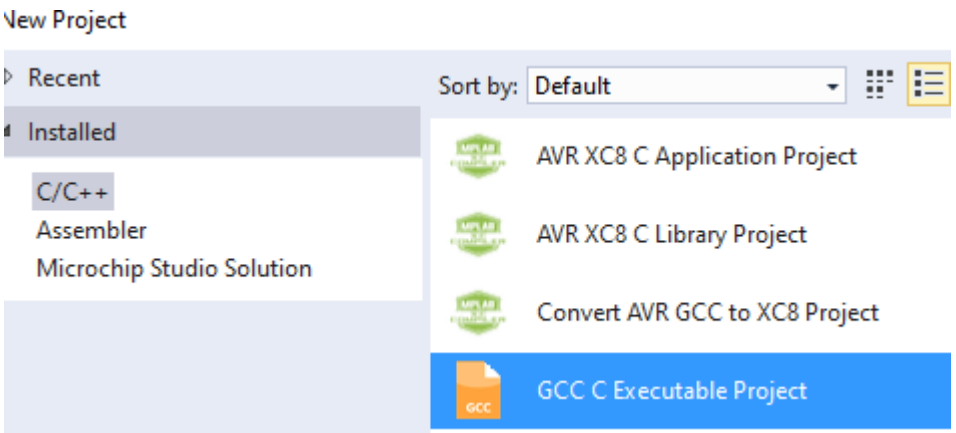


Ich empfehle diese Files in C:\AVRDUDE\.. zu kopieren

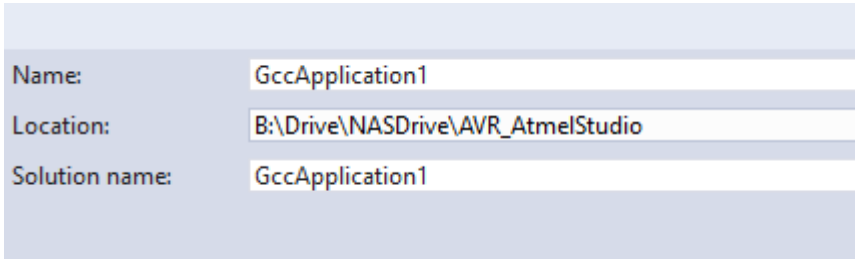
Dazu später mehr

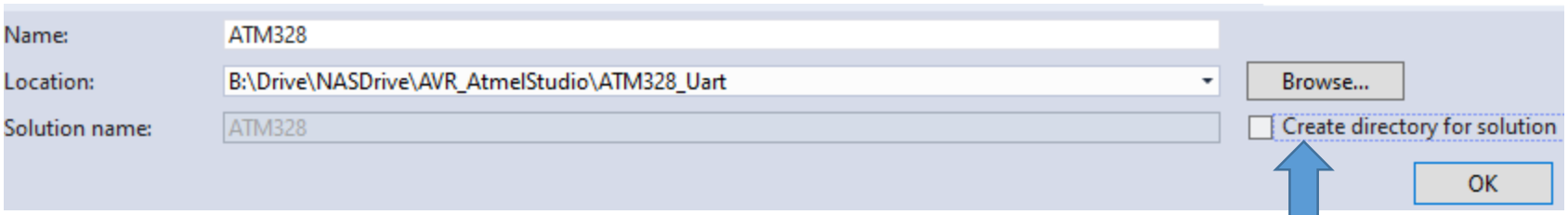
1. NEW Project

Dann sofort ändern.
auswählen



2. JETZT sofort
ändern z.B.
in

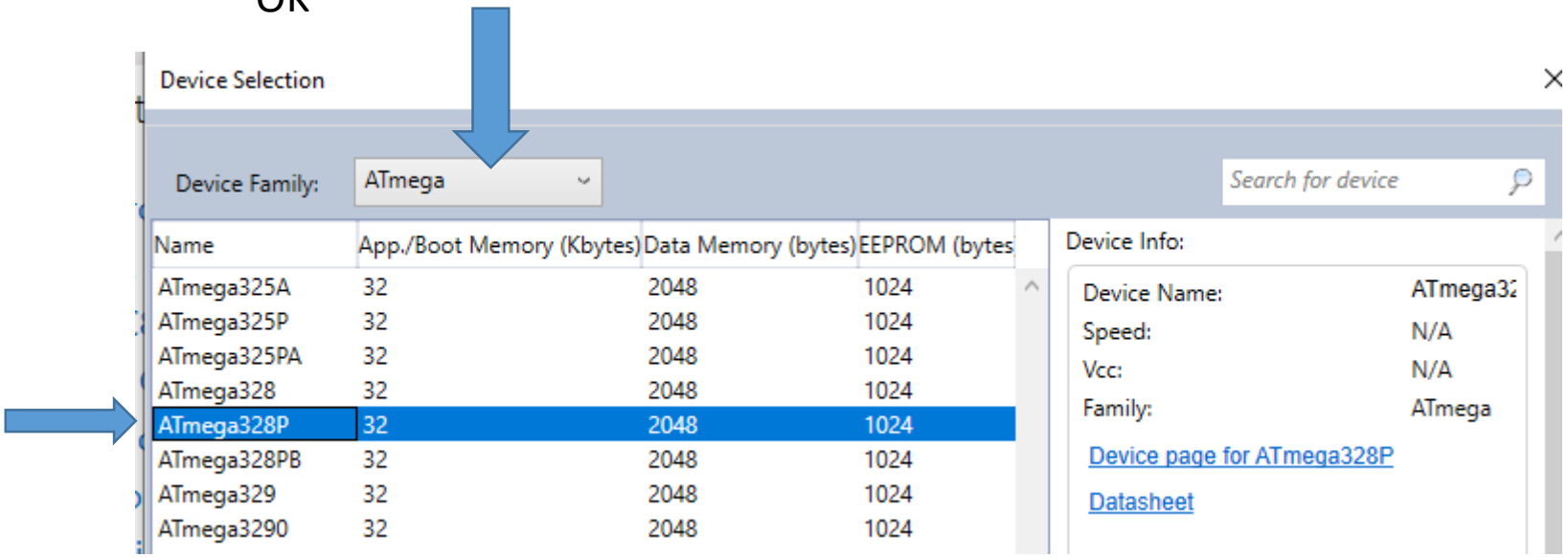




DAS
NICHT
ANKLICKEN

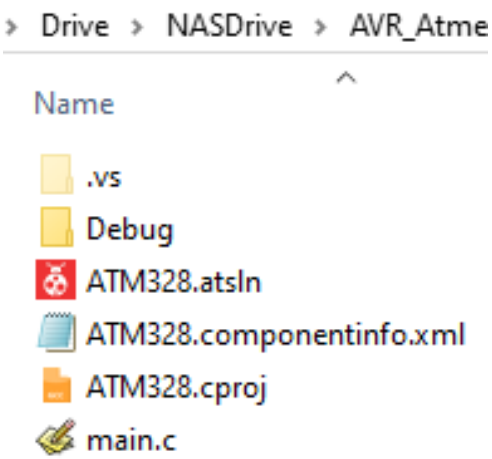
Name: ATM328_Main
Quelle Location:
B:\Drive\NASDrive\AVR_AtmelStudio\ATM328_LCD_UART
.....mit OK bestätigen.

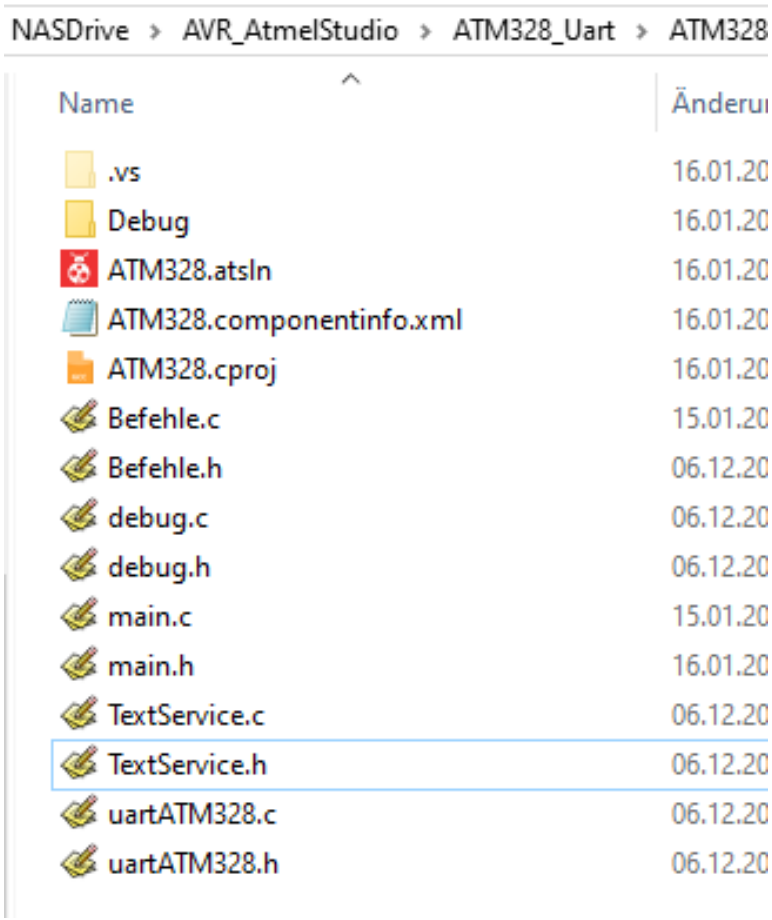
3. Prozessor wählen
ATMega328p
OK



4.
Dann: schließe Microchip Studio
Kopiere alle *.c *.h Files, die da sind oder die du brauchst in :
B:\Drive\NASDrive\AVR_AtmelStudio**\ATM328_LCD_UART\ATMega328**

Aus Fast leer wird →

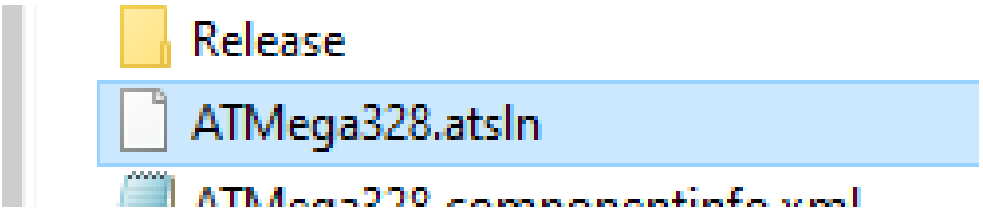


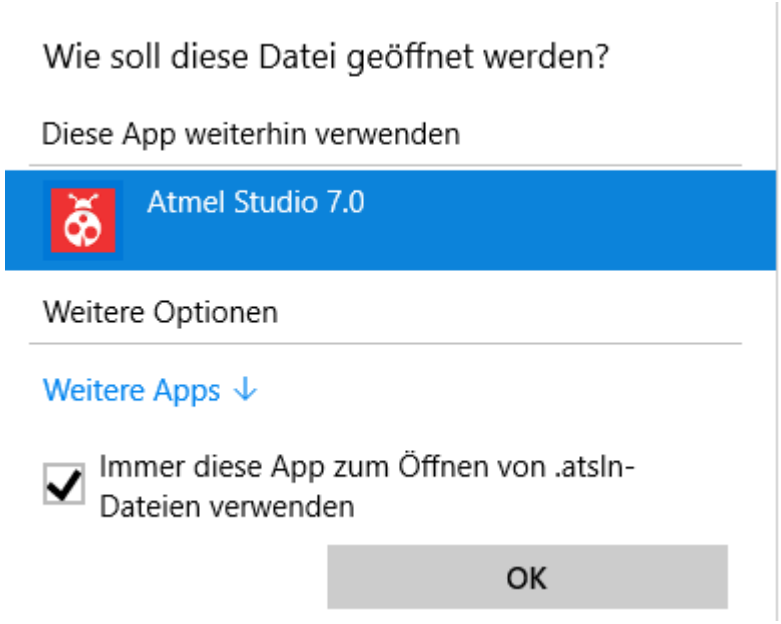


Öffne nach im Projektordner ….\AVR_AtmelStudio\ATM328_UART\ATMega328

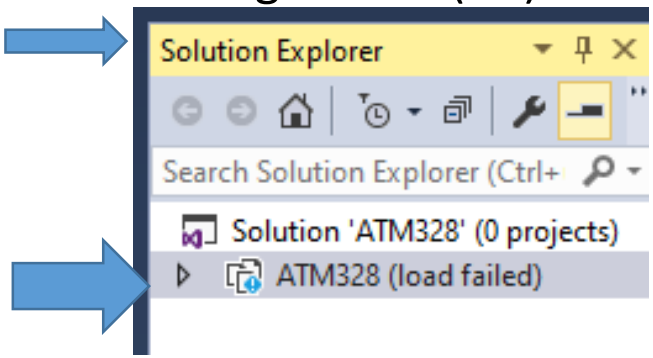
Die Datei ATMea328**.atsln**

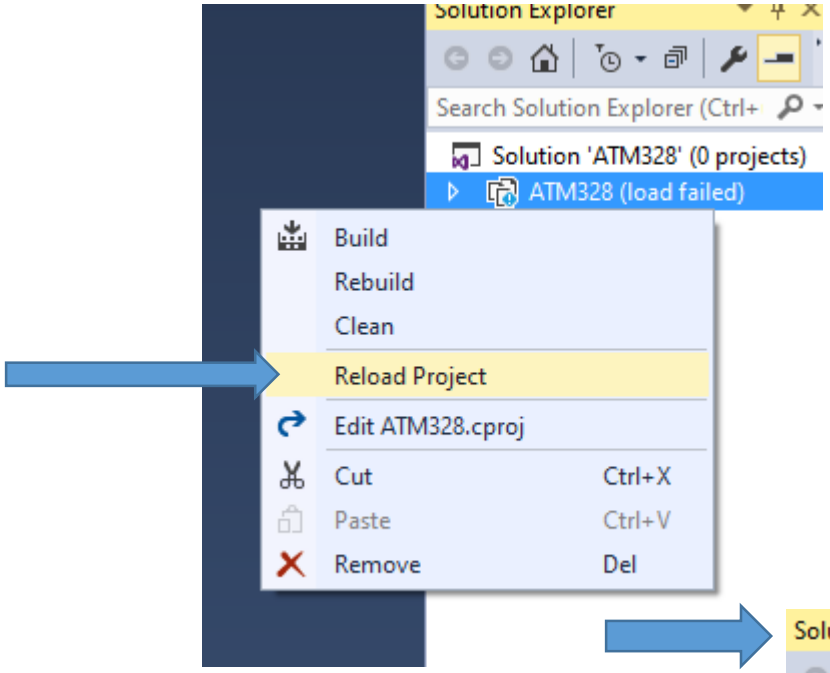
Sollte diese noch nicht mit Microchip Studio verknüpft sein, dann dies nachholen

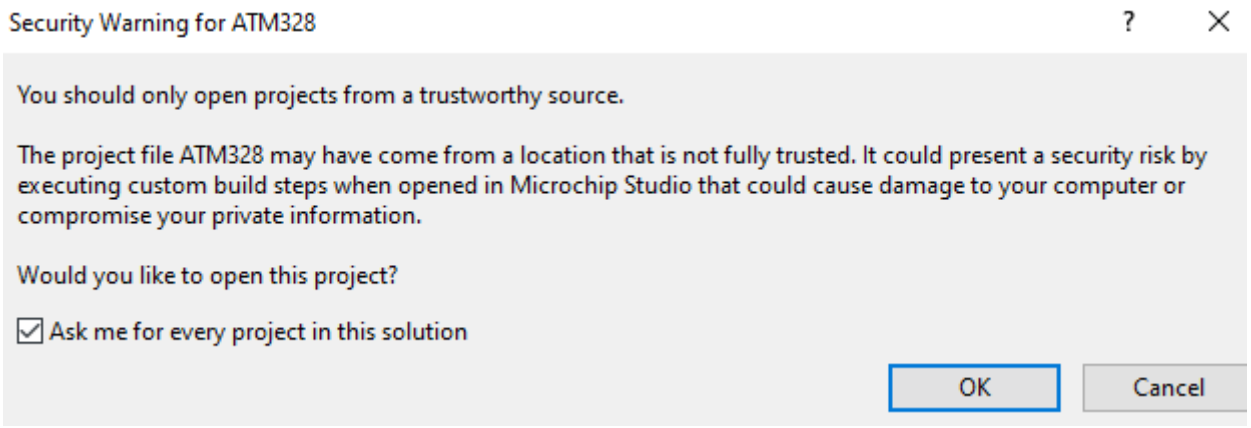


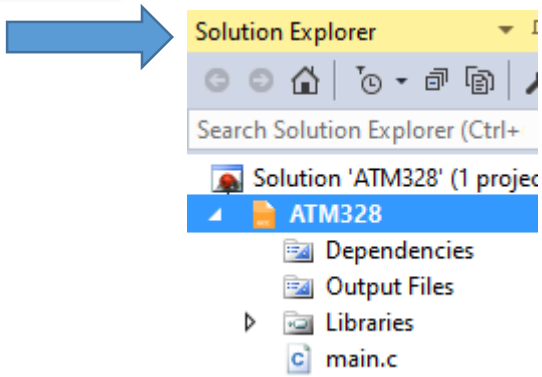


Sollte das auftauchen:
Right Click (RC)



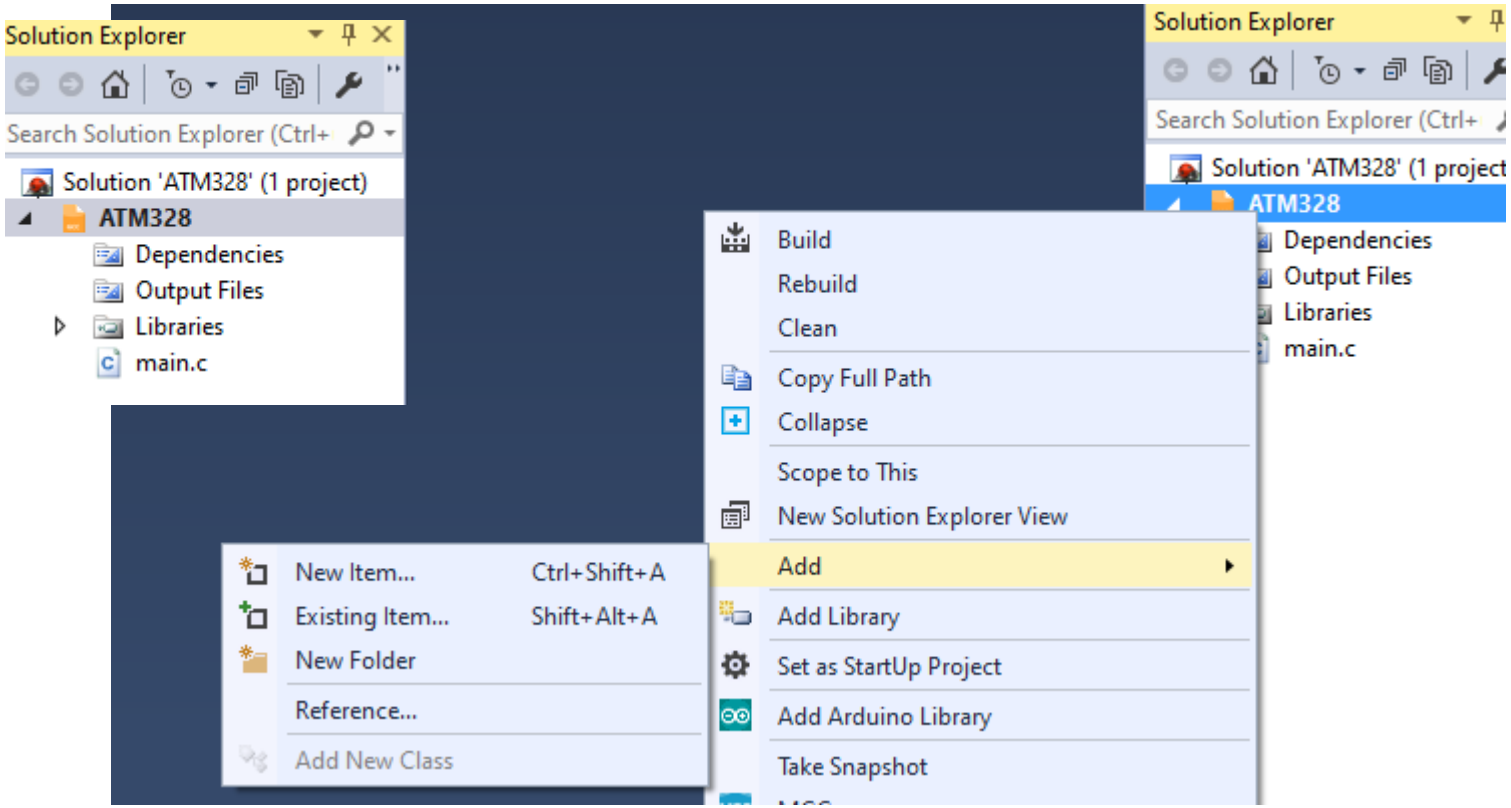






Aber wir sehen nur ein File Main.c

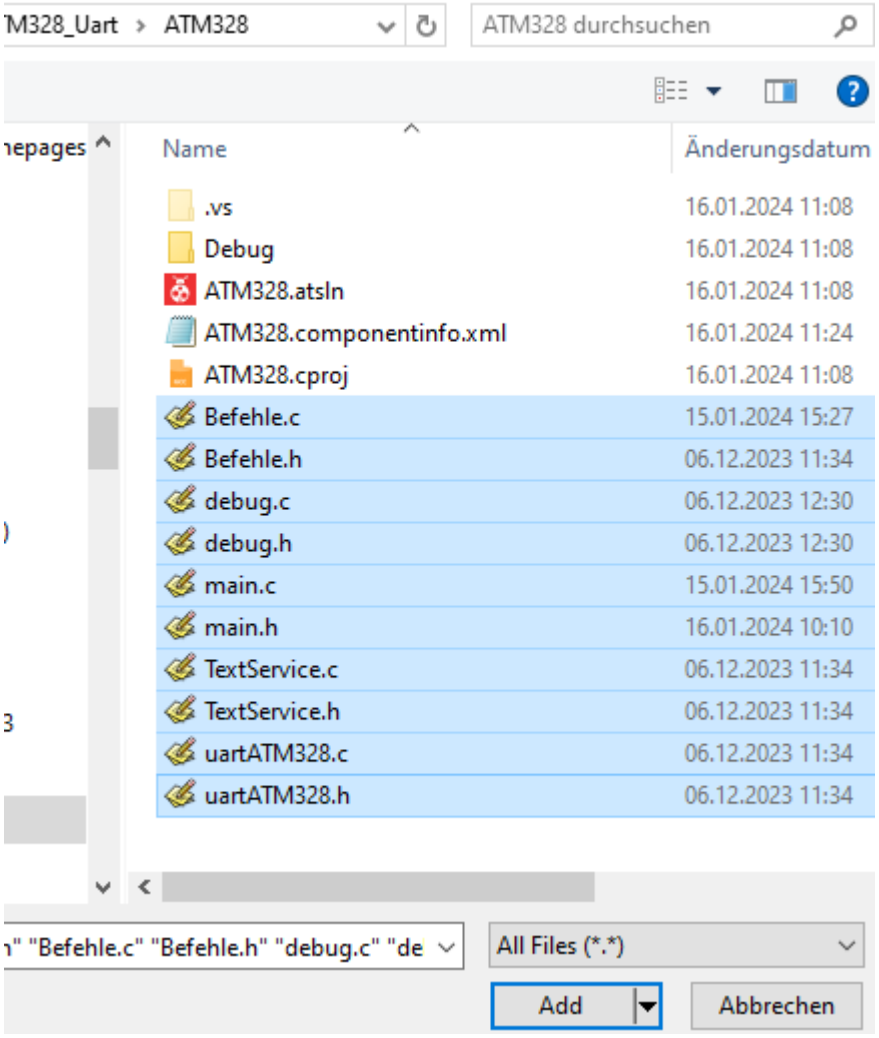
Einfügen der existierenden *.C *.h in den Solution Explorer
Rechte Mausklick auf ATMgea328
-→ nicht auf Solution „ATM328
ADD, existing Item



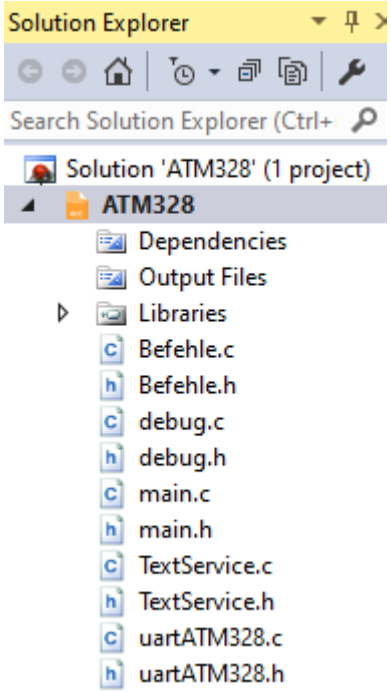
Alle Files:
main.h
UartATM3.h
Befehle.c
*.h
*.c
usw.

TIPP: alle File Markieren, zufügen

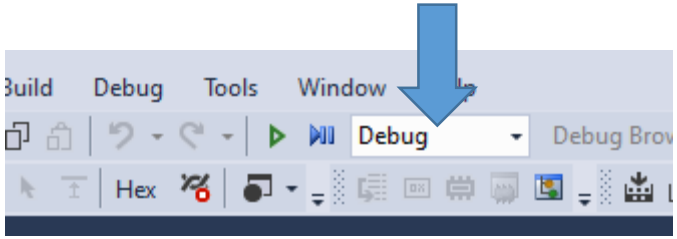
Alle *.C *.H gleichzeitig markieren. Dann „Add“

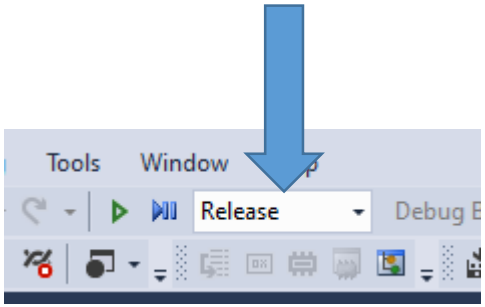


= >



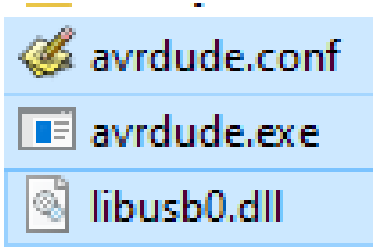
Stelle Compiler auf Release
Wir haben keinen Debugger





Es fehlt noch das Einbinden eines AVR Programmierprogrammes
Dieses heißt AVRDUDE.EXE

Es fehlt leider der AVR Programmer in der Toolchain
avrdue.exe
avrdude.conf
libusb0.dll
Warum?

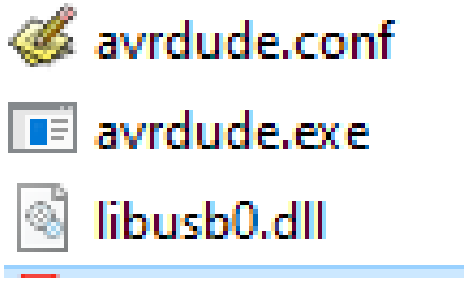


Ich empfehle diese Files in C:\AVRDUDE\.. zu kopieren

Einbinden von AVRDude in Microchip Studio – AVR Studio
Quelle: https://avr-programmieren-rh.de/tutorials/atmel-studio-avrdude/

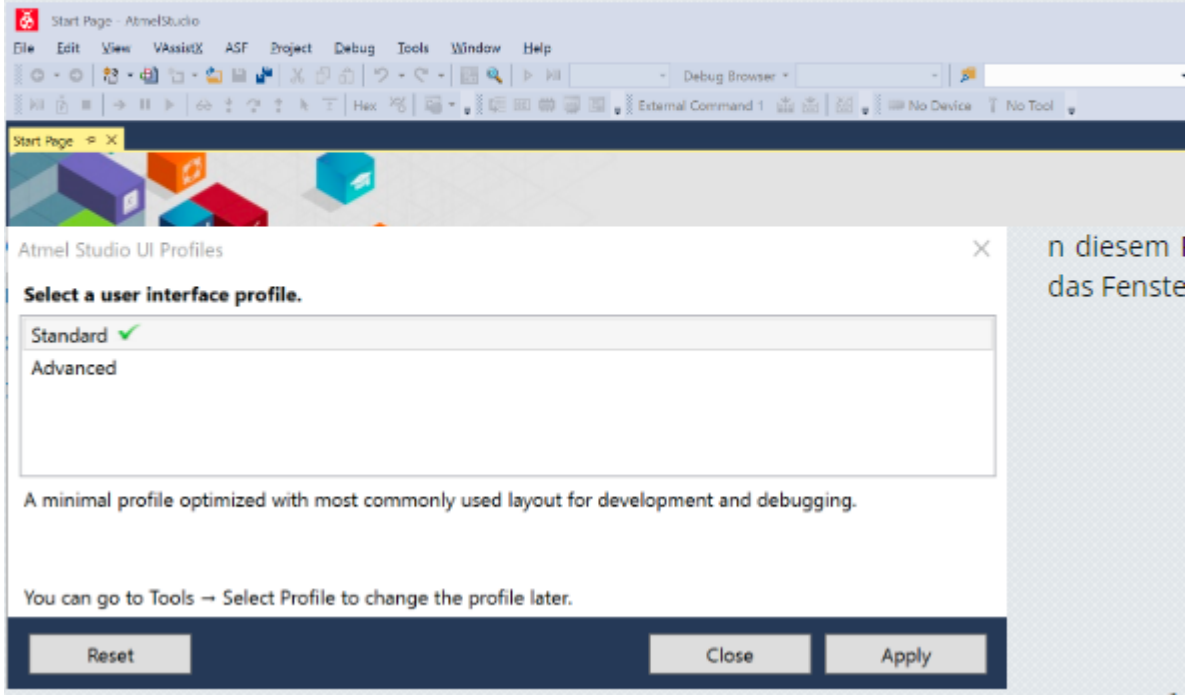
AVRDude.exe liegt z.B. auch in der Arduino DIE
C:\Program Files (x86)\Arduino\hardware\tools\avr\bin
oder C:\Program Files\Arduino IDE

In der modernen Arduino IDE ist kein AVR Programmer AVRDUDE.EXE

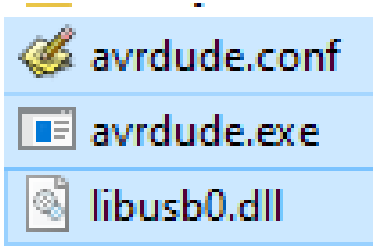


Jetzt öffnen wir das Programm "Atmel Studio".

Bevor wir mit Atmel Studio irgendwas machen müssen wir es noch auf Advanced umstelle
Farbe das Feld "Standard Mode" aus.

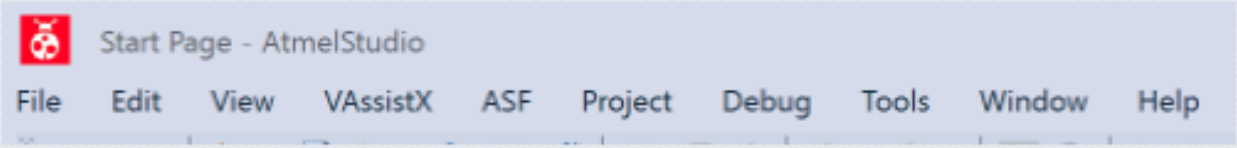


Zuerst empfehle ich diese Files in C:\AVRDUDE\.. zu kopieren

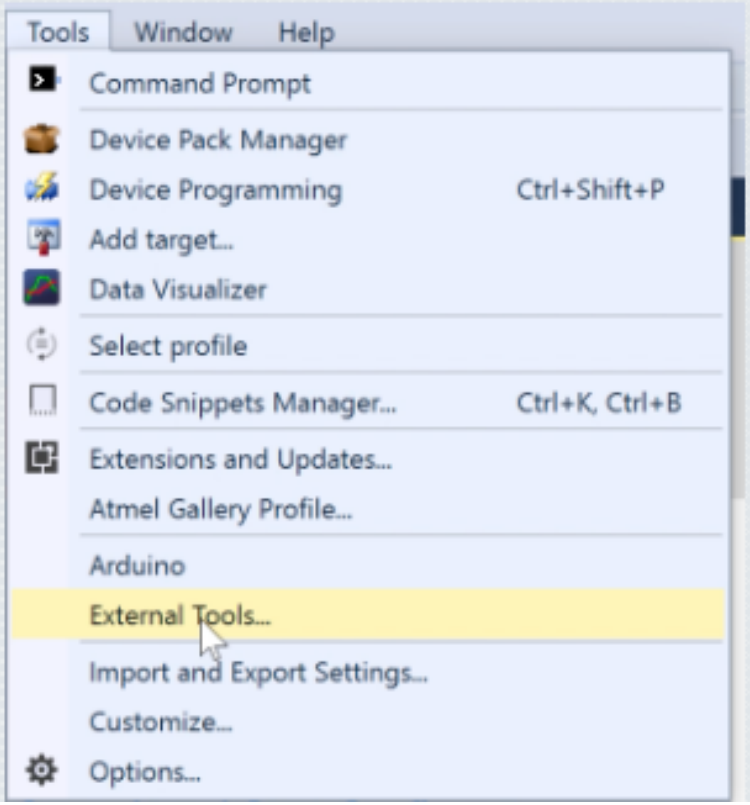


Jetzt können wir das tool "avrdude" in Atmel Studio einbinden.

In Atmel Studio wählen wir in der Toolbar den button "Tools" aus...

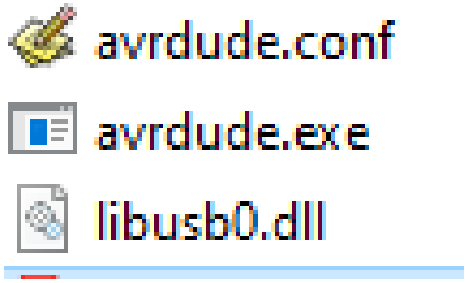


im Bereich "Tools" dann "External Tools..."



Stelle Compiler auf "Release"
Solution Explorer -> RC -> Properties -> Symbols -> Add --> "F_CPU=16000000UL"

Zielpfad für AVRDude Files
C:\AVRDUDE\avrdude.exe
C:\AVRDUDE\avrdude.conf



Kopiere AvrdudeFiles in C:\AVRDUDE\....

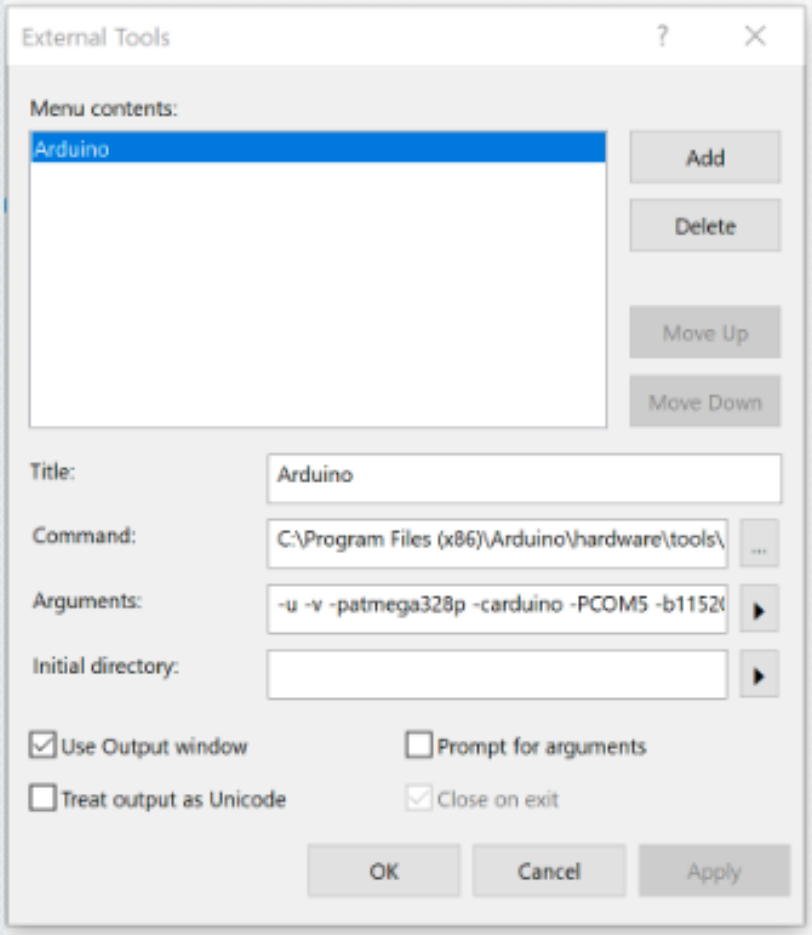
----------In External Tools einzutragen ----------
Title: „HexRelease_Arduino_Bootloader“ oder „HexReleaseToUSBasp
Command
oder Command **:** **C:\AVRDUDE\avrdude.exe**
ALLES IN EINE ZEILE:
Arguments: **-u -v -p atmega328p -c arduino -P COM8 -b115200 -U flash:w:"$(ProjectDir)Release\$(TargetName).hex":i -C "C:\AVRDUDE\avrdude.co**nf"
oder für den USBasp und andere Chips ohne Botloader:
Arguments: -**e -c USBasp -p ATtiny2313 -e -U flash:w:"$(ProjectDir)Release\$(ItemFileName).hex":a -v -v -B 100**
Initial direcory: **$(ProjectDir)**
Hacken bei "Use Output window"

C:\AVRDUDE>avrdude

```
Usage: avrdude [options]
Options:
  -p <partno>            Required. Specify AVR device.
  -b <baudrate>          Override RS-232 baud rate.
  -B <bitclock>          Specify JTAG/STK500v2 bit clock period (us).
  -C <config-file>       Specify location of configuration file.
  -c <programmer>        Specify programmer type.
  -D                     Disable auto erase for flash memory
  -i <delay>             ISP Clock Delay [in microseconds]
  -P <port>              Specify connection port.
  -F                     Override invalid signature check.
  -e                     Perform a chip erase.
```

```
 -O                      Perform RC oscillator calibration (see AVR053).
 -U <memtype>:r|w|v:<filename>[:format]
                         Memory operation specification.
                         Multiple -U options are allowed, each request
                         is performed in the order specified.
 -n                       Do not write anything to the device.
 -V                       Do not verify.
 -u                       Disable safemode, default when running from a script.
 -s                       Silent safemode operation, will not ask you if
                         fuses should be changed back.
 -t                       Enter terminal mode.
 -E <exitspec>[,<exitspec>] List programmer exit specifications.
 -x <extended_param>       Pass <extended_param> to programmer.
 -y                       Count # erase cycles in EEPROM.
 -Y <number>               Initialize erase cycle # in EEPROM.
 -v                       Verbose output. -v -v for more.
 -q                       Quell progress output. -q -q for less.
 -?                       Display this usage.

avrdude version 5.11.1, URL: <http://savannah.nongnu.org/projects/avrdude/>
```

Jetzt öffnet sich das Fenster "External Tools..."



Title:
hier wird der Name des Tool's eingeben. Bei mir heist das Tool Arduino.

Command:
hier wird der Pfad von unserem "avrdude" angeben.
(C:\Program Files (x86)\Arduino\hardware\tools\avr\bin)

Arguments:
hier geben wir dem tool "avrdude" Anweisungen bei der Übergabe des Hex-Files. Für uns ist nur interessant welchen Com-Port unser Arduino hat. Wenn wir den Port herausgefunden haben müssen wir diesen von

"-PCOM5" in z.B. -PCOM3 ändern.

(-u -v -patmega328p -carduino -***PCOM5*** -b115200 -Uflash:w:"$(ProjectDir)Debug\$(TargetName).hex":i -C"C:\Program Files (x86)\Arduino\hardware\tools\avr\etc\avrdude.conf")

Das was in der gelben Klammer steht kopieren und in "Arguments" einfügen.

Nochmals ALLES IN EINE ZEILE:

Wenn der USBasp einen SCK Fehler meldet, dann Firmware update oder in einem Parameter überschreiben –B 100 –i 10
→avrdude.exe: set SCK frequency to 8000 Hz
→avrdude.exe: warning: cannot set sck period. please check for usbasp firmware update.

Achtung Anführungszeichen muss OBEN sein “C:\AVRDUDE\avrdude.conf" **Dieses Powerpoint macht das falsch**

Arguments:
-p atmega328p -c arduino -P COM8 -b115200 -U flash:w:"$(ProjectDir)Release\$(TargetName).hex":i -i 10 -u -v -B 100 -C “C:\AVRDUDE\avrdude.conf"

oder für den USBasp und andere Chips ohne Botloader:
Arguments:

-p atmega328p -c arduino -P COM8 -b115200 -U flash:w:"$(ProjectDir)Release\$(TargetName).hex":a -u -v –B 100 –i 10 -C “C:\AVRDUDE\avrdude.conf"

-p ATtiny2313 -c USBasp -U flash:w:"$(ProjectDir)Release\$(ItemFileName).hex":a -u -v -B 100 -i 10 -C “C:\AVRDUDE\avrdude.conf"

"C:\AVRDUDE\avrdude.conf"

Initial direcory: **$(ProjectDir)**

Hacken setzen bei "Use Output window"

NOCHMALS:

Atmlel USBasp verwenden in Atmel Studio

Title: USBasp
Command: (eine Zeile)
C:\Program Files (x86)\Atmel\Studio\7.0\toolchain\avr8\avr8-gnu-toolchain\bin\avrdude.exe

Arguments: (eine Zeile)
-e -c USBasp -p **ATtiny2313** -e -U flash:w:"$(ProjectDir)**Debug**\$(ItemFileName).hex":a

Oder für Release
-e -c USBasp -p ATtiny2313 -e -U flash:w:"$(ProjectDir)**Release**\$(ItemFileName).hex":a

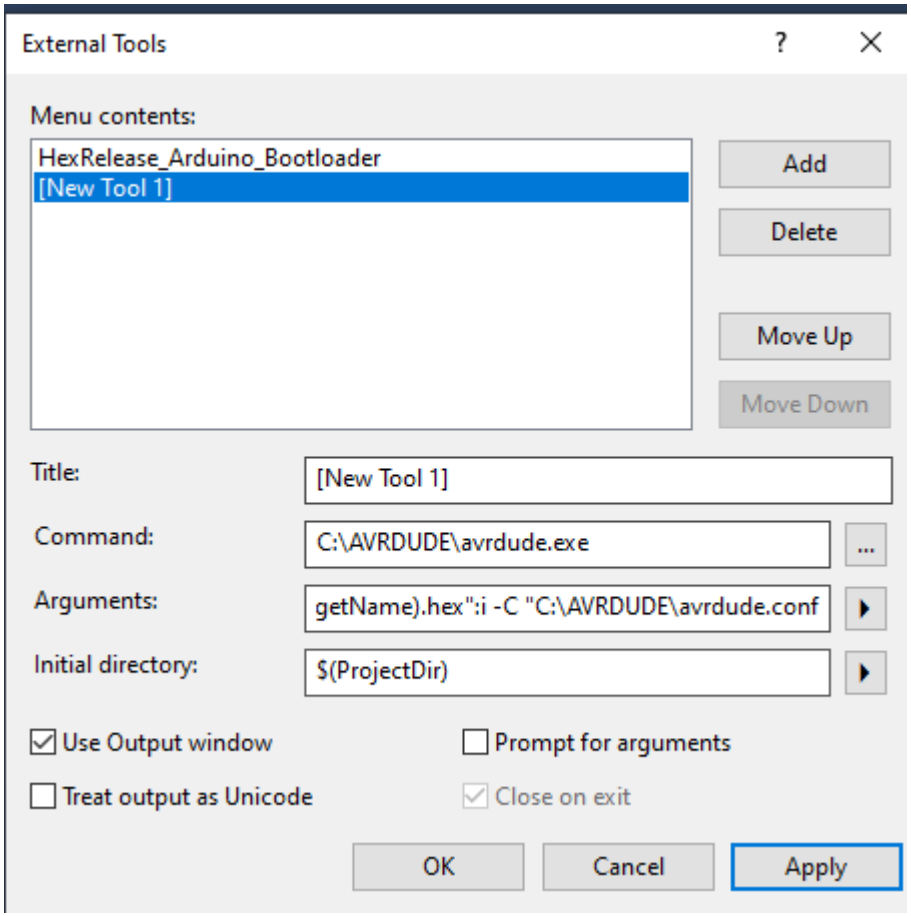
```
-p ATtiny2313 -c USBasp -U
flash:w:"$(ProjectDir)Release\$(TargetName).hex":a  -e -u -v -B 10 -C
“C:\AVRDUDE\avrdude.conf"
```

- Statt AtTiny2313 ATMega16, ATMega328p etc eintrageb
- Statt Debug Release Schreiben, je nach Compiler Options

-c usbasp -p t2313 -P usb
-c usbasp -p t2313 -P usb -B 8.0

So ist es richtig
-c usbasp -p t2313 -P usb -B 8.0 -U
flash:w:"B:\Drive\NASDrive\AVR_AtmelStudio\ATtiny2313(noA)_3xRelais_Aux_02.02.24\ATtiny2313(noA)_Main\Release\ATtiny2313(noA)_Main.hex**":a**

So soll es aussehen für den Arduino:



So kann es für den USBasp und anderen Chips wie ATMega16 oder ATTiny2313.

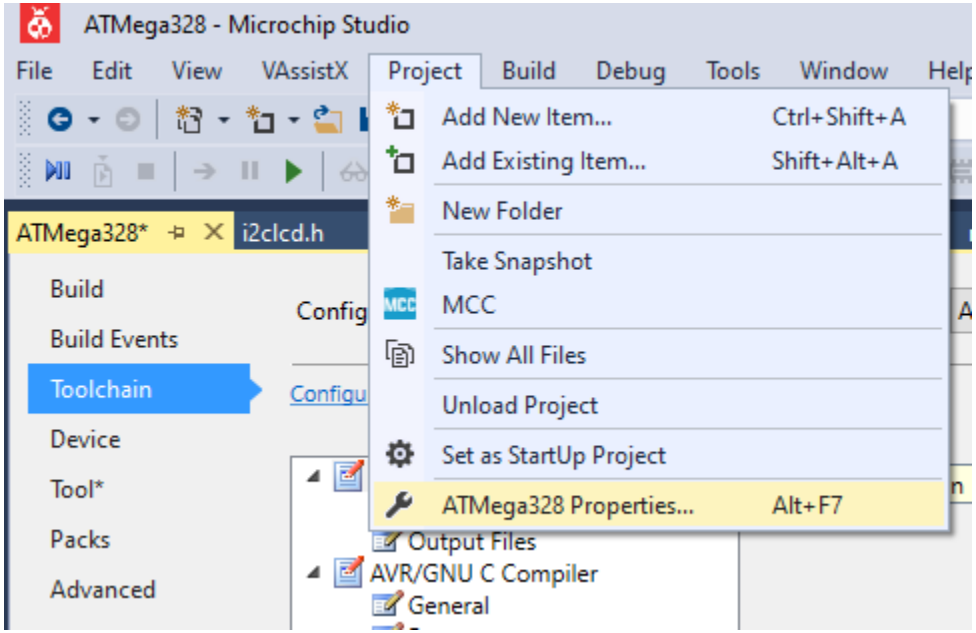
Arguments: -**e -c USBasp -p ATtiny2313 -e -U flash:w:"$(ProjectDir)Release\$(ItemFileName).hex":a -v -v -B 100**

Alles in eine Zeile
$(ProjectDir)

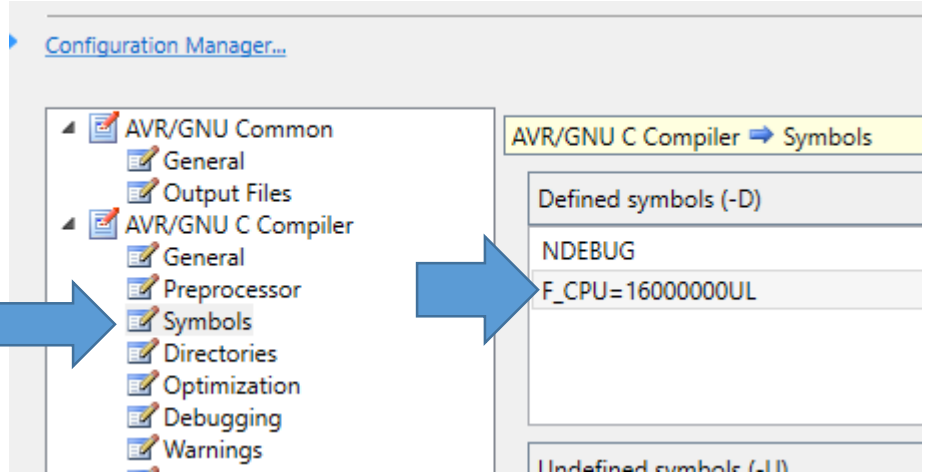
Quarzfrequenz : F_CPU Parameter einrichten

Project → ATMega328 Property → PLUS bei „Defined symbols“

WICHTIG



F_CPU=16000000UL



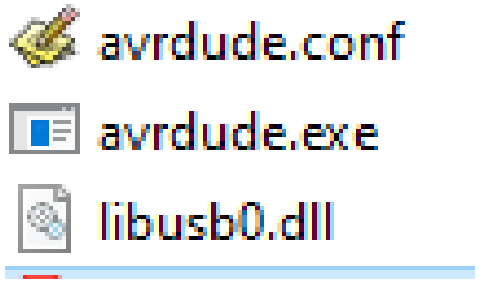
Zusammenfassung:
Eintragung für den Arduino UNO Bootloader

ArduinoToBootloader

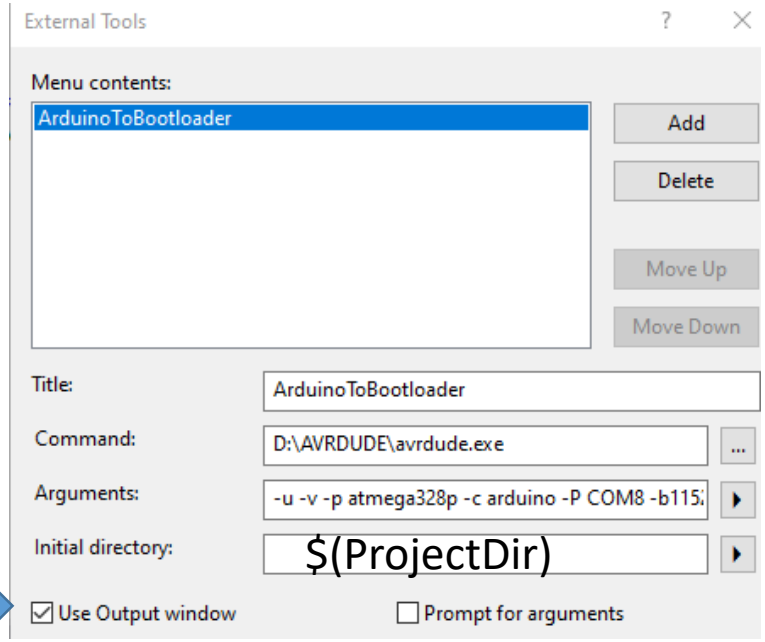
C:\AVRDUDE\avrdude

-u -v -p atmega328p -c arduino -P COM8 -b115200 -U flash:w:"$(ProjectDir)Release\$(TargetName).hex":i -C D:\AVRDUDE\avrdude.conf

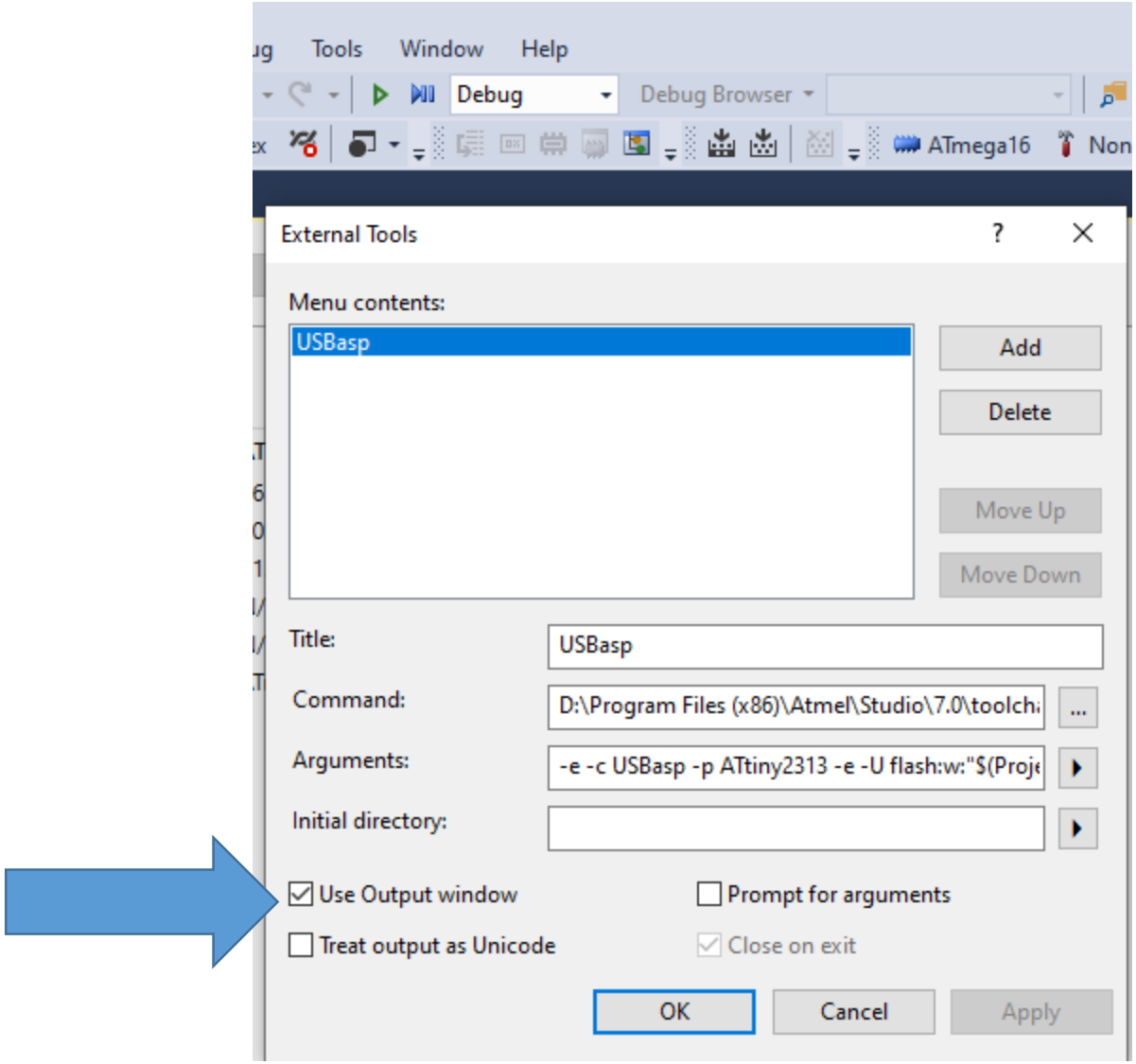
Lege in C: → C:\AVRDUDE\



$(ProjectDir)



Tools → External Tools → (Hacken bei Use Output Window)



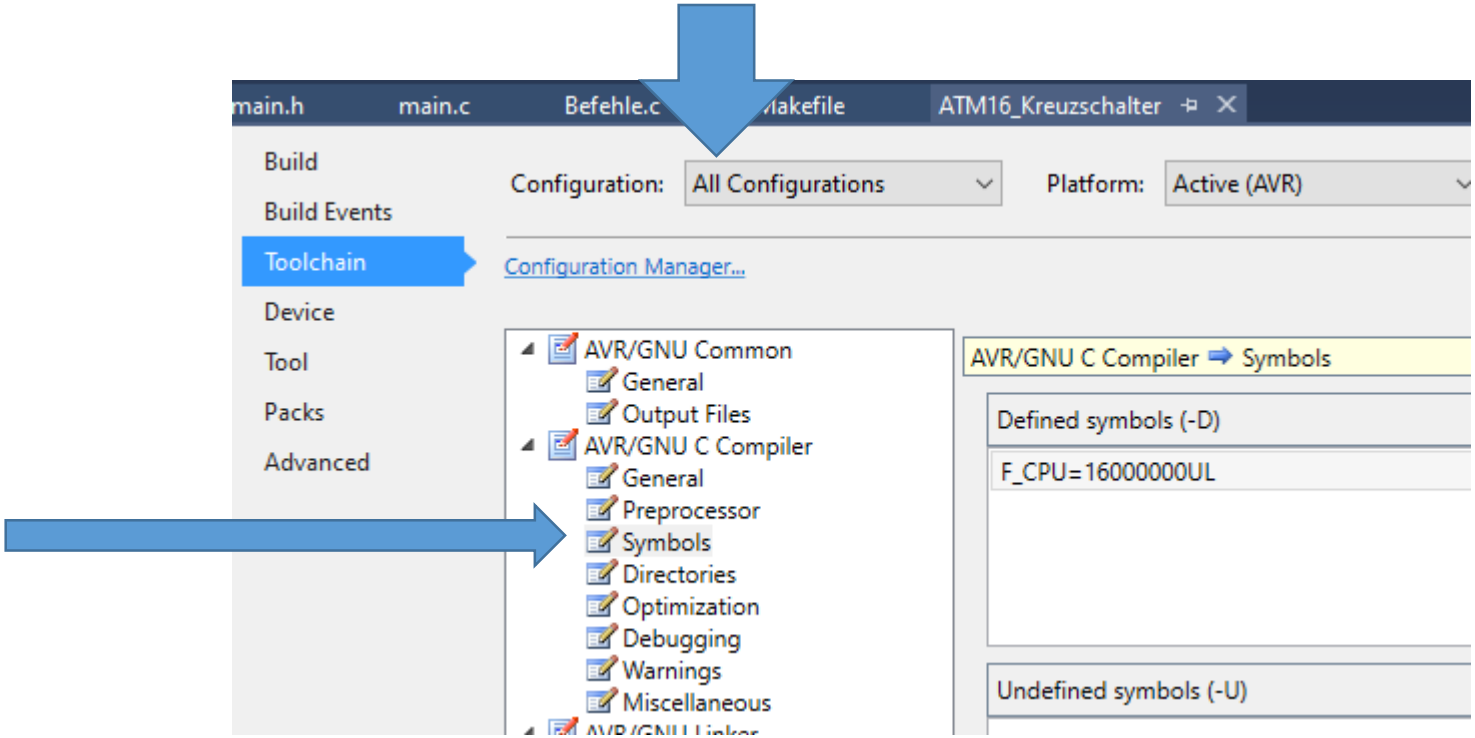
F_CPU zu Project Property zufügen

AVR/Gnu C Compiler → Symbols

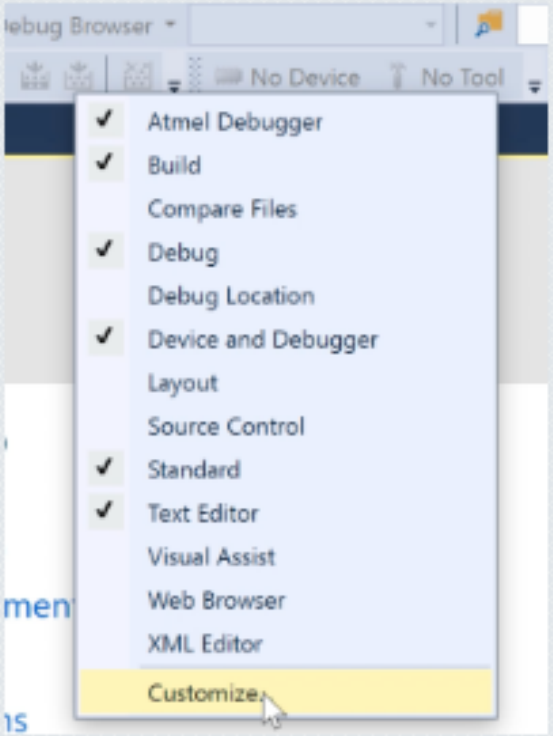
ALL Configurations

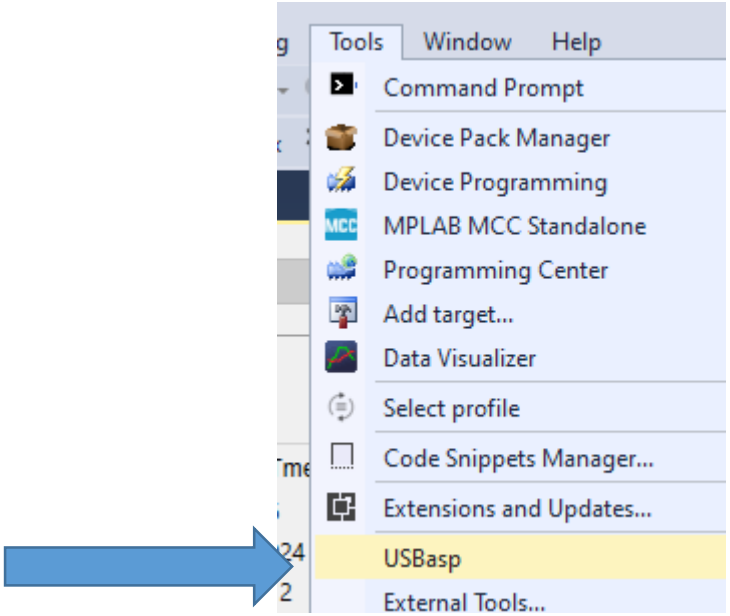
…Mit „Add +“ ein –D Symbol schreiben

F_CPU=16000000UL für eine 16MHz Quarz. UL= Unsigned Long



Nach dem wir alle Werte in "External Tools..." angepasst und eingetragen haben müssen wir nur noch die Toolbar anpassen damit wir mit einem Knopf den Arduino programmieren können.

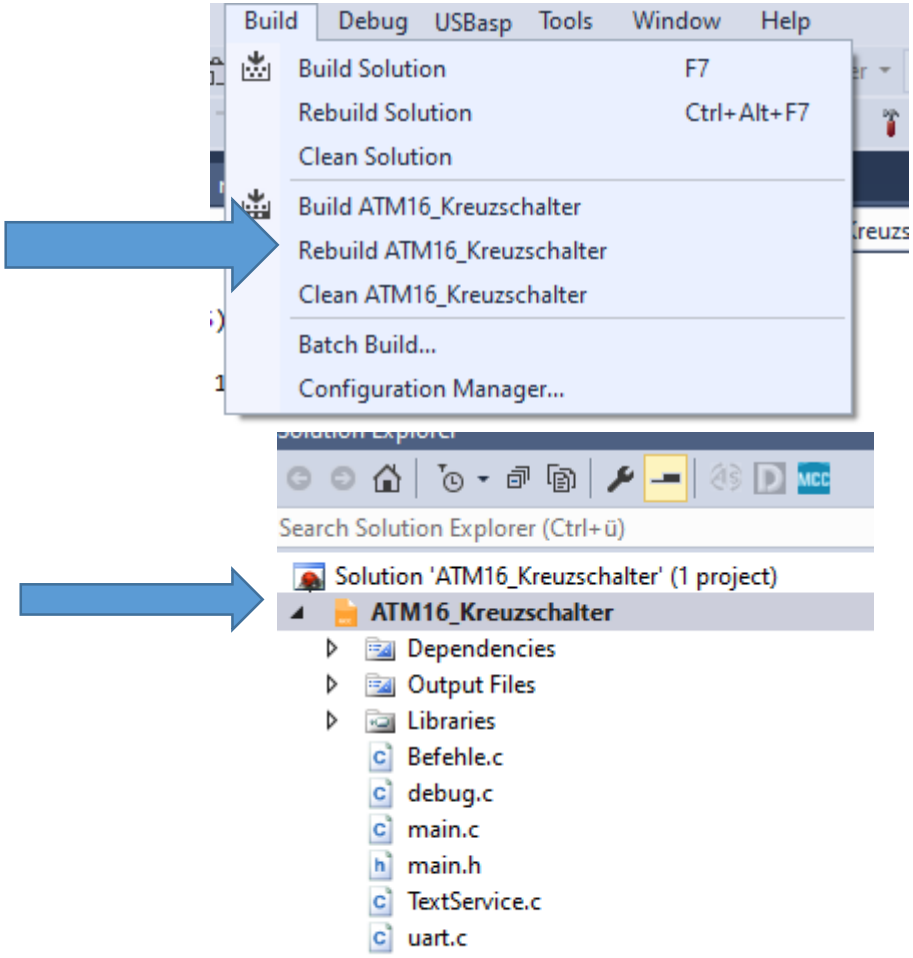




Weitere Attribute

#AVRDUDE -B 10 SCK period. Fehlerschutz

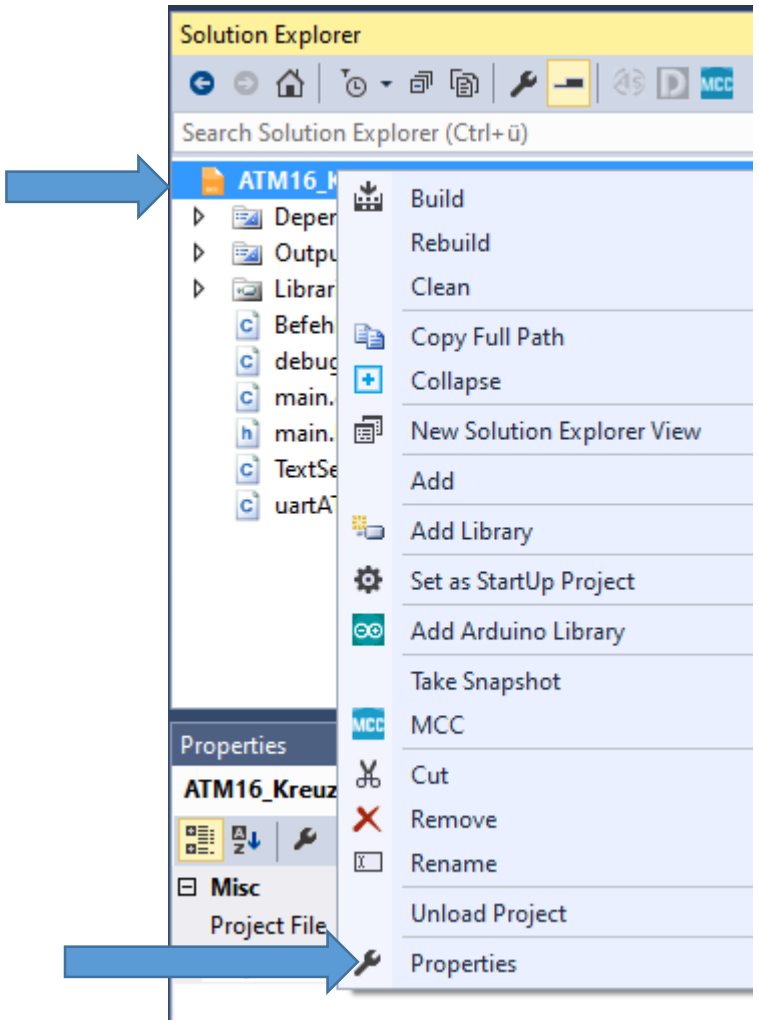
#AVRDUDE_VERBOSE = -v -v

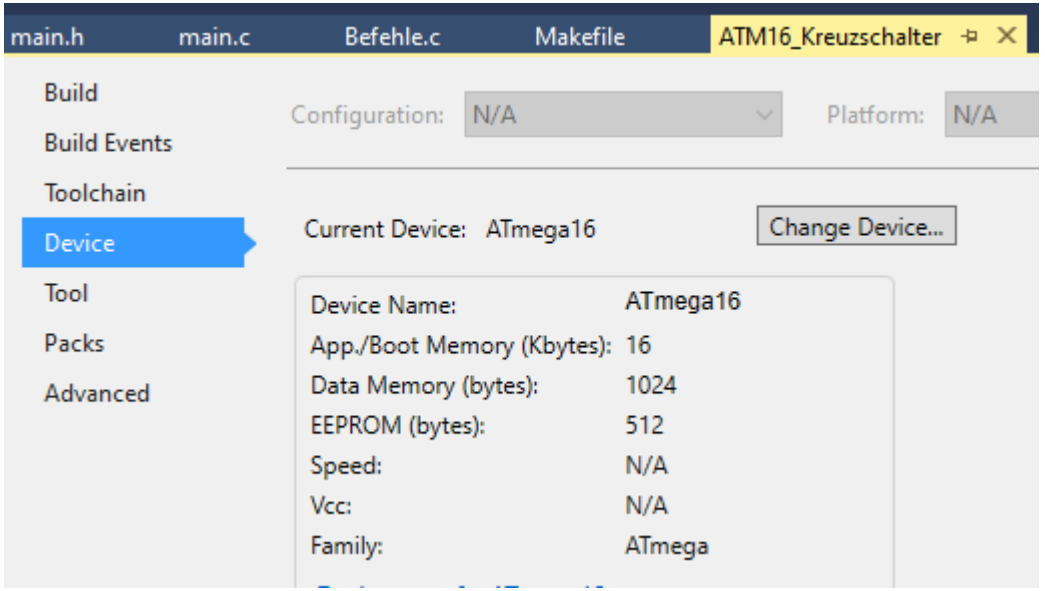


Nimm den Highlight Namen für das HEX File Sonst bekommt USBasp das falsche

Jetzt erst USBasp aufrufen

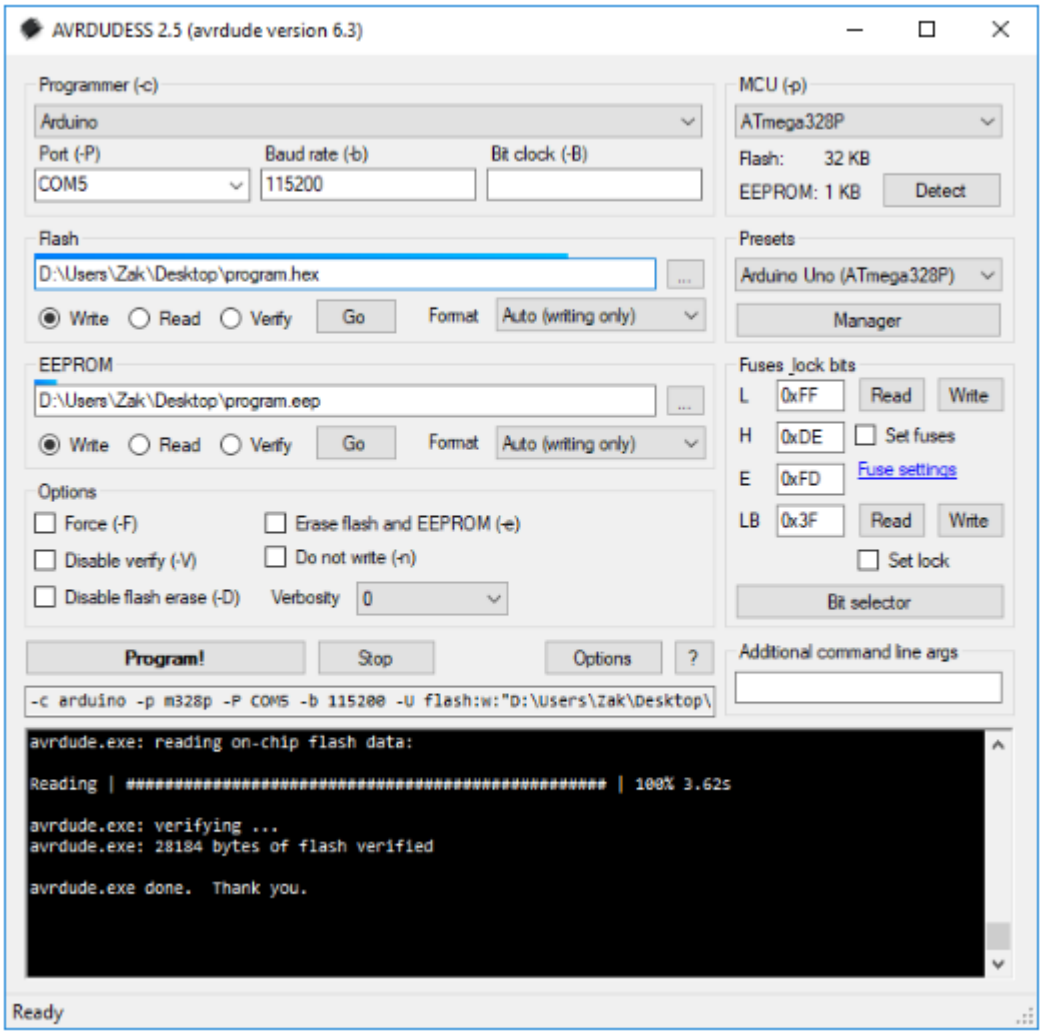
Change Device
Right Click auf Project





AVRDUDESS – A GUI for AVRDUDE

https://github.com/avrdudes/avrdude/releases/tag/v7.2



FUSE ATMega16 https://www.engbedded.com/fusecalc/

AVR part name: ATmega16 Select (141 parts currently listed)

Feature configuration

This allows easy configuration of your AVR device. All changes will be applied instantly.

Features

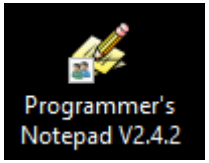
- Ext. Crystal/Resonator High Freq.; Start-up time: 16K CK + 64 ms; [CKSEL=1111 SUT=11]
- [x] **Brown-out detection enabled; [BODEN=0]**
- Brown-out detection level at VCC=2.7 V; [BODLEVEL=1]
- [ ] **Boot Reset vector Enabled (default address=$0000); [BOOTRST=0]**
- Boot Flash section size=1024 words Boot start address=$1C00; [BOOTSZ=00] ; default value
- [ ] **Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]**
- [x] **CKOPT fuse (operation dependent of CKSEL fuses); [CKOPT=0]**
- [x] **Serial program downloading (SPI) enabled; [SPIEN=0]**
- [ ] **JTAG Interface Enabled; [JTAGEN=0]**
- [ ] **On-Chip Debug Enabled; [OCDEN=0]**

Low	*High*	*Action*	*AVRDUDE arguments*
0x BF	0x C9	Apply values Defaults Apply manual changes to the values on the left side, or load factory default values for the selected device.	`-U lfuse:w:0xbf:m -U hfuse:w:0xc9:m` Select (try triple-click) and copy-and-paste this option string into your avrdude command line. You may specify multiple -U arguments within one call of avrdude.

Der bessere Programmiereditor

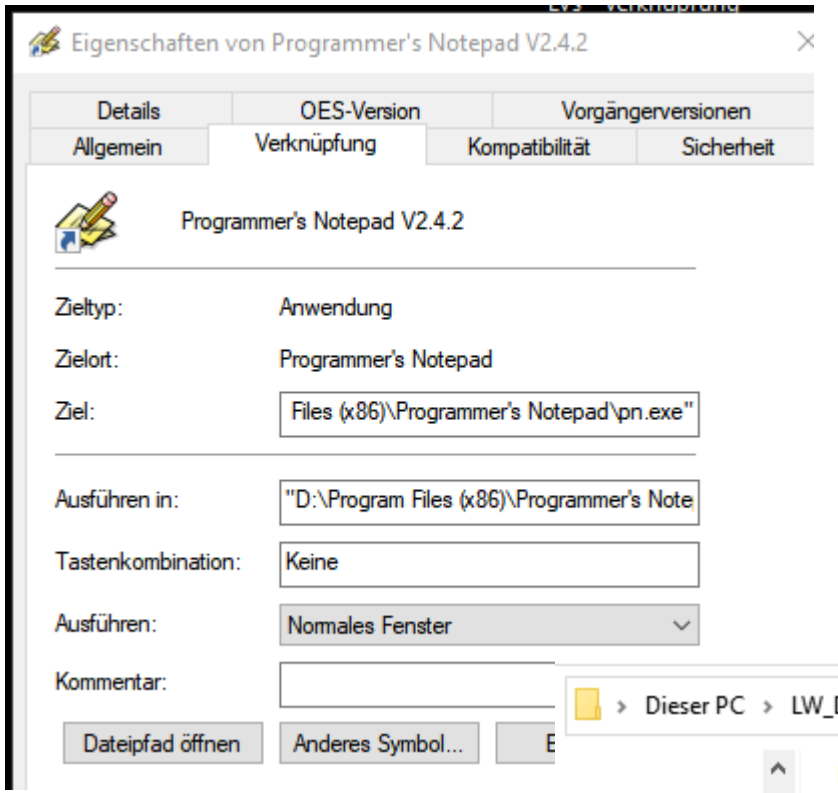
Programmers Notepad

Download mit: https://www.pnotepad.org/

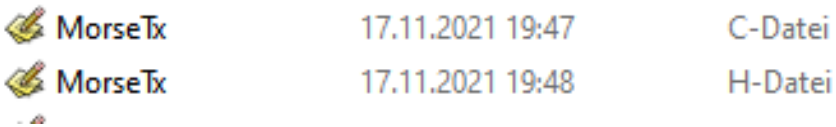


Installieren, dabei Symbol auf Desktop aktivieren.
Dann Rechte Maustaste: Eigenschaften, Installationspfad merken/in eine Merktettel.txt speichern.

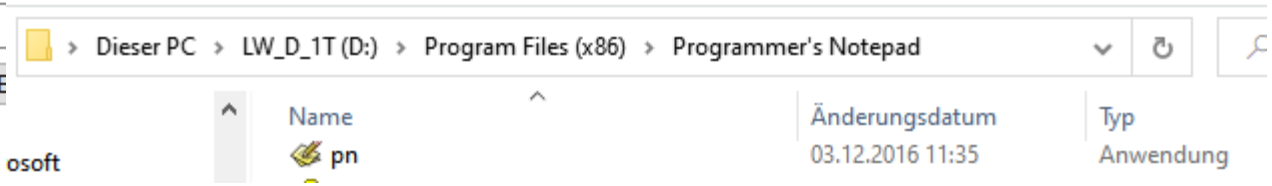
„C:\Program Files (x86)\Programmer's Notepad"



(später) auf eine xyz.c und xyz.h Datei gehen

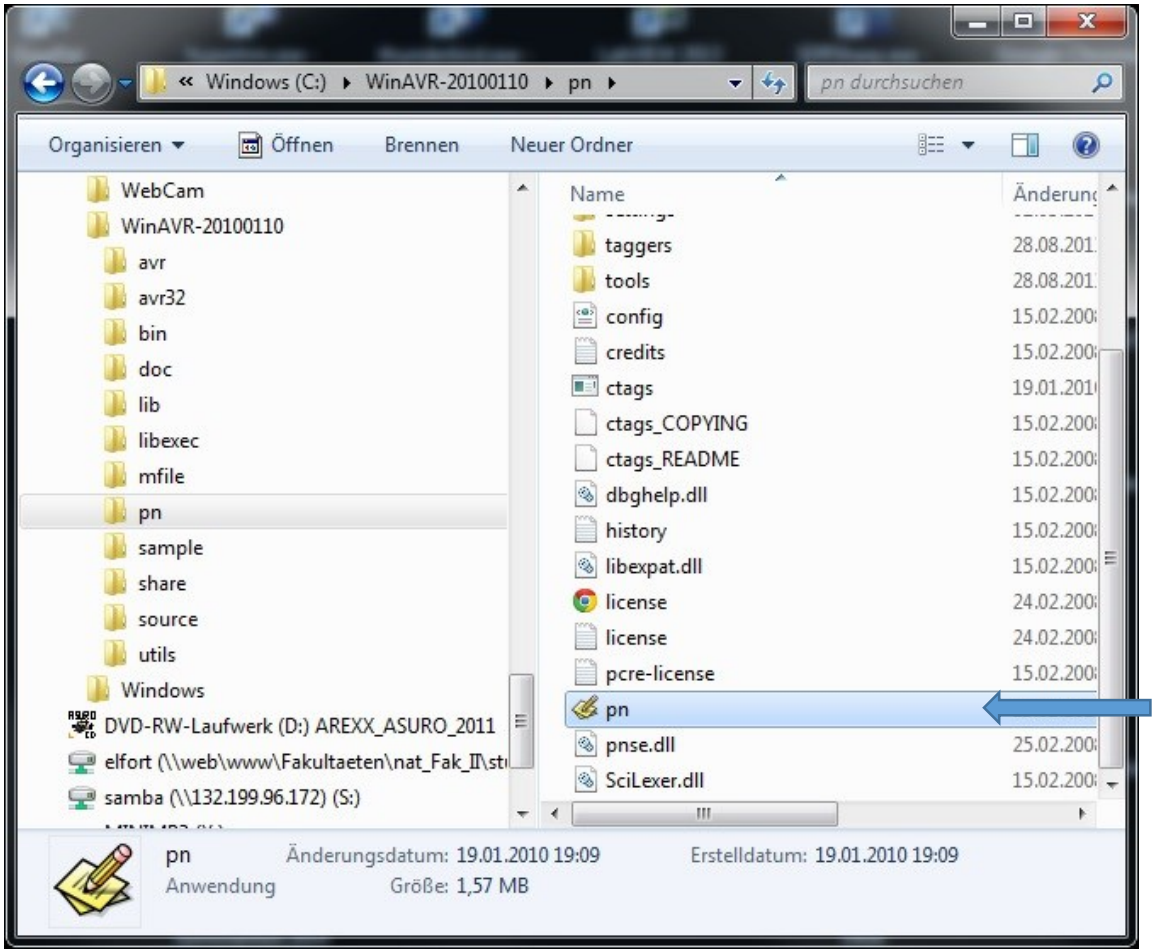


Right Click auf Datei: → Öffnen mit → Andere Ap →„Immer Öffnen mit“ anklicken..
,Dann auf besagten Ordner zuweisen → Dann auf dem PN Ordner PN.exe auswählen.



Der Programm-Editor “**Programmers Notepad**“ von **WINAVR** liegt auf **C:\WinAVR-20100110\pn\pn.exe**

Link erstellen: Datei Explorer öffnen: Das Programm **PN.EXE** (hier mit blauen Balken) mit **rechter Maustaste** anklicken. Senden an „Desktop“ .. Drücken



Dieser Link startet in Zukunft den „**P**rogrammers **N**otepad“ auf einfache Weise vom Desktop aus.

Mit diesem einfach zu bedienenden Editor können so gut wie alle Programmierarbeiten durchgeführt werden.
Mehr ist nicht notwendig.

So sieht der GNU-C für den **WINAVR** Compiler aus…

Programmer's Notepad

File Edit View Tools Window Help

C / C++ | PM_RELAIS_ON | Find

main.c

```c
/*****************************************************************************
*
* File Name:    main.c
* Project  :    SelfTest + Demo
*
* Description:
*
* Ver.      Date          Author            Comments
* -------   ----------    --------------    -----------------------------
* 1.00      14.08.2003    Jan Grewe         build
*
* Copyright (c) 2003 DLR Robotics & Mechatronics
*****************************************************************************/
/*****************************************************************************
 *
 *   This program is free software; you can redistribute
 *   it under the terms of the GNU General Public Licens
 *   the Free Software Foundation; either version 2 of t
 *   any later version.
 *****************************************************************************
#include "asuro.h"
#include "SelfTest.h"
#include "Demo.h"

int main (void)
{
    unsigned int  i,data[2];
    unsigned char count = 0,flag = FALSE;
    Init();
    for (i = 0; i < 0x1FFF; i++) {
        OdometrieData(data);
        if ((data[0] <  200) && (flag == TRUE)) {coun
        if (data[0] >= 600) flag = TRUE;
    }
```

Projects

- New Project Group
  - MayMainProjekt
    - SelfTest
      - Demo.c
      - Demo.d
      - Demo.h
      - Demo.lst
      - Demo.o
      - IRDemo.c
      - IRDemo.d
      - IRDemo.h
      - IRDemo.lst
      - IRDemo.o
      - LineDemo.c
      - LineDemo.d
      - LineDemo.h
      - LineDemo.lst

[1:1] : 39 | ANSI | CR+LF | INS | Project selected.

Ich schätze WINAVR wegen seiner Einfachheit..
Der „**Programmer's Notepad**“ ist gerade wegen seiner Übersichtlichkeit ideal.

Ein **Projects** Fenster erlaubt schnellen Zugriff zu allen *.c *.h Modulen.

Im Menü **Tools** ist Compilieren mit „**make**“ und das Chip-Flashen (Programm aufladen) mit „**avrdude**“.
Fertig.

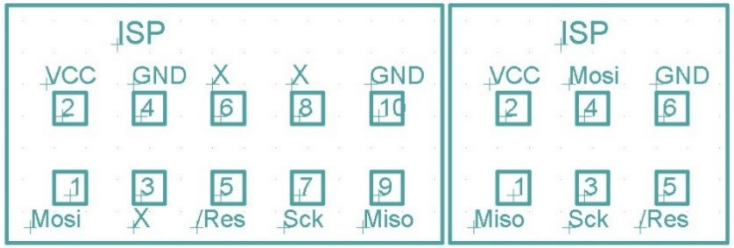
Workflow Turnaround

1. Schritt: Programmieren
2. Schritt: Kompilieren
3. Schritt: Auf den µC Laden (**flashen** genannt)
4. Schritt: Ausprobieren:
5. Schritt: Fehler suchen, beseitigen →

Das nennt sich **Workflow oder Turnaround**

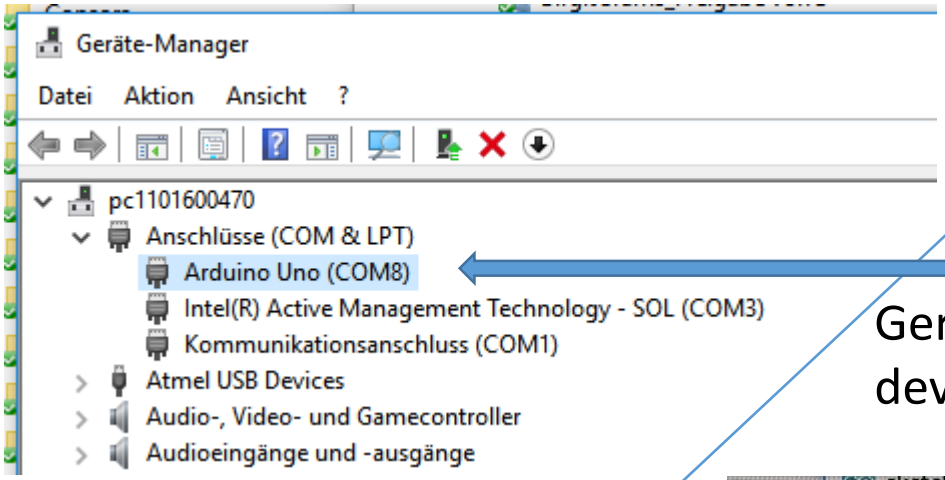
Bevor wir programmieren können, sollten wir wissen, wie wir das Programm in den Mikroprozessor bringen.
Diese Methoden sind so verschieden wie ihre Fähigkeiten und Eleganz

Bootloader,
ISP → „In System Programmer“, One-Wire-Debug,
JTAG → „Joint Test Action Group“ Debugschnittstelle
Debugwire → One-Wire Schnittstelle, für µC ohne JTAG. Bidirektional
(**/R**eset ohne Kondensator und für AVR Dragon) (hat nicht jeder µC



Standard
ISP= In-System-Programmer
Stecker: 6polig oder 10polig

Arduino UNO meldet sich als COM8 im Gerätemanager an

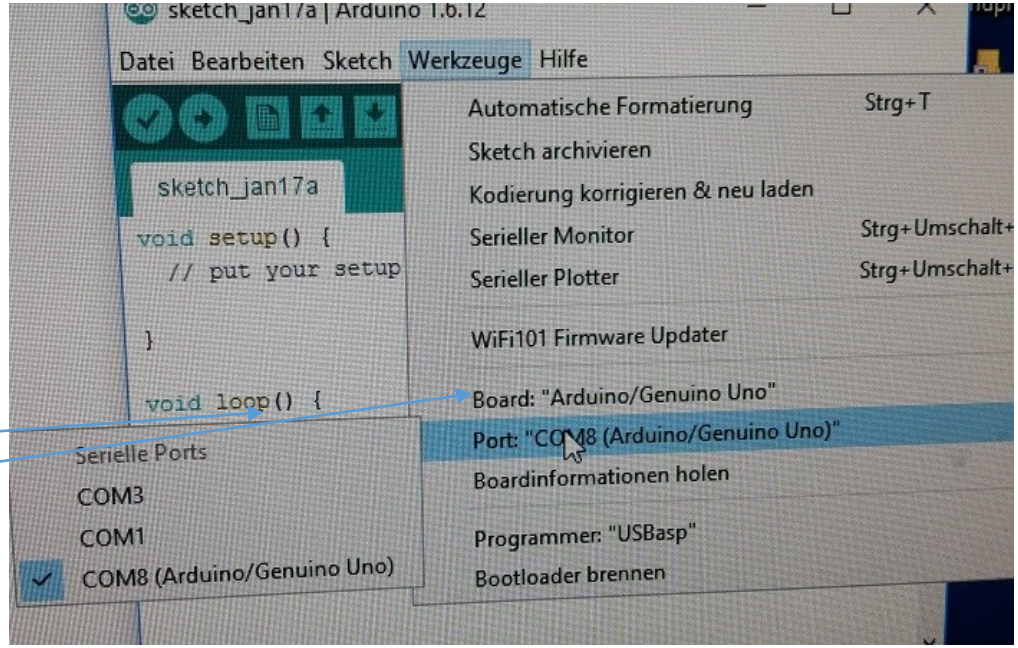


Die Arduino COM stellen wir auf „**COM8**“
Um eine gemeinsame Basis zu haben.

Gerätemanager Aufruf mit:
devmgmt.msc

Evtl muss erst ein Treiber installiert werden:
Suche nach **FTDI232**
C:\Program Files (x86)
\Arduino\drivers

Das Arduino SKETCH sollte unter Werkzeuge an Board: ***Arduino Uno*** und Port: auf **COM8** angepasst werden.



Zwischenspiel: Arduino Blinki

Blinki_Led_PWM_PB5_P13.ino

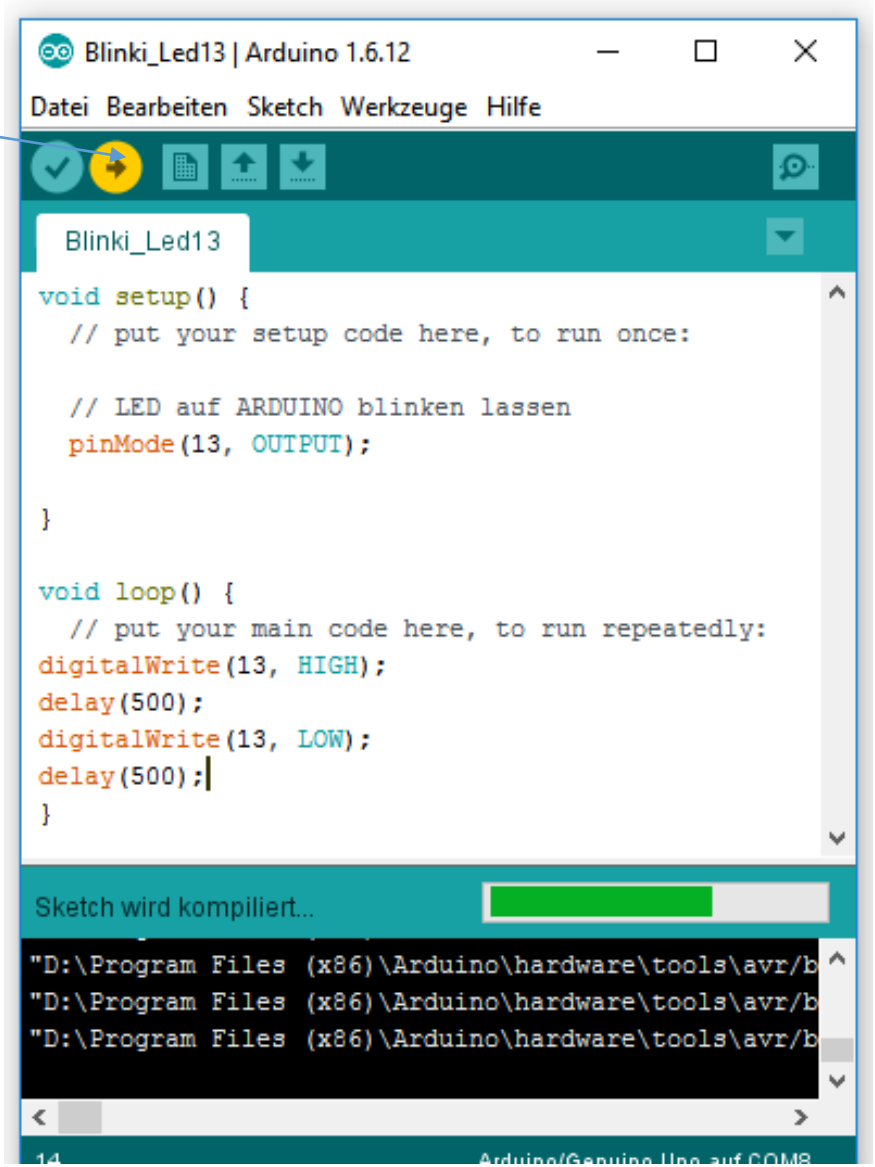
Gelbes Blinklicht

Kompilierern und laden

Die Zahl **13** ist von den Arduino Machern als **Pin** definiert worden, woran das Signal **SCK** sowie auch, hart verdrahtet, direkt abhängig die **gelbe LED** liegt.

Daher:
digitalWrite (13, HIGH);

Bemerkung:
Das klingt einfach,
ist jedoch **nicht so richtig gutes „C“**. Es verschleiert die interne µC Struktur.



Tatsächlich liegt die LED an einem PORT des µControllers namens „**B**“ an dem Kontakt „**5**“.

Also genauer
PORTB
PB5

Sketch repräsentiert somit nicht die Architektur des µC !

Hier wird **reines ANSI-C** verwendet.
Hier tauchen schon viele Methoden in „C“ auf.

1. **#include** Bibliothek

2. **#define** Aliase
3. #define **Bitmaske**

4. Bit-**Shift** einer 1 nach links, mit **(1<<n)**

5. **Setzen** eines Bits mit „ODER“
   → Ziel **|=** mask

6. **Löschen** eines Bits mit „UND-Komplement“
   → Ziel **&= ~**mask

Blinki_Led_PB5_P13-ANSI-C-Stile | Arduino 1.8.5

Datei Bearbeiten Sketch Werkzeuge Hilfe

Blinki_Led_PB5_P13-ANSI-C-Stile §

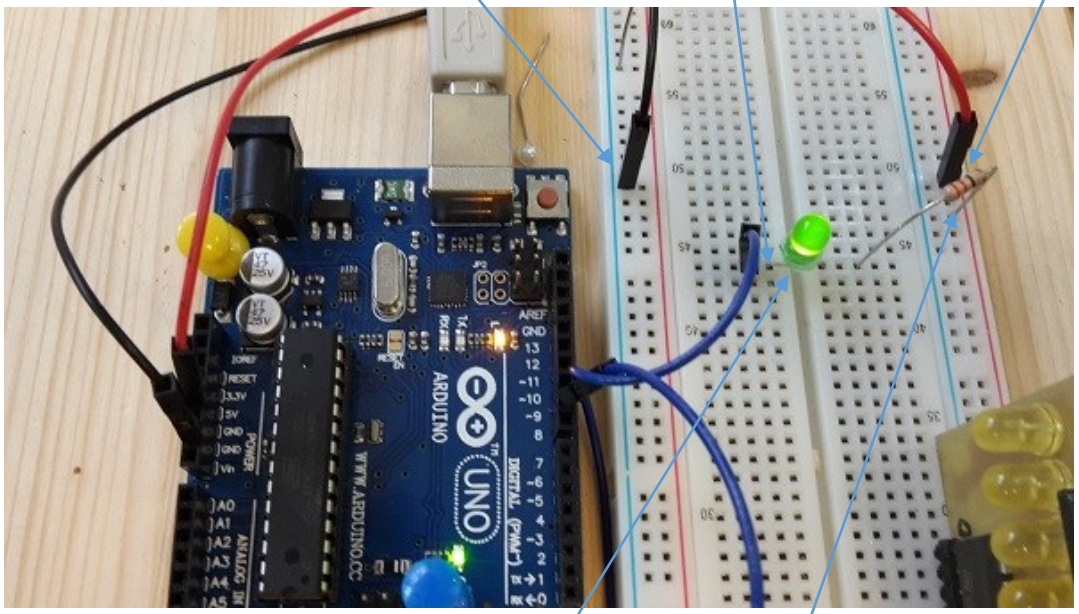
```
1
2  //Blinki_Led_PB5_P13-ANSI-C-Stile
3
4  #include <util/delay.h>
5
6  #define LED_DDRX DDRB      //Diese DEFINES können als Lesbarer Ersatz
7  #define LED_PORTX PORTB     //benutzt/eingesetzt werden
8
9  #define LEDPIN 5 //Weil LED ist an Port B, Bit 5. Nicht Arduino Nummer 13
10 #define LED_BITMSK (1 << LEDPIN)
11
12 void setup()
13 {
14   // put your setup code here, to run once:
15   // LED auf ARDUINO blinken lassen
16   DDRB |= (1 << 5);    // Setze das BIT 5 im Directine Regeister = Output
17       // LED_DDRX |= (1 << LEDPIN);  //Alternativ mit "Lesbaren" Defines
18 };
19
20 void loop()  // put your main code here, to run repeatedly:
21 {
22 PORTB |= (1 << 5);  // Setze das Bit 5 Mit "ODER"
23   //LED_PORTX |= LED_BITMSK;    // So ist es schön und lesbar
24 _delay_ms(200);
25 PORTB &= ~(1 << 5);  // Lösche das Bit 5 mit "UND Komplement"
26   //LED_PORTX &= ~LED_BITMSK;  // So ist es schön und lesbar
27 _delay_ms(800);
28 };
```

Externe dimmbare LED, mit **Sketch**

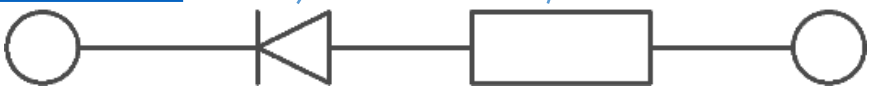
Lokal: K:\PHY\elfort\ 0_ARDUINO-Sktech\Dimm_P9_Led_PWM

```
#define  LEDPIN  9  //= tatsächlich Steckplatz 9
unsigned int gu16Val = 0;

void setup() {
  // put your setup code here, to run once:
  // LED extern PWM
  pinMode(LEDPIN, OUTPUT);
};

void loop() {
  // put your main code here

analogWrite(LEDPIN, gu16Val);
gu16Val++;
gu16Val &= 255;      //Modulo
delay(5);
};
```

Das Zauberwort heißt **PWM** = Pulsweiten Modulation mit ~**500Hz**
Vorwiderstand 330R – 560R = OHM
https://youtu.be/4bqq1U355Z0



Oszillogramm: https://youtu.be/CAvdiVNBoME



File Struktur mit der Arduino IDE

Diese Vorgehensweise bei der Arduino IDE ist umständlich.
Eine Versionsverwaltung durch Ordner-(Projekt)-namen
„MeinBlinki“ ist umständlich, da ebenfalls die, sonst übliche,:
„Main.c“ Datei *in MainBlinki.ino* „ umbenannt werden muss.

Mir ist erst später eine Vereinfachung eingefallen.
Konzept : Ein Ordner trägt den „ProjektNamen & Version und Datum“.

C:\ArduinoProgs\AT328Chopper_V2_15.03.23\...

Ein weiterer Ordner, <u>mit immer gleichen Namen</u> „main“ (Ordner!),
darin alle weiteren *.C und *.h Quellfiles .

C:\A ArduinoProgs \AT328Chopper_V2_15.03.23\main\main.ino

….und darin alle weiteren Files wie main.h, uart328.c, uart328.h….

Besonderheit bei Verwendung von Arduino-Sketch als C_Compiler

STRUKTUR:

1. Projektname = Ordnername
2. Darin \Ordner \main\
3. Darin = **main.ino (.ino statt .c)** = Basis C Datei + Header Datei **main.h**
4. Im selbem \main\ Order sind auch alle weiteren –C und .h Dateien

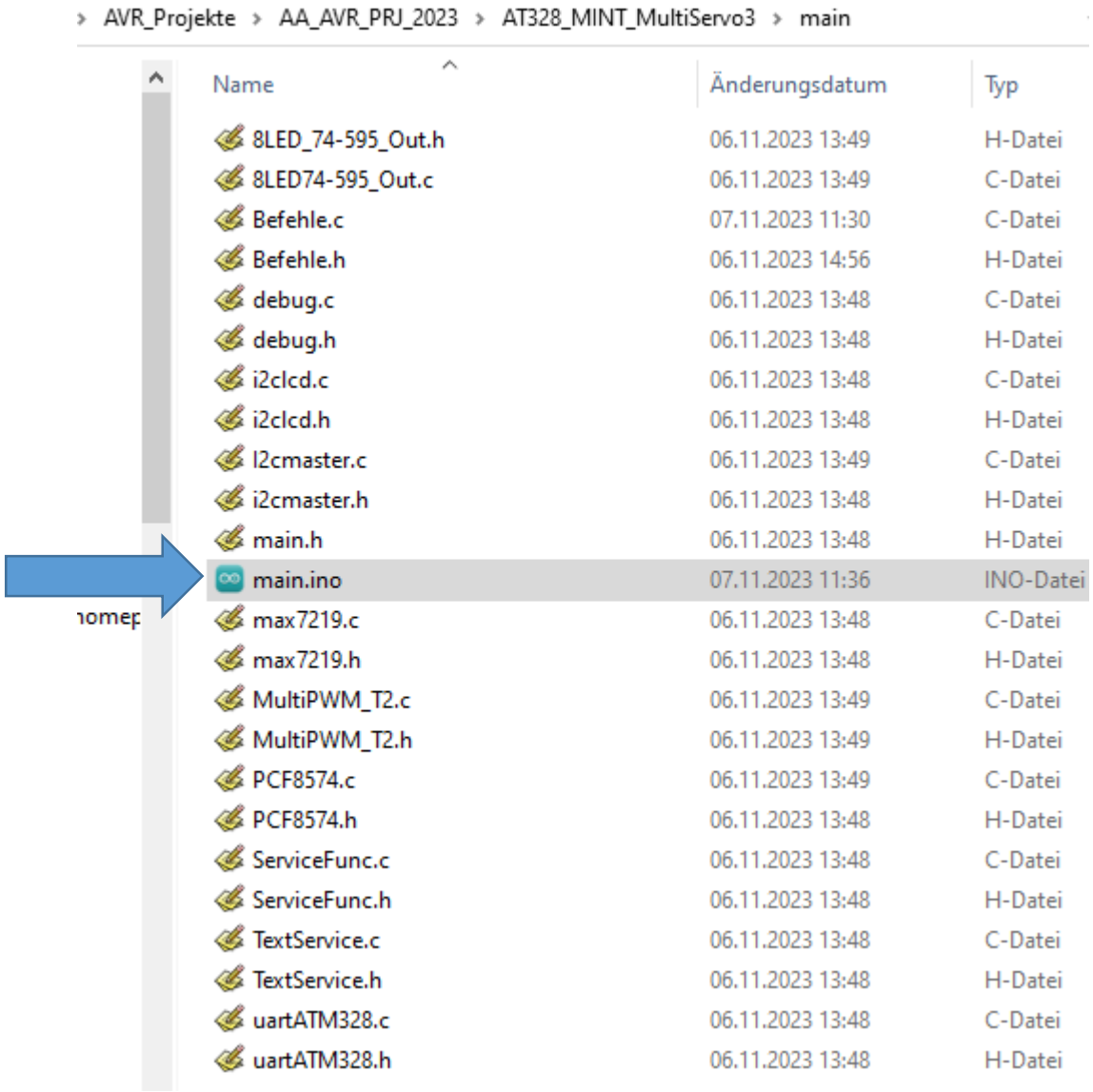
C:\AVR_Projekte\AA_AVR_PRJ_2023\AT328_MINT_MultiServo3\main

AVR_Projekte › AA_AVR_PRJ_2023 › AT328_MINT_MultiServo3 › main

Name	Änderungsdatum	Typ
8LED_74-595_Out.h	06.11.2023 13:49	H-Datei
8LED74-595_Out.c	06.11.2023 13:49	C-Datei
Befehle.c	07.11.2023 11:30	C-Datei
Befehle.h	06.11.2023 14:56	H-Datei
debug.c	06.11.2023 13:48	C-Datei
debug.h	06.11.2023 13:48	H-Datei
i2clcd.c	06.11.2023 13:48	C-Datei
i2clcd.h	06.11.2023 13:48	H-Datei
I2cmaster.c	06.11.2023 13:49	C-Datei
i2cmaster.h	06.11.2023 13:48	H-Datei
main.h	06.11.2023 13:48	H-Datei
main.ino	07.11.2023 11:36	INO-Datei

2. Möglichkeit: Einfacher zu handhaben, folgende Ordnerstruktur:

C:\AVR_Projekte\AA_AVR_PRJ_2023\AT328_MINT_MultiServo3\main



Projektname

Typische .c .h Dateien im selben **..\Projektname\main** Ordner

SKETCH Besonderheit

NOCHMALS: WICHTIG: ,,main**.c** in main**.ino** umbenennen!

Beispiel: ……**AT328MorsenTx\main\main.ino** (Das ist meine Methode)

AT328_ = Prozessor? = der AVR328p Prozessor .

Merke: **Ordnername = …..\Projektname\main\main.ino**

C:\…….\AvrProjekte**\SK328_Morsen_Tx**\main\main**.ino**

Und in diesen Ordner kommen alle anderen Files (*.c *.h) dazu.

ARDUINO Sketch C-Compiler (lästige) Besonderheiten

Arduino Laufzeitumgebung: Arduino Sketch Timer 0 Problem
Die Arduino-Umgebung verwendet diesen **Timer 0**, um verschiedene Prozeduren zu implementieren: Millis (), Micros (), Delay () , delayMicroseconds (), analogWrite (), tone () und noTone ()
Das ist lästig. Also „Don‘t use TIMER 0“

LÖSUNG: (Minimal-C)
NEW→ dann „lösche ALLES!“ → Schreibe:

```
#include <stdlib.h>
….
int main( void )
{
        while(1)
                {
                  // Your code goes here
                };
          return( 0 );
        };
```

TIPP Setup: Datei -- >Voreinstellungen: Da gibt es unter
Beispiel C:\Users\LocalAdmin\AppData\Local\Arduino15**preferences.txt**

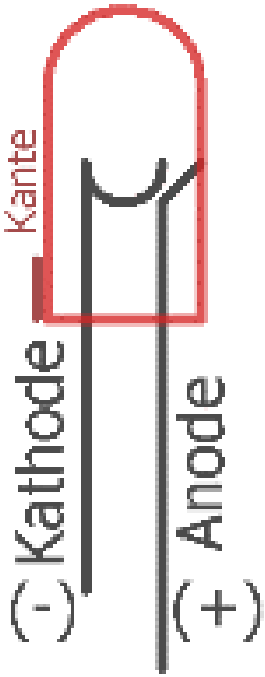
Packe alle Files :
HEADER .h
C-File .c
in den selben Ordner wie es das Arduino Sketch anlegt

```
extern „C“  //Die „extern“ Deklaration ist hier notwendig
        {
        #include „main.h“          // evtl nur main.h
        #include „uart328.h“
         };     ..usw.
                //Tipp: Alle sonstigen .h Files in main.h #includen
                // oder alle #Includes innerhalb von main.h


int main( void )
{
while(42)
                {
                  // Your code goes here
                };
  return( 0 );
};              //Dann kann man Sketch als alternative zum WINAVR akzeptieren
```

Typische anspruchsvolle Include Liste von Standart Includes

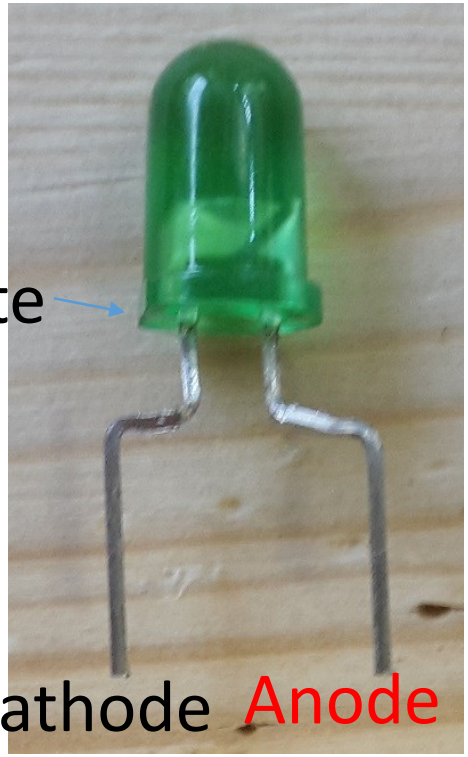
```
include <avr/sfr_defs.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <ctype.h>
#include <inttypes.h>
#include <string.h>
#include <avr/io.h>
#include <avr/wdt.h>
#include <avr/pgmspace.h>
#include <math.h>
#include <stdlib.h>
#include <avr/interrupt.h>
#include <util/delay.h> //delay_ms

extern "C„   // Besonderheit des Einladens von externen *.h Modulen
{
#include "main.h"
};
```

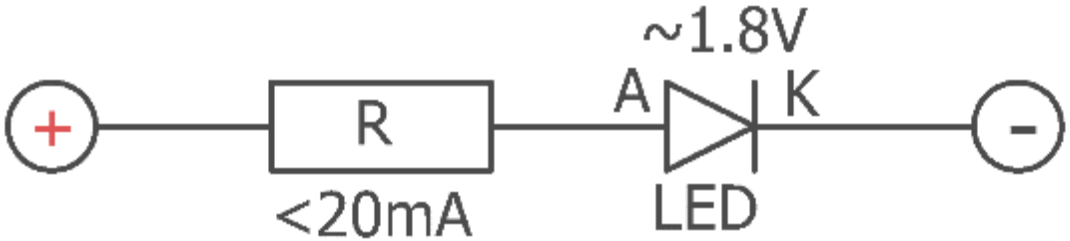
Vorbereitung Steckbrett mit LED **und** Widerstand



Berechnung des Vorwiderstandes.
Teile zusammensuchen und vorbereiten.
LED Anschlüsse **nicht am Gehäuse** biegen. ZANGE VERWENDEN und an Raster ausrichten. (Hinhalten abschätzen, kürzen „>=1cm“ einsteckl änge.
Nicht kürzer, sonst langt es nicht

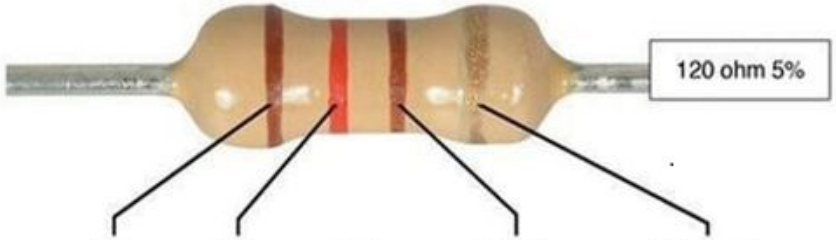
Rasterabstand beachten





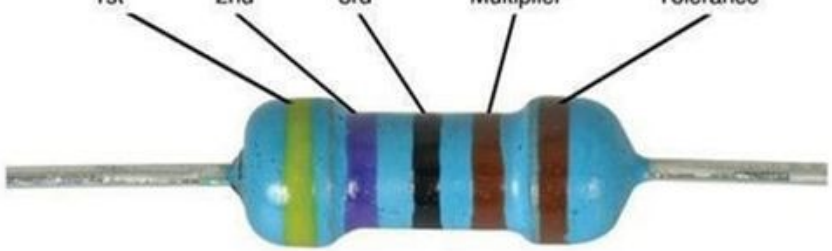
R=U/I = (5-1,8)V / 0.01A = 320--> ~330R

470R bis 560R tun es auch



120 ohm 5%

	1st	2nd	3rd	Multiplier		Tolerance	
0	Black	Black	Black	Black	1		
1	Brown	Brown	Brown	Brown	10^1	Brown	1%
2	Red	Red	Red	Red	10^2	Red	2%
3	Orange	Orange	Orange	Orange	10^3		
4	Yellow	Yellow	Yellow	Yellow	10^4		
5	Green	Green	Green	Green	10^5		
6	Blue	Blue	Blue	Blue	10^6		
7	Violet	Violet	Violet	Violet	10^7		
8	Grey	Grey	Grey	Grey	10^8		
9	White	White	White	White	10^9		
				Gold	0.1	Gold	5%
				Silver	0.01	Silver	10%



4.7k ohm 1%

WARUM „ANSI-C“ und nicht Arduino SKETCH?
Oder, die Sprache „C“ und ihre besondere Tauglichkeit mit 1 und 0 umzugehen.

Die Sprache „C“ ist Professionell. Und erlaubt verständliche, für Menschen noch gut lesbare Programme zu schreiben und gleichzeitig auf „**Maschinenebene = Bitebene**“ zu bleiben.
Leider ist „C“ nicht spontan intuitiv zu verstehen.
Die Bedeutung der oftmals **reduzierten, kringeligen Zeichen** muss man erlernen und sich daran gewöhnen.
Doch, **es wird überall mit Wasser gekocht**.

Aber dafür wird man belohnt und kann beim Programmieren immer gut erkennen, was „Sache der DINGE“ ist.
„C“ erlaubt „**Bit-nahe**“ und somit „den Gesetzen der binären Logik“ nahe zu bleiben.

Damit bleibt man direkt an den Drähten und Pins der Elektronik. Ideal also, um diese zu steuern.

So sah mein 1. Programmier-(Jugendstil)-Terminal 1984 aus..

```
// SWITCH in "C„  // Tasten prellen. BEWUSST UNSTABIL
extern "C" {
  //includes
#define LED_B5 0b00100000
#define SWITCH1 0b00000100
#define LEDB5_ON() (PORTB |= LED_B5)
#define LEDB5_OFF() (PORTB &= ~LED_B5)
bool  bState;
int main(void)
{
DDRB = 0b00100000;  //Setze Bit 5
DDRD &= ~SWITCH1;  // Lösche Bit2
PORTD |= SWITCH1;  // PullUP
while(42)
  {
 if( (PIND & SWITCH1) == false )
    {
    bState = !bState;
    if( bState == true)
      {      LEDB5_ON();      }
    else
      {      LEDB5_OFF();      };
    }; //sw
  }; //while
}; // loop
```

Füge einen Taster gegen GND an PINC Bit 2 = (2) ein.
LED = PORTB Bit 5 (13)

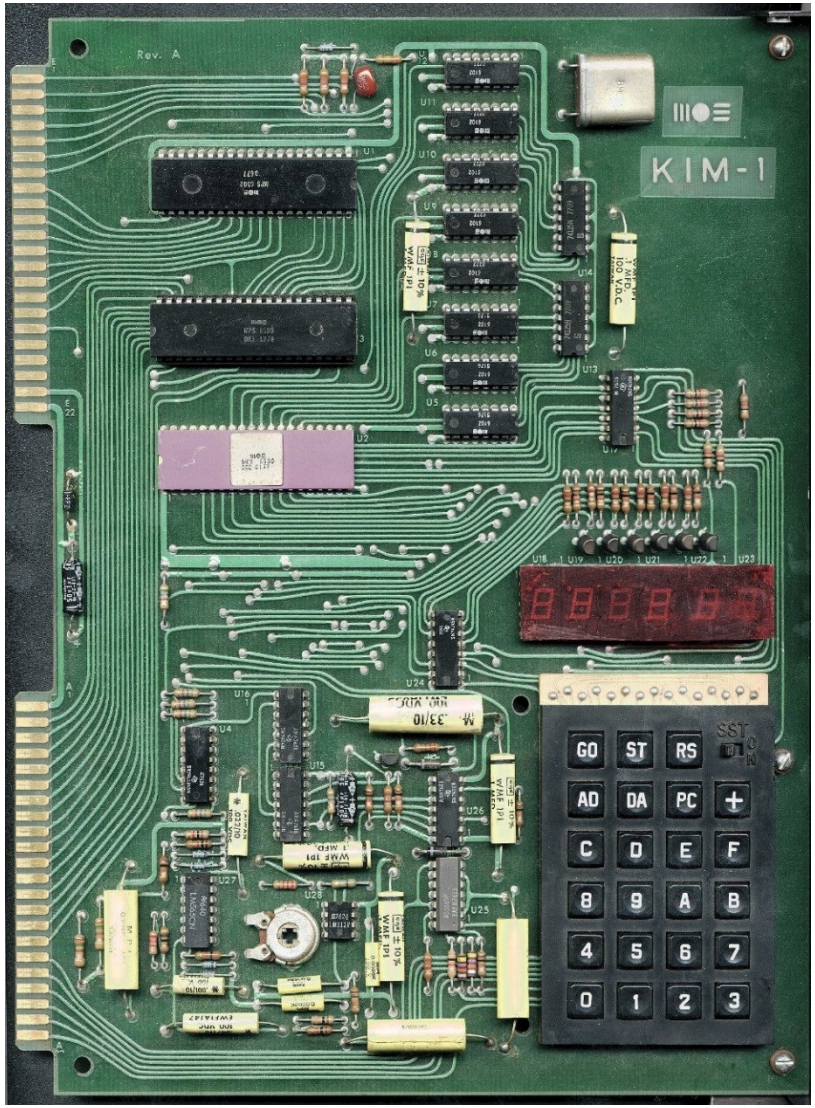
Gutes Youtube dazu:
https://www.youtube.com/watch?v=ORziktBksJw&t=448s

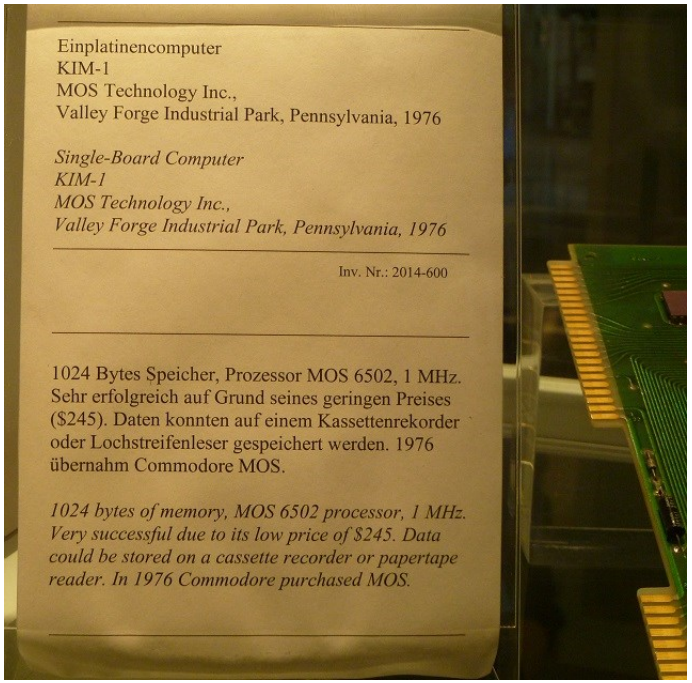
Teil 2
Die Programmierumgebung mit **ANSI-C** bis **C++**

Mikrocontroller arbeiten Maschinenbefehle ab. Sonst nichts !
Maschinebefehle sind intern vom Chip-Design „hardwired“ festgelegter ‚Microcode‘.

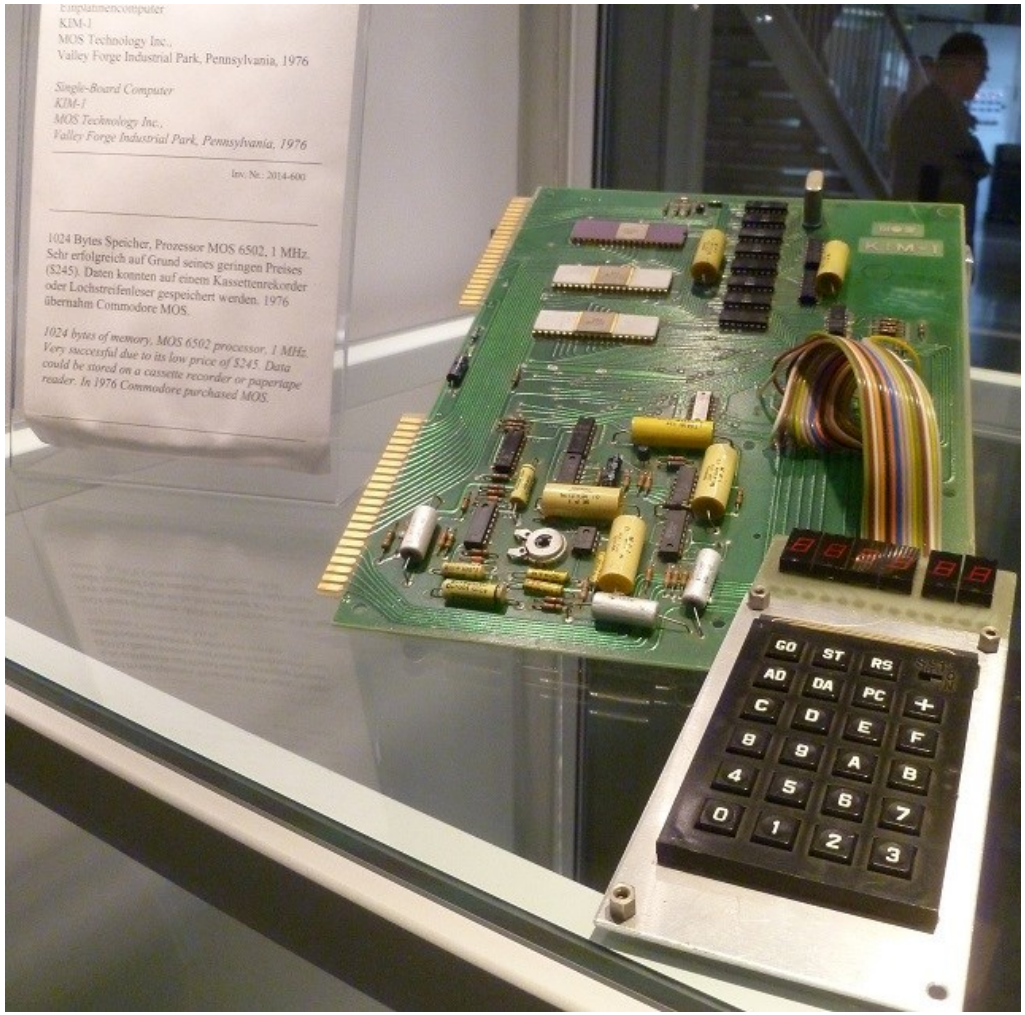
Mikroprozessoren der ersten Generation, wie der **KIM-1** von 1976 (Bild), hatten nur eine HEX-Tastatur (0..15) und Hex-Ziffernanzeigen (0..15). Sonst nichts !!

Das war so lehrreich wie minimalistisch steinig.
Um ein **Byte** mit Wert **6** in das Register **A** der „**Z80**A CPU“ zu schreiben konnten man entweder direkt Maschinencode schreiben
→ Tabelle: „**3E 06**“ oder den Alias-Ersatzcode,
Mnemonic Code genannt, verwenden.
→ „**LD A, 6**“. Das ist schon deutlich eleganter und fast schon wie lesbar für den Profi.
Doch Arithmetik möchte man so nicht machen.





Die **6502 CPU** mit 1 K externem RAM hat also weit weniger Hardware anzubieten als der ATMega328P mit dem wir hier arbeiten werden .
Eine 6502 CPU Erweiterung (**6510**) war die CPU des Commodore VC20 und C64



Der MOS **KIM-1** Einplatinen Computer, aus meiner Sammlung, von 1976 mit **6502** Prozessor und **1K RAM + ROM** ist jetzt im Deutschen Museum in der IT Abteilung ausgestellt.

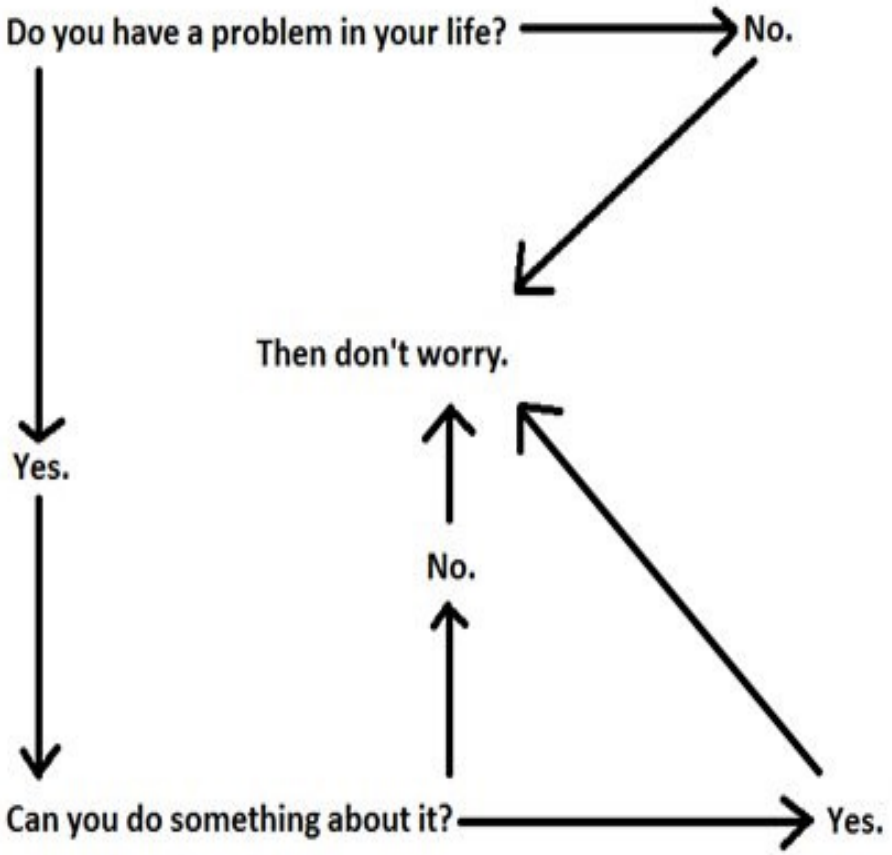
Was ist ein Algorithmus ?

Ein **Algorithmus** ist eine aus endlich vielen Schritten bestehende eindeutige Handlungsvorschrift.

Die Handlungsabläufe können Aufeinanderfolgen oder von Bedingungen abhängig sein.

Meist jedoch ergeben sich in der Laufzeit Daten oder Situationen, die Entscheidungen zulassen, wie es weiter geht.

Die Sprache C ist für die Implementierung bestens dafür gerüstet.



Wie sag ich es meinem µ-Controller in „C“

Um einen **Programmablauf** zu organisieren, gibt es in jeder Programmiersprache gewisse Standardtechniken die sich alle ähneln. *Dies soll jedoch kein **C-Crashkurs** sein*. Dennoch gehe ich *später* tiefer auf ***die besonders gelungenen binären Eigenschaften von „C“*** ein, die erklären warum „C“, trotz des explodierenden Entwicklungstempos im IT-Bereich, so erfolgreich durch die Jahrzehnte gereist ist.
Jede Stunde, in der ich persönlich „C“ gelernt habe, hat sich bezahlt gemacht!

unsigned int u16Wert = 123;

Unwahr = 0

Wahr ≠ 0

Zähl –Schleifen :

for(x= 0; x < 10, x++ **) {…};**

Logische Bedingung – Schleifen:

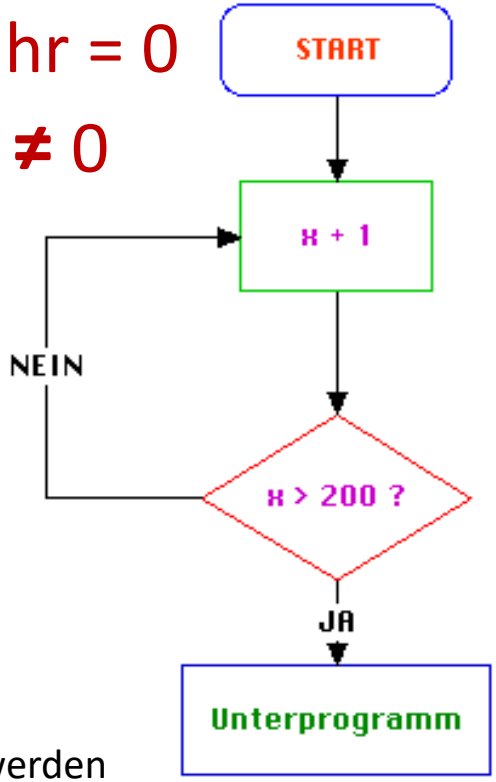
while(wahr/unwahr **) {…};**

do {….}while (wahr/unwahr **);**

Wenn, dann...sonst….. – Bedingungen

if(x > y) {} else {…};

Switch (expression**) case:** ….usw.



wobei in der Sprache „ANSI-C“ der
Wert „0 „als logisch **FALSE**
Werte „**Ungleich 0**“ = als logisch **TRUE** ausgewertet werden

Programm-Aufbau - ANSI C

*.c(pp) File → z.B. „main.c“

Alle notwendigen includes:

```
#include <math.h> ....
```

Globale Variablen:

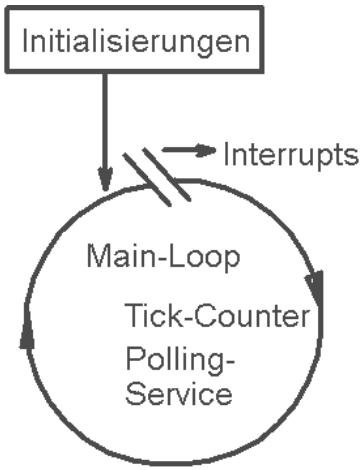
```
volatile unsigned long gu32Tick;
```

Interrupt Funktionen:

```
ISR (SIG_OUTPUT_COMPARE2)
  { gu32Tick++; };
```

Sonstige Funktionen:

```
void CheckCommand( unsigned char *pucRxStr )
{.........};

int main(void)
{
while(1)
   {

   };
};
```



*.h File → z.B. „main.h“

```
#ifndef MY_HEADER
#define MY_HEADER

//Konstanten
#define BAUD 115200
//Flags
#define ALLERLEI_FLAG       0x04
#define TASTEN_EREIGNIS   0x20

//Hardware Definitionen
#define LEDGREUN     (1<<5)

//Makros
#define _BV(x)  (1<<(x))
#define SET_L() (PORTB |= LEDGREUN)
#define CLEAR_L() (PORTB &=  ~LEDGREUN)

...usw..

#endif // MYHEADER
```

Minimale „C“ Programm Struktur im **„main.c“**

```
#include <avr/io.h>            //legt die Hardware des verwendeten µController fest
#include “main.h“      // Im Headerfile stehen grundlegende  Definitionen, Makros
//Funktionsprotoypen;  evtl in passenden  extern Deklarationen in „main.h“
// Globale Variablen;

//→ hier stehen  evtl. interrupt Funktionen.

int main(void)
    {
            Init();          // Mikrocontroller  initialisieren

            while(1)   // “!=0“ oder „1“ ist immer wahr”
                                                Bedeutet: laufe endlos im Kreis.
                {
                        //hier steht was immer zu tun ist und die Funktionsaufrufe.
                };      // Ausführung →    > ~500.000 mal pro Sekunde
    };                  // ende Main.
```

//mein Stil ist es, die Arbeitsblöcke mit untereinander stehenden Klammern zu schreiben.

Folgendes ist historisch,
aber ich lasse es mal noch im Skript

WINAVR Installieren:

Quelle:

https://sourceforge.net/projects/winavr/

https://sourceforge.net/projects/winavr/files/latest/download

WINAVR.20100110-install.exe ist die schon „sehr alte“ letzte Version.

„Es wird berichtet dass die PATH Variable überschrieben wird.“ !?

Ich hatte jedoch niemals Probleme.

Systemsteuerung → System & Sicherheit → System →
Erweiterte Systemeinstellung → Umgebungsvariablen
PATH →
da steht bei mir :

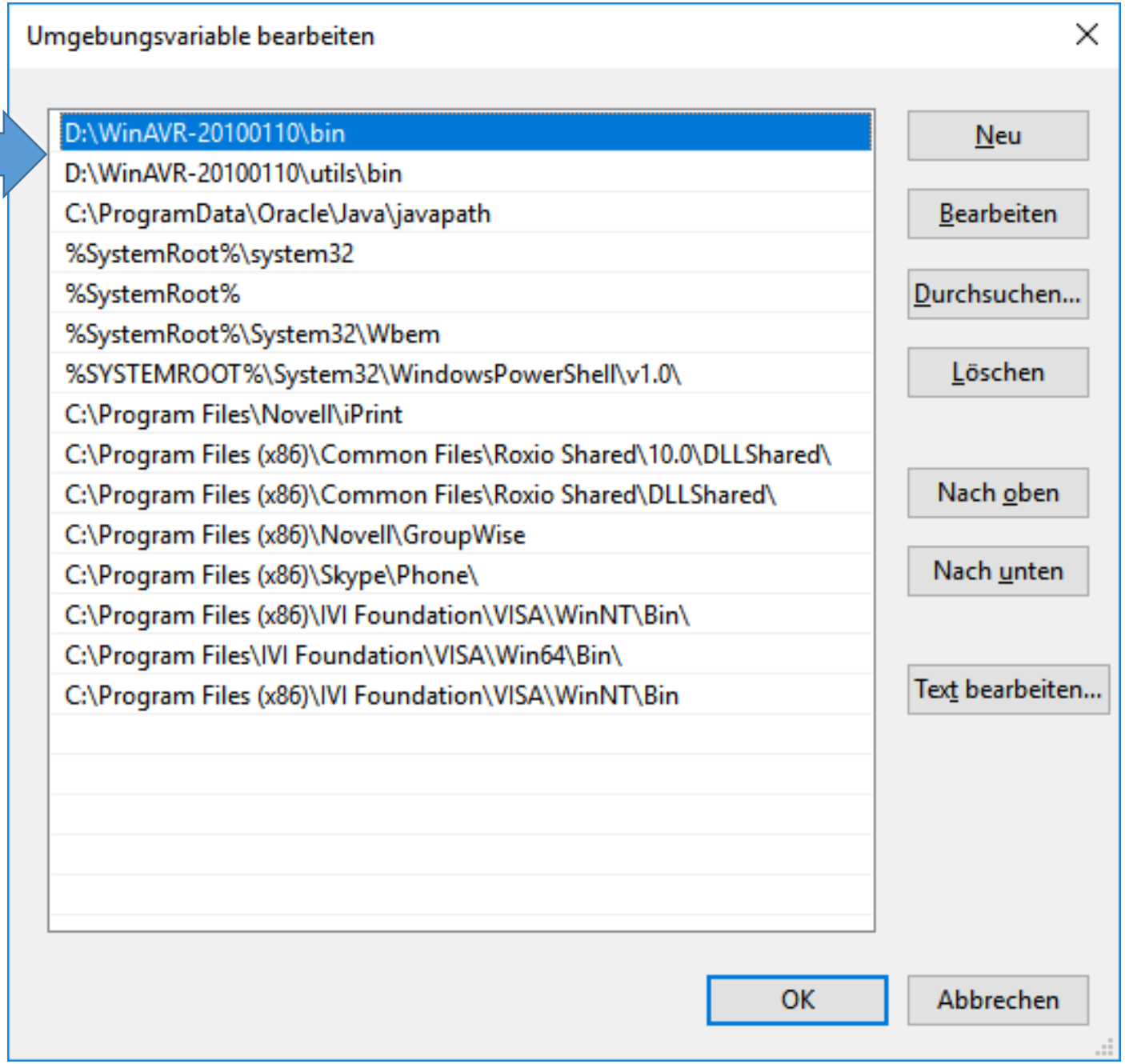
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps;

Und unter **Systemvariablen**: → siehe nächste Seite

„Es wird berichtet dass die **PATH** Variable überschrieben wird.“ !?

Ich hatte jedoch niemals Probleme.

Hier sieht man Die beiden Einträge



FIX Probleme mit WINAVR unter Windows 8x/10

Der Compiler meldet einen „**Sync Child**“ Fehler (Ui-Jui: was immer das hier soll)

main] sh 4208 sync_with_child: child 4432(0x124) died before initialization with status code 0x0
37434 [main] sh 4208 sync_with_child: *** child state waiting for longjmp

die **Gute Nachricht**, es gibt eine einfache Lösung: (für 32Bit und 64Bit gertrennt)
Und diese steht in meiner Webseite:
http://homepages.uni-regensburg.de/~erc24492/WINAVR_unter_WIN10/WINAVR_unter_WIN10.html
Oder K:\PHY\elfort\1A WINAVR und AVRStudio Compiler
Dazu: https://www.mikrocontroller.net/articles/WinAVR
Zu Unterscheiden ist nach 32Bit und 64Bit Windows
Oder:
Download:
http://www.madwizard.org/electronics/articles/winavrvista
Unter → Other Problems: gehe zu Link
→ Download: msys-rebased.zip oder msys-1.0-vista64.zip, entpacke…
msys-1-0.dll und **kopiere** diese in den Ordner:
C:\WinAVR-20100110\utils\bin.
(and overwrite the existing msys-1.0.dll)

Windows 10 Problem mit „Sync Child“ Error Meldung des Compilers

Bei **Windows 10** gibt es Probleme beim Compiler, die aber lösbar sind.

Siehe auch https://www.mikrocontroller.net/articles/WinAVR
und auch **http://www.madwizard.org/electronics/articles/winavrvista**
Für 32Bit:

Download msys-rebased.zip and overwrite the existing msys-1.0.dll in te **utils\bin** subdirectory of your WinAVR director

Für 64Bit W10
If you are using Vista 64-bits, you will probably have the sync_with_child or similar errors.
In that case download msys-1.0-vista64.zip with a modified msys-1.0.dll.
--> Lade für windows **64Bit** folgenden Ordner bzw Datei "**msys-1.0.dll**"
innerhalb von WINAVR Win10 Problem/msys-1.0-vista64.zip
oder direkt: msys-1.0-vista64/msys-1.0.dll
msys-1.0-vista64

found solution.
Kopiere diese in
utils\bin directory (WinAVR)

ANSI-C Compilertest

Projekt mit WinAVR

C-Compiler starten → **C:\WinAVR-20100110\pn.exe** starten

Bsp: Da liegen alle *.c *.h und mymainPrj.pnprj Files Öffnen..

Diese Adresse Zeile kopieren und diese dann in WinAVR öffnen.

Open Project mymain.pnprj

→ WINAVR MENÜ: Projekt mit WINAVR öffnen:

main.c
main.h
make
MyMainPrj.pnproj

Fiel: Open Project(s)

→ „MyMain.pnproj“

Arbeiten: Menü: Öffne “main.c “

Compilieren: Menü: Tools→ Make All

WICHTIG WINAVR GNU-C Library Help

Die Hilfe zur Funktions-Bibliothek ist zu finden im Ordner:

file:///C:/WinAVR-20100110/doc/avr-libc/avr-libc-user-manual/modules.html

Neuer Tab | avr-libc: Module Index

file:///C:/WinAVR-20100110/doc/avr-libc/avr-libc-user-manual/modules.html

Apps | Simplicity Studio – ... | IT-Einkauf - Universi... | Google Übersetzer | Google | Weitere Lesezeichen

AVR Libc Home Page

AVR Libc Development Pages

Main Page | User Manual | Library Reference | FAQ | Alphabetical Index | Example Projects

Modules

Here is a list of all modules:

- <alloca.h>: Allocate space in the stack
- <assert.h>: Diagnostics
- <ctype.h>: Character Operations
- <errno.h>: System Errors
- <inttypes.h>: Integer Type conversions
- <math.h>: Mathematics
- <setjmp.h>: Non-local goto
- <stdint.h>: Standard Integer Types
- <stdio.h>: Standard IO facilities
- <stdlib.h>: General utilities
- <string.h>: Strings
- <avr/boot.h>: Bootloader Support Utilities
- <avr/eeprom.h>: EEPROM handling
- <avr/fuse.h>: Fuse Support
- <avr/interrupt.h>: Interrupts
- <avr/io.h>: AVR device-specific IO definitions
- <avr/lock.h>: Lockbit Support
- <avr/pgmspace.h>: Program Space Utilities
- <avr/power.h>: Power Reduction Management
- <avr/sfr_defs.h>: Special function registers

Die traditionelle Compiler Steuerdatei „makefile“

Hier nur die wichtigsten:

In „makefle“ (ohne **.**Suffix) stehen von Nerds entwickelte Steuerkommandos:

MCU = atmega328p
F_CPU = 16000000 #hier kein UL anhängen

AVRDUDE_PROGRAMMER = arduino
AVRDUDE_LOADERBAUD = 115200
AVRDUDE_PORT = com8 # programmer connected to serial device

Das muß man selbst einstellen

TARGET = main
SRC = $(TARGET).c

AVRDUDE_FLAGS = -p$(MCU) -P$(AVRDUDE_PORT) -b$(AVRDUDE_LOADERBAUD) -c $(AVRDUDE_PROGRAMMER)

 # ist nichts anderes als:
#AVRDUDE_FLAGS = -p$(atmega328p) -P$(com8) -b$(115200) -c $(arduino)

Auszug aus dem „makefile“ des GCC Compilers für die „AVRDUDE“ Programmer Software, und dem internen Bootloader

F_CPU = 16000000 Quarz
MCU = atmega328p

AVRDUDE_PROGRAMMER = arduino
AVRDUDE_LOADERBAUD = 115200
AVRDUDE_PORT = com8 # programmer connected to serial device

#Flags in Alias Namen verteilt → TARGET = main
AVRDUDE_WRITE_FLASH = -D -Uflash:w:$(TARGET).hex:i
AVRDUDE_VERBOSE = -v –v
#Volle Zeile:
AVRDUDE_FLAGS = -p$(MCU) -P$(AVRDUDE_PORT) - b$(AVRDUDE_LOADERBAUD) -c $(AVRDUDE_PROGRAMMER)

AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)

program: $(TARGET).hex $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH)
avrdude –p atmega328p –P com8 –b115200 –c arduino -D -Uflash:w:main.hex:I

Oder→ (alles eine Zeile)
avrdude –p MCU –P AVRDUDE_PORT –b AVRDUDE_LOADERBAUD –c arduino -D -flash:w:main.hex:I

Datentypen in ANSI-C89

Die Sprache C geht sehr streng mit Datentypen um.

Das ist „**nicht lästig**“ sondern ein **großer Vorteil**.

Denn so wird man gezwungen:

A) …zu überlegen, und so zu verstehen was man tut.
B) …Speicher wird optimal und sparsam genutzt.
C) …Man weiß immer, mit welchen (Datentypen) man es zu tun hat.

(sofern man seine Variablen etc. sprechende Namen und eine standardisierte Schreibweise gibt.
Dies erläutere ich noch ausführlich und ist wichtiger Bestandteil des Kurses.)

Wertebereiche & Datentypen in Ansi-C89

Hier folgt eine Liste aller elementarer numerischer Variablentypen die in C zur Verfügung stehen:

Type	Keyword	Bytes	Range
character	char	1	-128 .. 127
unsigned character	unsigned char	1	0 .. 255
integer	int	2	-32 768 .. 32 767
short integer	short	2	-32 768 .. 32 767
long integer	long	4	-2 147 483 648 .. 2 147 483 647
unsigned integer	unsigned int	2	0 .. 65 535
unsigned short integer	unsigned short	2	0 .. 65 535
unsigned long integer	unsigned long	4	0 .. 4 294 967 295
single-precision floating-point (7 Stellen)	float	4	1.17E-38 .. 3.4E38
double-precision floating-point (19 Stellen)	double	8	2.2E-308 .. 1.8E308

Die „tatsächlich“ verwendete Bitbreite ist jedoch Maschinen bzw. Prozessor abhängig

Datentypen in GNU-C und WINAVR

Im Headerfile **<stdint.h>** sind die oft verwendeten Kurzschreibweisen für Standard-Datentypen definiert in der die **Bit-Zahl** erkennbar ist.

Exact-width integer types
Integer types having exactly the specified width

typedef signed char	**int8_t**
typedef unsigned char	**uint8_t**
typedef signed int	**int16_t**
typedef unsigned int	**uint16_t**
typedef signed long int	**int32_t**
typedef unsigned long int	**uint32_t**
typedef signed long long int	**int64_t**
typedef unsigned long long int	**uint64_t**

float = 4 Byte
double = 8 Byte

Das ist soweit selbsterklärend.

Eine BYTE-Variable mit 8 Bit wird also so deklariert:
uint8_t u8StatusByte;

Leeres Ansi-C Programm → Compilertest.

```
//Interne LED
#include <avr/io.h>  //darin enthalten # <avr/sfr_defs.h>


// ******************
int   main( void )
// ******************
{
DDRB = 32;          Port PB.5 = Output = LED auf Platine
                    //32 = 0b00100000 = 0x20
while(42)   //for ever true {die Antwort auf alle Fragen}
            {   //keine zeitliche Bremse
             PORTB = 32;       //0b00100000;    // LED = ON
             PORTB = 0;        //0b00000000;    // LED = OFF
            };
};
```

LED → >~500000/Sekunde On-Off

…siehe Oszilloskop Bild von PB.5

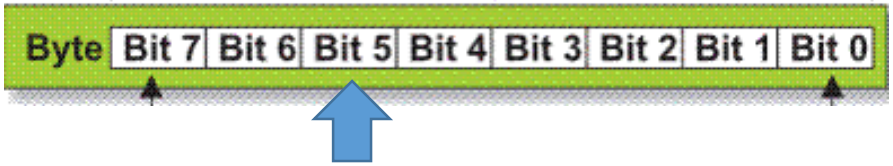
Vorwegnahme

Wir wollen ja bald was am laufen haben
Warum, wie, klärt sich bald..

ANSI-C WINAVR intern Blink

µC „Hallo Welt“

```
#define SOFTWARE     "Main-Blinki"
#define HARDWARE     "Arduino-UNO ATMega328"
#define DATUM        " 12:00Uhr 29.03.2016 "
#define AUTOR        "=/\= Christof Ermer =/\= "
"

#include <avr/io.h>
#include <util/delay.h>

// ******************
int   main( void )
// ******************
{
DDRB  = 32;    //0b00100000;  ///Data Direction Register B
while(1)
            {
             PORTB = 32;   //warum 32, dazu später mehr
                _delay_ms( 500 );
            PORTB = 0;
                _delay_ms( 500 );

            };
};
```



Das fünfte Bit, woran die LED liegt, ist

0b00**1**00000 =

(1<<**5**) =

32

0x20

oder #define PB5 (1<<5)

und nicht „5“ !!

Man könnte also auch schreiben

#define YELLOW_LED (1<<5)

…doch das sehen wir später genauer

SOS - Experiment

Interne LED

Output: Arduino ist Pin13

```
#define T_PUNKT                 300
#define T_STRICH                700
#define T_LUECKE                150
#define T_ZEICHEN_ABSTAND       1200
#define T_WORT_ABSTAND          1500

#define LED_BIT   32      //0b00100000 = Bit5

void TX_3Zeichen( unsigned int   u16OnTime)
{
        PORTB = LED_BIT;
        _delay_ms( u16OnTime );
        PORTB = 0;
        _delay_ms( T_LUECKE );

        PORTB = LED_BIT;
        _delay_ms( u16OnTime );
        PORTB = 0;
        _delay_ms( T_LUECKE );

        PORTB = LED_BIT;
        _delay_ms( u16OnTime );
        PORTB = 0;

        _delay_ms( T_ZEICHEN_ABSTAND );
};
```

```
// ******************
int main(void)
// ******************
{
DDRB  = LED_BIT;   //Data Direction Register B
while(1)
        {
        TX_3Zeichen( T_PUNKT );
        TX_3Zeichen( T_STRICH );
        TX_3Zeichen( T_PUNKT );
        _delay_ms( T_WORT_ABSTAND );
        };
};
```

µC Ctrl → Special Funktion Register (SFR)

Vorwegnahme

PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PINC – The Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Die wichtigsten **SFR**, beispielsweise von **PORTC**, sind 8-Bit große Speicherbereiche, die wie Variablen geschrieben (man spricht „gesetzt“) oder gelesen werden können.
Das **DDR**x (x= B,C,D) ist das „**D**ata **D**irection **R**egister“.
Wie der Name schon sagt. Eine „**1**“ macht das zugehörige Bit des Portes „x“ zum Ausgang. Standardmäßig ist es „**0**“ und damit ein Eingang →(ist passiv).
BSP: DDRB = 0b00000001; → das Bit-0 wird auf „1“ gesetzt→ Bit-0 = Ausgang.

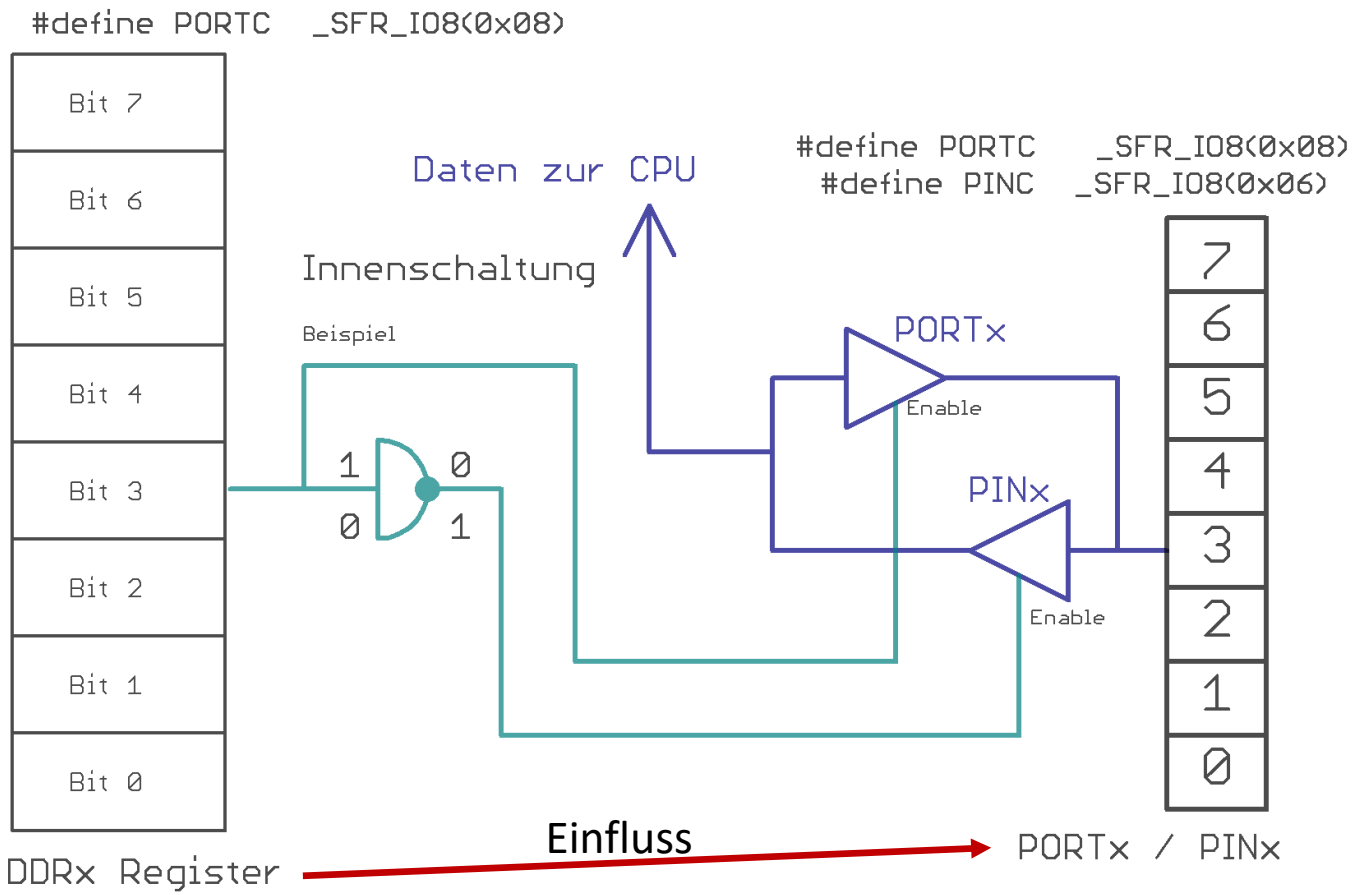
Einfluss des DDRx Registers auf IN-OUT Richtung von Port-Pins mit Special Function Register - SFR

Vorwegnahme

Wie schon erwähnt, kann mit Hilfe der den PORT-Pins zugeordneten SFRs jedes einzelne Bit für sich getrennt konfiguriert werden.

Auch wenn ein Port nur 6 Bit hat, ist ein SFR-Byte immer 8 Bit.

DDRx:
Bit = 0 Input
Bit = 1 Output

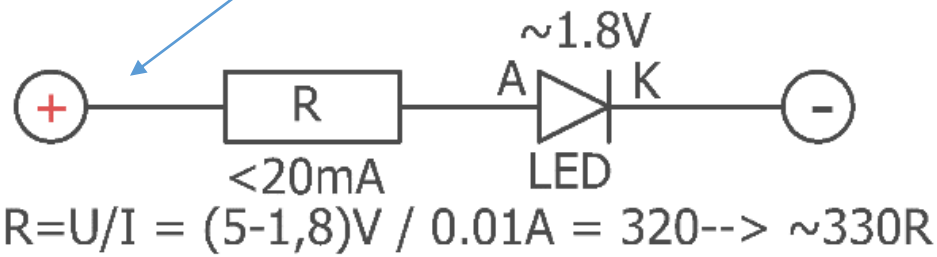


WINAVR GNU C Beispiel: Externe LED an PORTB.5

```
#include <avr/io.h>
#include <util/delay.h>

int main(void) →
        {
        DDRB = 32;
        while( 1 )
                {
                PORTB = 32;
                _delay_ms(500);
                PORTB = 0;
                _delay_ms(500);
                };

        };
```

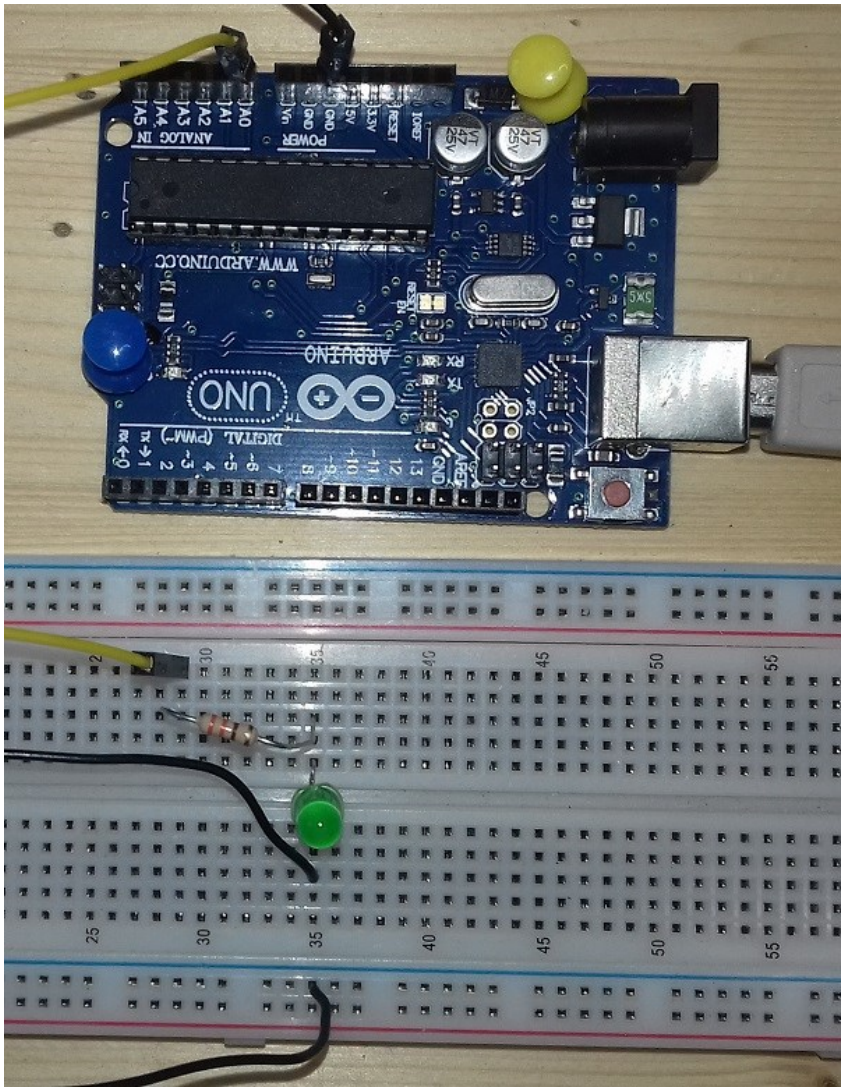
Wie und warum dieses Programm funktioniert, sehen wir *später……*
sehr viel später…

PORTB.5 = [13] auf Arduino Uno Board

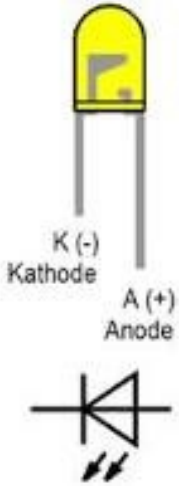


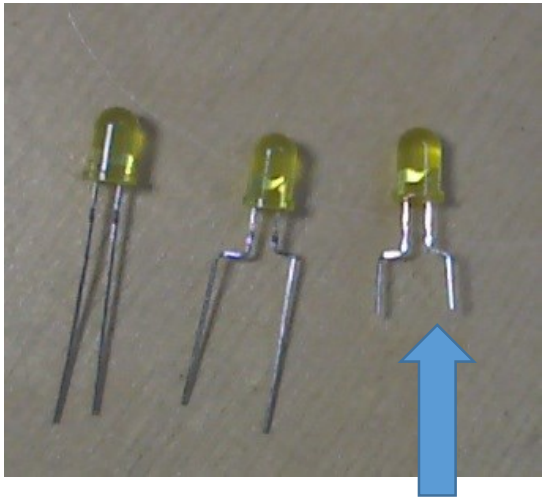
Aufbau mit Steckbrett:
PINB.5 → ~330..470..560R → Anode LED →Kathode GND.

Eine LED an PORTB, Bit 5



Die LED Drähte niemals schief, wie die Pilze im Wald, biegen !!
Zange benutzen!
Vorher Pin-Abstand abmessen .





Einsteckbare Beinchen.
Nicht zu kurz !: > 1 cm

Oben = PLUS 5V

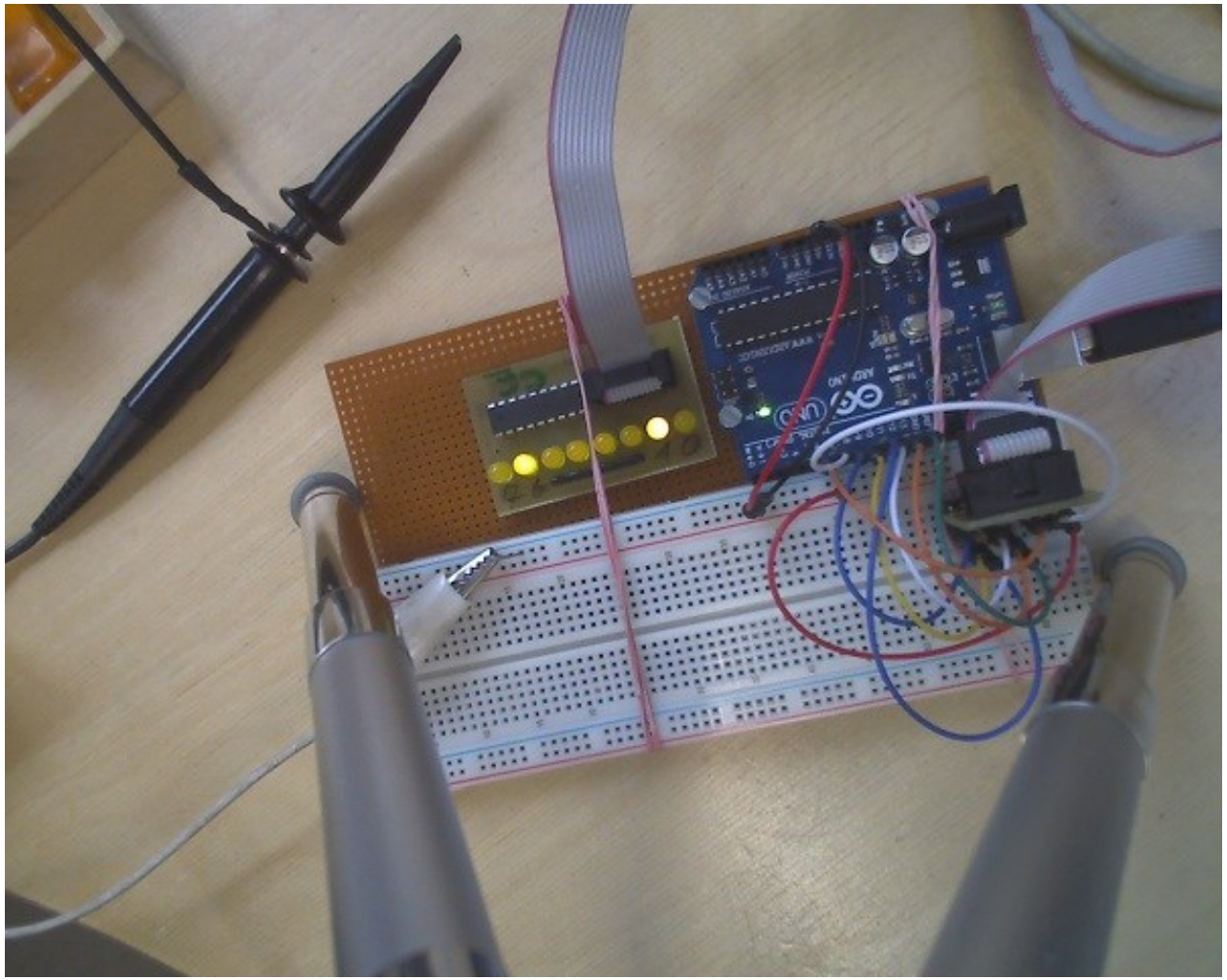
Youtube→
http://youtube/7z8I2Ib0cSU

Unten = MINUS/GND

En kleines Beispiel für eine laufende Bitmuster Ausgabe

Typischer Aufbau: Befestigung auf **nichtleitende** Trägerfläche (bsp. Holz)

Youtube Link: http://youtu.be/bCqgU04aVAY



Vorwegnahme

Ard328_main.h

```
#ifndef MAIN_HEADER  //Prevents multiple includes
  #define MAIN_HEADER

extern "C" {
#include <inttypes.h>
#include <ctype.h>
#include <avr/io.h>
#include <stdlib.h>
#include <util/delay.h>
#include <avr/wdt.h>
}

#define INTERN_LED_BIT      (1 << PB5)
#define INTERN_LED_INIT()   (DDRB |= INTERN_LED_BIT)
#define INTERN_LED_ON()     (PORTB |= INTERN_LED_BIT)
#define INTERN_LED_OFF()    (PORTB &= ~INTERN_LED_BIT)
#define INTERN_LED_TOGGLE() (PORTB ^= INTERN_LED_BIT) //XOR

      #define PUNKT 100
      #define STRICH 300
      #define SPACE  500
#endif
```

„C“ in Arduino IDE

```
// SWITCH in "C„  // Morsen. mit _delay_ms()
extern "C" {
 #include "Ard328_main.h"
}
// ************************
void Wait( double dDelay )
// ************************
{
for(double dNN=0; dNN < dDelay; dNN++)
  {
  wdt_reset();  //WATCHDOG
  _delay_ms(1);
  }
};
// ****************************
void Morse( double dPuls )
// ****************************
{
INTERN_LED_ON();
Wait( dPuls );
INTERN_LED_OFF();
_delay_ms( 100 ); //Lücke
}
```

```
// ######################################
int main(void)
// ######################################
{
INTERN_LED_INIT(); //DDRB = 0b00100000;  //Setze
Bit 5
wdt_enable(WDTO_1S);  //wdt_disable();

while(42)
  {
  wdt_reset();  //WATCHDOG
  Morse( PUNKT  );
  Morse( PUNKT  );
  Morse( PUNKT  );
  Wait( SPACE);

  Morse( STRICH  );
  Morse( STRICH  );
  Morse( STRICH  );
  Wait( SPACE);

  Morse( PUNKT  );
  Morse( PUNKT  );
  Morse( PUNKT  );
  Wait( SPACE);

  Wait( 2000 );
  }; //while
}; // loop
```

Prioritäten Reihenfolge der ANSI-C Operatoren

Übersicht aller C-Operatoren mit Vorrang und Assoziativität
Siehe auch:
https://docplayer.org/8573029-Programmieren-in-c-und-c-universitaet-regensburg-fakultaet-physik.html

( ) Funktionsaufruf

[ ] Arrayelement

. Strukturelement

-> Zeiger auf Strukturelement

`!` logisch NOT

`~` bitweises Komplement

`-` unitäres Minus

`++` Inkrement

`--` Dekrement

`&` Addresse

`*` Inhalt

`( type )` Cast

`sizeof` Größe in Byte

`*` Multiplikation

`/` Division

`%` Ganzzahl-Modulo

`+` Addition

`-` Subtraktion

`<<` bitweise Linksverschiebung

`>>` bitweise Rechtsverschiebung

< arithmetisch kleiner als

> arithmetisch größer als

<= arithmetisch kleiner gleich

>= arithmetisch größer gleich

== arithmetisch gleich

!= arithmetisch ungleich

`&` bitweise AND links nach rechts

`^` bitweise XOR links nach rechts

`|` bitweise OR links nach rechts

`&&` logisch AND links nach rechts

`||` logisch OR links nach rechts

`? :` Fragezeichenoperator (verkürztes if) **links nach rechts**

Zuweisungsoperator(en)

`=`
`*=` **`/=`**
`+=` **`-=`**
`&=` **`und gleich`**
`|=` **`oder gleich`**
`^=` **`XOR gleich`** usw. **rechts nach links**

`,` Kommaoperator links nach rechts

Standard Bit Operatoren versus logische Operatoren

```
#define _BIN_AND_          &          //Binär AND
#define _BIN_OR_           |          //Binär OR

                                      And/OR Zuweisungsoperatoren:
#define _BIN_AND_SET       &=          //Und=  löscht Bits OHNE Komplement
#define _BIN_OR_SET        |=         //Oder= setzt Bits

#define _BIN_CLEAR_SET_    &= ~       //Und=~ löscht Bits MIT Komplement   „~“

#define _TOOGLEBIT_        ^=         //XOR= TOGGLE  BITS  Wie ein Wechselschalter

#define _UNGLEICH_         !=

#define _AND_LOG_          &&         //Logisch AND   ( nicht Binär )
#define _OR_LOG_           ||         ((LOGIISCH OR   ( nicht Binär )
```

IT-Wissen. Von Bits und Bytes

Grundlagen der
Programmierung in „C“
weitgehend systemunabhängig.

Wir benötigen einen leider etwas langen Anlauf, bis wir zum praktischen Arbeiten mit dem Mikrocontroller übergehen können…
……..Geduld…….

0x03 = 0b00000011 = 0d3

ENDLICH ECHTER DIGITAL-UNTERRICHT

Programmierung im professionellem Stil und mit bewährten Methoden

Ausblick:
Bewährte Strukturen & Methoden:

Irgendwann komm dies alles vor, nur jetzt noch nicht

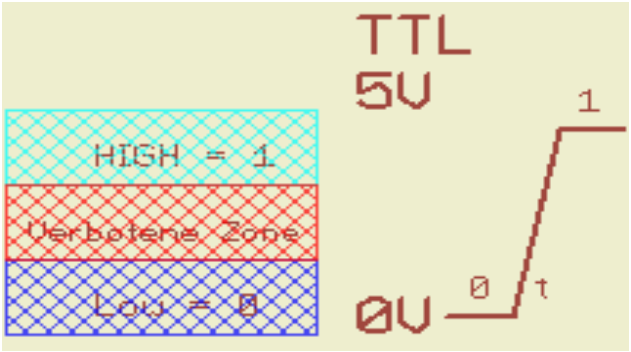
* Programmieren mit klarer Konvention des Programmierstiles:
  „**gu16V**ariablen**N**amens**G**ebung“
* Konsequente Programmlauf Steuermethode via „**Einzelbit Flags**“.
* **Tick Counter**, Timer Interrupt getrieben.
* Asynchrone **serielle Daten** empfangen´(**Rx**), sammeln und senden (**Tx**).
* Text- Befehle + Daten interpretieren und ausführen. Ping → Pong
* Vermeidung von **Polling** (=unkontrollierte, häufige Abfrage in „main“).
* Debouncing (entprellen) von Tasten.
* Daten Input Output mit ge**shift**eten **CLK - DATA** IO Protokoll.
* Sensorsignale normieren und auswerten.
* Register Verwendung um die µController-Fähigkeiten kontrolliert zu nutzen.

Repräsentiert werden die Zustände **0** und **1** einfach mit **Spannungspegeln**.
Spötter meinen: „wahrscheinlich sind es mehr Nullen als Einsen“.

Denkbar wären auch Zahnräder, Dampf, Relais, Elektronenspins oder Quantenzustände, aber ‚diese‘ wohl erst in der Zukunft.

Null ist = ~0V
Eins ist = ~5V = **TTL** Spannung = **T**ransistor-**T**ransistor-**L**ogik.
Gebräuchlich sind auch **3.3V**olt Logikpegel, und weniger.

Frage: **Ab wann wird die 0 zur 1 oder umgekehrt** ?



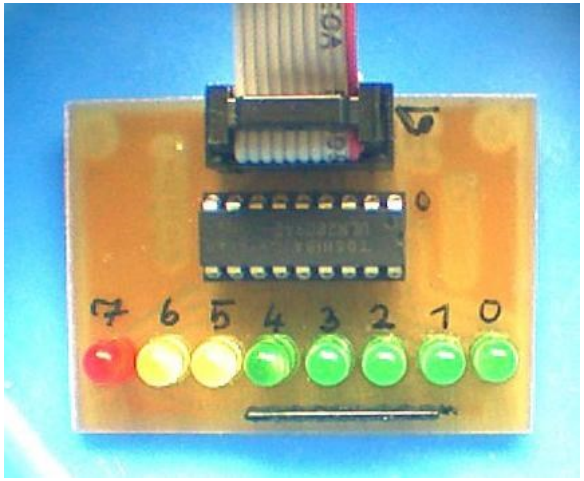
Es reicht in etwa die Vorstellung →Spannungslevel:
Oberes Drittel = 3/3 Zone von z.B. 5V = **schon EINS**
mittleres Drittel = verbotener, unklarer Bereich!
unteres Drittel = 1/3 Zone von z.B. 5V = **noch NULL**

Bits und Bytes

Merke: Das **Byte** ist die kleinste binäre Dateneinheit in unseren Computersystemen. **8 Bit** nennt man **ein Byte**

Die Bits werden von **rechts nach links** , **beginnend** mit der **0** durchnummeriert.

Bit Nummer 0 bis 7.
…von rechts nach links.
Niemals 1 bis 8 !



Bit: **7 - 6 – 5 -4 – 3 – 2 – 1 - 0**

Sehr brauchbar:
Ein Byte angezeigt als LED Lämpchen.

So kann der Byteinhalt direkt „gesehen“ werden.

Beschriftung:
Bit: **7-6-5-4-3-2-1-0**

8 Bit = 2^8 bieten **256** Kombinationsmöglichkeiten von **0**en und **1**sen in einem Byte.

Byte Segmente

Um die einzelnen Bits im Byte besser kontrollieren und verstehen zu könne hat sich die Aufteilung eines 8 Bit Bytes in zwei 4 Bit Segmente durchgesetzt.

Ein Byte kann dezimale Werte **von 0 bis 255** *tragen*.
(weil $\mathbf{2^8}$ → 256 Kombinationen von **0 und 1** möglich sind)
Das ist ganz „nett“ aber unübersichtlich wenn man gezielt auf ein bestimmtes **Bit** innerhalb des Bytes zugreifen möchte.

Ein **4 Bit Segment** dagegen hat nur $\mathbf{2^4 = 16}$ Kombinationsmöglichkeiten.

Das ist schon deutlich übersichtlicher..
Jetzt wäre es schön, **jedem möglichen binären Zustand** in einem **4 Bit Segment** eine **eigene** eindeutige **Bezeichnung** geben zu können.

Unser **10er** Zahlensystem ist da nicht ideal.
Ein **16er** Zahlensystem dagegen würde **jede Kombinationsmöglichkeit** in einem **4 Bit Segment** abbilden.

….doch dazu später mehr….!!

Das Mikrocontroller Innenleben

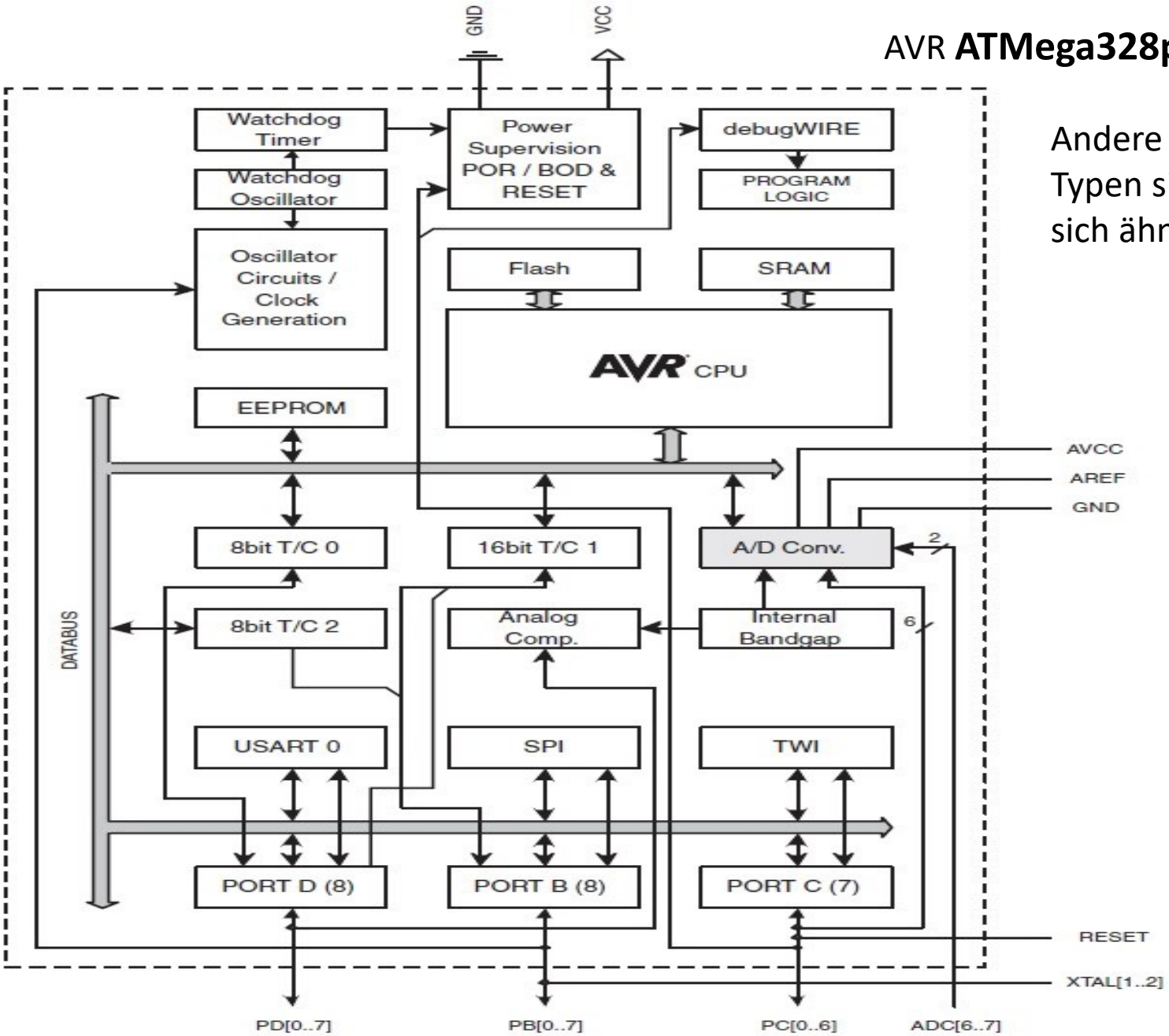
Steuerung des Mikrocontrollers

mit Hilfe der

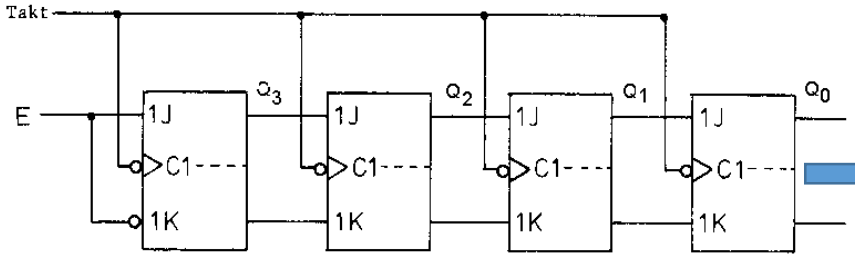
Special Function Register - SFR

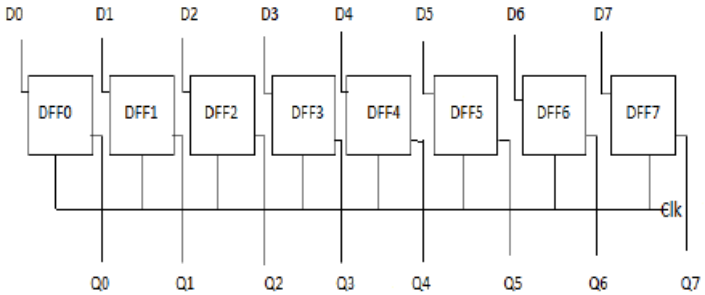
AVR **ATMega328p**

Andere µC Typen sind sich ähnlich.

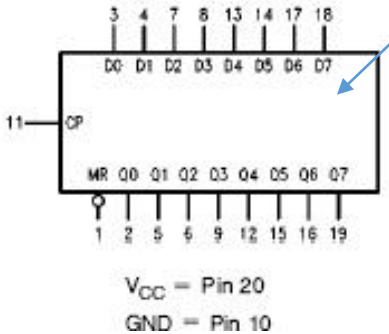


TTL/CMOS Logikgatter versus µKontroller *Special Function* **Register**



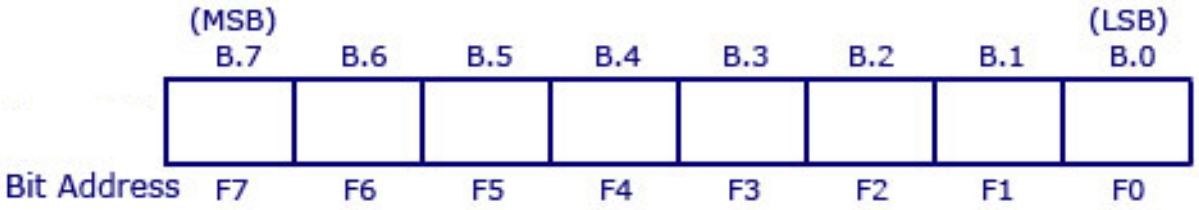


Logikgatter
Jonson Kelly oder D-FlipFlop Shift Register
oder 8-Bit Store Register TTL 74273



Ein **µKontroller Register** repräsentiert die Fähigkeit diskreter Logikgatter. Die Software bestimmt die Funktion

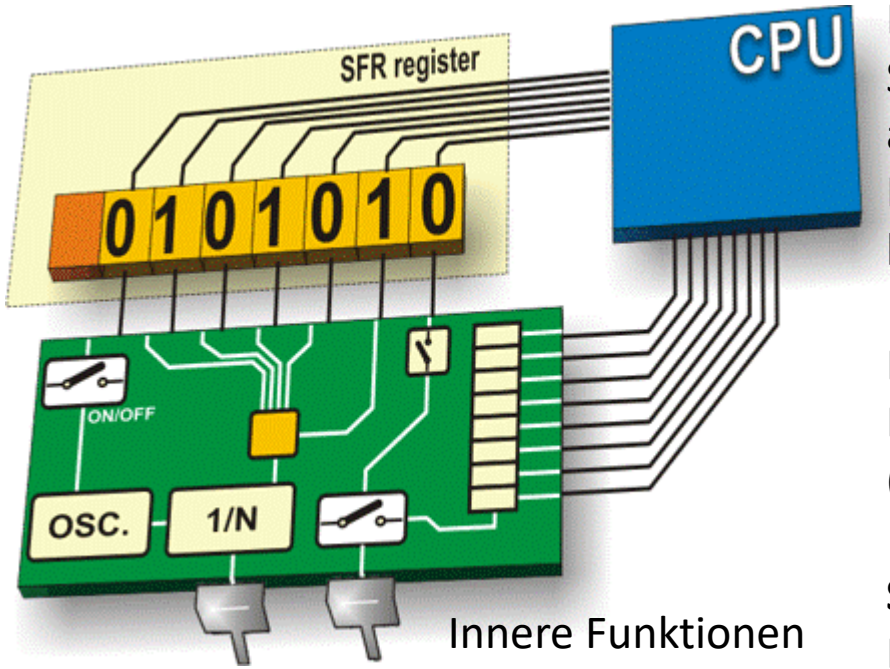
Register B



Register sind das Tor zum Mikrocontroller

Register sind ganz einfach „Positionen mit Namen“ im Mikrocontroller, die eine besondere Funktion mit Auswirkungen auf das innere Geschehen haben.

Man nennt diese „**S**pecial **F**unction **R**egister“ → **SFR** genannt.



Innere Funktionen im µC

Die einzelnen Bits im SFR sind wie Schalter. Schalter **die innere Funktionen** ein oder ausschalten.
Diese Funktionen können **nur EIN Bit** betreffen oder **Gruppen von Bits.**

Eine „0“ = AUS oder passiv.
Eine „1“ = EIN oder aktiv.
(meistens jedenfalls)

SFRs gibt es für alle ‚eingebauten‘ Funktionen im µController.

SFRs sind die Steuerzentrale.

Special Funktion Register (SFR)

BYTE	Bit 7	BIT 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

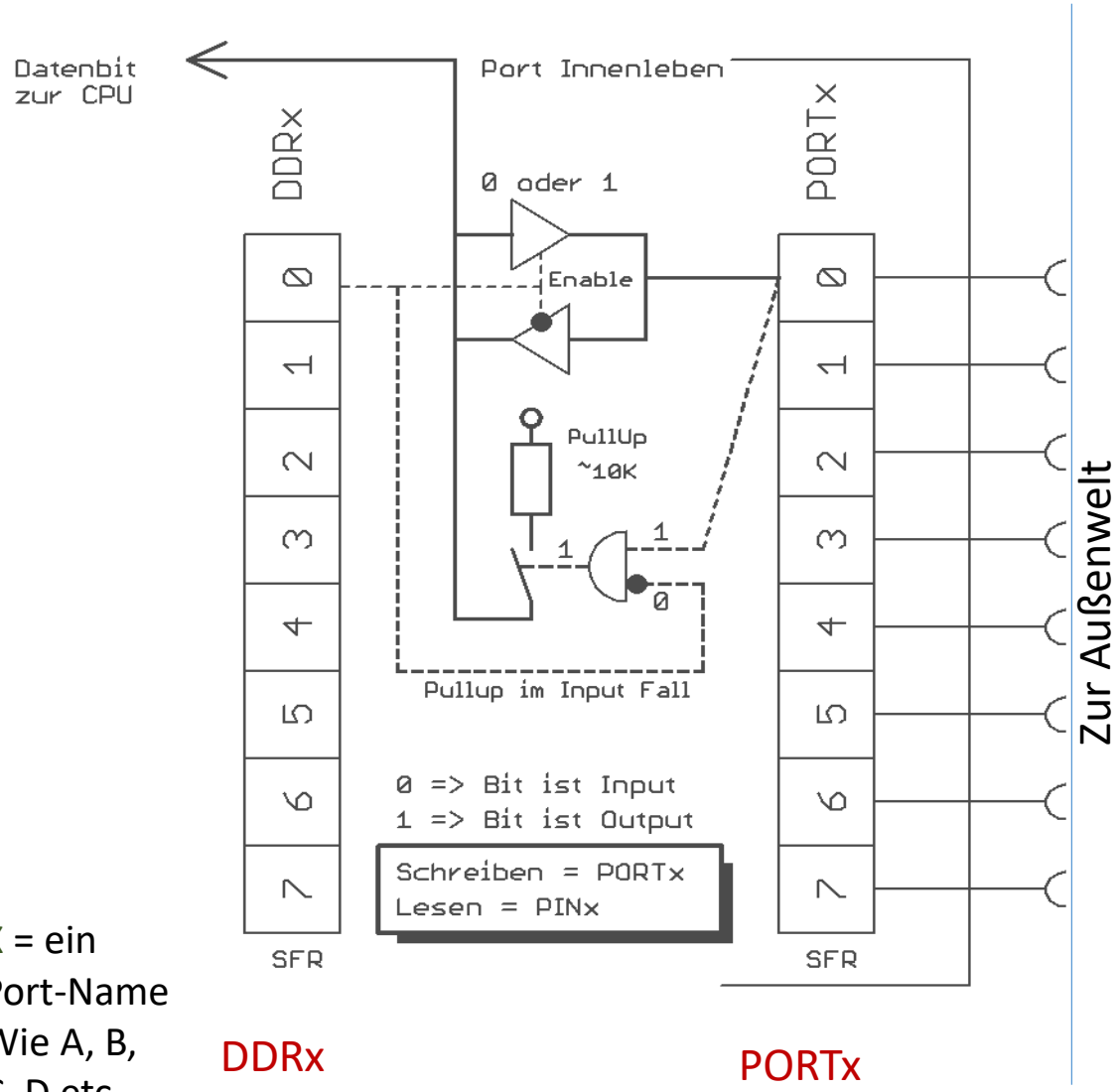
PINC – The Port C Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Die wichtigsten **SFR**, beispielsweise von **PORTC**, sind 8-Bit große Speicherbereiche, die wie Variablen geschrieben (man spricht „gesetzt“) oder gelesen werden können.
Das **DDR**x (x= B,C,D) ist das „**D**ata **D**irection **R**egister“.
Wie der Name schon sagt. Eine „**1**“ macht das zugehörige Bit des Portes „x“ zum Ausgang. Standardmäßig ist es „**0**“ und damit ein Eingang →(ist passiv).
BSP: DDRB = 0b00000001; → das Bit-0 wird auf „1“ gesetzt→ Bit-0 = Ausgang.

Die Kommando Leistelle

Wie arbeiten „**Special Function Register**“ (**SFR**) zusammen?



X = ein Port-Name Wie A, B, C, D etc.

DDRx

PORTx

Die Portfunktion (*Input oder Output*) wird vom DDRx Register bestimmt.

Zudem wird im dem Fall: „Bit ist Input“ das Portx Register benutzt um einen internen **PullUp** Widerstand zu aktivieren.

Merke:

Datenrichtung → DDRx

Geschrieben →PORTx

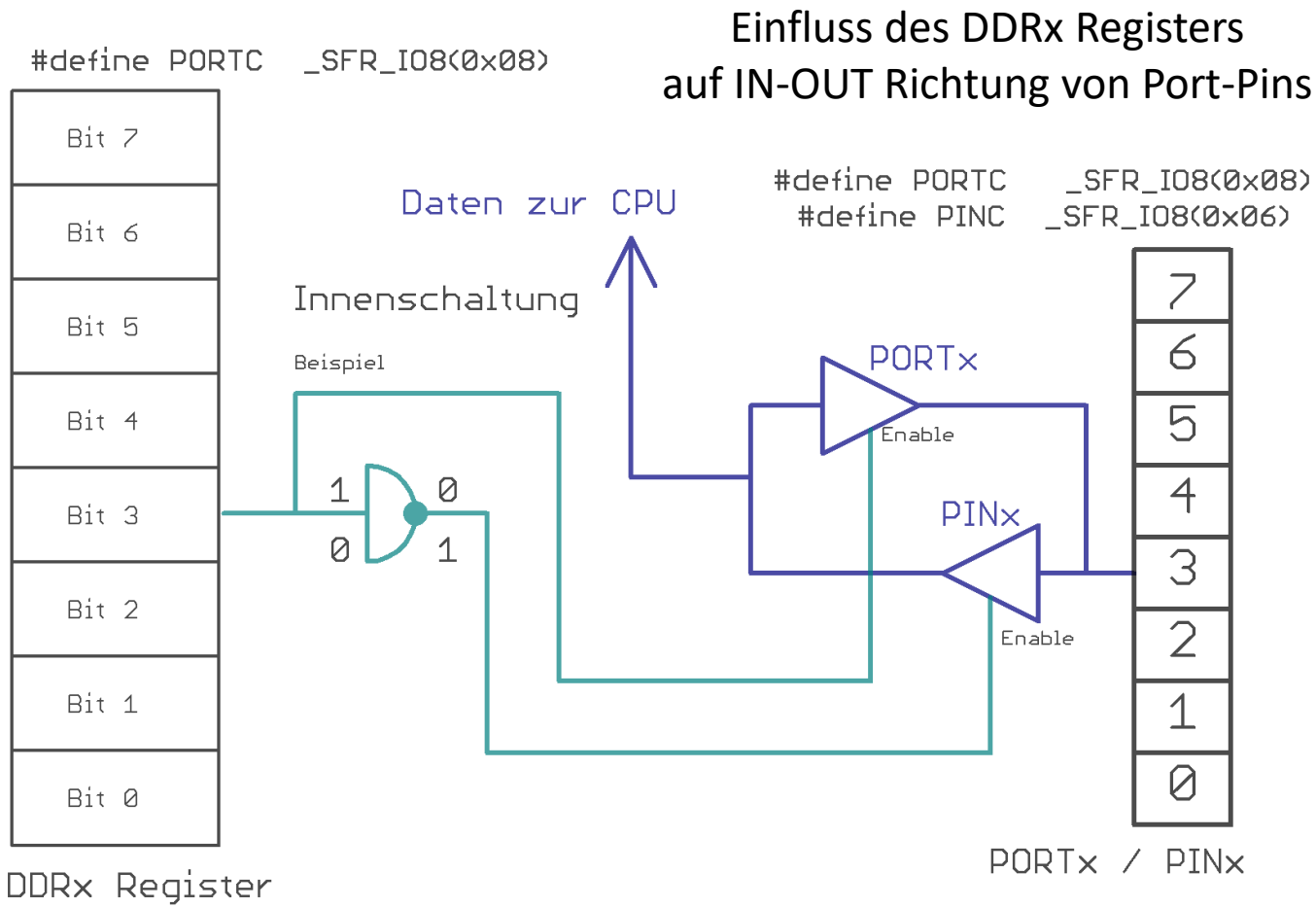
Gelesen → PINx

Einfluss des DDRx Registers auf IN-OUT Richtung von Port-Pins mit Special Function Register - **SFR**

Wie schon erwähnt, kann mit Hilfe der den PORT-Pins zugeordneten SFRs jedes einzelne Bit für sich getrennt konfiguriert werden.

Auch wenn ein Port nur 6 Bit hat, ist ein SFR-Byte immer 8 Bit.

DDRx:
Bit = 0 Input
Bit = 1 Output



SFR Auszug aus der originalen „iom328.h“ Datei

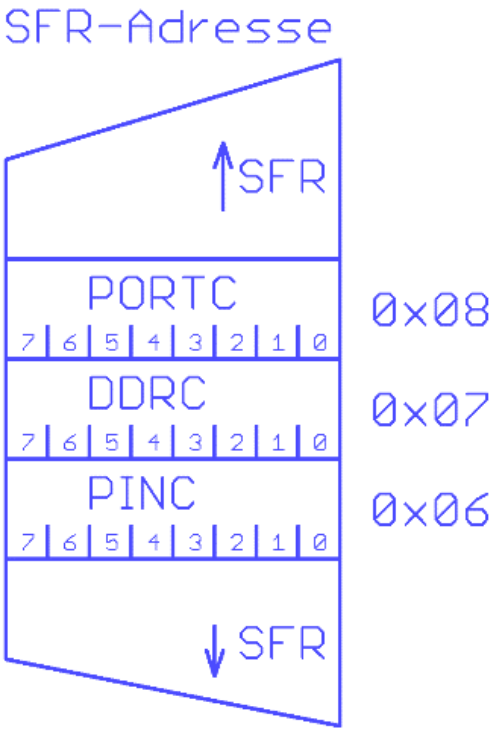
```
#define PINC    _SFR_IO8(0x06)
#define PINC0 0
#define PINC1 1
#define PINC2 2
#define PINC3 3
#define PINC4 4
#define PINC5 5
#define PINC6 6
```

```
#define DDRC    _SFR_IO8(0x07)
#define DDC0 0
#define DDC1 1
#define DDC2 2
#define DDC3 3
#define DDC4 4
#define DDC5 5
#define DDC6 6
```

```
#define PORTC    _SFR_IO8(0x08)
#define PORTC0 0
#define PORTC1 1
#define PORTC2 2
#define PORTC3 3
#define PORTC4 4
#define PORTC5 5
#define PORTC6 6
```

Im **µC** spezifischen Header-File iom328.h sind **ALLE SFR Register** und die **darin enthalten Funktionsbits** definiert.
Diese Register sind, durch die µC Architektur festgelegte**,** bestimmte Adressen.

Speicherverteilung



In io*.h vordefinierte Bit-Namen für I/O-Register und andere SFR

Bit-Namen bzw. Nummern sind in den **io*.h Dateien** des verwendeten Kontrollers definiert

→ zu finden in „**C:\WinAVR-20100110\avr\include\avr/**“ → **iom328p.h**

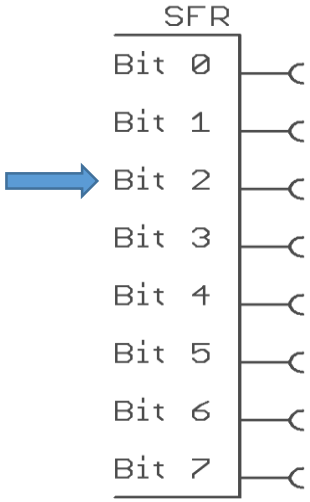
C:\WinAVR-20100110\avr\include\avr\iom328p.h

Man muss diese Definitionen nicht verwenden.

Für den Compiler sind z.B. die Ausdrücke (1<<PORTC7), (1<<DDRC7) und (1<<PINC7) identisch zu **(1 << 7)**

genauer: der Präprozessor ersetzt die Ausdrücke (1<<PC7)... zu (1<<7).

Wir sehen zur Verdeutlichung beispielsweise einen Ausschnitt der Definitionen für PORTC eines ATmega32 aus der **iom32.h** Datei (analog für alle weiteren Ports):




```
/* PORTC */
#define PORTC7   7
#define PORTC6   6
#define PORTC5   5
#define PORTC4   4
#define PORTC3   3
#define PORTC2   2
#define PORTC1   1
#define PORTC0   0
```

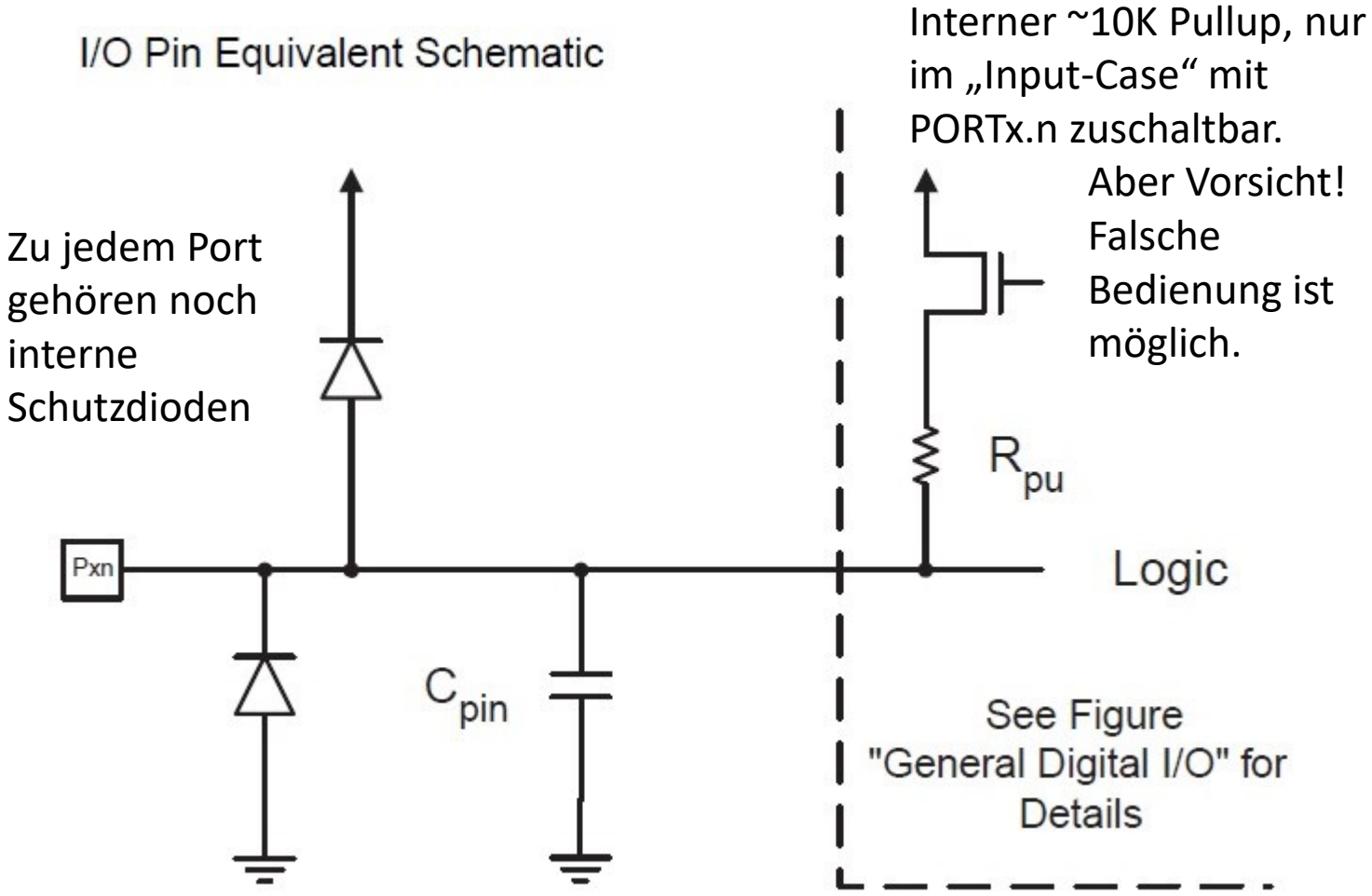
```
/* DDRC */
#define DDC7   7
#define DDC6   6
#define DDC5   5
#define DDC4   4
#define DDC3   3
#define DDC2   2
#define DDC1   1
#define DDC0   0
```

```
/* PINC */
#define PINC7   7
#define PINC6   6
#define PINC5   5
#define PINC4   4
#define PINC3   3
#define PINC2   2
#define PINC1   1
#define PINC0   0
```

0b00000100 = (1 << PORTC2);

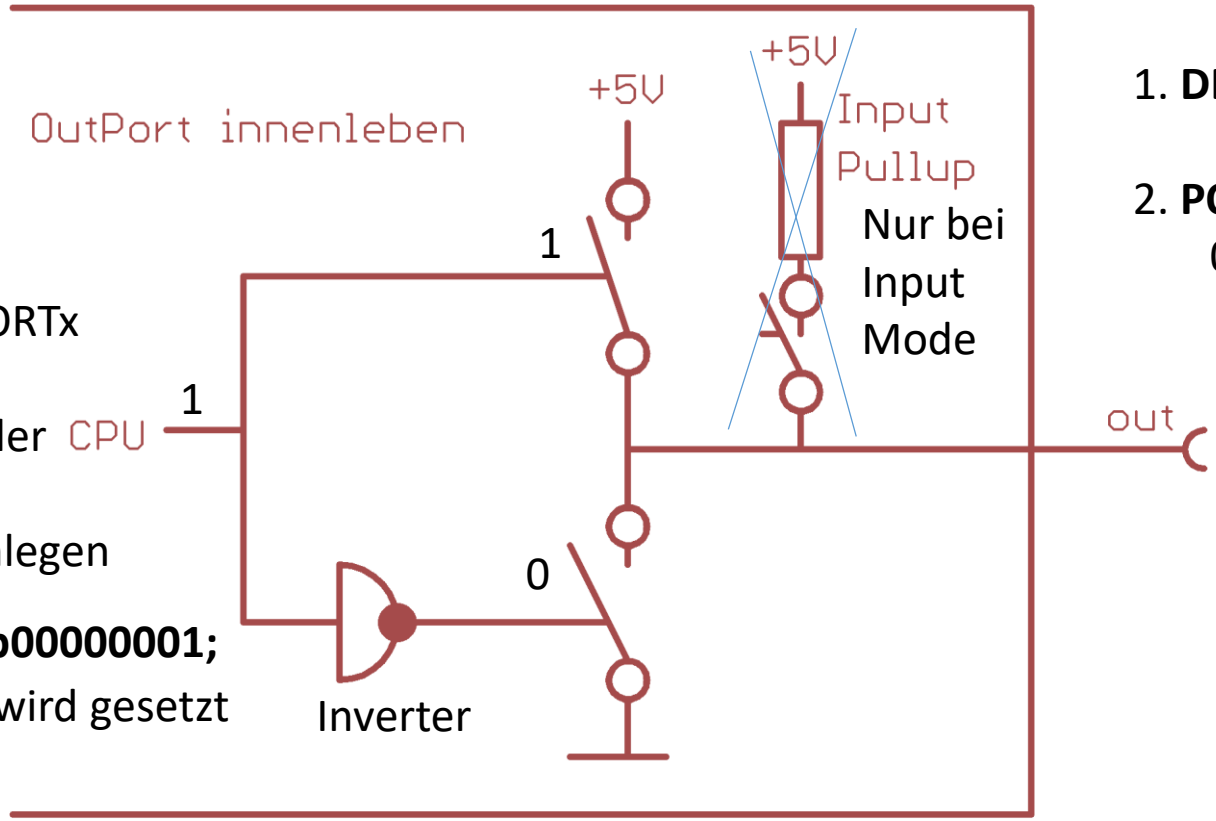
Anwendung der Bit-Namen

So sieht die Port Äquivalent Schaltung/Schematic in der „original Manual PDF“ des ATM328p aus



Schreiben eines Datenbit → PORTx [x=A,B,C,D…]

Reduzierte, aber reale geistige Vorstellung.
So sieht „EIN“ Bit aus, dass letztlich an einen Pin nach außen geht.
Vorrausetzung ist, dass das zugehörige **Daten-Direction-Register** Bit „**1**“ ist



1. **DDR**x (Bit)=1

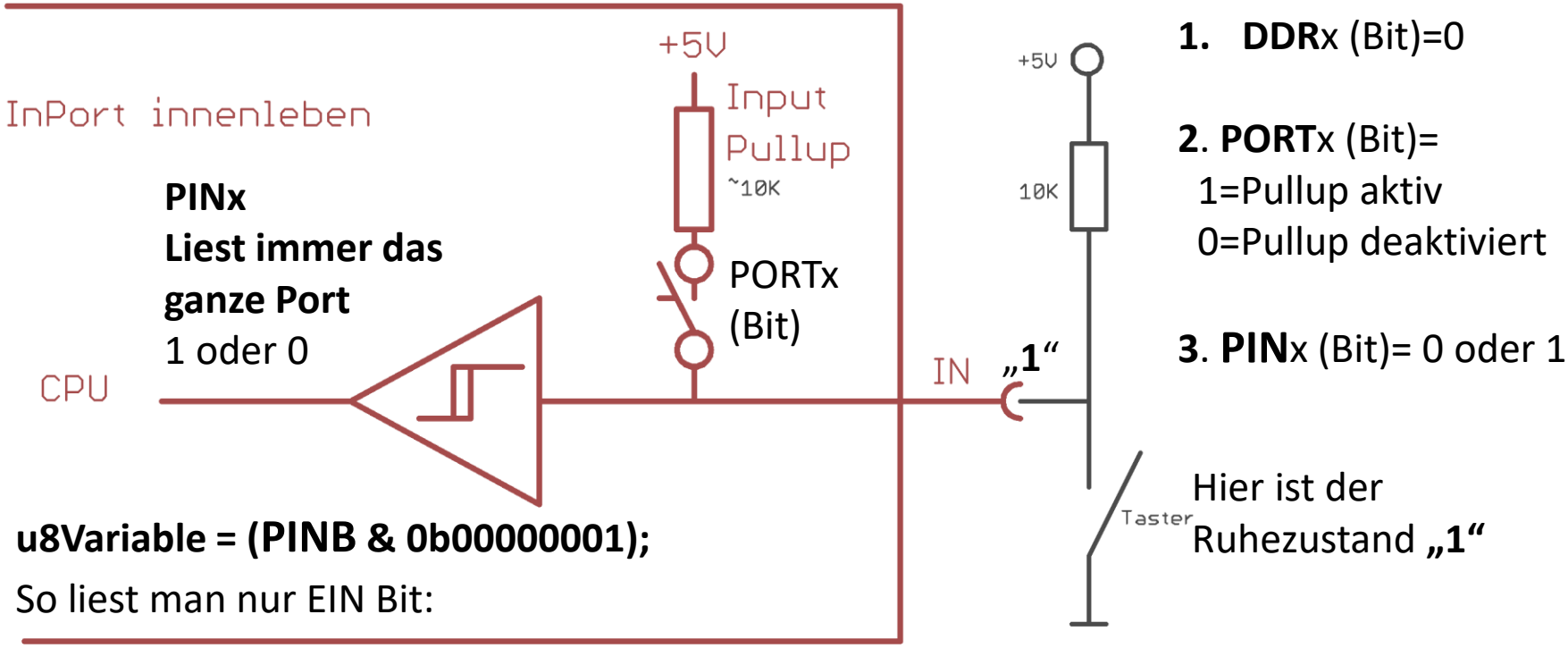
2. **PORT**x (Bit)= 0 oder 1

PORTB=0b00000001;
Das 0te Bit wird gesetzt

Eine „1“ am Schalter schließt diesen. Ergo, eine „0“ öffnet den Schalter.

Lese eines Datenbit → PINx (nicht PORTx !) [x=A,B,C,D...]

Reduzierte, aber reale geistige Vorstellung.
So sieht „EIN“ Bit aus, dass an einen Pin gelesen werden soll.
Vorrausetzung ist, dass das zugehörige **Daten-Direction-Register** Bit „**0**“ ist
+5V= Bit „1“ am µC Pin wird als logische 1 gelesen, 0V als logische 0

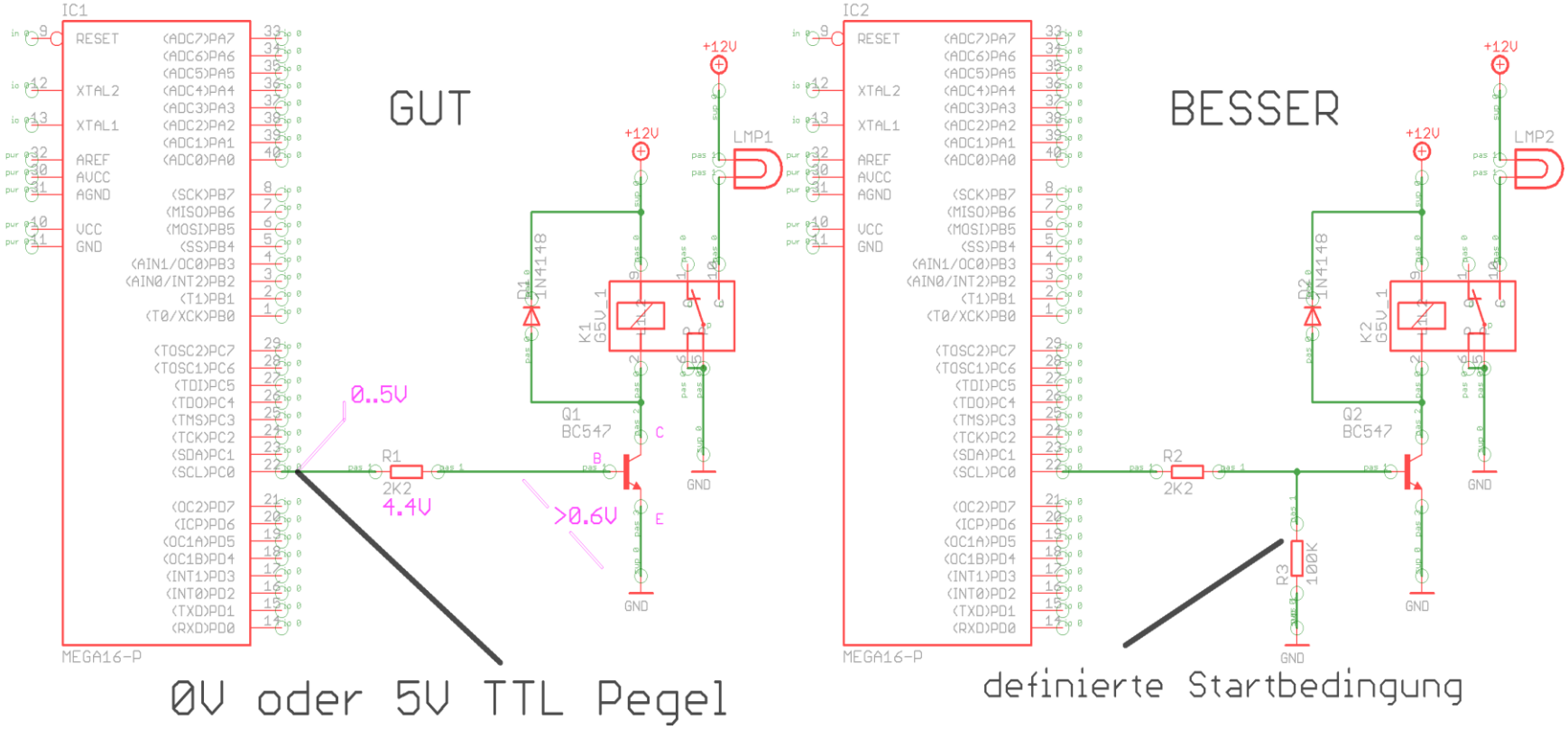


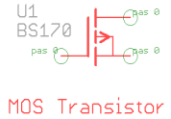
u8Variable = (PINB & 0b00000001);

So liest man nur EIN Bit:

Das Bit, dass PORTx.N definiert wird im INPUT Mode des Bits genutzt um den internen Pullup zu aktivieren

Gute bis sichere I-O Port Nutzung

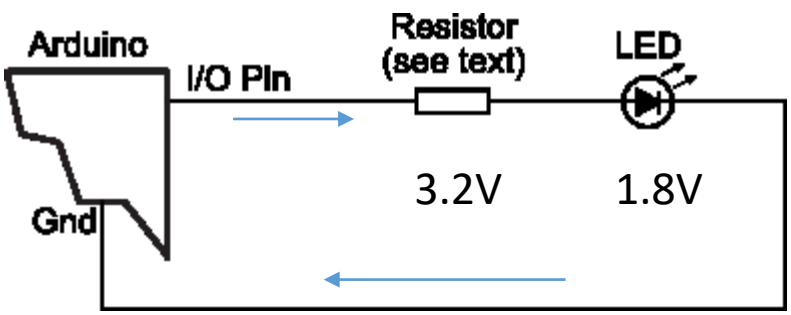




Ports immer mit **Strombegrenzung (Vorwiderstand)** betreiben.
Die Summe der Ströme im Auge behalten, und max. Strom/Port beachten. ~10mA
Rückspannungen abfangen.
Ports haben meist einen besseren „**sink-current**“ → Strom gegen Masse.
Startsituation nicht vergessen. (Flatternde Pegel/Relais im Startfall)

Das hatten wir schon, doch jetzt wissen schon etwas mehr…….

Bleiben wir erstmal dabei eine LED an einem Port-Bit zu betreiben



I = 5..10..max 20mA

Merke! Halbleiter niemals ohne berechneten Vorwiderstand an eine Spannung anlegen.

Strom durch die Diode
= ~5..10mA, max.20mA

→ 5V - 1.8V(Diffusionsspannung) =
3.2V Spannungsabfall am Widerstand

→ **R= U/I**
3.2V/0.01A = >320R = ~**330R**

~330R, oder **470**/560R sind also genau richtig.

Vorschau: Binär und die dezimale Entsprechung

BYTE	Bit 7	BIT 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Dez:	**128**	**64**	**32**	**16**	**8**	**4**	**2**	**1**

2^BitNr = Wert

Wandlung und Rechenmethode:

Binär → Dezimal, oder auch umgekehrt

Bsp. 01011101 = dezimal 93

0	**1**	**0**	**1**	**1**	**1**	**0**	**1**
0 +	64 +	0 +	16 +	8 +	4 +	0 +	1 = 93
Dez: 128	**64**	**32**	**16**	**8**	**4**	**2**	**1**

Wenn an der binären Bitposition „**1**“ dann addiere zur Bitposition gehörende dezimale Zahl, sonst addiere „**0**“ .

Single-Bit Dezimalwerte

1 = 0000 000**1**
2 = 0000 00**1**0
4 = 0000 0**1**00
8 = 0000 **1**000
16 = 000**1** 0000
32 = 00**1**0 0000
64 = 0**1**00 0000
128 = **1**000 0000

Single „1“ Bitpositionen ergeben diese Dezimalzahlen

Hexadezimale Zahlen

16 Kombinationen
von „0“ und „1“

Wir zählen nur bis 10, genauer 0..9. (weil wir 10 Finger haben)
Dann kommt der Übertrag → 10, 11, 12, 13, 14, 15..
Unpraktisch und unübersichtlich für ein Binärsystem

Zählen wir mal binär.
Da ist zuerst die 0, und dann kommt die 1.
Und schon ist der erste Übertrag: → 10
Dann 11, schon wieder ein Übertrag → 100
Dann 101, 110, 111, schon wieder ein Übertrag → 1000
Dann 1001, 1010, 1011, schon wieder ein Übertrag → 1100
usw..

UMSTÄNDLICH…und schlecht lesbar. Merkbar schon gar nicht..

Deshalb bedient man sich **eines anderen Zahlensystems**, das es ermöglicht, ALLE Kombinationsmöglichkeiten von 0en und 1en in 4 Bit darzustellen.
4 Bit → **16 Kombinationen** von **0**en und **1**sen
Diese Kombinationen nummerieren wir von 0..15 → 16 mal.

Bin	Hex -Dezimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	0xA -10
1011	0xB -11
1100	0xC -12
1101	0xD -13
1110	0xE -14
1111	0xF -15

Hexadezimal von 0..15 zählen

Wir erweitern unsere Zahlen
0..9 mit ergänzenden Buchstaben
A..F und erhalten:

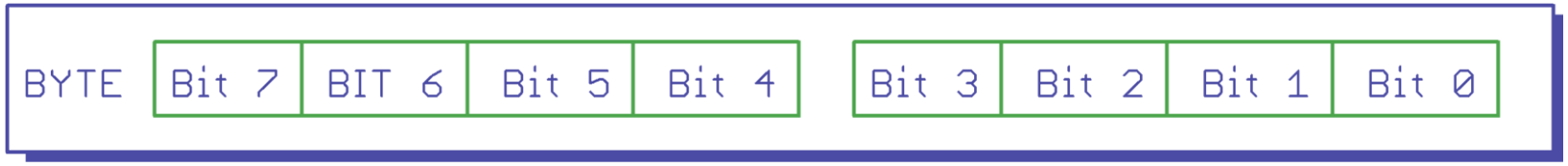
Dezimal: **0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15** = 16 Stellen
Hex: **0 1 2 3 4 5 6 7 8 9 A B C D E F** = 16 Stellen

Die Binärtabelle rechts beschreibt wie binär von rechts nach links gezählt wird. Nach jeder „1“ kommt der Übertrag nach links

00
01
10
11
100 usw…
101 110 111 1000

Bin	Hex -Dezimal
0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**0xA -10**
1011	**0xB -11**
1100	**0xC -12**
1101	**0xD -13**
1110	**0xE -14**
1111	**0xF -15**

Ab jetzt kann jeder **4 Bit 0..1er Kombination** eine Zahl von **0..F** zugeordnet werden.
Jetzt ist es etwas einfacher 4 Binärstellen einen eindeutigen Namen zu geben.



0xFF = F→11110000 + F→00001111

Um eleganter mit Binärzahlen zu arbeiten nutzt man den fälschlich sogenannten ***Hexadezimalcode***

Wir schreiben ein „0x“ davor !

Bei Oktalen Zahlensystem (0..7) eine „0“

Dieser beschreibt **4 Bit**.

0000…1111 = d0..d15 oder 0x0..0xF

Ein Byte hat jedoch 8 Bit !
Wir können zwei Hexadezimalzahlen zusammenschreiben.

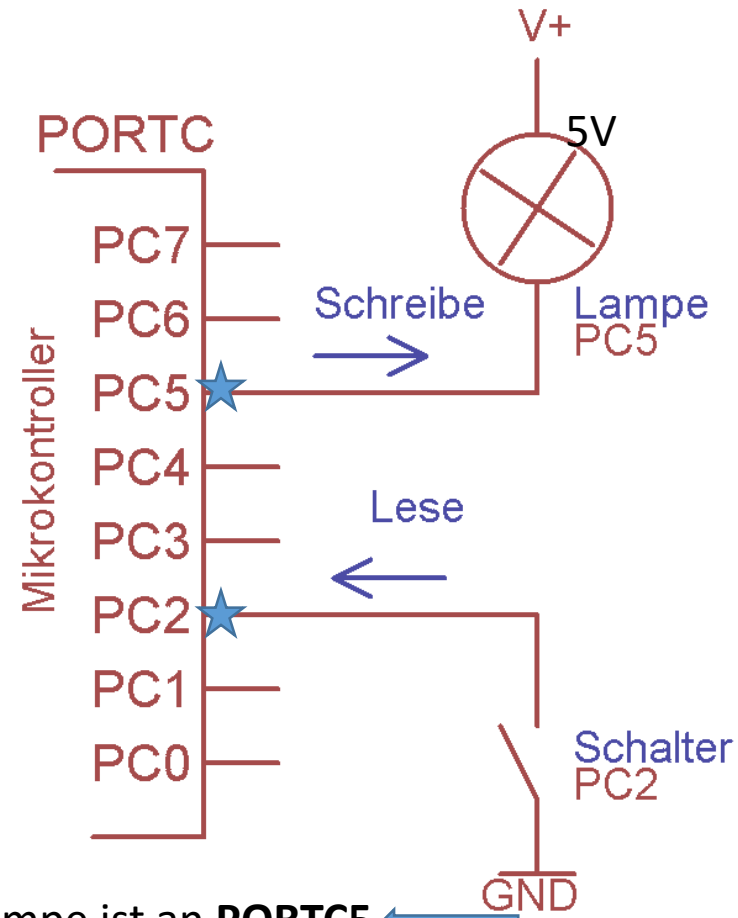
Bsp.

Binär		Hexadezimal
1011 0010	ist	**0x**B2
0110 0100	ist	**0x**64
0000 1000	ist	0x08

Bin	**Hex -Dezimal**
0000	**0**
0001	**1**
0010	**2**
0011	**3**
0100	**4**
0101	**5**
0110	**6**
0111	**7**
1000	**8**
1001	**9**
1010	**0xA -10**
1011	**0xB -11**
1100	**0xC -12**
1101	**0xD -13**
1110	**0xE -14**
1111	**0xF -15**

Das Schreiben und das Lesen,
ist nie mein Fach gewesen.....
(aus einer Oper)

Gemischte Port-Nutzung



Das Ziel…
(fast) Jeder ‚Draht' des Mikrocontrollers, somit der Elektronik, soll frei und exakt gewählt und angesprochen werden können.

Also benötigen wir Techniken, wie wir gezielt elektrische Signale **lesen** und **schreiben**.
Als Ergebnis bekommt man „1"en und „0"en die interpretiert werden wollen.

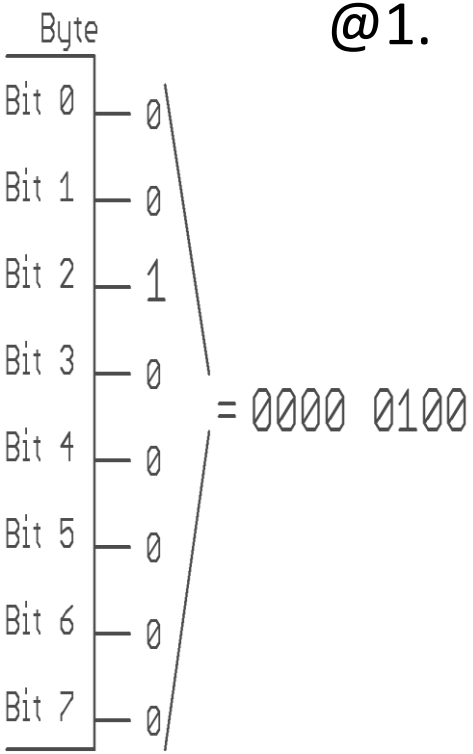
Dazu kommt, dass diese Signale meist aus einer Gruppe vieler anderer Leitungen (Bits) rausgefischt werden müssen.

ALLE Signale sind namentlich definiert in der Datei "**iom386.h**" Um diese Namen geht es.

Lampe ist an **PORTC5**
Schalter ist an **PINC2**

genauer, dort heißt es jedoch **PORTC5 und PINC2**
Dazu später mehr

Methoden der Zuweisung



Bsp.

PORTx = Value;

@1. Die **direkte Zuweisung** ist sehr effektiv, weil **alle** Bits gleichzeitig gesetzt werden, hat aber genau deshalb den Nachteil dass **alle** Bits beeinflusst werden. Auch die, bei denen *keine Änderung* stattfinden soll.

Wir benötigen also dringend eine Möglichkeit, einzelne Bits oder Gruppen von Bits isoliert zu beeinflussen.
Sprich, einzelne Bits in einem Byte zu setzen oder zu löschen.

Hier helfen die Gesetze der Logik weiter.

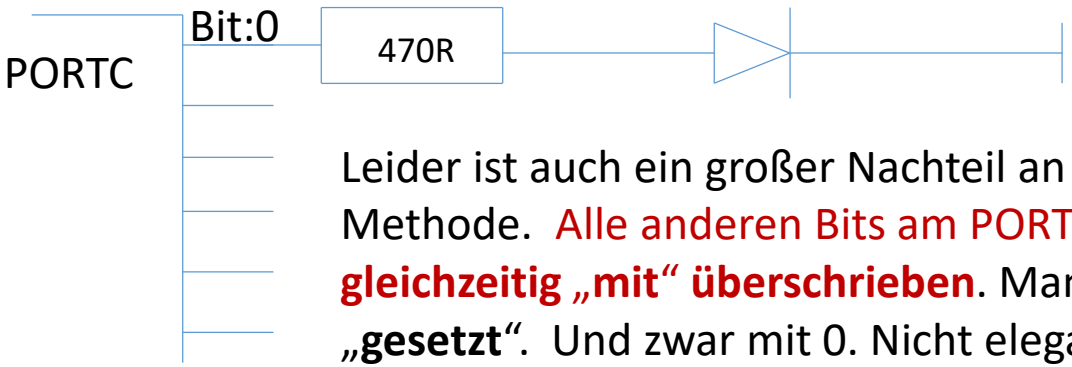
…dies sehen wir später genauer an…

banal: Die „Direkte Zuweisung“ =

An einem Mikrocontroller ist ein Port namens **C**, sagen wir **PORTB**, welches **8 Bit** hat.
Die Bits heißen **7..0**
Die LED liegt ja an Bit 0, was ja als 0b 0000 000**1** binär, oder **1** dezimal, darstellbar ist.

PORTB = 1; dann wird genau nur dieses Bit am PORTC auf „1“ gesetzt.
Die LED leuchtet.

Wenn wir sagen **PORTB = 0;** dann geht die LED aus.



Leider ist auch ein großer Nachteil an dieser Methode. Alle anderen Bits am PORT**B** werden **gleichzeitig „mit“ überschrieben**. Man sagt „**gesetzt**“. Und zwar mit 0. Nicht elegant aber effektiv. Dies kann man aber auch ausnutzen.

Siehe Tabelle:

2^n

1	= 0000 000**1**
2	= 0000 00**1**0
4	= 0000 0**1**00
8	= 0000 **1**000
16	= 000**1** 0000
32	= 00**1**0 0000
64	= 0**1**00 0000
128	= **1**000 0000

Dezimal ← → Binär

kleines Programm:

Die LED ist an ein Bit eines x-beliebigen Ports angeschlossen.

Dazu müssen wir zuerst entscheiden **ob** und **welches Bit** an **welchen Port** zu Verfügung steht.

Sagen wir, wir wollen eine LED an **PORTC**, **Bit 0** betreiben → **PC0**

Dieses **Bit an PORTC** muss also ein **Ausgang** sein.

(wie ?)

Zuerst wird es notwendig, zu garantieren dass dieses Bit ein Ausgang und kein Eingang ist.

(Default = 0 = Input)

Dann sollte man festlegen, auf welchem Startpotential dieses Bit nun ist („0“ oder „1“)

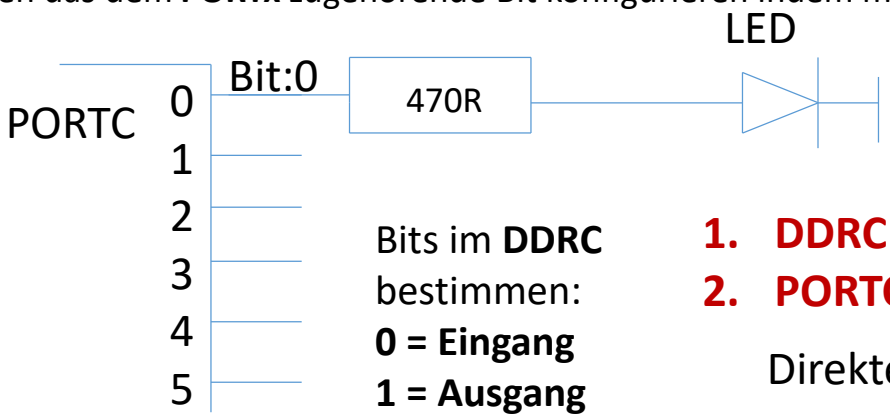
(Default = 0)

Schön wäre es, wenn es einen gut zu merkenden Platz gäbe, wo man das einfach „festlegen“ oder „initialisieren“ kann. Am besten mit *sprechendem Namen* wie

(**Signal-Richtungs-Einstellungs-Schalter** für **PORTC**) → **D**ata-**D**irection-**R**egister-**C**

könnte es heißen und die Abkürzung „**DDRC** “ tragen.

Wir können das dem **PORTx** zugehörende Bit konfigurieren indem man sagt:



Bits im **DDRC** bestimmen:
0 = Eingang
1 = Ausgang

1. **DDRC = 0b0000 0001;**
2. **PORTC = 0b0000 0001;**

Direkte Zuweisung

LED an PORTC Bit 0 →

C-Compiler starten → WINAVR.

In den Projektordner mit minimalen Projektdateien wechseln:

main.c
main.h
make
mymain.pnproj

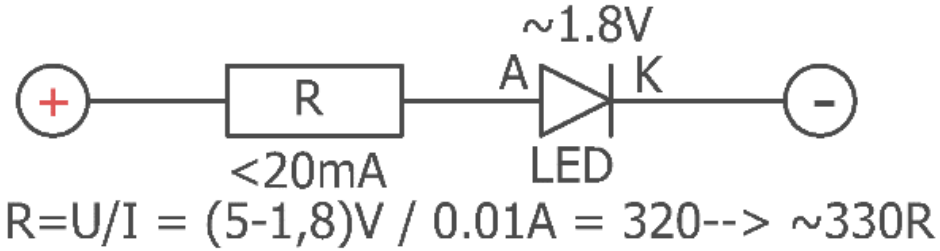
Projekt mit WINAVR öffnen: → „mymain.pnproj“

DDRC Register LED-Bit auf 1 = Output;

```
void main(void) →
{
DDRC = 1;
while(1)
        {
        PORTC = 1;
        _delay_ms(500);
        PORTC = 0;
        _delay_ms(500);
         };
};
```

1 = 0000 0001
2 = 0000 0010
4 = 0000 0100
8 = 0000 1000
16 = 0001 0000
32 = 0010 0000
64 = 0100 0000
128 = 1000 0000

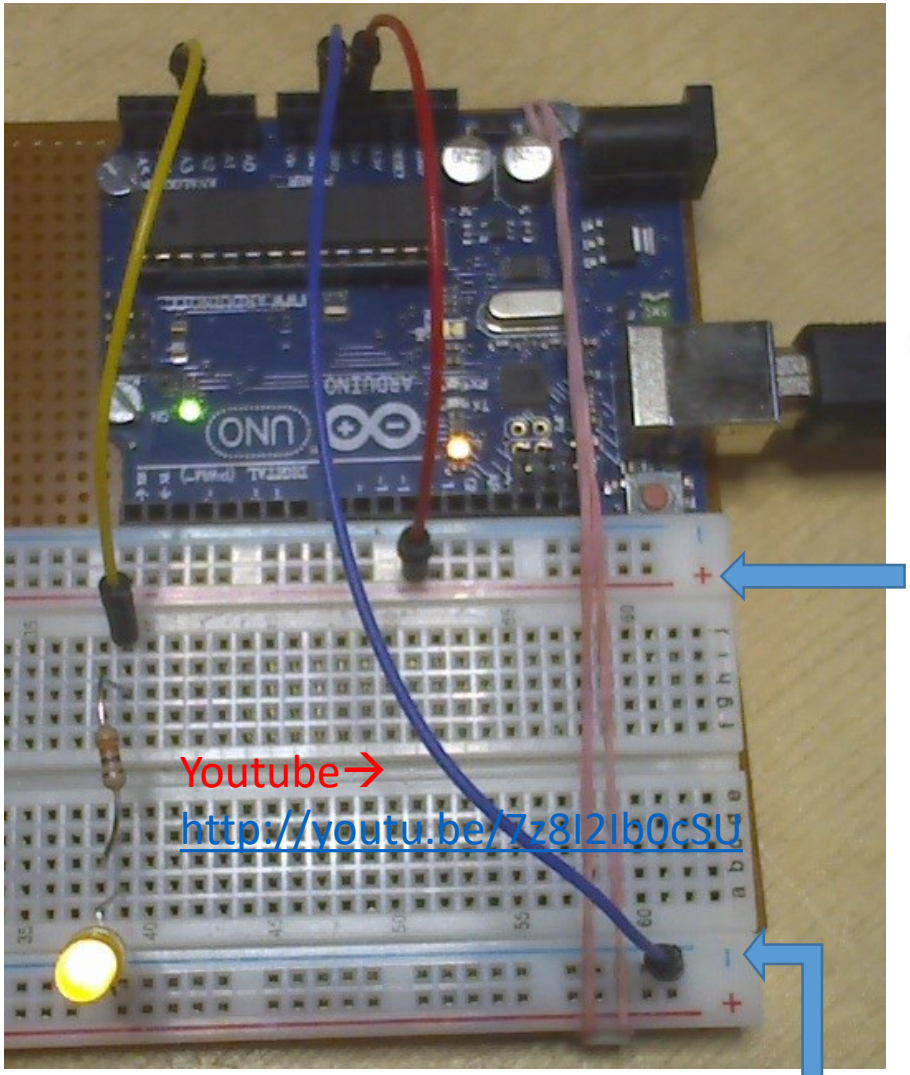
Suche das 0.te Bit von PORTC.
Power GND-(Schwarz) an Steckbrett (-),
+5V (ROT) an Steckbrett (+),
Von „A0“ = PC0 zum Widerstand



Aufbau mit Steckbrett:
PINC.0 → ~470..560R → Anode LED →Kathode GND.

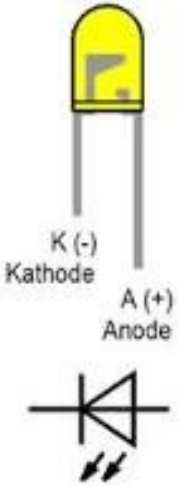
Die selbe Situation wie vorher schon…:

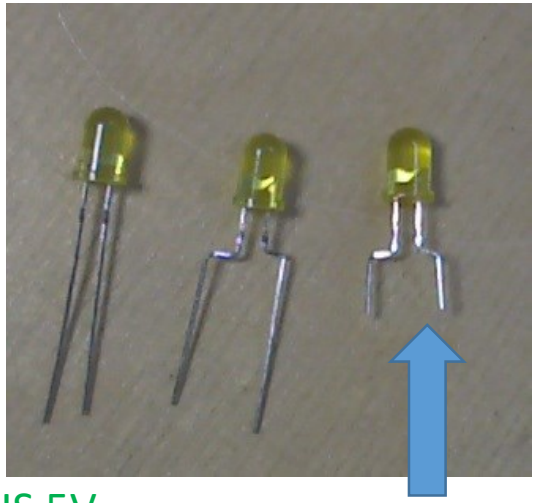
Eine LED an PORTC, Bit 2



Unten = MINUS/GND

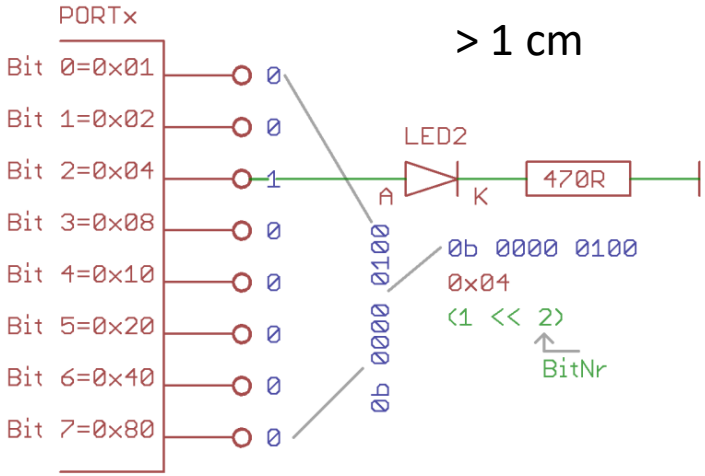
Die LED Drähte niemals wie die Pilze im Wald biegen !!
Zange benutzen.
Vorher hinhalten und abmessen.





Oben = PLUS 5V

Nicht zu kurz !:
> 1 cm



C-Programm, das eine LED an PORTC → Bit ‚2' ein- und ausschaltet.
→ „2A1-ATM328-Blink_1Hz _PORTC.2"

```
#include <avr/io.h>
#include <util/delay.h>

// ******************
int main(void)
// ******************
{                          // Portrichtung bestimmen → OUT
DDRC = 0b0000100;          // binäre Schreibeweise
while( 1 )  // endless loop
            {
            PORTC = 0b00000100;      //{oder 4 , oder 0x04}
              _delay_ms( 250 );
            PORTC = 0;
              _delay_ms( 250 );
            };            //while

};          //main
// *** MAIN END ***
```

Test Programmgröße:
Nach dem Compilieren sieht man:
Size after: AVR Memory Usage

Device: atmega328p

Program: 176 bytes (0.5% Full)
(.text + .data + .bootloader)

Data: 0 bytes (0.0% Full)
(.data + .bss + .noinit)

Funktion:
Direkte Zuweisung → DDRC = 4; //Bit-PC2 =Out
Loop:
Bit2=4;
warte ein Weilchen
Bit2=0;
warte ein Weilchen

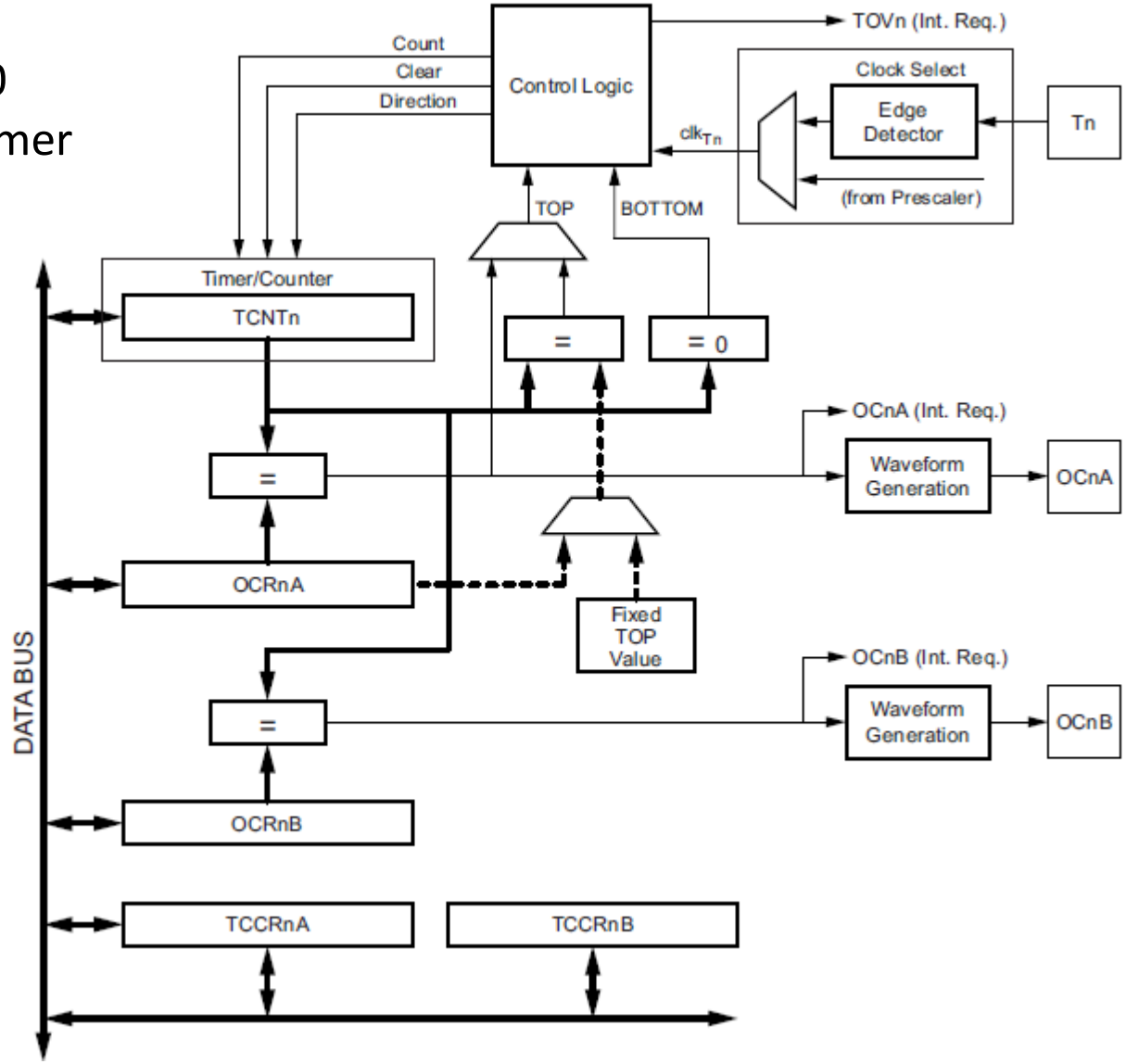
Spickzettel:

1 = 0000 0001
2 = 0000 0010
4 = 0000 0100
8 = 0000 1000
16 = 0001 0000
32 = 0010 0000
64 = 0100 0000
128 = 1000 0000

Wir basteln eine sehr genaue 1 Sekunden Uhr → TICK Counter

Timer 0
8 Bit Timer

Figure 14-1. 8-bit Timer/Counter Block Diagram



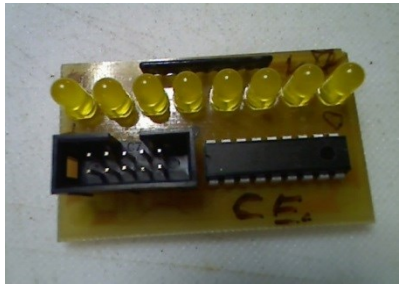
Wir basteln eine sehr genaue 1 Sekunden Uhr → TICK Counter

Quarz= 16MHz → /256 (Vorteiler) = 62500 → „/250 = 250“
250/250 → alle 250 * ist dann genau eine Sekunde.

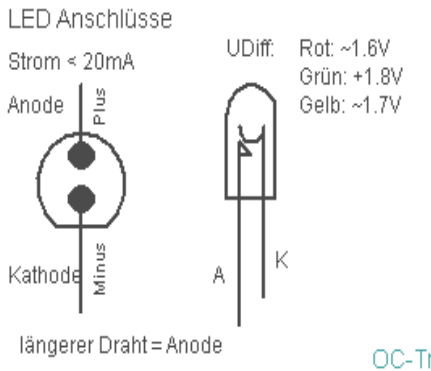
```
// Timer - TickCounter starten
TCNT0 = 0;
OCR0A = 250;                    //Compare Value
TCCR0A = (1<< WGM01) ;          //CTC Mode = Compare Timer Counter mit OCR0A
TIMSK0 = (1<< OCIE0A);          //compare Interrupt aktivieren
TCCR0B = (1<< CS02);            //Vorteiler /256
sei();
```

```
ISR( TIMER0_COMPA_vect )        //250 */sec
{
gu32_Ticks++;
if( !(gu32_Ticks % 250) )
        {
        DDRB  |= (1<<5);        //Bit 5 von Port B = Output
        PORTB ^=  (1<<5) ;      // XOR=   toggle Interne Led
        };
};
```

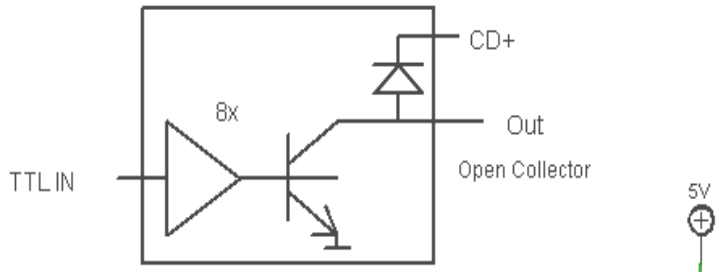
8 Bit LED Leiste, µC gerecht ausgeführt

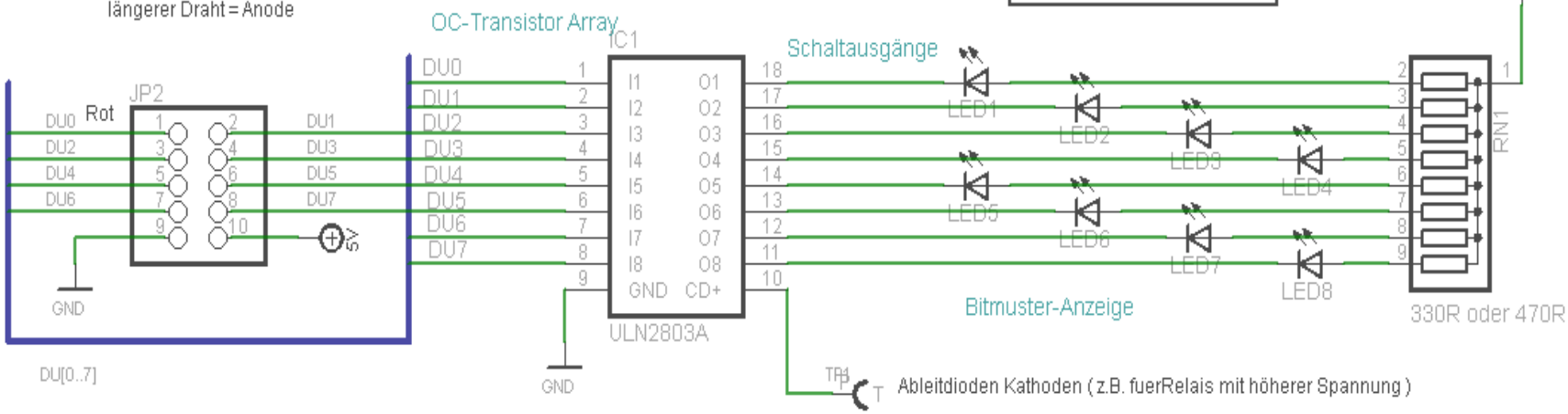


Da **µC**ontroller Pins nur begrenzt belastbar sind <~10mA sollte in der Summe (8 LEDs = ~80mA) ein Treiber eingesetzt werden.
Ein guter OC (Open Collector) gegen GND (negativ) = **ULN2803**. Ein Positiv-Treiber = **UDN2981A** (teuer)



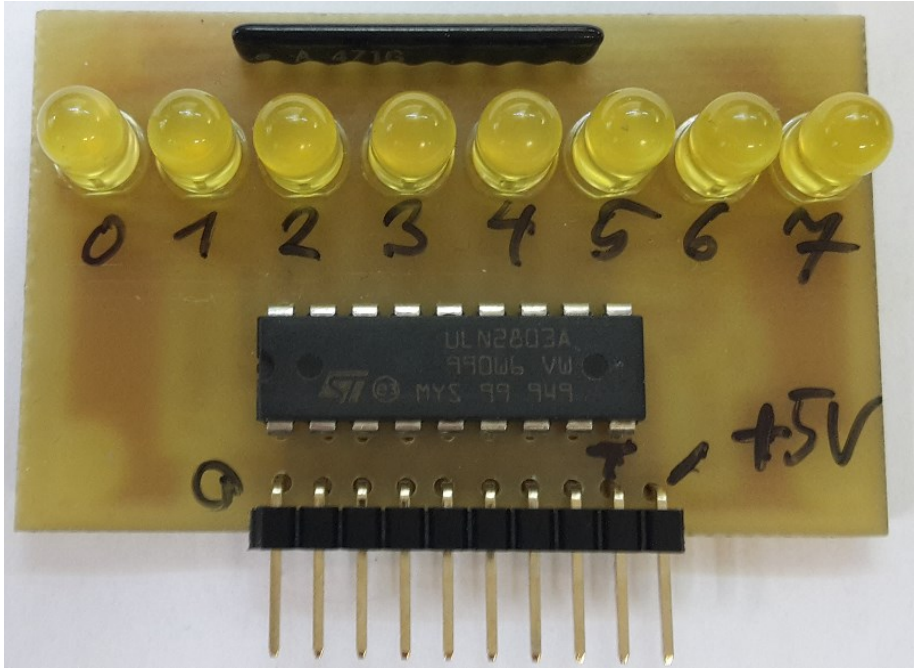
Im ULN2803 sind an allen Ausgängen Dioden in Sperrichtung.
Die Kathoden sind zusammengeführt und gemeinsam an CD+ verfügbar.
Damit können dann induktive Lasten (Relais) angeschlossen werden.



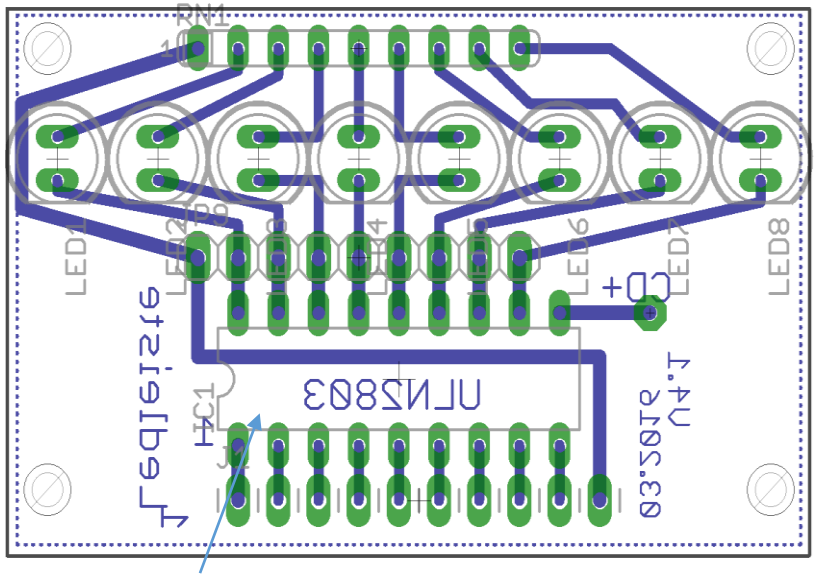


8+ Led-Leiste mit Steckbrett Stiften

Reales selbstgebautes Board



Eagle Board (wie Glasdurchsicht)



ULN2803 „open collector“ Array schaltet ~500mA gegen GND/Kanal.

Bit - 0 1 2 3 4 5 6 7 +5V

GND= 0V

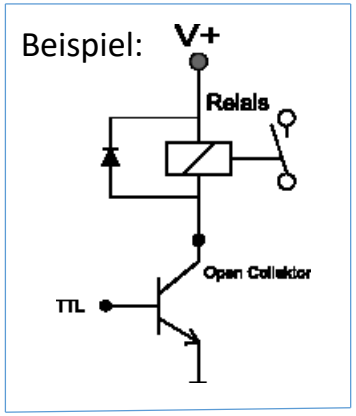
PLUS-MINUS NIEMALS verwechseln.
Zuerst GND anschließen.

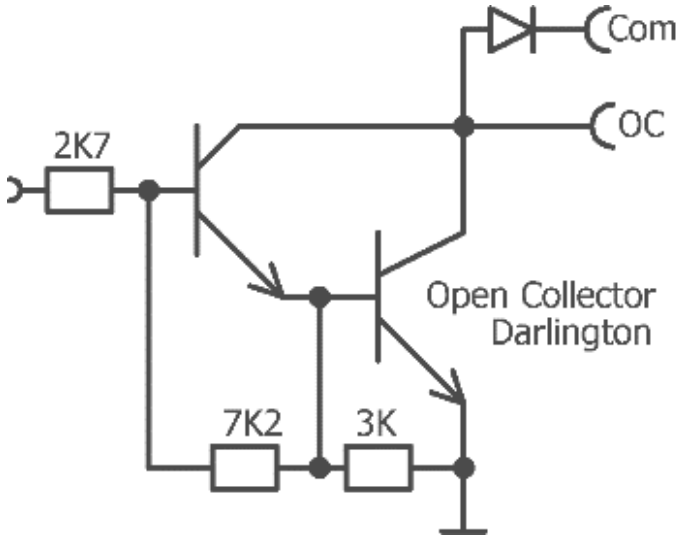
Die Bits 0..7 **„bastelt“** man sich, „wenn nicht genügend Leitungen/Port frei sind“, aus **freien Bits** zusammen und kombiniert so verschiedene Ports

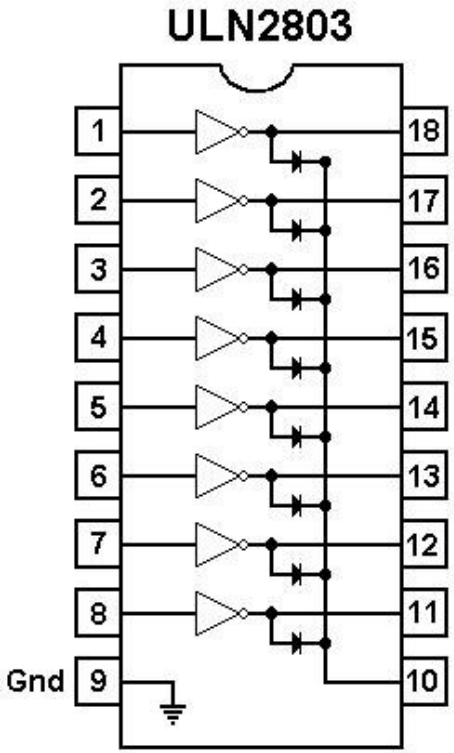
Ein beliebter **Transistor Array Baustein** **ULN2803**

ULN2803 Transistor Array IC
„Open Collector“ Hier **mit 8 Darlington Transistoren** und 8 **integrierten Freilauf-Dioden** an einem gemeinsamen **Common** Anschluss

Damit kann man 8 Bauteile, die
1.) hohen Strom fordern (<=500mA) und
2.) die an V+ liegen mit einem „s**chwachen**“ µC Steuersignal auf GND, und somit AN schalten.







LED-Leiste mit Shift Out Baustein 74xx595

SHIFT Out Baustein 74xx595

LED richtig herum einbauen

Erst "EINE" Seite anlöten, richtigen Sitz kontrollieren...

dann erst "ANDERE" Seite löten..

LED Anschlüsse

Strom < 20mA

Anode

Plus

Minus

Kathode

UDiff: Rot: ~1.6V
Grün: +1.8V
Gelb: ~1.7V

A K

längerer Draht = Anode

OC-Transistor Array

DS

SH_CP

ST_CP

Carry

JP2 JP1

5V

GND

Pins Nr.

IC1

SER QA QB QC QD QE QF QG QH

SCK SCL

RCK

G QH*

74HC595N

StromControlle
Schaltausgänge

LED1 LED2 LED3 LED4 LED5 LED6 LED7 LED8

Bitmuster-Anzeige

RN1

330R oder 470R

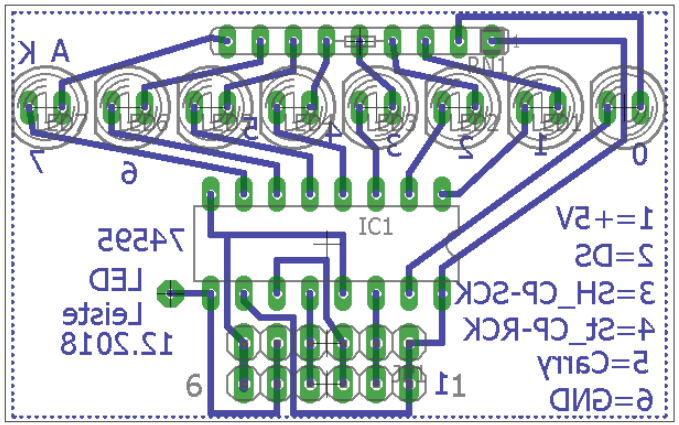
IC1P

VCC GND

TP1

1= Pin 1= Plus +5V
2= DS
3= SH_CP
4=ST_CP
5=Carry
6= Ground (Minus)

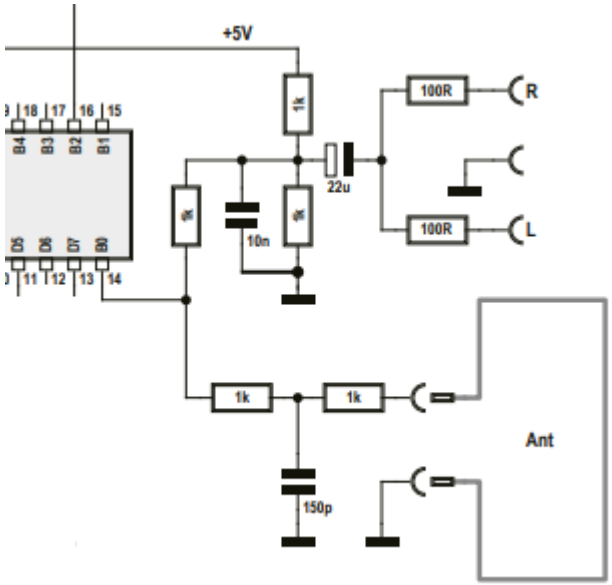
Sicht von oben



+5V
DS
SH_CP
ST_CP
Carry
Ground

PIN: 6-5-4-3-2-11

Portbit „wie ein“ Open Kollektor Transistor ansteuern



Vorteil: Der µC muss nicht den (+) Strom liefern

Auszug aus einem Artikel von Burkhardt Kainka

Man steuert nicht das Portregister an, sondern das Datenrichtungsregister, zum Beispiel für den Anschluss PB0. Mit PORTB.0 = 0 und DDRB.0 = 1 (Ausgang) hat man einen niederohmigen Low-Zustand.
Schaltet man dann DDRB.0 = 0 (normalerweise macht man das, wenn man digitale Signale einlesen will), wird der Port hochohmig.
Diese beiden Zustände entsprechen genau einem FET mit offenem Drain.
Wir können nun an PB0 eine beliebige „Drain“-Spannung anlegen
(wenn diese nur im Bereich 0 V bis 5 V bleibt).
Mit DDRB.0 = 1 wird diese auf Masse gezogen, etwa im Takt des HF-Signals.
Und natürlich kann diese Drain-Spannung sogar moduliert werden,
zum Beispiel mit einem NF-Signal.

Fund with Flags

Ein Flag ist ein einzelnes Zeichen, ein Lämpchen, minimal ein BIT. So kann man also **EINE**, wenn auch wichtige, **Informationseinheit** setzen, verteilen lesen, Zustände:

0 = aus

1 = an

VORAUSBLICK:

Mit Flags kann man hervorragend Ereignisse festhalten.

```
#define TIME_EVENT    0x02          //definiere „irgendein“ Bit
```

```
ISR( TIMER0_COMPA_vect )  //250 Hz  Interrupt.
{
gu32_Ticks++;   // allg. Zähler
if( !(gu32_Ticks % 250) )    //Modulo 250  →alle 250*=True
            {
            gu8ProgStatusFlags |= TIME_EVENT;    //Setze FLAG
            };
};
```

In der Main() Sektion

```
if( gu8ProgStatusFlags & TIME_EVENT )  //Lese Flag: TRUE ?
            {
            gu8ProgStatusFlags &= ~ TIME_EVENT;  // Lösche Flag
            DDRB  |= (1<<5);
            (PORTB ^=  (1<<5) ;// xor toggle Interne Led
            };
```

Die „C“ #define Direktive

Ein unerwartet „**mächtiges Werkzeug** in „**C**“ ist die **#define** Direktive.

Eigentlich nur eine ‚**Alias**‘ Namensvergabe.

....und ist genau genommen gar kein Element der Sprache „**C**“, sondern gehört zu den sogenannten **Präprozessor**en.

Damit sind Anweisungen an den Compiler (Übersetzter) gemeint. Diese Präprozessoren sind sehr wichtig und sehr gelungen und machen, intelligent eingesetzt, einen Teil der großen Leistungsfähigkeit von **C** aus.

#define LED_GELB 0x20 *//0x20 bedeutet → 0b 00**1**0 0000*

So kann man „**sprechend**“ irgendein **namenloses Bit** benennen.

Dieses „irgendwo-Bit“ heißt nun sprechend „**LED2_GRUEN**“.

Im Programm kann nun **LED_GELB** verwendet werden,

statt ***nichtssagender* Zahlen, u**nd… man kann keine Verwechslungsfehler mehr machen, wie Zahlendreher oder vergessene **0**en etc.

Zudem können wir alle **konstanten Zahlenwerte** zentral verwalten.

Das werden wir mit der Zeit noch vertiefter betrachten…

Beispiele: Inhalte von Headerfiles → main.h etc.

```
#define  FALSE      0
#define  TRUE       1
#define GREEN       1
#define RED         2
#define YELLOW      3
```

Durch die #define Methode wird aus „nebulösen Zahlen und Zeichen“ ein lesbarer, mit „Bedeutung“ dokumentierter Text.

Makro Definition:

//ich schreibe eine “**()**” dahinter, so sieht man den Funktionscharakter

```
#define RED_LED_ON()    (PORTD |= (1<< RED))
```

Hier ein „Standard Define Set“ für ein belibiges µC Portbit

```
#define     LED_PIN_NUMMER   5
#define     LED_BIT          (1 << LED_NUMMER)
#define     LED_DDRX         DDRD
#define     LED_PORTX        POTRD
#define     INIT_LED()       (LED_DDRX  |=  LED_BIT)
#define     SET_LED()        (POTRD  |=  LED_BIT)
#define     CLR_LED()        (POTRD  &= ~LED_BIT)
```

```
#define _BV(x)  (1<<(x))
```

Makro: …sprich „Bit Value. ->hier mit Parameterübergabe (x)

Beispiel einer **#define** Anwendung für Operatoren um diese verständlich bzw. lesbar zu machen, oder dies sich einfach so zu merken.
Aber mit etwas Übung ist dies einfach unnötig.

Bitoperatoren:

```
#define  _AND_BIN        &
#define  _OR_BIN         |
#define  _XOR_BIN        ^
```

Zuweisungen:

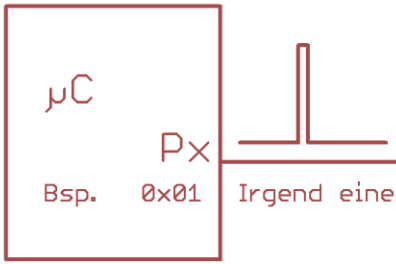
```
#define  _SETBIT         |=        //Oder= setzt Bits
#define  _CLEARBIT       &=~       // Und=~ löscht Bits mit Komplement
#define  _CLEARMASK      &=        // Und= löscht Bits OHNE Komplement
#define  _TOOGLEBIT      ^=   //XOR= TOGGLE  BITS. Wie ein Wechselschalter
```

Logische Abfrage:

```
#define  _AND_LOG        &&
#define  _OR_LOG         ||
#define  _UNGLEICH       !=
```

Debugging Trick-Techniken: Ein **Toggle Bit**

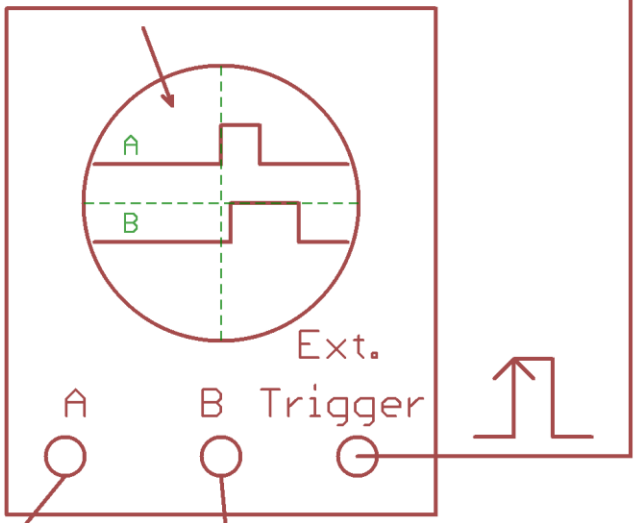
(Was passiert Wann-WO ?)



Trigger-Makro an die Codeposition
wo der Signalzustand interessant ist

Irgend einen freien PIN

```
#define TEST_DIRREG DDRC
#define TESTPORT    PORTC
#define TEST_BIT    0x01
TEST_DIRREG |= TEST_BIT; //0x01

void Toggle_Test(uint8_t ucCnt)
     {
     uint8_t ucNN;
     for(ucNN=0; ucNN < ucCnt; ucNN++)
        {
        TESTPORT |= TEST_BIT; // 0x01
        _delay_us(1);
        TESTPORT &= ~TEST_BIT; //0x01
        _delay_us(1);
        };
     };
```



Externer Trigger Eingang
Steigende Flanke
TTL-Pegel

Messpunkte

Messpunkte

DEBUGGING Hilfsfunktionen

```
//mit Oszilloscope Testpin am µController an geeigneter Stelle überwachen
// Zahl der Toggles gibt Position an.

#ifdef  TOGGLE_FLAG_USED
// ****************************
void Toggle_Bit(uint8_t u8Cnt)
// ****************************
 {
TOGGLE_DIRREG |= (1 << TOGGLE_BIT);
while(u8Cnt--)
        {
        TOGGLE_PORT |= (1 << TOGGLE_BITNR);
        _delay_us(1);
        TOGGLE_PORT &= ~(1 << TOGGLE_BITNR);
        _delay_us(1);
        };
};
#endif
```

Im Debug.h file:

```
#include <stdint.h>
#include <avr/io.h>
#include <util/delay.h>
#include "debug.h"
```

```
#define TOGGLE_PORT        PORTC
#define TOGGLE_DIRREG      DDRC
#define TOGGLE_BITNR       PC3
#define TOGGLE_FLAG_USED
```

Debugging Hilfsfunktionen

```
char * ByteToBin(uint8_t u8Byte)
{
static char caBin[9];                //groesse 8+1
char * pcParser = caBin;            //Pointer als Parser
int8_t i8NN=7;
do
        {
        if( u8Byte & (1 << i8NN) )
                {
                *pcParser = '1';
                }
        else
                {
                *pcParser = '0';
                };
        *pcParser++;
        }while( i8NN- -);
caBin[8]=0;             //Terminierung
return  caBin;
};                      // Bsp.  ergibt        →    „00100110“
```

Wir brauchen noch einige Werkzeuge die Mikrocontroller typisch sind

Beispielweise möchten wir einen beliebigen Text, eine Zustandsanzeige oder einen errechneten Wert **regelmäßig, langsam, lesbar** ausgeben, oder eine Aktion langsam ausführen.

z.B. ein Analogwert, der nicht öfter als jede Sekunde gebraucht wird, und sei es ein Lämpchen langsam blinken zu lassen..,..

Dann soll das ohne nutzlose wie unsinnige Warteroutinen ablaufen.

Das geht mit einer Art **Galeeren Pauke**nschlag. (Synchronisierung)
Oder auch TICK-Counter.
Ein „**Tick Counter**“ ist oft einfach eine Zählervariable die evtl. genau 1000*/Sekunde um 1 hochgezählt wird. Somit ein **Millisekunden Timer**.
So etwas leitet man aus der Megaherz großen Quarzfrequenz ab.
Und zwar mit einem Zähler, oder Timer im CTC Mode:

16MHZ Quarz= F_CPU

→ F_CPU / 256 / **250** = **250 Pulse /Sekunde**

Ein langsamer Zähler: Tick-Counter

Eine automatische Routine (**Timer_Compare Interrupt**) sorgt dafür das ein langsamer Zähler zu Verfügung steht. Damit können Zeitvergleiche gemacht werden.
Bsp: Der Takt 16Mhz wird 256 mal geteilt und zählt dann noch bis 256
Das bedeutet 16^6 / 256 / 250 = **250** * **pro Sekunde**

Was brauchen wir?

```
        volatile uint8_t      gu8Status_Flags;
        #define TICKS_1SECOND    250
        #define TICK_EVENT        0x01
ISR(TIMER2_COMPA_vect)    // TIMER0_OVF_vect
 {
 static volatile uint32_t    gu32_Ticks;    // Interrupt Laufzähler
 static volatile uint32_t   gu32_TickComp;     //Vergleicher
 gu32_Ticks++;
 if( (gu32_Ticks - gu32_TickComp) > TICKS_1SECOND)      // F_CPU →10^16/256/250 =
250
        {
        gu32_TickComp = gu32_Ticks;            //Zeit wieder merken !!
        gu8StatusFlags |= TICK_EVENT;          // Ein Flag „setzen“ um die
Information zu speichern
        };
};
```

1 Sekunden TICK COUNTER

Initialisierung TIMER 2 beim ATMega328p

```
TCNT2 = 0;
OCR2A = 250;
TCCR2A = (1<< WGM21); // CTC Mode
TCCR2B = (1<< CS22) | (1<< CS21); // /256 Vorteiler -->16^6 /256
TIMSK2 = (1<<OCIE1A); //TOIE2 Aktiviert Int0 OVL Interrupt
```

Warum OCR2A=250?

→ 16^6 / Vorteiler= 16^6/256 / **250** = 250 (ebenfalls 250)

#define TICK_1SECOND 250

Und zufällig ist auch der CTC Ladewert im OCR2A = 250;

Auswertung

….weit unten in .main…..

```
if (gu8StatusFlags & TICK_EVENT)    // ??? Ist das Bit gesetzt ???
   {
   gu8StatusFlags &=~ TICK_EVENT;   -→
            //lösche das Bit UND  tue somit was jede Sekunde
   };
```

Das Problem ist nicht Daten im Controller zu verarbeiten, sondern Daten formatiert in den Controller zu laden.

Hier eine „fast“ gelungene quantisierte Digitalisierung der schiefen Ebene.
(gesehen-Keilberg Regensburg).
Hier reicht offenbar die Auflösung oder Bitzahl zur Digitalisierung der schiefen Ebene nicht aus um die Schräge ausreichend darzustellen.

Es wird Zeit für was praktisches, haptisches, reales und gleichzeitig überzeigendes.

Auch wenn wir jetzt noch nicht alles kennen und wissen.

Mit der direkten Zuweisung lässt sich schon was machen

..bevor es losgeht Programmier Stil

Theoretisch kann man Syntaxgerechten C-Code schreiben der funktioniert, aber der keinerlei Rückschlüsse darauf zulässt, **warum**!

……..Manche machen eine Sport daraus ☺

Ein Progammierstil ist eine Konvention, aber auch ein Pflicht!

Sobald man **nicht mehr alleine** ist, z.B. *in einer Firma*, wird Soll zum **Muss**.

Der Lohn:

Fehlervermeidung, Transportabel, Pflegbare Programme, weil lesbar, vermittelbar, verständlich und gut kommentiert.

Für sich selbst merkt man bald, dass man auch nach 4 Wochen noch weiß, wie es gemeint ist, wo, wie weitergeht, warum was wann passiert.

Der Aufwand lohnt sich.!!

Siehe: http://de.wikipedia.org/wiki/Programmierstil

Ich möchte in folgenden Seiten immer wieder neue Aspekte eines guten Programmierstils vorstellen.

Ich möchte an dieser Stelle eine **wichtige**

Schreibweisen-Konvention

anregen wie ***Variablen, Konstanten*** etc. geschrieben werden.

Konventionen sind **Vereinbarungen** und keine zur Sprache gehörenden, zwingend vorgeschriebene Regeln. Jedoch sind sie Extrem hilfreich und ein erkennungsmerkmal der Professionalität. Es nennt sich „Professioneller Stil“.

Der Sinn, ist:

A: Im Programm sprechende, und mit Datentyp markierte Variablen/Komponenten zu schreiben und so selbst immer zu wissen was im Code an dieser Stelle eine Variable oder Konstante bedeutet.

B: Sich selbst daran zu halten, zu gewöhnen, und so eine einheitlichen STIL zu entwickeln.

C: Durch die Weitergabe der Konventionen wird das Programm auch für Andere verständlich und somit **pflegbar,** z.B. wenn man beweisbar Programme auf Ihre Sicherheit (Luftfahrtbundesamt etc.) schreibt.

Mit dieser „Methode!“ kann man im Code beim Schreiben an dieser Stelle sofort sehen, was ein beliebiger „Ausdruck“ bedeutet, und kann so Fehler vermeiden.

Programmier Stil

MERKE: Auch wenn in Lehrwerken so gut wie aus immer EIN-Buchstabenvariablen ala „ for (i=0; blabla ……. usw. geschrieben werden, so ist dies ausschließlich als eine Beschreibung der Sprache zu verstehen,

WIE die Sprache im Lehrwerk funktioniert.
<u>Nicht jedoch</u> ob dies so anzuwendender „guter Programmierstil“ ist.

Gerade innerhalb von Firmen mit mehr als „1nem“ Programmierer ist dies Pflicht.!

Das wollen wir in Zukunft besser und gemeinsam machen.
„Darauf lege ich grossen Wert!“
Selbst gesehen, gelernt habe ich dies von Microsoft Programmierern. (vor 30 Jahren)
Siehe auch: https://de.wikipedia.org/wiki/Programmierstil

Bsp:
Schreibe bitte „zum Anfang“ Variablen immer sprechend, aber kurz:

statt i → u16Cnt soll für Counter stehen

u16= Datentyp > unsigned 16 Bit = 0..65535
Cnt sagt, was es ein soll. Eben ein Zähler

Die weiteren Regeln sind jetzt verlagert im Anhang so nähe Seite 570

Das alles ist nicht frei erfunden und daher gesagt.
Es gibt eine Handlungsrichtline:
MISRA C:2012

Info hier:
https://barrgroup.com/embedded-systems/books/embedded-c-coding-standard
(frei verfügbar)

und hier:
https://www.misra.org.uk/Publications/tabid/57/Default.aspx

Diese Information ist von Frank Büchner.

Aber wie ich erfahren habe, scheiden sich die Geister.
Viele halten meine „polnische Notation“ für eine alten, überflüssigen Hut.
Ich mag es. Punkt.

Standard Beschreibung von „C“
versus Anwendung Stil

Der Sprachstandard **ANSI-C** ist im Handbuch von „Kernighan und Ritchie“ =K&R Standard, präzise definiert.
Das Buch „C Programming Language“ beschreibt bestens die
„formale Struktur der Sprache ANSI **C“**
http://cs.indstate.edu/~cbasavaraj/cs559/the_c_programming_language_2.pdf

Jedoch nicht, was „**professioneller Programmierstil“** ist.

Damit ist gemeint, sind daraus ergebende:
Programmier-Strukturen wie die Nutzung von Einzel-Flagbits für die Programm-Ablaufsteuerung,
intensive Nutzung von **#define** bis Inline Makro Nutzung,
Statemachine,
Interface Methoden wie **CMSIS** bei ARM µControllern, etc. und vieles mehr…..

Programmgliederung in „C“
Wo steh was?
Folgendes steht eher im zugehörigen „header file“

Funktions Prototypen:

-evtl. gleich ***extern*** deklarieren, wenn in anderen Modulen gebraucht:

```
void  ReadADC(unsigned char ucMux );
extern void MyFunction( unsigned int  gu16Value );
```

Globale Variable im ganzen Programm bekanntgeben:

```
extern char gcaStr[ STRMAX + 1 ];
```

mit der “**inline**” Deklaration wird der Code einer Funktion direkt eingefügt und so kompiliert, und ist somit **kein Funktionsaufruf**:

```
//Bewusst anwenden !!.  Kostet jedes mal Speicher
inline void StatusLED( unsigned char Color );
```

MERKE:

Wichtig ist es, **schon ab den ersten Zeilen**, beim Programmieren
LESBAREN CODE zu schreiben.
Das heißt: (aber es ist nur eine Empfehlung von mir, nicht pflicht)
Schon von Anfang an Variablen der Bedeutung entsprechende Namen zu geben.

Damit ist gemeint, EIN-Buchstaben Variablen ohne Kennzeichnung des Datentyps zu vermeiden.
Auch wenn dies laut K&R FORMAL Richtig ist.

„Also nicht weiterzumachen wie bisher.!“

Deshalb darf und soll man dennoch **dazulernen,** heißt **den eigenen Stil erweitern** und z.B. durch **SPRECHENDE NAMEN** professioneller Programmieren

Mit Sprechend ist nicht geschwätzig gemeint:

u32DieseVariableDrehSichWieEinWurmDurchDEnCompiler

Warum **keine „Ein“-Buchstaben Variablen** . NIE WIEDER !

Betrachten wir mal ein Beispiel nach K&R Handbuch:

Bsp: for(i=99999**UL**; i < -430, i--) { ..};

was ist i ? also:

→ welcher Datentyp hat i ?=

→ für was steht i ?

→ kann i überhaupt einen Wert von 99999**U**L speichern?

(**UL** sagt dem Compiler → unsigned long für Konstanten gößer „int“)

→kann i überhaupt negativ werden ?

Wenn ich, satt dessen, den **Typ der Variable sichtbar** mache und **sinnvoll benenne**:

signed long s32AbstandCount; // Deklaration der Variable

for(s32AbstandCount =99999UL; s32AbstandCount < -430, s32AbstandCount --) { ..};

…kann ich sofort sehen, **ohne eine weit entfernte Deklaration zu kennen**,

1.)welchen Datentyp eine Variable hat, hier: „s32 Bitbreite“

2.) und für was es steht. Offenbar ein Zähler für irgendeinen Abstandsmesser der auch negativ werden kann und soll.

Es schadet also nicht, dazuzulernen, es besser zu machen…

Schreibweise:
Konstanten und **Definition**

Ausnahmslos Immer in → **Großbuchstaben**

Also **ALLE #define, Makros(), const PROGMEM**

wie Texte und Byte-Sequenzen

```
#define MASTER_PLAN        123
#define YELLOW_LED_NR      0x01       //Aufzählung
#define YELLOW_LED_BIT     0x02   oder  (1<< YELLOW_LED_NR)
#define LED_PORTX          PORTC
define  SETYELLOW_LED()    (LED_PORTX |= (1 << YELLOW_LED_BIT )

const char MCA_ARDUINO_SM[] PROGMEM =
            {0x08,0x0C,0x04,0x06,0x02,0x03,0x01,0x09};
```

usw..

DIES IST SEHR WICHTIG !! Für einen guten Stil

Wir wissen jetzt schon:

benutze niemals nur EIN-Buchstabe Variablen.
Auch wenn dies in 1000 Lehrwerken so steht.
WIR MACHEN DAS AB SOFORT BESSSER

Schreibe:
statt **i=** → **Count=** **(IMMER)**
Das ist schon sehr, sehr, sehr viel besser: versprochen.
…..und keine Arbeit. Denn Copy – Paste erledigt dies und das auch noch fehlerfrei.
Aber hilfreich wäre es noch am Wort „Count“ sehen zu können

A: Wie viel, wie weit diese Variable zählen kann?
B: ob dieses Count auch negativ sein kann?
C: ob Count eine lokale oder eine globale Variable ist?
D: ob dies Ganzzahlen oder **float** sind?
E: ob dies eine normale Variable oder ein Pointer ist…
usw.
Das geht. Durch Voranstellung eines PREFIX-Codes in Kleinbuchstaben
Bsp: **gu16**Count → sprich laut: „global unsigned 16Bit Counter“
i32 Count → sprich laut: „ local integer 32Bit Counter“
gpi8 Coutn→ sprich laut: „global pointer auf integer 8Bit Counter“

Eine übersichtliche Anzahl Regeln für Variablen, Konstanten, Funktionen, sogenannte **Präfix-Bezeichner,** haben sich bewährt. Diese sind fast selbsterklärend und stammen von Profi-Programmieren und werden so, oder so ähnlich, in Firmen angewendet. **Und ich mache es auch so!**

Beispiel: → unsigned int * **gp16**Parser; →

Nach dem Typ-Präfix kommt noch die Bitzahl der Variablen.

g = Global
p = Pointer
c = char
b = BYTE = unsigned char
ca = char-arrray
i = Integer
u = unsigned
w = WORD = unsigned int
l=long
f = float
d = double
z = zeroed array = STRING

(wird so gern vom Mikrosoft Programmierern verwendet)

(das ganze ergänzt **durch die Zahl der Bitbreite** der Variable):

„**u16**VariablenName“ → lokale unsigned int.

Konstanten, Makros und Definitionen werden immer mit GROSSBUCHSTABEN geschrieben.

Makroparameter

```
#define   STRMAX   80
#define   _BV(x)      (1<<(x))
#define   LED_YELLOW_ON()     (PORTB |= 0x20)
#define   LED_YELLOW_OFF()    (PORTB &=~0x20)
```

Anwendungsbeispiel:

u8NN → unsigned-8Bit Zählervariable

gau8Feld = Global unsigned 8Bit Array

Beispiele: benutze silbenweise Großbuchstaben in Variablennamen.
schreibe den Datentyp der Variable in Buchstabenkürzel davor.

gui16 DiesIstEineVariable; lese: **g**lobal-**u**nsigned-**i**nteger-**16**Bit-DiesIstEineVar..

---

eine Variable mit **8 Bit** aus char, wird so geschrieben:
char **c**Zeichen;

ein Char-Array mit **8 Bit** aus char, wird so geschrieben:
char **ca**VariablenName[STRMAX]; // Lese **char array** <Name>
Bsp: caLCDText[20];

eine Variable mit **8 Bit** aus **unsigned char** wird so geschrieben
uint8_t **u8**VariablenName; Bsp: u8KeyFlag;

eine Variable mit **16 Bit** aus **unsigned int** wird so Geschrieben
uint16_t **u16**VariablenName; // Lese: unsigned int

eine Variable aus pointer auf char wird so Geschrieben
char* **pc**VariablenName; //Lese: Pointer auf char <VariablenName>

Kombination:
volatile static uint8_t **gu8**RrogStatus; → globales Statusbyte

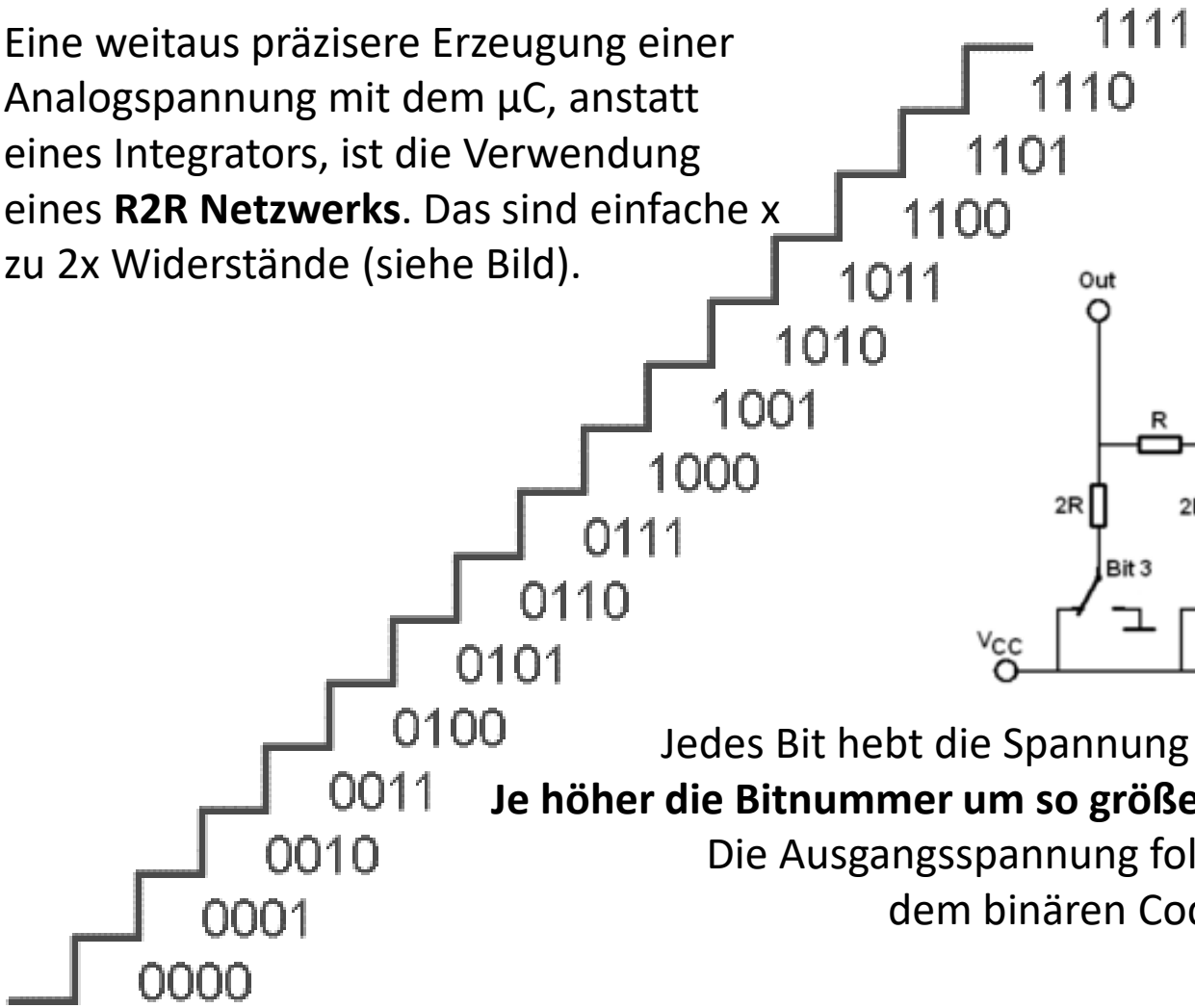
Zwischenspiel:

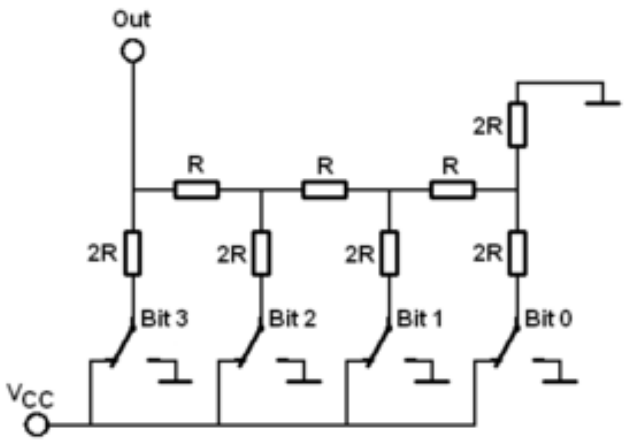
Analogsignal Erzeugung durch
Wertzuweisung
an ein Widerstandnetzwerk
Digital Analog Converter
DAC mit
R2R

POOR MAN‘S FUNCTION GENERATOR

Digital-Analog-Konversion – **DAC** mit einem R2R Netzwerk

Eine weitaus präzisere Erzeugung einer Analogspannung mit dem µC, anstatt eines Integrators, ist die Verwendung eines **R2R Netzwerks**. Das sind einfache x zu 2x Widerstände (siehe Bild).



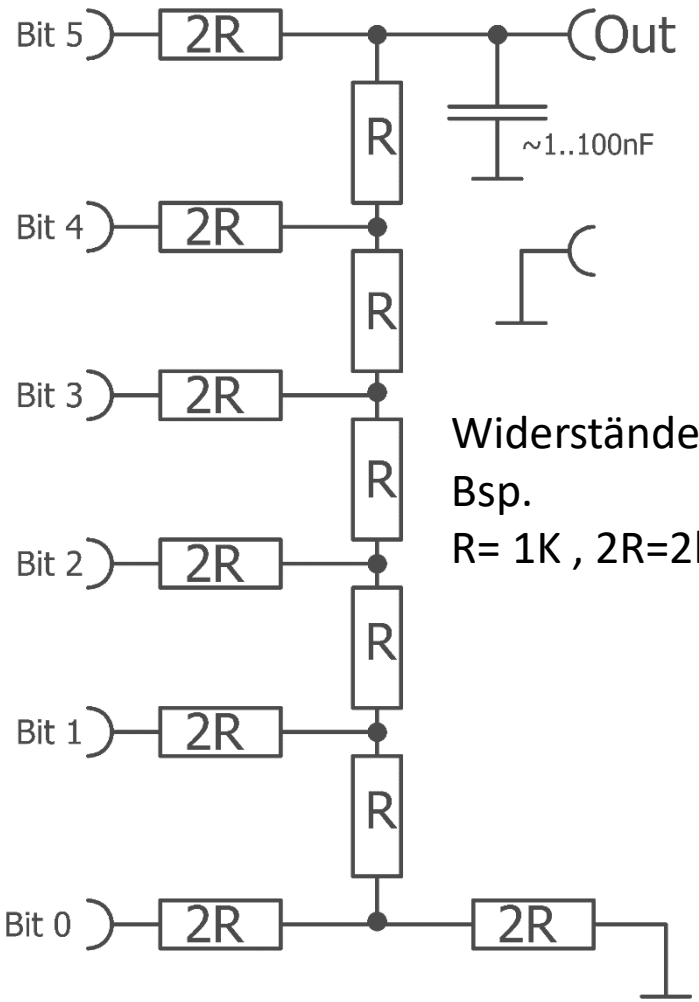


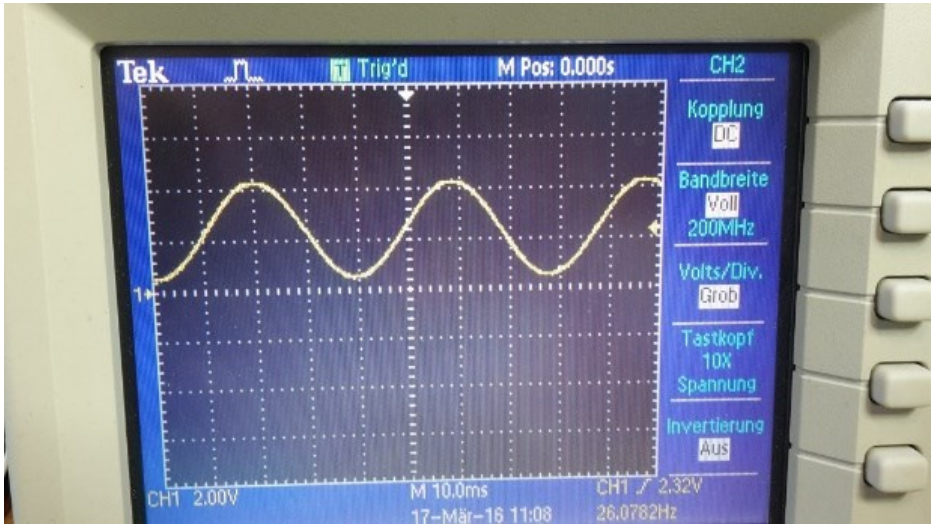
Jedes Bit hebt die Spannung um eine Stufe.
Je höher die Bitnummer um so größer sein Einfluss.
Die Ausgangsspannung folgt **unmittelbar** dem binären Code am Eingang.

„Poor mans function generator“

Sehr schneller Digital Analog Konverter „DAC“

Steckbrettaufbau für 6 Bit →0..5 mit PORTB

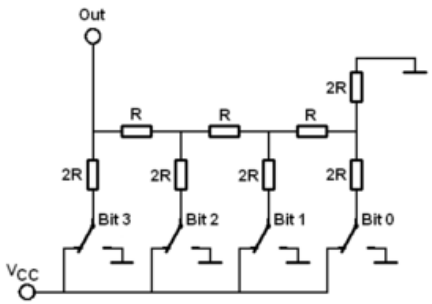




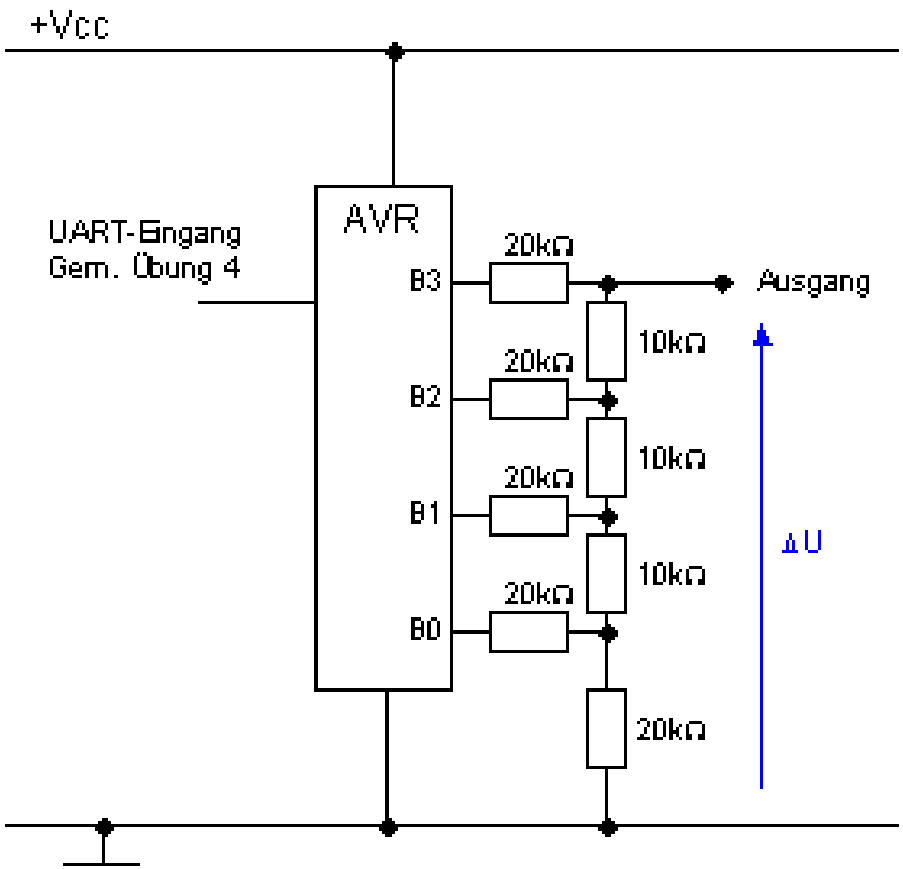
Widerstände:
Bsp.
R= 1K , 2R=2k oder ~2K2

Out an Oszillograph anschließen.
Ein Transistor-Emitterfolger kann einfach als Impedanzwandler eingesetzt werden.

Jedoch werden ~0.7 Volt BE–Spannung Mindestspannung benötigt.
(Offsetspannung)



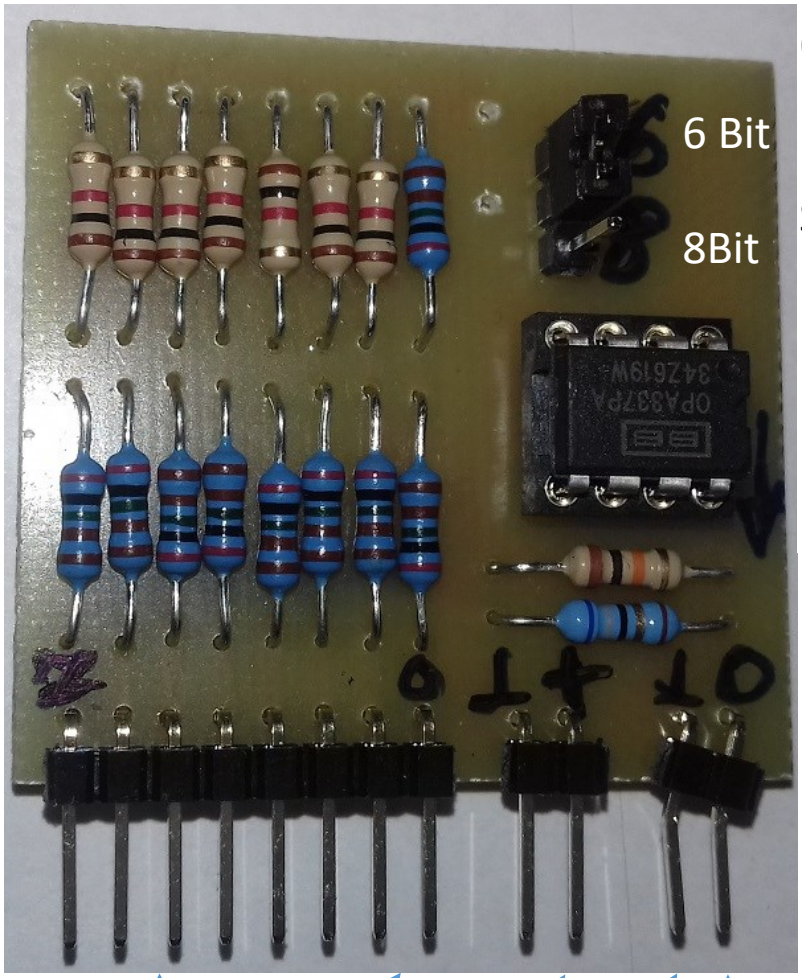
Die Ausführung ist einfach, wenn auch aufwendig.
Bei 8 Bit Auflösung sind es schon 16 Widerstände.
Ein echtes Widerstandsgrab. Aber es bliebt simpel.



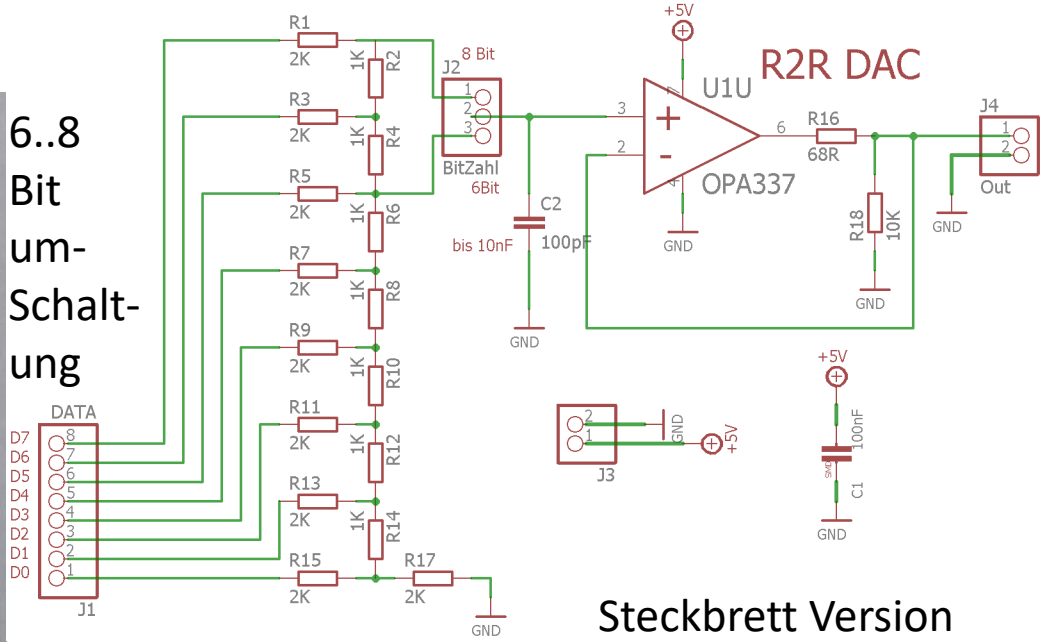
Nimm zum Probieren
irgendwas im ~1:2 Verhältnis:
1K + ~2K oder 4,7K + 10K

```
uint8_t u8NN=0;
DDRC = 0b00111111; //0x3F; //6Bit OUt
PORTC = 0;

while(42) /* loop forever */
  {
  wdt_reset();
  //Toggle_Bit( 1 );
  //6Bit Sinus = 64
// Width=64 → Mitte + Ooffset +/-
  PORTC = 31+31*sin( 2* M_PI * (u8NN/64.0) );
  u8NN++;
  u8NN &= 63; //u8NN %=64;
  };
```

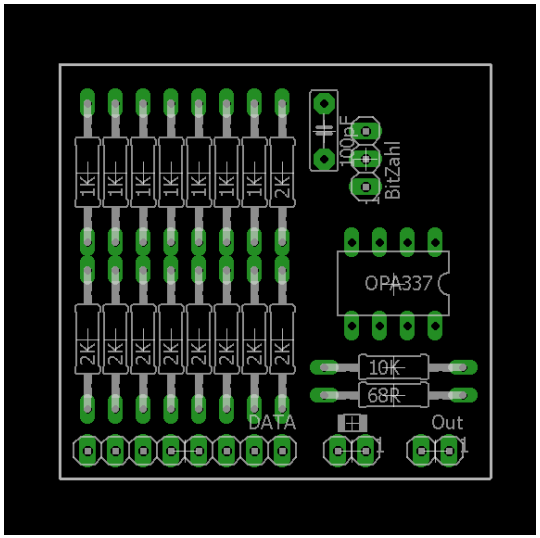
DAC mit R2R 2017



6..8 Bit um-Schalt-ung



Steckbrett Version

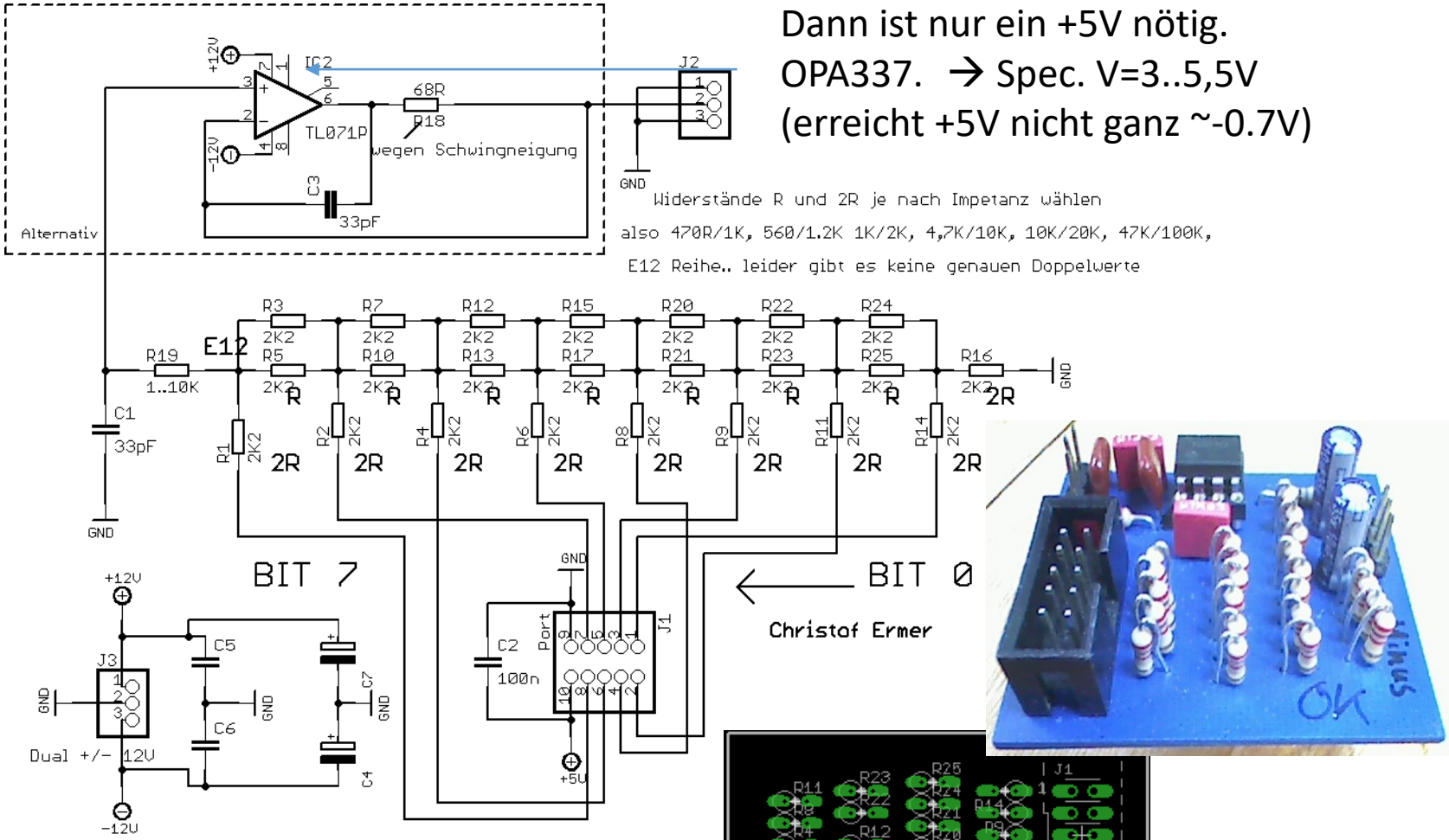


OP evtl. herausnehmen und Pin (3)+Inp mit Pin(6)Out mit kurzem Draht brücken.

z.B. 2cm Widerstands-beinchendraht

D7..D5.....D0, GND,+5V, GND, Analog

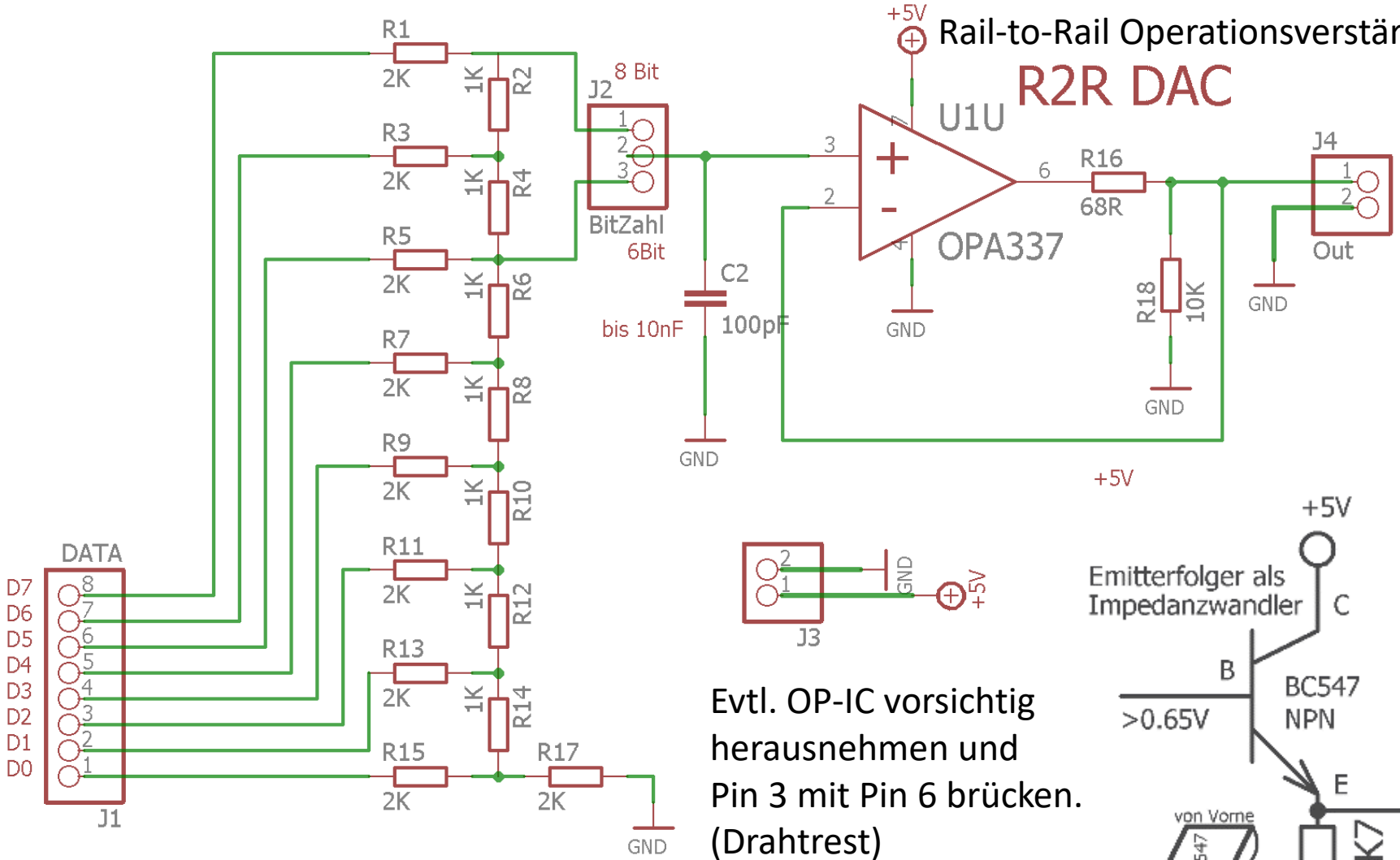
Rail-To-Rail OP empfohlen,
Dann ist nur ein +5V nötig.
OPA337. → Spec. V=3..5,5V
(erreicht +5V nicht ganz ~-0.7V)



Evtl OP vorsichtig
herausnehmen und
Pin 3 mit Pin 6 brücken.
(Drahtrest)
Alternativ: Pin3 mit Pin2

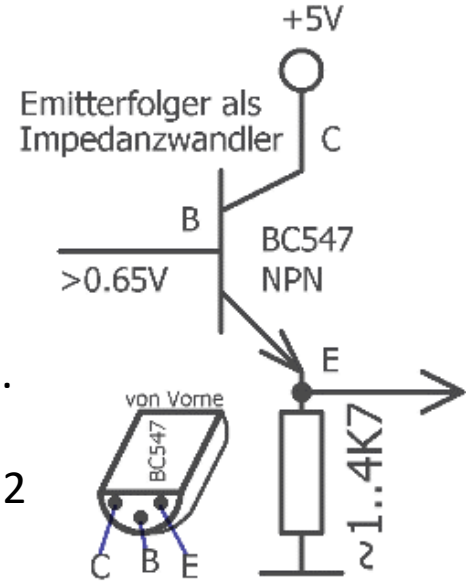
Rail-to-Rail Operationsverstärker

R2R DAC

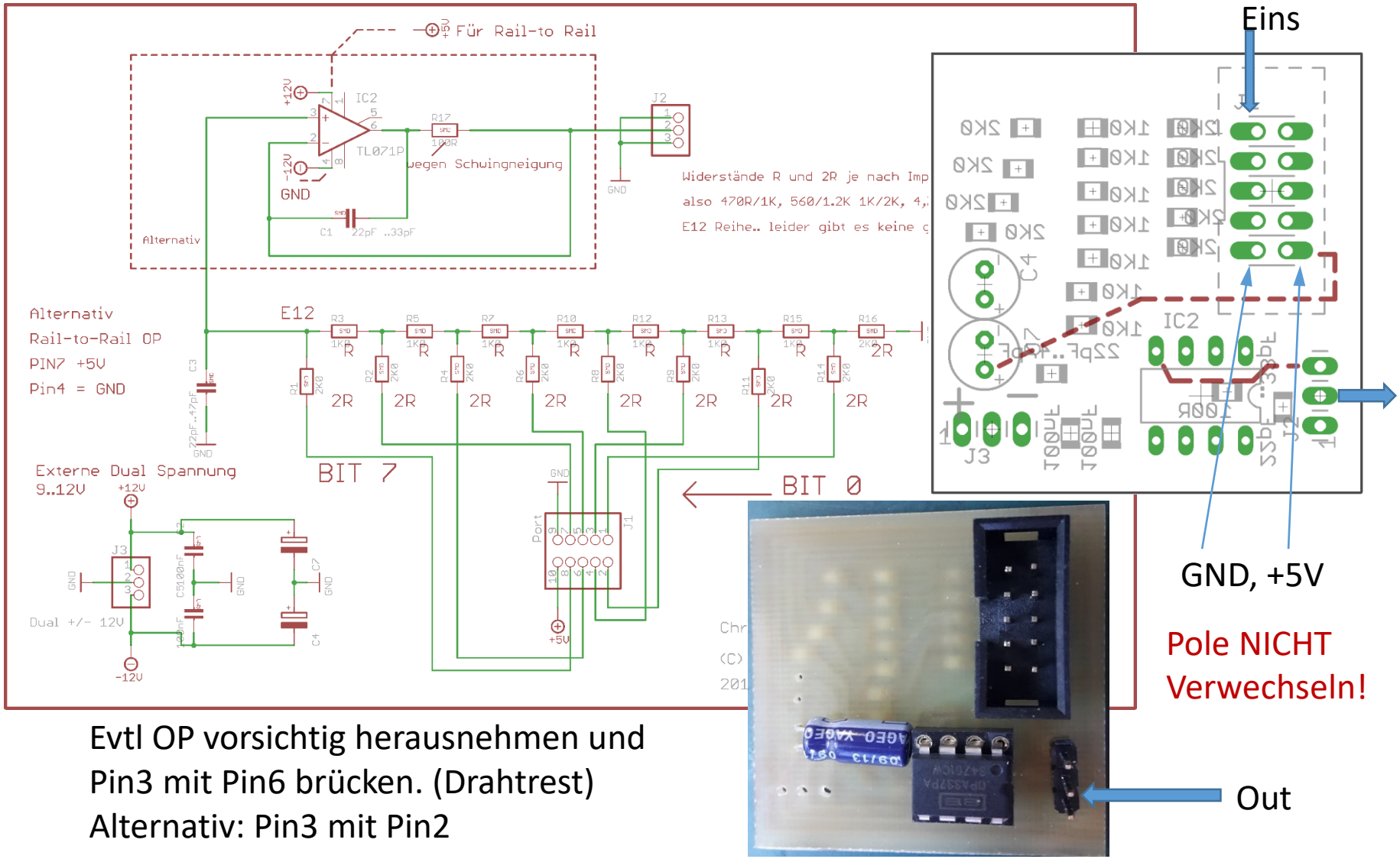


Evtl. OP-IC vorsichtig herausnehmen und Pin 3 mit Pin 6 brücken. (Drahtrest)
Alternativ: Pin3 mit Pin2

Oder ein Emitterfolger→



R2R-Variante mit SMD Widerständen und alternativen Rail-to-Rail-OP aufgebaut auf Testplatine

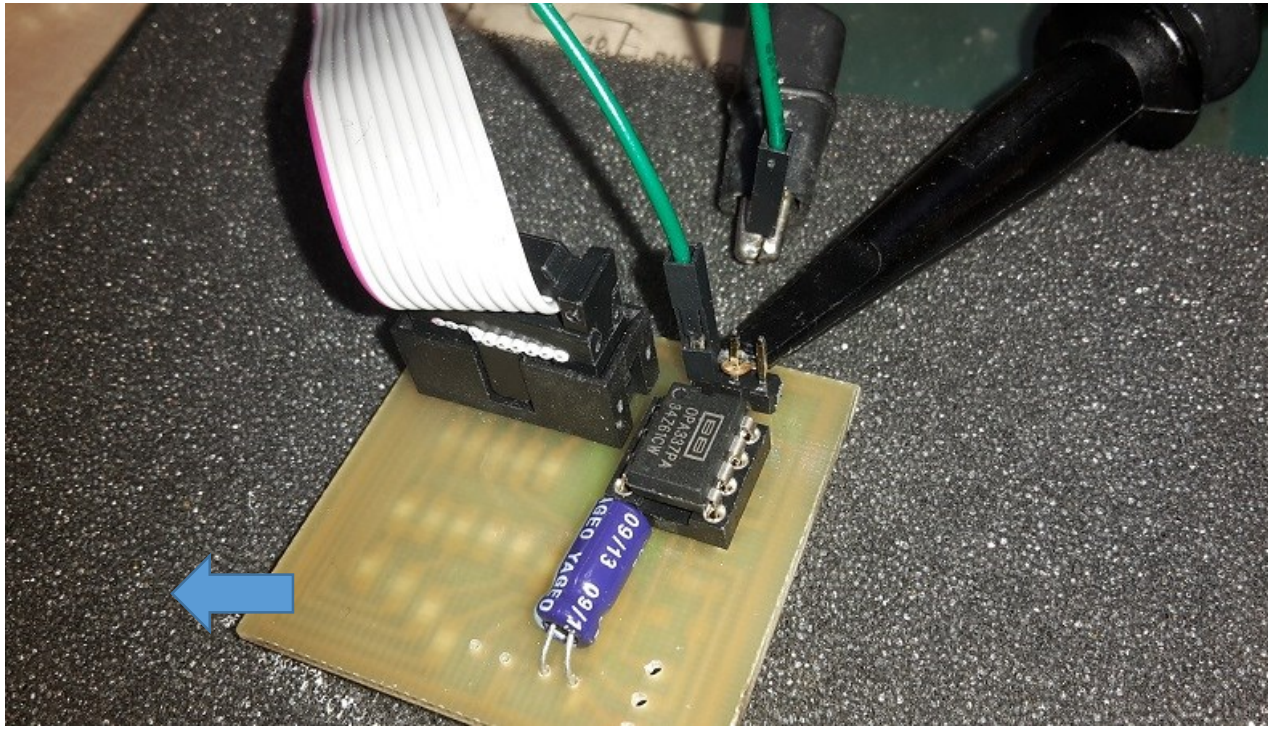


Evtl OP vorsichtig herausnehmen und
Pin3 mit Pin6 brücken. (Drahtrest)
Alternativ: Pin3 mit Pin2

Signalsynthese mit Testplatine

MERKE:
PORTA hat nur 6 BIT
Beim ATMgea328p.

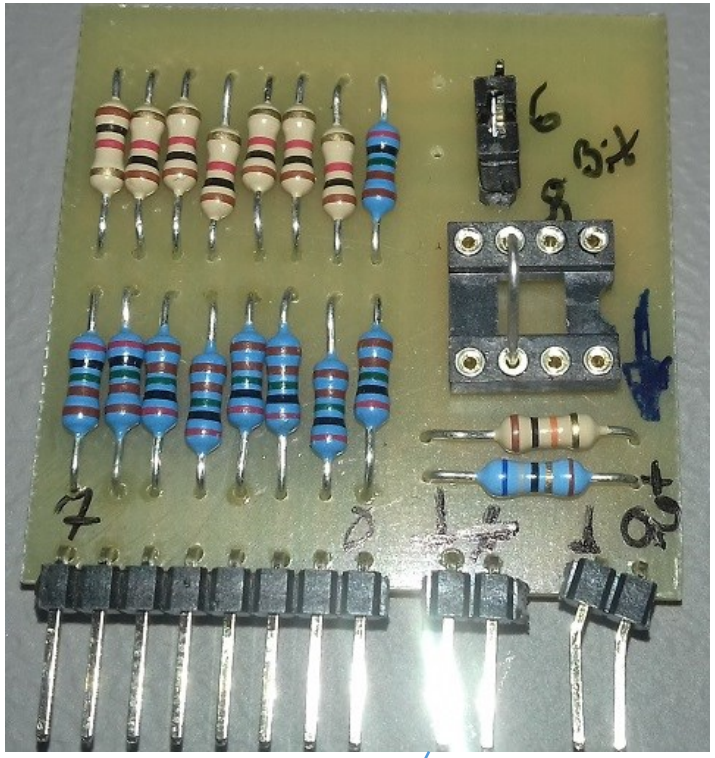
Weiche auf PORTD aus.

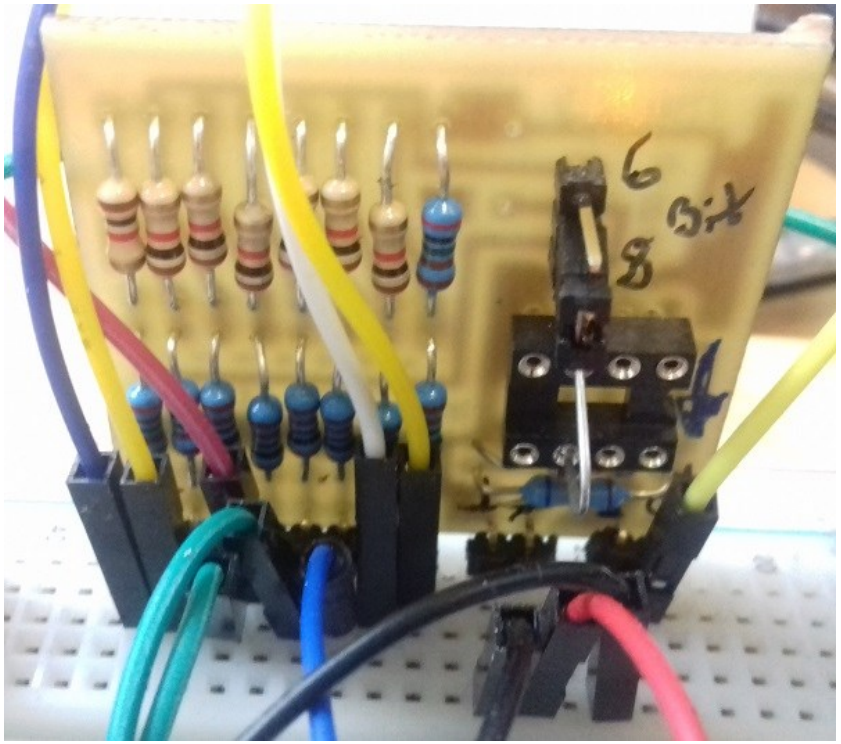


```
uint8_t u8NN;
DDRA = 0xFF;
// ************ MAIN LOOP FOR EVER ********************
for (;;)  /* loop forever */ //Aber schnell > 200.000 mal/Sekunde
    {
    // "Hauruck" --verfahren

    //PORTA= u8Level; //Rechteck
    //PORTA= 0; //Rechteck

    u8NN++;
    //PORTA= u8NN++;//Sägezahn

    PORTA= 127+ 100 * sin( 2.0 * M_PI * (u8NN / 256.0));  //SINUS
    }
```

Wie man sieht wird trotz 16MHz gerade mal ein 26Hz Sinus daraus.

Steckbrett - letzte Reihe in
Blickrichtung im oberen Feld
Am Jumper 6-8 Bit Betrieb wählen.





Bit 7……..0

Out

0V=Masse=GND -

Plus 5V

Weiterer GND Pin

Nicht „niemals“ Plus und Minus
verwechseln. BITTE !
Ardunio Power Pins:
Pin 5V
Pin GND

Evtl OP vorsichtig
herausnehmen und
Pin 3 mit Pin 6 brücken.
(Drahtrest)
Alternativ: Pin3 mit Pin2

Compiler Turnaround:

Kopiere die Beispiel Ordner *„2C4-ATM328-PORTD6-SINUS-8Bit“* in deinen **„lokalen Ordner“** z.B. „……..\my_AVR_Cs\“ z.B: auf dem Desktop:

→ \meinOrdner\2C4-ATM328-PORTD6-SINUS-8Bit\

Öffne „Programmers Notepad“

→ C:\WinAVR-20100110\pn\pn.xe

Gehe mit den Dateiexplorer zum „**lokalen**“ Programmordner

Bei mir:

„**K:\PHY\elfort\0000 Kurs2018_WINAVR-Progs\2C4-ATM328-PORTD-SINUS-8Bit**“

→Kopiere diesen Link aus der Adresszeile in die Zwischenablage

Öffne→ File → Open Project →

→kopiere die Zwischenablage in „ Dateiname“

→drücke Return

→ klicke auf „**MyMainProjekt.pnproj**“

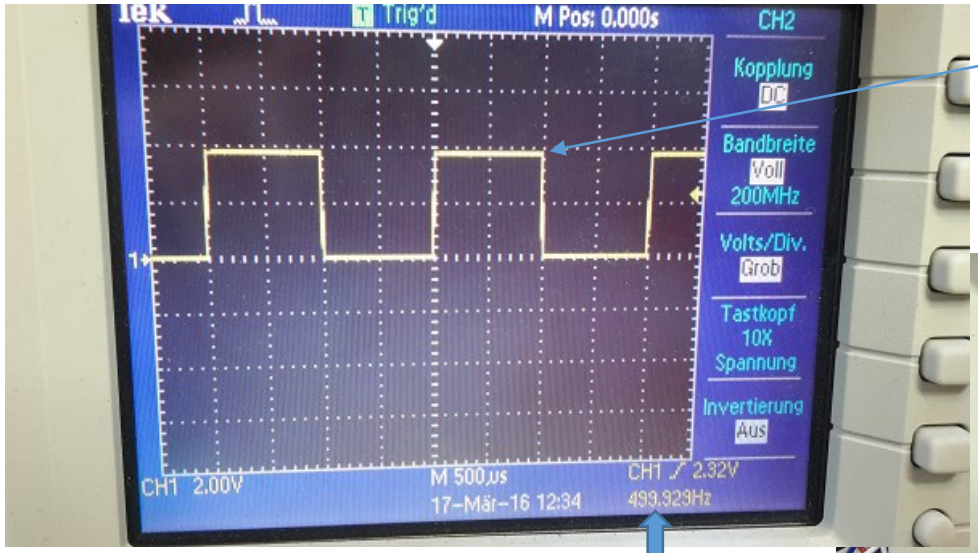
Öffne im Projects Fenster „**main.c**“ mit Klicken..

Editieren bei Bedarf:

dann im PN Menü → Tools—>Make All →

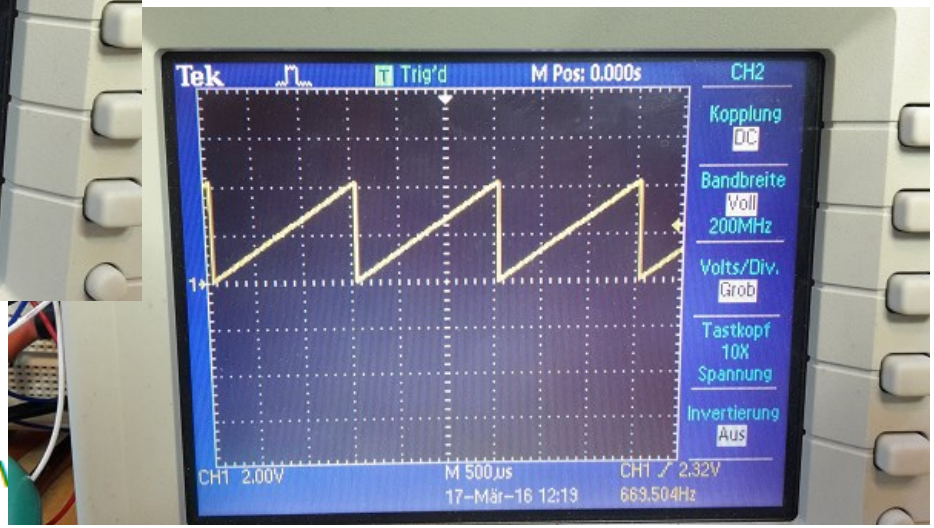
→ Programm

Signale mit Software erzeugen



Pegel wie u8Level
= 5V/256 *
u8Level = ~3,9 V

~500Hz

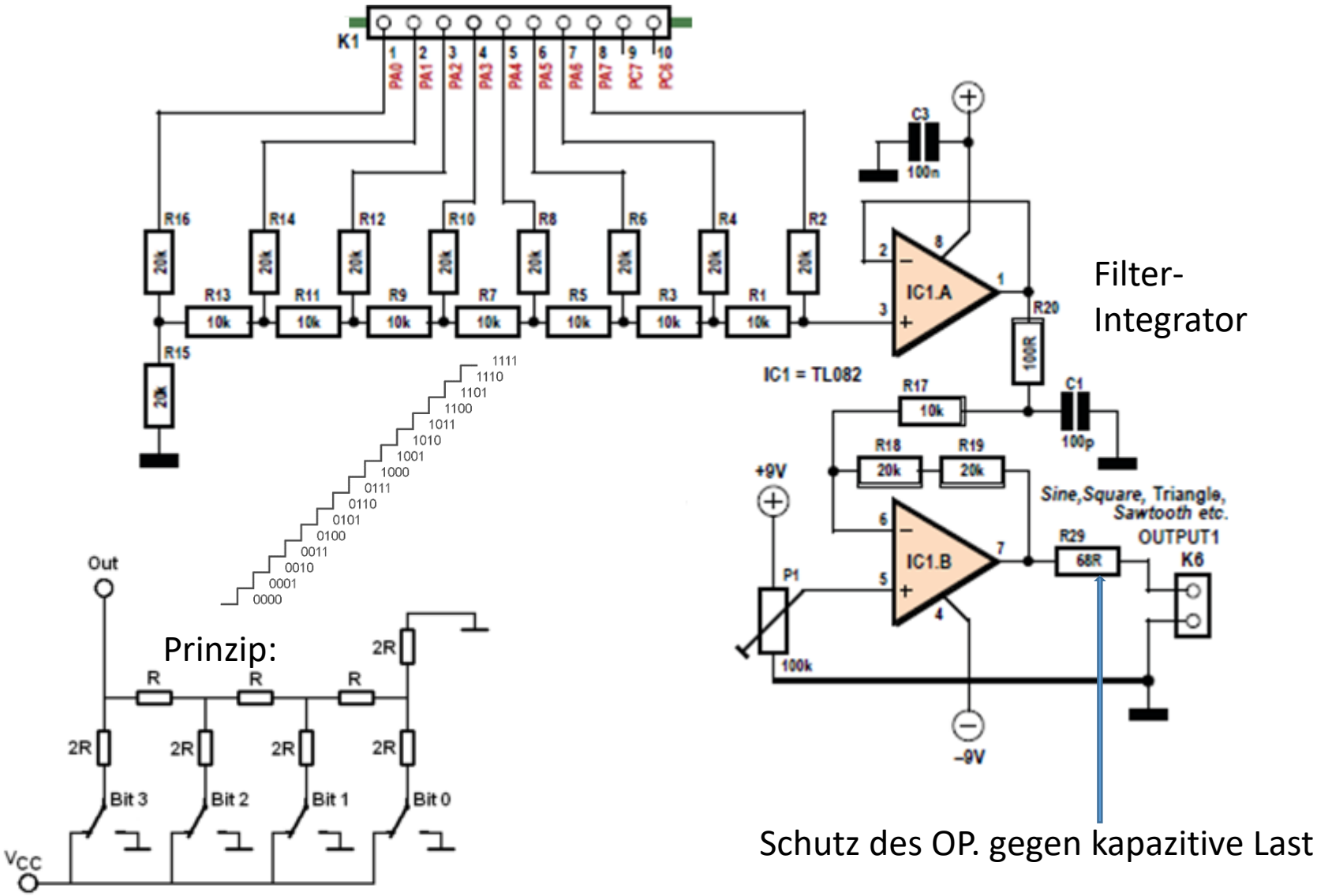



```
u8Level = 200;
// ************ MAIN LOOP FOR EV
for (;;)
    {
    // "Hauruck" --verfahren

    PORTA= u8Level; //Rechteck
    _delay_ms(1);
    PORTA= 0; //Rechteck
    _delay_ms(1);
```

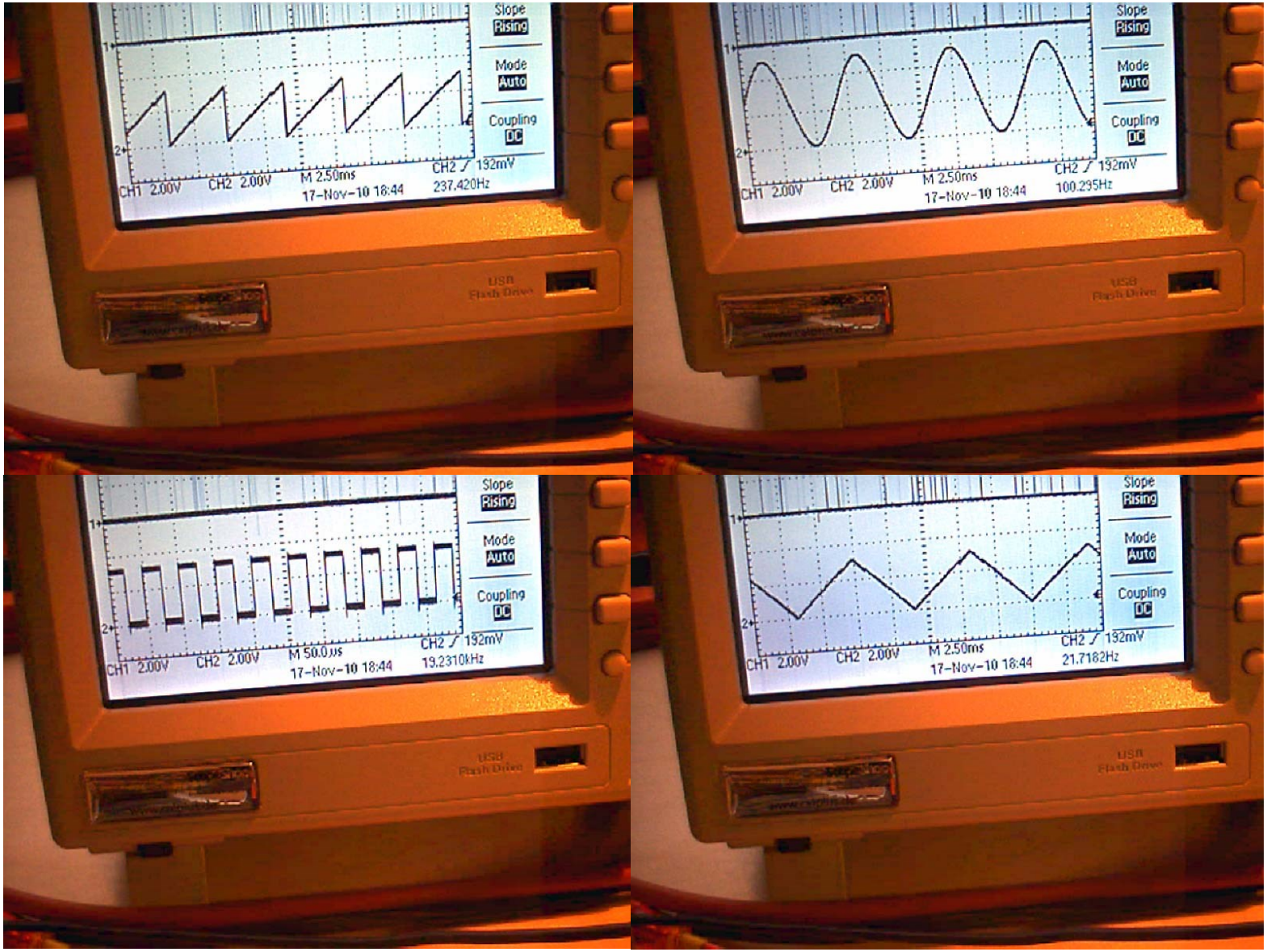
```
u8NN %= 220;
PORTA= u8NN++;//Sägezahn
```

670Hz mit 16MHZ →ohne Zeitbremse,
Modulo verlangsamt ganz ordentlich.

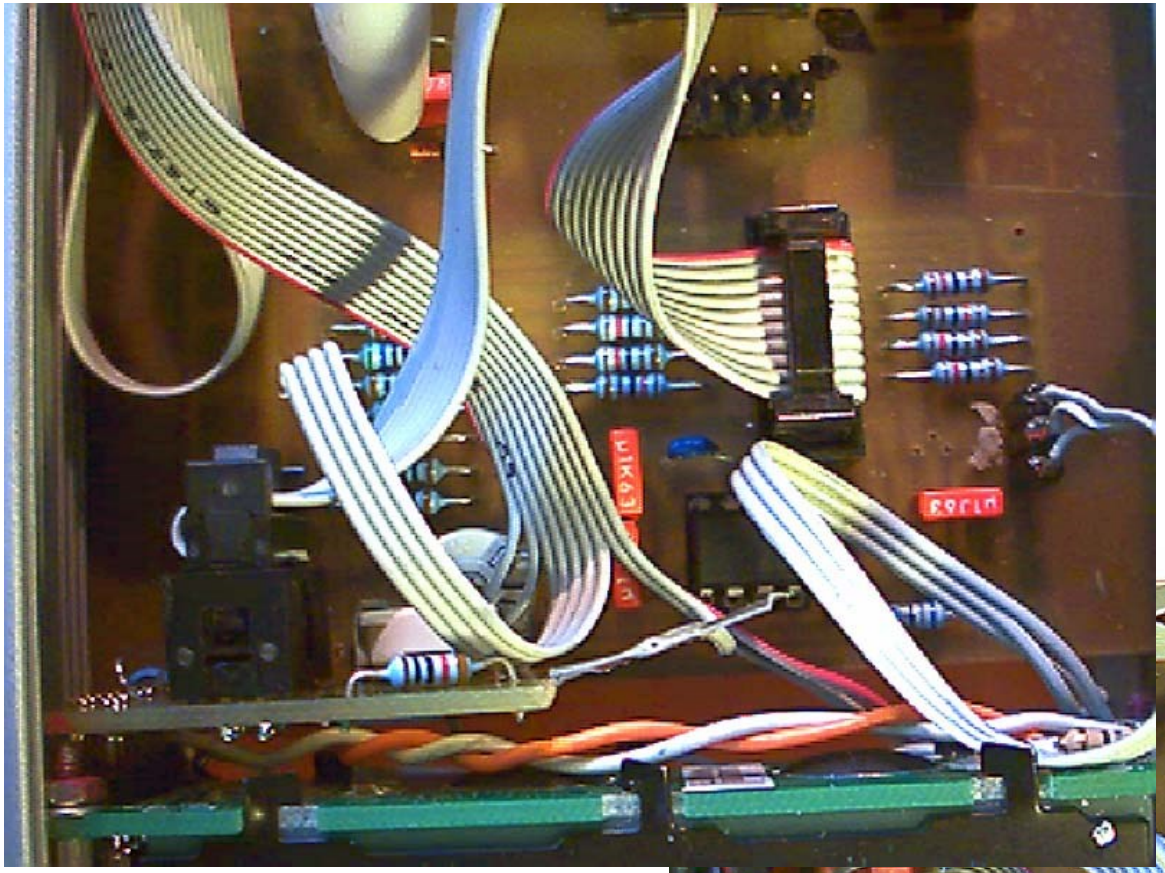
Bsp. Auszug einer realen Schaltung mit R2R



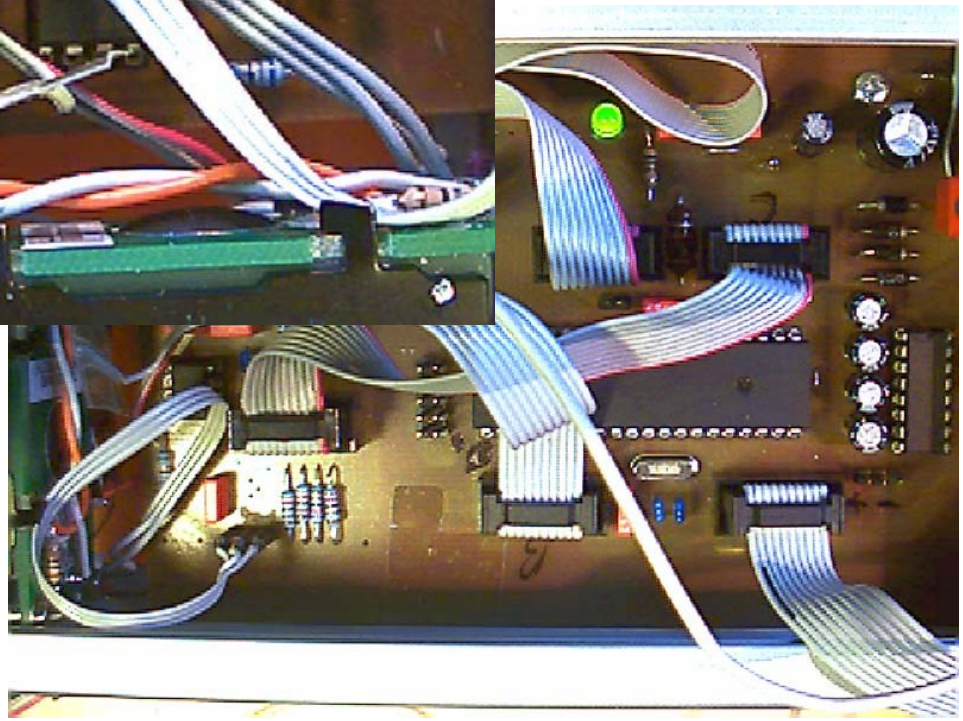
Diese Signalvielfalt ist locker möglich



Selbstgebauter Funktionsgenerator. NUR mit R2R DAC.
Die , die Amplitude und Kurvenform ist per Befehl einstellbar.



Links die R2R Widerstände.
Unten das LCD Display
...und das übliche Kabelallerlei.



Rechts der ATMega16 Controller mit serieller Schnittstelle.

Die **Sinuswerte** können entweder ‚online' berechnet werden:

uint8_t gu8SineVal[256]; → iNN= 0..255

gu8SineVal[iNN] = round(**127 + 127.5 * (sin(2*M_PI* (iNN/256.0)) + 1)**);

Einfacher:

gu8SineVal[iNN] = (uint8_t)round(**128 + 127 * sin(2*M_PI* (iNN/256.0))**);

…..oder viel schneller - als **Wertetabelle** - abgespeichert werden:
(wie das genau geht, dass **const-Arrays** fest in das Programmflash, und nicht in den RAM, geladen und von dort wieder gelesen werden, sehen wir später…)

```
const char MCA_SINUS[] PROGMEM = {      // "256 Byte SINUS von 0..255
128,131,134,137,140,143,146,149,152,155,158,162,165,167,170,173,176,179,182,\
185,188,190,193,196,198,201,203,206,208,211,213,215,218,220,222,224,226,228,\
230,232,234,235,237,238,240,241,243,244,245,246,248,249,250,250,251,252,253,\
253,254,254,254,255,255,255,255,255,255,255,254,254,254,253,253,252,251,250,\
250,249,248,246,245,244,243,241,240,238,237,235,234,232,230,228,226,224,222,\
220,218,215,213,211,208,206,203,201,198,196,193,190,188,185,182,179,176,173,\
170,167,165,162,158,155,152,149,146,143,140,137,134,131,128,124,121,118,115,\
112,109,106,103,100,97,93,90,88,85,82,79,76,73,70,67,65,62,59,57,54,52,49,47,\
44,42,40,37,35,33,31,29,27,25,23,21,20,18,17,15,14,12,11,10,9,7,6,5,5,4,3,2,\
2,1,1,1,0,0,0,0,0,0,0,1,1,1,2,2,3,4,5,5,6,7,9,10,11,12,14,15,17,18,20,21,23,\
25,27,29,31,33,35,37,40,42,44,47,49,52,54,57,59,62,65,67,70,73,76,79,82,85,88,\
90,93,97,100,103,106,109,112,115,118,121,124};
```

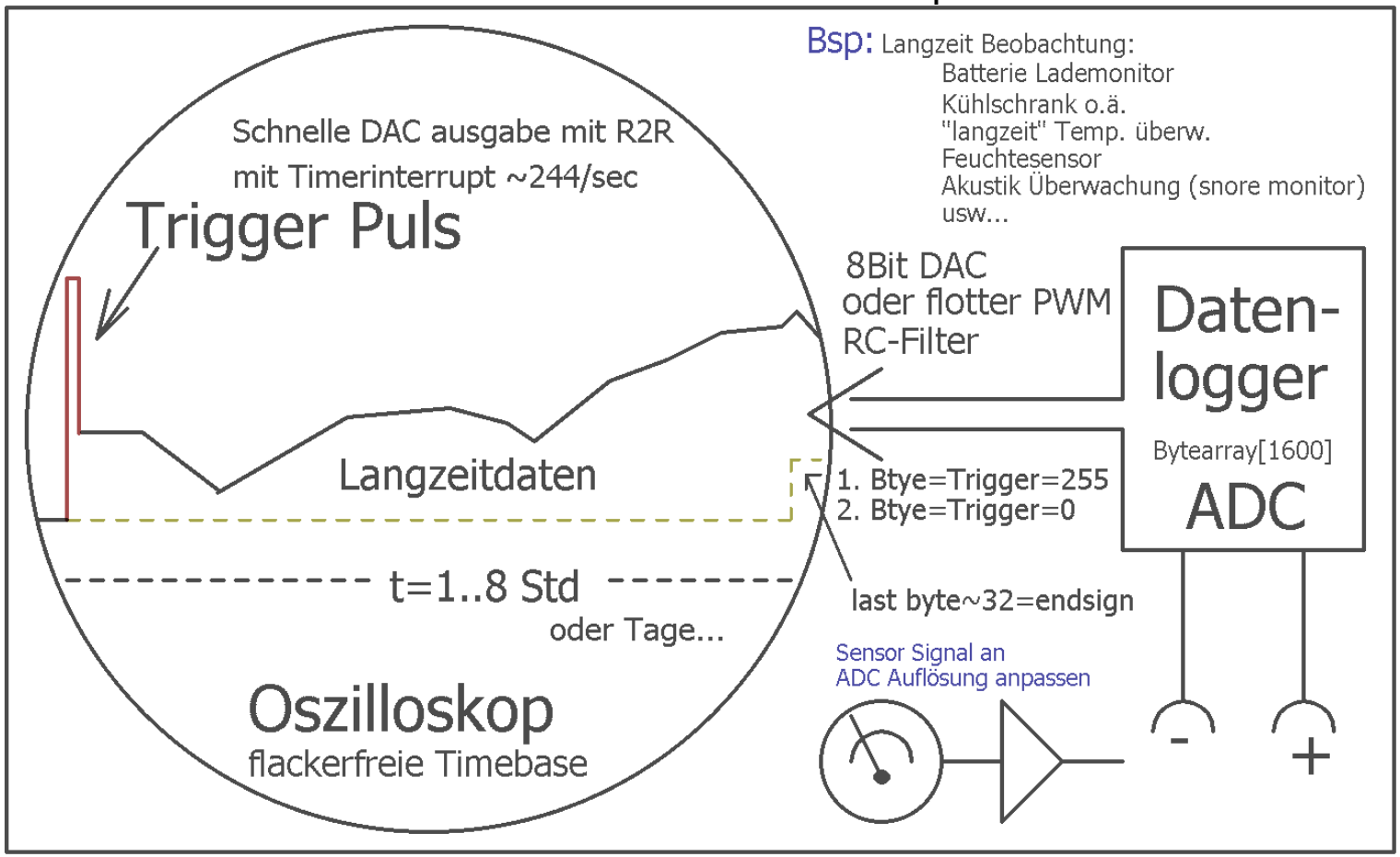
// Anwendung: u8Val = pgm_read_byte_near(**MCA_SINUS + u8Position++**);

Nützliche R2R Anwendung:

Projektidee:

Datenlogger für Langzeitdaten.
Ausgabe über Oszilloskop

Bsp: ADC liest periodisch Daten und legt diese normiert in einem (Byte)Array ab.
DAC im Timer Interrupt schreibt mit Trigger Puls diese Daten permanent flackerfrei an ein Oszilloskop.



Einstellbare Logger Zeit

Reset

Auto. ADC Skalierung: Wenn mögl. = Max-Min an Auflösung Anpassen.

Code:

```
//#define DATA_ARRAY_SIZE  1600
uint8_t gu8aDataLog[ DATA_ARRAY_SIZE ];
uint16_t gu16DataParser=0;
float gf_DataMax = DATAMAX_DEFAULT;            //für bessere ‚sicht'Auflösung
float gf_DataMin = DATAMIN_DEFAULT;            //für bessere ‚sicht'Auflösung
ISR( TIMER0_OVF_vect )    //Dieser Interrupt Vector steht in der „iom328p.h"
{
DataToOscilloscope();  Stabil und Regelmäßig alle Daten schreiben
};

void ClrDataArray(void)    // Init Phase
{
uint16_t u16NN;
for(u16NN=0; u16NN < DATA_ARRAY_SIZE; u16NN++)
        {
        gu8aDataLog[ u16NN ] = 0;    };
        gu16DataParser=0;
        gu8aDataLog[ gu16DataParser++ ] = 255;          //Trigger Puls
        gu8aDataLog[ gu16DataParser++ ] = 0;
        gu8aDataLog[ DATA_ARRAY_SIZE-1 ] = 16;          // END Sign
        };
```

```
void SampleData( float fData, float fDataMin, float fDataMax )
{               //Normiere auf DATAMAX/MIN VALUE & Auflösung R2R
gu8LastData  = (uint8_t)round( (fData - fDataMin) * (256.0/(fDataMax-fDataMin)) );
if( gu16DataParser < (DATA_ARRAY_SIZE - 2) ) // Vorletztes Zeichen
                {
                gu8aDataLog[ gu16DataParser++ ] = gu8LastData;
                }
else
                {                                   // Der Parser ist jetzt immer an vorletzter Stelle
                PushArraytoLeft();
                gu8aDataLog[ gu16DataParser ] =  gu8LastData //immer ins „vorletzte“ Feld
};
```

```
void DataToOscilloscope( void )
{  // Quick Ausgabe an R3R
uint16_t u16NN;
for(u16NN=0; u16NN < DATA_ARRAY_SIZE;
u16NN++)
  { // Daten irgendwie an Port(s) senden
   Use8Bit( gu8aDataLog[ u16NN ] ); //R2R
  };
};
```

```
void PushArraytoLeft(void)
{// ERHALTE Trigger und Platz für letzten Eintrag
uint16_t u16NN;
for(u16NN=2; u16NN < (DATA_ARRAY_SIZE-1);
u16NN++)
                {
                gu8aDataLog[u16NN]=
                gu8aDataLog[u16NN+1];
                };
};
```

Kommunikation mit der Außenwelt

Die **serielle (UART) Schnittstelle**

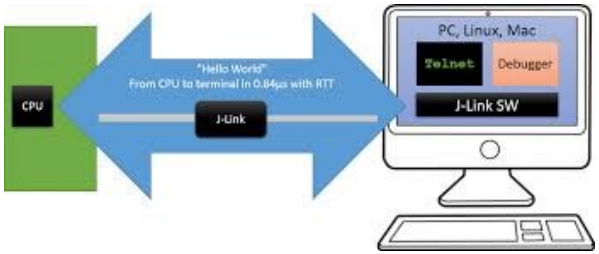
Aus der Frühzeit des IT:

.welche mitnichten aus der Mode ist.

Zuverlässig, leicht zu verstehen und anzuwenden.
Es verbindet die unterschiedlichsten Welten miteinander.
→Unterschiedliche Betriebssystem Welten,
→ beliebige! Mikrocontroller mit
→ beliebige andere Maschinen, durch einen
„globalen, gemeinsamen Zeichenstandard→ ASCII“

ASCII = American Standard Code for Information Interchange

Kommunikation mit dem Mikrocontroller



Es gibt eine „gemeinsame Welt“ zwischen allen Betriebssystem, Programmier-Sprachen und der Elektronik.

„TEXT“

Ein „A“ ist immer ein „A“. Egal ob vom Menschen via Terminal, oder von einer „Text-Sendemaschine“ wie dem Mikrocontroller oder PC. Genauer:
In einer uralten „**ASCII**-Tabelle“ ist <u>jedem</u> Zeichen eine Zahl zugeordnet. „**A“=65**
Tipp: Halte die ALT-Taste und gebe im Nummernblock 65 ein →„A“.
Ein „Befehlsinterpreter“ erlaubt Kommandos zu Senden, Daten zu empfangen.

Bsp.: Mit einem „**Terminal-Programm**“ kann man Text via serieller Schnittstelle senden und empfangen. Dies mit einer einstellenbaren Bitrate→ **115200*,8,n,1***
Die **Paramter** heißen: **BAUD** (Bits/Sekunde) meist 9600 oder **115200**
+ **8 DatenBit**, **1 Stopp Bit**, No Parity, No Xon/Xoff)
Bsp: Ich sende ein Kommando **„PING“** und bestätige das mit **CR→**
Also Antwort kommt vom Mikrocontroller **„PONG“**

Irgendwas passiert Irgendwann → Asynchrone Ereignisse

Eine (unter vielen) bewährten Methoden ist es, ein Mikrocontroller Programm so vorzubereiten, dass es „automatisch“ alle „irgendwann“ langsam **eintröpfelnden Zeichen** an der seriellen Schnittstelle **(UART)**
der Reihe nach einsammelt und der Reihe nach ablegt.

Dazu gibt es Vereinbarungen:
Dies tut der µC so lange, bis ein Wagenrücklaufzeichen „Carriage Return“ als Zeichen empfangen wird.
Da ja in der **ASCII-Tabelle** jedem Zeichen eine Zahl zugeordnet ist, ist die die Zahl **13=CR**

Dieses **CR=13** → wird als End-Of-Text Signal verwendet. →**ETX**
Wir haben aber kein Start-Of-Text Signal → **STX** ??
Nun, das geht deshalb meist gut, weil man ja weiß, dass nach einem Ende ein Start kommen muss.

Es gäbe sogar „echte **STX und ETX** Signale die im **ASCII-Code** reserviert sind.
STX = 2
ETX = 3

Aber um bequem mit **Terminal-Programmen** arbeiten zu können, die einen Wagenrücklauf als Signal kennen, nehmen wir die 13=CR
„Ping<13>

1. P
2. Pi
3. Pin
4. Ping
5. Ping<13>

USASCII code chart

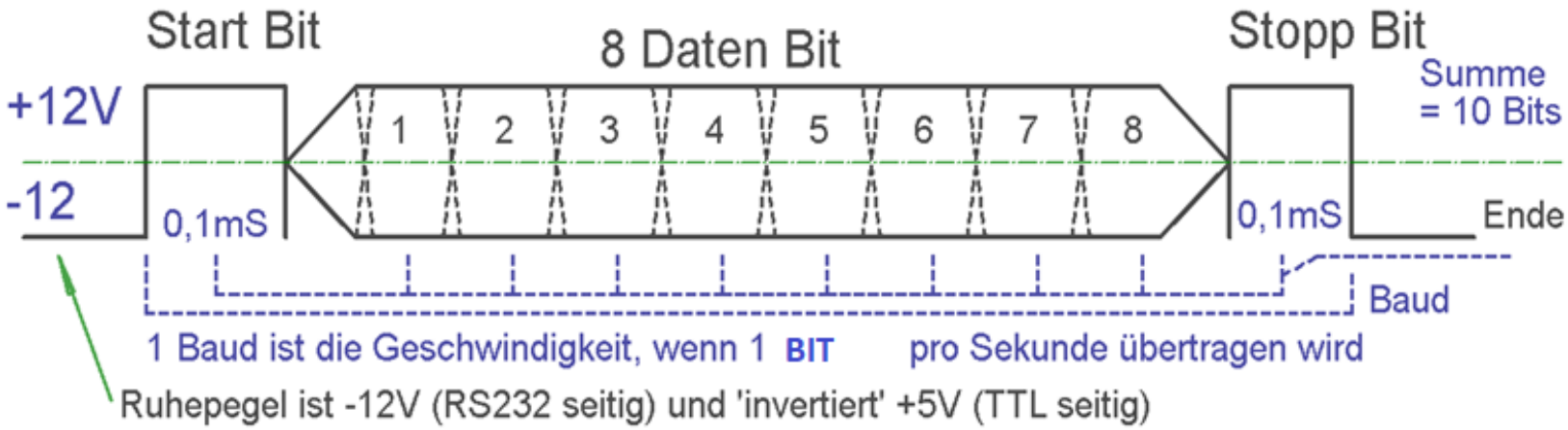
b_7 b_6 b_5 →					0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Column → / Row ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Wie bekommen wir ‚Human lesbare' und sonstige Daten in die Maschine ?

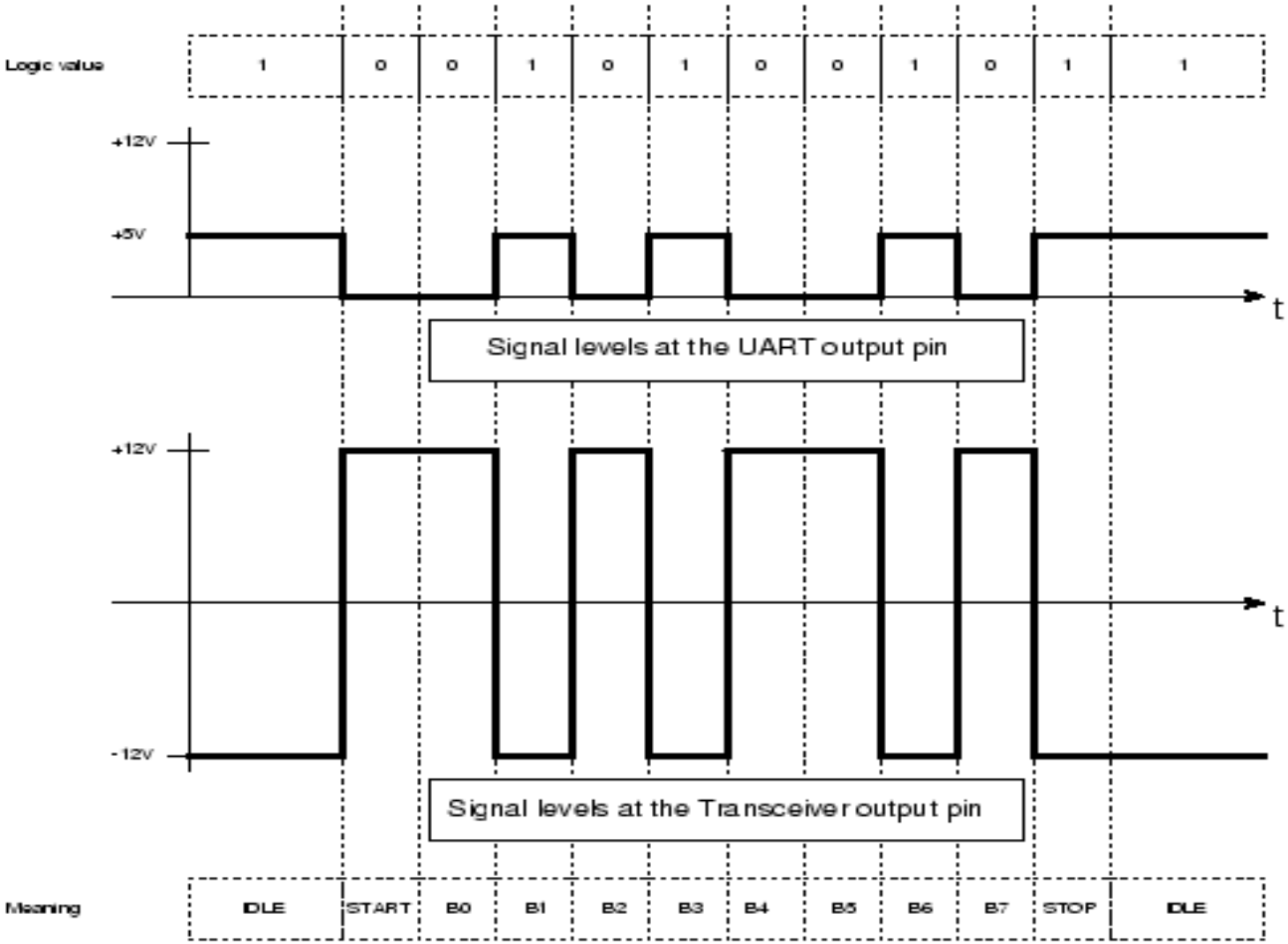
Die Schnittstellen-Techniken sind vielfältig:
Spannungspegel gesteuert, Strompegel gesteuert, parallel, seriell, Codierungen, Protokolle. Doch am meisten hat sich die **Serielle RS232** Übertragung bewährt.
TTL wird invertiert
TTL 0V= +12V **BAUD** = 9600..**115200**, Kein Handshake, keine Parität, Xon/Xoff
TTL 5V = -12V

Zeitlich von links nach rechts

technischer Signalpegel der Seriellen Schnittsstelle



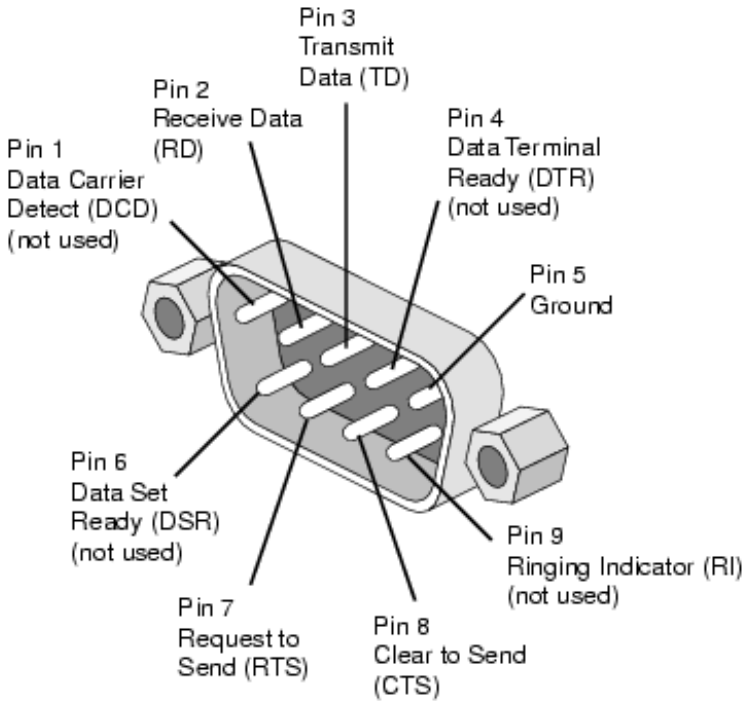
RS232 Transmission of the letter 'J'



Die serielle Schnittstelle

→ hier die PC Stecker Seite:

seltener Geworden, aber immer noch wichtig und notwendig zu verstehen.
Wenn auch nur als virtuelle serielle **COM-Schnittstelle**, meist mit dem FTDI232 Baustein, oder den weniger geliebten chinesischen Klonen wie **PL2303_Prolific**.

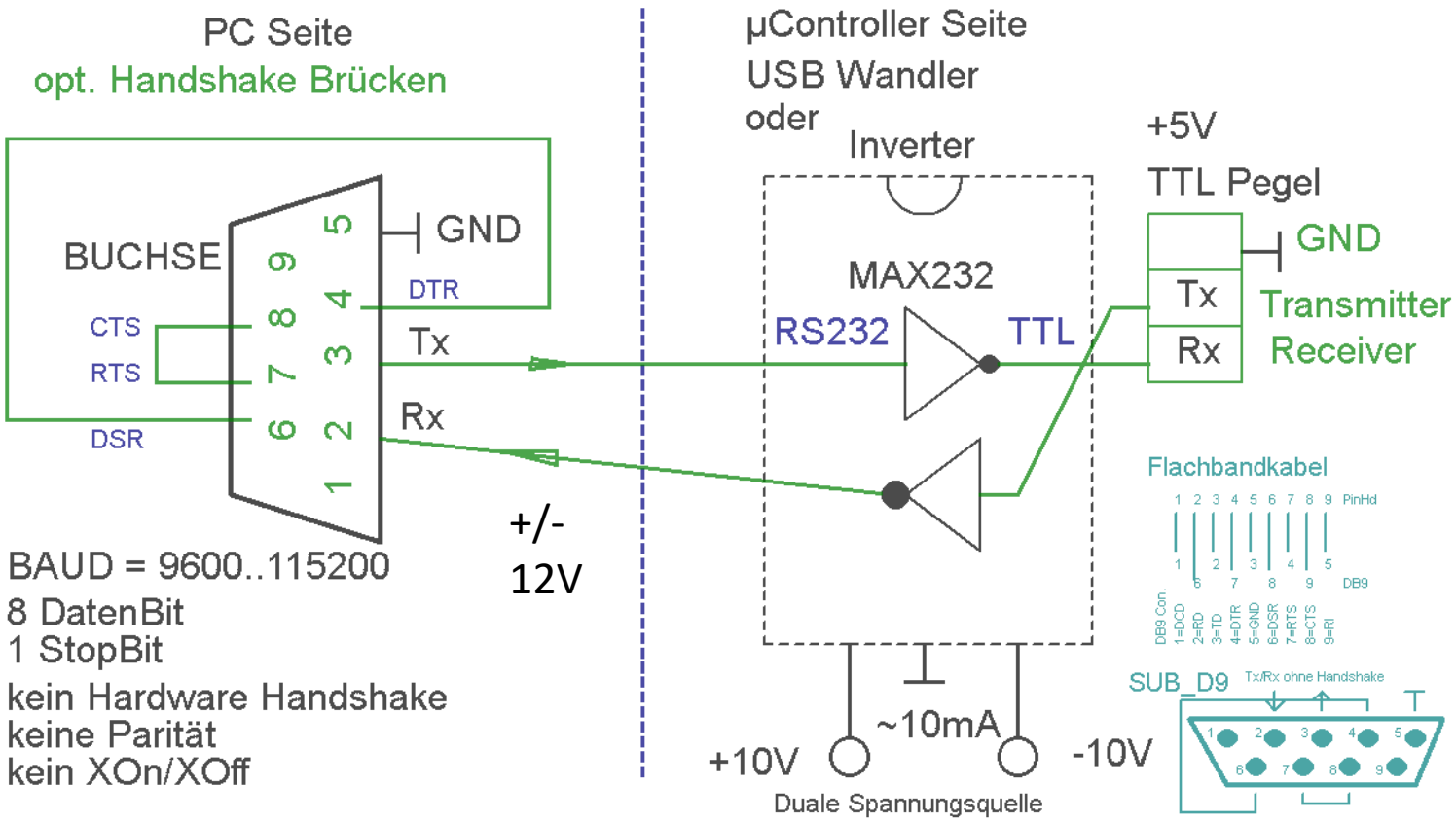


Wir brauchen meist nur
Pin 3 Tx = Transmit Data
Pin 2 Rx = Receive Data
Pin 5 GND = 0V

Achtung:
Pegelwandlung und Invertierung
zu TTL erforderlich

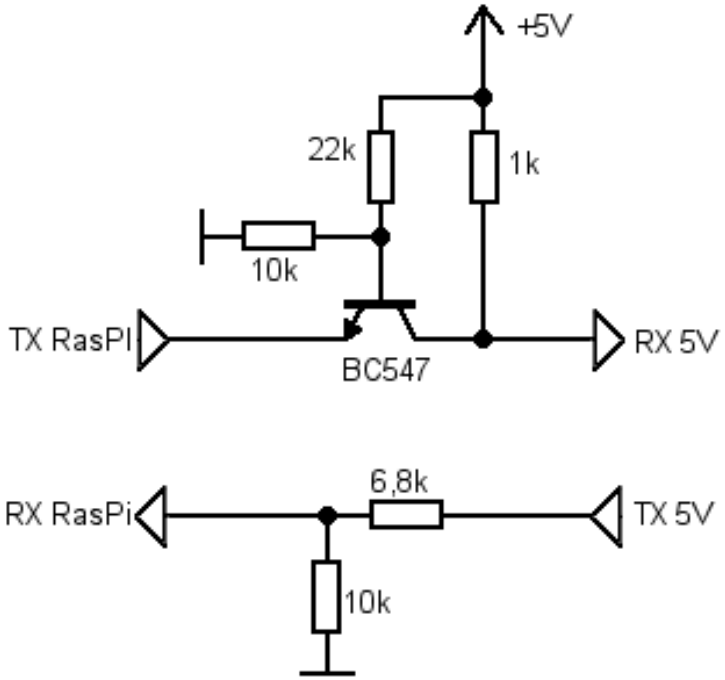
Selten muss man die
Handshake Leitungen brücken.
DSR-DTR
CTS-RTS

UART - TTL Signal → RS232

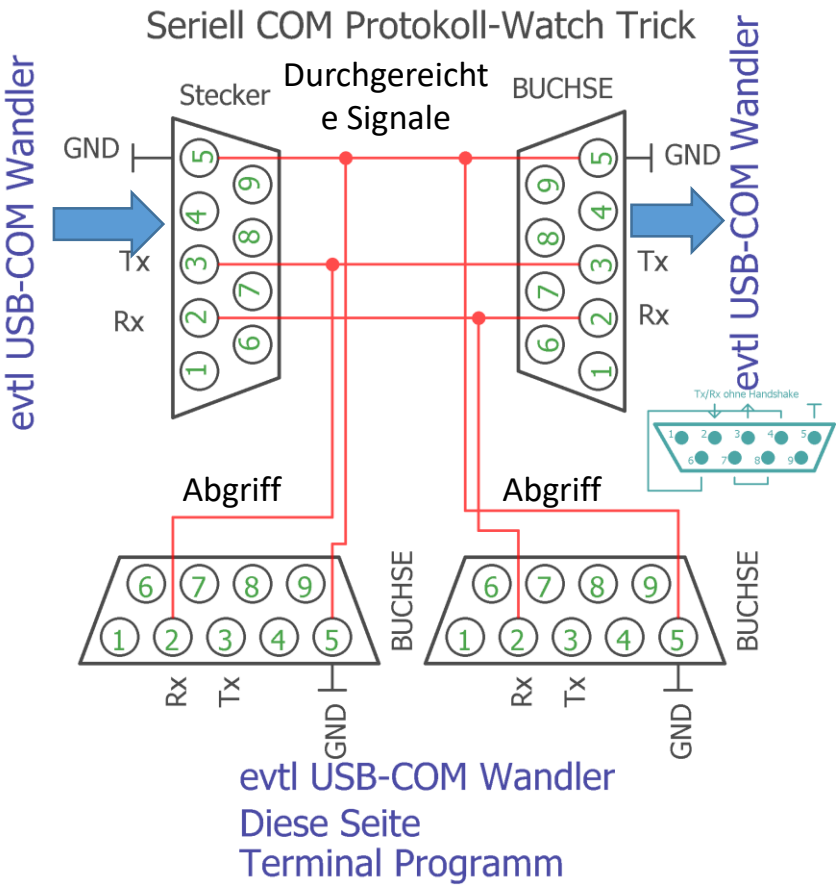


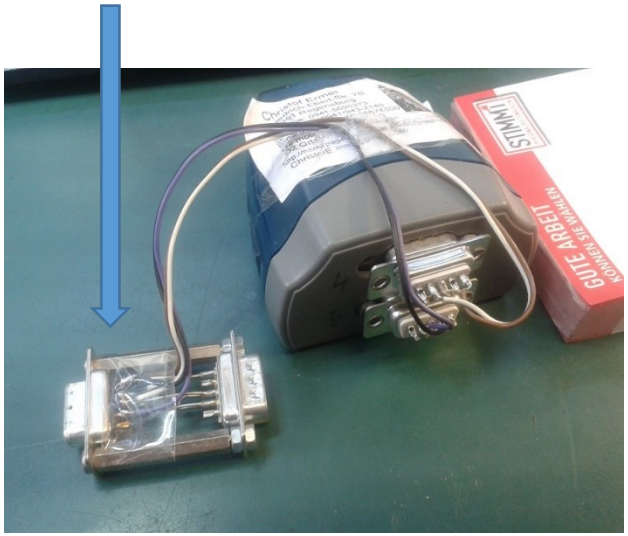
Merke:	TTL	RS232 (=invertiert)	USB
Logisch 0 =	5V	-12V	nicht direkt messbar.
Logisch 1 =	0 V	+12V	

Pegelanpassung 5V TLL an 3.3Volt Partner
z.B. Arduino (5V TTL Pegel) an Raspberry PI (3.3V)
(Tipp: 3.3V versteht der Arduino auch so , also kann man den Transistor einfach vergessen. Direkt ein Draht. Fertig.
3.3V->5V tut nicht weh.
Anders herum schon !! Da braucht es eine 1/3 zu 2/3 Teilung
2K und 1K, oder 10K und 6.8K, wie im Bild

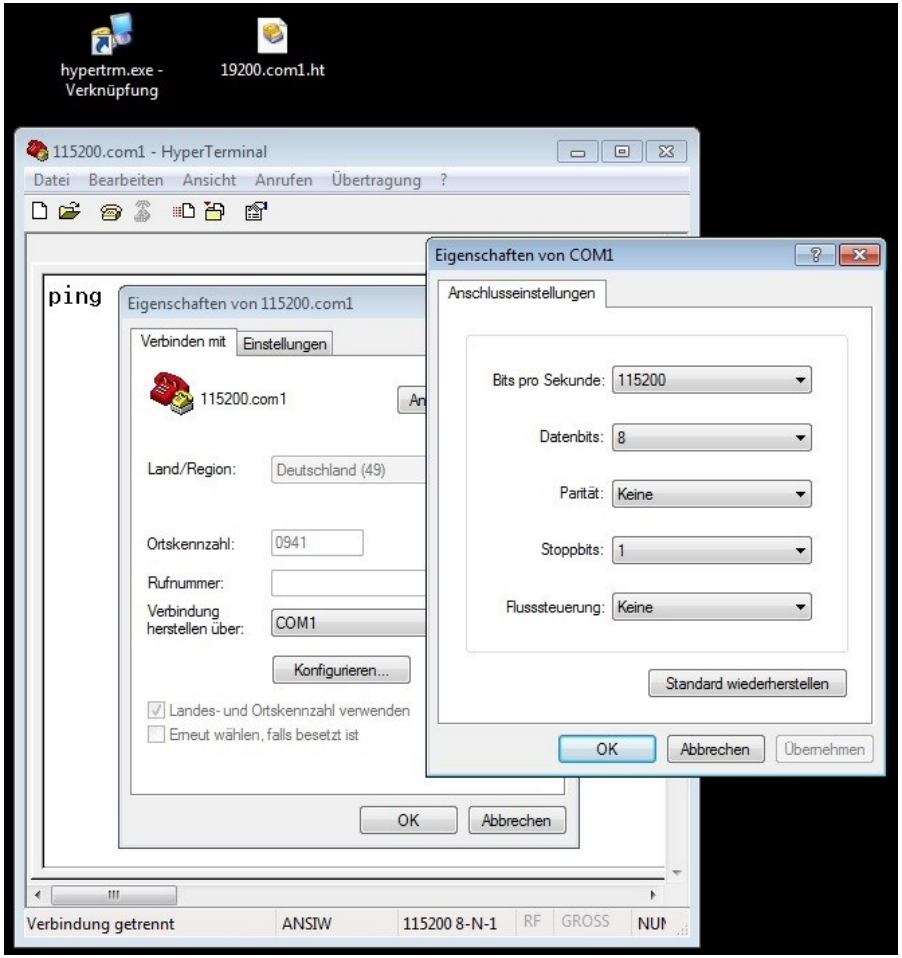


Debug Com Watch Trick. Billig und brauchbar.
Die seriellen 3-Leitungen Tx/Rx/Gnd mit Buchse-Stecker Kombination durchschleifen...
Dann die Tx, Rx, Signale auf zwei SUBD-9 Buchsen Pin-2 und GND Pin-5 leiten.
Mit w Terminal Programmen, wie Putty, mitlesen was gesendet wird





XP-Hyper Terminal zum Text Tx-Rx über die Serielle Schnittstelle (UART)



Neuere Windows Versionen haben kein Terminalprogramm mehr „Onboard“. Aber es gibt viel Freeware und das alte **XP Hyperterminal** geht (noch) recht gut.

Nach etwas Gewöhnung und Starteinstellungen läuft es prima.

Dazu kann man mehrere Voreinstellungs-Dateien vorbereiten.

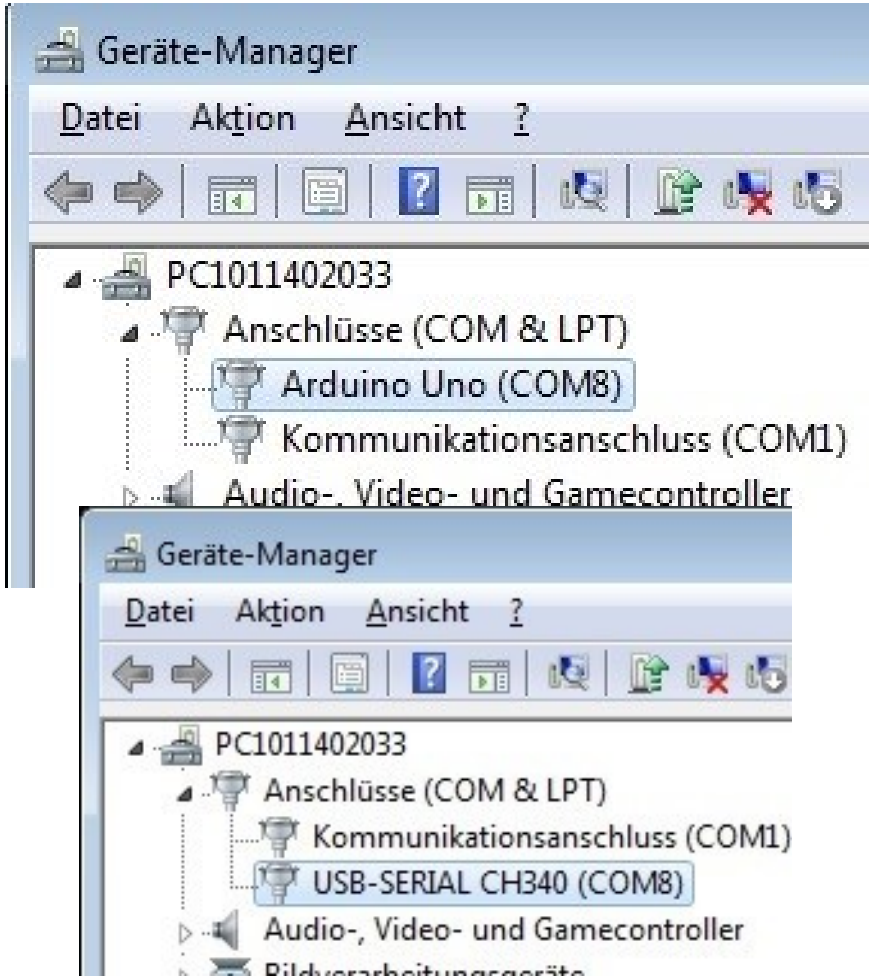
Bsp.: **115200.com.ht**

Parameter:

BAUD:	9600..115200
Datenbits:	8
Parität:	Keine
Stopbits:	1
Flusssteuerung:	Keine

Ermitteln des COM-Ports

BAUD →115200.com8.ht



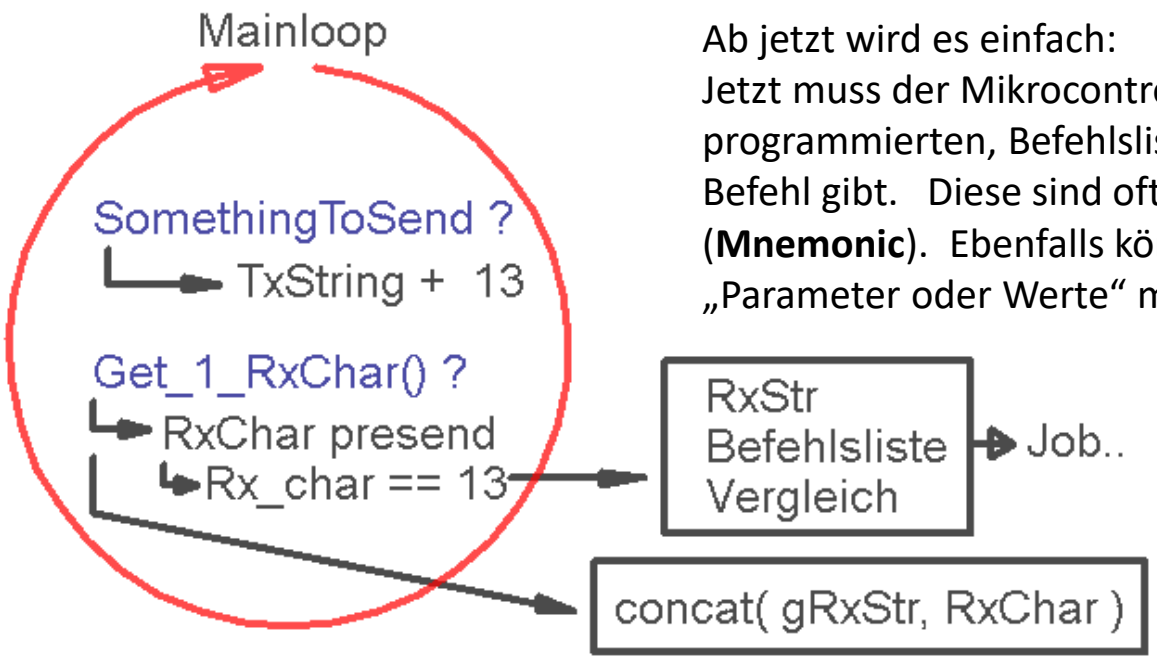
Nach der Installation des ARDUINO Boards:
Ein USB Treiber ist erforderlich.
Original FTDI-232 Treiber
(vor Version 12 oder danach)
Oder USB CH340 (China-Klon)

Dann kann das benutzte COM-PORT ermittelt werden.
Tastenkombination:
WINDOWS-FENSTERTASTE + PAUSE
=> Geräte Manager

Sollte die COM-Nummer unbrauchbar sein kann diese geändert werden.
Rechte Maustaste → Eigenschaften, Anschlusseigenschaften, Einstellungen..

Wie bekommen wir human lesbare Daten von außen in die Maschine ?

Die Text-Sammelmaschine im Mikrocontroller als Befehlsinterpreter



Ab jetzt wird es einfach:
Jetzt muss der Mikrocontroller nur in seiner, selbst programmierten, Befehlsliste nachsehen, ob es diesen Befehl gibt. Diese sind oft kryptisch reduziert. (**Mnemonic**). Ebenfalls könnten auf diesem Weg „Parameter oder Werte“ mitgegeben werden.

Die Kommandos werden gern in einer mit „**Komma**“ getrennten Liste geschrieben. So etwas nennt sich „**Delimiter-Liste**“ oder auch „**Mnemonic code**“.
Bsp.: „**MO, 230,230 <CR>**“
Heißt→ Motor On, mit 230 für links, 230 für rechts **+ CR**

Bsp.:

Ein GPS –Modul liefert Seriell einen Text-String mit folgendem Aussehen:

eine professionelle **GPS** Datenübertragung im NEMA-Protokoll:

$GPGGA,084116.999,5200.0000,N,03000.0000,E,1,05,1.8,501.6,M,,,,0000*09„

Darin sind die einzelnen Elemente streng durch einen „Delimiter“ -_> , getrennt.
Die Reihenfolge und die Bedeutung liegt sinnvollerweise in einer Struktur
Siehe:
https://homepages.uni-regensburg.de/~erc24492/GPS_Secrets/GPS_Secrets.html

Und diesen String wollen wir empfangen und in seine Elemente Zerlegen.

$GPGGA,084116.999,5200.0000,N,03000.0000,E,1,05,1.8,501.6,M,,,,0000*09,,

TOKENS:

GPGGA Global Positioning System Fix Data

123519 Fix taken at 12:35:19 UTC

4807.038,N Latitude 48 deg 07.038' N

01131.000,E Longitude 11 deg 31.000' E

1 Fix quality: 0 = invalid

1 = GPS fix (SPS)

2 = DGPS fix

3 = PPS fix

4 = Real Time Kinematic

5 = Float RTK

6 = estimated (dead reckoning) (2.3 feature)

7 = Manual input mode

8 = Simulation mode

08 Number of satellites being tracked

0.9 Horizontal dilution of position

545.4,M Altitude, Meters, above mean sea level

46.9,M Height of geoid (mean sea level) above WGS84 ellipsoid

(empty field) time in seconds since last DGPS update

(empty field) DGPS station ID number

*47 the checksum data, always begins with *

UART-Schnittstelle und Mnemonic Steuerung

Ab sofort können wir Befehle und Daten einschleusen und so den µController dazu veranlassen auf Kommandos zu reagieren.
siehe „**Befehle.c“** (File→ in meiner Homepage)

```
if(!strcasecmp_P( gsCmd.ucaCmd, PSTR("TO")))            // Flags setzen
        {
        gu8Status |= STATUS_TRIGGER_OUTPUT;
        return;
        };

if(!strcasecmp_P( gsCmd.ucaCmd, PSTR("TF")))            // Flags löschen
        {
        gu8Status &= ~STATUS_TRIGGER_OUTPUT;
        return;
        };

if(!strcasecmp_P( gsCmd.ucaCmd, PSTR("SPORTC")))     // Set PORTC
        {
        PORTC = gsCmd.i16CmdVal_1 & 0xFF;
        return;
        };
```

Hilfsfunktionen, die ständig gebraucht werden, sobald man Text verarbeiten muss.

Es ist sinnvoll sich ein „**Set**“ Funktionen in eine immer wieder verwendete Datei *.c *.h zu schreiben, die einen hohen Nutzwert haben, stabil und Fehlersicher sind, und einen vor Vergesslichkeit schützt. (.,..wie war das wieder?)

Dazu gibt es **ein immer gleiches Set** an Variablen, Definitionen, Funktionen die man im Schlaf und auswendig kennen soll.

2 General Purpose String Arrays (Immer verwendbar → für strcpy und strcat usw…)

```
#define STRMAX 63      //das reicht für die meisten Fälle
#define NUMMAX 31
```

Global definiert:

```
char gcaStr[STRMAX + 1];          //+1 for the 0
char gcaNumStr[NUMMAX + 1];       //+1 for the 0
```

Im zugehörigen HEADER File noch dem Rest des Programmes bekannt geben:

```
extern char gcaStr[STRMAX + 1];
extern char gcaNumStr[NUMMAX + 1];
```

Um diese

Dazu basteln wir uns Basis-Funktionen:

Vergiss nicht
#include <stdlib.h>

```
char*  IntToNumStr(int16_t lVal)
{
return itoa(lVal, gcaNumStr, 10 );
};

char*  UIntToNumStr(int16_t lVal)
{
return utoa(lVal, gcaNumStr, 10 );
};

char*  LongToNumStr(int32_t lVal)
{
return ltoa(lVal, gcaNumStr, 10 );
};
char* ULongToNumStr(uint32_t ulVal)
{
return ultoa(ulVal, gcaNumStr, 10 );
};

char*  FloatToNumStr(double fVal)
{//[-]d.ddd
return dtostrf(fVal, FLOAT_PRAEDIGITS_MAX, FLOAT_DIGITS_MAX, gcaNumStr );
};
```

```
#define FLOAT_PRAEDIGITS_MAX  7
// Zeichen insgesamt  123.456
#define FLOAT_DIGITS_MAX  3
//Nachkommastellen
```

Nutzvolle UART Programmierwerkzeuge im µC
→ AVR Typisch

Erinnere Dich: String ist und verbraucht ohne „Sonderbehandlung“ RAM

Wissensbasis:

```
uart_puts(„Vorsicht String lieg tim RAM“  );   //String ausgeben
uart_puts_p( PSTR(„String ist im Programmspeicher") );   //String ausgeben
```

Einfache, RAM basierte Stringausgabe:

```
uart_puts( gcaNumStr);
uart_putc(13); // Carriage Return
uart_putc(10); // Line Feed
```

```
uart_puts( ltoa( gi32SM_Position , gcaNumStr, 10 ) );    //Long Zahl to Str
uart_puts( ultoa( TCNT1 ) , gcaNumStr, 10 ) ); ];          //ULONG to Str
```

Float Zahl als String ausgeben:

```
#define FLOAT_PRAEDIGITS_MAX 8 // Output String width  (including the '.')
#define FLOAT_DIGITS_MAX   5 // Digits nach dem Punkt;
dtostrf(fValue, FLOAT_PRAEDIGITS_MAX, FLOAT_DIGITS_MAX, gcaNumStr ); //[-]dd.ddd
```

dtostrf(double __val, char __width, char __prec, char *__s)

Aus einem Forum: Übernommen weil sehr gut erklärt:

dtostrf hat 4 Argument
double __val
char __width
char __prec
char* s

Was könnte wohl was sein?

__val dürfte klar sein.
Wenn die Funktionen für einen double eine String Repräsentierung bestimmen soll (aka ihn in einen String umwandelt), dann wird die Funktion wohl auch den doublebenötigen.
Also da wird wohl der double reingesteckt werden müssen.

Auf der anderen Seite wird die Funktion wohl auch das String-Array benötigen, indem sie die String Repräsentierung ablegen kann.
Da wir wissen, das Arrays in C immer per Pointer übergeben werden und 's' der einzige Character Pointer im Aufruf ist, wird das dann ja wohl die Angabe des Character Arrays sein.

Aber was ist __width und __prec.
Nun, da fragen wir uns doch gleich mal:
Was wollen wir denn von der Funktion?

Ja klar, sie soll eine String Repräsentierung einer
double Zahl bestimmen.

Aber wie soll die aussehen?
Soll eine
Zahl 3.141592654 auch tatsächlich so als String ausgegeben werden,
oder wäre es nicht toll, wenn man sagen könnte:
3 Nachkommastellen sind genug, ich bin also mit "3.142" zufrieden.

Wäre es nicht auch gut, wenn ich sagen könnte:
In meinem Character rray, das ich als Aufrufender zur Verfügung stelle, ist
Platz für maximal 8 Zeichen.
Das wäre schon gut, denn eine Zahl 3.141592654
mit seinen insgesamt 11 Positionen (inklusive Dezimalpunkt) wird
nie und nimmer in ein Array mit 8 Zeichen hineinpassen.

__width wird also etwas mit der kompletten 'Breite' der Ausgabe zu tun haben, während __prec möglicherweise etwas mit der Anzahl der Nachkommastellen zu tun hat.

Und genau so ist es auch:
Mittels _width legt man fest, welche Feldbreite die komplette Ausgabe belegen soll. Dabei sind alle Zeichen mit eingeschlossen. Also auch Dezimalpunkt und ein mögliches Vorzeichen.
Mittels __prec legt man fest, wieviele Stellen dieser Feldbreite für den Nachkommanteil benutzt werden können.

Bsp

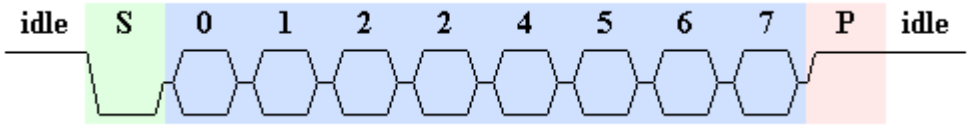
```
double  f = 2.0;
char ascii[10];

dtostrf( d, 7, 2, ascii );
```

dtostrf soll also den Wert in eine Text Repräsentierung
überführen, wobei der Text in ASCII landen soll.
Der Text soll 7 Zeichen enthalten, von denen maximal 2 die Nachkommastellen darstellen sollen.

Funktionen in den Header-Files nachzuschlagen, ist zwar für einen Fortgeschrittenen eine gute und vor allem schnelle Möglichkeit,
zu Informationen zu kommen.
Für einen Anfänger ist das aber eher nichts.
Da bietet sich eine Googel Suche an oder aber der Ankauf von Literatur.
Vor allem letzteres ist sehr zu empfehlen, da ein gutes Buch durch nichts zu ersetzen ist. Es zeigt die Funktionen und ihre Verwendung normalerweise im Kontext von Aufgaben und Übungsaufgaben und enthält auch Erläuterungen und Erklärungen die man so im Web meist nicht findet. Ganz abgesehen davon, führt einen ein Buch systematisch durch alle Bereiche einer Programmiersprache und genau an diesem Punkt fehlt es bei vielen Online Tutorials.

Möglichkeit: **SOFTWARE UART**

Siehe: https://rn-wissen.de/wiki/index.php/Software-UART_mit_avr-gcc



Läuft der AVR mit einer Taktrate von **F_CPU** und die Übertragung mit der Baudrate **BAUDRATE**, dann dauert ein Bit

F_CPU = 16MHz

Gewünschte Baudrate = 9600

F_CPU / 9600 = 16^6 / 9600 = 1666.6 Zyklen pro Bit

Das Ganze *10? = 16666.666 Taktzyklen pro BYTE {10Bit};

Der RX = Receiver-Eingang liegt an **ICP1** (InputCapture-Eingang)

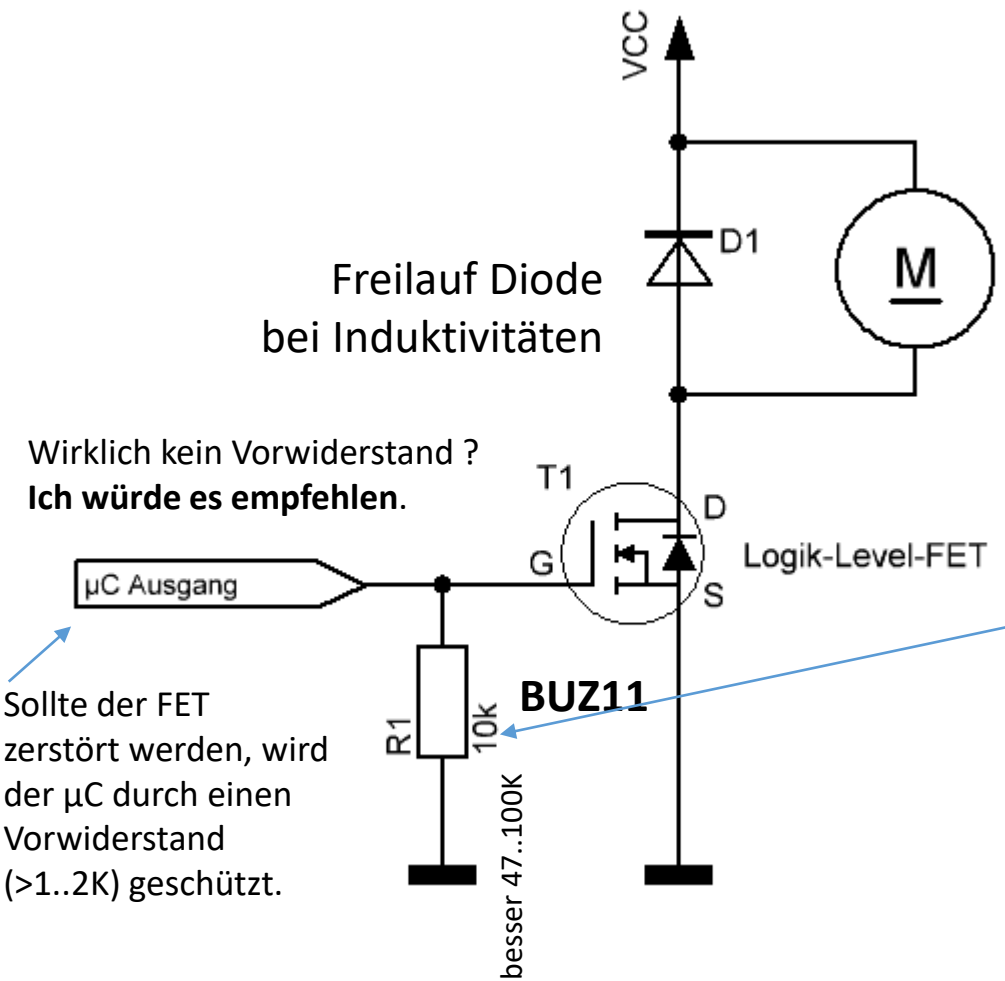
Timer 1 = CTC,

Externe Hardware ansprechen

dazu Elektronik CAD-Programm **Eagle**
Schaltpläne lesen und als
Quell-Dokumentation
für die Programmierung nutzen

Geräte ein oder ausschalten.

Wer eine LED ein und ausschalten kann….
„kann alles andere auch!“



Freilauf Diode
bei Induktivitäten

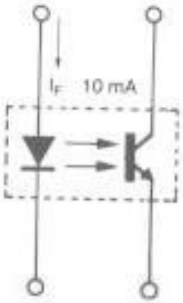
Wirklich kein Vorwiderstand ?
Ich würde es empfehlen.

Sollte der FET zerstört werden, wird der µC durch einen Vorwiderstand (>1..2K) geschützt.

Leistungs-MOS-FETs,
hier ein N-Kanal MOS-FET.
z.B. vom Typ **BUZ11**
Sind besonders einfach anzusteuern.
Dieser kann „**15 A**mpere“ schalten.
(dennoch die Verlustleitung zu beachten, also kühlen)

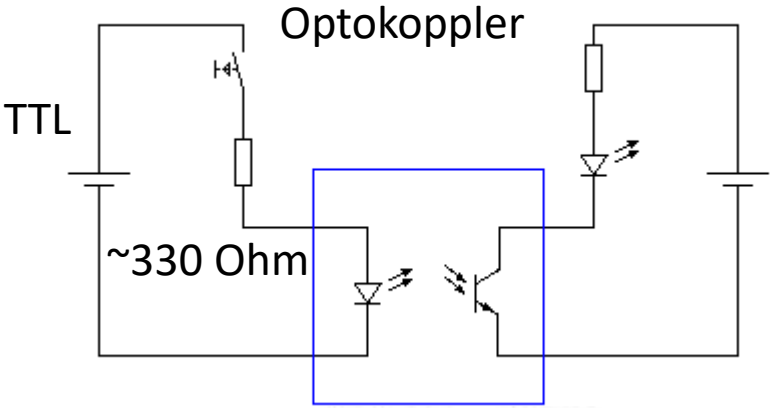
Der Gate-GND-Widerstand dient dazu, den FET *definiert abzuschalten*, wenn der **µC** aus irgendeinem Grund die Leitung nicht aktiv ansteuert.

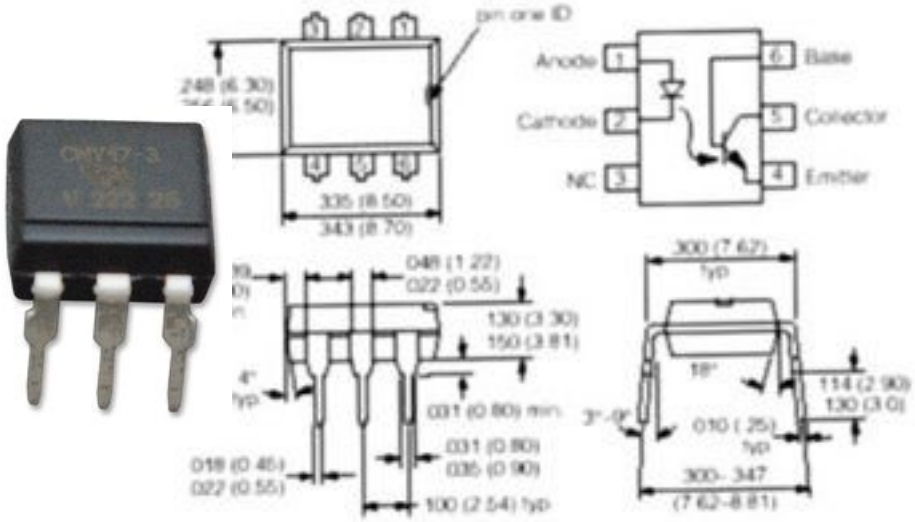
µC schaltet externe Komponenten



Anstelle von mechanischen Relais kann man Solid State Relais nutzen.
Der Vorteil ist, dass auch höhere Schaltfrequenzen nutzbar sind (Regelungen)
Der Relais-Eingang ist zudem **optisch** isoliert. (Galvanische Schutz-Trennung)

Solid State Relais. 10A

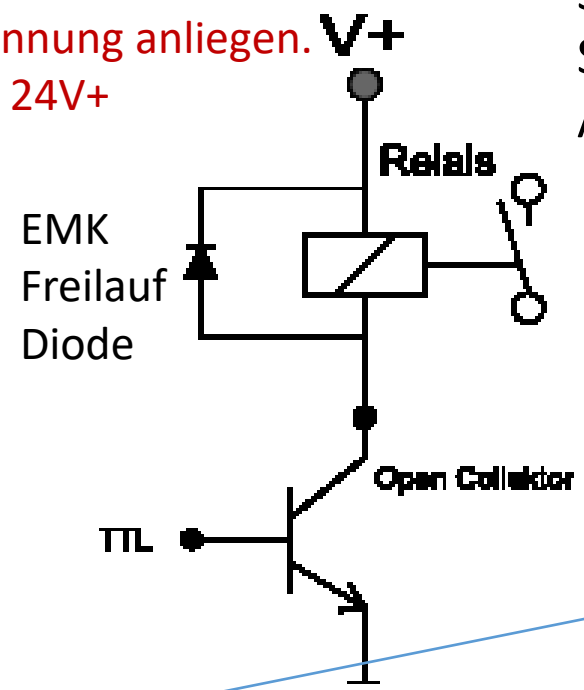


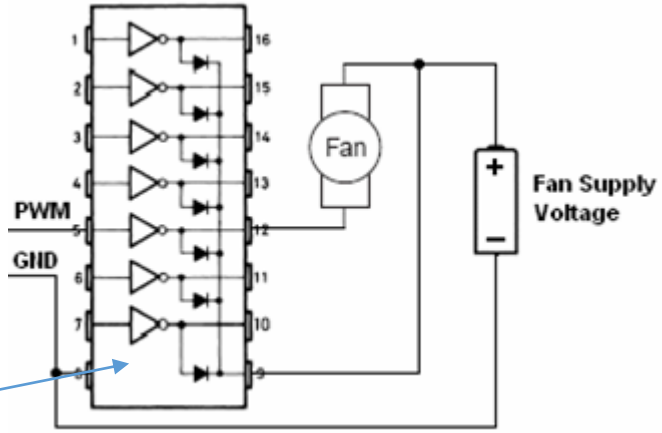


„Open Collector“ → OC Konzept

Wichtiger Vorteil:
Hier kann eine viel höhere Spannung anliegen.
z.B. 24V+

So ein „Open Collector“ ist also wie ein Stromtreiber gegen GND → **Sink Current**
Signaltechnisch ist das eine Invertierung.
Aus TTL+ wird Signal -. Daher der Kreis



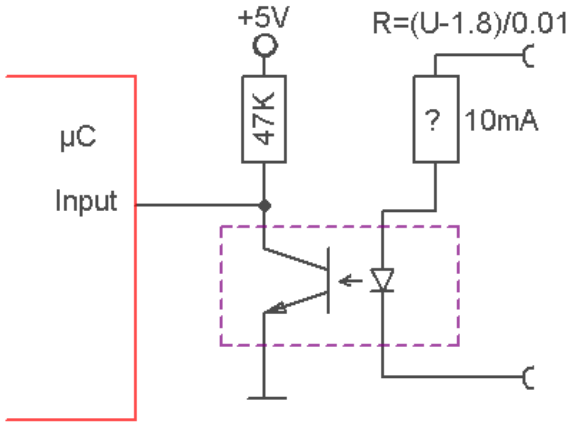


ULN2003 (7 Trans.)
besser: **ULN2803** (8 Trans)

ULN2003 „Sieben“ Transistor Array IC
So wie es im Arduino Kit enthalten ist.
Dieser hat aber „dummerweise“ **nur 7** Transistoren

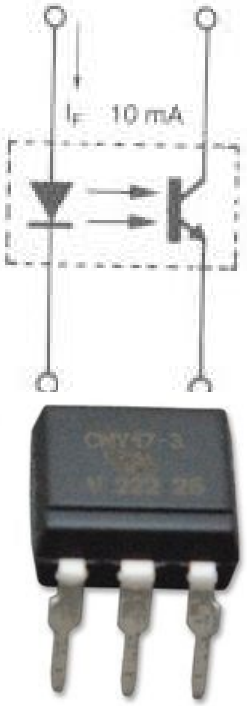
(Ich bevorzuge den **ULN2803** mit **8 OC-Transistoren)**

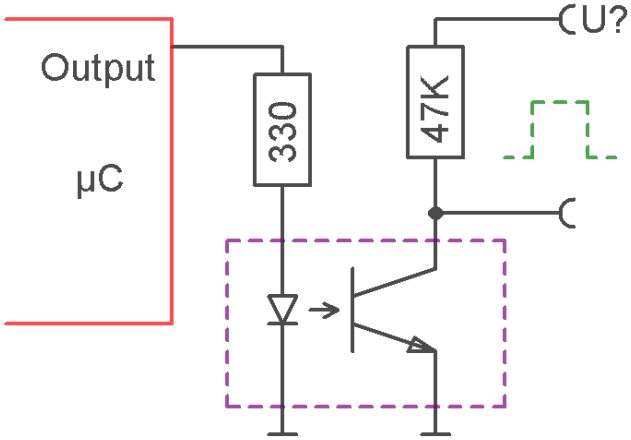
Optokoppler als In und Output

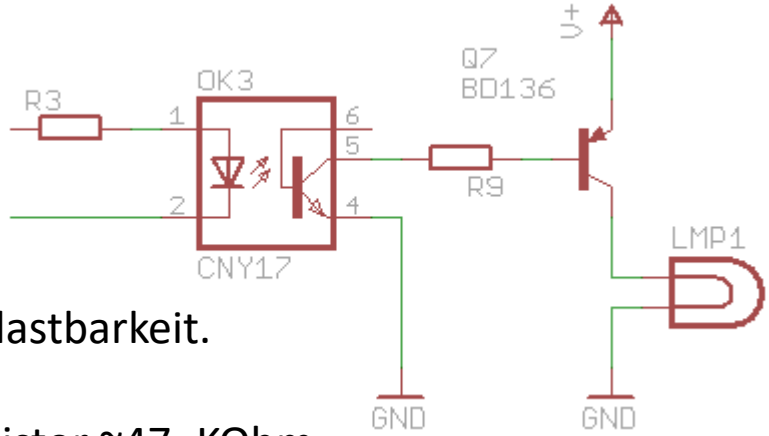


Sinn:
Galvanische Trennung von µC und Last

Optokoppler sind träge. Die Ladungsträger am Transistor bauen sich langsam ab. Deshalb sollte der Kollektor-R groß (~50K) gewählt werden. Der LED Strom sollte ~>10mA sein





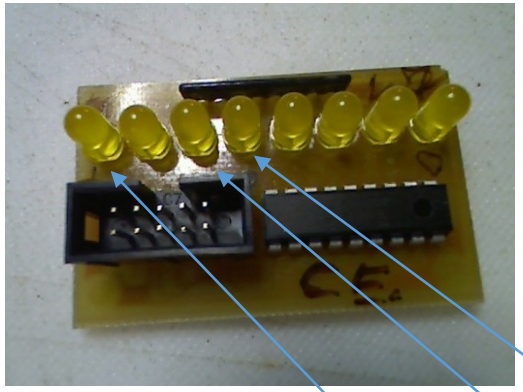


Reiner Signalausgang ohne große Belastbarkeit.
Keine GND Trennung
Dimensionierung: LED ~220..R, Transistor ~47..KOhm

...weil es ab jetzt hilfreich sein kann.

Einfache Analyse und Fehlersuche/Debugging Methoden

Es gibt selbstverständlich gute Debug-Techniken **wie JTAG**. Leider erfordert dies dazu fähige **Extra-Hardware** (Dragonboard etc.), und einen **µC** der eine **JTAG** Schnittstelle hat, und/oder ein **1-Wire Debug,** und eine Software der man sich unterwerfen muss, *und..und..und..* Und es geht dennoch eher zäh! Es geht auch mit Methode und Hirn, und oft viel einfacher und besser…..:



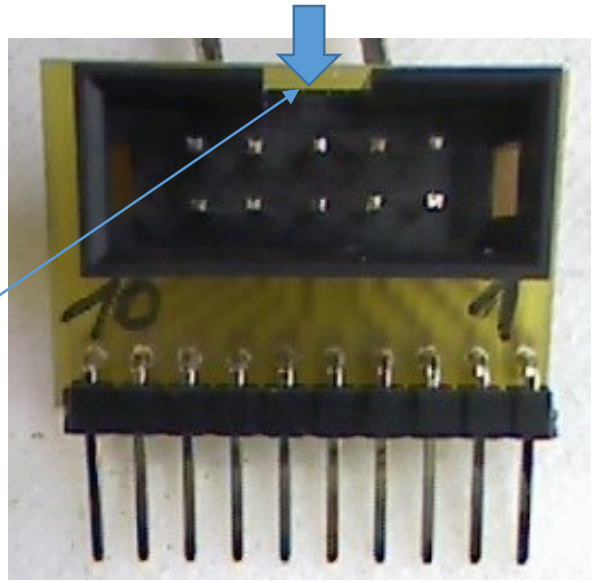
Meine Lieblingsmethode ist eine einfache **8-Bit LED Leiste**.
Ab sofort kann **Online in Echtzeit** jedes Byte angezeigt und somit überwacht werden
PORTx (LEDPort) = **MCUCR**;
oder PORTx = g**u8StatusFlag**s;
oder PORTx = (1<< (u8NN++) % 8); //Lauflicht

MCU Control Register – MCUCR

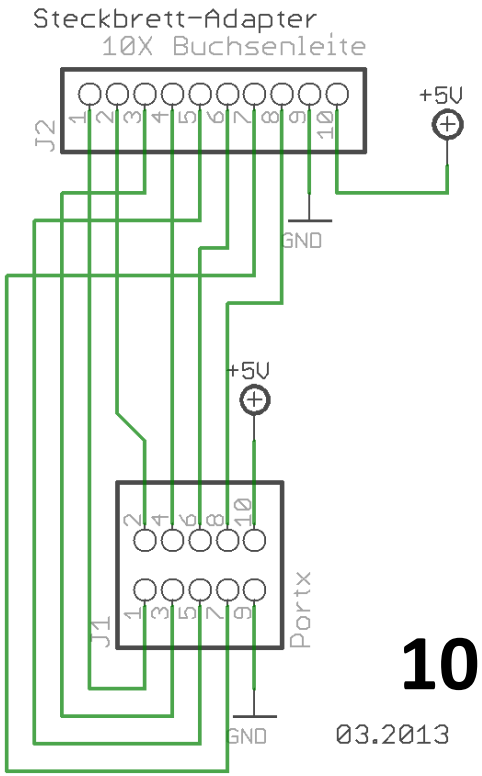
The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

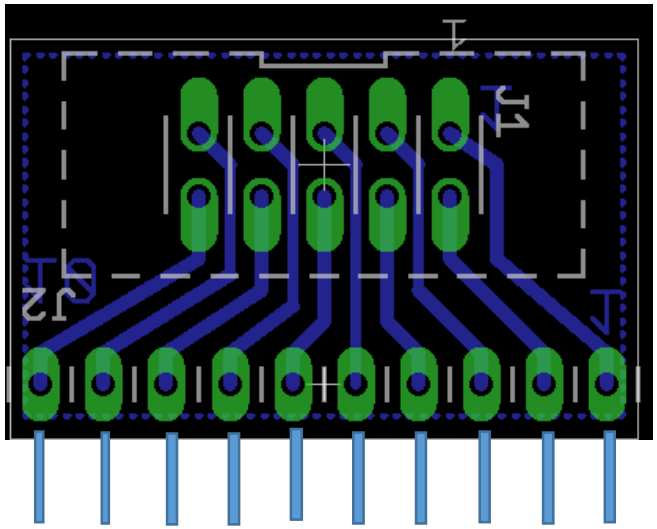
Steckbrett Adapter
→ 10 Poliges Flachbandkabel



Nase beachten

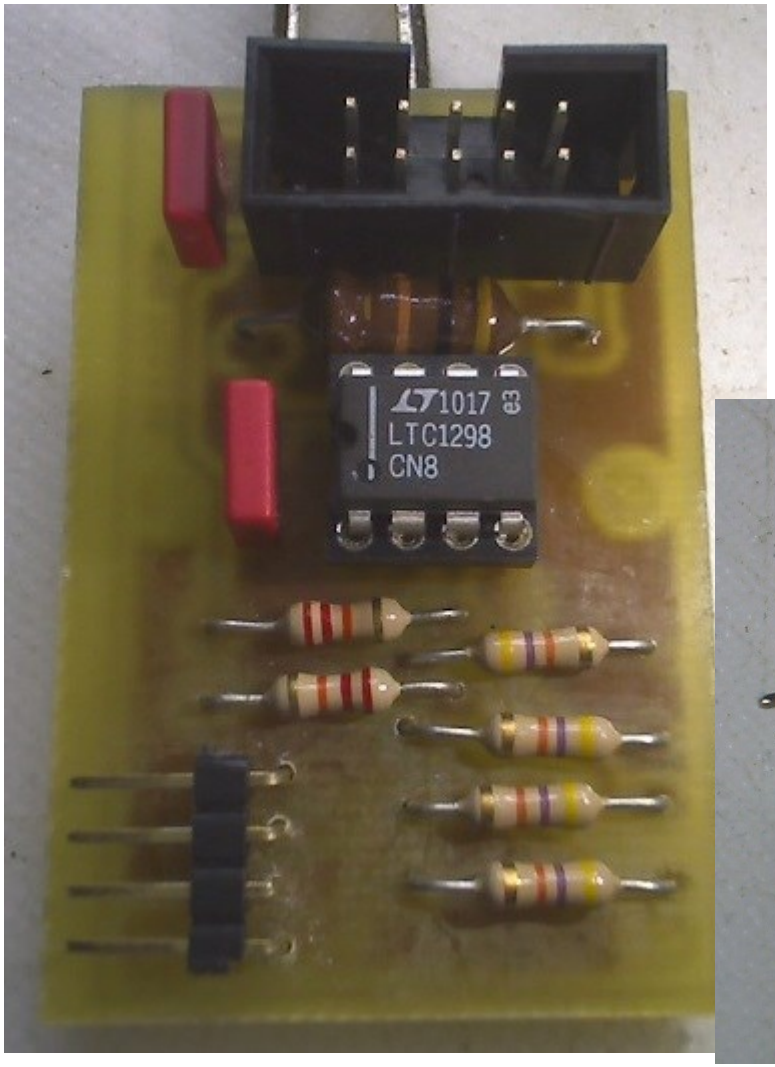


Von oben

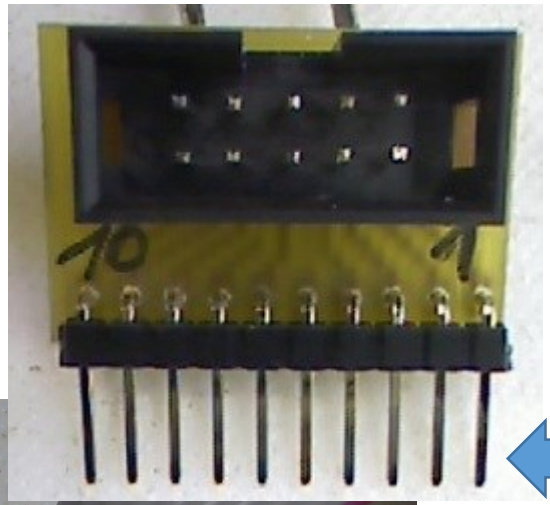


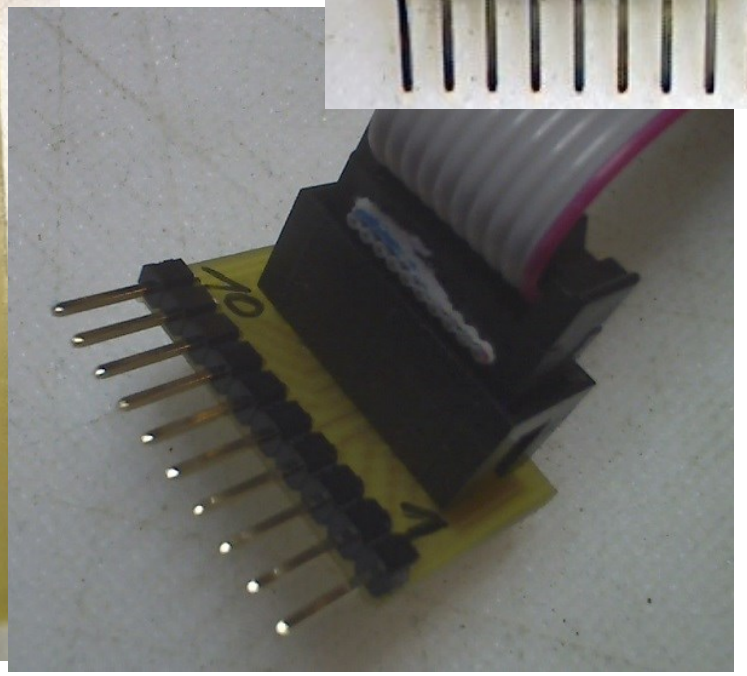
03.2013

Praktische Flachbandkabel-Anbindungen externer kleiner Platinen an ein Steckbrett:

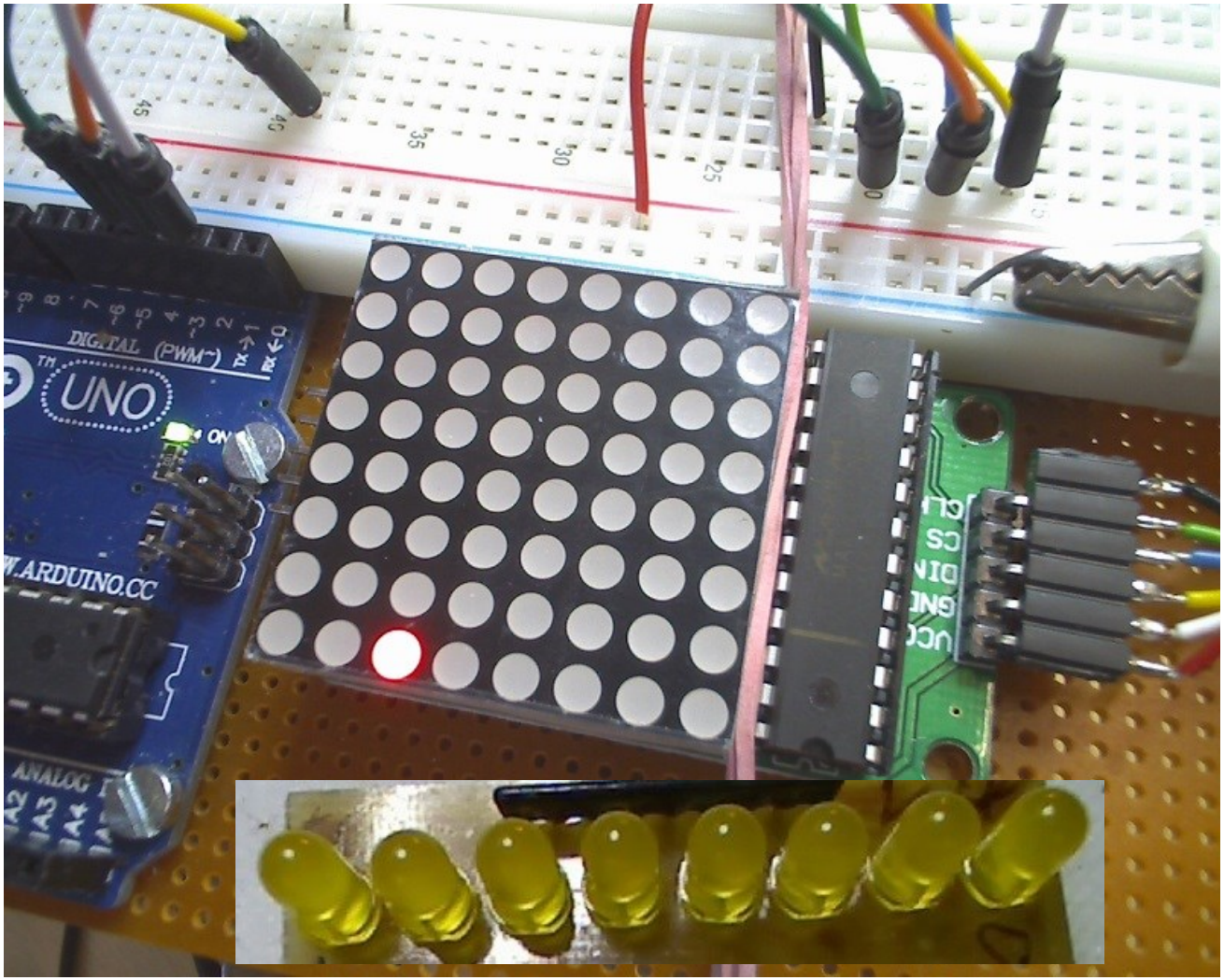


Immer auf die „**1**“ und GND=(9) +5V=(10) achten.





8x8 LED Matrix als universelles Anzeige- und Debugging-Instrument



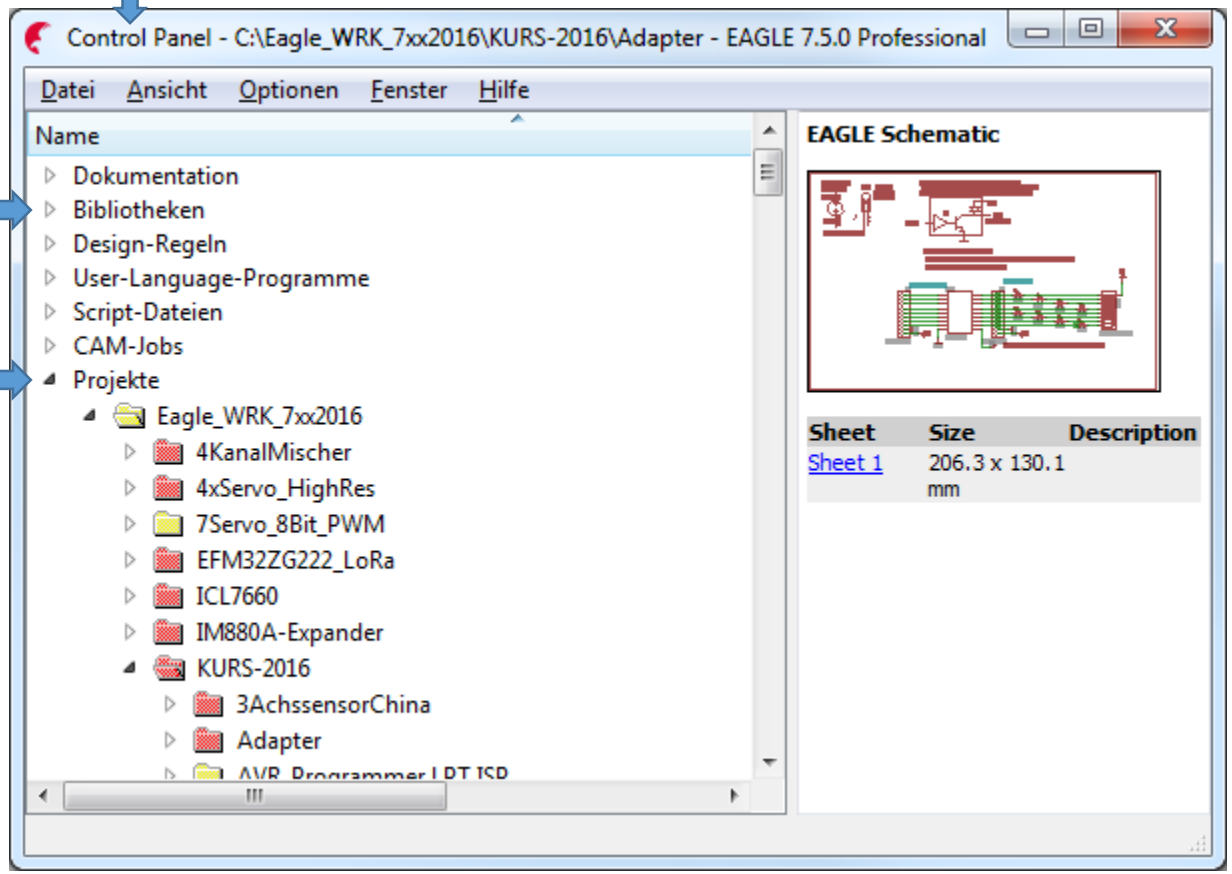
Da der Arduino kein vollständig freies 8-BIT Port hat, ist eine 8-Bit-LED-Leiste nicht ideal.
Auf einer 8x8 Matrix kann jedoch quasi 8 * eine 8Bit LED-Leiste realisiert werden.
Die Anbindung erfordert 5 Leitungen
3 Datenleitugen
DIN / CS / CLK
und **+5V / GND**
WIE ?
Das sehen wir jetzt…

Schaltpläne lesen und erstellen mit Eagle

Eagle ist ein ‚altes' Layout Programm und in Europa verbreitet.
Es kann mit Skript Text gesteuert werden. Was bei präziser Bibliothekserstellung hilfreich ist. Es besteht aus 4 Teilen. Aber so genau ist das gar nicht zu sagen.

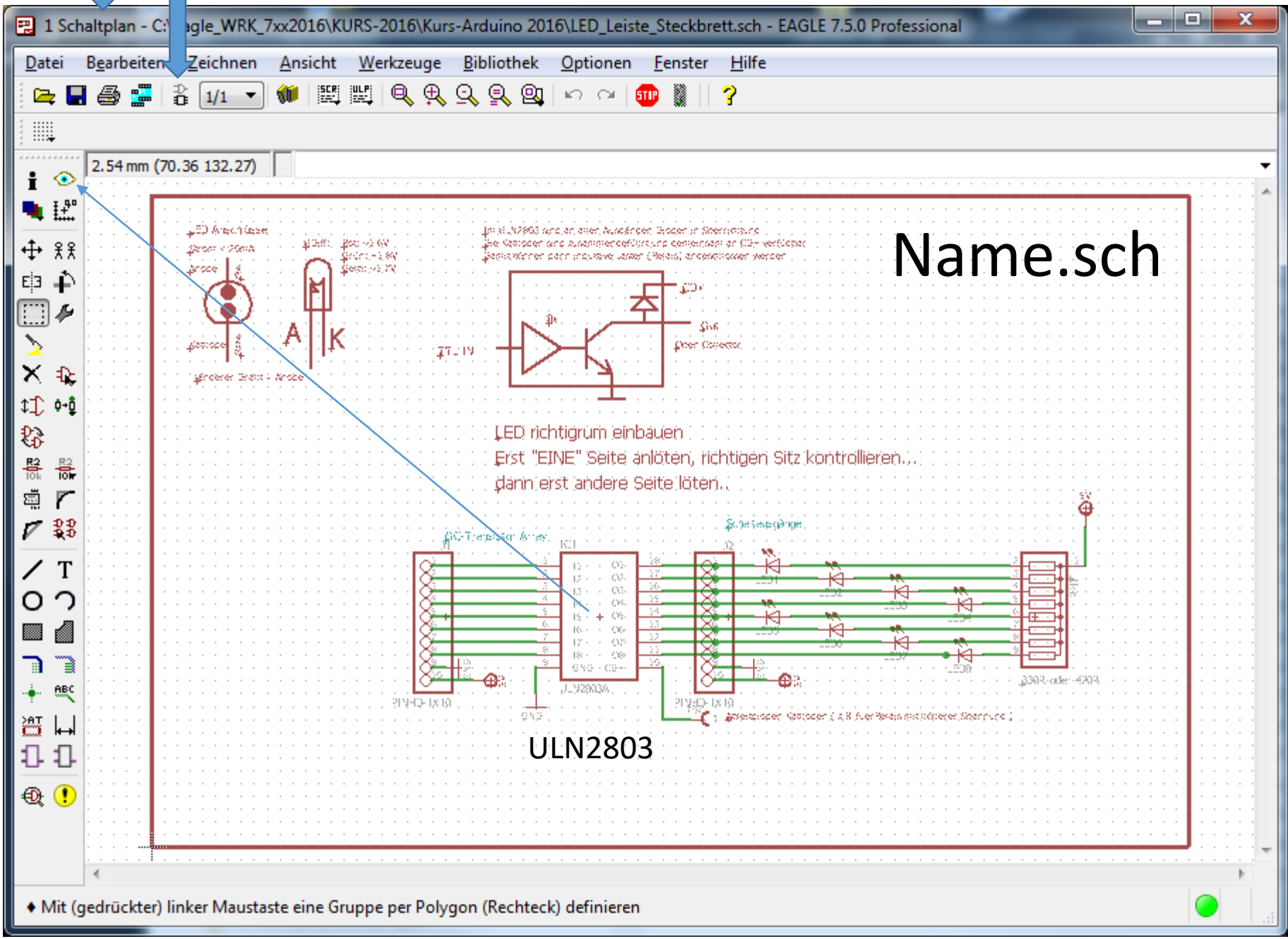
Typischer Skript Text.
Mit Notepad→Eagle.txt
So können Zeichnungen automatisiert Erstellt werden.

wichtig



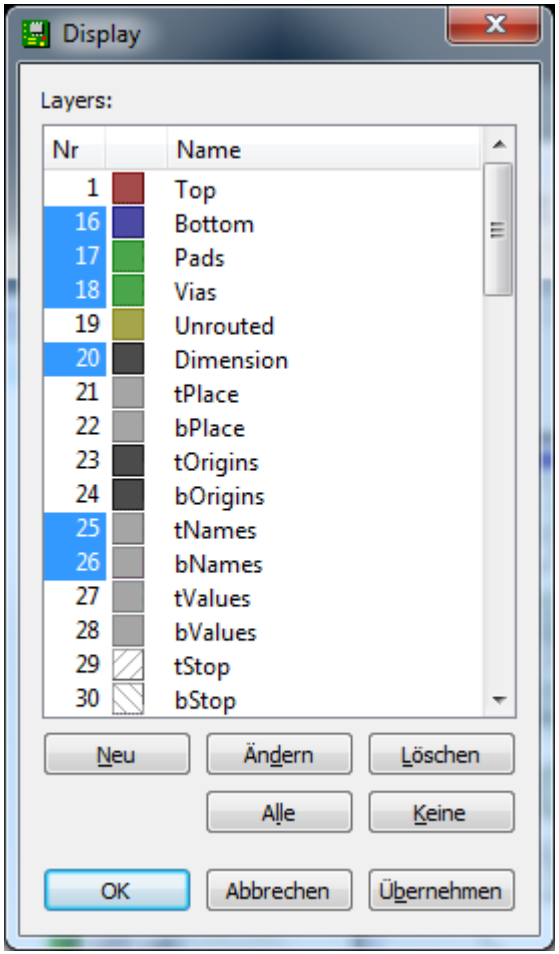

```
change width 0.2032;
grid mm 1 alt mm 0.1;
change layer dimension;
wire (0 0)(160 0);
wire (160 0)(160 100);
wire (160 100)(0 100);
wire (0 100)(0 0);
```

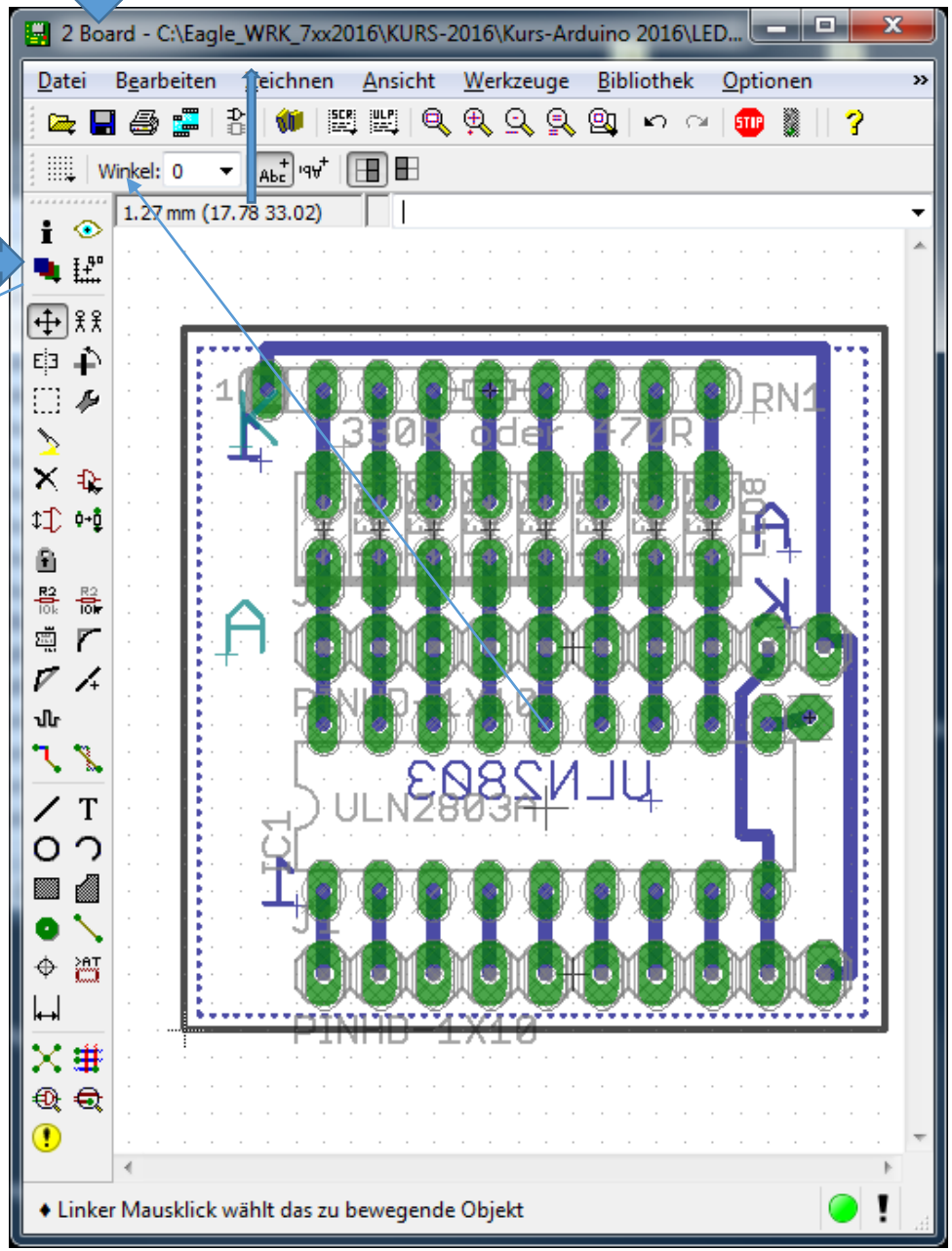
Umschaltung der Anzeige Board /Schematic



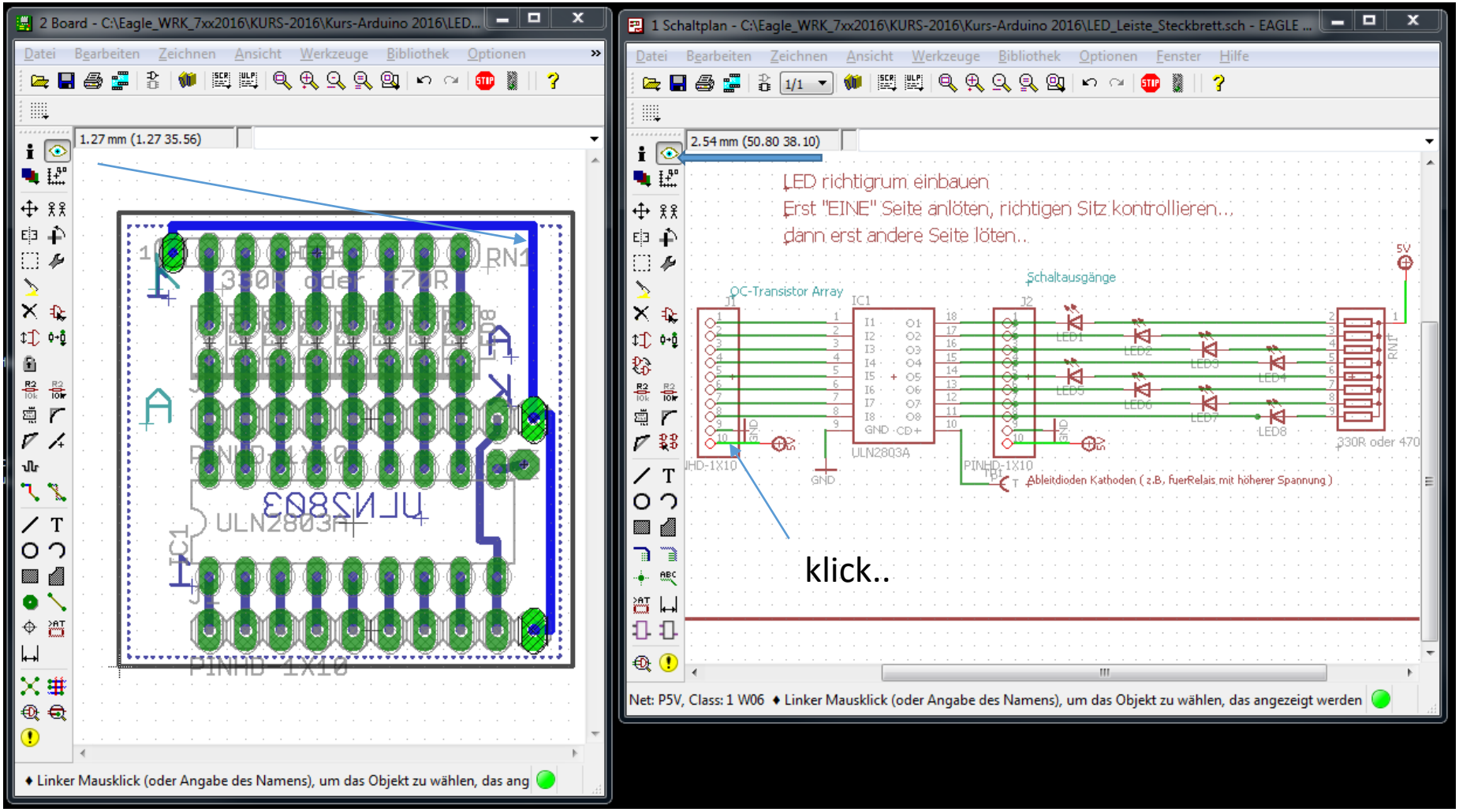
Layer werden selektiert angezeigt.
Rot = Top
Blau = Bottom
Pads, Vias, Dimension, Holes
Dazu: tName, tValue, bName, bValue,
t/b Origins usw.

Name.brd





.brd. & .sch Fenster nebeneinander stellen, Ideal sind 2 Bildschirme. Mit „AUGE“ im Schaltplan auf Leitung klicken. Im Board leuchtet die Leitung auf. Und umgekehrt.

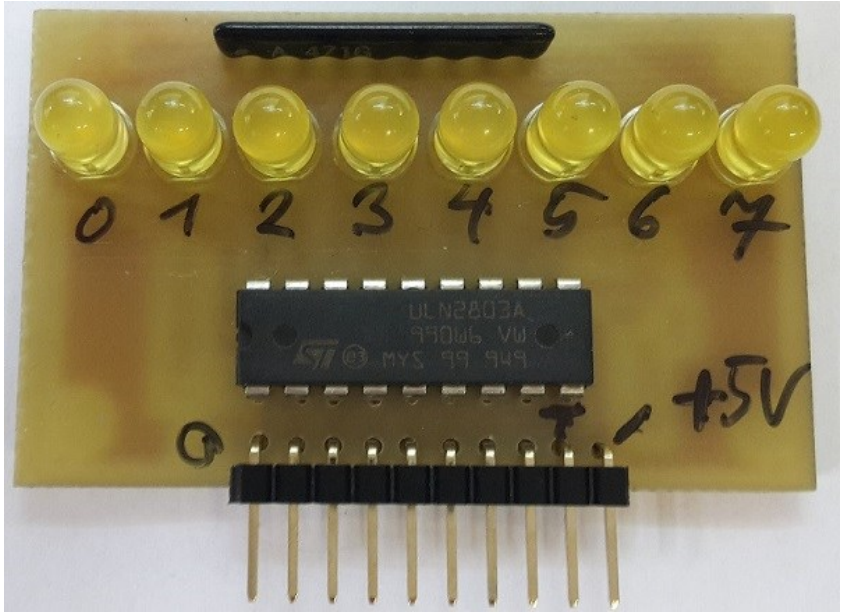


LED-Anzeige für ein Byte

1= Bit 0
2= Bit 1
3= Bit 2
4= Bit 3
5= Bit 4
6= Bit 5
7= Bit 6
8= Bit 7

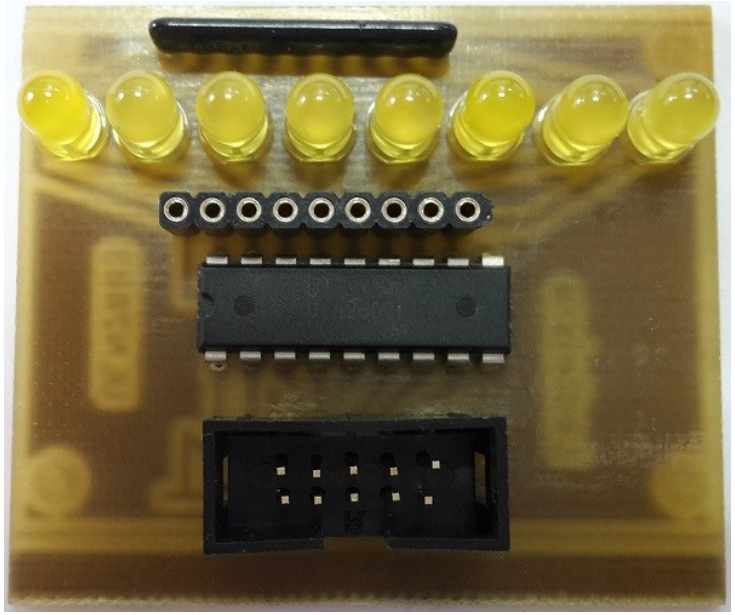
9= GND

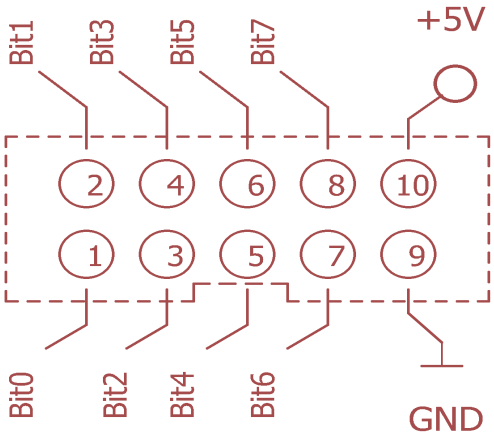
10= +5V



Pin 1 2 3 4 5 6 7 8 9 10

G N D +5V





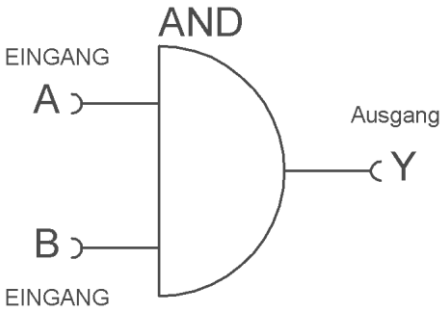
Noch etwas Theorie und Fachwissen aus verschiedenen Gebieten ist erforderlich, bis wir sinnvoll arbeiten können.

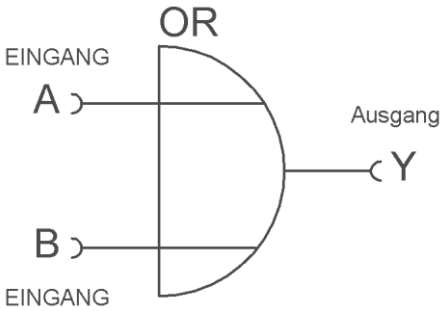
Auch bei vorhandenen Vorkenntnissen stelle ich einige IT-Grundlagen der Vollständigkeit halber nochmals vor.

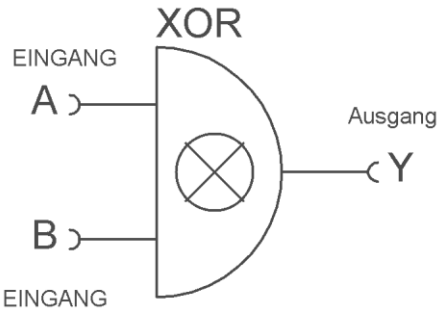
So kann ein gemeinsames Niveau erreicht werden.
Oder einfach nochmals zum Nachblättern…

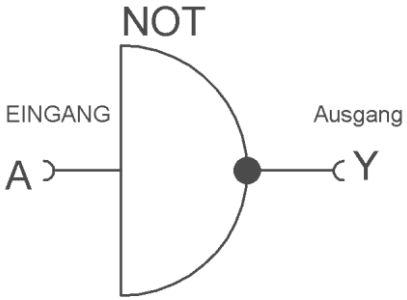
Binäre Logik

Digitale Datenverarbeitung, und damit binäre Logik, lassen sich auf wenige Grundkonzepte reduzieren.
Und, es ist „einfach“. Man muss nur die sprechenden Begriffe kennen.
Es gibt nur **4** elementare logische Funktionen.









Das klassische Symbol dafür ist mir am liebsten.
Es hat zwei Eingänge, „A“ und „B“ und einen Ausgang „C“ oder “y“

Legt man **0** oder **1** an den Eingängen an, je nach Logik, gibt es eine bestimmte „Antwort“ am Ausgang.

Diese „Antworten“ **zeigen** die folgenden Logik- Tabellen an.

Add.
Kombiniert man AND, OR mit NOT am Ausgang wird dies NAND, NOR genannt..

LOGIK
AND OR EXOR NOT

Ein Software Ingenieur (Programmierer) und seine Frau:

Sie: "Schatz, wir haben kein Brot mehr, könntest du bitte zum Supermarkt gehen und 1 holen?
Und
wenn sie Eier haben, bring 6 Stück mit."

Er: "Klar Schatz, mach ich!"

Nach kurzer Zeit kommt er wieder zurück und hat 6 Brote dabei.

Sie: "Warum nur hast du 6 Brote gekauft?!?"
Er: "Sie hatten Eier.

Er hat alles richtig gemacht!
Bedingung „AND“ ist erfüllt

„Binäre Logik“ oder „Wahrheitstabelle“

Diese Funktionstabellen zeigen die elementaren logischen Funktionen.

AND OR NOT EXOR.

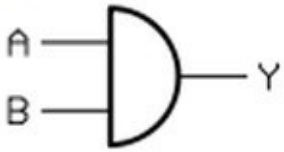
Die Bedeutung erschließt sich aus dem Vergleich des Namens mit den Ein-Ausgangszuständen.

Dies ist sehr wichtig für das Verständnis in der hardwarenahen bitorientierten und logischen Programmierung!

A	B	Y
0	0	0
1	0	0
0	1	0
1	1	**1**

UND

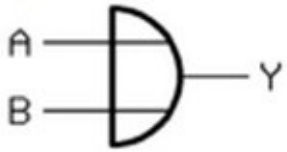
AND



A	B	Y
0	0	0
1	0	**1**
0	1	**1**
1	1	**1**

ODER

OR

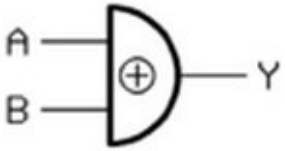


A	B	Y
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

(wieder NULL wenn beide Eingänge 1 sind)

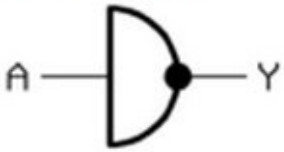
EXKLUSIV ODER

EXOR



A	Y
0	**1**
1	**0**

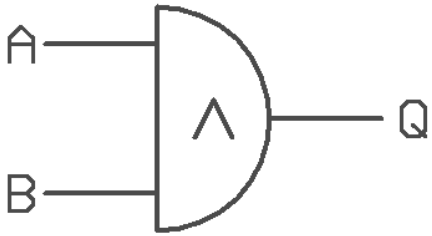
NICHT NOT



Einfache Invertierung

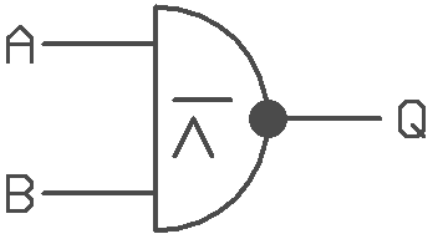
Spickzettel

&



AND

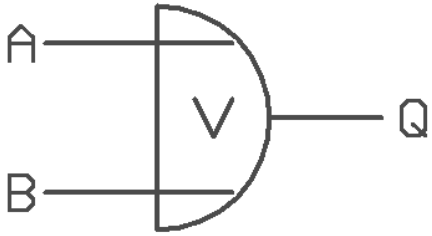
A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1



NAND

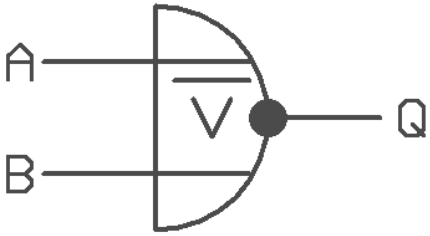
A	B	Q
0	0	1
1	0	1
0	1	1
1	1	0

|



OR

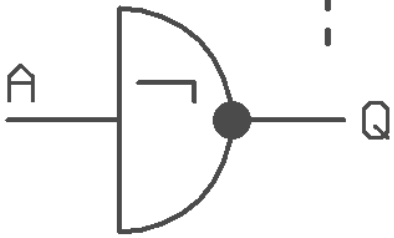
A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1



NOR

A	B	Q
0	0	1
1	0	0
0	1	0
1	1	0

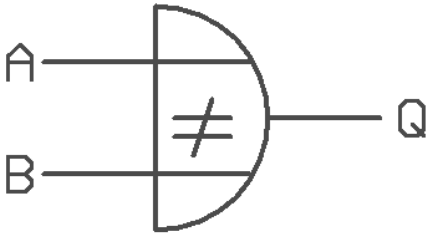
!



NOT

A	Q
0	1
1	0

^



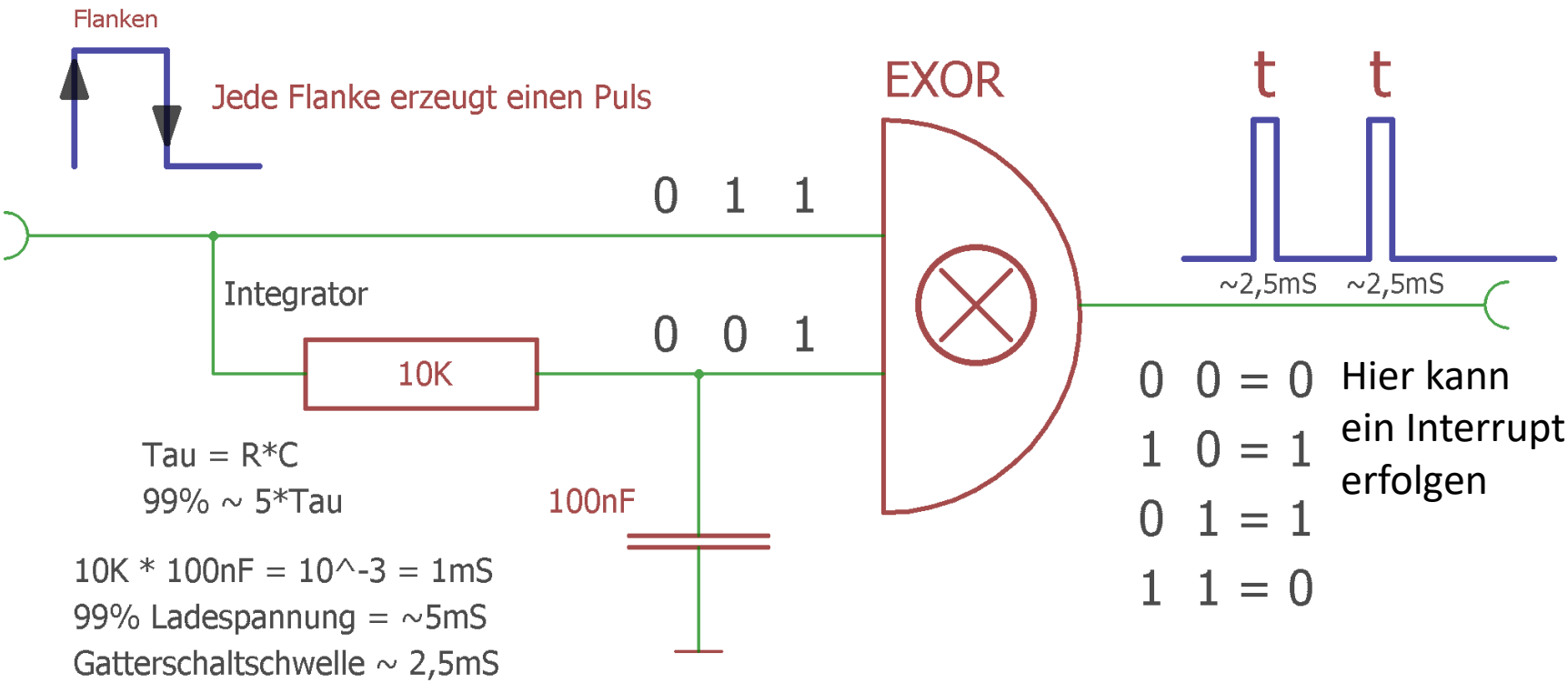
EXOR

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0

Einige interessante Anwendungen mit Standard Logikgattern:

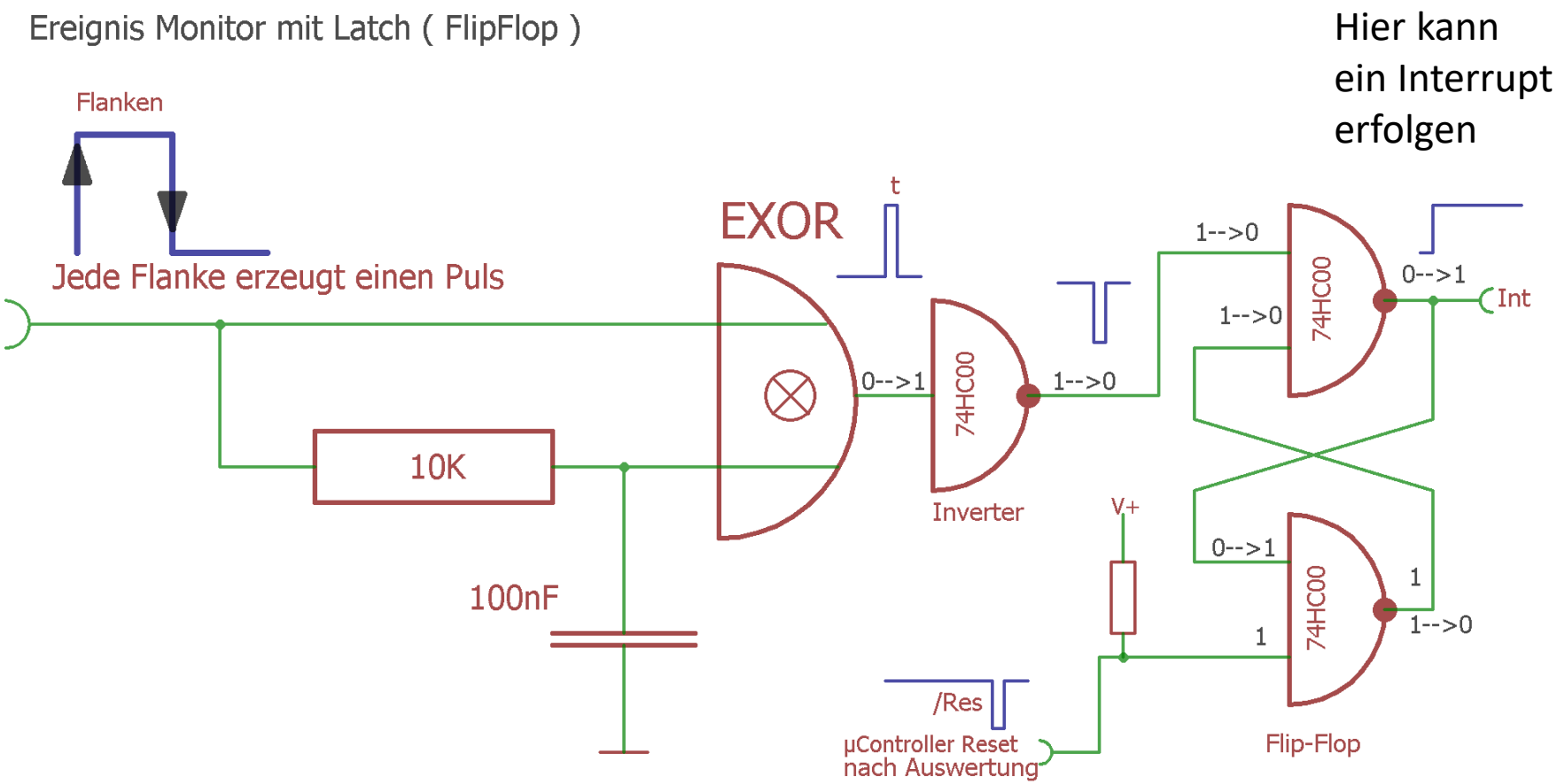
Ereignis Monitor mit Exor

Ereignis Monitor



Einige Anwendungen:

Ereignis Monitor mit Exor und Latch

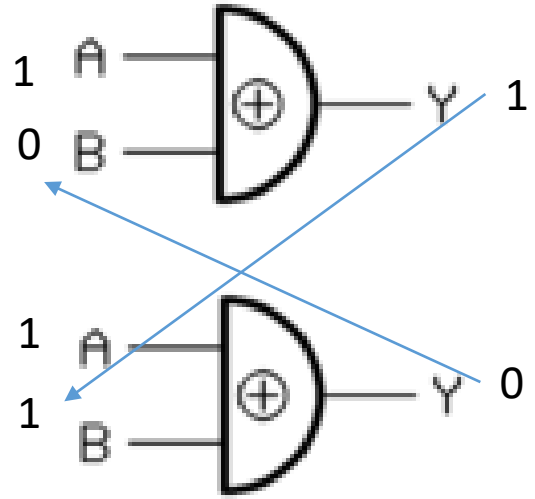


Toggle Bit mit XOR

Toggle = Umschalten
Eine besonders trickreiche Form „**Bit Blinken lassen**“ ist mit XOR zu arbeiten.

xor

A	B	Y
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**



Legt man an einen Eingang dauerhaft eine „1“ und an den anderen das Bit des Zustandes (PORT), so wechselt das Ergebnis den Zustand in das Gegenteil

In „C“ ist der **XOR Operator** ein **^**

1. Durchlauf:
PORT = 0000 0000
LED2 = 0000 0100
→ 0000 0**1**00

2. Durchlauf:
PORT = 0000 0100
LED2 = 0000 0100
→ 0000 0**0**00

Man definiert zuerst das Bit, das man **umschalten** möchte.

```
#define  LED2   (1<<PC2)   // das ist die 1 an XOR-A
while(1)
        {
        PORTC = PORTC ^ LED2;        //blinkt
        _delay_ms( 500 );
        };
// bedenke die Alternative PORTC ^= LED2;
```

„C“ Logik –Bitweise und Logisch

Die Sprache „**C**“ unterscheidet **zwei logische Verarbeitungen**.

Die **bitweise binäre,** somit „bitorientierte Logik“ und die
logische „**True-False** Logik“.

Doch beide arbeiten exakt nach der **Wahrheitstabelle** für AND/OR/NOT/EXOR und die Kombinationen mit NOT, NOR, NAND.

Bitweise-binäre Operatoren in „ANSI-C“ :

& → UND Gut zu erkennen an den **einzeln** stehenden Operatoren
| → ODER
^ → XOR Exklusiv-Oder
! → NOT
und Kombinationen **&=** Und-Gleich
|= Oder-Gleich
^= Exklusiv-Oder-Gleich

Bsp: 0x0A **&** 0x33

```
          00001010
 &        00110011
-------------------------------
UND       00000010
```

0x0A **|** 0x33

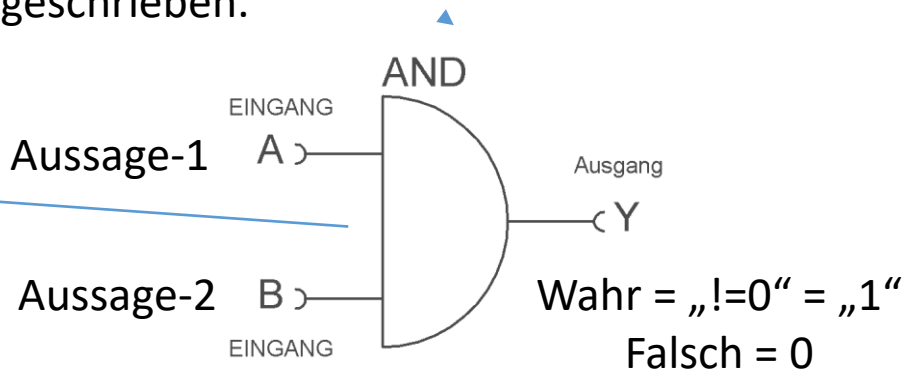
```
          00001010
 |        00110011
-------------------------------
ODER      00111011
```

„C“ Logik – TRUE/FALSE logisch gesehen

„UND **&**, ODER **|**, Exklusiv **^**, NOT **!** werden für „**binäre“ Logik-Abfragen** und Zustand-Speicherungen eingesetzt.

Nicht verwechseln: Für „**logische Wahrheitsabfragen**“ werden die Operatoren in „C“ einfach doppelt geschrieben.

logisch UND: **&&**
logisch ODER: **||**
logisch Exklusiv ODER: **^^**
logisch NOT: **!!**



Diese werden meist bei **IF - THEN** oder **WHILE** Abfragen gebraucht:
Siehe auch die Klammern, die die Zuordnung genau definieren.
Remember → **TRUE = 1**, FALSE **ungleich (!=) 1**
Bsp:


```
if(    (fValue < 1.23)    &&   (u8Flag & 0b00000001)    )
        {   //hier mische ich eine binäre Bit-Abfrage rein
        tuwas();
        };   //nur wenn beide Bedingungen erfüllt sind = TRUE
```

Werte Check

```
#define STRING_SIZE_MAX   8
stchar * ByteToBin(uint8_t u8Byte)          //erzeuge 0..1 TEXT aus Byteinhalt
{atic char ucBin[STRING_SIZE_MAX + 1];   //Größe  soll:8 + 1
char * pcParser = ucBin;
int8_t      i8NN=7;
do
{
if( u8Byte & (1 << i8NN) )       //analysiere Byte auf 0en oder 1en
            {
            *pcParser++ = '1';
            }
else
            {
            *pcParser++ = '0';
            };
}while(i8NN--);
ucBin[8]=0;             //Terminierung
return  ucBin;
};
#endif
```

Zusammenfassung:
Binäre -Hex-Darstellung - dezimale Entsprechung

2^BitNr = Wert

BYTE	Bit 7	BIT 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Hex :	0x**80**	0x**40**	0x**20**	0x**10**	0x**08**	0x**04**	0x**02**	0x**01**
Dez:	128	64	32	16	8	4	2	1

Bitposition

0x**01** = 0000 000**1**
0x**02** = 0000 00**10**
0x**04** = 0000 0**100**
0x**08** = 0000 **1000**
0x**10** = 000**1** 0000
0x**20** = 00**10** 0000
0x**40** = 0**100** 0000
0x**80** = **1000** 0000

Dezimal	HEX	Binär
1	0x01	0000 0001
2	0x02	0000 0010
4	0x04	0000 0100
8	0x08	0000 1000
16	0x10	0001 0000
32	0x20	0010 0000
64	0x40	0100 0000
128	0x80	1000 0000

Bin	Hex
0000	0
000**1**	**1** 0x01
00**10**	**2** 0x02
0011	3
0**100**	**4** 0x04
0101	5
0110	6
0111	7
1000	**8** 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

Dezimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15** = 16 Stellen
Hex:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F** = 16 Stellen

Anwendung des IT Wissen:

Obfuscated (verschleiert) „C“ oder kleine Tricks..

https://www.ioccc.org/

n=n+2 → n+=2 → ++n++ ..... ? Geht das ?

Modulo: (n**%**8) → (n**&**0x7); **warum? -> schneller , kleinerer Code**.....

while(1) →while(42) → for(;;) Mehrere Wege zum Ziel möglich

char acText=„Christof“; → acText[2]==‘r‘ → *(acText+2)==‘r‘

Pointerarithmetik → zumeist schnellerer Code

Fund with Flags

Ein Flag ist ein einzelnes Zeichen, ein Lämpchen, minimal **EIN BIT**. So kann man also **EINE**, wenn auch wichtige, **Informationseinheit** setzen, verteilen lesen, Zustände: **0 = aus**

1 = an

D and 4

Techniken zum gezielten Setzen und Löschen von Bits!

Wir wollen **ein** oder **mehrere Bits** in einem Byte gezielt
auf 1 setzen oder
auf 0 löschen.

Aber das Handwerkszeug haben wir fast schon!
Jetzt wird es Zeit das Wissen in der Praxis anzuwenden.
Also so, wie es in der Programmierung gemacht wird.

https://chaosblog.wordpress.com/2014/09/15/ein-einziges-ping/

EINE Information in **einem BIT** (weniger geht nicht) speichern.
Zwei Methoden können angewendet werden:

PLAN A: Bit(s) in einem BYTE Setzen, löschen, abfragen

PLAN B: Die „C“ eigene „Bit-Field“ Methode nutzen.

Vorbereitung zu **Methode A**. Zuerst: mit #define sprechenden Alias definieren:

```
#define  SYSTEM_MODE1          0x01  ( Single Bit  2^n)
#define BOMBE_IST_SCHARF       0x80 ( Single Bit 2^n)
```

Vorbereitung zu **Methode B.** → **Syntax beruht auf Strukturen**

```
typeddef struct
        {
        unsigned char SysMode     : 3;  //3 Bits = 8 cases
        unsigned char BombIsSharp: 1; // 1 Bit
        }BOMBFLAG_TYPE;
```

volatile BOMBFLAG_TYPE gstBombFlags; Struktur global instanzieren

→ *gstBombFlags.BombeIsSharp = 1;* *benutzen*

Bit Daten mit ENUM
Plan C: Anstelle mit Defines oer Bit-Felder zu arbeiten:

„C“ lässt noch einen Datentyp zu, der genutzt werden kann→ enum

(integer bzw Aufzählungstyp)

enum {KEYWORD = 1, EXTERNAL = 02, STATIC = 0 4};

Die Zahlen müssen Potenzen von 2^x Sein

Anwendung:

u8Flags **|=** EXTERNAL | STATIC;

u8Flags **&= ~** KEYWORD;

u8Flags &= ~(EXTERNAL | STATIC); Löscht mehrere Bits

If(u8Flags & (EXTERNAL | STATIC)) == 0)……

Noch einmal Bit-Felder
BITFELDR SIND VORZEICHENLOS !!

Daher „unsignrd“

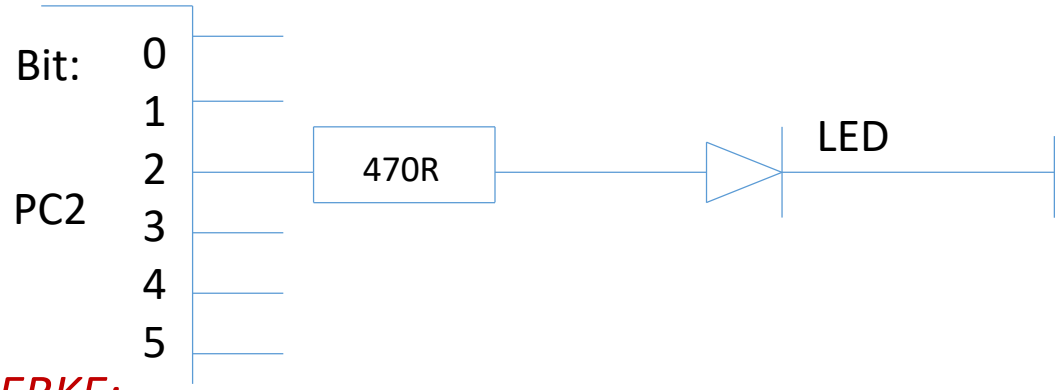
```
struct{
unsigned int        KeyWord : 1;
unsigned int        Extern     : 1;
unsigned int        Static      : 1;
unsigned int        Modus     : 7;    //128 Modi möglich
}stFlags;

stFlags.Extern = 1;
stFlags.Modus = 47;

If( stFlags.Modus  == 33 )  ……................
If( stFlags.Extern == 1 )
            {
            stFlags.Extern = 0;
            };  ….
```

Techniken zum gezielten Setzen und Löschen „eines“ Bits!

Wie wollen z.B. eine LED an PORTC einschalten und ausschalten.

Die LED ist an Bit-2 eines x-beliebigen Ports angeschlossen.

```
#define LED2  0b00000100
```



Bin	Hex
0000	0
000**1**	**1**
0x01	
00**1**0	**2**
0x02	
0011	3
0**1**00	**4**
0x04	
0101	5
0110	6
0111	7
1000	**8**
0x08	
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

MERKE:

*3 (Haupt)**Möglichkeiten** in C ein **Bit an einer Position** zu definieren:*

```
        #define LED2   0b0000 0100
oder    #define LED2   (1 << PORTC2)   //(1 << 2) = 0b00000100
oder    #define LED2   0x04
```

Technik zum „Setzen“ eines Bits

Zum gezielten ‚**Setzen**‘ eines oder mehrerer Bits auf **EINS** wird dieses **Bit einfach mit dem Ziel-Byte** ***verODERt*****.**

Alle anderen Bits bleiben unverändert !

#define 0b00000**1**00

alternativ: #define LED2 0x04

alternativ: #define LED2_BIT (1 << **2**)

//mit ‚2‘ (=Bit 2) für die Nutzung des SHIFT Operators

0b0**1**000000 // Ausgangsituation

PORTC = PORTC **|** 0b00000**1**00; // ausführliche Zuweisungs-Schreibweise

0b0**1**000**1**00 // …..nach dem bitweisen **Ode**r

einfacherweise schreibt man PORTC **|=** 0b0000**1**00;

PORTC **|=** 0x04; //sprich „Oder-Gleich“

ODER PORTC **|=** (1<< PC2); //sprich 1 shift 2*

Oder ‚sauber‘ mit den #define……

PORTC **|= LED_BIT**; // **verODERn** setzt die Bits !

Nun leuchtet die LED, weil eine EINS = +5V am PORTC, Bit2 anliegt.

Technik zum „Löschen“ von Bits

Zum gezielten „**Löschen**“ eines oder mehrerer Bits auf **NULL** werden diese **Bits**

in **zwei Schritten** folgendermaßen behandelt: Die **TILDE** **~** wird hier wichtig.

1. das ‚Komplement‘ des/der Bits werden gebildet: **~ =Invertierung von 0 und 1**

2. dieses ‚Komplement‘ wird dann mit dem **Ziel-Byte *verUNDet.***

#define LED2 0x04 // 0b 0000 0**1**00

Tilde **~**LED2 → 0b 1111 1**0**11

PORTC = PORTC **&** **~** 0b00000**1**00;

das **Komplement** ist: PORTC = PORTC **&** 0b1111**1**011;

Einfacherweise schreibt man: PORTC **&= ~** 0b0000 0100;

PORTC **&= ~** 0x04; //sprich: „**Und-Gleich-Tilde**“

oder: PORTC **&= ~** (1<< 2); // (1<<PC2)

Und nun ‚sauber‘ mit den #define……

PORTC **&= ~**LED2; //Man erkennt **~Komplement bilden** und dann **‚VerUNDen‘.**

Nun ist die LED aus, weil eine NULL = 0V am PORTC, Bit2 anliegt.

Merke: Mit diesem Trick kann gezielt **ein oder mehrere** Bits gelöscht werden.

Mehrere Bits gleichzeitig setzen.

- ….geht genauso einfach wie das Setzen einzelner Bits.
- Egal, ob die Bits „0“ sind, oder schon „1“ waren,
  nach dem ‚Verodern‘ sind diese **„1“**.

- **Bsp.: Quellbyte →** **0b10010000** **0x90**
  zu setzende Bits: **|** **0b00010110** **0x16**
  ‚Verodern‘ ergibt: **0b10010110** **0x96**

In ANSI-C : Startwert **= 0x90;**
verodern Ergebnis = Startwert **| 0x16**; // ergibt 0x96

Kurzform: Startwert **|= 0x16;** // ergibt 0x96

Bin	Hex
0000	0
000**1**	**1** 0x01
00**1**0	**2** 0x02
0011	3
0**1**00	**4** 0x04
0101	5
0110	6
0111	7
1000	**8** 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

verodern

Mehrere Bits gleichzeitig löschen.

- Mehrere Bits löschen geht genauso einfach wie das Löschen einzelner Bits.
- Egal ob die Bits „1“ sind, oder schon „0“ waren, nach dem mit dem **Komplement ~ ‚Verunden‘** sind diese **„0“**.

• Bsp.: Quellbyte →		0b10010000	0x90
zu löschende Bits:		0b00010110	0x16
1.**Komplement** davon:	**~**	0b11101001	0xE9
2. **verUNDen**‘ ergibt:	**&**	0b10000000	0x80

In ANSI-C : **Quelle = 0x90;**

0x90 &= ~ 0x16; // ergibt 0x80

Bin	Hex
0000	0
000**1**	**1** 0x01
00**1**0	**2** 0x02
0011	3
0**1**00	**4** 0x04
0101	5
0110	6
0111	7
1000	**8** 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

Merke:

Durch eine einfache Zuweisung ,**=**, in der Form: → Ziel = 0x16;
hat man zwar auch die Bits gesetzt, jedoch evtl. **andere Bits überschrieben**/gelöscht.
Durch die Technik des

Ver**oder**ns zum **‚Setzen‘ , mit** **|=** und

‚Ver**und**ens‘ mit dem **Komplement** zum **‚Löschen, mit** **&= ~**

von Bits können einzelne Bits beeinflusst werden ohne ungewollt andere Bits zu überschreiben.

Diese Methode ist sehr wichtig!.

So kann man ohne Beeinflussung anderer Bits, gezielt Bits setzen und löschen

Ziel |= 0x16;

Ziel |= BITs

Ziel &= ~ 0x16;

Ziel &= ~BITs

Setzen → |=

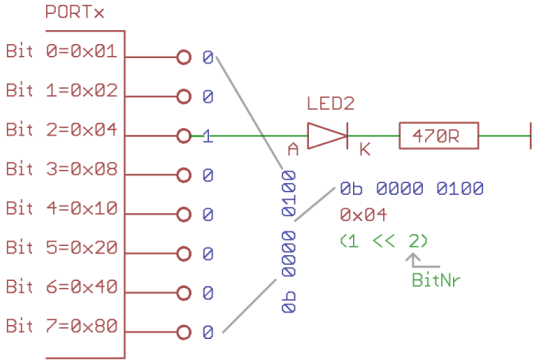
Löschen → &= ~

ÜBUNGEN

Test-Programm LED ON/OFF
mit #define im **BIT-SHIFT** Format
Hier geht es um den Programmierstil mit #define

```
/* blinkende LED, mit Code in ANSI C  */
/* nur direkte Zuweisung */

#include <avr / io.h>
#include <util / delay.h>

                        //→ : #define PC2  2

#define LED1_BIT        (1 << 2)   //(1<<2) → weil LED ist an Bit 2
#define LEDPORT         PORTC
#define LED_DDR_X       DDRC
#define BLINKDAUER      500

int main (void)
{
LED_DDR_X = LED1;       //Bit auf Ausgang setzen;

while (1)
        {
        LEDPORT =  LED1_BIT ;       //Led an Bit 2 ON
        _delay_ms (BLINKDAUER);
        LEDPORT = 0;                        //OFF
        _delay_ms (BLINKDAUER);
        };
}; //main
```

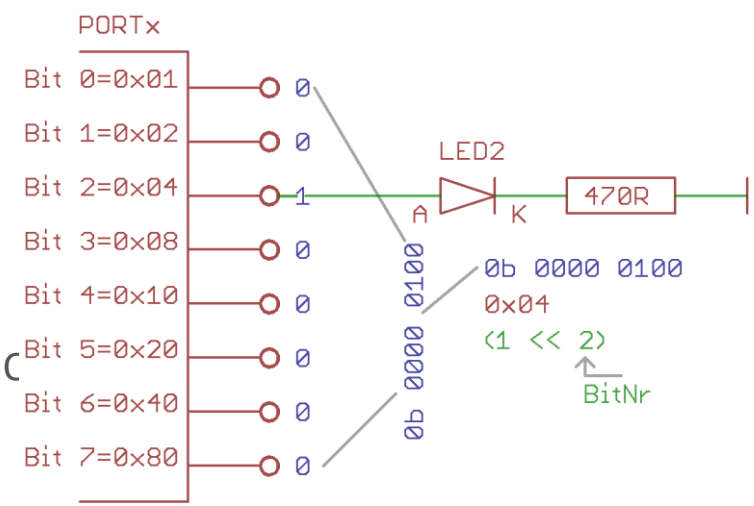
Bin	Hex
0000	0
0001	**1** 0x01
0010	**2** 0x02
0011	3
0100	**4** 0x04
0101	5
0110	6
0111	7
1000	**8** 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12



ÜBUNGEN

Test-Programm LED ON/OFF mit **Setzen und Löschen von Bits**

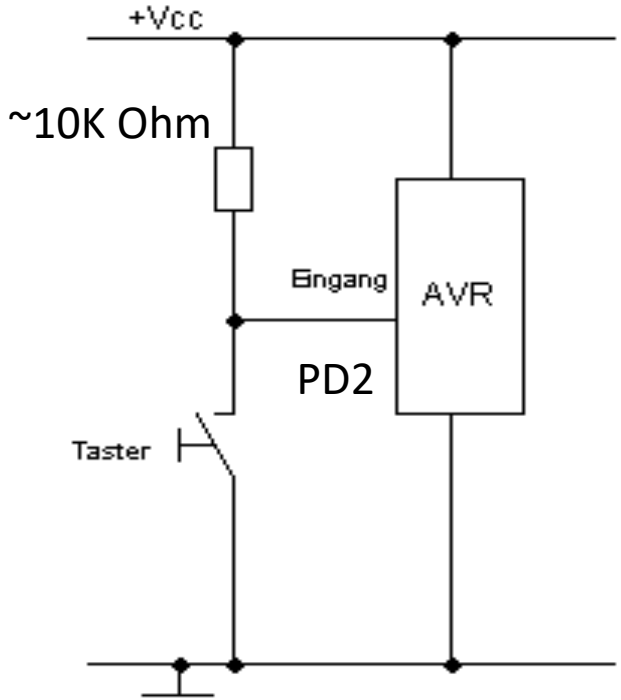
```
/* blinkende LED, mit Code in ANSI C  */
/* nur direkte Zuweisung */

#include <avr / io.h>
#include <util / delay.h>

                    //remember : #define PC2  2

#define LED1_BIT                (1 << PC2)    //PC2 weil LED ist an Bit 2
define BLINKDAUER   500

int main (void)
{
DDRC = LED1_BIT;    //Bit auf Ausgang setzen;

while (1)
        {
        PORTC    |= LED1_BIT;          //Led an Bit 2 C
        _delay_ms (BLINKDAUER);
        PORTC &= ~ LED1_BIT;           //OFF
        _delay_ms (BLINKDAUER);
        };
};
```



Lesen ‚eines' Port Bits mit PINx

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

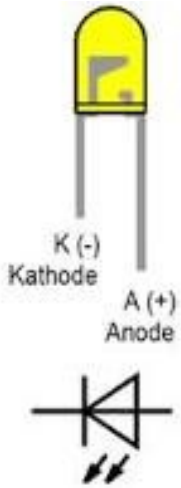


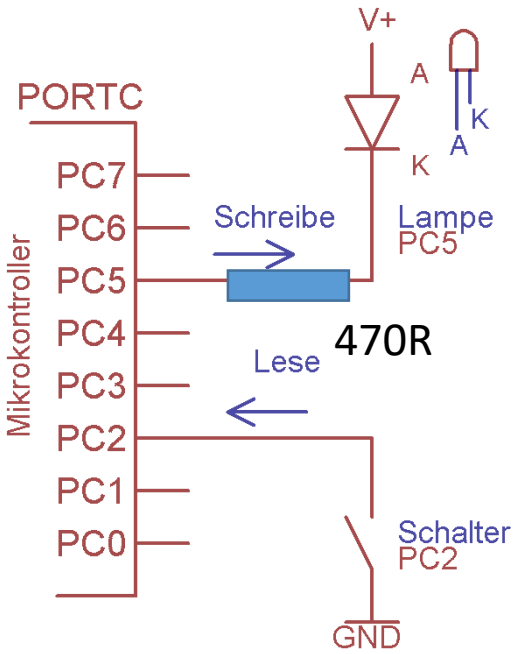
Wir lesen eine Taste an PORTD Pin 2 = PD2 .
Hinweis: Der Widerstand könnte auch nur der interne **Pullup** sein !
Dieser PIN könnte gleichzeitig die Funktion *INT0* sein. Aber das später….

Zum Lesen wird nicht PORTD verwendet sondern **PIND.** PINx liest immer das ganze Port.
PIND & (1<< PD2); // nur Bit 2 lesen
In der Theorie lesen wir:
Ruhezustand = „HIGH". → Value = 4
Taste gedrückt =: „LOW". → Value = 0

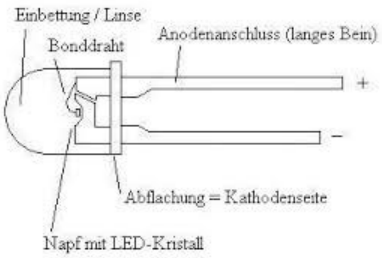
2. Schreibweise: Konsequente Nutzung der **#define Technik**

Mixed Portbetrieb. Ein Taster steuert synchron eine LED.

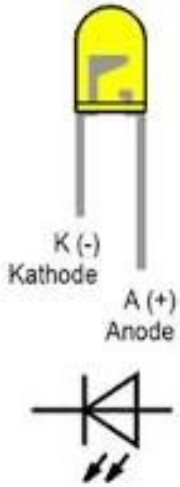


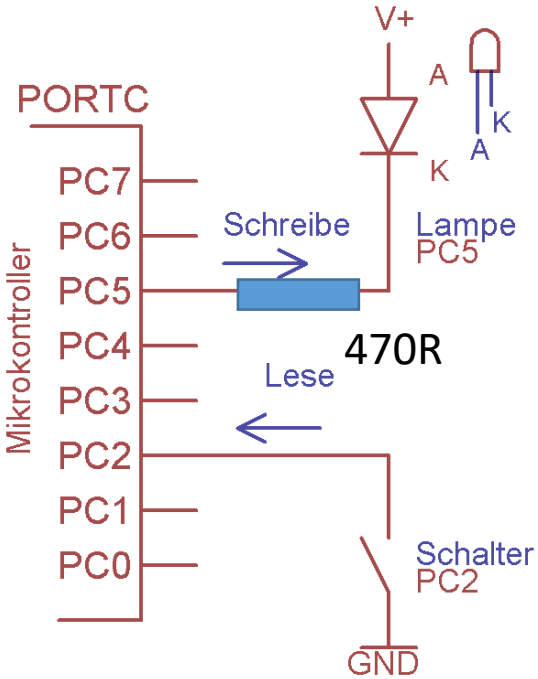



```
#define SWITCH_PIN      PINC
#define SWITCH_PORT     PORTC
#define SWITCH_DDR      DDRC
#define SWITCH1         (1 << PINC2)

#define LEDPORT         PORTC
#define LED_DDR         DDRC
#define LED0            (1 << PORTC5)

int main (void)
{
LED_DDR |= LED0;                    //Bit auf Ausgang setzen;
SWITCH_DDR    &= ~SWITCH1;          //Bit auf Input setzen;
SWITCH_PORT  |=    SWITCH1;         //internen Pullup Enable

while (1)
            {
            if( SWITCH_PIN  &  SWITCH1 )        //  Taste gedrückt ?
                        {
                        LEDPORT |= LED0;        //Bit an Bit 5 = 1
                        }
            else
                        {
                        LEDPORT  &=  ~ LED0;    //Bit an Bit 5 = 0
                        };
            };
};
```

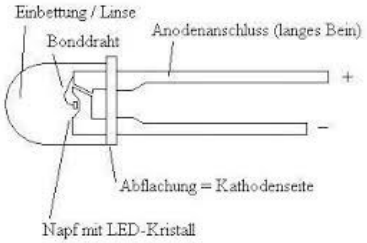


Jetzt wird es Zeit für praktische Versuche.
Gemischter Portbetrieb. Ein Taster steuert synchron eine LED





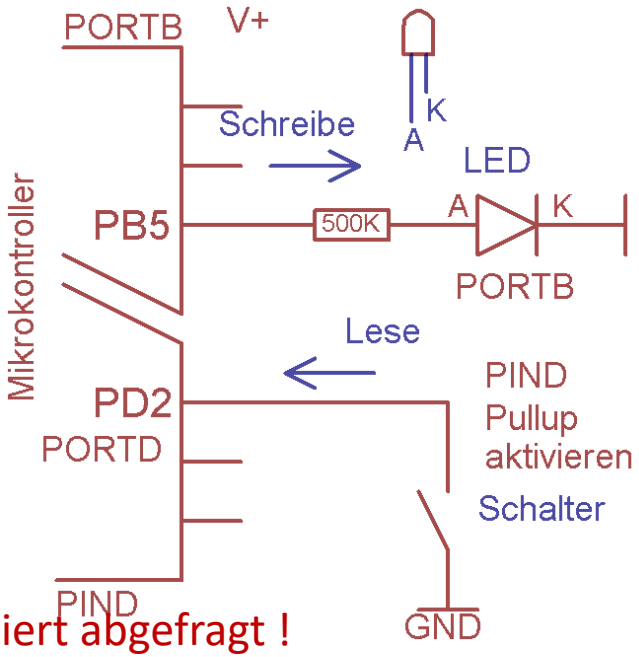

```
#include <avr/io.h>
#include <util/delay.h>

#define SWITCH1        (1 << 2)  →PC2
#define LED1           (1 << 5)  →PC5

int main (void)
{
DDRC   &= ~(1 << 2);        //Bit auf Input setzen;
PORTC  |= (1 << 2);         //internen Pullup Enable

DDRC |= (1 << 5);                    //Bit auf Ausgang setzen;

while (1)
        { //WICHTIG:  Hier wird mit „&“ ein Bit isoliert abgefragt !  MERKEN !!
        if( PINC & (1 << 2)  )        //Taste gedrückt ? Merke: →Lesen mit PINx
                {
                PORTC  |=   (1 << 5);    //Bit an Bit 5 = 1
                }
        else
                {
                PORTC &=  ~ (1 << 5);    //Bit an Bit 5 = 0
                };
        };
};
```



Anwendung der #DEFINES

```
#define SWITCH1    (1 << 2)
#define LED1       (1 << 5)

int main (void)
{
DDRC   &= ~SWITCH1;          //Bit auf Input setzen;
PORTC |= SWITCH1;            //internen PullUp Enable
DDRC |= LED1;                //Bit auf Ausgang setzen;
while (1)
        {  //WICHTIG:  Hier wird mit „&“ ein Bit isoliert abgefragt !  MERKEN !!
        if( PINC & SWITCH1  )        //Taste gedrückt ? Merke: ?Lesen mit PINx
                {
                PORTC |= LED1;       //Bit an Bit 5 = 1
                }
        else
                {
                PORTC &=  ~LED1;     //Bit an Bit 5 = 0
                };
        };
};
```

Weiteres Beispiel Verschiedene Ports




```
#define GREEN_LEDBIT   (1<<PORTB5)  //PB5
#define TASTER1        (1<<PIND2)   //PD2

int main(void)
{
DDRB |= GREEN_LEDBIT;      //LED liegt an PORTB
DDRD &= ~ TASTER1;         // Taster Bit  an PIND)
PORTD |= TASTER1;          // Pullup aktivieren
while(1)   // endless loop
           {//WICHTIG:  Hier wird mit „&“ ein Bit isoliert abgefragt !

           if( PIND & TASTER1 )        //  PINx nicht PORTx
                      {
                      PORTB &= ~GREEN_LEDBIT;   // lösche wenn 1
                      }
           else
                      {
                      PORTB |= GREEN_LEDBIT;    // setze wenn 0
                      };
           };//while
};
```

WICHTIGES SOFTWARE WEKZEUG !!!:

Bitweise Analyse der Bitfolge in einem Byte/Word etc mit Hilfe „einer“ durchgeschobenen „1“

Gegeben:

Sagen wir, ein Byte/Word etc. hat den Inhalt

0b0011010101011001 (jetzt gleichmal 16 Bit)

Wir wollen der Reihe nach (Bit Nummer 0..15) dieses Byte analysieren, wo und ob eine „1“ oder eine „0“ an der selektierten Position steht,

Das geht so:

```
uint16_t u16Testword = 0b0011010101011001;
uint8_t u8NN=0;
do
  {
  if(  0b0011010101011001  &  ( 1 << u8NN)   )
    {     SET_LED();          }
  else
    {     CLEAR_LED();  };
  u8NN++;
  }
 while( u8NN < 16);
```

FLAG-BITS als Steuerzentrale

Ich betone die Wichtigkeit folgender Methode:
Jedes einzelne Bit eines Bytes kann Zustände signalisieren.

Und zwar beliebig. Und dies funktioniert extrem schnell und effektiv.

Dazu „*erfindet*“ man **benötigte** und **beliebige** Bitpositionen mit sprechenden Namen.
Bsp: → Global definiertes Statusbyte: **uint8_t gu8Motorstatus**;

So **transportiert man jedwede Informationen** global durch den µController, auch aus zeitkritischen Prozessen wie Interrupts.

Im Headerfile:

```
#define MOTOR_ENABLED          0x01
#define MOTOR_DIRECTION        0x02
#define MOTOR_RAMPE_USED       0x04
#define MOTOR_RAMPE_START      0x08
#define MOTOR_RAMPE_AKTIVE     0x10
#define MOTOR_RAMPE_READY      0x20
#define MOTOR_INDEX_OK         0x40
#define MOTOR_CURRENT_LIMIT    0x80
        // uint8_t gu8Motorstatus
```

Sicheres Bit Manipulieren mit den „Präprozessor Makros“ = sprechende alias Namen

Tipp: Nun können wir diese Technik gut benutzen um saubere, fehlervermeidende und gut lesbare Makro-Definitionen zu schreiben. (einmalig definieren, häufig anwenden….)

```
#define LED1_ON()        (PORTC |=   0x04)
#define LED1_OFF()       (PORTC &= ~0x04)
```

statt 0x04 kann auch (1<<PC2) stehen

Jetzt kann man im Programm „fehlerarm“ und „gut Lesbar“ die LEDs ein und ausschalten. ..ohne diese komplizierte Schreibweisen.
Man muss nur **„einmal“ ein #define Makro** anlegen-

*Gleichzeitig sieht es **wie ein Funktionsaufruf** aus. funktion();*

```
        LED1_ON();
oder    LED1_OFF();
```

Einzelne Bit-Flags steuern das Programm

Bitte ein Bit

Ein FLAG ist ein Signal, bestehend aus der kleinsten Einheit :

Wir können ein Byte nutzen um bis zu **8** völlig **frei erfundene Bedeutungen** zu **definieren**, zu speichern, Zustandsänderungen zu erkennen, diese **im ganzen Programm** zugänglich zu machen.

EIN SEHR MÄCHTIGES WERKZEUG !

Reihenfolge Egal

```
#define TASTENDRUCK_ERKANNT     0x01
#define LED_BLINK_MODUS         0x02
#define INTERRUPT_EVENT         0x04
#define MOTOR_DIRECTION         0x08
#define MOTOR_ON                0x10
#define INDEX_OK                0x20
// gu8Status                     usw..
```

BIT-Nr	HEX 0x	MASKE 0b
0	0x01	0b 0000 0001
1	0x02	0b 0000 0010
2	0x04	0b 0000 0100
3	0x08	0b 0000 1000
4	0x10	0b 0001 0000
5	0x20	0b 0010 0000
6	0x40	0b 0100 0000
7	0x80	0b 1000 0000

Ereignisse in einem Bit verwalten

Bitte ein Bit

Wir haben, frei erfunden, ein meist globales Statusbyte definiert
z.B.: unsigned char **gu8Status**. //Dieses soll beliebige Zustände speichern.
Dabei wird nur 1 Bit genutzt.

Zudem wir „meist im Headerfile“ eine Liste von #defines angefertigt, die ebenfalls <u>nur ein Bit</u> namentlich definieren. Ebenfalls frei erfunden!

```
#define MOTOR_ON        0x10
#define INDEX_OK        0x40
// gehört zu gu8Status  usw..
```

Jetzt kann überall im Programm zu jeder Zeit →

Ereignis setzen bzw. festhalten:

```
gu8Status |= INDEX_OK;
```

Jetzt kann überall im Programm zu jeder Zeit →

Ereignis löschen:

```
gu8Status &= ~ MOTOR_ON;
```

WICHTIG !

Methode: Zustands-Flag gezielt abfragen

Es ist ganz einfach, nachdem ein Bit definiert und einem zugehörigen Speicherbyte gesetzt oder gelöscht ist, kann dieser Zustand auch jederzeit abgefragt werden durch einen einfachen **& Abgleich.**

Gegeben:

```
#define MOTOR_ON                    0x10
unsigned char  gu8MotorStatus;
irgendwo war  → gu8MotorStatus |= MOTOR_ON ;
```

if(gu8MotorStatus & MOTOR_ON)

```
 {
//Tue etwas
MotorStop();          // das hat Konsequenzen für den ON State
 ( gu8MotorStatus &= ~MOTOR_ON;  )  / lösche ON Flag;
};
```

Nutzung der Flagbit Technik zur Programmsteuerung

Also zuerst steht irgendwo im HEADER-FILE ein **#define**

```
#define INT0_INTERRUPT_EVENT        0x40
```

Zudem existiert ein „globale“ Flag-Variablembyte

```
uint8_t  gu8ProgMode;
```

Bsp. Dann passiert was im Interrupt…… irgendwann….

```
ISR   INT0_vect()                    // Interrupt Service Routine
  {
  gu8ProgMode  |=  INT0_INTERRUPT_EVENT; // setze Eventflag
  // jetzt kann über im Programm  erkannt werden, dass was passiert ist.
  //Tue irgendwas ….  z.B.  gu16IndexPos = 0;
 };
```

→irgendwo im Programm ist das evtl. interessant…

```
If(gu8ProgMode  & INT0_INTERRUPT_EVENT  )
        {
        gu8ProgMode  &=~ INT0_INTERRUPT_EVENT; // lösche Eventflag
        //Tue irgendwas
        };
```

Beispiel… In einem Interrupt wird irgendeinem Ereignis festgestellt. z.B ein regelmäßiger Zeittackt = Tick-Event
Dies wird im Interrupt als Ereignis festgehalten durch setzen eines Bits:

1.) gu8Status **|=** **INTERRUPT_EVENT**;

2.) Ereignis abfragen: (binärer Vergleich)
irgendwo im Programm (main-while(1) loop)

```
if( gu8Status & INTERRUPT_EVENT )
 {
 3.) Ereignisflag löschen
 gu8Status &= ~INTERRUPT_EVENT
 ...tuh_was
 };
```

Ereignis setzen bzw. festhalten:

```
gu8Status |= INDEX_OK;
```

Ereignis löschen:

```
gu8Status &= ~ INTERRUPT_EVENT;
```

Bit-Felder bzw. Bit-Fields in ANSI C

Die Erfinder von „C“ wollten die Einzelbit Behandlung gleich mit in die Sprache integrieren. Das ist ihnen gründlich misslungen. (meine Meinung) Da man in der Praxis aber darüber stolpert sollte man diese „Bää“ Lösung zumindest wiedererkennen:

Bit Field Declaration:

```
struct {
   unsigned int  age : 3;       // 3 = Bit-breite des Begriffes Age. Verbraucht
}sAge;                          // aber ein volles “int” im Speicher. Vorteil: Keiner.

 sAge.age = 7;        // schon sind alle Bit verbraucht
```

Reales Bsp. aus PID:

```
struct GLOBAL_FLAGS {  //! True when PID control loop should run one time
  uint8_t pidTimer:1;
  uint8_t dummy:7;            // was macht man bloß mit den restlichen 7 Bit ?
} gFlags = {0, 0};  // ?? ist das nun „union“ oder wird ein weiteres Byte
verbraten ?  Anwendung:
gFlags.pidTimer = TRUE;    // verhält sich, als wenn man ein Bit gesetzt hätte
```

```
typedef struct  {
        volatile char parity_err        :1          ;//0 Hilfs Parity
        volatile char parity_P1         :1          ;//1 Berechnetes Parity P1
        volatile char parity_P2         :1          ;//2 Berechnetes Parity P2
        volatile char parity_P3         :1          ;//3 Berechnetes Parity P3
        volatile char dcf_rx            :1          ;//4 Es wurde ein Impuls empfangen
        volatile char dcf_sync          :1          ;//5 Uhr  ist syncronisiert
        volatile char dcf_NoSignal      :1          ;//6
        }ST_DCF_FLAGS_TYPE;
```

dann: → **volatile ST_DCF_FLAGS_TYPE gstDCF_Flags**;

```
if( gstDCF_Flags.dcf_sync == 0 ) {..............};    //Anwendungsbeispiel

        u8NN  |= (gstDCF_Flags.parity_err << 0);
        u8NN  |= (gstDCF_Flags.parity_P1 << 1);
        u8NN  |= (gstDCF_Flags.parity_P2 << 2);
        u8NN  |= (gstDCF_Flags.parity_P3 << 3);
        u8NN  |= (gstDCF_Flags.dcf_rx << 4);
        u8NN  |= (gstDCF_Flags.dcf_sync << 5);
        u8NN  |= (gstDCF_Flags.dcf_NoSignal << 6);
uart_puts( ByteToBin( u8NN ) );                 // Anzeige der Bits
```

Weiteres Bit-Field Beispiel:

```
typedef struct
        {
        unsigned char Hour          :6      ;//6 Bits für die Stunden
        unsigned char Min           :7      ;//7 Bits für die Minuten
        unsigned char Sec           :7      ;//7 Bits für die Minuten
        unsigned char Weekday       :3      ;//3 Bits für den Wochentag
        unsigned char Month         :5      ;//3 Bits für den Monat
        unsigned char Year          :8      ;//8 Bits für das Jahr
        }DCF77_BITS_TYPE;
```

typedef union Das seltene UNION. Eine Art verkappter Typecast

```
        {
        DCF77_BITS_TYPE     stbitfDCF_Bits;
        uint32_t            u32DCF_RxBuffer; //    D
        }DCF77_UNION_T;
```

→ **volatile DCF77_UNION_T gstunDCF77_Bits; // union Struct**

MERKE: UNION ELEMENTE teilen sich den selben Speicherplatz. Hier reinschreiben, woanders auslesen.

Schreibe→ gstunDCF77_Bits. stbitfDCF_Bits = 31;

LESE → uart_puts(U32ToBin(**gstunDCF77_Bits**.u32DCF_RxBuffer));

Anwendung der Steuerbits
Bit Setzen und Löschen

Ein mächtiges Werkzeug!!! Ich kann es gar nicht genug betonen

Bsp: Aktion „Motor Start“ global signalisieren

Motor Start → **gu8Motorstatus |=** MOTOR_ENABLED;

Diese *Information* kann nun ÜBERALL IM PROGRAMM abgefragt werden:

```
→ if ( gu8Motorstatus & MOTOR_ENABLED)
→          {
           tue irgendwas…
           };
```

….und auch gelöscht werden, falls Bedarf:

gu8Motorstatus &= ~ MOTOR_ENABLED;

Anwendung der Steuerbits
Ablauf mit Bedingungen organisieren

Ein mächtiges Werkzeug!!! Ich kann es gar nicht genug betonen

```
→ if ( gu8Motorstatus & MOTOR_ENABLED )   //True = ungleich NULL
        {
        if( (gu8Motorstatus & MOTOR_RAMPE_USED ) &&      //Anforderung
NOT→      ! (gu8Motorstatus & MOTOR_RAMPE_START) )        //wenn noch nicht….
            {
            gu8Motorstatus |= MOTOR_RAMPE_START | MOTOR_RAMPE_AKTIVE;
            ……. Alle Aufgaben für Start, wie Speed langsam stellen usw…… Dann
            gu8Motorstatus &= ~MOTOR_RAMPE_START;
            };
        };
```

Wir organisieren hier eine Startsituation

1. ob diese überhaupt angefordert wird,
2. ob diese schon ausgeführt wird und signalisieren den Startmoment **und** den „Aktive“ Status.
3. Die Startsituation kann nach Initialisierung gelöscht werden.

Anwendung der Steuerbits Quittierung von Zuständen

Ein mächtiges Werkzeug!!! Ich kann es gar nicht genug betonen

Im Falle des Ablaufes kann nach Erreichen eines neuen Zustandes durch einfaches Löschen von Zustand Bits die Ausführung in anderen Programmteilen beeinflusst werden.

```
if(  (gu8Motorstatus & MOTOR_RAMPE_USED )  &&        //Anforderung
        (gu8Motorstatus & MOTOR_RAMPE_AKTIVE )  )            //noch aktiv?
            {
             if( Istdrehzahl < Solldrehzahl )
                {
                Istdrehzahl++;
                }
             else
                {
                Istdrehzahl = Solldrehzahl ;
                gu8Motorstatus &= ~MOTOR_RAMPE_AKTIVE;
        gu8Motorstatus |= MOTOR_RAMPE_READY;
                };
            };
```

Typischer Ablauf im 3-Gespann der Anwendung

```
// Interrupt Section
#define STATUS_TICK_EVENT        0x01
ISR( TIMER2_OVF_vect )    //Dieser Vector steht in „iom328p.h“
{volatile static uint32_t u32Merke_T;
gu32_Ticks++;
if( ( gu32_Ticks - u32Merke_T) > TICK_1000MS)
        {
1.      u32Merke_T = gu32_Ticks;     //neue Zeit merken
        gu8Status |= STATUS_TICK_EVENT;
        };          //…..und andere Jobs…..
};
```

```
//Es folgt in der main Loop
while(1).. {
2.  if( gu8Status & STATUS_TICK_EVENT )
        {
3.      gu8Status &= ~STATUS_TICK_EVENT;
        //tue etwas
        usw……
        };
```

Anwendung der Steuerbits
Ausführung in anderen Programmteilen

```
// ************************
ISR( TIMER0_COMPA_vect )     // Hier in einem Timerinterrupt
// ************************
if(  (gu8Motorstatus  & MOTOR_RAMPE_USED )  &&          //Anforderung + logisch UND
            (gu8Motorstatus  & MOTOR_RAMPE_AKTIVE )  )    //noch aktiv?
            {
                        if( Istdrehzahl < Solldrehzahl )
                           {
                           Istdrehzahl++;
                           }
                        else
                           {
                           Istdrehzahl = Solldrehzahl ;
                           gu8Motorstatus  &= ~MOTOR_RAMPE_AKTIVE;
                           gu8Motorstatus  |= MOTOR_RAMPE_READY;
                           };
            };
```

WICHTIG!!!
Merke dir diese
Flagbit -Methode

BYTE	Bit 7	BIT 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0

Ziel |= BITs Setzen von Bits

Ziel &= ~BITs Löschen von Bits

Dezimal	HEX	Binär
1	0x01	0000 0001
2	0x02	0000 0010
4	0x04	0000 0100
8	0x08	0000 1000
16	0x10	0001 0000
32	0x20	0010 0000
64	0x40	0100 0000
128	0x80	1000 0000

Bin	Hex
0000	0
0001	1 0x01
0010	2 0x02
0011	3
0100	4 0x04
0101	5
0110	6
0111	7
1000	8 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

```
// Anlegen von Defines, →Headerfile
#define LED1_BIT          (5)
#define LED1_BIT_MSK      (1 << LED_BIT)
#define LED1_DDRX         DDRB
#define LED1_PORTX        PORTB
#define SET_LED1()        (LED1_PORTX |= LED1_BIT_MSK)    //Makro
#define CLR_LED1()        (LED1_PORTX &= ~LED1_BIT_MSK) //Makro
```

```
// Flag Bits für gu8Status Byte
#define TICK_EVENT          0x01
#define LED_BLINK_MODUS     0x02
#define MOTOR_DIRECTION     0x10
#define MOTOR_ON            0x20
#define INDEX_OK            0x80
// .......usw
```

Ereignis festhalten: (z.B. im Interrupt)

```
gu8Status |= TICK_EVENT;
```

Ereignis auswerten: (wo anders z.B. main)

```
if( gu8Status & TICK_EVENT )
        {
        gu8Status &= ~TICK_EVENT ;  //Clear Flag
        ....tue was zu tun ist....
        };
```

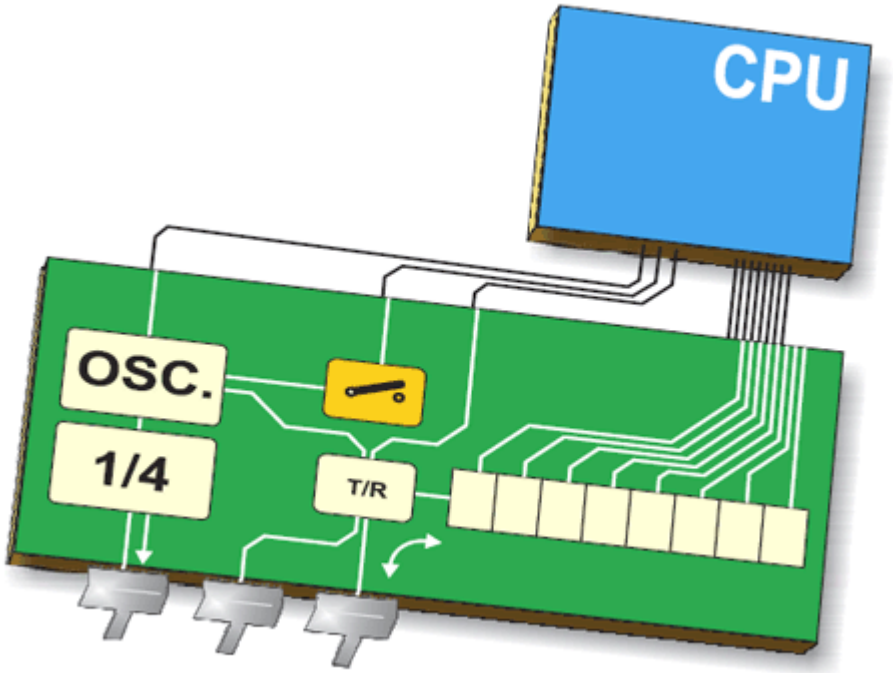
SFR

Aktivierung der Im µController eingebauten autonomen Hardware mit den Special Function Registern

Jetzt ist es Zeit die Handbuch-PDF zum Mikrocontroller aufzuschlagen und sich einen Überblick zu verschaffen.

Was ist alles neben der µC Typischen Umgebung an Zusatzhardware im µC eingebaut ?:

8-16Bit Timer(s),
A/D Converter, Watchdog,
Analog-Comparator,
UART (serielle Schnittstelle)
TWI = (I2C-Bus), Bandgap Reference, SPI=(synchronous serial Port Interface)



Welche SF-Register gehören zu den Komponenten?
Es hilft die PDF quer zu lesen, Spickzettel anzufertigen, bis man Vertrauter wird.

Beispiel: Special Function Register (SFR) mit mixed select: → **MCUCR**

Die einzelnen Bitpositionen sind wie Schalter, die Aktionen auslösen.
Special Function Register MCUCR -->MCU-Control Register
= eine Adresse im Speicher, die direkt mit der Hardware verbunden ist

Bit 7	SM2
Bit 6	SE
Bit 5	SM1
Bit 4	SM0
Bit 3	ISC11
Bit 2	ISC10
Bit 1	ISC01
Bit 0	ISC00

Power Management

Sleep Enable
SE = 1 = Enable Sleepmode

Sleep Modes

INT1 Flanken Erkennung

INT0 Flanken Erkennung

Sleep Modes

SM2	SM1	SM0	Description
0	0	0	IDLE
0	0	1	ADC Noisereduction
0	1	0	Power down
0	1	1	Power save
1	1	0	Standby
1	1	1	Extended Standby

Tabelle: INTx Flanken Erkennung

INT0 wie INT1

ISC01 ISC11	ISC00 ISC10	Description
0	0	INT0 Low Level
0	1	INT0 Any Change
1	0	INT0 Falling Edge
1	1	INT0 Rising Edge

```
#define MCUCR    _SFR_IO8(0x35)
```

MCU Control Register – MCUCR

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 34. Interrupt 1 Sense Control

ISC11	ISC10	Description
0	0	The low level of INT1 generates an interrupt request.
0	1	Any logical change on INT1 generates an interrupt request.
1	0	The falling edge of INT1 generates an interrupt request.
1	1	The rising edge of INT1 generates an interrupt request.

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits 7, 5, 4 – SM2..0: Sleep Mode Select Bits 2, 1, and 0**

These bits select between the six available sleep modes as shown in Table 13.

Table 13. Sleep Mode Select

SM2	SM1	SM0	Sleep Mode
0	0	0	Idle
0	0	1	ADC Noise Reduction
0	1	0	Power-down
0	1	1	Power-save
1	0	0	Reserved
1	0	1	Reserved
1	1	0	Standby(1)
1	1	1	Extended Standby(1)

Note: 1. Standby mode and Extended Standby mode are only available with external crystals or resonators.

Durch einfaches **Suchen mit STRG-F** findet man in der riesigen Manual-PDF die relevanten Punkte.

Die Kunst ist es, den Blick auf die wichtigen Begriffe und Einstellungen zu finden. Das kommt mit der Erfahrung.

Diese Vorgehensweise ist exemplarisch, wie die Steuer-Register ausgewählt und in der passenden Reihenfolge benutzt werden.

Hilfreich ist es eben, die Manual PDF zum µC nicht auswendig zu lernen, aber einen „in etwa“ Überblick über die Möglichkeiten der inneren µC Hardware zu haben und dann einfach die entsprechenden Passagen zusammenzusuchen.
Ideal ist es, die gewonnenen Erkenntnisse in Form von #define in einem „header file„ gleich programmiertauglich umzusetzen. Es lohnt sich!
z.B. #define MCUCR_SLEEP_ENABLE 0x40

Das Interrupt Konzept

Noch etwas aus der Softwarewelt.

….Jetzt etwas Anspruchsvoller.

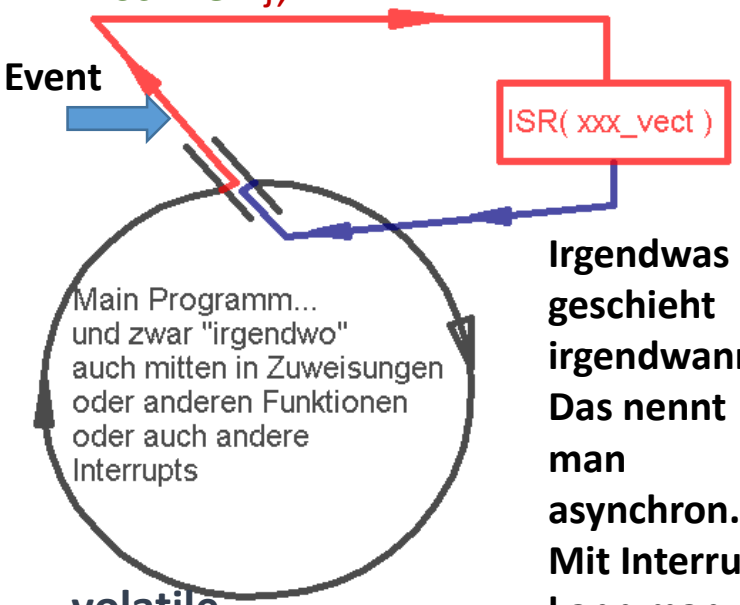
Software oder Hardware Ereignisse exakt auszuwerten.

Genaues Verständnis ist jedoch erforderlich um Fehler zu vermeiden.

Interrupts....ein wichtiges, schwieriges Konzept...

Interrupts sind Ereignisgesteuert.
Dieses ruft eine Vector-Funktion() auf.

ISR (INT0_vect)
{ tue etwas sofort und schnell };



volatile
Datentyp Variablen verwenden!

Irgendwas geschieht irgendwann. Das nennt man asynchron. Mit Interrupts kann man gezielt darauf reagieren.

Interrupts haben es in sich!
Doch sie sind absolut notwendig, um bestimmte Jobs unverzüglich und fehlerfrei zu erledigen.
Leider sind sie deshalb auch ***oft*** eine Fehlerquelle, was schnell mal ein kniffliges Problem beschert.

Interrupts sind meist Hardware Ereignisse:

Externe Signal-flanken → INT0 etc.
PinChange →PCINTnn

Interne Signale von interner Hardware:

Timer overflow
Timer compare A/B
ADC (ready)
ANALOG_COMP (OP)
UART, TWI, EE_READY, SPI
usw..

Die Interrupt Vector Tabelle in „iom328p.h“

```
/* Interrupt Vectors */   /* Interrupt Vector 0 is the reset vector. */
#define INT0_vect        _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect        _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect      _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect      _VECTOR(4)   /* Pin Change Interrupt Request 0 */
#define PCINT2_vect      _VECTOR(5)   /* Pin Change Interrupt Request 1 */
#define WDT_vect         _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect _VECTOR(8)   /* Timer/Counter2 Compare Match A */
#define TIMER2_OVF_vect   _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect  _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect   _VECTOR(13)  /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect   _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect      _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect     _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect   _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect     _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect          _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect     _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect  _VECTOR(23)  /* Analog Comparator */
#define TWI_vect          _VECTOR(24)  /* Two-wire Serial Interface */
#define SPM_READY_vect    _VECTOR(25)  /* Store Program Memory Read */
```

C-Code Bsp:

```
ISR( INT0_vect )
  {
   schnelles.....
   keine loops
  };
```

Interrupt Routine

Der Umfang der Interrupt-Vector-Tabelle ist stark von den Hardwarefähigkeiten des Mikrocontrollers abhängig. →
…einfach die passende „iom328p.h“ durchsuchen. (Da ist es dokumentiert)
INT-Vektoren sind feste Speicheradressen, deren Inhalt die Einsprungadressen zu den Interrupt-Service-Routinen sind, die der Compiler erzeugt.
Diese ISR-Funktionen() werden im Interrupt-Fall aufgerufen.

Interrupt→Call→(ISR_Funktionsadresse())←“Feste Vektoradresse.

```
#define INT0_vect      _VECTOR(1)
#define INT1_vect      _VECTOR(2)
```

Also schreiben wir im C-Code die Funktion:

```
ISR   INT1_vect()                    // Interrupt Service Routine
{
//Tue irgendwas ….  z.B.   gu32Ticks++;
};
```

Diese Funktion wird nur vom Interrupt aufgerufen!
ACHTUNG:
Globale Variablen, die nur im Interrupt verändert werden, müssen (**volatile**) deklariert sein

```
volatile unsigned int gu32Ticks;
```

Der µC interne Tick-Counter

…ist eine einfache Variable, die von einem Hardwarezähler im Mikrocontroller hochgezählt wird. Dies geschieht **~244 mal/Sekunde**
gu32_Tick → ++ → „globale unsigned 32Bit Variable“ namens Ticks
Damit kann man alle Zeitvergleiche und regelmäßige Jobs überwachen oder auslösen.
Ein unerwartet mächtiges Steuerwerkzeug!

```
#define TICK_1500MS 423
#define TICK_1000MS 244
#define TICK_500MS 122
#define TICK_250MS 71
#define TICK_100MS 24
```

Initialisierung beim Programmstart:
u32MerkeT = gu32_Ticks; //Jetzt

```
if( (gu32_Ticks - u32MerkeT)  >  TICK_500MS)        //(jetzt – früher) -> Differenz
                    {
                    u32MerkeT = gu32_Ticks;         //merke dir wieder das „Jetzt“
                    PORTC ^= RED_LED;               //EXOR = BLINKEN LASSEN
                    };
```

ZIEL: regelmäßige Aktivität via Timer, Prinzip:

Hier ein Beispiel wie man es in anderen Quellen finden kann, Nicht ganz ideal:

→ Erzeugung eines Timer-Events.

```
/*! \brief 8 Bit Timer interrupt to control the sampling interval  */
ISR( TIMER0_OVF_vect )      //Interrupt Service Routine
{
  volatile static uint16_t    i  =  0;
  if( i < TIME_INTERVAL )        //~ z.B. 128
            {
   i++;                    // interrupt Zähler  +1
            }
   else
            {
            // Hier wird der „Außenwelt“ das Ereignis bekannt gegeben
   gFlags.pidTimer = TRUE;
   i = 0;                  // interrupt Zähler zurücksetzen
            };
};
```

Initialisierung des Timers (vor dem while(1))

```
 TCNT0 = 0;                                  //Vorteiler
            //  TCCR0 = (1<<CS00);           // %1
  TCCR0 = (1<<CS02);                         // %256
  TIMSK |= (1<<TOIE0);
```

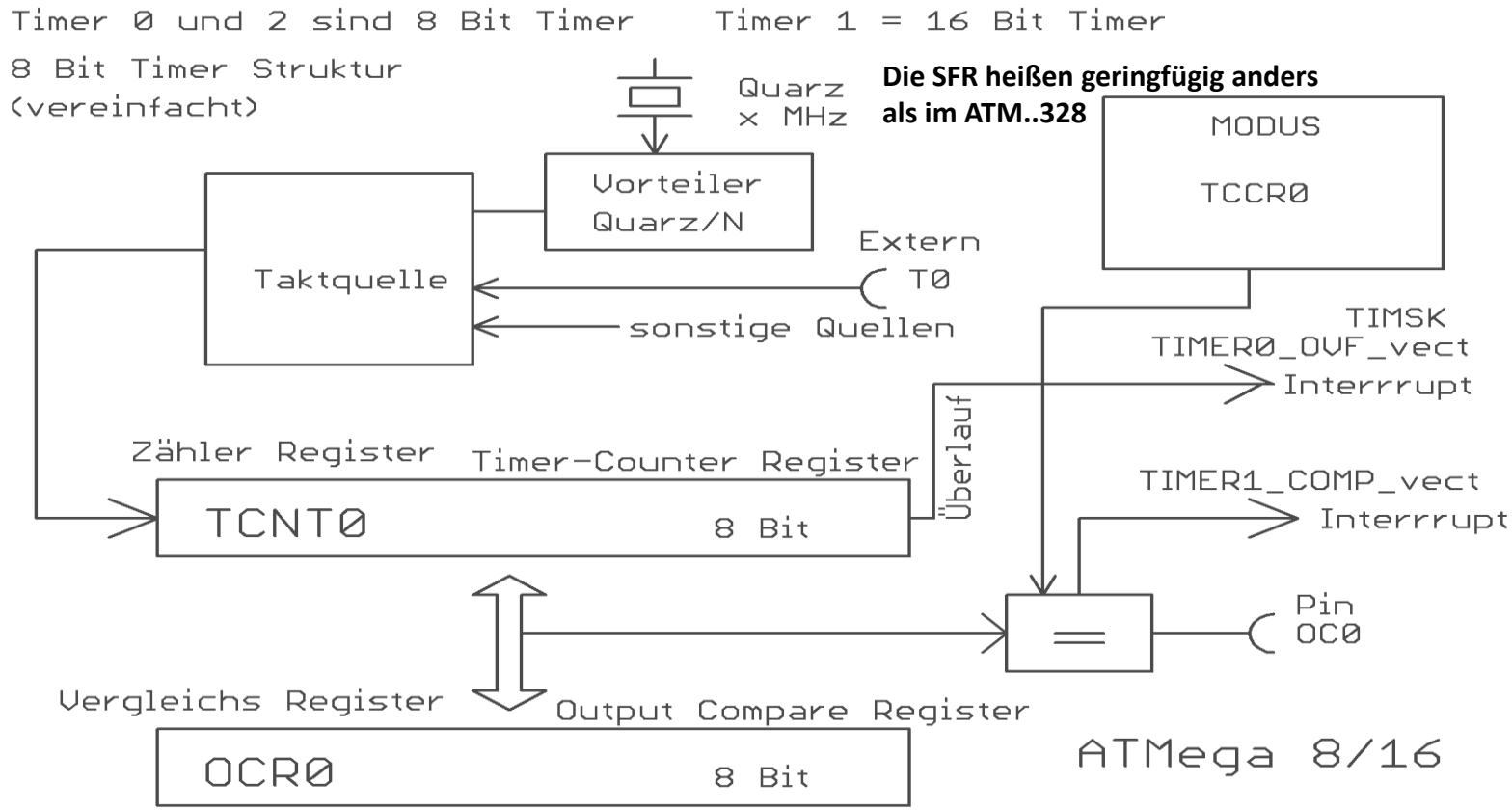
Auswertung des Timerinterrupt TICK-Signals z.B. im main→while(1)-loop

```
while(1)
    {  ....
            blah        blah        blah……
      if(gFlags.pidTimer)
        {
            //Hier ist der JOB
            referenceValue = Get_Reference();
            measurementValue = Get_Measurement();
            inputValue = pid_Controller(referenceValue,
            measurementValue, &pidData);
            Set_Input(inputValue);

          gFlags.pidTimer = FALSE;
        };
      };
```

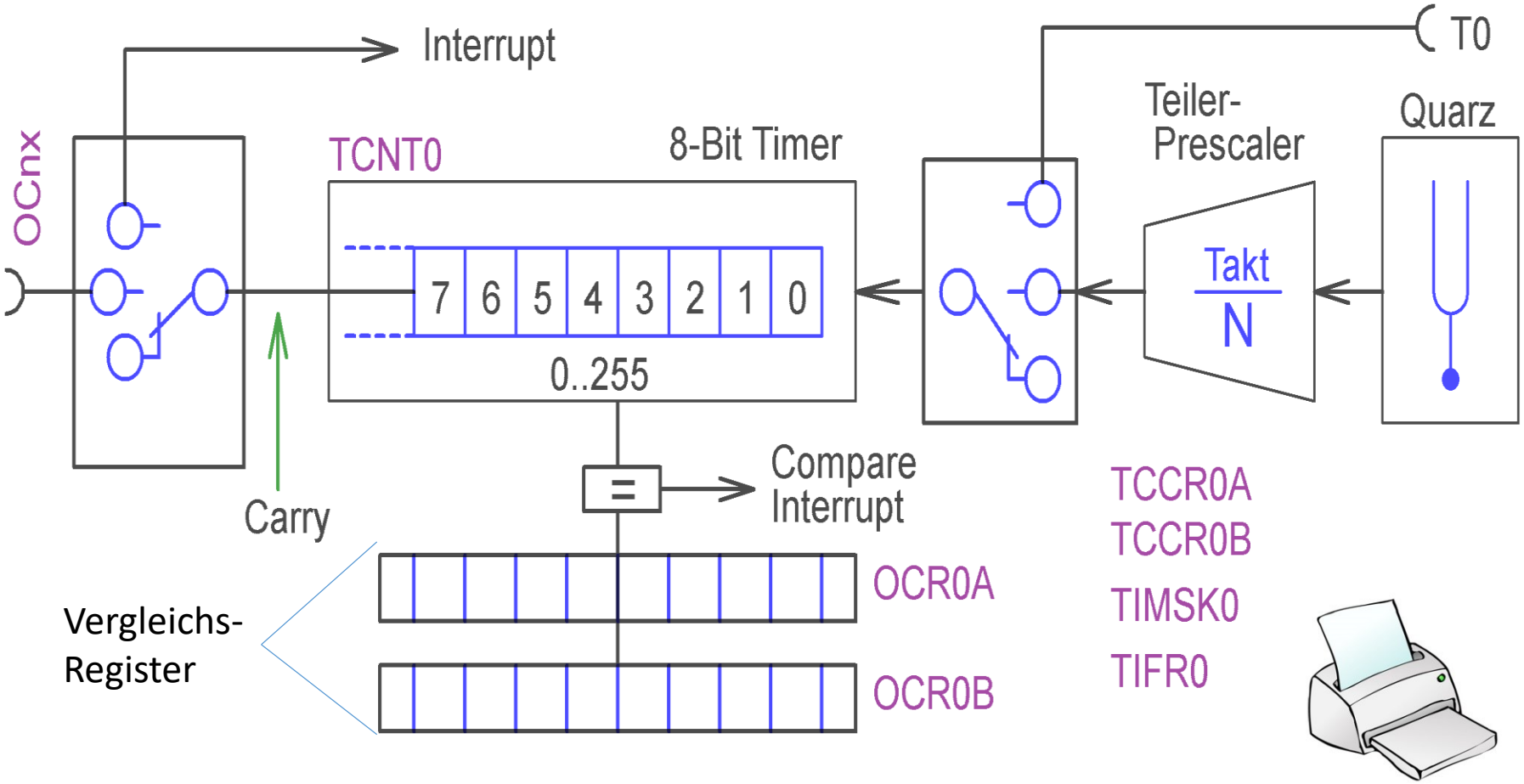
8-Bit Timer/Counter 0 für festenTakt (Galeere)

Wir brauchen so gut wie immer einen Taktgeber. Ähnlich wie in der Galeere. Der Taktgeber (**Tick-Counter**) gestaltet bzw. synchronisiert Abläufe nachvollziehbar langsam, z.B. eine 2 sekündliche Textausgabe, so dass Zeit zum Lesen bleibt.

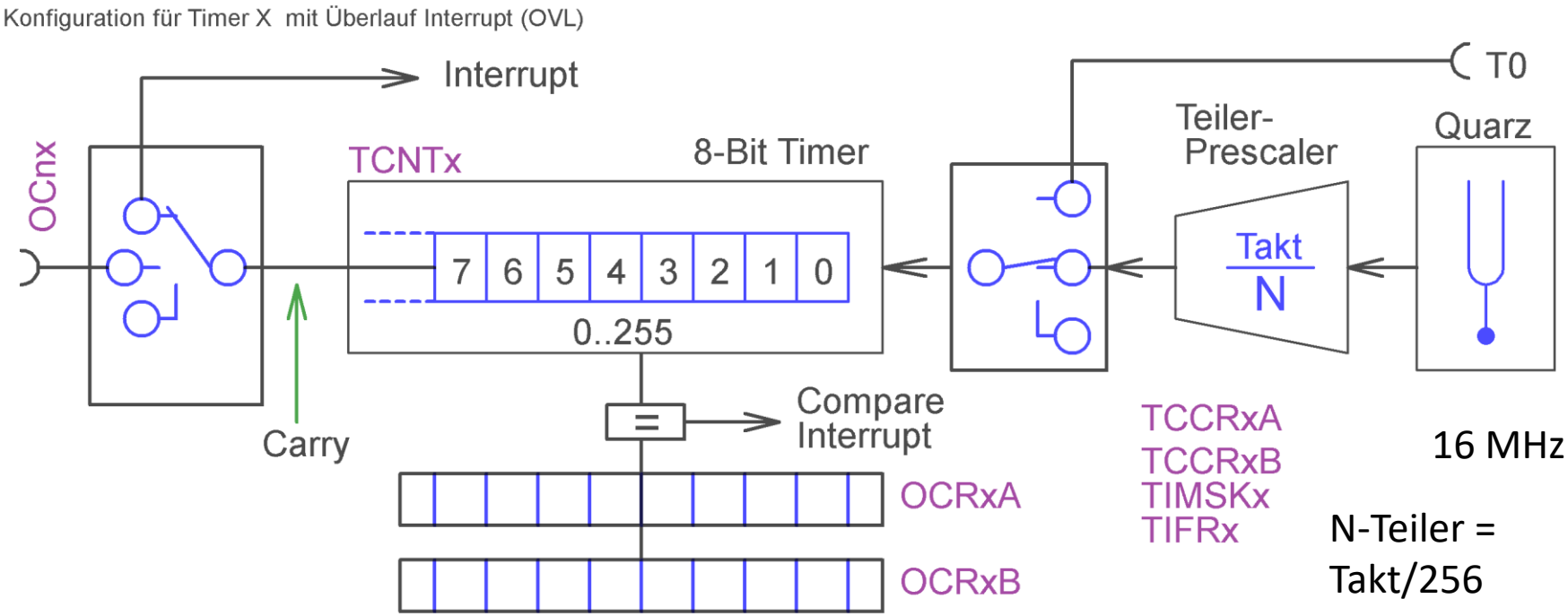
Es ermöglicht Aktivitäten wie eine Blinkfrequenz, die nicht auf hässlichen „Delays“ basiert, was nichts anderes ist als Prozessorzeitverschwendung.



Ein Blick in die Handbuch-PDF →**ATMega328.pdf** hilft weiter.
Um das Innenleben zu verstehen ist es sinnvoll sich eine grobe **Stichwort Liste** und **Skizze** der wesentlichen Komponenten zu machen, sowie die relevanten **SF-R**egister namentlich zu erfassen →„related Terms“ mit [STRG+F]
Aus den vielen Seiten der PDF kann man so relativ einfach die wesentliche Information ziehen. Suche: Timer0 → Registername → Funktionstabellen usw..



Timer-x Überlauf Interrupt



Ziel ist es, einen für µC langsamen Zähler zu erzeugen.
Ein interner kleiner 8-Bit Timer soll dies bewerkstelligen. Wir wählen mal den Timer 2.
Wir suchen uns also mal die dazugehören SF-Register zusammen. Dann kann man in den SFR-Tabellen suchen, was die einzelnen Bits darin bedeuten.

TCNT2 OCR2A OCR2B TCCR2A TCCR2B TIMSK1 TIMSK2 TIFR2

TIPP: OCRx = Quarz / Vorteiler / Gewünschte

Einstellung des Vorteilers für den 8-Bit-Timer 2

TCNT2 OCR2A OCR2B TCCR2A TCCR2B TIMSK1 TIMSK2 TIFR2
Die Suche nach TCCR2B ergibt:

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Wir wissen, dass alle in den Registern vorkommenden Kürzel in der „iom328.h“ Datei definiert sind. Bsp.: In der Form **#define CS21 1** . Jetzt kann dieses leicht angesprochen werden. Form: → (1 << CS21). Bsp. **TCCR2B |= (1 << CS21);** usw..

Ein Volldurchlauf von TCNT2 : 00000000 bis 11111111 ist schon eine Zähler Teilung / 256. Dann kommt der Überlauf-Übertrag 1-00000000. Dies ist die Zählsituation.

TCNT2 – Timer/Counter Register

Bit	7	6	5	4	3	2	1	0	
(0xB2)	TCNT2[7:0]								TCNT2
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR2B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
(0xB1)	FOC2A	FOC2B	–	–	WGM22	CS22	CS21	CS20	TCCR2B
Read/Write	W	W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Mode	WGM22 WGM2	WGM21 WGM1	WGM20 WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Vorgabe 16MHZ Quarz.
8Bit Timer OVL ist selbst /256;
Gewünschter „langsamer“ Takt irgendwo zwischen ~ 200… 300 mal /Sekunde

Um nun also diesen „langsamen“ Zählerüberlauf zu erreichen brauchen wir also eine Vorteiler der **16^6/256 =** 62500/Sekunde hardwaremäßig feste Teilung.
Wir wollen in die Nähe von ~200..300 Zählungen pro Sekunde.
Testweise kalkulieren wir mal den nötigen Vorteiler und nehmen die vorgegebene feste Teilung von 62500 / 300 = 208,333. In der Tabelle finden wir als nächsten Näherungswert T2 /256 → Jetzt genau: 62500/256 = *244.140625*. Das passt !
Wir wählen /N=256→ **CS22** und **CS21** von **TCCR2B** müssen gesetzt werden.

Table 17-9. Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler) ★
1	1	1	clk_{T2S}/1024 (From prescaler)

Wir Programmieren:

```
TCCR2B |=
(1 << CS22) | (1 << CS21);
```

Der Zähler läuft ab sofort nun mit Vorteiler clk/256

Timer2 Overflow Interrupt aktivieren

Nach einigem stöbern in der „ATMega328.pdf“ finden wir:

TIMSK2 – Timer/Counter2 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x70)	–	–	–	–	–	OCIE2B	OCIE2A	TOIE2	TIMSK2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Darunter stehen gleich die Beschreibungen der einzelnen Bits (FLAGS). Uns interessiert das **TOIE2** Bit:

- **Bit 0 – TOIE2: Timer/Counter2 Overflow Interrupt Enable**

When the TOIE2 bit is written to one and the I-bit in the Status Register is set (one), the Timer/Counter2 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter2 occurs, i.e., when the TOV2 bit is set in the Timer/Counter2 Interrupt Flag Register – TIFR2.

Wir aktivieren den Überlauf-Interrupt (OVL2) mit:

TIMSK2 = TIMSK2 **|** (1<< **TOIE2**); Kurzform: TIMSK2 |= (1<< **TOIE2**);

Im C-Code steht: **ISR(TIMER2_OVF_vect) {….}; //interrupt Service Routine**

Die Interrupt Vector Namen stehen in der „iom328.h“

C-Code – Tick-Counter mit Timer 2 und Interrupt


```
#define LED1_BLINK      (1<<PC0) //LED an BIT 0
#define TICK_1000MS   244U
// GLOBALE VARIABLEN
volatile uint32_t gu32_Ticks;        //  T2 TimerCounter
// Interrupt Section
ISR( TIMER2_OVF_vect )    // Dieser Vector steht in der „iom328p.h“
{              // Interrupt Variablen müssen "volatile" deklariert sein
gu32_Ticks++;            //hochzählen → 244*/Sekunde
};
int main(void)           // *******  MAIN *************
{
static   u32Now =0;
DDRC |= LED1_BLINK;                          //Bit 0 von PORTC als Ausgang
TIMSK2 = ( 1<< TOIE2 );                      //Interrupt aktivieren
TCCR2B = (1 << CS22) | (1 << CS21);          //Vorteiler   clk/256 einschalten
sei();        //enable Interrupt
for(;;)       // oder while(1)
        {
        if( (gu32_Ticks - u32Now) >  TICK_1000MS )  // Prüfe, ob Zeitdifferenz erreicht ist.
                {
                u32Now = gu32_Ticks;             //neue Zeit merken
                PORTC ^= LED1_BLINK;             // ab  jetzt langsame ~1 Sekunden Aktion
                }; }; };     // sorry die miese Fomatierung. SEITE zu klein
```

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Wir finden 2 Control Register für Timer 0, weil eines nicht mehr ausreicht. Dazu brauchen wir die Tabelle der einstellbaren Funktionen.

COMx(X/B)n	befasst sich mit den Outputpin Modi des Timer/Counter
WGMnn	befasst sich mit den Arbeitsmodi des Timer/Counter
CSnn	befasst sich mich dem Vorteiler des Timer/Counter
FOCxn	Force Output Compare: only NON-PWM Mode. Strobe/Trigger Compare-Match case für Synchron Operationen.

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Kleine Besonderheiten sind zu beachten : Bsp.: Case: WGM02:0 = **FAST PWM Mode**

Table 14-6. Compare Output Mode, Fast PWM Mode(1)

COM0B1	COM0B0	Description
0	0	Normal port operation, OC0B disconnected.
0	1	Reserved Kein Toggle
1	0	Clear OC0B on Compare Match, set OC0B at BOTTOM, (non-inverting mode)
1	1	Set OC0B on Compare Match, clear OC0B at BOTTOM, (inverting mode).

TCCR2A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
(0xB0)	COM2A1	COM2A0	COM2B1	COM2B0	–	–	WGM21	WGM20	TCCR2A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

Vorteiler Register des Timer/Counter 0

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

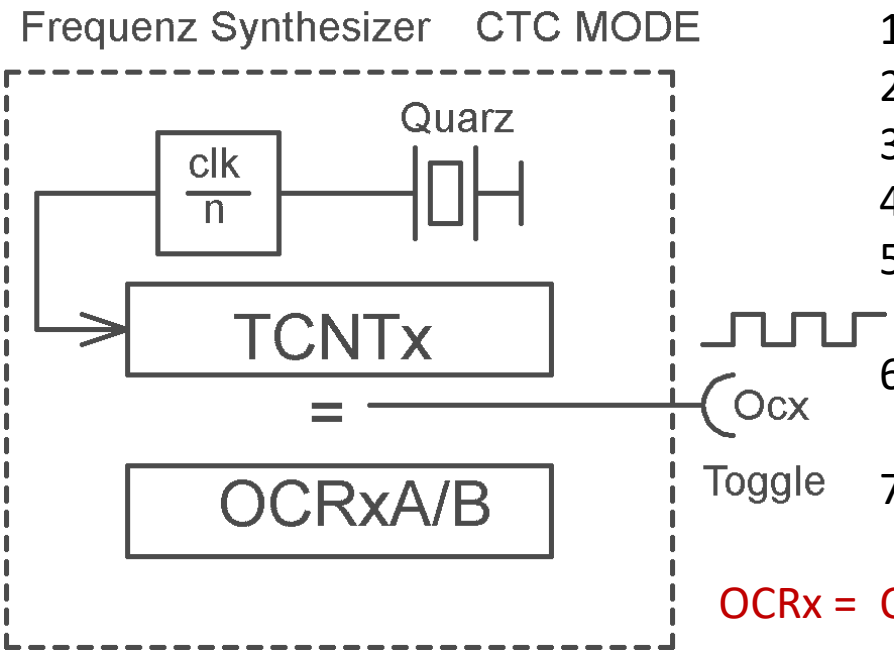
Anwendungsbeispiel: Clk/1024 → TCCR0B |= (1<< CS02) | (1<<CS00);

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Synthesizer mit Timer/Counter

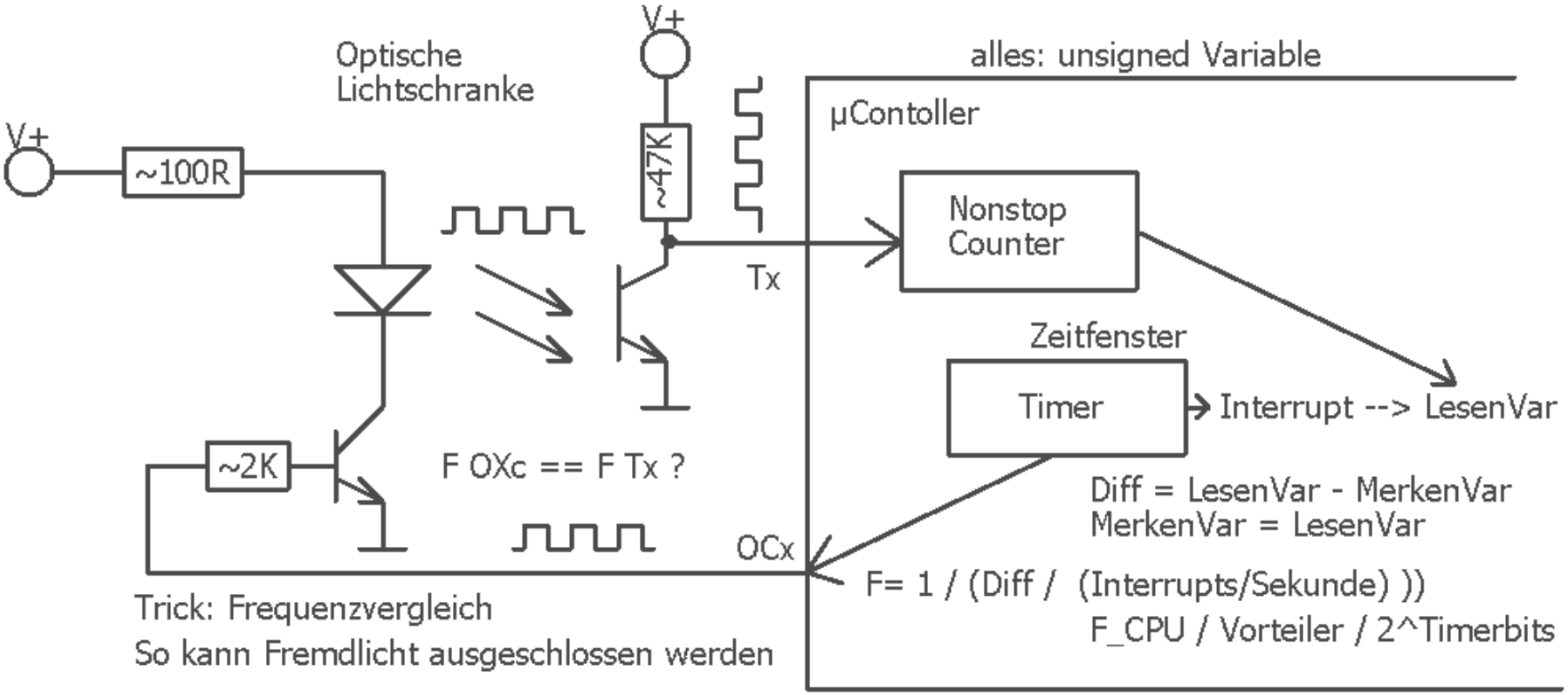
Mit dem Vorteiler und dem Compare Register kann also die Zeit eingestellt werden mit der periodisch der Compare-Match ausgelöst wird.
Wiederholende Vorgänge haben eine feste, jedoch einstellbare .
Wir brauchen einen Modus in dem das TCNTx Register wieder auf 0 gestellt wird.
Dieser Compare-Match kann direkt mit Port-Toggle an einen Ausgang OCx gelegt werden, oder an einen Interrupt. Dies geschieht im CTC Modus der mit WGMxx eingestellt wird. **CTC Clear Timer on Compare Match (Auto Reload)**



1. Soll- bestimmen
2. Vorteiler kalkulieren
3. Passenden OCRx Wert bestimmen.
4. CTC-Modus einstellen
5. Compare Fall bestimmen
   Ausgang und/oder Interrupt
6. Optionale Interrupt Vector Funktion
   im Code implementieren
7. Am Schluss kalkulierten Vorteiler
   aktivieren.

OCRx = Quarz / Vorteiler / Gewünschte

Trick: Sichere Lichtschranke durch vergleich

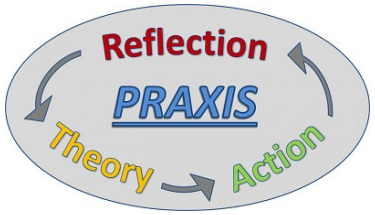


Programmierübung:
Counter, mit einer Lese_Torzeit, als zähler. F= Counts/Zeit
Zugleich wird eine erzeugt.
Nur wenn diese Übereinstimmt, wird das Signal akzeptiert.
Das Ergebnis wird dann z.B. in einem Event-Flag ausgewertet

Jetzt wird es Zeit endlich etwas in Bewegung zu setzen

Der SERVO und PWM Puls Erzeugungsgabe mittels Timer

Praxis: Servo – Timer - PWM



Da eben µKontroller und dessen Software untrennbar mit der Hardware verknüpft sind, wird es Zeit beides in guter Zusammenarbeit zu bringen.

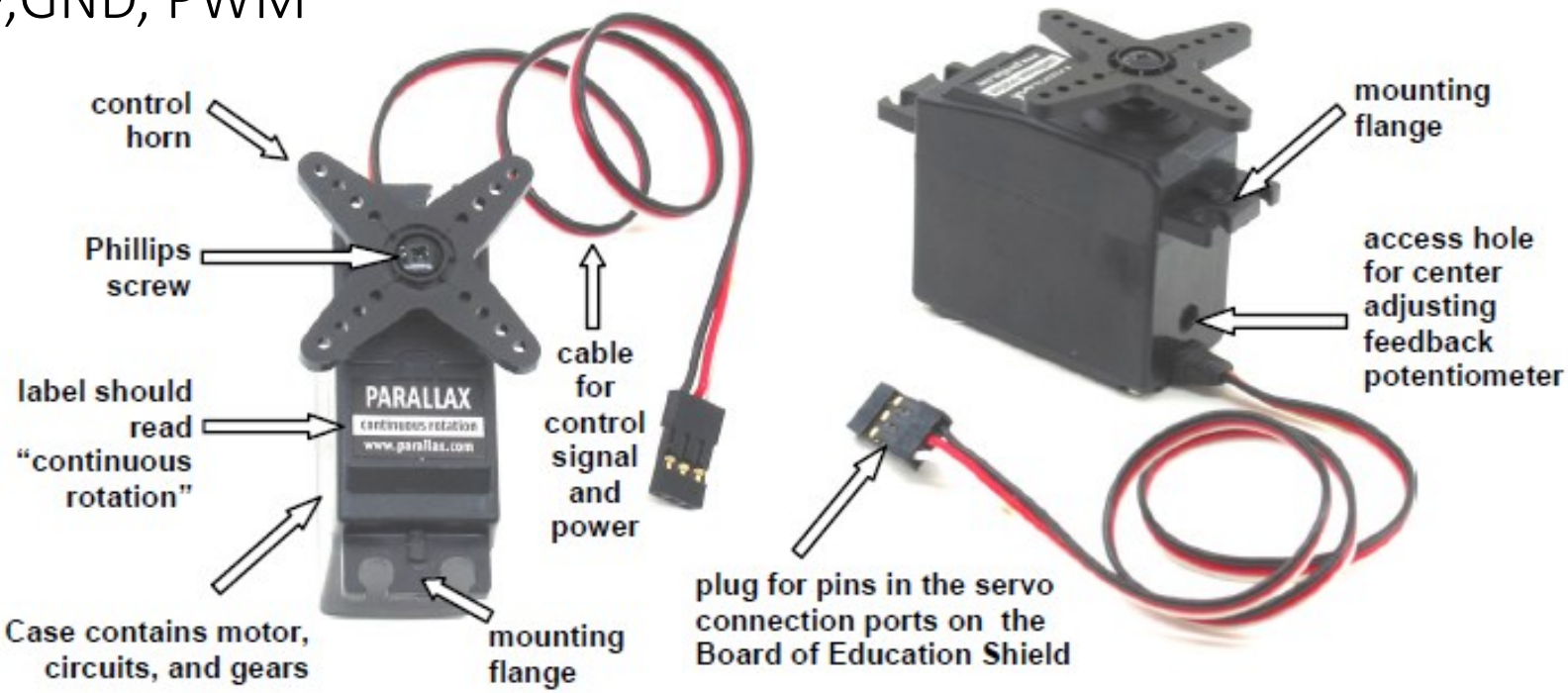
Ein Beispiel ist Servos, auch Leistungs-Servos mit dem µController anzusteuern.
Servos sind sehr beliebt, und können für 1000+1 Zweck gebraucht werden.

Die Anforderungen bestimmen den Lösungsweg:
* Hohe Kraft

* Mehrere Servos unabhängig,
aber in Abhängigkeit zueinander (Roboter)

* Hohe Auflösung ..
Oder alles Bedingungen zusammen.
Dennoch soll der Aufwand übersichtlich bleiben.
Merke: **Gute Lösungen sind verblüffend einfach**.

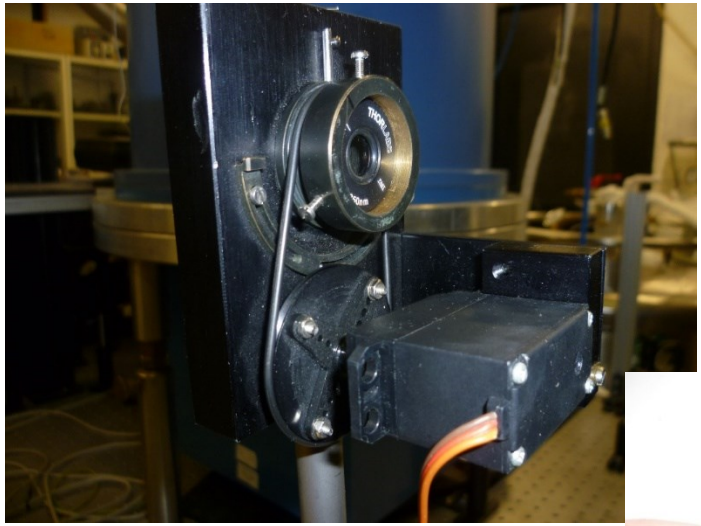
Servo mit 3Leiter Anschluss V+,GND, PWM

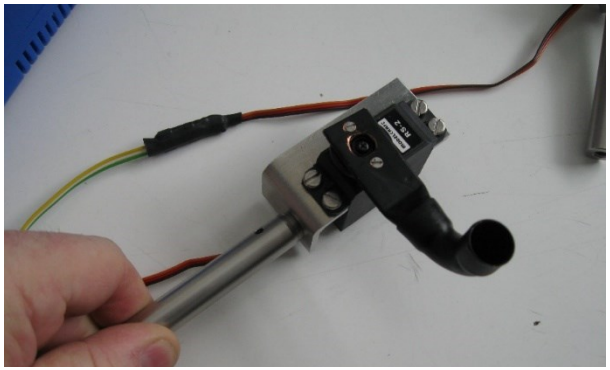


Angesteuert wird ein mit einem (stromschwachen) **PWM** Signal. Dieses trägt die ‚Information' der Servoposition
PWM = Pulsweitenmodulation.

Belegung:
Braun/schwarz = GND
ROT o.ä. = 5..7V
Orange/weiß o.ä. =
PWM Signal 1mS..2mS

Servo Anwendungsbeispiele





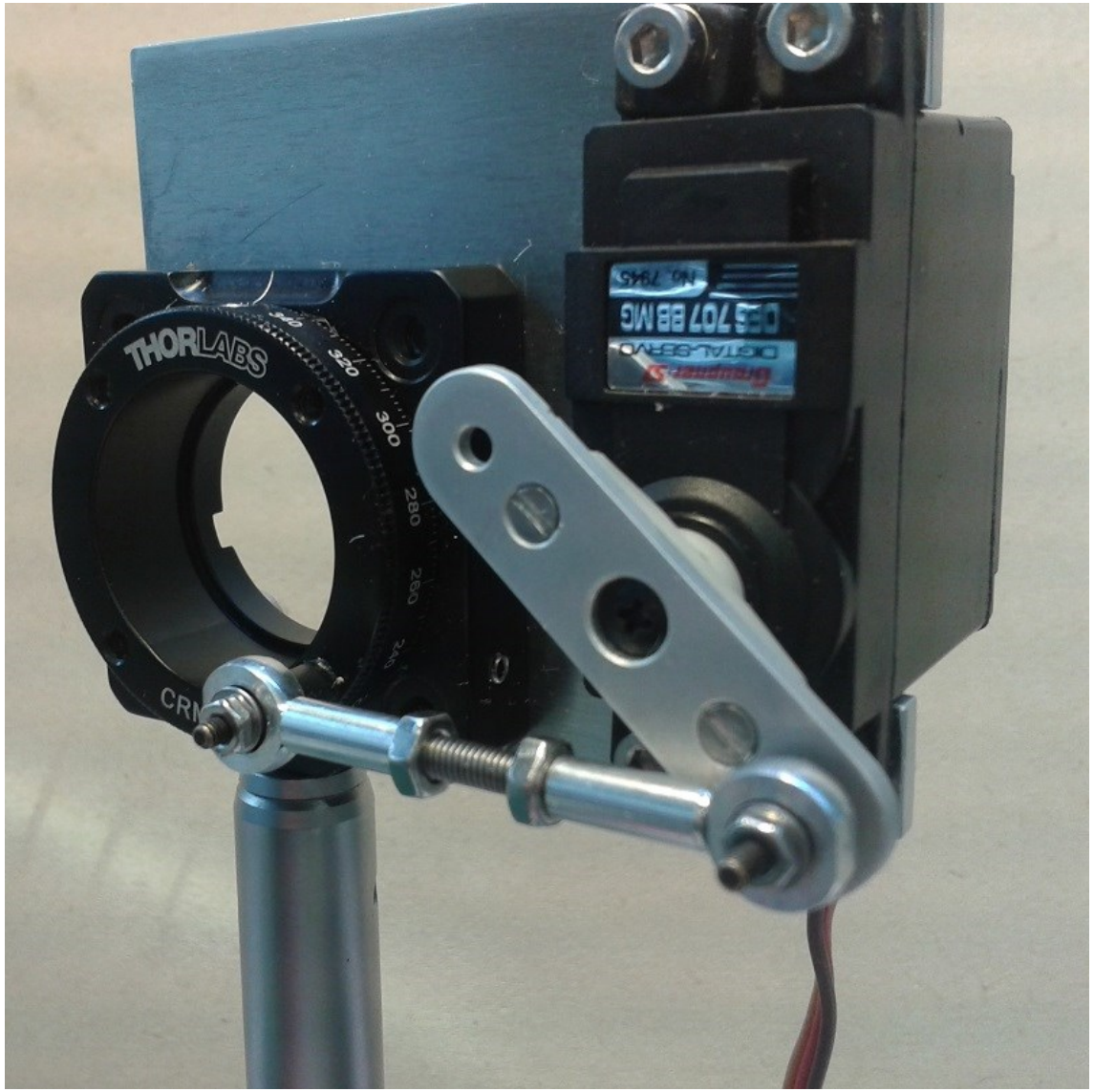
Servo **dreht** im Laserlabor ein Laser Lambda/4 Blättchen.

Laserfangrohr an ein Servo montiert, das sich auf einen textbasierten Befehl so dreht, dass der Laser in das Rohr fällt und sich darin verliert.

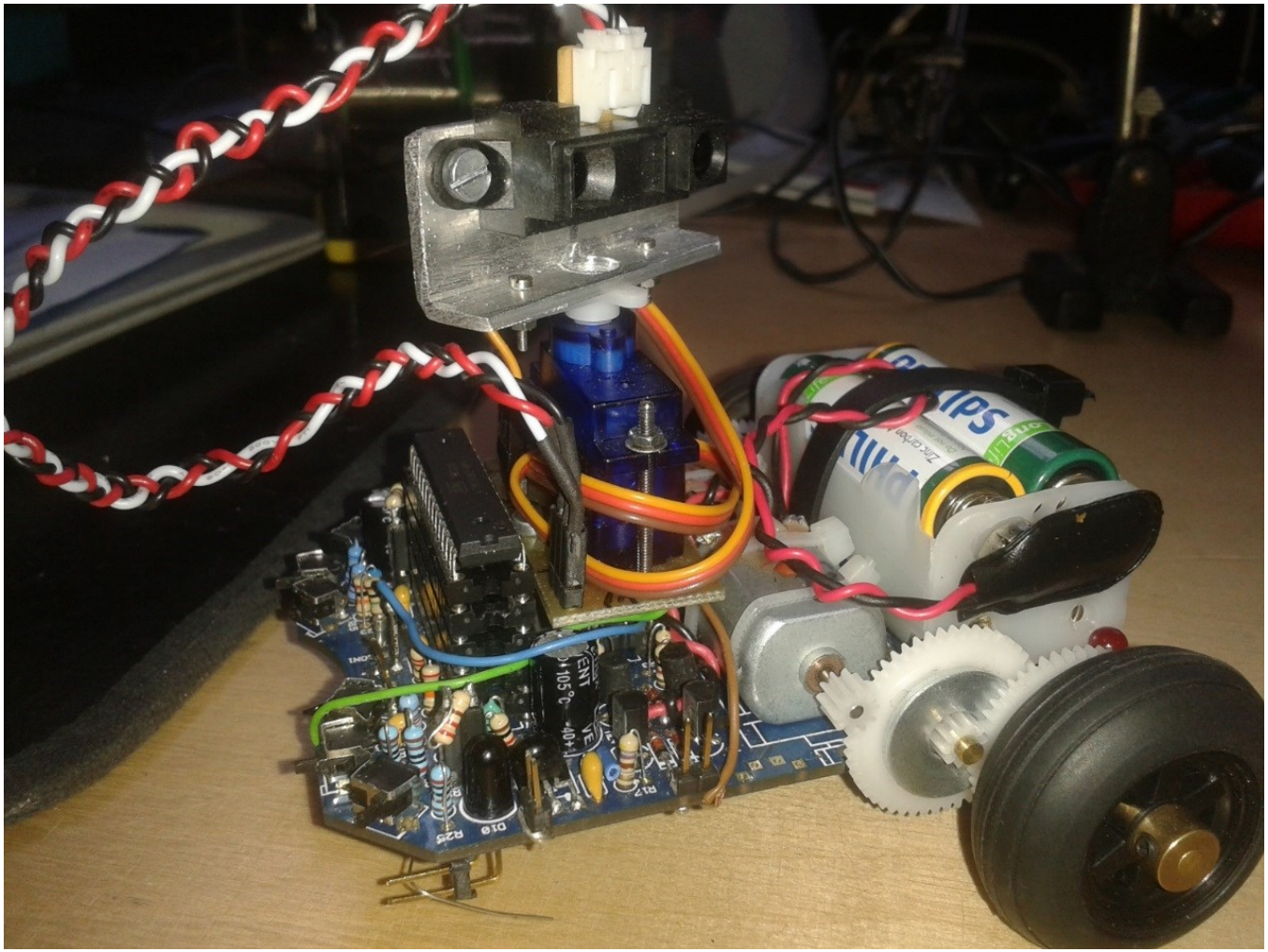
Youtube: http://youtu.be/Ads4GSefew

Youtube Link: http://youtu.be/yundxoedF1s
und https://youtu.be/ih3EetI1m4g

Laser Filter Drehvorrichtung in beliebige Winkel wie 90 Grad.

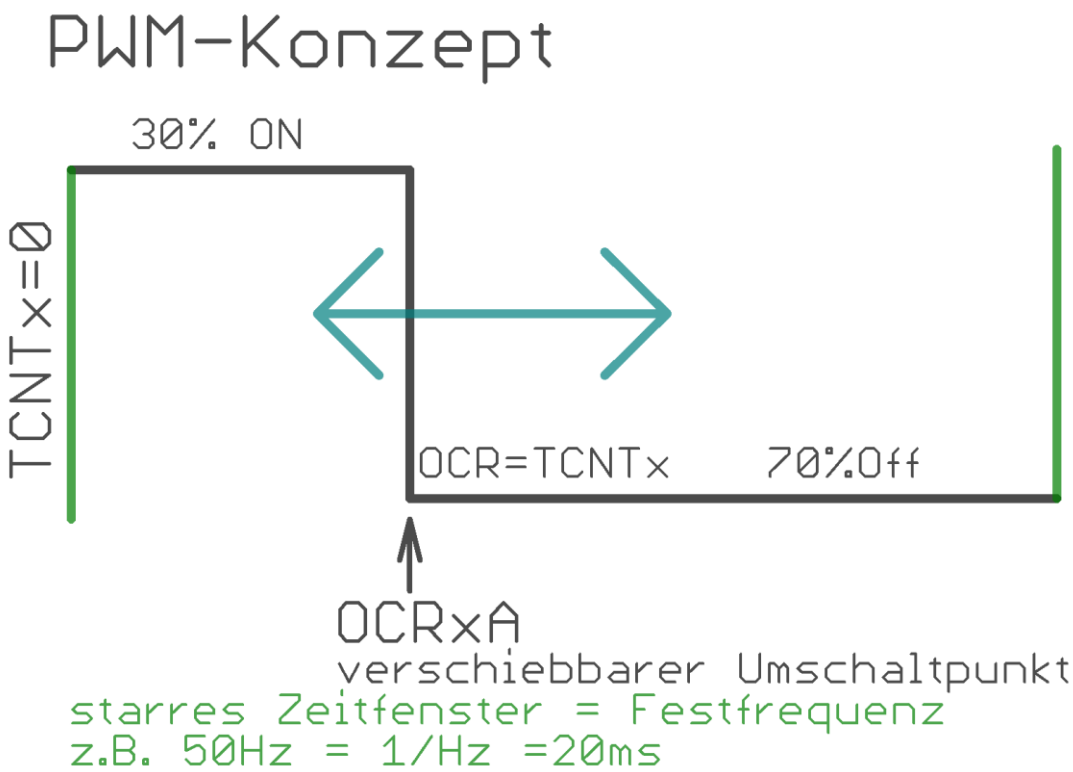


Servo dreht bzw. schwenkt einen IR-Entfernungsmesser wie ein Radar.(im Asuro DLR Robot)
Youtube: http://youtu.be/dstx46pDZzQ



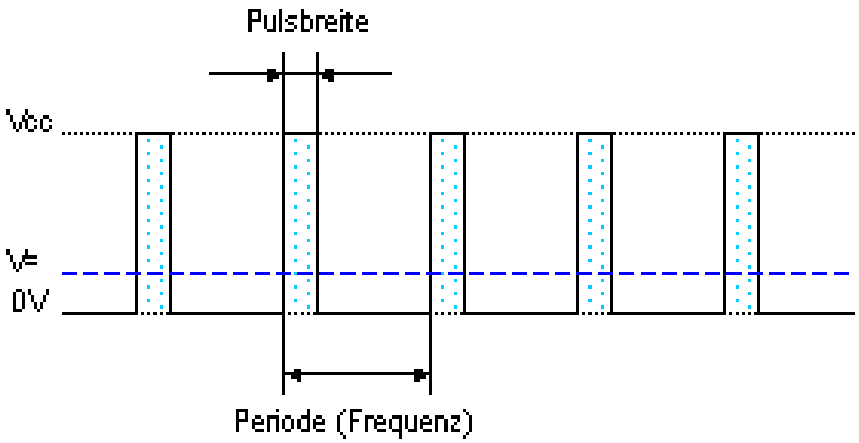
Puls-Weiten-Modulation - PWM

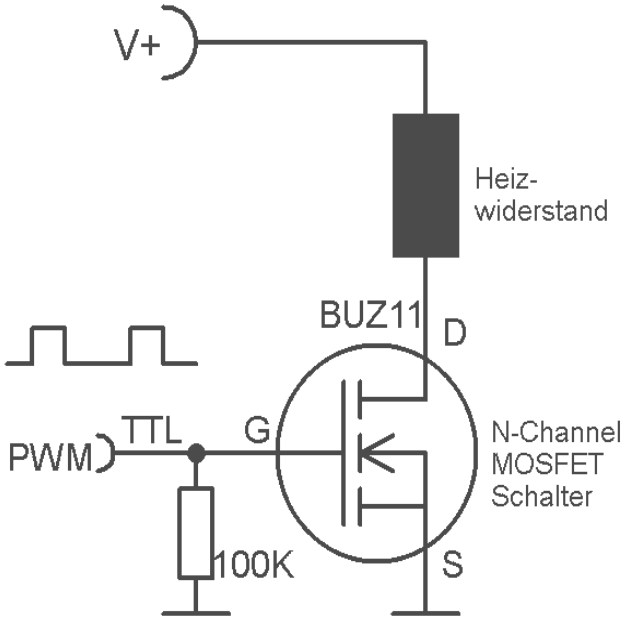
PWM ist einfach eine prozentuale Einschaltzeit mit einem festen Zeitrahmen (). Somit auch eine Leitungsregelung. Zudem kann mit Hilfe der PWM über nur einen Draht komplexe Information transportiert werden.



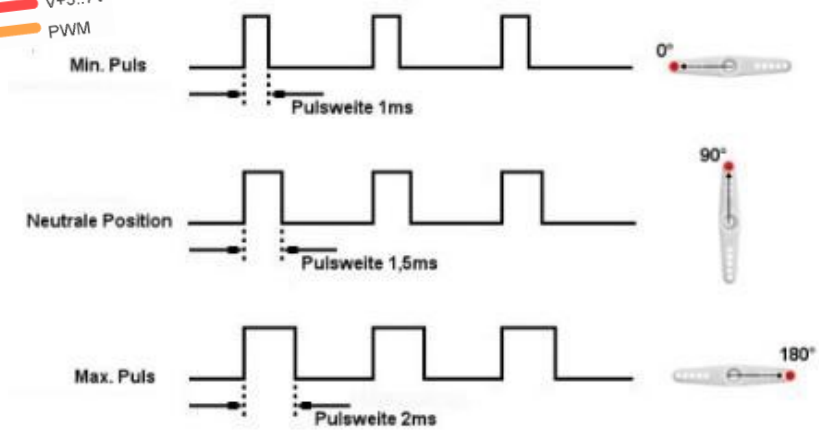
PWM Anwendungen

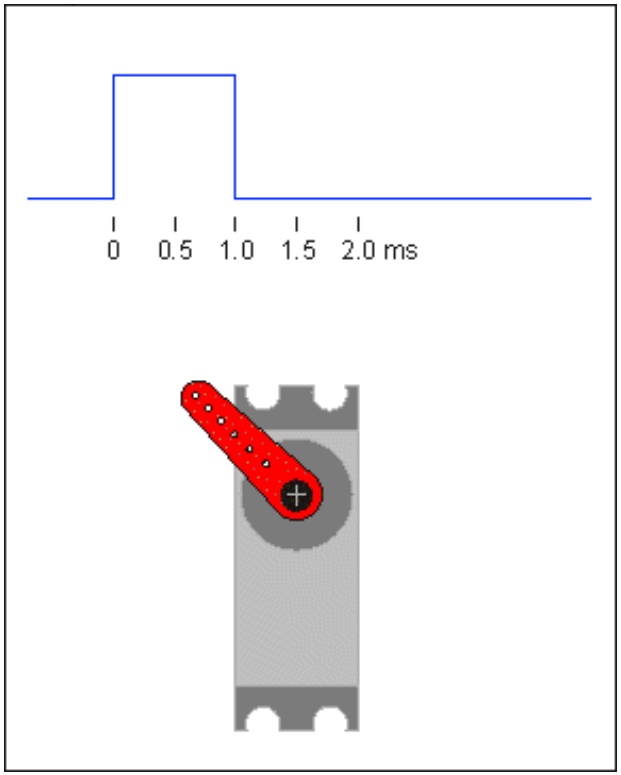
Mit PWM können durch das Pulsbreiten-Verhältnis der Einschaltzeit - ohne großen Leistungsverlust des elektronischen Schalters (MOSFET sind ideal) - große Leistungen gesteuert werden.
Der gezeigte BUZ11 kann **15A** gegen Minus (=GND=0V) schalten.
Das sind bei 40V schon 0..600 WATT.. Je nach PWM-Einschaltzeit.
Siehe den **Sicherheits-Pulldown**-Widerstand am Gate.
Verlustleistung am Schalttransistor:
OFF = 0Watt → ON= ~0 Watt Verlust wird nur bei Transienten erzeugt.





PWM an **SERVO**S aus dem Modellbau

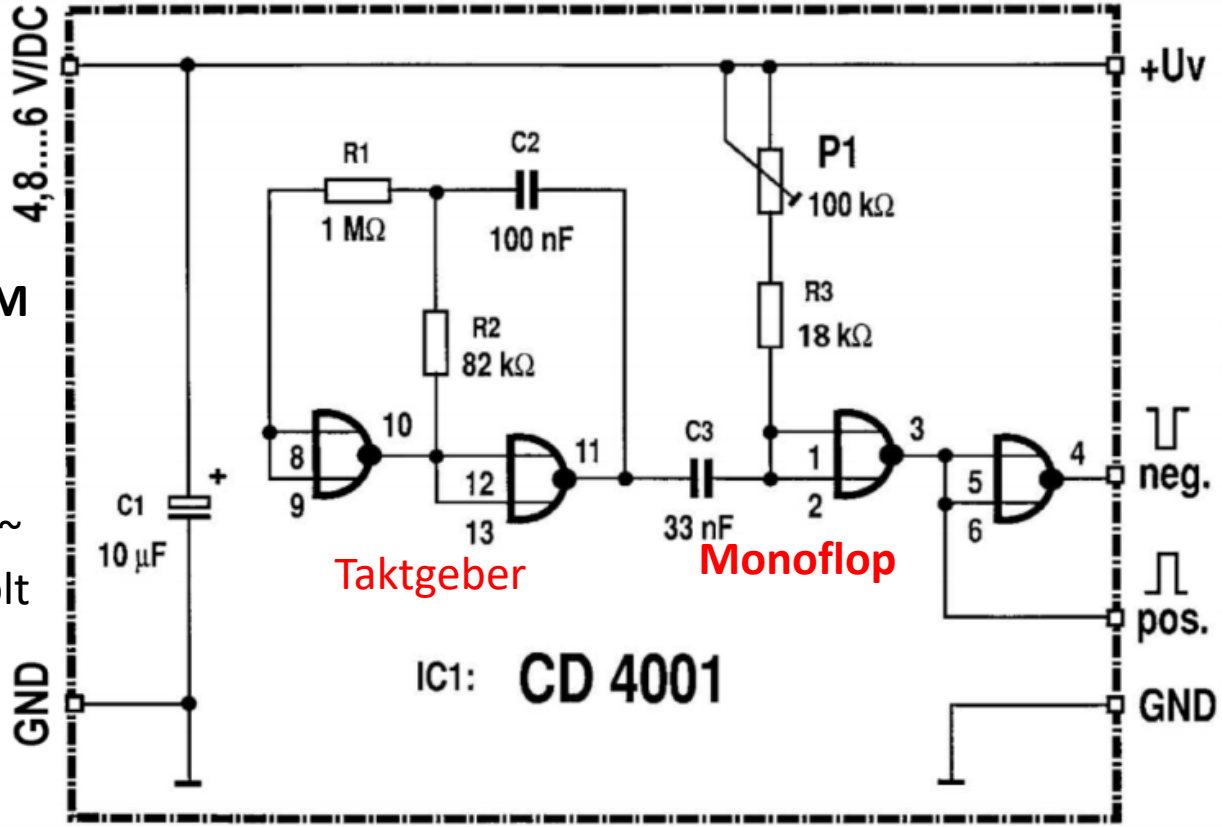




Servos haben eine PWM Steuerung
Dieser folgt mechanisch der PWM-ON Zeit.
„Mindestens“ 1 Millisekunde = links
2 Millisekunden = rechts → Mitte = ~1.5mS
Die PWM = ~50Hz =1/T = 20 mS
Achtung: Anschläge unbedingt vermeiden
Merke: **Erst PWM a**nlegen,
dann erst V+ Spannung anlegen,
oder mit FET GND gegen 0V=Masse schalten

PWM Signal mit klassischer Elektronik erzeugen

Um ein **Servo** zu steuern brauchen wir also ein **PWM** mit **1 mS** bis **2 mS**
Also **niemals 0 mS**.
(=kein Signal)
Dieser Puls wird ungefähr ~ 50 mal/Sekunde wiederholt (**50Hz**)
Realisiert wird dies mit einem 50Hz Oszillator und einem einstellbaren Monoflop.



CMOS Baustein
CD4001 = 4* Nor And

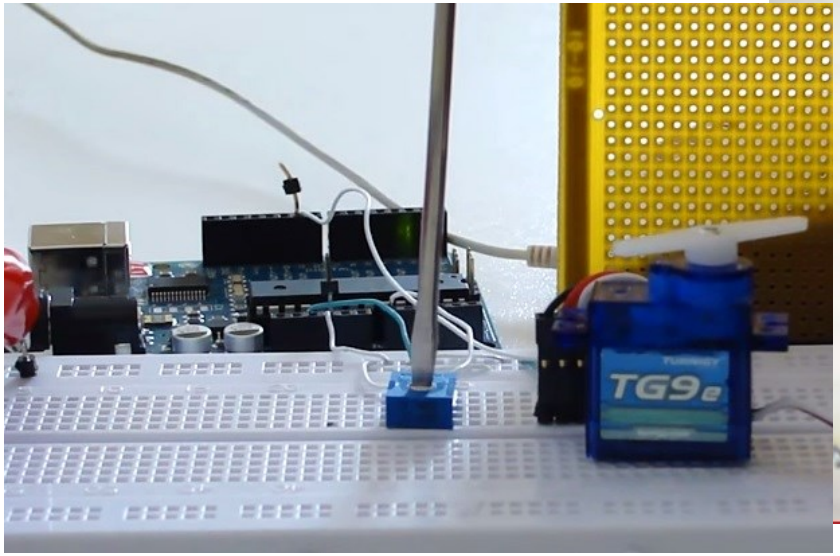
*mS = Millisekunde

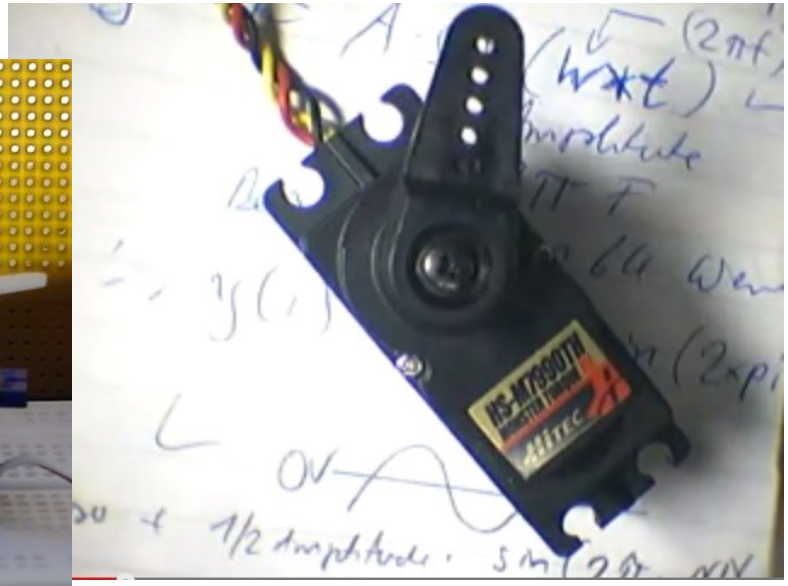
Vorhaben: Wir wollen einen Servo wie ein Pendel als Sinus schwingen lassen

Youtube Video des Beispiels

Sinus Servo
https://www.youtube.com/watch?v=wDE_EwiImvc

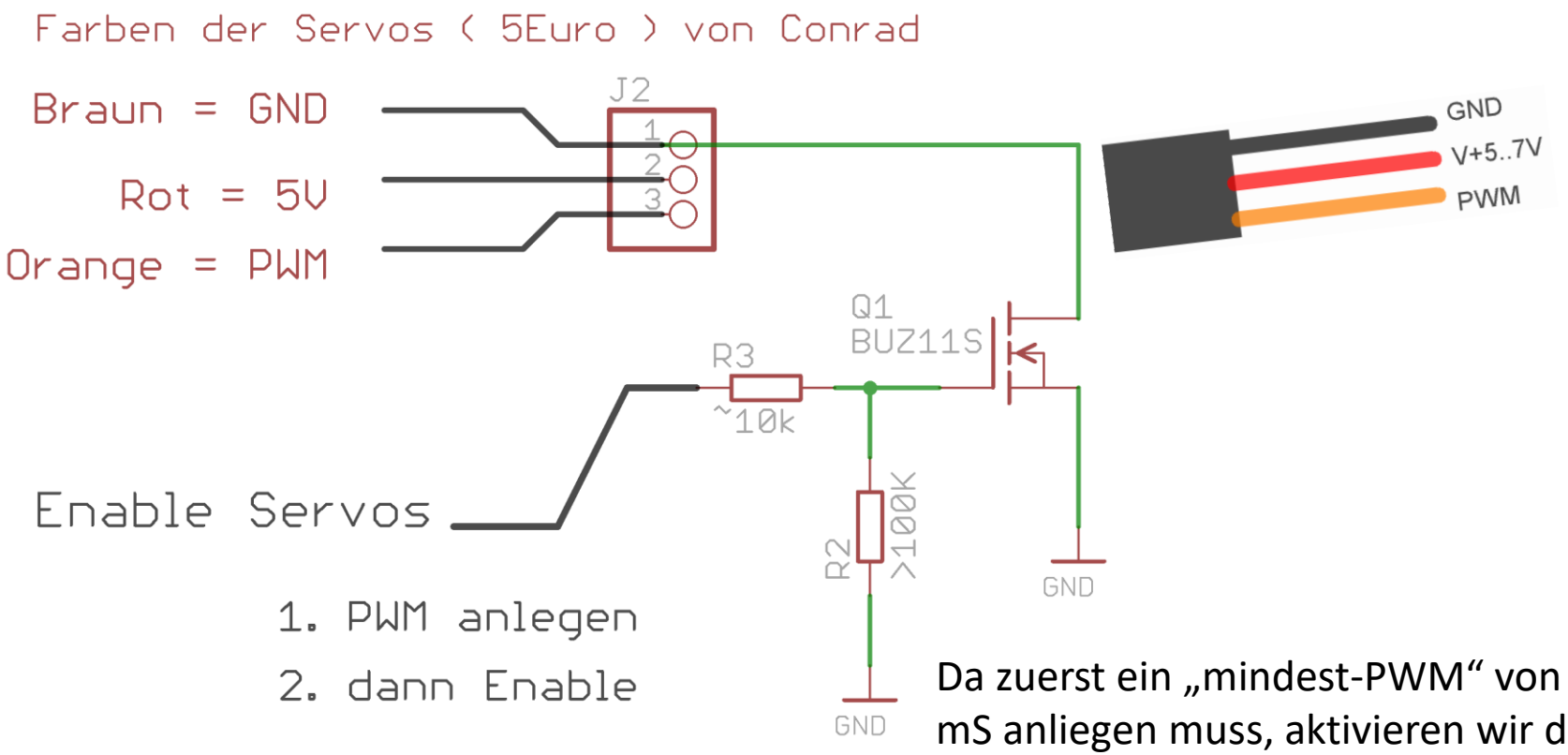
Sinus LED
http://youtu.be/zxT49ZjLUkE





Anwendung:CellStrech: https://www.youtube.com/watch?v=naDMfTPF6WI

Servo richtig Anschließen und Steuern



1. PWM anlegen
2. dann Enable

Da zuerst ein „mindest-PWM“ von 1 mS anliegen muss, aktivieren wir den Servo **erst nach** anlegen des PWM Signals,….. und dann erst durch „Schalten“ des GND-Anschlußes über einen Transistor nach Masse=Gnd

Reales Beispiel für ein universelles Experimentalboard mit 4fach Servo und Enable

Supply ab 200mA

X2-1 D1 1N4007 Verpolungsschutz

Entweder

X2-2

Wenn ein Netzteil benutzt wird, dann bitte "exakt" auf +5V einstellen.

Servo PWM zwischen 1mS und 2mS
50Hz = 18mS Pause

+5V

X1

...oder USB

USB- USB+

F1 PFRA

C1 C3 C4 C5 C2

J8 J10 J1 J3

PWM PWM PWM PWM

Braun = GND
Rot = <6V
Orange = PWM

Extern USB als +5V Versorgung oder auf Servo Spezifikation

(E)

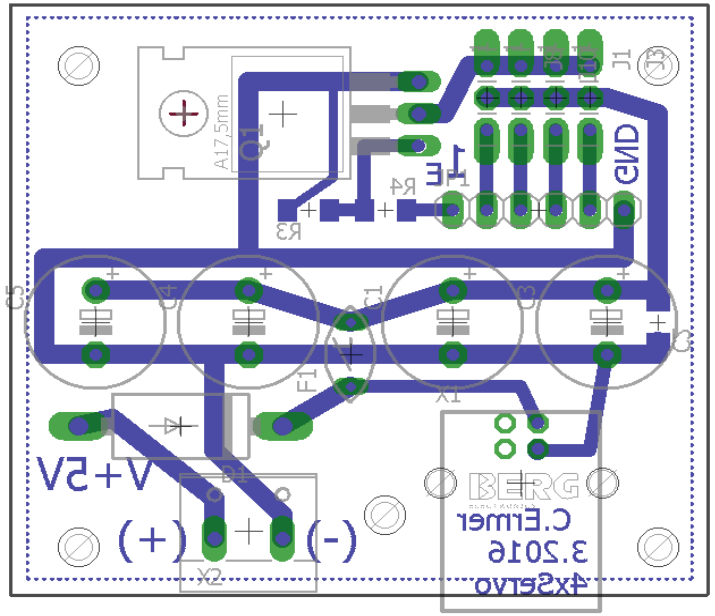
JP1

Enable mit 5V+ = '1'

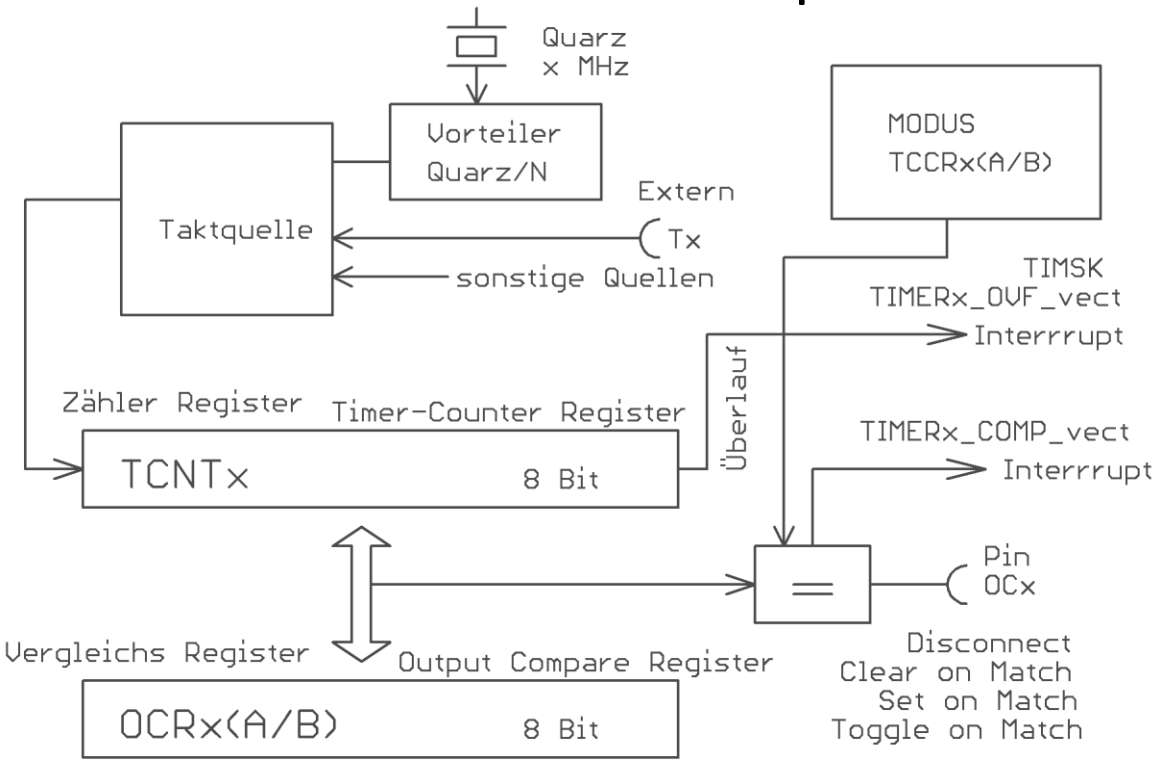
R4 SMD

Q1 BUZ11 bis 15A

R3 SMD



Timer Compare Interrupt



Anstelle des einfachen Timer-Register Überlauf können wir auch das ***Vergleichs/Compare-register*** *nutzen.*
Dazu gibt es viele unterschiedliche Modi, wie die Timer-Unit darauf reagiert.
So kann im Fall TCNTx == OCRx(A/B) unverzüglich ein zugehöriges Portbit gesetzt werden:

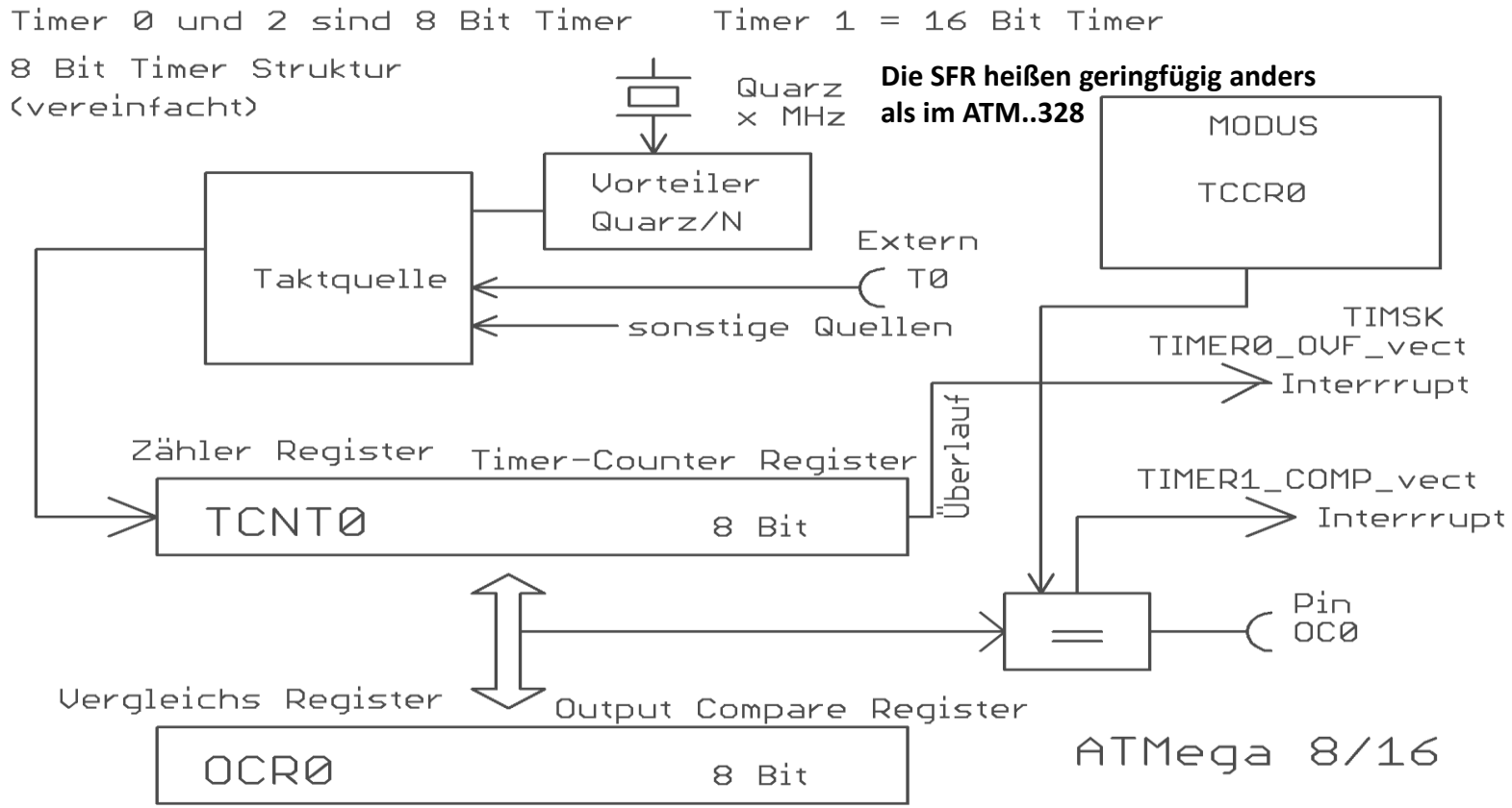
OCRx = Quarz / Vorteiler / Gewünschte

Isoliert kann der **Compare-Match** einen Interrupt auslösen. Es gibt beim ATM328 **2** Compare-Register im 8-Bit Timer. Im Gegensatz zum ATM16/8.
TIMER0_COMPA_vect
TIMER0_COMPB_vect etc..
Dieser Modus wird ebenfalls über SF-Register aktiviert.

8-Bit Timer/CounterX mit PWM

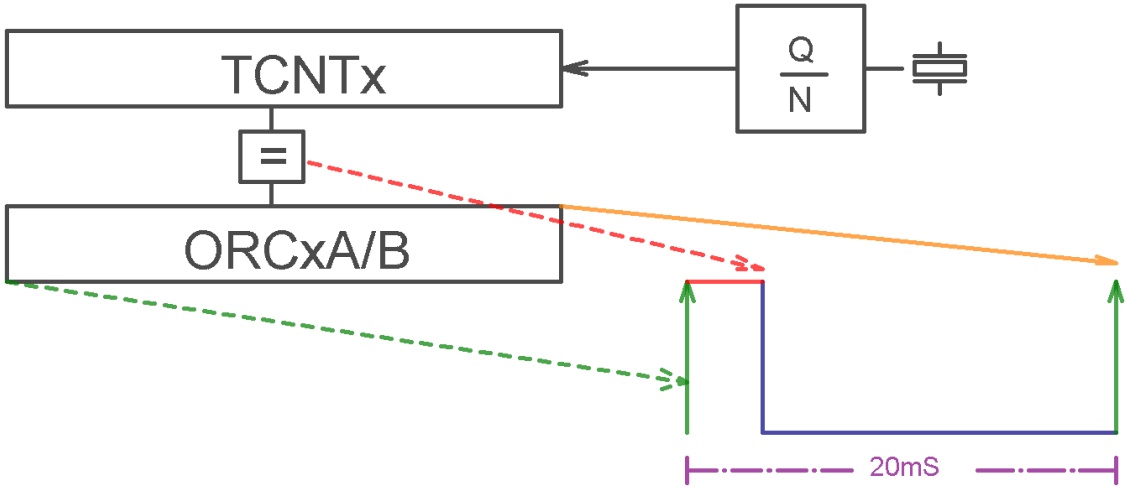
Wir brauchen so gut wie immer einen Taktgeber. Ähnlich wie in der Galeere. Der Taktgeber (**Tick-Counter**) gestaltet bzw. synchronisiert Abläufe nachvollziehbar langsam, z.B. eine 2 sekündliche Textausgabe, so dass Zeit zum Lesen bleibt.

Es ermöglicht Aktivitäten wie eine Blinkfrequenz, die nicht auf hässlichen „Delays“ basiert, was nichts anderes ist als Prozessorzeitverschwendung.



Timer x für Servo-PWM Erzeugung nutzen.

Wir wissen, dass ein SERVO **PWM-ON** Zeiten von **~1mS bis ~2mS** *ohne mechanischen Anschlag* hat. Zudem sollte die PWM so um die **~50HZ** betragen.



Natürlich gibt es dafür genau die richtige Methode bzw. Modi.

Mit dem OCRxA/B Register wird der 1. Zeitpunkt, der Wechsel am PIN **OCx** von **1→0** eingestellt.

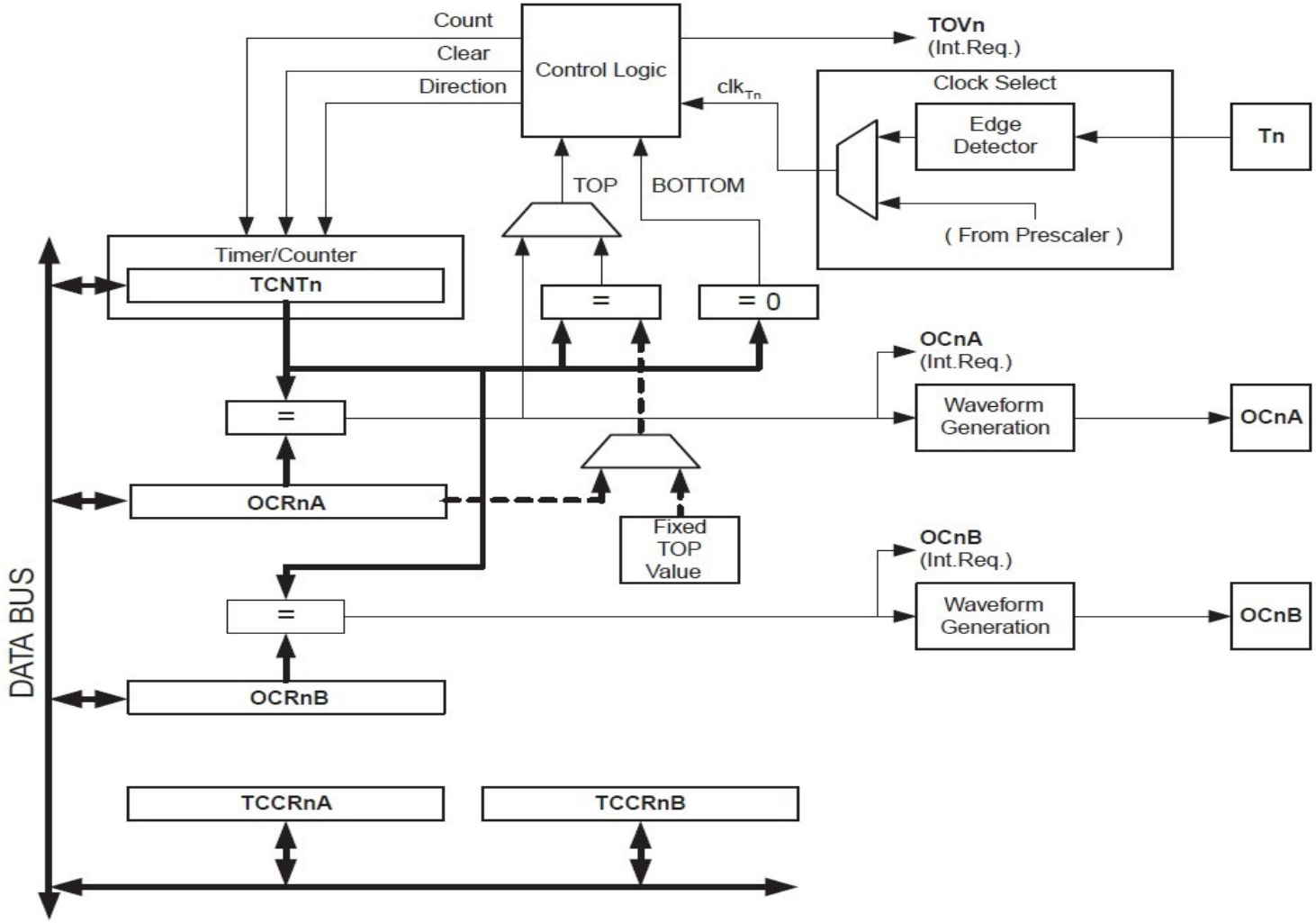
Der **FAST-PWM** Modus sorgt dafür, dass im 2. Zeitpunkt, dem **Overflow-Moment** des Timers, der **OCx** Ausgang wieder von **0→1** wechselt.

16MHZ / 256{Timerdurchlauf} -> 62500/{50Hz} → 1250.
Wir haben einen Vorteiler von 1024 zur Verfügung.
62500 / 1024→61.035 Hz. Das ist genau genug für Servos.
Der **Vorteiler** ist also **1024** → 16.384 mS pro Periode(Zeit) = 1/F
Servomitte = 1.5mS → 16.384/256= **0.064 mS/Bit**
1mS = 1/0.064 ->15.625 → runden: ~OCRx=**16**
1.5mS = 1.5/0.064 →23.4375 c→ runden: OCRx=**23** **(Mitte)**
2mS = 2/0.064 → 31.25 → runden: OCRx=**31**.25
Servo-Range ist also von 16..31 +/- ausprobieren:

OCRx =
Quarz / Vorteiler / gewünschte

Timer 0 Übersicht..

Figure 14-1. 8-bit Timer/Counter Block Diagram



Timer 0 - SFRs für PWM

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match ★
1	1	Set OC0A on Compare Match

Erste Recherchen bringen uns wieder einen Satz SF-Register zu Tage……

→ **TCCR0A = (1 << COM0A1);**

Auch der Modus ist schnell ausgemacht: → Fast PWM
Action: Comparematch → TNCT0=0, Max Value

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM ★	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

→ **TCCR0A = _BV(WGM00) | _BV(WGM01) | _BV(COM0A1);**

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

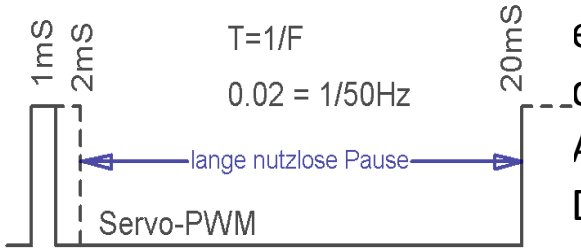
→**TCCR0B = _BV(CS02) | _BV(CS00);** // 1024

→Und jetzt noch: **DDRD |= _BV(PD6);** //OC0A

//16Mhz / 1024 / 256 = ~ 61 Herz

Die PWM Funktion in C

```
// ************************************
void Timer0_PWM( uint8_t u8PwmRange)
// ************************************
{
// 8BIT TIMER T0
// CLK_1024  FAST_PWM OC0_MODE
OCR1A = u8PwmRange; TCCR0A = _BV(WGM00) |
_BV(WGM01) | _BV(COM0A1);
TCCR0B =  _BV(CS02) | _BV(CS00);      // 1024
//16Mhz / 1024 / 256 = ~ 61 Herz
DDRD |= _BV(PD6); //OC0A
};
```




```
/*
Statt ewig (1<< xx ) zu schreiben wurde vorweg dies definiert:
 #define  _BV(x)  (1<<(x))
*/
```

SOFORT nach Einschalten eine **Mindest-PWM** von **> 1mS .. < 2mS** einstellen, sonst läuft der SERVO an den Anschlag.
Das kostet viel Strom und der Servo kann kaputt gehen auf Dauer.

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Ovl. und CompA. Interrupt . aktivieren… TIMSK0 = (1<< OCIE0A) | (1<< TOIE0); //OVL + COMPA

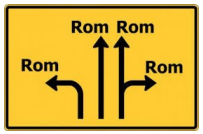
```
ISR( TIMER0_OVF_vect )    //Dieser Vector steht in der „iom328p.h“
{  volatile static uint8_t u8T0_PWM_Cnt=0;
if( !((++u8T0_PWM_Cnt % 5) ) )   // ~244/5=48.8HZ
    { PORTD |= (1<< PD6);            // OC0A=1, Nur  alle 5 MAL
    };
};
ISR( TIMER0_COMPA_vect ) //Dieser Vector steht in der „iom328p.h“
{ PORTD &= ~(1<< PD6);    // OC0A };

// jedes PORT-BIT kann eigentlich verwendet werden
void Timer0_PWM_Start( uint8_t u8PwmRange) //Normal Mode
{
OCR0A = u8PwmRange;
DDRD |= (1<< PD6); // OC0A
TCCR0A = 0;                // simple Mode
TCCR0B = (1<< CS02); // clk/256 = ~244HZ´→ 16^6/256/256=~244
TIMSK0 = (1<< OCIE0A) | (1<< TOIE0); //OVL  + COMPA
};
```

Jetzt braucht nur **OCR0A** verändert werden:
Tb=Zeit/Bit = 1/ (16^6 /256)

1mS= 0.001 / Tb= 62.5
1.5mS=0.0015 / Tb=92.75
2mS = 0.002 / Tb =125
OCR0A = 93; ist also Servo-Mitte

MERKE: TIMER wird nur im SIMPLE MODE betrieben.
Jedes beliebige Bit könnte genutzt werden.

Eine andere Technik.. Tick Interrupt nutzen.
Multi-PWM möglich, zudem völlig freie Portnutzungen möglich.

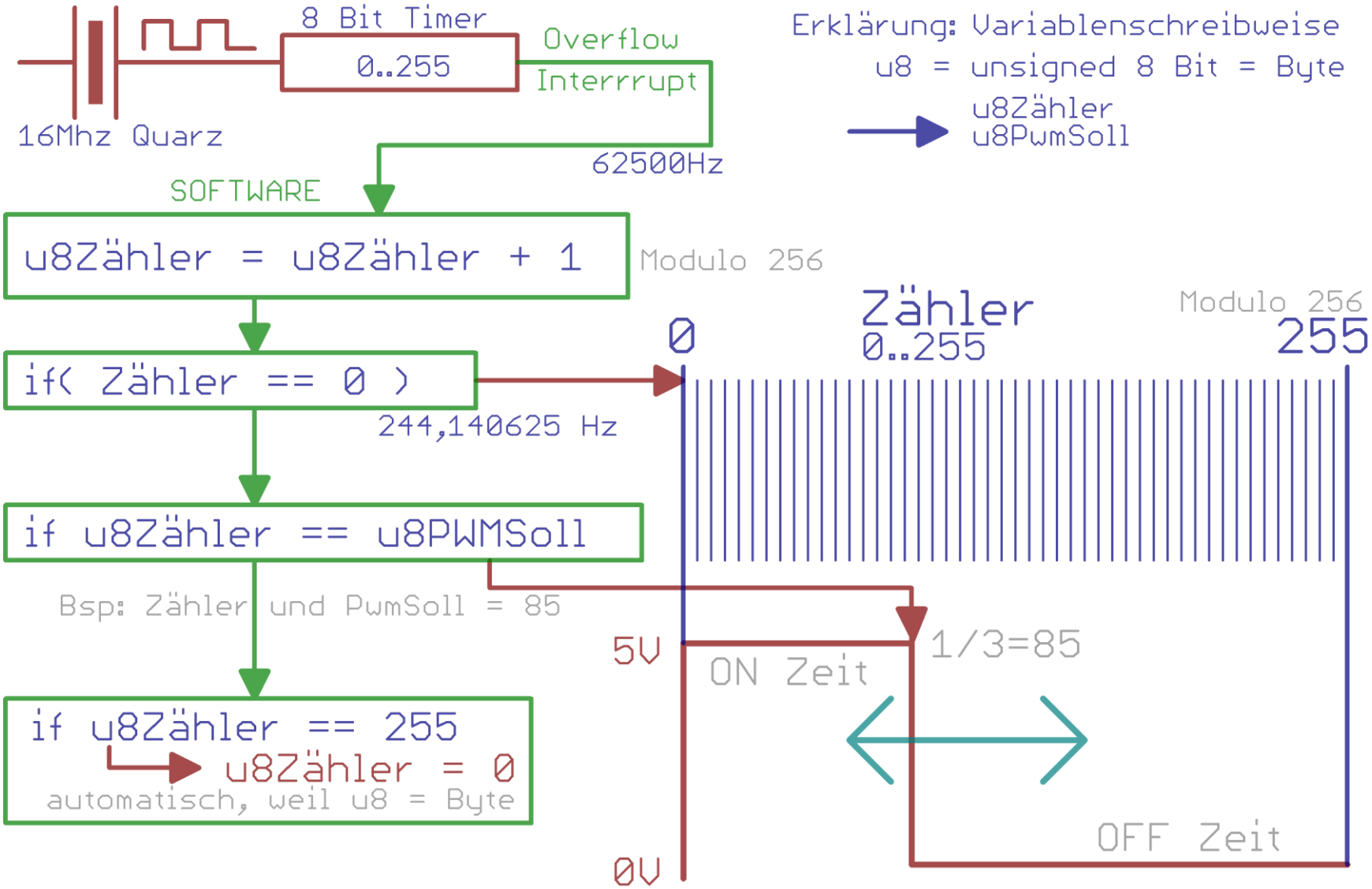



```
unsigned char gu8PWM_D11_Val = PWM_MITTE;          // Compare PWM
// 8MHu Quarz , also Werte anpassen falls 16 MHz
/* uses timer2 (36kHz for IR communication */
/* counts falling and rising edge => 36kHz*2 = 72kHz */

#define PWM_1MS     72
#define PWM_2MS     144
#define PWM_MITTE   108

ISR( TIMER2_COMPA_vect )          //→ 8MHz Quarz
{
gu8Count72kHz++;
//gu8PWM_D11_Val trägt die PWMZeit für Servos

if( (! gu8Count72kHz) && !( gu32_Ticks % 5) ) //281/5 so wird ~55HZ PWM draus.
        {                         //nur alle 5*
        SERVO_D11_HIGH();         //ALLE SERVOS HIGH -  MAKRO PWM PIN +5V
        };

if( gu8Count72kHz == gu8PWM_D11_Val)
        {
        SERVO_D11_LOW();          //MAKRO PWM PIN +0V
        };

If ( !gu8Count72kHz )   // TEILER / 256          //! bedeutet: wenn 0 dann  =
        {
        gu32_Ticks++; //~282      Zählt ~8MHz / 111 = 72000/256=281.5315315
};
```

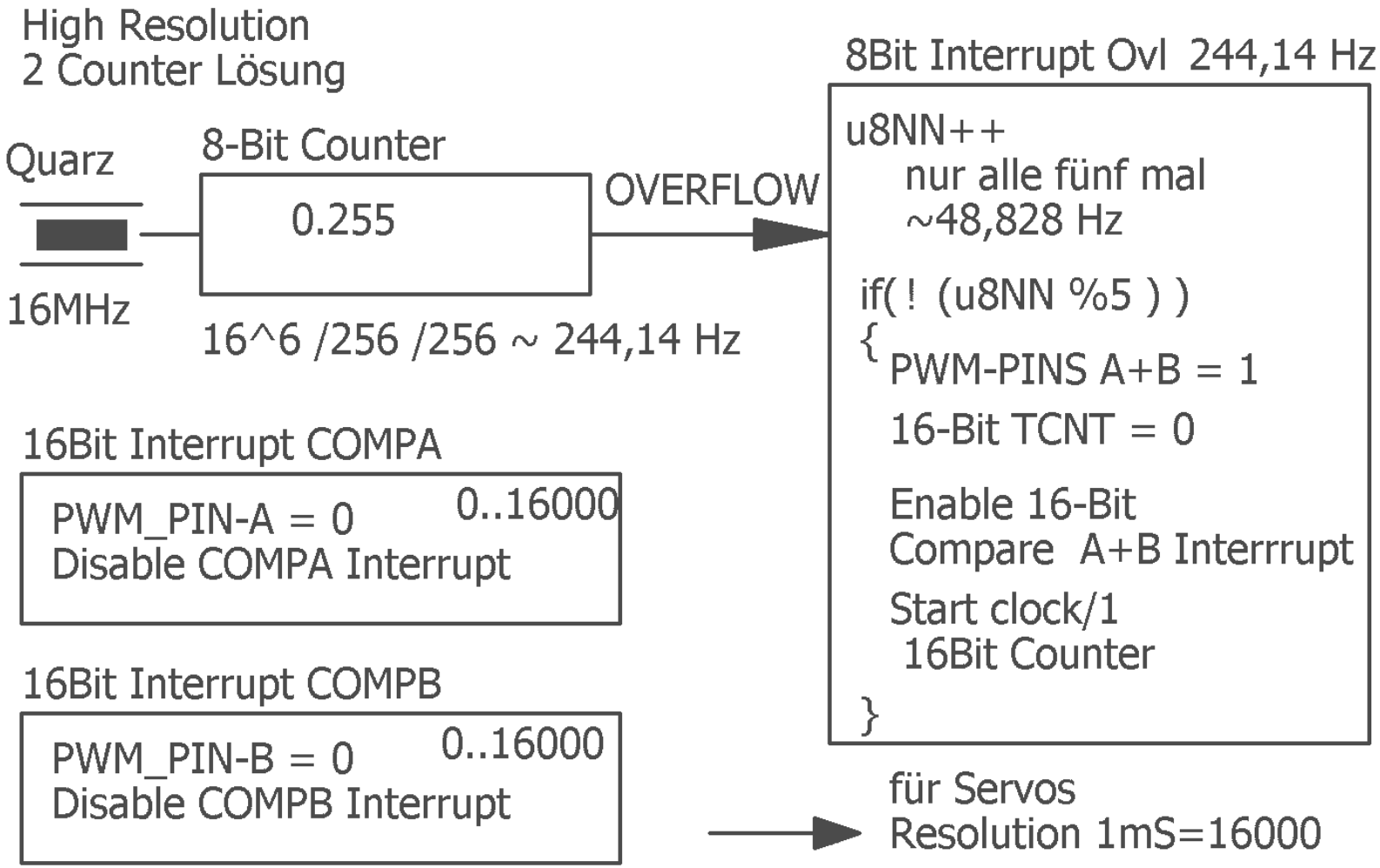
2tes Beispiel. Multi PWM

```
/*
|-------|          |-- PWM
|       |          |
|       |---------|
*/
SIGNAL( TIMER0_OVF_vect )   // 65500 Interrupts/Sekunde
{
                // 16MHhz/65535 = 244.14   overflow/sec     | 1/244.14 = 4.096ms
static uint8_t u8T0_Runs;
static uint8_t u8PWMPause;
// weil die Periode 4.096 MS beträgt, jedoch 20Ms=50Hz erforderlich..
// Trick: u8T0_Runs von Runs
sei();
if( !u8T0_Runs++ )          // also 0
                {
                if( !((++u8PWMPause) % 5) )             // NUR alle 5 Durchläufe 1*ON
                                {
                                MULTI_SERVO_PORT |= MULTI_SERVOS_MASK_1; // ALLES ON
                                };
                };
if( gu8aMultiPWM[0] == u8T0_Runs)  {MULTI_SERVO_PORT &= ~0x01;};
if( gu8aMultiPWM[1] == u8T0_Runs)  {MULTI_SERVO_PORT &= ~0x02;};
if( gu8aMultiPWM[2] == u8T0_Runs)  {MULTI_SERVO_PORT &= ~04};
//   …usw
};
```

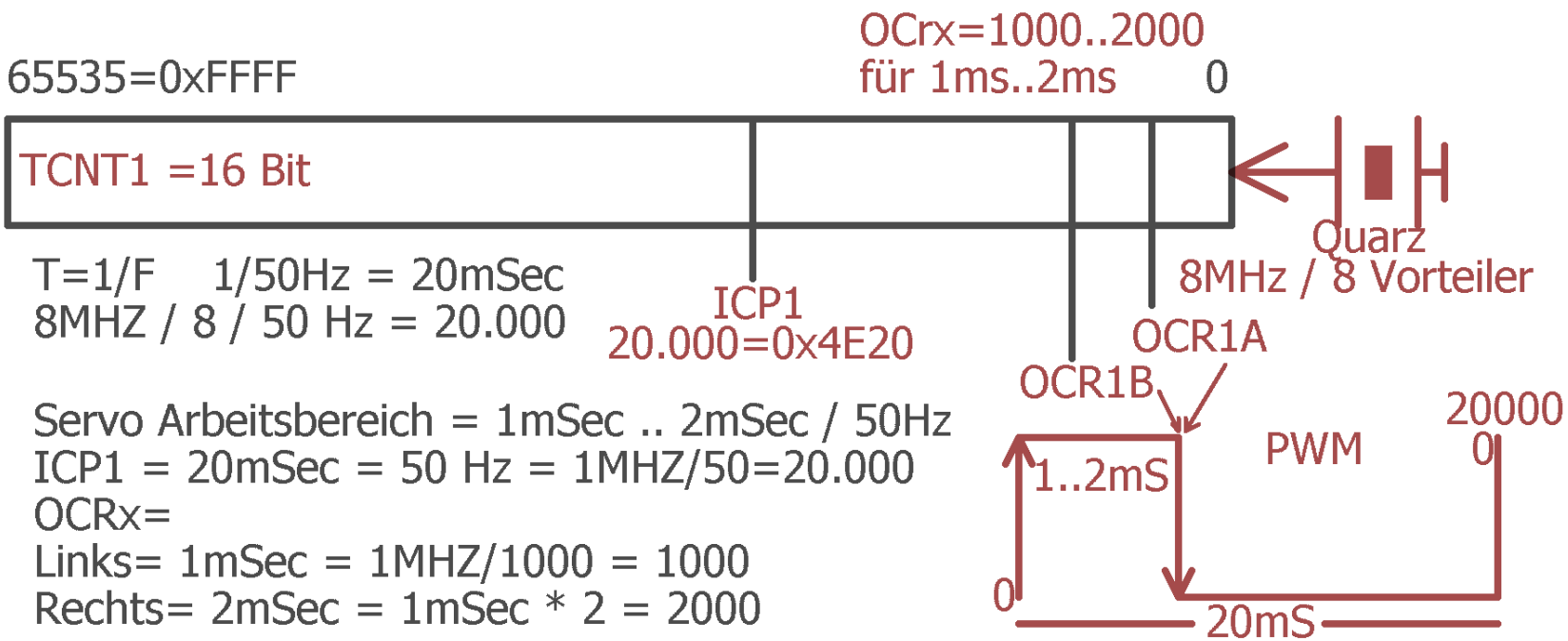
8Bit Software PWM mit Counter



High Resolution Software PWM mit Counter



2 PWM mit dem 16 Bit Timer T1 und seinen 2 Vergleichsregistern

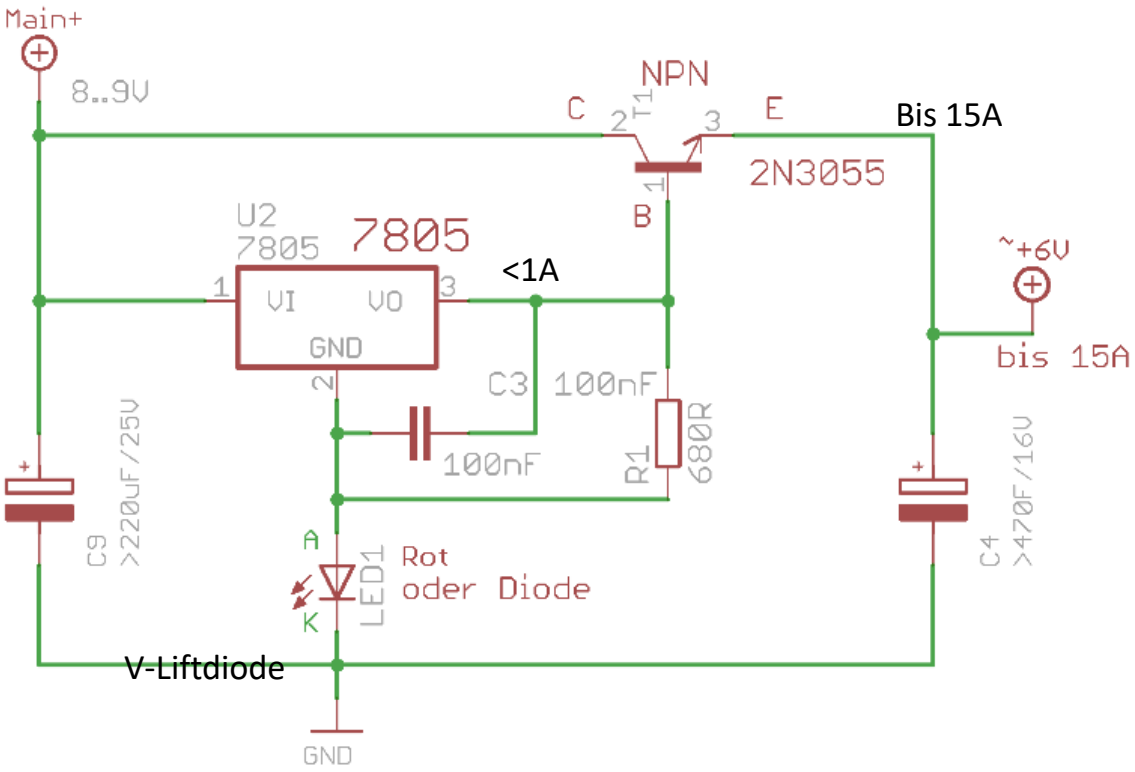


T=1/F 1/50Hz = 20mSec
8MHZ / 8 / 50 Hz = 20.000

Servo Arbeitsbereich = 1mSec .. 2mSec / 50Hz
ICP1 = 20mSec = 50 Hz = 1MHZ/50=20.000
OCRx=
Links= 1mSec = 1MHZ/1000 = 1000
Rechts= 2mSec = 1mSec * 2 = 2000

Trick: Stromaufstockung eines Festspannungsreglers

Manchmal ist es erforderlich erhöhte Strompulsfestigkeiten, die z.B. **Servos** erfordern, bereitzustellen. Gleichzeitig soll die Spannung konstant bleiben.

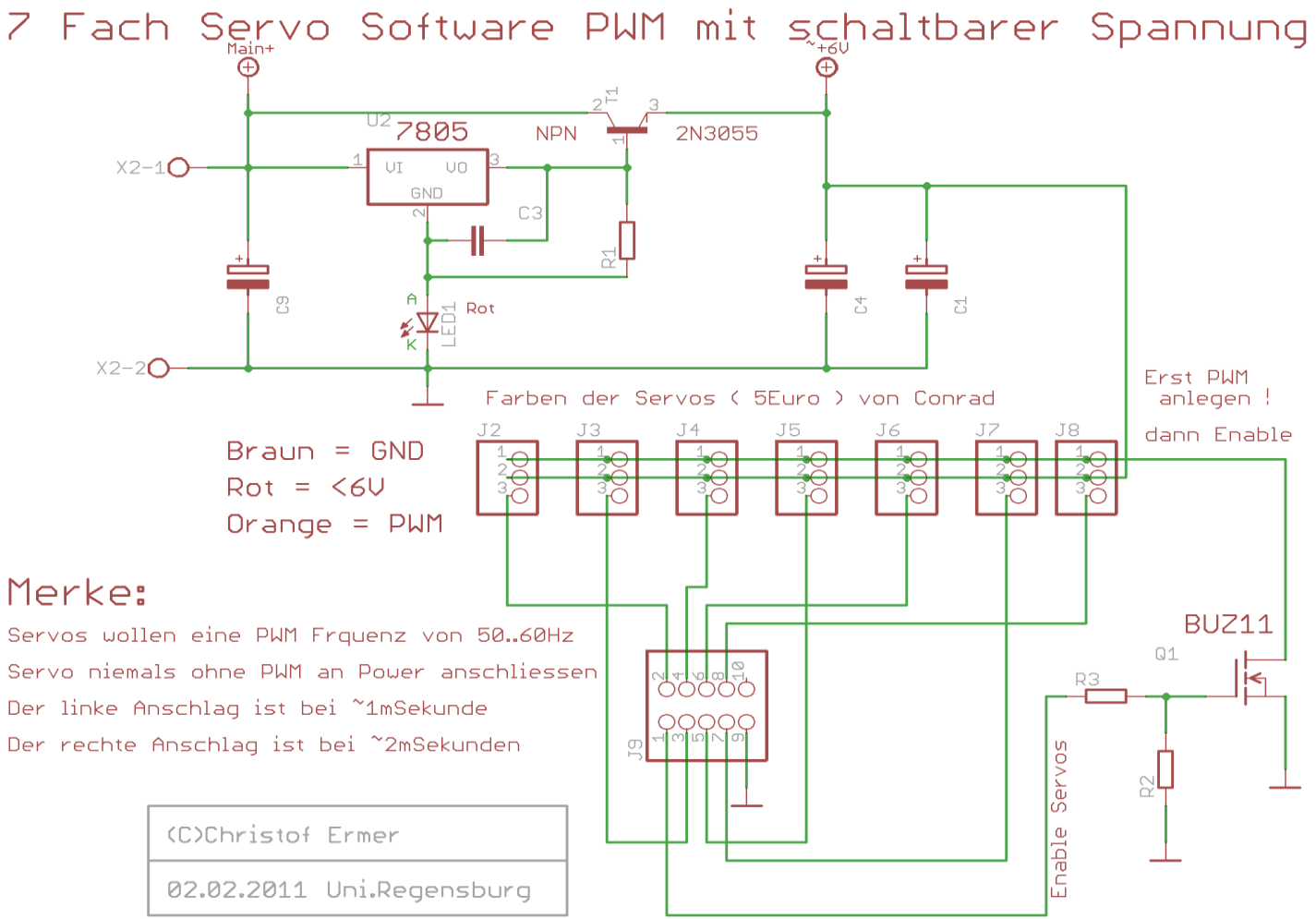


Mit diesem Trick ist dies ohne großen Aufwand realisierbar.
Aber
ACHTUNG:
Keine Kurzschlussfestigkeit!

Der Spannungsverlust am Transistor (~0.6V) wird mit der Spannunglift-Diode am Reglerfuß korrigiert..

Der Regler ist Masseisoliert zu montieren

Zusammenspiel von Hardware und Software
Multi Servos mit Software PWM → 1mS..2mS



Transistor Schaltet auf Masse

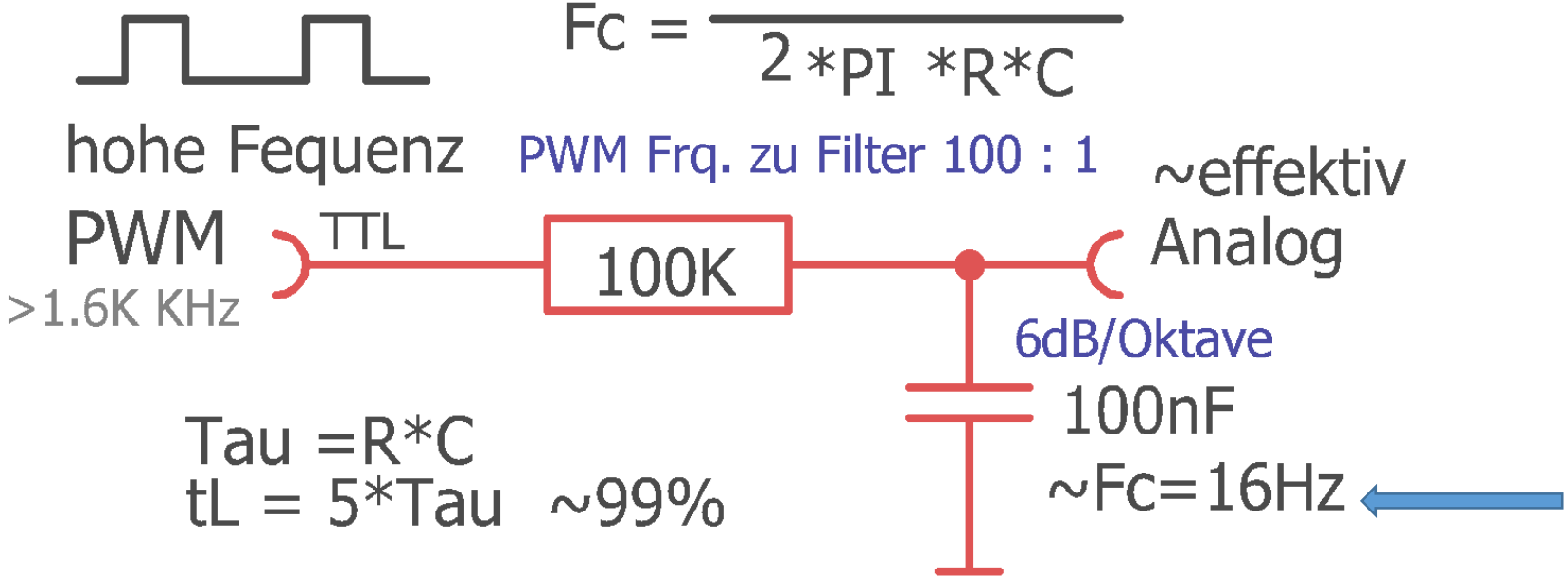
```
// **********************
void XY_ServoT1_Init(void)
// **********************
{
// CTC MODE 14;     50 = 20MS  Range = 1..2 mSec;
//T1 /8 /50Hz = 20000=ICP1 Set ON Range 1000..2000

TCNT1= 0;
OCR1A = 1500 //T1_50HZPWM_CENTER;
OCR1B = 1500;        //OCRx = 1000…2000.  Mitte=1500
ICR1 = 20000U;        // 20000 = 1MHz/50 Hz
            //TIFR |= (1<< ICF1 ); TIMSK |= (1<< TICIE1 );  // ICP //Kein Interrupt benötigt
            // Phase correct MODE 14  ->  WGM 13, 12, 11
            //14 1 1 1 0 Fast PWM ICR1 BOTTOM TOP
TCCR1A = (1<< WGM11) | (1<< COM1A1) | (1<< COM1B1);   //Clear by Match
TCCR1B = (1<< WGM13)  | (1<< WGM12) | (1<< CS11);  // ICR Compare  FCPU/8
DDRB |= (1<<PB1) | ( 1<<PB2);  //DDRB Output  nicht vergessen, PORTB= unnötig
};
```

DAC – Digital Analog Converter

RC-Filter zur Glättung eines PWM oder Pulssignals in eine (effektive) Analogspannung.
Vorteil: minimaler Aufwand, lineare = stufenlose Ausgangsspannung.
Nachteil: Eingangsfrequenzabhängig , träge, eine gewisse Restwelligkeit im Ausgang.

20 log(1:100)= -40dB

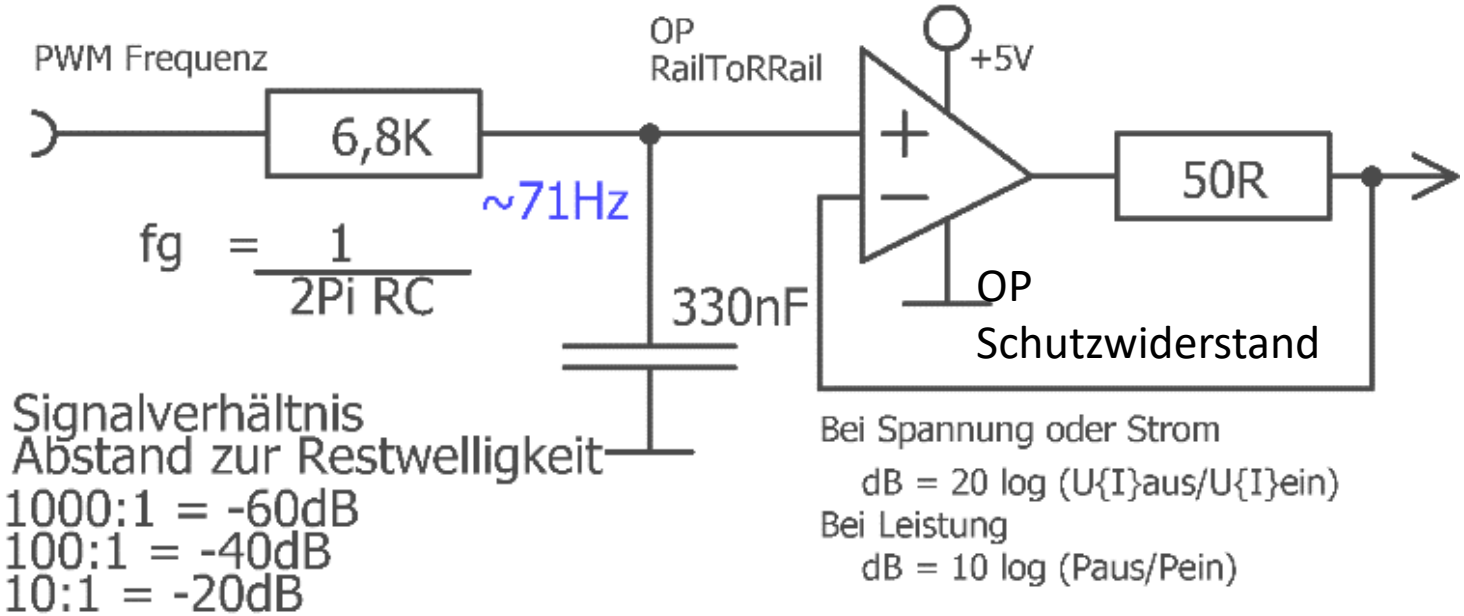
$$Fc = \frac{1}{2 * PI * R * C}$$



Als Grenzfrequenz fg wird diejenige Frequenz bezeichnet, bei der der ohmsche Widerstand (Wirkwiderstand) R genau so groß ist wie der Blindwiderstand XC.

Um den Tiefpass nicht zu belasten, kann man, je nach Eingangsimpedanz des ADC, oder aus anderen Gründen, einen OP als reinen Impedanzwandler nachschalten.

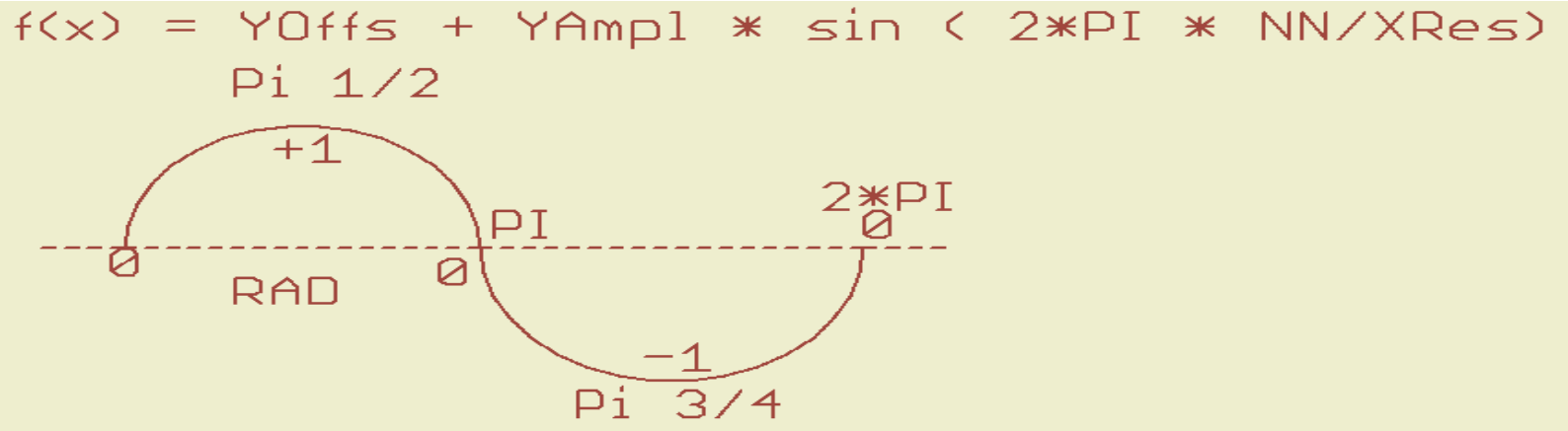
F_grenz = ~71Hz



Max. PWM bei 16 MHZ Quarz = 16Mhz/256 = 62.5 KHz
Fpwm / Fgrenz → 62.5KHz/71 = ~1:880 ~ **-58.9dB** Signalverhältnis.

Die Unterdrückung des AC Anteils an der DAC Spannung >= ~-59dB
Randnotiz: Leistung(dB), 0d + 0dB = 3dB => 10lg(Paus/Pein)

Sinus Formel – Prinzip



Ein Sinus geht von 0 bis 2 Pi

..zwischen Max-Positiv Plus und Min-Negativ auf der Y-Skala

Wir wollen aber ganze positive Zahlen.

Also muss die Kurve um die „halbe Amplitude“ in Y angehoben werden.

Das heißt, **Uss/2** wir als Offset addiert.

yR = resolution Y

xR = resolution X

fx= ½yR + ½yR * sin(2* Pi * {Faktor: 0 bis 1})

…..also mit einstellbarer Auflösung in schritten von 0..1

½yR * **sin(2* Pi * n/xR)** //mit n=0, n++, n<=xR

Sinus – Formel,

Erster Lösungsansatz:

$f(x) = Y_{Offset} + Y_{Resolution}/2 * \sin(2 * Pi * n+1/X_{Resoultion})$;

- Xn+1/XResolution ist ein Multiplikator Quotient der 0..2*Pi mit (0… <1) abdeckt.
- Merke: sinus() liefert Faktoren im Bereich -1 bis +1.

Praktisch sieht es für einen Byte-Wertebereich jetzt etwa so aus.

Das prüfen wir mal bei Gelegenheit…..

- F(x) = 128 + 127* sin(2*M_PI * iNN/256)

für iNN → 0..255

// 127, damit die Werte innerhalb des Bytebereichs bleiben.

So realisiert man z.B. eine Sinuswerte-Array-Tabelle

```
void Calc_SinusTable( uint8_t u16OffSet, uint16_t u16Amplitude, uint16_t  u16XAufloesung )
{
uint16_t u16NN;
//f(x)= OFFSET + Amplitude * sin ( 2* M_PI * NN/X_AUFLOESUNG )
for( u16NN = 0; u16NN < u16XAufloesung; u16NN++)
        {
        gu8aKurve[u16NN] =  round( u16OffSet + ((u16Amplitude/2.0) - 0.5) *
        \           sin(2*M_PI* u16NN/u16XAufloesung) -0.5 );
        };
};
```


```
//Anzeige der Werte bsp. im mainloop…
if( (gu32_Ticks - u32Now) > PULS_500MS_Tick )
                {
                u32Now = gu32_Ticks; //Zeit merken !!
                PORTA = (1 << gu8aKurve[ gu8KurvePos++ ] );
                };
```

Servo-Sinus Software... Mehrere SW-Module sind erforderlich.

<u>1. Tick-Interrupt:</u> Erweitern für schnellere Berechnung

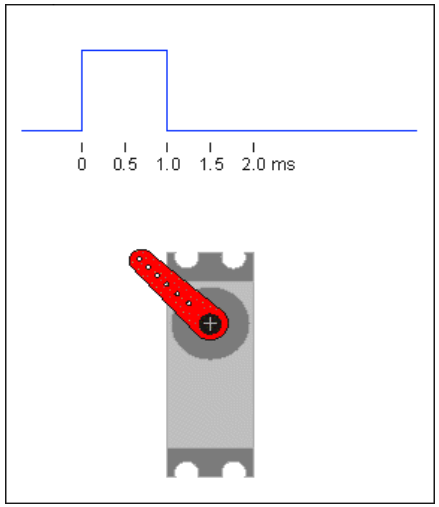
```
 if( ( (gu32_Ticks++) - u32PWM_T) > TICK_100MS )
            {  u32PWM_T = gu32_Ticks;   //neue Zeit merken
             gu8Status |= STATUS_PWM_EVENT;
            };
```

<u>2. Funktion:</u> #define XRES 64.0

```
uint8_t Sinus_Servo(uint8_t u8XResN )
{return  PWM_OFFSET + HALBE_PWM_RESOLUTION + HALBE_PWM_RESOLUTION
* sin( 2* M_PI * u8XResN/XRES );
};
```

<u>3. MAIN-Loop:</u>

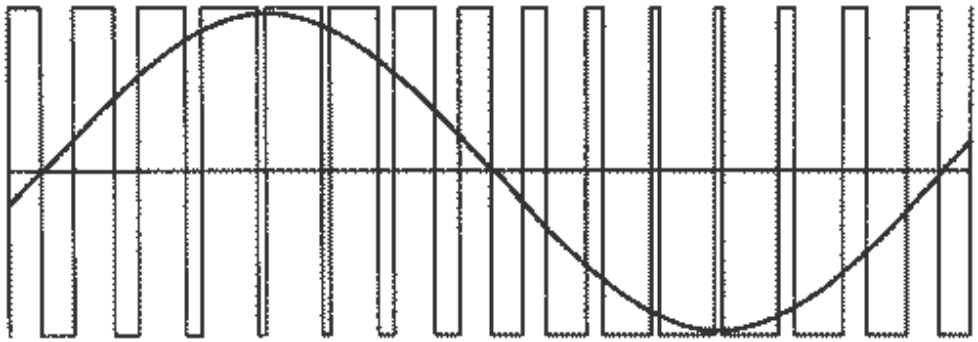
```
if( gu8Status & STATUS_PWM_EVENT)
            {
            gu8Status &= ~STATUS_PWM_EVENT;
                 // Hier die einzelnen X-Schritte berechnen
            OCR0A = Sinus_Servo( u8NN % (uint8_t)XRES );
            u8NN++;
            };
```



PWM mit quasi analoger Sinus Modulation

Um Leistung (Watt) digital zu Regeln, kann man einen PWM mit starrem Zeitfenster quasi analog modulieren.
Je **>** der äquivalente Analogwert, desto **>** die Einschaltzeit, und umgekehrt.

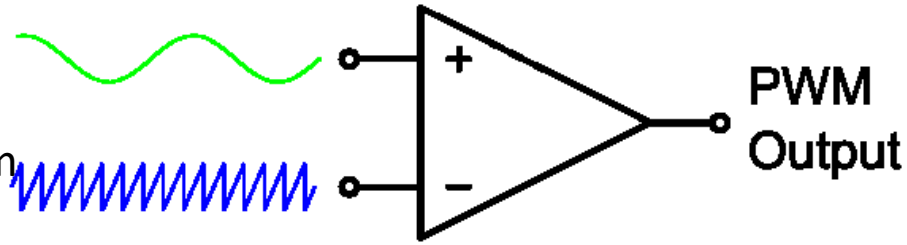
Da der vom PWM gesteuerte Transistor ja immer **nur EIN oder AUS** ist, ist die Verlustleitung an ihm eher gering. Verluste sind eher bei den Schalttransienten.
Dafür gibt es schnelle IGBT Transistoren.
Isolated **G**ate **B**ipolar **T**ransistoren. Es folgt nach dem Schalttransitoren eine LC Tiefpass zur Filterung der PWM .
Deshalb wird diese Technik gerne bei Wechselrichtern angewendet.
Wirkungsgrad >90..99%



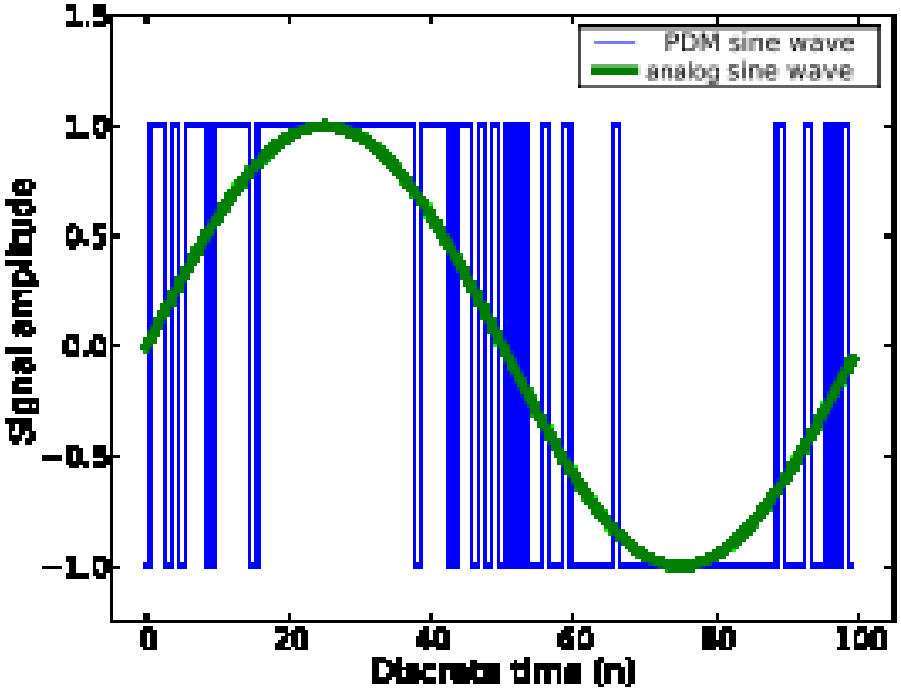
Wie Realisert? Simpel→
OCRx Register Wert = Analog Amplitudenwert.
WICHTIG: Das Verhältnis: Simulierte zur PWM sollte **1:100** sein. Bsp:
50HZ → 5KHZ PWM

Um die „Auflösung“ dieses PWM digitalisierten Analogsignales zu verbessern kann neben der Einschaltzeit-Modulation auch die Zeitbasis in Grenzen parallel dazu moduliert werden. Siehe Bild. Nur wie geht das?

Ein weiteres Konzept ist der Sinus Dreieck Vergleich. Es kommen nach einem Komparator Pulse heraus. Überlege warum und wie!.



Die Dreieck =hoch, relativ zum Sinus



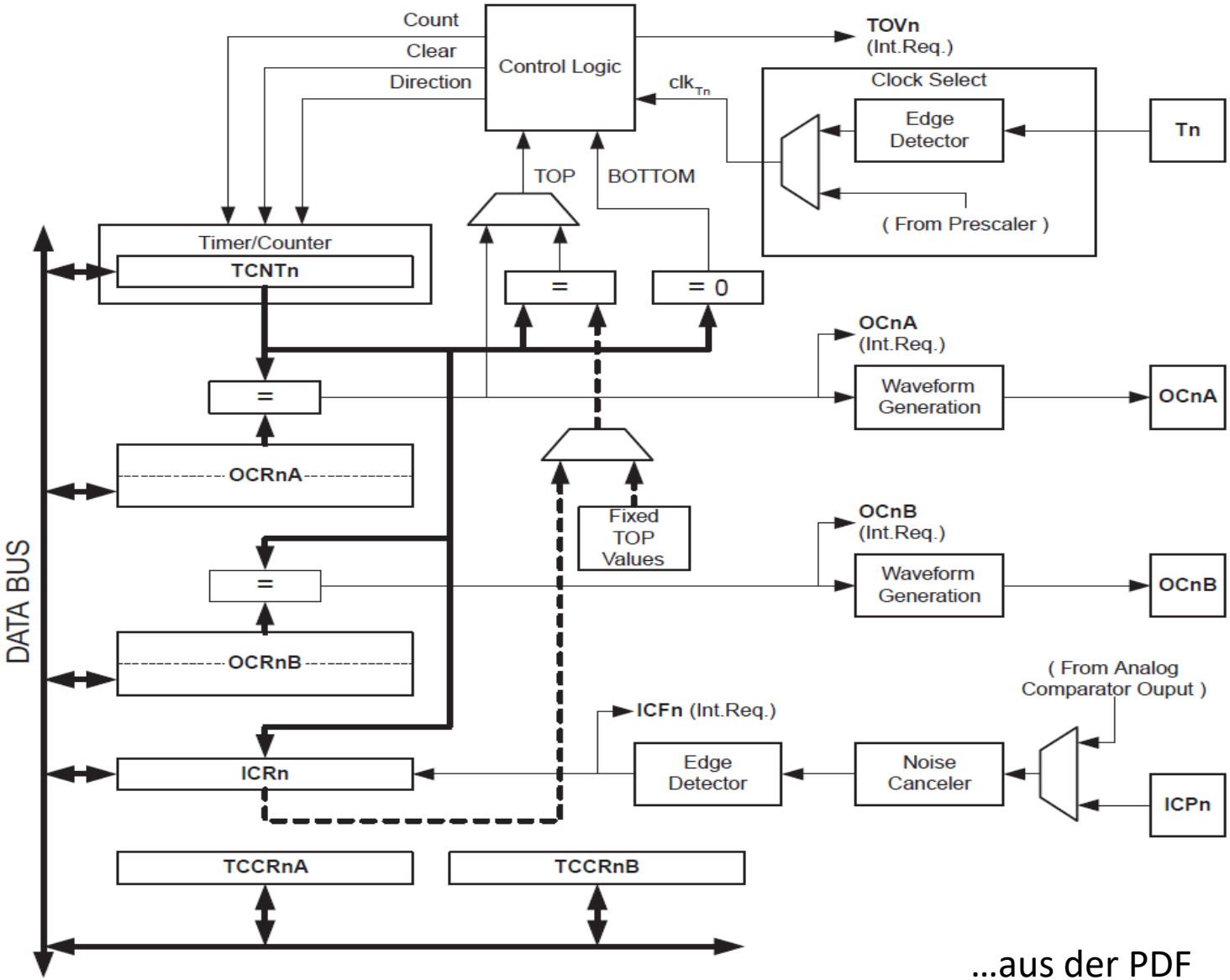
Quellen: Wikipedia.org

Tatsächlich werden so heute sogar klangneutrale AUDIO Leistungsendstufen für PA Anlagen gebaut. Man nennt diese Endstufen dann Class-D Verstärker
Ein Beweis, wie gut dieses Konzept funktioniert.

Timmer als Synthesizer

Mit dem 16 Bit Timer 1

Figure 15-1. 16-bit Timer/Counter Block Diagram[1]



…aus der PDF

wichtige 16 Bit Timer 1 Register

TCCR1A – Timer/Counter1 Control Register A

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	–	–	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-1. Compare Output Mode, non-PWM

COM1A1/COM1B1	COM1A0/COM1B0	Description
0	0	Normal port operation, OC1A/OC1B disconnected.
0	1	Toggle OC1A/OC1B on Compare Match.
1	0	Clear OC1A/OC1B on Compare Match (Set output to low level).
1	1	Set OC1A/OC1B on Compare Match (Set output to high level).

Table 15-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1X at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

TCCR1B – Timer/Counter1 Control Register B

Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 15-5. Clock Select Bit Description

CS12	CS11	CS10	Description
0	0	0	No clock source (Timer/Counter stopped).
0	0	1	$clk_{I/O}/1$ (No prescaling)
0	1	0	$clk_{I/O}/8$ (From prescaler)
0	1	1	$clk_{I/O}/64$ (From prescaler)
1	0	0	$clk_{I/O}/256$ (From prescaler)
1	0	1	$clk_{I/O}/1024$ (From prescaler)
1	1	0	External clock source on T1 pin. Clock on falling edge.
1	1	1	External clock source on T1 pin. Clock on rising edge.

$$\frac{\text{Quarz / Vorteiler}}{\text{OCRxA/B}} = \text{Frequenz}$$

16 Bit Timer 1 Interrupt Register

TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Mit TIMSK1 |= (1<<OCIE1A) wird der Compare Interrupt freigegeben
Dies wird im TIFR1→ OCF1A als Bit angezeigt.
Laut Dokumentation muss man dieses Flag im Register löschen, indem man dieses Bit <u>setzt.</u> „**dies ist merkwürdig**“ → TIFR1 |= OCF1A; //→clear event.

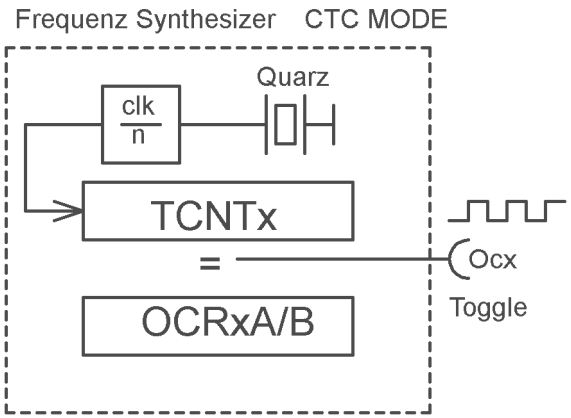
TIFR1 – Timer/Counter1 Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x16 (0x36)	–	–	ICF1	–	–	OCF1B	OCF1A	TOV1	TIFR1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

C-Code – 16 Bit Timer 1 Synthese

//16 Bit Timer 1 erzeugt hörbare Ton- 0.2384HZ bis 15625 Hz.


```
#define LED1_BLINK     (1<<PB5);    //LED an BIT 5
ISR( TIMER1_COMPA_vect  )    // Dieser Vector steht in der „iom328p.h“
{              // Interrupt Variablen müssen "volatile" deklariert sein
PORTB ^= LED1_BLINK;
};

int main(void)
{
DDRB |= LED1_BLINK;          //Bit 0 von PORTC als Ausgang
DDRB |= (1<<PB1);                    //OC1A Ausgang OCxx
TCCR1A |= (1<<COM1A0);               // Toggle OC1A
TIMSK1 |= (1<<OCIE1A);               // Interrupt aktivieren
OCR1A =  17;                         //~1000 Hz
TCCR1B |= (1<<WGM12) | (1<<CS12) | (1<<CS10) ;      // CTC + Vorteiler /1024
sei();
for(;;)
        {
        _delay_us(16); //~4sec
        OCR1A++;  // 16 BIT = 65536
        };....  und so weiter
```



16^6/1024=15625

OCR1A = F_CPU / 8.0 / (SOLLFREQUENZ * 2);

$$\frac{\text{Quarz / Vorteiler}}{\text{OCRxA/B}} = \text{Frequenz}$$

Bsp. (16^6/1) / 65536 =214 = niedrigste Frq.

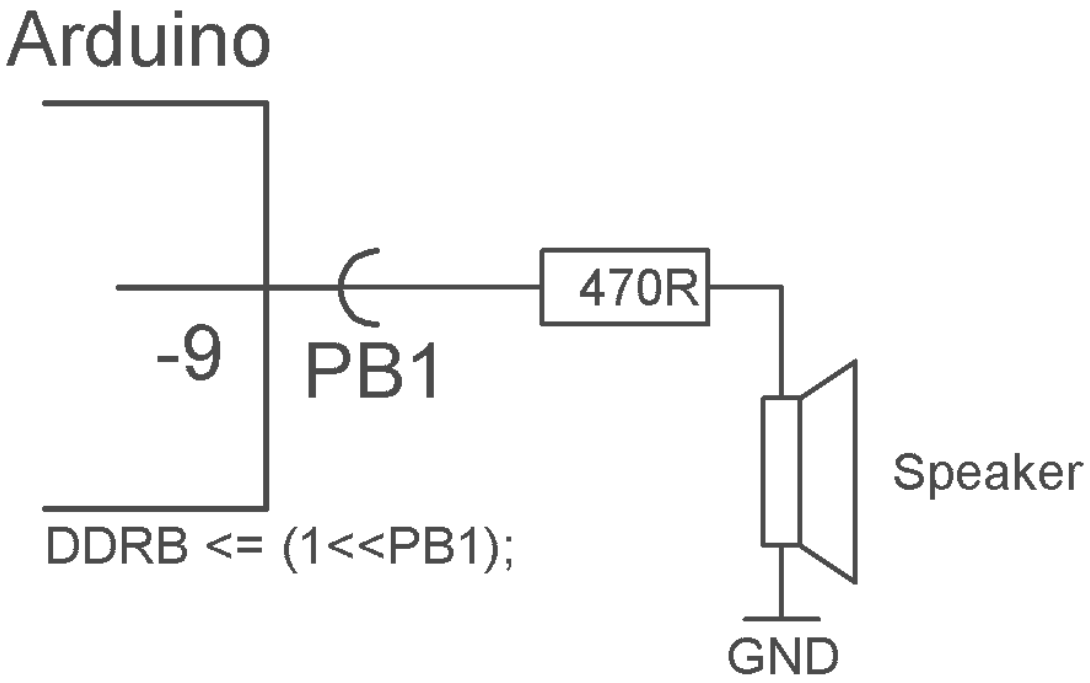
OCRx = Quarz / Vorteiler / gewünschte Frequenz

```
// Musisk Dudel Maschine mit TIMER1 , klingt wie ein Arcade Spiel der 70er

#define SOLLFREQUENZ 800U

DDRB |= (1<< PORTB1 );        // OUPUT OC1A
TCCR1A  =  _BV(COM1A0);
OCR1A = round( F_CPU / 8.0 / (SOLLFREQUENZ * 2)) ;  //FLOAT Rechnung
TCCR1B  = _BV( WGM12) | (1<< CS11 ) ; // /8  CTC
            //TCCR1B  = _BV( WGM12) | (1<< CS10 ) ; // /1 CTC

while(1)
            {
            OCR1A = 10 + random() % 1000;
            _delay_ms(100 );
            };
```

OCRx = Quarz / Vorteiler / gewünschte

ANMEKRUNG: Es gibt welche, die eine Float Rechung innerhalb eines Interrupts als No-Go sehen.
Ja.. Ist zum Teil berechtigt der Einwand.
Wenn man das aber weiß und berücksichtig, kann man damit leben…..
Es ist ja nur ein Beispiel.

Aufbau mit Beeper oder besser Speaker



…grobes *Überschlagen* des Vorwiderstands =
U / ~Max-Strom
5/~10mA = ~500R → 330R…470..560R

Es muss ebenfalls das Port als Ausgang aktiviert werden.
DDRB |= (1<<PB1);

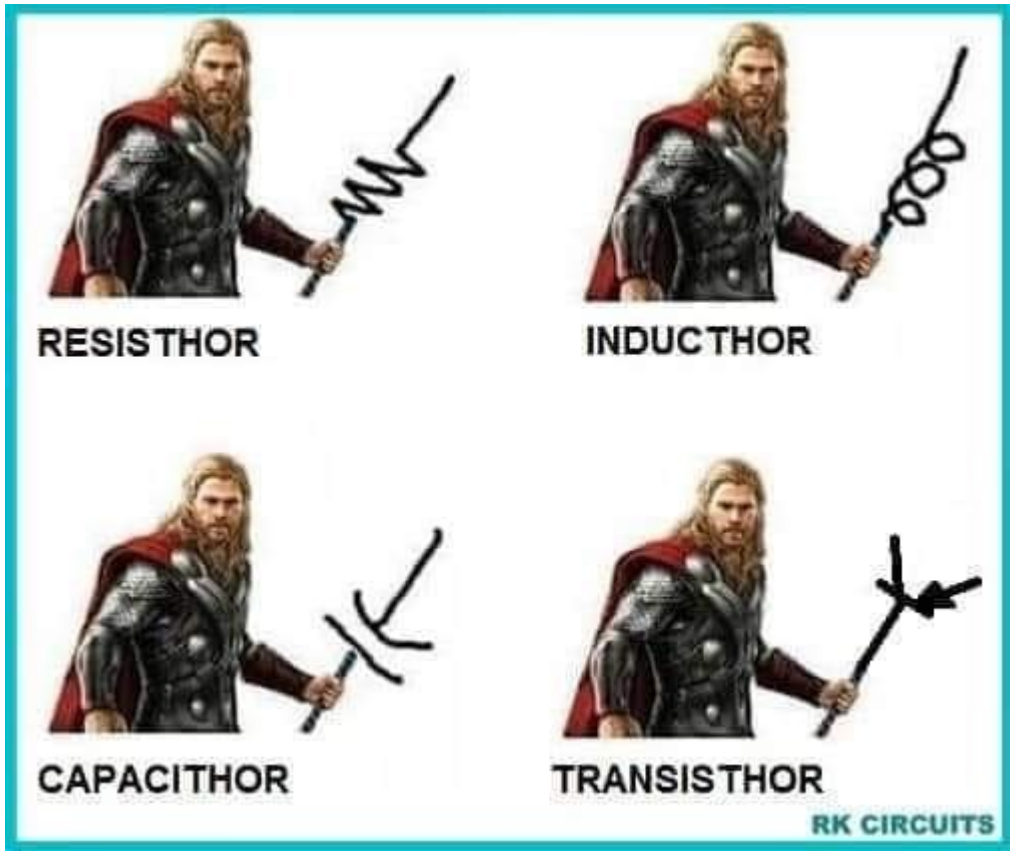
Entweder mit festen OCR1A Werten experimentieren oder auf die Quantisierung der Töne beim Hochzählen von OCR12A achten.
Je kleiner die Werte in OCR1A werden um so gröber wird die Auflösung.

```
void SelectBestPreScaler_T1( float fSollFrq )
{  fSollFrq*= 2;   // Toogle = / 2
if( (F_CPU / 1.0 / fSollFrq ) > 0xFFFF)  {
        if( (F_CPU / 8.0 / fSollFrq ) > 0xFFFF)  {
                if( (F_CPU / 64.0 / fSollFrq ) > 0xFFFF)
                        {
                        if( (F_CPU / 256.0 / fSollFrq )  > 0xFFFF)  {
                                TCCR1B = (1<< WGM12) | (1<<CS12) | (1<< CS10); //CTC 1024
                                OCR1A =  round (F_CPU / 1024.0 / fSollFrq);
                                }
                        else {
                                TCCR1B = (1<< WGM12) | (1<<CS12); //CTC 256
                                OCR1A =  round (F_CPU / 256.0 / fSollFrq);
                                };
                        }
                else {
                        TCCR1B = (1<< WGM12) | (1<<CS11) | (1<<CS10); //CTC 64
                        OCR1A =  round (F_CPU / 64.0 / fSollFrq);
                        };
                }
        else {
                TCCR1B = (1<< WGM12) | (1<<CS11); //CTC 8
                OCR1A =  round (F_CPU / 8 / fSollFrq);
                };
        }
else {
        TCCR1B = (1<< WGM12) | (1<<CS10); //CTC 1
        OCR1A =  round (F_CPU / fSollFrq); // /1
        };
};
```

Automatische Vorteilerberechnung für gewünschte Frequenz

ANMEKRUNG: Es gibt welche, die eine Float Rechung innerhalb eines Interrupts als No-Go sehen.
Ja.. Ist zum Teil berechtigt der Einwand.
Wenn man das aber weis und berücksichtig, kann man damit leben…..
Es ist ja nur ein Beispiel.

Noch a bisserl Elektronik



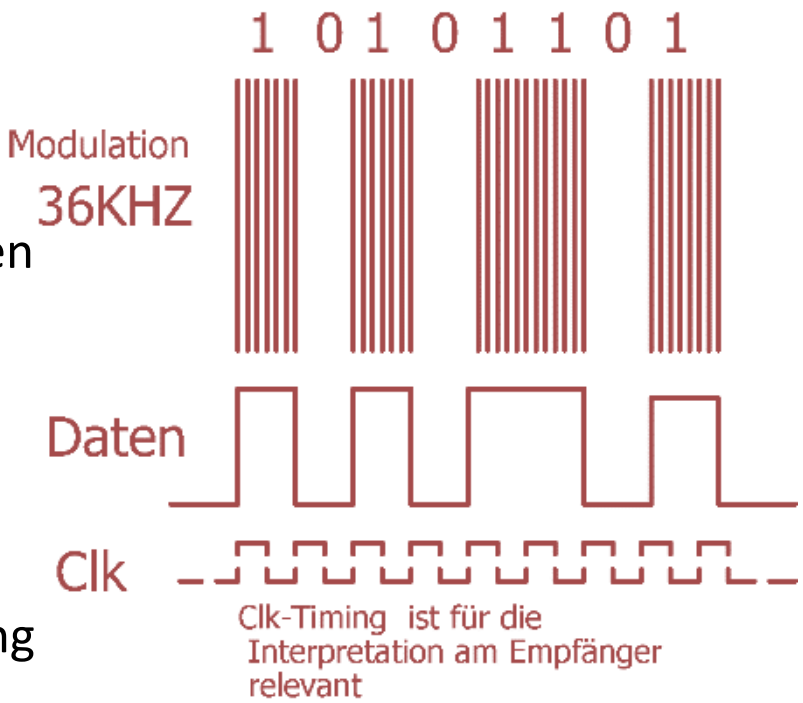
Infrarot Datensender mit Timer als Generator

Prinzip: Ein Timer soll ein 36KHz Signal erzeugen. Dazu wird mit 36*2 =72*KHz an einem Compare Ausgang „OCRx“ der Bitzustand ‚getoogelt‘ (umgeschaltet).
2*Toogle ist eben ein Signalzyklus. So werden 36 KHz daraus. Das geschieht mit dem CTC-Mode und dem OCxA Vergleichsregister.
Das Vergleichsregister wird überraschen einfach berechnet mit:

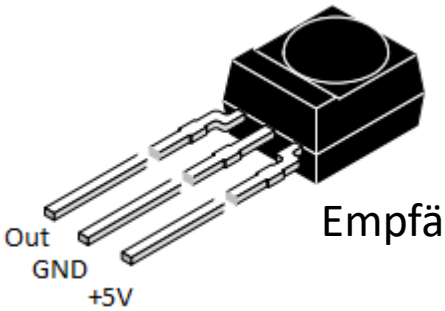
OCRxA = F_CPU / Vorteiler / Sollfrequenz

Also: → (F_CPU = Quarzfrequenz) Prescaling =1

OCR0A = round((F_CPU / 1 / 72000.*0*));



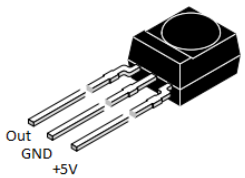
Diese 36 KHz sind eine ‚typische‘ Trägerfrequenz für Infrarot Empfängerbausteine. Diese haben interne Filter für genau diese .

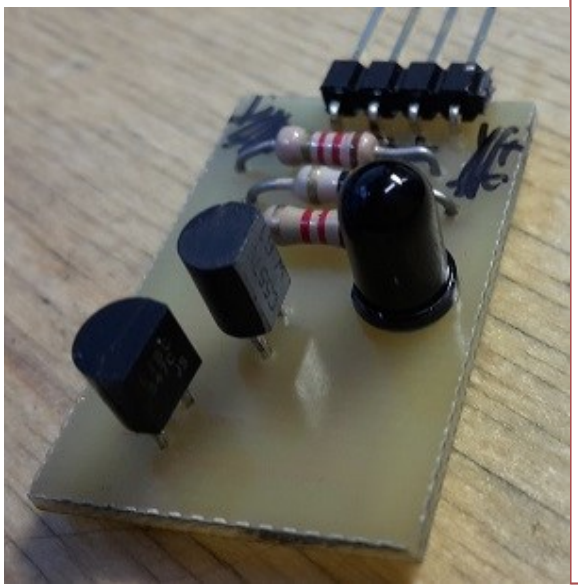


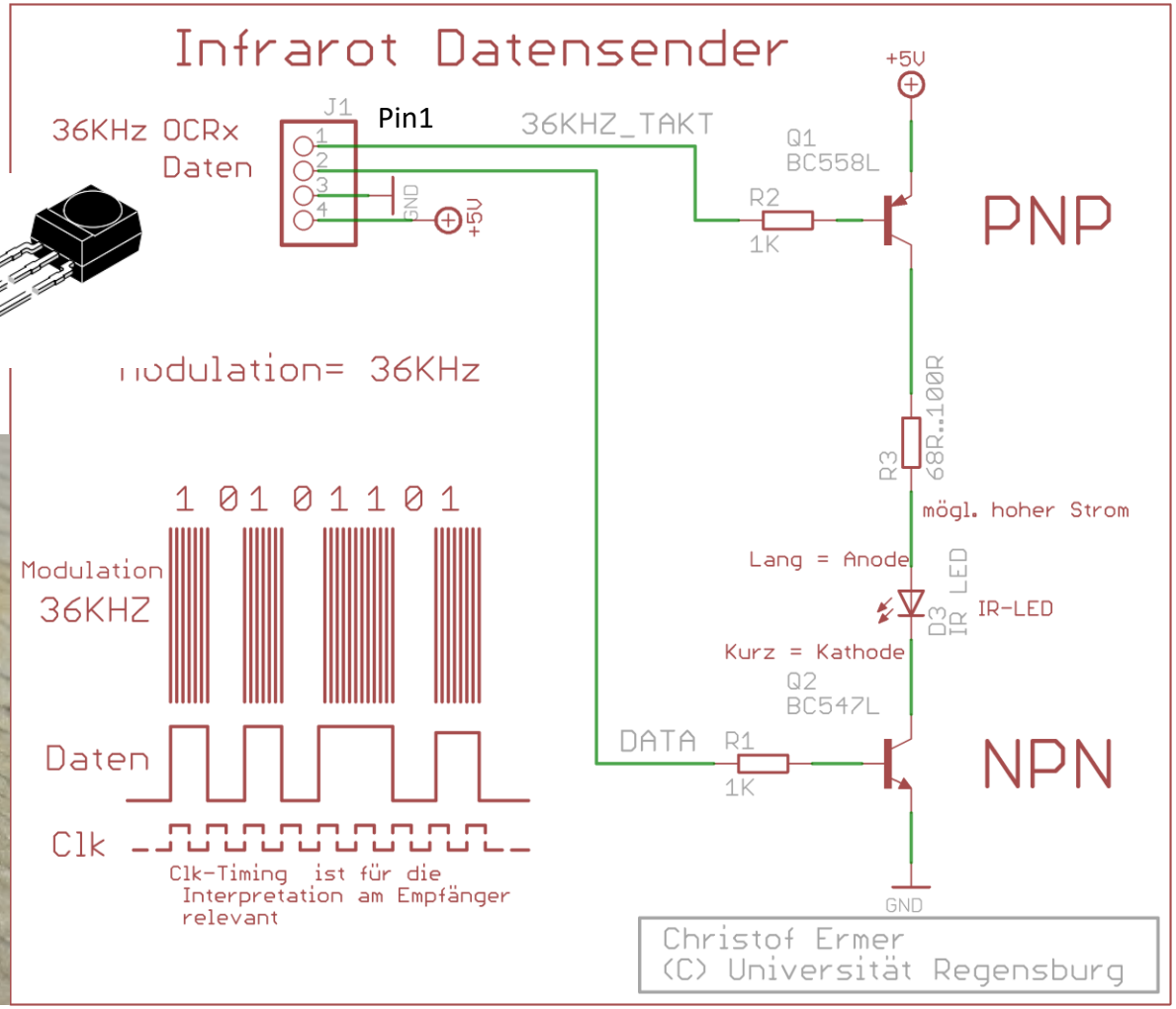
Empfänger.

Um die µController Pins nicht zu belasten schalten wir geschickt Treibertransistoren an die IR-Sendediode. Das ganz ist sehr einfach aufgebaut. Der 36KHz Ausgang des Timers X (8Bit Timer reicht) liegt permanent an. Die Datenleitung schaltet die Trägerfrequenz quasi ein und aus.

Jetzt kann mit einem Empfängerbaustein dieses Signal wieder als Datensignal abgefragt werden.







```
//****************
void Init_36KHz_Timer(void)
//****************
{
DDRD |= (1<< 6); // OC0A TOGGLE BIT
OCR0A = round( ( F_CPU / 1 / 72000.0) ); //No PreScaling /1
TCCR0A = (1<< WGM01) | (1<< COM0A0); // CTC Toogle OC0A
TIMSK0 = (1<< OCIE0A); // Compare interrupt optional  →
TCCR0B =  (1<< CS00);  // /1 RUN
};
```

```
// ***************************
ISR( TIMER0_COMPA_vect )  //72KHZ
//  ***************************
{
if( gu8Status_Morse & TX_TIMMER_ACTIVE )
        {
        if( gu32Quick72KHz_Tick > gu32Tx_Time_Limit )  // Ende erreicht
                {
                gu8Status_Morse &= ~TX_TIMMER_ACTIVE;
                DATA_OFF();
                CLR_MORSE_PIN();
                BEEPER_OFF();
                };
        gu32Quick72KHz_Tick++;
        };
};
```

Mit dem 72KHz Interrupt lassen sich schnelle Jobs erledigen: Siehe:

Weil wir es bald brauchen:

ADCSRA &

Wichtige Softwaretechnik:

Wie kann ich den Zustand EINES Bits in einem Byte oder Word abfragen.?

So lässt sich, z.B. der Zustand eines Bits in **Special Funktion Register** abfragen.

Ganz einfach.. Beispiel Sehen wir mal nachdem „**ADSC**“ Bit im „**ADCSRA**“ Register

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Das **ADSC** Bit kann binär exakt so dargestellt werden → **(1<< ADSC)**

Um zu Fragen, ob das Bit 1 ist oder 0, wird diese Bit mit dem zu untersuchen Byte ver**und**et: in Klammern!

if (ADCSRA **& (1<< ADSC))**

Das Ergebnis der „if“ Abfrage = **FALSE**, wenn das Bit „**0**“ war.
Das Ergebnis der „if“ Abfrage = **TRUE**, wenn das Bit „**1**“ war.

Genau so kann jede beliebige Variable abgefragt werden.

PWM mit deutlich höherer Auflösung mit dem 16Bit Timer „1“
Timer1 im Phase Correct

Ziel: Servo PWM mit 50Hz und 1mS..2mS und höher Auflösung als 10Bit
Dazu brauchen wir das 16Bit ICR1 Register.

```
//Clear OC1A/OC1B on CompareMatch  WGM11 , WGM10
//Fast PWM mode, Set OC1A/OC1B on TOP, WGM13:0 set to fats PWM
#define PWM_TEILER_FUER50HZ        40000U   // für 16MHZ
#define PWM_1000uS_VAL             2000U    //= 1MS
#define PWM_2000uS_VAL             4000U    //= 2MS
ICR1 = PWM_TEILER_FUER50HZ; // Fixed Top Value
OCR1A = PWM_1000uS_VAL;
//PWM PHASE CORRECT VIA ICR1
TCCR1A = _BV( COM1A1) | _BV( COM1B1) | _BV( WGM11 );
//ClockSelect /8  + PWM PHASE CORRECT
TCCR1B =  _BV( WGM13 ) | _BV( WGM12 ) | _BV( CS11 );
DDRB |= (1<<PB1);  // OCR1A out, Pin 15, PB1
```

Berechnung: 16^6/8/40000 = 50 //Hz Value für das ICR1 Regsiter
16^6/8/ 1000mS = 2000 // Value für das OCR1A Register

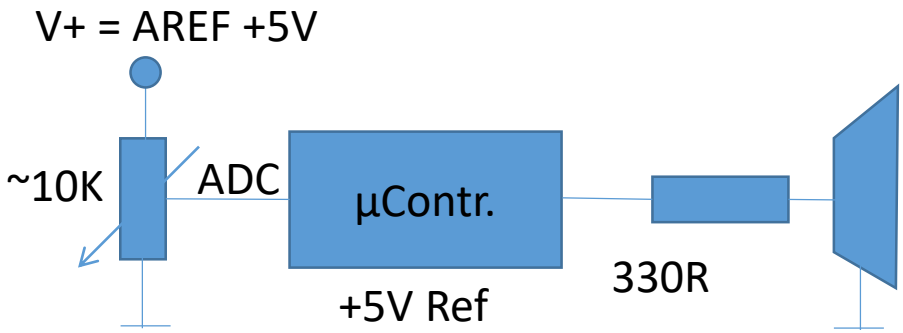
Interner Analog Digital Converter
10 Bit ADC
Auflösung: Reference/1024

Vielfältige Features und Einstellmöglichkeiten..

23.1 Features

- **10-bit Resolution**
- **0.5 LSB Integral Non-linearity**
- **± 2 LSB Absolute Accuracy**
- **13 - 260 µs Conversion Time**
- **Up to 76.9 kSPS (Up to 15 kSPS at Maximum Resolution)**
- **6 Multiplexed Single Ended Input Channels**
- **2 Additional Multiplexed Single Ended Input Channels (TQFP and QFN/MLF Package only)**
- **Temperature Sensor Input Channel**
- **Optional Left Adjustment for ADC Result Readout**
- **0 - VCC ADC Input Voltage Range**
- **Selectable 1.1V ADC Reference Voltage**
- **Free Running or Single Conversion Mode**
- **Interrupt on ADC Conversion Complete**
- **Sleep Mode Noise Canceler**

µController als durchstimmbarer generator



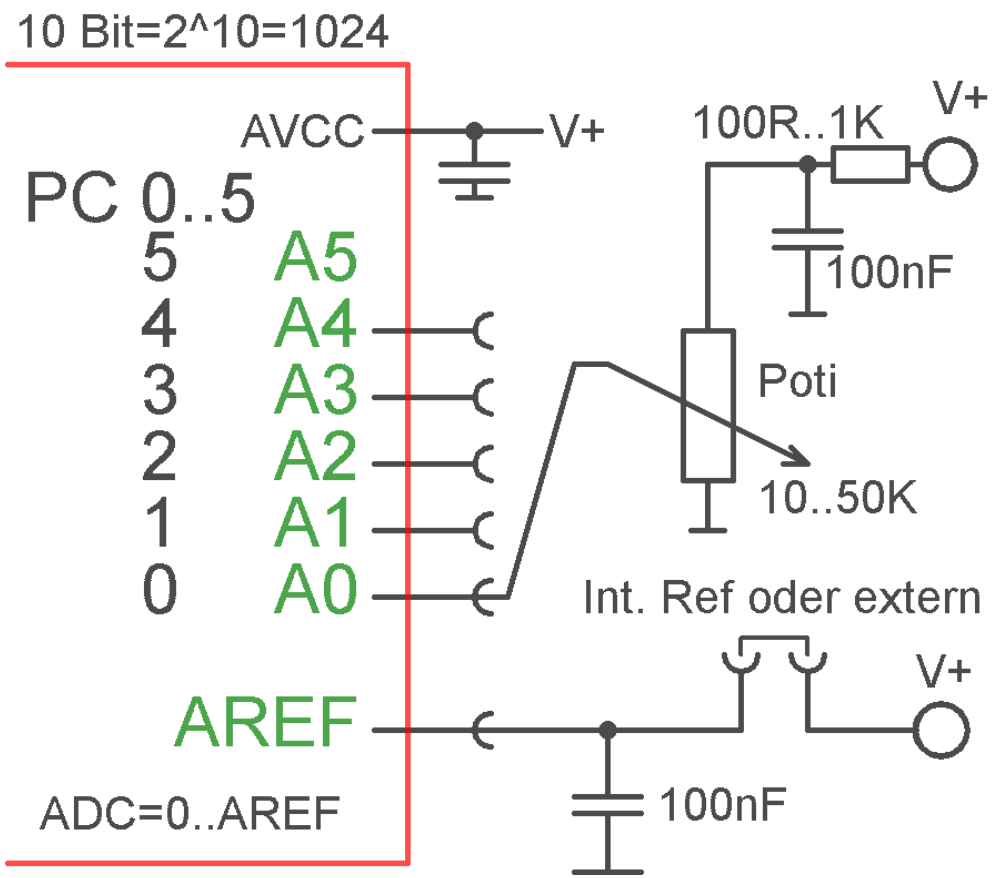
Um nun die des Timers von außen mit einem Poti (Potentiometer) einstellbar zu machen, brauchen wir eine

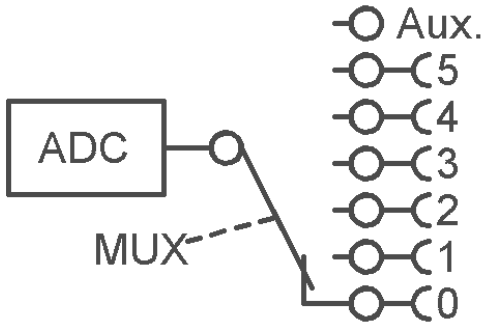
Analog-Digital-Wandlung. ADC genannt.

Damit beschäftigen wir uns jetzt..

10 Bit **A**nalog **D**igital **C**onverter → ADC

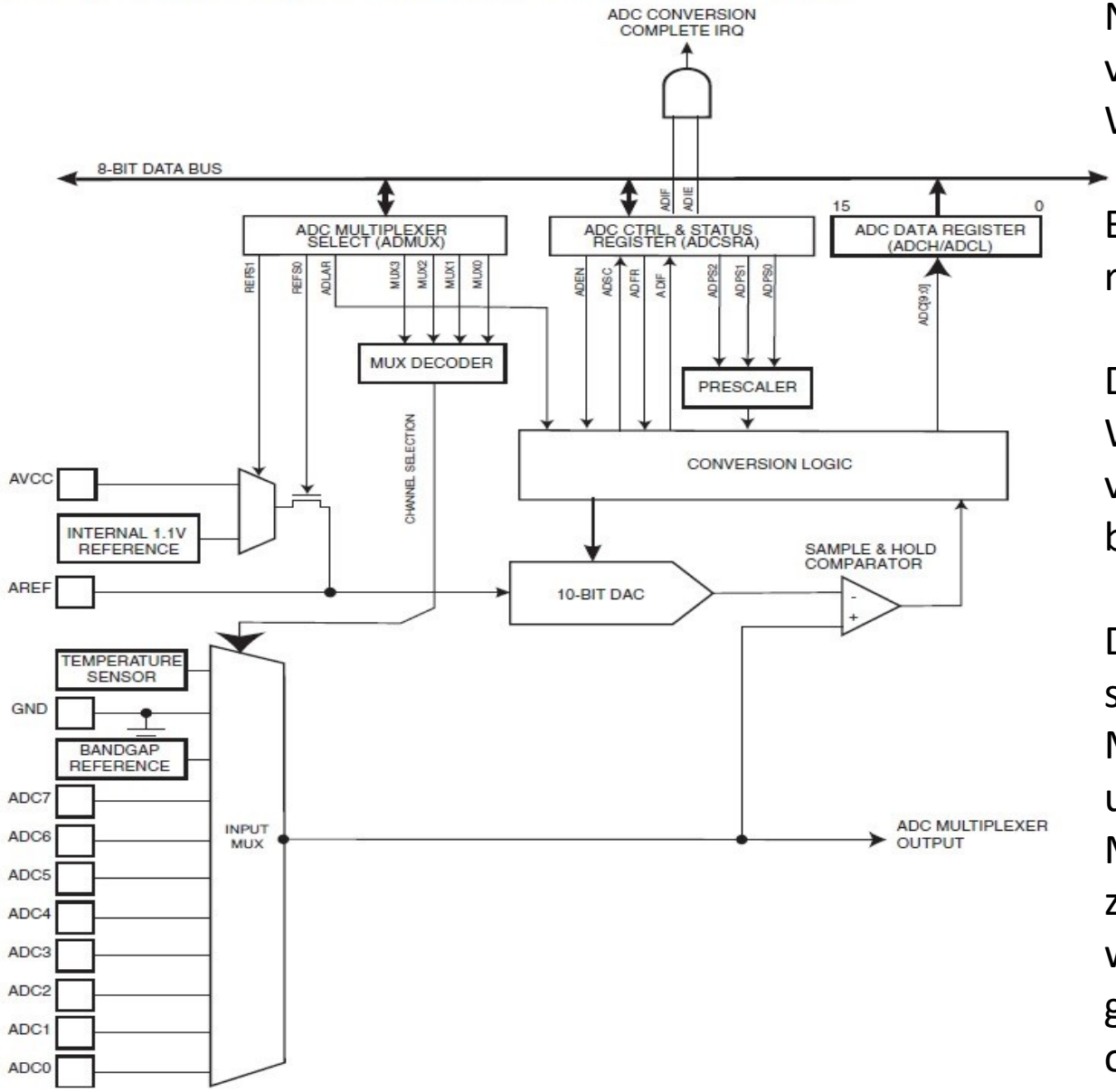
Jetzt ist es an der Zeit wie man „**von außen**“ an dem Wert im Register **OCR1A** „**drehen**“ kann. Dazu ist im Mikrocontroller ein ADC eingebaut. In unserem Fall des **AVR-ATMega328p** ein **10 BIT ADC**. Das heißt, dieser kann Werte von **0..1023** liefern, weil **2^10 = 1024**. Je nach Eingangsspannung und gewählter Referenz.





Wie wir sehen, wird der ‚*eine*‘ interne ADC über ein **MUX**-Select an die Eingänge geschaltet.
Dazu gibt es wieder verschiedene Modi. Auch Differenz Eingänge.
Zudem sollte man dafür sorgen, dass die Spannungen ‚sauber‘ sind.
Deshalb die **Sieb-integratoren** an **AVCC** und **AREF**.
Sonst ‚flattern‘ die letzten Bits.

Figure 23-1. Analog to Digital Converter Block Schematic Operation,



Nicht von der Vielfalt verwirren lassen.
Wir benötigen für den ‚normalen' Betrieb nur 3 relevante Punkte.

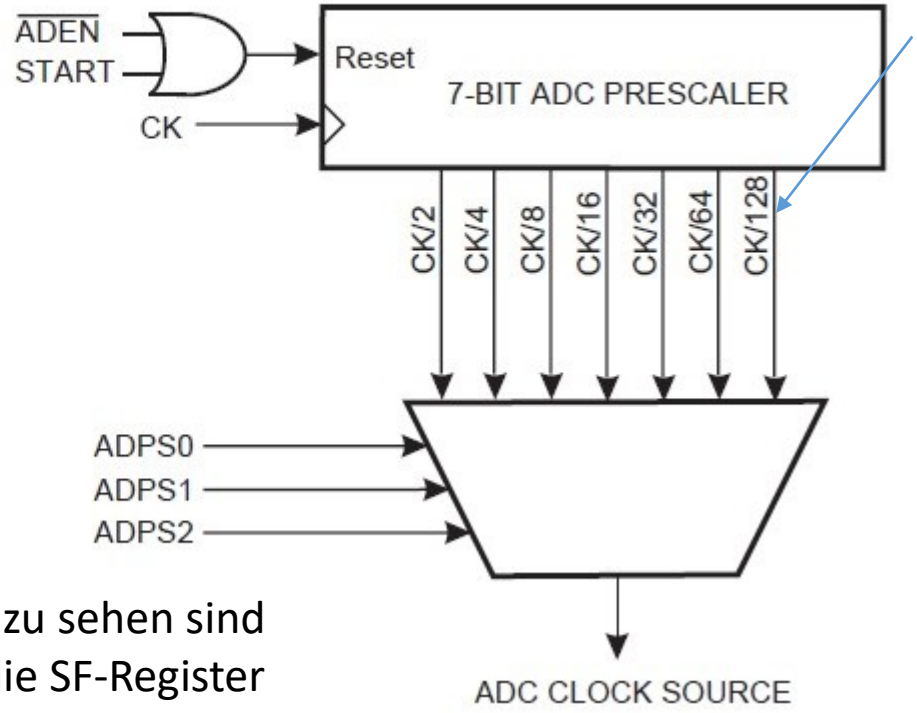
Die interne ADC Wandler Takt- wird vom Prescaler bestimmt.

Die zuständigen SFRs steuern das Hardware-MUX (Multiplexer) und andere Möglichkeiten.
z.B.
welche UREF wird genutzt (Bandgap) oder externe UREF)

Auszug aus PDF: By default, the successive approximation circuitry requires an input clock frequency between **50kHz and 200 kHz** to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate.

Prescaling and Conversion Timing

Figure 23-3. ADC Prescaler



Ermittlung des Vorteilers:
16MHZ/ 200KHz → ~80
Wir haben aber nur /64 und /128 zur Verfügung.
Wir nehmen den schnellstmöglichen <200KHz → /128
16MHZ /**128** = 125KHz.

Das ist schnell genug !

Schön zu sehen sind auch die SF-Register Bit-Namen

REFSx Referenz Spannung

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 23-3. Voltage Reference Selections for ADC

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal V_{ref} turned off
0	1	AV_{CC} with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 1.1V Voltage Reference with external capacitor at AREF pin

Vorschlag → **ADMUX = (1<< REFS0)**

..weiter geht es… die MUX Bits sind ebenfalls in ADMUX Register.

ADMUX – ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ADC0..7 sind schön binär gezählt. Dies verführt zu der Idee, den Eingang gleich *auswählbar* zumachen → **ADMUX = (1<< REFS0) | u8ADCChannel**

Table 23-4. Input Channel Selections

MUX3..0	Single Ended Input
0000	ADC0
0001	ADC1
0010	ADC2
0011	ADC3
0100	ADC4
0101	ADC5
0110	ADC6
0111	ADC7
1000	ADC8[(1)]
1001	(reserved)

Wähle aus:

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Weiter geht es mit dem Prescaler. Wir wollen /128 für den 16 MHZ Quarz.
16MHZ /**128** = 125KHz.

Table 23-5. ADC Prescaler Selections

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

→ **ADCSRA = (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);**

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Noch zwei Bits im ADCSRA Register sind interessant.
Auszug aus der Original-PDF des ATMega328p:

- **Bit 7 – ADEN: ADC Enable**

Writing this bit to one enables the ADC. By writing it to zero, the ADC is turned off. Turning the ADC off while a conversion is in progress, will terminate this conversion.

- **Bit 6 – ADSC: ADC Start Conversion**

In Single Conversion mode, write this bit to one to start each conversion. In Free Running mode, write this bit to one to start the first conversion. The first conversion after ADSC has been written after the ADC has been enabled, or if ADSC is written at the same time as the ADC is enabled, will take 25 ADC clock cycles instead of the normal 13. This first conversion performs initialization of the ADC.

ADSC will read as one as long as a conversion is in progress. When the conversion is complete, it returns to zero. Writing zero to this bit has no effect.

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Noch was.. Die ADC-Unit (Hardware) selbst ist (default) deaktiviert. Das spart Strom! Diese schalten wir noch ein: Mit „**ADEN**“ → Analog device enabled

ADCSRA = (1<<ADEN) | (1<<ADPS2) | (1<<ADPS1) | (1<<ADPS0);

- Der ADC hat nun einen Arbeitstakt.
- Eine Reference ist ausgewählt
- Ein Eingangskanal und Modus ist selektiert.
- <u>Jetzt wird die eigentliche Messung gestartet</u>

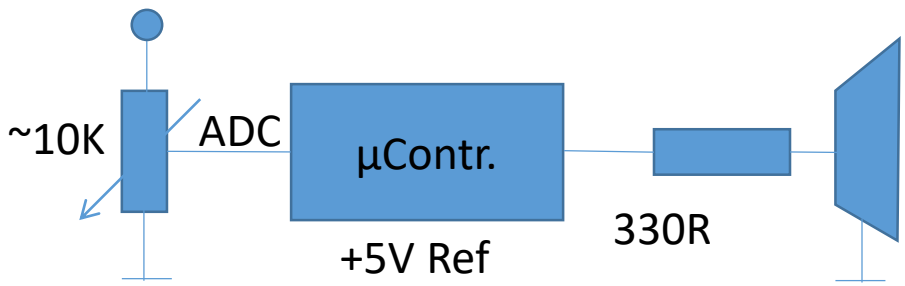
ADCSRA |= _BV(ADSC); //ADC **S**tart **c**onversion

Fall 1: WARTEN→ bis der ADC fertig ist. (**ADSC-Bit** wird wieder **0**)
…das geht befriedigend gut.

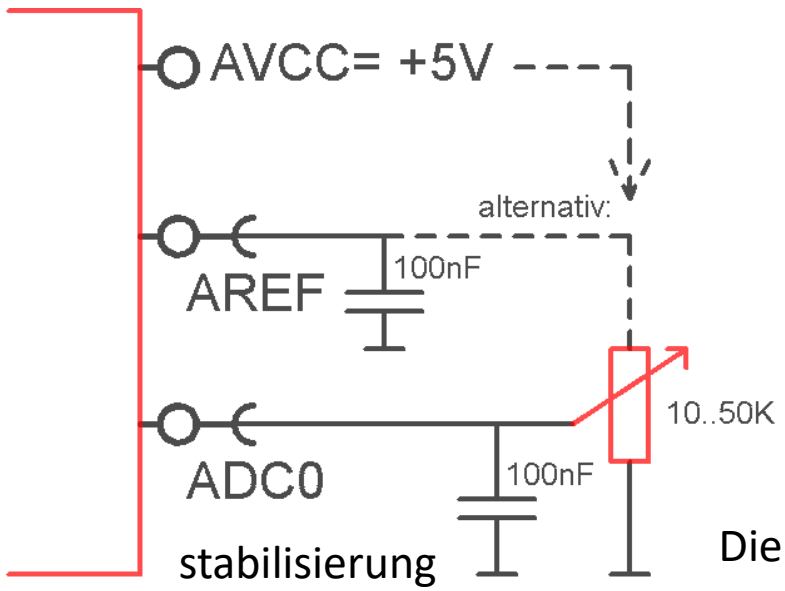
Fall 2: Alternative Möglichkeit ist einen ADC Interrupt zu aktivieren.
...dann muss das Ergebnis *schnell* in der Interrupt Routine verwaltet werden.

Ton-Generator (mit kleinen Schwächen)

Vorsicht. Fehler: Interrupts FRQ kann > mainloopspeed sein…



Erwartung:
je niedriger die Spannung, desto höher der Ton:
Warum? → Je kleiner der Wert im Compare-Register, umso schneller erfolgt der Match.



Ton ist etwas wackelig..
Da die letzten Bits etwas unstabil sind.
http://www.mikrocontroller.net/articles/AVR_16-Bit-Register

Der „**Quantisierungs**“-Effekt ist um so größer, je kleiner de OCR-Register Wert ist.

Die Referenzspannung für das POTI kann geschickt aus der AREF gewonnen werden.
Vorteil: Schwankungen der AREF gehen synchron mit der Poti-ADC0 Spannung

```
// ******************************
uint16_t ADC_10Bit( uint8_t u8Chan )
//  AREF_Extern
// ******************************
{
DDRC &= ~(1<<u8Chan);          //Port C Channel Bit Input
PORTC &= ~(1<<u8Chan);        //Wegen Pullup abschalten
                                 // Activate ADC with Prescaler 16 --> 16Mhz/128 = 125
KHz
ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) |_BV(ADPS0);
ADMUX = (1<< REFS0) | u8Chan;            //AVCC und AREF mit C an GND
ADCSRA |= (1<< ADSC);          //Start conversion
while (ADCSRA & (1<< ADSC) )               // wait <150 uS until conversion completed
            {  //150µS
            //wdt_reset();
            };
return  ADC;            // 16Bit Rueckgabe.  get converted value
};
```

Die Laufzeit der **ADC_10Bit(x)** Funktion ist <**150µS**
Bsp.: Beim Timer ist die Max , die damit gesteuert werden kann: F=1/T = ~6666
~6600 Wandlungen/Sekunde.

ADC ,mit Interrupt → **Bit 3 – ADIE: ADC Interrupt Enable**

ADCSRA – ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
ISR( ADC_vect )    // Dieser Vector steht in der „iom328p.h“
{
OCR1A= ADC * 64U;  // 16 BIT = 65536
};

void ADC_10Bit_INT( uint8_t u8Chan )
{
DDRC &= ~_BV(u8Chan);                          //Portx Channel Bit Input
PORTC &= ~_BV(u8Chan);                         //Pullup abschalten
                //Activate ADC with Prescaler 16 --> 16Mhz/128 = 125 Khz
ADMUX = (1<< REFS0) | u8Chan;          //AVCC und AREF mit C an GND
ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) |_BV(ADPS0);
ADCSRA |= _BV(ADSC) | (1<< ADIE);    //Start conversion + ADC INT ENABLE
};
                // IN MAIN LOOP
if( !(ADCSRA & (1<< ADSC)) ) // Prüfe ob nicht schon eine Wandlung läuft.
                {
                ADC_10Bit_INT( 0 );          // Funktionsaufruf
                };
```

Diskutiere
??
Was passiert wann und warum ?

Erwartung: Je niedriger die Spannung, desto höher der Ton:
Warum? → Je kleiner der Wert im Compare Register, umso schneller erfolgt der Match.

Methode: Bitweise Abtastung einer Variable beliebiger Größe oder **Bitweiser-Scan**

n	7	6	5	4	3	2	1	0	BitNr.	
1	0	1	0	0	1	1	0	0	Value	
0	0	0	0	0	1	0	0	0		(1<< 3)
0	0	0	0	1	0	0	0	0		(1 << 4)

Eine BYTE/WORD wird **bitweise analysiert** indem man eine **„1“** an jede einzelne Stelle durchschiebt und mit **if(uxVariable & (1<< N))**
an dieser Stelle eine **binären UND Vergleich** vollzieht.
Wenn die Stelle eine „1“ war ist die if Bedingung **TRUE**, bei „0“ **FALSE**.

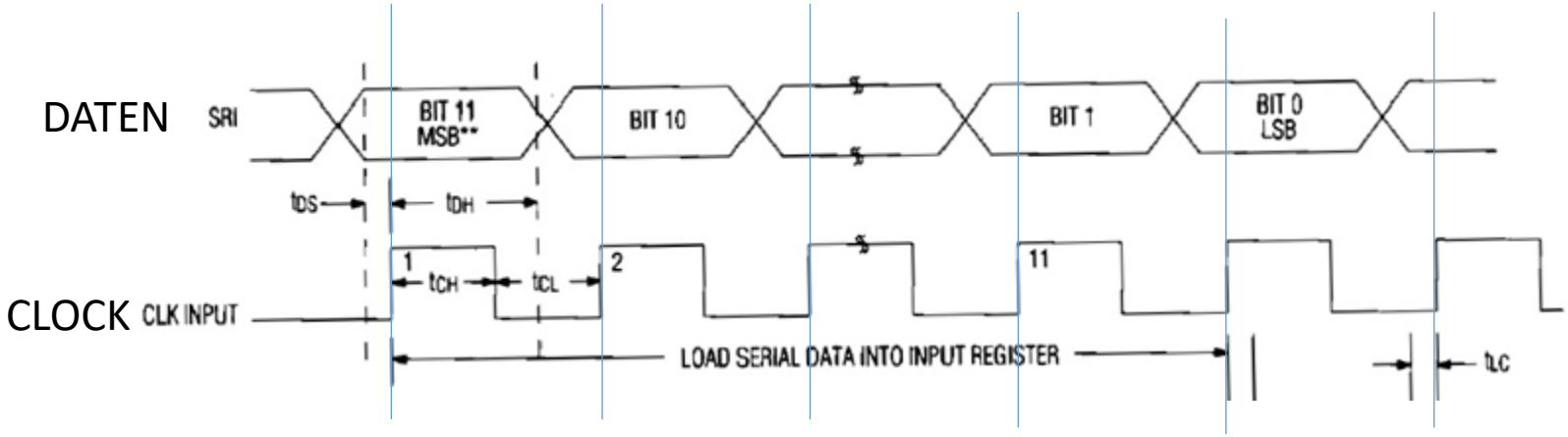
MERKE: Der Scan kann “von rechts (Bit0) nach links (Bit 7/15) oder umgekehrt, also von **0..bis..N** oder von **N..bis..0** erfolgen. Also auch **RÜCKWÄRTS**

Methode: Bit-Scan angewendet um nach dem Bit-Zustand in einer Variable zu handeln.

1.) Dazu bracht es einfach eine Schleife mit einem Zähler für die zu scannenden Bits. Es können ja auch viel mehr (z.B. 32 Bits)
2.) Dann folgt der Scann
3.) Darin dann die Entscheidung was zu tun ist. Meist wird synchron ein Bit an einem Port gesetzt oder gelöscht.

```
void ScanWord( uint16_t u16Value )
{
uint8_t u8NN = 15;                // 16-1   Rückwärst Scan…
do
        {
        if( u16Value   &   (1UL << u8NN ) )
                {
                SetDataPin();     _delay_us(1);
                }
        else
                {
                ClearDataPin();  _delay_us(1);
                };
        }while( u8NN--)
```

Bitweise Abtastung einer Variable (Prinzip)



Analysiere eine Variable bitweise mit dem **UND**-Vergleicher einer geschobenen **„1“**

```
u8NN = ShiftBitsCount - 1:      //→z.B 8Bit -1
do  // Clock out bits
   { if (   u8OutVal  &  (1U << u8NN)  )   // Bitweiser & vergleich?
                        {  SET_PIN_TO_1();  }            //Wenn  „1“ Dann setze Data auch 1
            else
                        {  SET_PIN_TO_1();  };           //Wenn  „1“ Dann setze Data auch 0
           _delay_us(1);                        // Warte bist Bit stabil steht
                                                //Clock Job
           CLOCK_TO 1();   _delay_us(1);            //Shift Clock = Übertragung des Bits
           CLOCK_TO 0 ();  _delay_us(1);
}while( u8NN-- );
```

Selbstbau-**ADC** z.B. mit dem internen Komparator OP

Es gibt trickreiche Methoden:
Dual-Slope ADC (auch nicht der schnellste Weg, aber schneller als..
Delta Sigma ADS (langsam, hohe Genauigkeit)
Sukzsessive Approximationsverfahren (Das schnellste Verfahren . Auflösung hängt von der Qualität des Aufbaus ab, Integrationszeiten, Rauschen, Schwingen etc..) Was das ist… Eine Wissenschaft für sich . Selbst einfachmal Googeln.

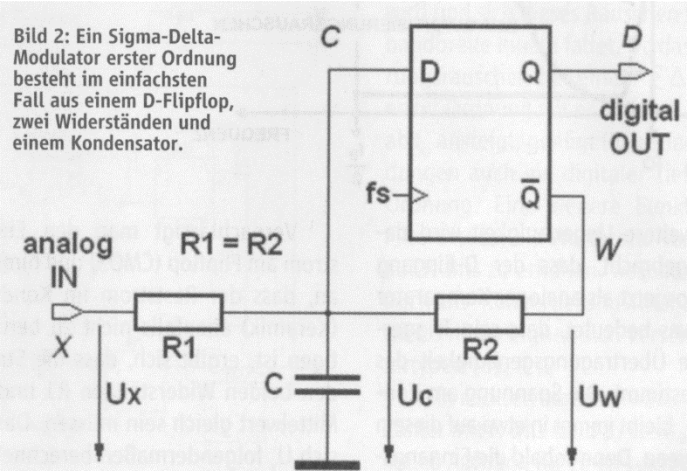
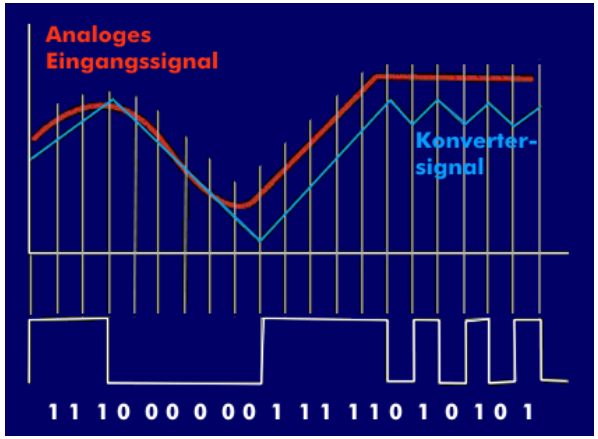


Bild 2: Ein Sigma-Delta-Modulator erster Ordnung besteht im einfachsten Fall aus einem D-Flipflop, zwei Widerständen und einem Kondensator.

Hier einige Prinzip Bilder zum Delta Sigma



Delta Modulation

D-FlipFlop + Fs Signal
https://www.itwissen.info/Sigma-Delta-Wandler-SDW-sigma-delta-converter.html

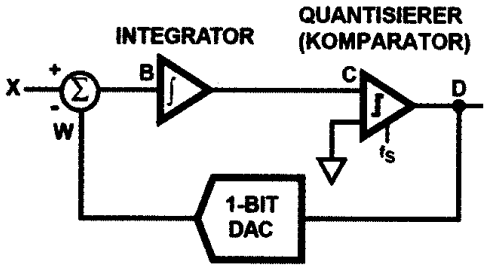
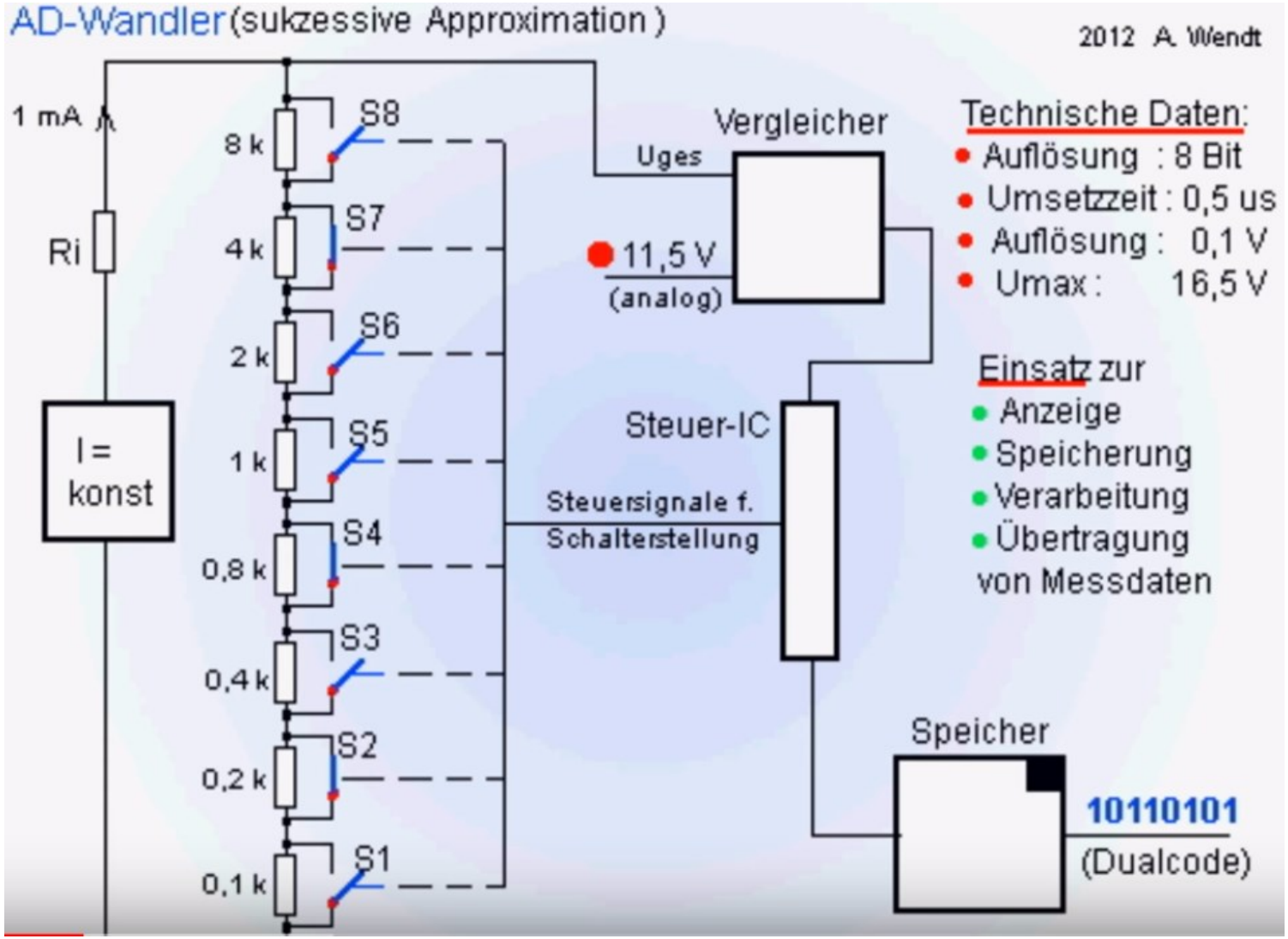
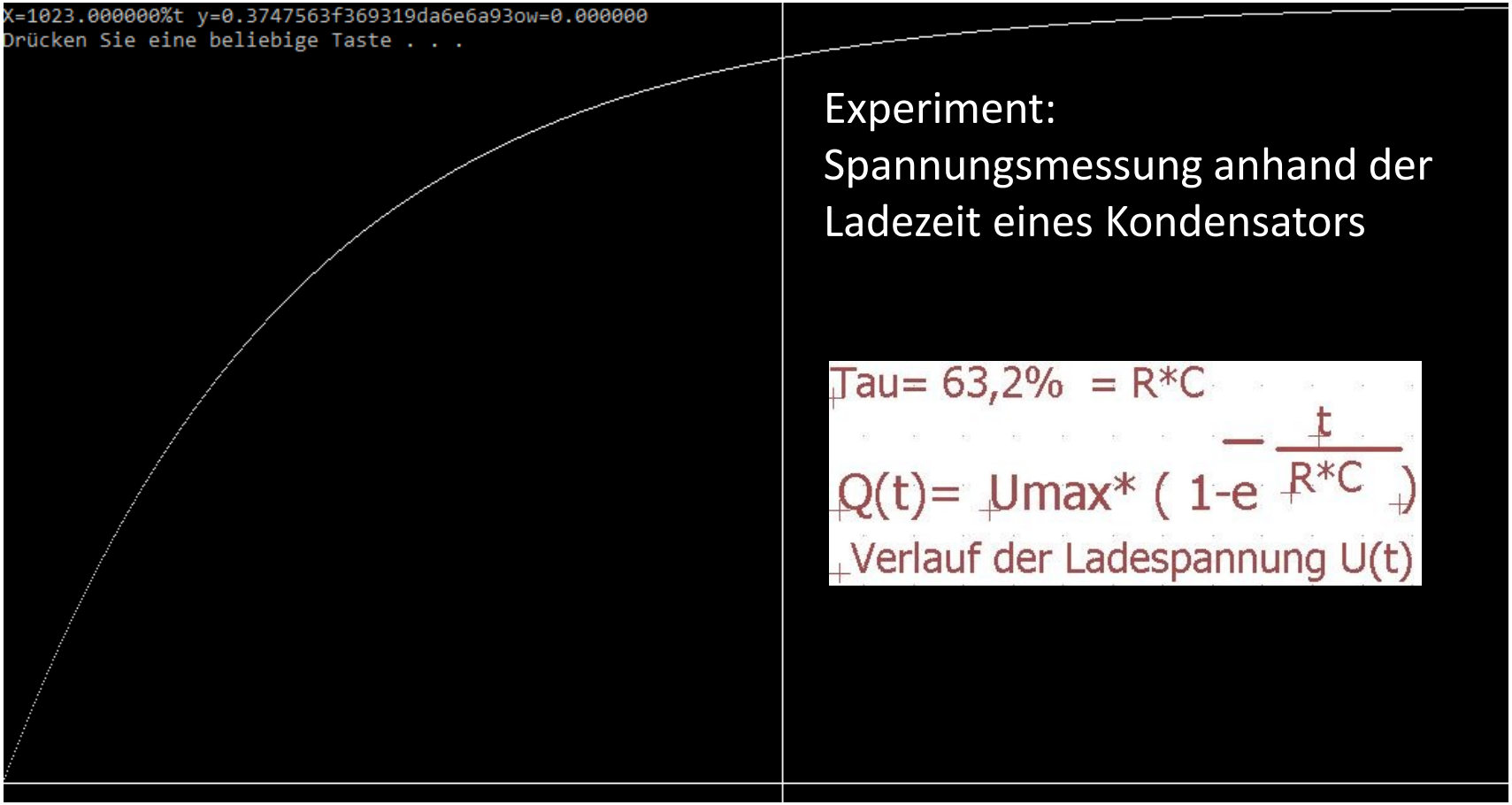


Bild 1: Blockschaltbild eines Sigma-Delta-Modulators erster Ordnung.


AD-Wandler (sukzessive Approximation)
2012 A. Wendt
1 mA
Ri
I = konst
8 k
4 k
2 k
1 k
0,8 k
0,4 k
0,2 k
0,1 k
S8
S7
S6
S5
S4
S3
S2
S1
Vergleicher
Uges
11,5 V
(analog)
Steuer-IC
Steuersignale f.
Schalterstellung
Speicher
10110101
(Dualcode)
Technische Daten:
Auflösung : 8 Bit
Umsetzzeit : 0,5 us
Auflösung : 0,1 V
Umax : 16,5 V
Einsatz zur
Anzeige
Speicherung
Verarbeitung
Übertragung
von Messdaten



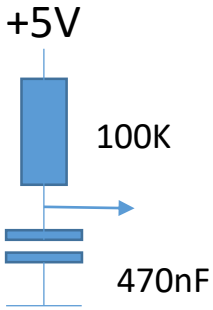
Experiment:
Spannungsmessung anhand der Ladezeit eines Kondensators

Tau= 63,2% = R*C

$$Q(t) = Umax * \left(1 - e^{-\frac{t}{R*C}}\right)$$

Verlauf der Ladespannung U(t)

5 Tau Kurve



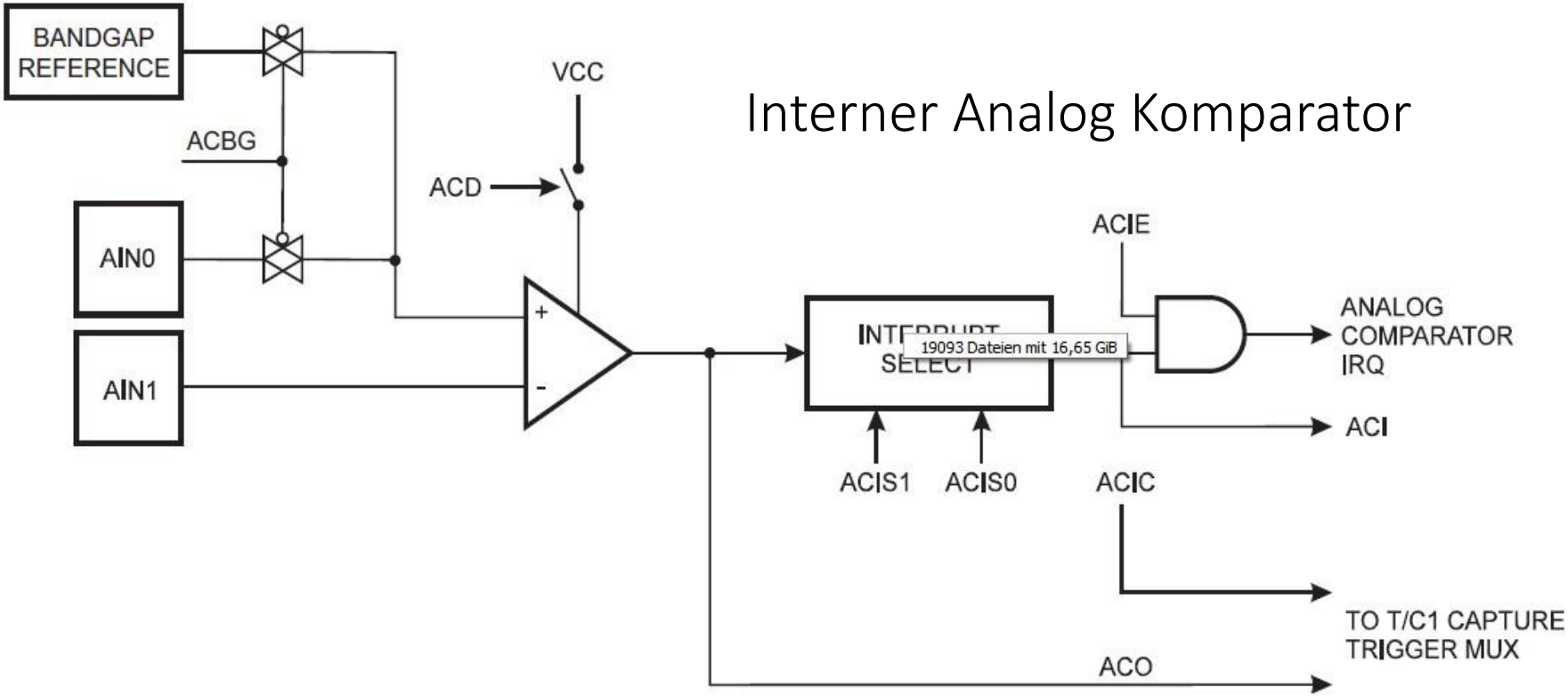

```
#define EE 2.71828182818281828
f16YY =  2.5*  (1 - pow(EE, (-1.0*(fTT/(fRR*fCC))))) ;
```

oder

```
f16YY =  2.5*  (1 - exp( (-1.0*(fTT/(fRR*fCC))) ));
```

T-Const?=47mS --> 1Tau = 63,2% Umax
Grenzfrequenz = ~3.9Hz

Interner Analog Komparator



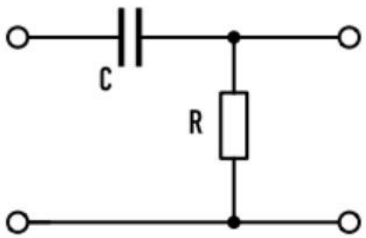
ACSR – Analog Comparator Control and Status Register

Bit	7	6	5	4	3			0	
0x08 (0x28)	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

DIDR – Digital Input Disable Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	–	–	–	–	–	–	AIN1D	AIN0D	DIDR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Analog Comparator als Spannungsmessgerät



Bassabsenkung

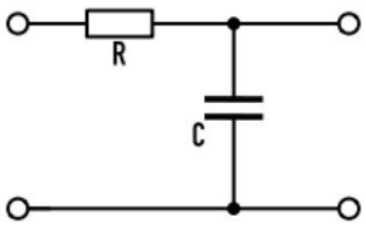
$$f_c = \frac{1}{2 \cdot \pi \cdot R \cdot C} = \frac{1}{2 \cdot \pi \cdot \tau}$$

$$\tau = R \cdot C = \frac{1}{2 \cdot \pi \cdot f_c}$$

Beziehungsgleichungen:

f_c in Hz = 159155 / τ in µs

τ in µs = 159155 / f_c in Hz



Höhenabsenkung

An den Analog Comparator anschließen.

*AN1 (+) = LDR (Bsp.)

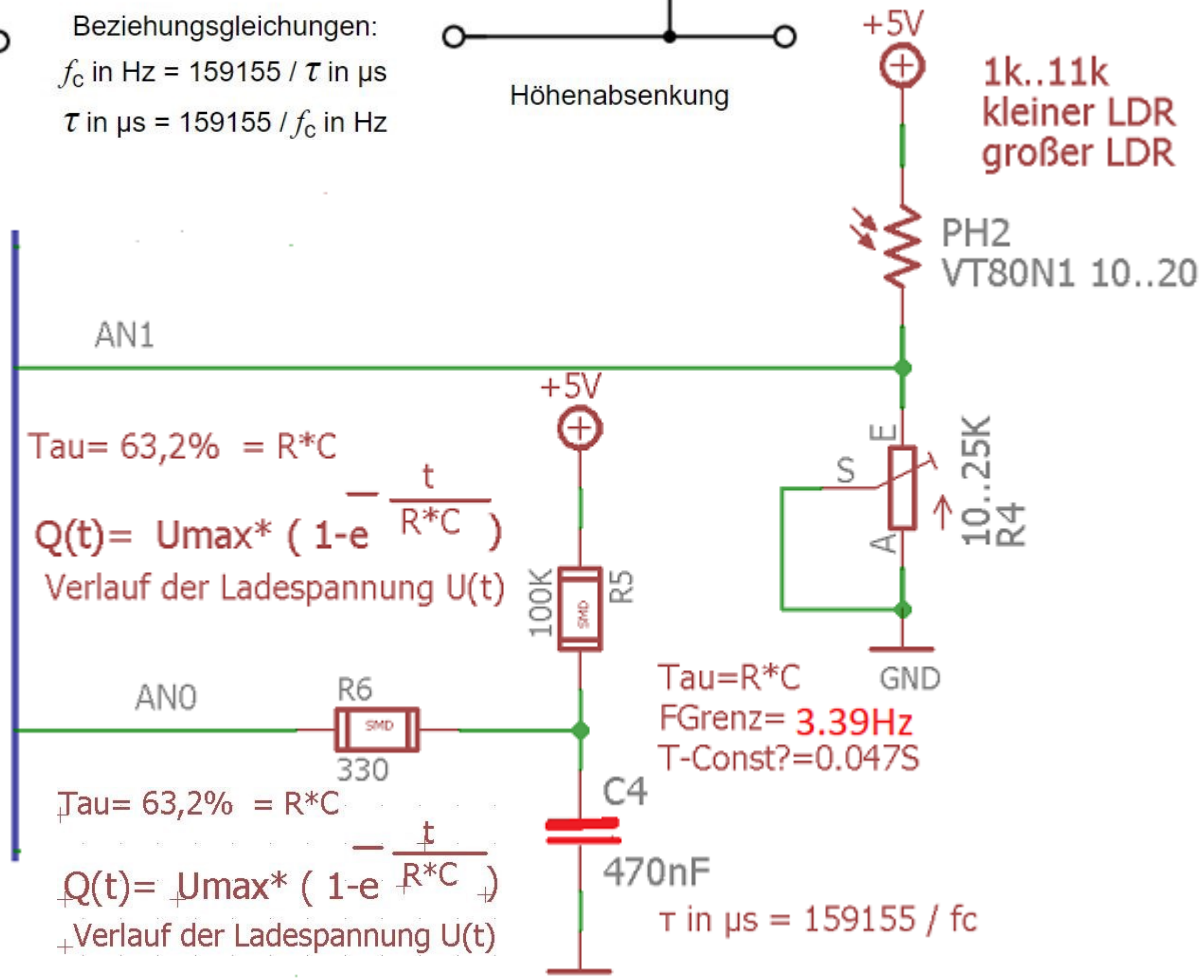
*AN0 (-) =1.Out=0V= Kondensator löschen.

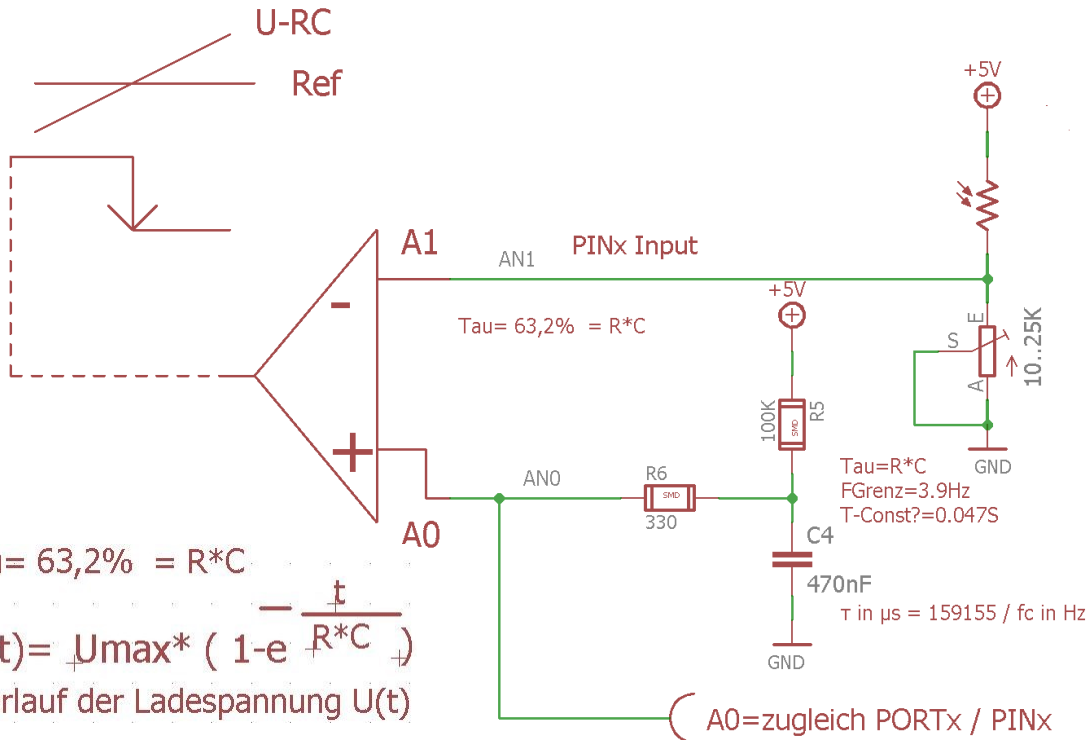
*Timer 16Bit Starten,

*AN0(-)=Input Amp-Comp.

*Case Compare→ Comparator Interrupt liest Timerstand →T.

*Mit E-Funktion des T, die Spannung ermitteln.

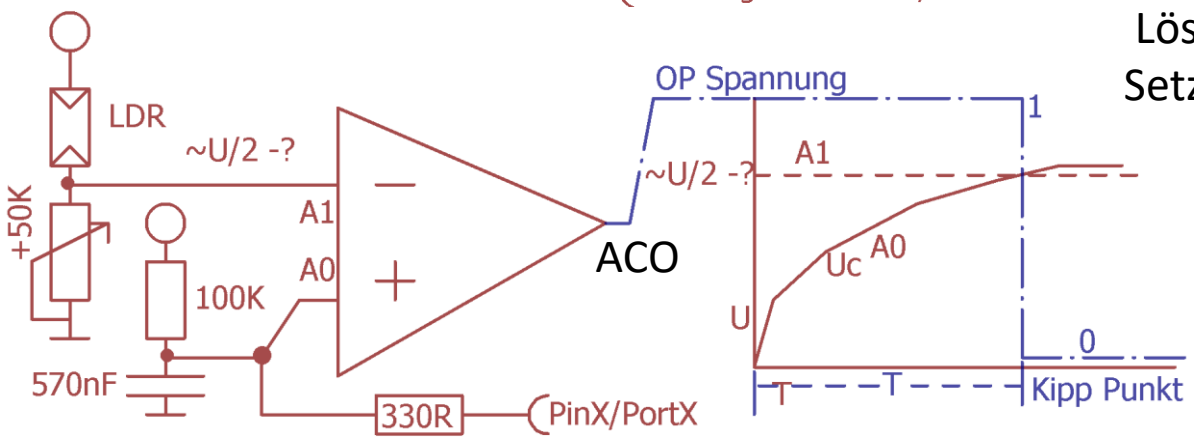




Tau= 63,2% = R*C

$$Q(t) = Umax * \left(1 - e^{-\frac{t}{R*C}}\right)$$

Verlauf der Ladespannung U(t)



gfAnaCompVolt = U_MAX_INTEGRATOR * (1.0 - exp(-1.0 * ((ICR1/TIMER1_TICKS_PRO_SEC)/(R_INTEGR*C_INTEGR))))

Sekundenweise Triggern:

Lösche RC-Glied an PORTx-A0=
A0 Pin auf Output mit 0V .
Stelle 16Bit Timer1 auf 0
Stelle AnaComp Parameter mit
Übertrag von TCNT1 auf ICR1.
Aktiviere AnaComp Interrupt
auf fallende Flanke.
A0 auf Input schalten, PORTx=0
Aktiviere Timer1 mit F_CPU/64

In ISR(ANALOG_COMP_vect)

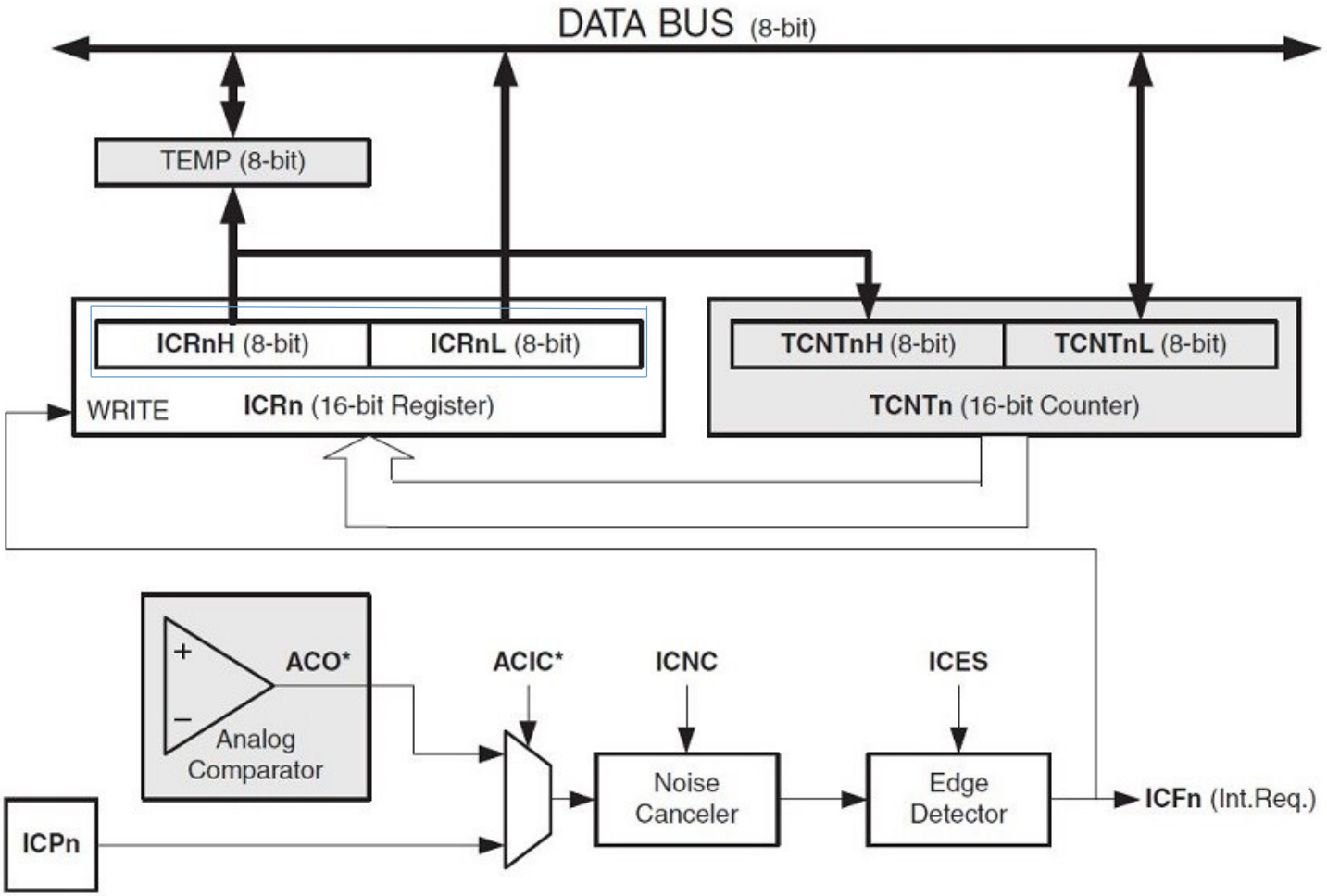
Stoppe Timer1
Lösche wieder RC AN A0 mit 0V
Setze „AnaCompReadyFlag“

In main():

if(AnaCompReadyFlag)
Berechne mit ICP1
Wert und E Funktion
die Spannung
TimerTicks/sec=
F_CPU/64
Ruhezustand setzen.

Der Analog Komaparator kann den 16 Bit Timer-1 Steuern

Figure 16-3. Input Capture Unit Block Diagram



```
//Analog Comp
#define ANACOMP_DDR         DDRD
#define ANACCOMP_PORT       PORTD
#define ANACCOMP_PIN        PIND
#define AN0_PLUS_PIN        6
#define AN1_MINUS_PIN       7
#define AN0_PLUS_BIT        (1 << AN0_PLUS_PIN) // Aus der Pin Nummer wird die reale Bitposition
#define AN1_MINUS_BIT       (1 << AN1_MINUS_PIN) // Aus der Pin Nummer wird die reale Bitposition
// ---
#define READ_A0_PLUS()      (ANACCOMP_PIN & AN0_PLUS_BIT)
#define ANA_COMP_A0_PLUS_RC_INPUT()                 (ANACOMP_DDR &= ~(AN0_PLUS_BIT))
#define ANA_COMP_A0_PLUS_RC_OUTPUT()                (ANACOMP_DDR |= (AN0_PLUS_BIT))
#define ANA_COMP_A0_PLUS_RC_Clear()    (ANACCOMP_PORT &= ~(AN0_PLUS_BIT))
//#define ANA_COMP_A0_PLUS_RC_SET()    (ANACCOMP_PORT |= (AN0_PLUS_BIT))
// ---
#define ANA_COMP_A1_MINUS_LDR_INTPUT()                          (ANACOMP_DDR &= ~(AN1_MINUS_BIT))
#define ANA_COMP_A1_MINUS_LDR_CLEARPULL()           (ANACCOMP_PORT &= ~(AN1_MINUS_BIT))


#define U_MAX_INTEGRATOR 5.0
#define R_INTEGR 100000.0
#define C_INTEGR 0.000000575
#define MEASURED_UNLOADINGTIME 150 //<100 uS Mikrosek
#define TIMER1_TICKS_PRO_SEC                (F_CPU/64.0)
// ..........................................

#define STATUS_TICK_EVENT                   0x08
#define RC_ADC_READY_FLAG                   0x10
#define RC_ADC_LOCKED                       0x20
#define RC_ADC_NEW_VALUE_COMPUTED  0x40
//gu8StatusFlags
```

```
// ********************************************
void TriggerAnalogComparator_RC_Messung(void)
// ********************************************
 {
//ICNC      ICES        ACIC        ACO*        ICP*        ICF1        ICR1H ICR1L            ICR1
ANA_COMP_A1_MINUS_LDR_INTPUT();
ANA_COMP_A1_MINUS_LDR_CLEARPULL();
TCCR1A = 0;
TCNT1 = 0;
Clear_RC_Glied();
ACSR = _BV(ACIE) | _BV(ACIC) | _BV(ACIS1);  //Int Enabel,InpCapture ICP1,Falling Edge,ACI=Flag=1(clear)
DIDR1 = _BV(AIN1D) | _BV(AIN0D);  // //not neccessary  Schalte A1=minus, A0=plus reine AnaComp
ANA_COMP_A0_PLUS_RC_Clear(); // jetzt lädt der Kondensator
ANA_COMP_A0_PLUS_RC_INPUT();
//  --> ICR1 = Input capture 16 Bt Register
// MERKE NICHT TCNT1 sondern ICR1 bekommt Wert von TCNT1 im AC=Comp Fall.
// Stelle Timer1   auf 16MHz/64
TCCR1B = _BV(CS10)| _BV(CS11);   //  F_CPU/64
};
```

```
// ************************************
ISR( ANALOG_COMP_vect ) //ANA_COMP_vect
// ************************************
{
TCCR1B= 0;  TCCR1A=0;
gu8StatusFlags |= RC_ADC_READY_FLAG | RC_ADC_LOCKED;  // neue Messung vorbereiten
};
```

```
// *************************
void Clear_RC_Glied(void)
// *************************
{
DIDR1 &= ~_BV(AIN0D); //not neccessary, A0 Plus auf normale Port Operation
ANA_COMP_A0_PLUS_RC_OUTPUT();
ANA_COMP_A0_PLUS_RC_Clear();
wdt_reset();
_delay_us(MEASURED_UNLOADINGTIME); //~50 <100 uS
wdt_reset();
gu8StatusFlags &= ~RC_ADC_LOCKED;
};
```

Im main() Sekunden Tick:

……while(1) …..

if(gu8StatusFlags &STATUS_TICK_EVENT)

Tau= 63,2% = R*C

$$Q(t) = Umax * \left(1 - e^{-\frac{t}{R*C}}\right)$$

Verlauf der Ladespannung U(t)

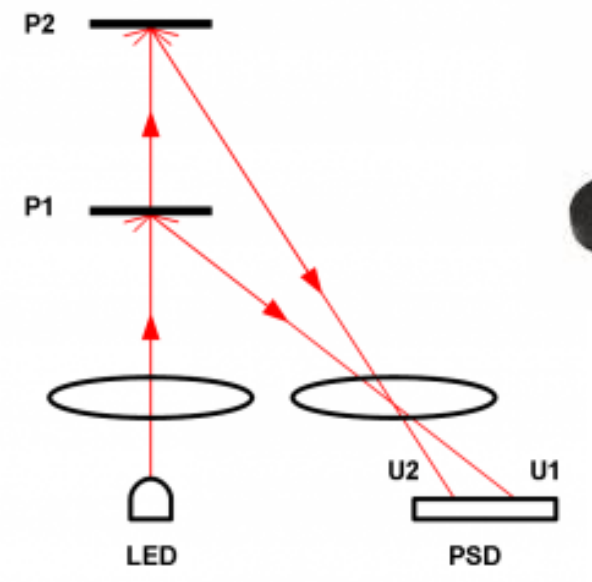
```
        {
        gu8StatusFlags &= ~STATUS_TICK_EVENT;
        if( !(gu8StatusFlags & RC_ADC_LOCKED) )
                {
                TriggerAnalogComparator_RC_Messungg();
                }
        else
                {
                if( gu8StatusFlags & RC_ADC_READY_FLAG)
                        {
                        // ICR1 Input Capture Register
                        //U=ZMAX'1-e^(-t/(R+C))
gfAnaCompVolt = U_MAX_INTEGRATOR * (1.0-                                    \
exp(-1.0*((ICR1/TIMER1_TICKS_PRO_SEC)/(R_INTEGR*C_INTEGR)) ));
                        gu8StatusFlags &= ~RC_ADC_READY_FLAG;
                        };
                Clear_RC_Glied();
                };
        uart_puts_p( PSTR("AnaCmp_Volt:") )
        dtostrf(gfAnaCompVolt, 8, 8, gcaNumStr );   //[-]dd.ddd;
        uart_puts( gcaNumStr );
        };
```

Software Technik
viele PWM mit Interrupt

Praxis - Analogeingang
Bsp.: Nutzung als IR-Entfernungsmesser

Grundlage: **Analog Sharp Entfernungsmesser GHP2D12**
Teuer ~15€/Stück
Nur 3 Leitungen:
+5V, GND, ANALOG OUT .-.**0 .. 2.25** V → siehe Datenblatt

Messungen ~10cm..50cm..max 80cm



z.B. Fahrzeug mit integriertem Mikrocontroller

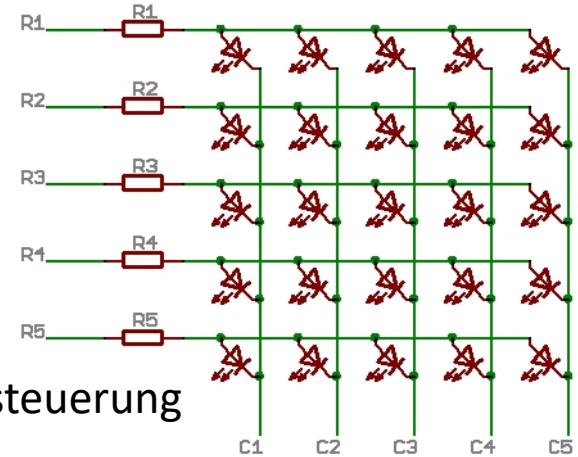
Fortschrittliche Software Techniken

Interrupt Methoden der Anwendung

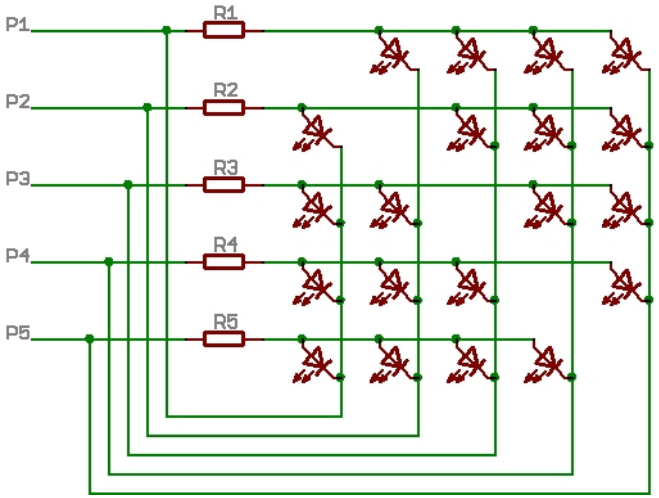
Charlie Plexing

Mir hat sich die Frage gestellt, wie man **8x7**=56 LEDs nur mit 8 Steuerleitungen „A-H“ darstellen will.
Der Trick heißt **„Charlie-Plexing**“. → das How to ?
Ein Pin hat bis zu 3(~4) Zustände:

1. Output: **0** (0V)
2. OutPut: **1** (~5V)
3. Input-mit Pullup: schwache_5V (nicht relevant)
4. Input: **Hochohmig=TRI-STATE**
   Hier wird es Interessant!..



Normale Matrix Ansteuerung



Verdrahtung:
Charlie-Plexing Ansteuerung

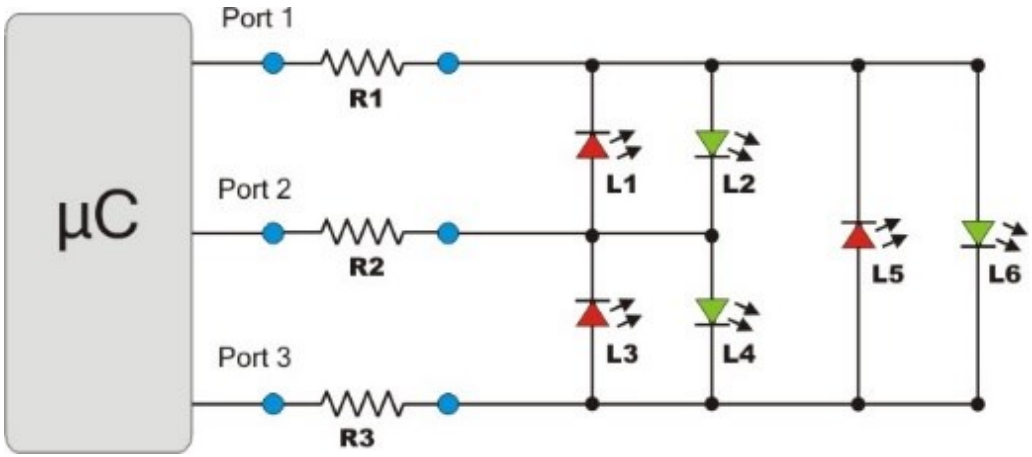
Youtube: https://youtu.be/fM7R5IJtgz0

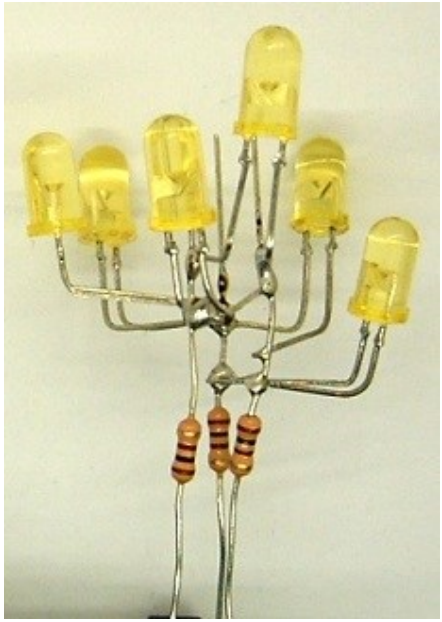
http://www.elektronik-labor.de/AVR/Charlieplexing.html
https://learn.sparkfun.com/tutorials/sparkfun-led-array-8x7-hookup-guide/all

Aus den 4 möglichen Zuständen lassen sich neue Kombinationsmöglichkeiten ableiten.
Und diese lassen sich Kombinieren

P1:+, P2:-, P3:Tri-state
P1:-, P2:+, P3:Tristate
P1:Tristate, P2:+, P3:-
P1:Tristate, P2:-, P3:+
P1:+, P2:Tristate, P3:-
P1:-, P2: Tri-state, P3:+
→ 6 Kombinationen mit Tri-state, mit 3 Leitungen. Dazu kommen noch „ohne Tri-state“.
Also müssten auch 6 LEDs mit 3 Leitungen steuerbar sein.
LEDs sind ja selbst Dioden, und somit in der Matrix definiert ansprechbar.

aus





Aus:
http://www.elektronik-labor.de/AVR/Charlieplexing.html

Organisation 8x7

H	G	F	E	D	C	B	A	
7	6	5	4	3	2	1	0	
							1	0
						1		1
					1			2
				1				3
			1					4
		1						5
	1							6
1								7

oberste Reihe selektiert

Alles was OUT=0V ist, leuchtet

Alles was NC=Input ist, ist dunkel

unterste Reihe selektiert

Die „**1**“ ist das Reihen Selektier Bit mit +3.3V..5V

8 Reihen
7 Spalten

1. Die Spalte hat 7 Bits bei 8 Reihen.
2. Das 8te Bit selektiert die waagerechte Linie mit +5V
3. Da das Reihen Selektier-Bit sich durchschiebt, ist das Datenbyte(7Bit) an dieser Stelle zu Splitten, und die rechte Hälfte um 1 nach rechts zu schieben .
4. Alles was NICHT leuchten soll, bekommt im Daten Direktion Register eine 0=Input auf dieses Bit. Das dazugehörige PORT Bit = 0 = „Pullup Off“
5. Alles was leuchten soll, bekommt im Daten Direction Register eine 1, Aber im PORT Register wir an diesem BIT eine 0=0V geschrieben.

ERGO, Das Port bekommt immer eine 0, außer das Selektier Bit. ***PORTD=(1<<u8NN);***

https://homepages.uni-regensburg.de/~erc24492/Charlie_Plexing/Charlie_Plexing.html

Beispiel der FONT Datei font7x8.h

```
#include <avr/pgmspace.h>
ifndef FONT7X8_H
        #define FONT7X8_H
#define FONT_CHAR_FIRST    32
#define FONT_CHAR_LAST     126
#define FONT_CHARS         95
#define FONT_WIDTH         7
#define FONT_HEIGHT        8
#define FONT_SPACING       1
{
PROGMEM static const prog_char cmaFont7x8[] = {
        0,   0,  0,  0,  0,  0,  0, // ' '  32
        0,   6, 95, 95,  6,  0,  0, // '!'  33
        0,   7,  7,  0,  7,  7,  0, // '"'  34
        20, 127, 127, 20, 127, 127, 20, // '#'  35
        36, 46, 107, 107, 58, 18,  0, // '$'  36
        70, 102, 48, 24, 12, 102, 98, // '%'  37
        48, 122, 79, 93, 55, 122, 72, // '&'  38
        4,   7,  3,  0,  0,  0,  0, // '''  39
        0,  28, 62, 99, 65,  0,  0, // '('  40
        0,  65, 99, 62, 28,  0,  0, // ')'  41
..............................  usw..
```

```
// 6 Bits auf D
#define A_F_DATA_MASK          0b00111111
#define A_F_DATA_SHIFT_NR 2 //<<
#define A_F_PORT               PORTD
#define A_F_DDR                DDRD
#define A_F_PORT_MASK          0b11111100
// 2 Bits auf B
#define G_H_DATA_MASK          0b11000000
#define G_H_DATA_SHIFT_NR      6        //>>
#define G_H_PORT               PORTB
#define G_H_DDR                DDRB
#define G_H_PORT_MASK          0b00000011
```

Noch sehr wichtig.

Zur Ansteuerung benötigt man eine genaue wie gleichmäßige Zeitscheibe.
Das geht nur über den 8 Bit Timer.
Flackerfrei geht das nur über eine Geschwindigkeit von ~30*/Zeile/Sekunde
30*8 = 240 ist mindestens 240*/Sek. Oder schneller ! Ich wählte 977/Sek.

```
OCR0A = 127; // PWM=Halbe Helligkeit
TCCR0A = 0;
TIMSK0 = (1<<TOIE0) | (1<<OCIE0A);
TCCR0B = (1<<CS01)| (1<<CS00); // F_CPU/256/64= 977

ISR( TIMER0_OVF_vect )
 {
ByteToMatrix(gu8RowNN);
gu8RowNN++;
gu8RowNN %= 8;
};
```

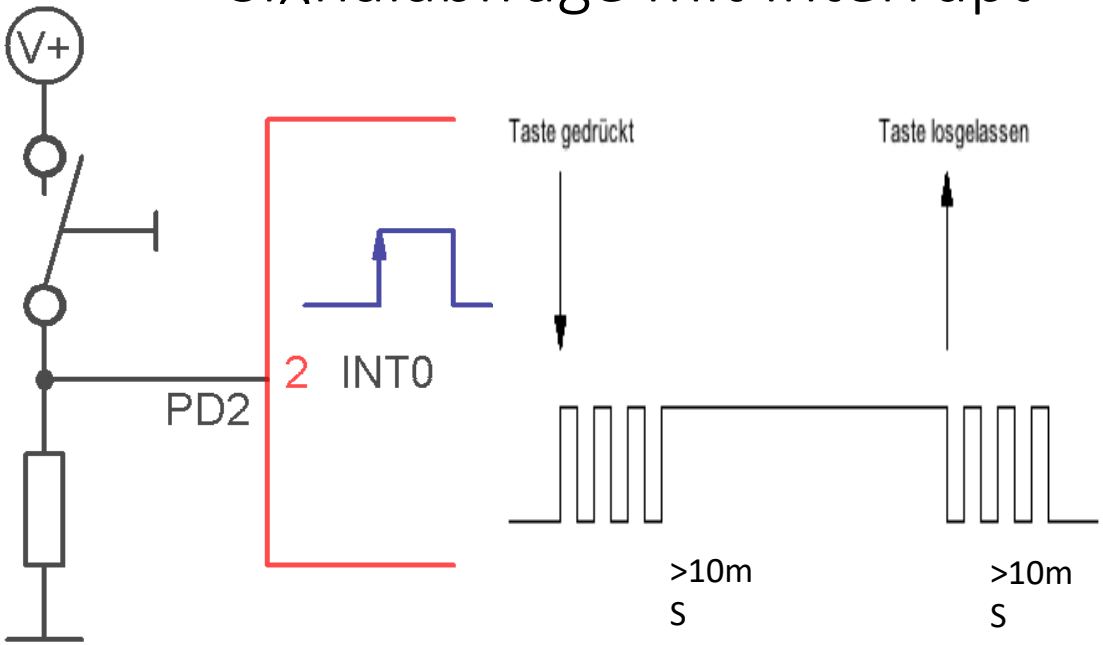
(dazu kommt noch ein PWM Abschaltinterrupt, der die Helligkeit regelt.

```
ISR( TIMER0_COMPA_vect )
{
A_F_DDR &= ~A_F_PORT_MASK;          G_H_DDR &= ~G_H_PORT_MASK;
A_F_PORT &= ~A_F_PORT_MASK;         G_H_PORT &= ~G_H_PORT_MASK;
};
```

```
// ******************************
void ByteToMatrix( uint8_t u8Row )
// ******************************
{
uint8_t u8Split_Low_Data;
uint8_t u8Split_High_Data;
uint8_t u8SParkFun8x7Data;
uint8_t u8Low_0_Maske;
uint8_t u8High_0_Maske;

//Data sind schon "left flush"
u8Low_0_Maske  = (1<<u8Row) - 1;  //bsp3: 1000-"1"=00000111
  // 11110111  & ~00000111   = 11110000

u8High_0_Maske = ~(1<<u8Row) & ~u8Low_0_Maske;
// +5V Selektiert Reihe,                0V = LED ON
u8Split_Low_Data = ((gu8ParkFun[u8Row]) >> 1) & u8Low_0_Maske;
u8Split_High_Data = gu8ParkFun[u8Row]  &  u8High_0_Maske;
u8SParkFun8x7Data = (u8Split_High_Data | u8Split_Low_Data) | (1<<u8Row);

//lösche Port
A_F_DDR &= ~A_F_PORT_MASK;
G_H_DDR &= ~G_H_PORT_MASK;
A_F_PORT &= ~A_F_PORT_MASK;
G_H_PORT &= ~G_H_PORT_MASK;
};
```

```
if( u8Row < 6)
{
A_F_DDR |= (((u8SParkFun8x7Data & A_F_DATA_MASK) | (1<<u8Row) ) << A_F_DATA_SHIFT_NR)
& A_F_PORT_MASK ;

G_H_DDR |= ((u8SParkFun8x7Data & G_H_DATA_MASK) >> G_H_DATA_SHIFT_NR) &
G_H_PORT_MASK;

A_F_PORT |= ((~A_F_DDR) | ((1<<u8Row) << A_F_DATA_SHIFT_NR))  & A_F_PORT_MASK;
G_H_PORT |= (~G_H_DDR) & G_H_PORT_MASK;
}
else  // Row>=6
{
A_F_DDR |= ((u8SParkFun8x7Data & A_F_DATA_MASK) << A_F_DATA_SHIFT_NR) &
A_F_PORT_MASK ;

G_H_DDR |= (((u8SParkFun8x7Data & G_H_DATA_MASK) | (1<<u8Row)) >>
G_H_DATA_SHIFT_NR) & G_H_PORT_MASK;

A_F_PORT |= (~A_F_DDR) & A_F_PORT_MASK;
G_H_PORT |= ((~G_H_DDR) | ((1<<u8Row) >> G_H_DATA_SHIFT_NR)) & G_H_PORT_MASK;
};
```

Signalabfrage mit Interrupt



Vorteil.
Unverzügliche Bearbeitung:

Hier gibt es Besonderheiten die man wissen sollte.
Mechanische Taster prellen.
Und dies sehr oft und sehr lange
>~2, 3, 5..10mS …
Das müssen wir beachten.

Jede Signalflanke löst *(nach entsprechender Hardwaresituation)* ein Interrupt Ereignis aus.
Es wäre fatal, wenn jeder Preller zu sich stapelnden Interrupt Ereignissen führt. *(Stack-Überlauf und Absturz möglich).*
Eine Debouncing Methode ist erforderlich.

Signalauswertung mit Interrupt → INT0

Wir suchen einfach mal nach den SFR für den PIN INT0 und finden:

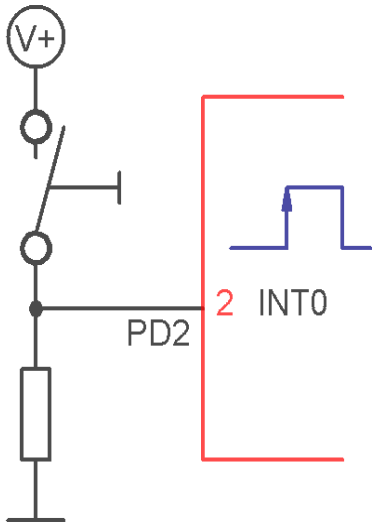
EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 12-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.



Der ATM328 **unterscheidet sich in den SFRs etwas von den 16/8er Typen.**
Um also eine steigende Flanke auszuwerten nehmen wir:

EICRA **|=** (1<< ISC01) **|** (1<< ISC00);

Externen Interrupt einschalten →INT0

Die SFR für den externen Interrupt heißen etwas anders als bei ATM 8/16 etc.
Aber das Prinzip ist immer dasselbe.
Nach der Auswahl auf welche Signalart der externe Interrupt reagieren soll, folgt:

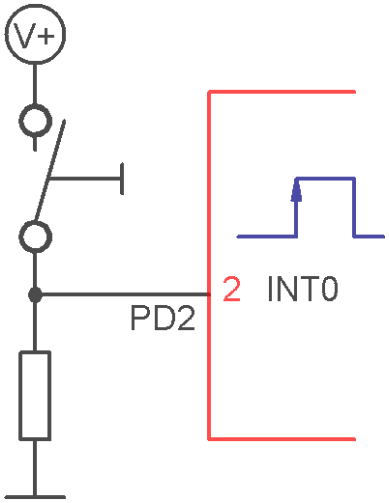
EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD &= ~(1<< PD2); //Port auf Input;

EIMSK |= (1<< INT0); // Interrupt aktivieren

sei(); //globales Flag .→set enable interrupt
→ MCUCR



Ohne Debounce

```
define TASTER_DDR     DDRD
#define TASTER_PORT PORTD
#define TASTER_PIN    PIND
#define TASTER1       (1 << PORTD2)
```

```
ISR( INT0_vect )
{
Toggle_Bit(1);          //Test mit Osci., was passiert
GREEN_LED_PORT ^= GREEN_LED;          // EXOR ist toggle LED
};

int main(void)
{
DDRC |= (1<<PC4);                  // out→TESTbit  für mainspeedtest

GREEN_LED_DDR |= GREEN_LED;          //LED liegt an PORTB
TASTER_DDR &= ~TASTER1;              //(1<< PD2) Port auf Input;
TASTER_PORT &= ~TASTER1;             //(1<< PD2) Pullup löschen

EICRA |= (1<< ISC01) | (1<< ISC00); // Aktiviere steigende Flanke an INT0
EIMSK |= ( 1<< INT0 );             // Interrupt aktivieren
sei();                     // WICHTIG:  global set  enable interrupt
while(1)                   //endless loop
        {
        PORTC ^= (1<<PC4); // Testbit für „Mainllop“speedtest
~1.6MHz
        };    //while
};//main
```

Simple Interrupt:

Beobachte das Verhalten der Software:

Manchmal scheint die LED zu stottern, wenn man auf die Taste drückt. Das liegt am **Prellen** der Taster-Mechanik

EICRA – External Interrupt Control Register A

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
(0x69)	–	–	–	–	ISC11	ISC10	ISC01	ISC00	EICRA
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

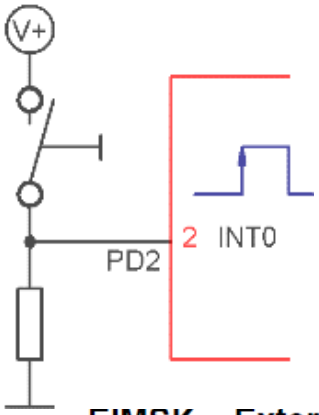


Table 12-2. Interrupt 0 Sense Control

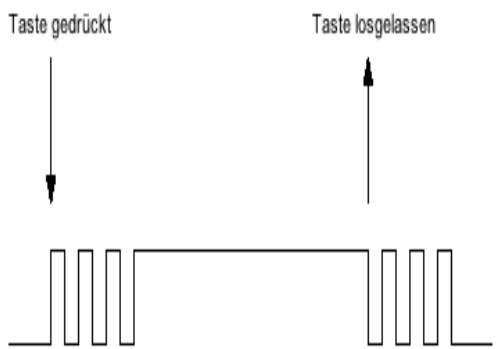
ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

EIMSK – External Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x1D (0x3D)	–	–	–	–	–	–	INT1	INT0	EIMSK
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
EICRA |= (1<< ISC01) | (1<< ISC00); // Aktiviere steigende Flanke an INT0
EIMSK |= ( 1<< INT0 );             // Interrupt aktivieren
sei();  // WICHTIG:  global set  enable interrupt
            ISR( INT0_vect )   {…….schnell und sofort …..};
```

Debouncing Taster-Prellen



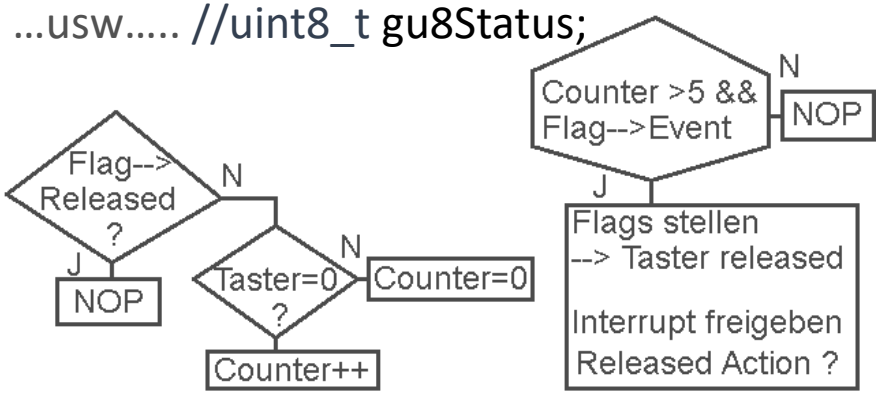
Wir brauchen einen Algorithmus um dies softwaretechnisch zu lösen.
Denn Software kann man anpassen und geht nicht kaputt.
Dazu eignen sich Flags..
Sie verbrauchen nur ein Bit und sind sehr schnell.

Eine der möglichen Verfahren…..
Phasen:
Drücken erkennen nur erlauben, wenn vorher
Taster „nicht gedrückt“ erkannt.
Taster Ereignis →Taster Detect → Flags setzen.

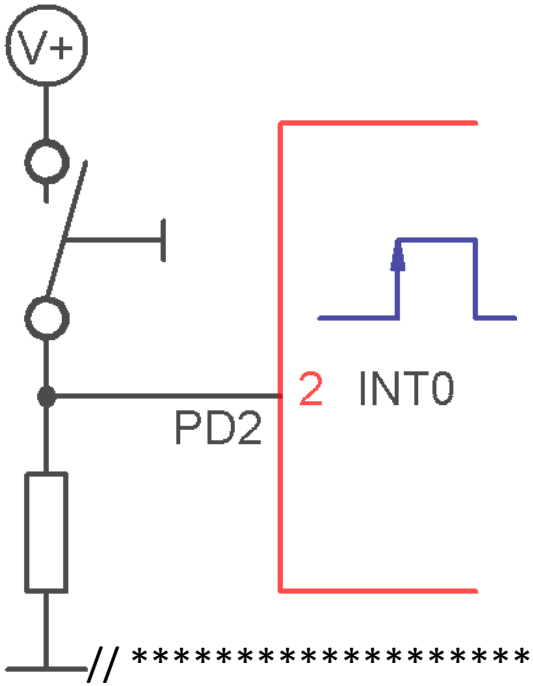
In Mainloop:
Flags prüfen→Event? Wenn Taster wieder frei→
mehrfach lesen, ~1mSec Abstand
Wenn 3..5 mSec ungedrückt → Interrupt wieder freigeben. →Flags löschen bzw. setzen.

Flags anlagen:

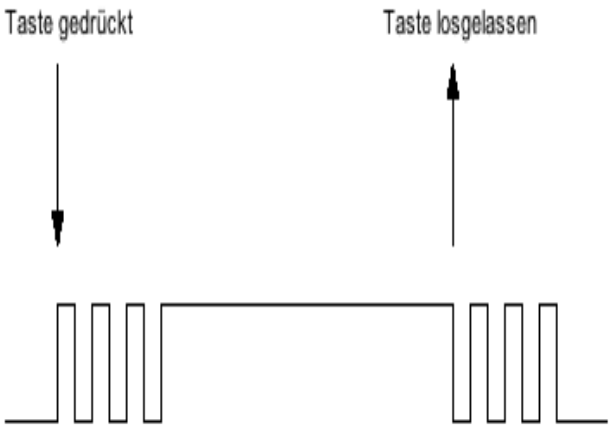
```
#define SWITCH_PRESSED_DETECT 0x01
#define SWITCH_PRESSED        0x02
#define SWITCH_RELEASE_DETECT 0x04
#define SWITCH_RELEASED       0x08
   …usw….. //uint8_t gu8Status;
```



Debouncing von Signalen von z.B. mechanischen Schaltern.



Merke: der Interrupt triggert nur das Debouncing




```
// ******************
ISR( INT0_vect )
// ******************
{
if( !(gu8Status & STATUS_INT0_EVENT) )                    // NICHT !!  Nur wenn Event Bit = 0
        {
        EIMSK &= ~( 1<< INT0 );          // Interrupt DEKATIVIERT SICH SELBST
        gu8Status &= ~STATUS_SWITCH_RELEASED;              //Clear
        gu8Status |= STATUS_INT0_EVENT;                    // Set Flag
        };
};                      //Dann wird in einem Timer oder mainloop() das Flag ausgewertet
                        → if( gu8Status & STATUS_INT0_EVENT )  ……
```

„seltsames“ Löschen eines Interrupt Ereignis Flags →

Wenn Sie sichergehen wollen, dass ein Interrupt nicht von einem unbeabsichtigten früheren Ereignis ausgelöst wird, muss das Interrupt-Ereignis-Flag gezielt gelöscht werden.

1.) ABER ! Laut Handbuch geht das durch **Schreiben einer „1“** auf das entsprechende Bit. „Read your fucking manual“.

2.) Dazu darf es **keinen Lesevorgang** des FLAG-Registers geben (sonst werden andere Int-Flags ebenfalls gelöscht.

Die Form EIFR **|=** (1<<INTF1); **IST FALSCH**

Richtig ist EIFR **=** (1<<INTF1); mit „=„ Zeichen

EIFR – External Interrupt Flag Register

Bit	7	6	5	4	3	2	1	0	
0x1C (0x3C)	–	–	–	–	–	–	INTF1	INTF0	EIFR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Das Problem tritt bei Ihrem ersten Versuch auf, weil Sie eine Lese-Änderungs-Schreiboperation verwenden, d.h. Sie löschen versehentlich alle anstehenden Interrupts. Sie hätten das gleiche Ziel erreichen können, indem Sie GIFR |= 0 oder GIFR = 0xFF schreiben;
Da anstehende Interrupts beim Schreiben einer Null auf diese Bitposition nicht gelöscht werden, müssen Sie Nullen in die Bits schreiben, die Sie unverändert lassen möchten.

Beachten Sie, dass dies nur und erhalten, wenn sie sich anders verhalten als die Interrup für Interrupt-Flag-Register gilt. Wenn es andere Statusbits gibt, müssen Sie diese entsprechend ausblenden Bits.

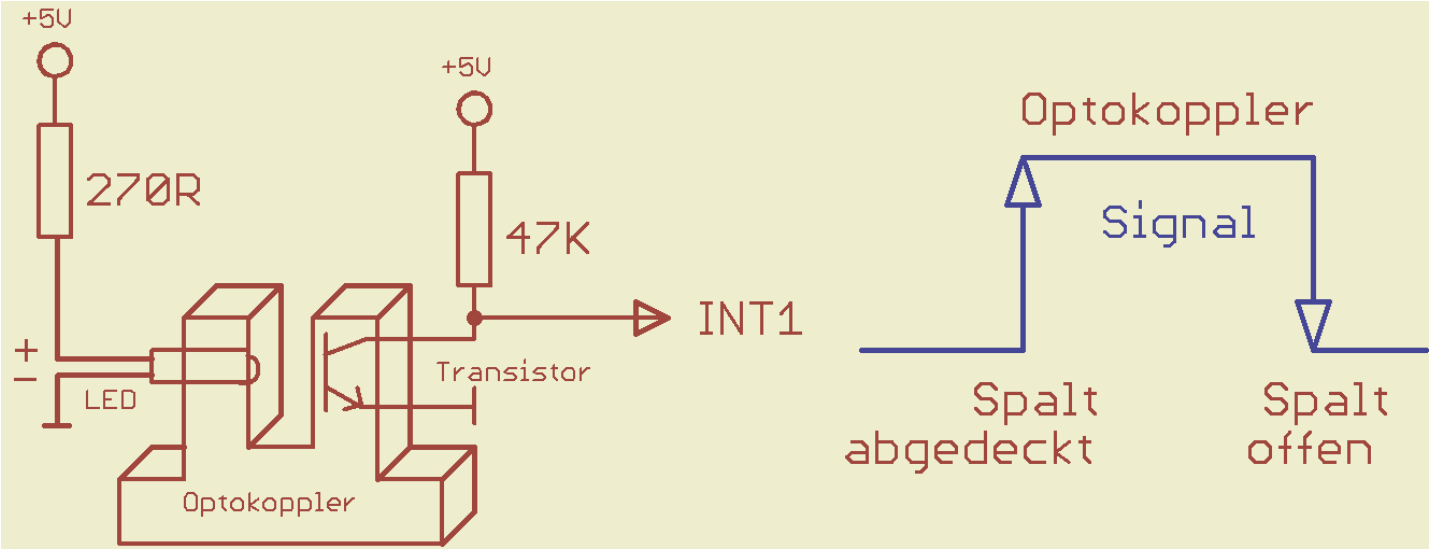
Bsp. Ein Timmer 2 Interrupt FLAG Register → **TIFR |= (1<< OCF2A);**

TIFR2 – Timer/Counter2 Interrupt Flag Register

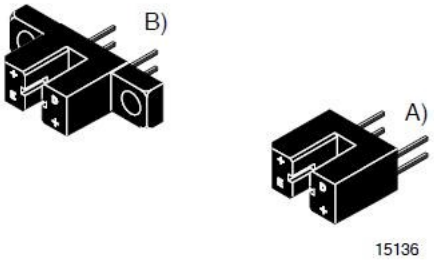
Bit	7	6	5	4	3	2	1	0	
0x17 (0x37)	–	–	–	–	–	OCF2B	OCF2A	TOV2	TIFR2
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

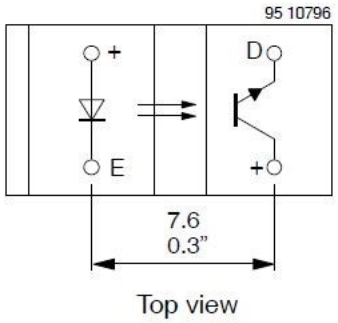
Interrupt Signal Quelle

Um ein konkretes Beispiel zu haben stellen wir uns vor, wir wollen an einer sich drehenden Achse wissen, wann ein festgelegter Punkt der Achse etc. an einer bestimmten Position ist. (ein Index !) Dazu ist ein Fähnchen auf der Achse angebracht, das somit beim Drehen durch einen Gabel-Optokoppler geht.
Beim „Eintauchen“ deckt dieser den Spalt ab.
Beim „Austauchen“ ist der Spalt wieder frei.
Folge: eine **steigende** Flanke im *Eintauchmoment* →Indexfall.

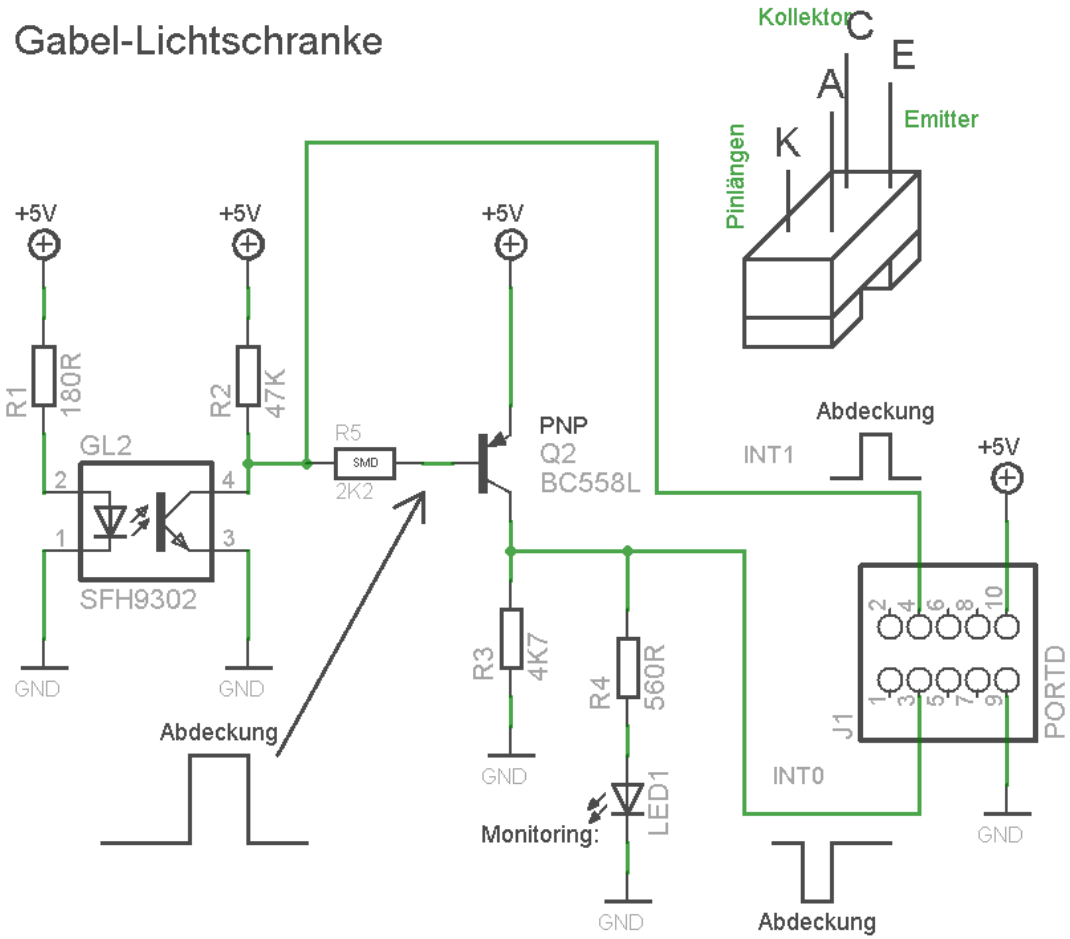


Gabellichtschranke mit Interrupt

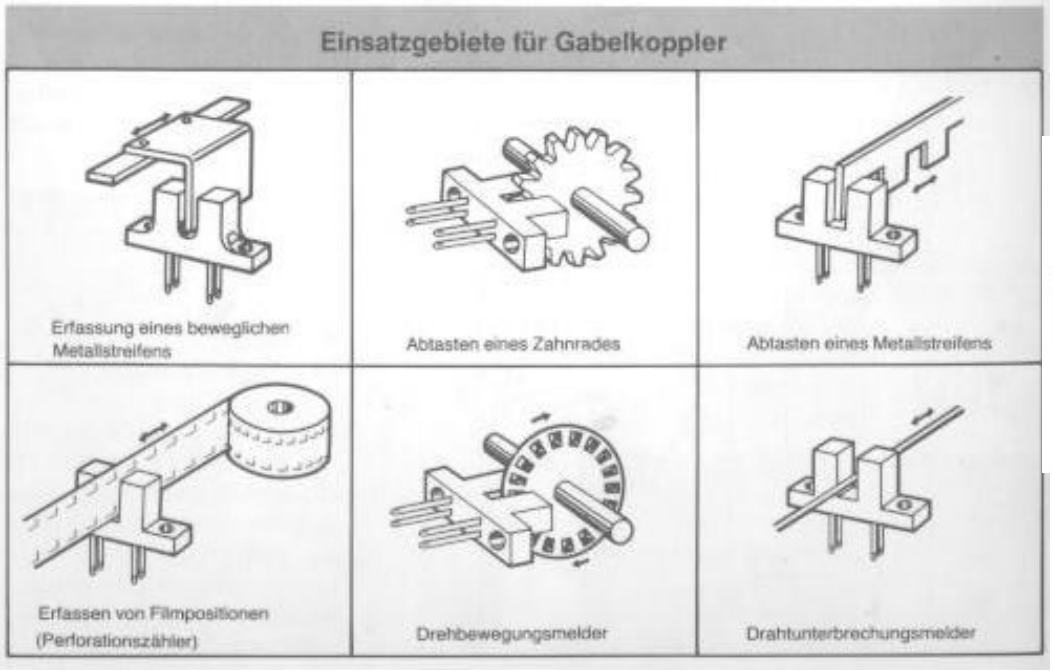


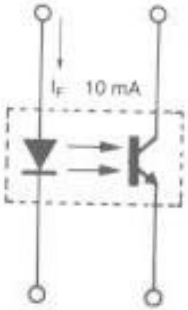


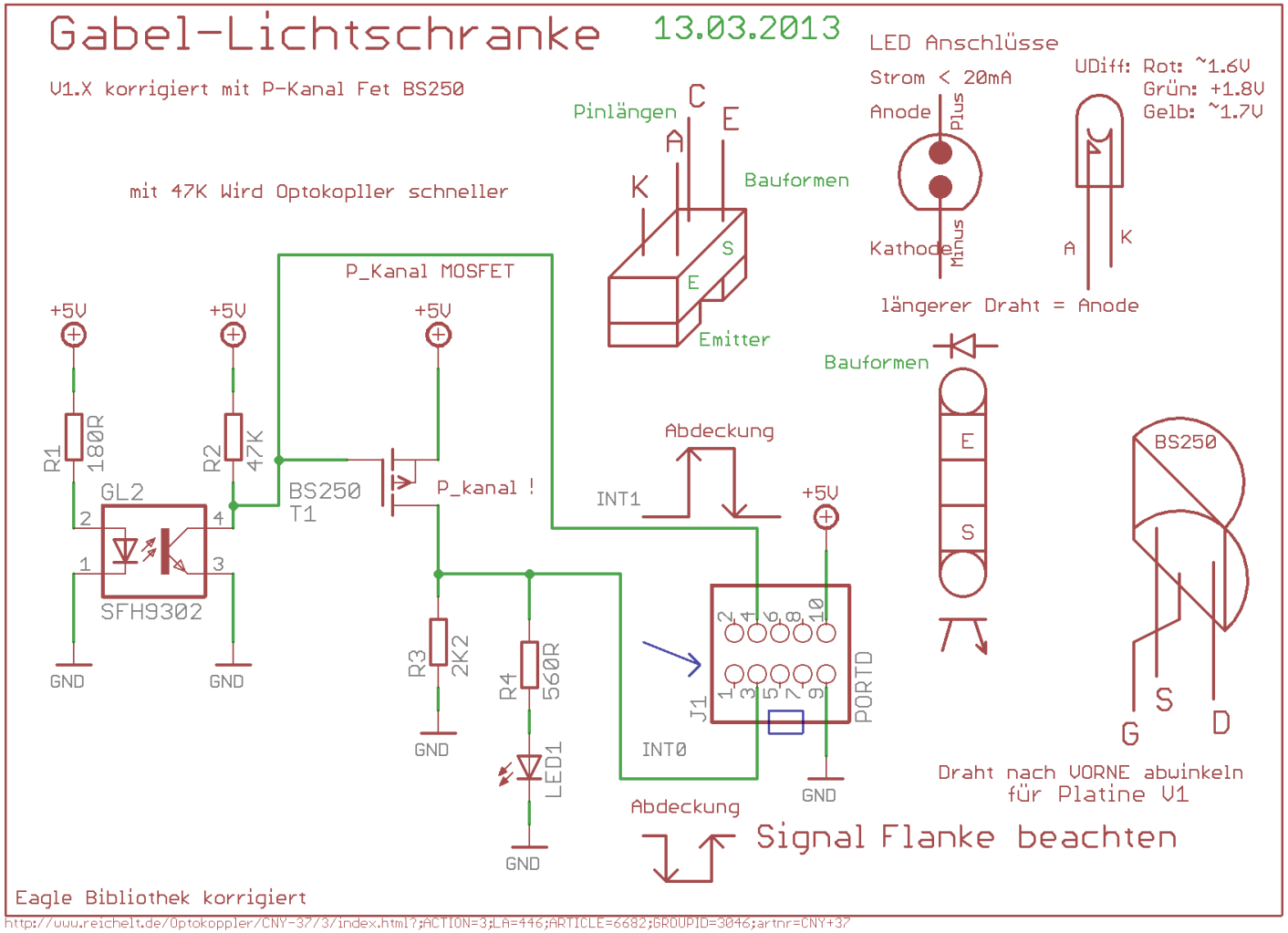
Beispiele, wie das Signal des Foto-Transistors ausgewertet werden kann. Der 2. Transistor dreht die Phase, Buffert das Signal und treibt die Monitor-LED.



Bsp.: Verschiedene Realisationen von Signalquellen z.B. mit Gabelkoppler und anderen Sensoren.







Es gibt zwei Methoden Eingaben zu realisieren. Einfaches =**PIN**x im Polling-Verfahren (= so schnell wie möglich lesen): Besser via **Interrupt**: → **INT0/1**. Diese Schaltung erlaubt die Abfrage des simplen momentanen Pegels, oder (zilegerichteter) steigender und fallender Flanke

Bitte versuchen Sie diesen Stil für sich selbst zu etablieren. IMMER.

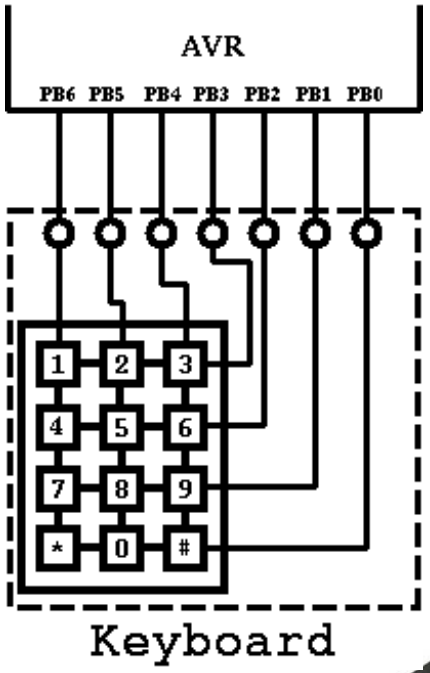
Vorbereitend definieren: Später sauber auslagen in die ***.h** Date

Bsp: (ALLES GROßBUCHSTABEN)

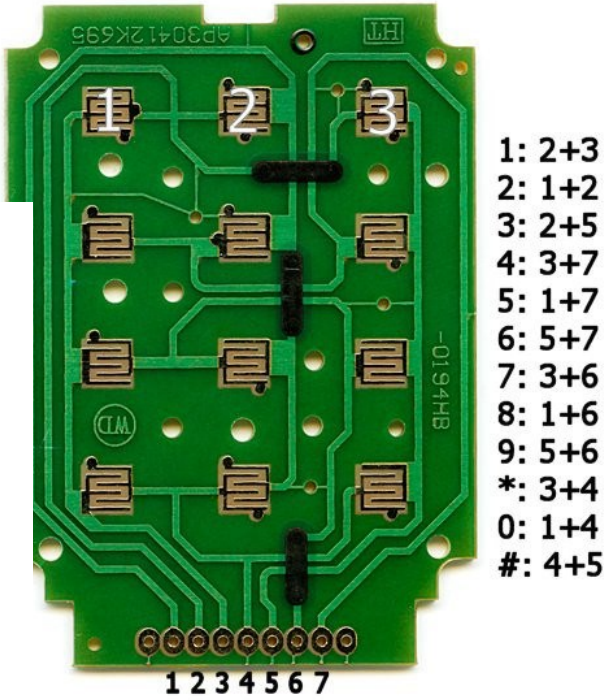
```
//Hardware Zuordnungen mit sprechenden Namen
#define LED_DDRX                DDRB
#define LED_PORT                PORTB
#define INTERNE_LED_BIT         (1<< PORTB5)
// MAKROS
#define INT_LED_ON()            (PORTB |= INTERNE_LED_BIT)
#define INT_LED_OFF()           (PORTB &= ~INTERNE_LED_BIT)

#define TICK_EVENT                      0x01
#define STATUS_HYSTERESE        0x02
//gu8Status;

#define TICK_TIME_1000MS        244
#define HYTERESE_TIME           (4 * TICK_TIME_1000MS)
// PIN auf Ausgang

#define LICHTSCHRANKEN_DDRX             DDRD
#define LICHTSCHRANKEN_PINX             PIND
#define GABLELLICHTSCHRANKEN_BIT        (1<<PORTD2)
#define GABLELLICHTSCHRANKEN_INIT()     (LICHTSCHRANKEN_DDRX &= ~LICHTSCHRANKEN_BIT)
#define GABELLSCHRANKE_READ_TRUE()      (LICHTSCHRANKEN_PINX & LICHTSCHRANKEN_BIT)
```

```
int main(void)                          //Bsp: für eine Abfrage nur mit PINX
{
GABELLICHTSCHRANKEN_INIT();
while(1)
        {
        if(  !(gu8Status & STATUS_HYSTERESE)  )
                {
                if( GABELLSCHRANKE_READ_TRUE() )
                        {
                        OCR2A = 15;
                        u32ZeitMerker = gu32_Tick;
                        gu8Status |= STATUS_HYSTERESE;
                        }
                else
                        {
                        OCR2A = 28;
                        u32ZeitMerker = gu32_Tick;
                        gu8Status |= STATUS_HYSTERESE;
                        };
                                }
        if(   (gu8Status & STATUS_HYSTERESE)        &&
              (gu32_Tick - u32ZeitMerker) > HYTERESE_TIME)     //1/2 Sec
           )
                {
                gu8Status &= ~STATUS_HYSTERESE;
                };
        };
```

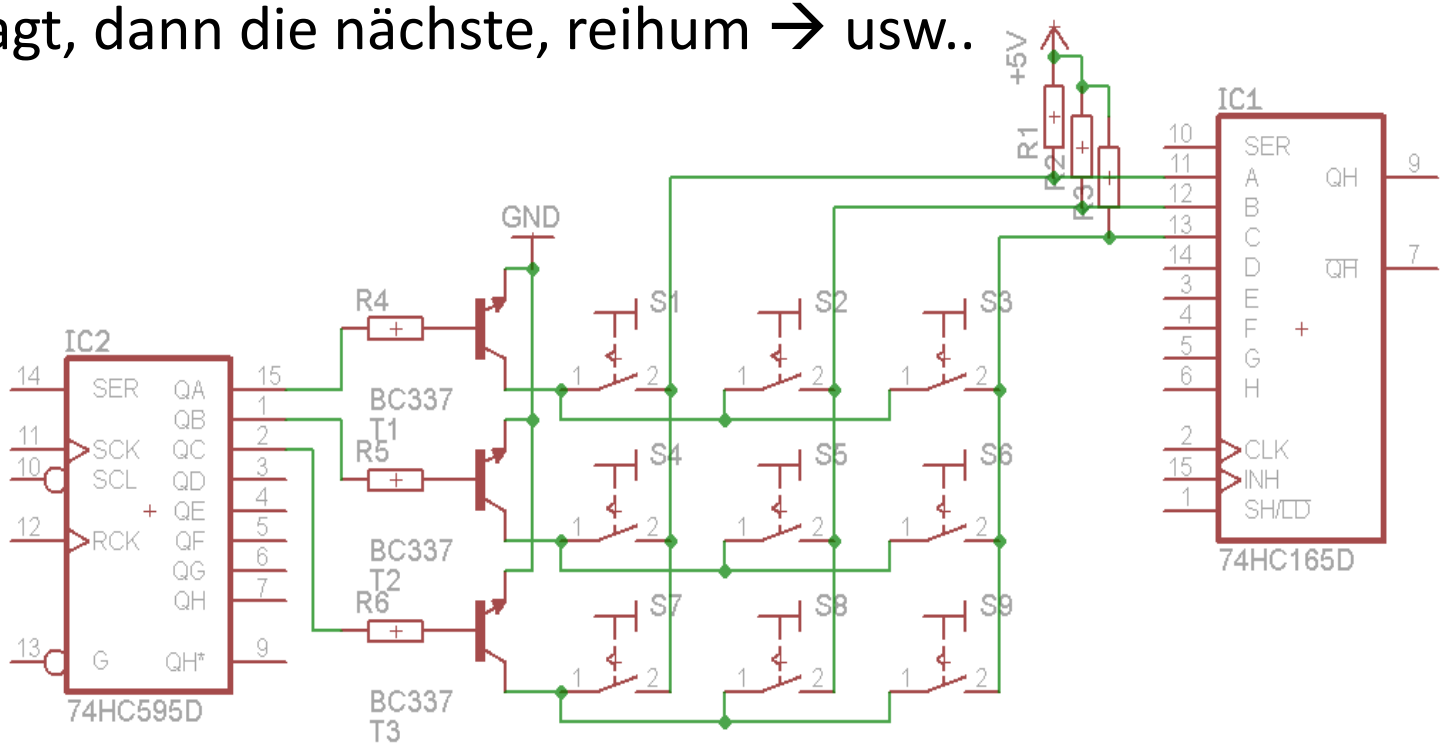
Tasten Matrix



Um Leitungen zu sparen werden Tasten in einer Matrix angeordnet. Diese werden in einem Multiplex-Verfahren reihen/kolumnenweise gescannt. Dies zudem sehr schnell.



Tastenmatrix Beispiel mit In-Out Erweiterungs-ICs. (später im Skript behandelt…)
Gezeigt wird das Prinzip: Nur eine Reihe wird LOW gelegt und abgefragt, dann die nächste, reihum → usw..



Externe Bibliotheken nutzen Bsp. → “uart.c“

```
/*****************************************************
Function: uart_init()
Purpose:  initialize UART and set baudrate
Input:    baudrate using macro UART_BAUD_SELECT()
Returns:  none
*****************************************************
void uart_init(unsigned int baudrate)
{
    UART_TxHead = 0;
    UART_TxTail = 0;
    UART_RxHead = 0;
    UART_RxTail = 0;

#if defined( AT90_UART )
    /* set baud rate */
    UBRR = (unsigned char)baudrate;

    /* enable UART receiver and transmmitter and receive
    UART0_CONTROL = _BV(RXCIE)|_BV(RXEN)|_BV(TXEN);

#elif defined (ATMEGA_USART)
    /* Set baud rate */
    if ( baudrate & 0x8000 )
    {
        UART0_STATUS = (1<<U2X);  //Enable 2x speed
        baudrate &= ~0x8000;
    }
    UBRRH = (unsigned char)(baudrate>>8);
    UBRRL = (unsigned char) baudrate;

    /* Enable USART receiver and transmitter and receive
    UART0_CONTROL = _BV(RXCIE)|(1<<RXEN)|(1<<TXEN);

    /* Set frame format: asynchronous, 8data, no parity,
    #ifdef URSEL
    UCSRC = (1<<URSEL)|(3<<UCSZ0);
    #else
    UCSRC = (3<<UCSZ0);
    #endif
```

„main.c“

```
#include “main.h”
#include “Befehle.h”
#include “max7219.h”
#include “debug.h“
#include “uart.h”
```

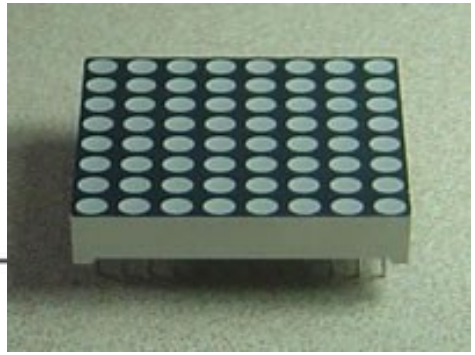
“makefile”

SRC= …. …. uart.c

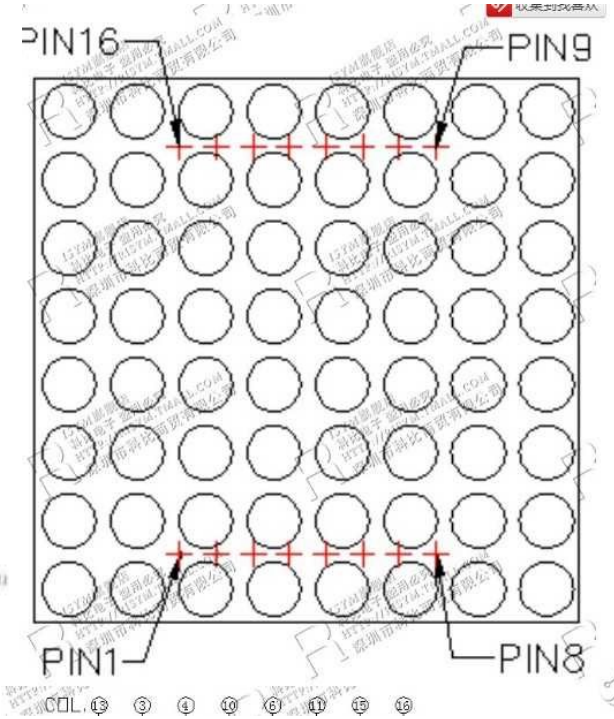
Anwendung in „main.c“:

uart_init(UART_BAUD_SELECT_DOUBLE_SPEED(UART_BAUD_RATE, F_CPU));

LED-MATRIX 8x8



Aufdruck
HL-M1588BR
ist „oben“



Typ M1588BR siehe auch
http://www.mikrocontroller.net/articles/LED-Matrix
und http://arduino.cc/en/Tutorial/RowColumnScanning

MAX7219

LED MATRIX 8x8

Steuerung

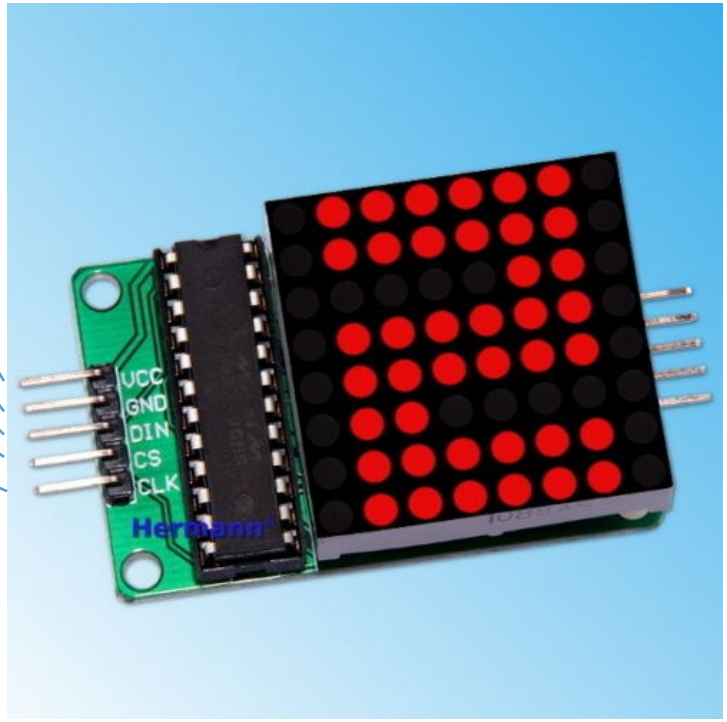
VCC

GND

DIN

CS

CLK

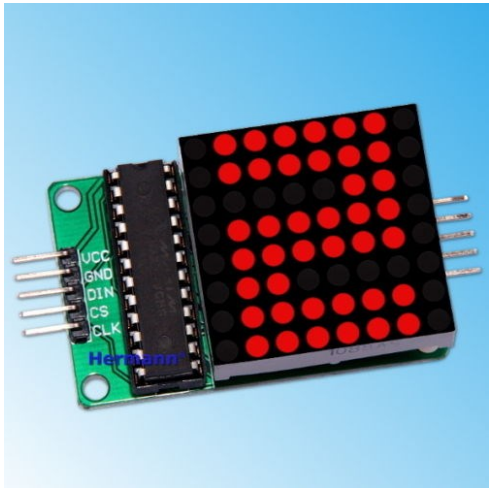


Specifications:
A single module can drive an 8 * 8 common cathode lattice
Operating voltage: **5V**
Size: 5 x 3.2 x 1.5 cm (L x W x H)
LED color: red
With four screws hole, aperture 3mm
With input and output interfaces, supports multiple modules cascade

Wiring instructions:
1. The module left side as an input port and the output port on the right
2. To control a single module, just connect the input port to the CPU
3. When cascading multiple modules, the input port of the first module connect to the CPU, the output port connect to the input port of the second one, and so on.

MAX7219 Matrix mit externer Bibliothek

VCC	**VCC**
GND	**GND**
PD2	**DIN**
PD3	**CS**
PD4	**CLK**



max7219.c
max7219.h
Belegungen **frei**, aber sinnvoll im *.h definieren:

```
PORTD
#define MAX7219_DIN          PD2
#define MAX7219_LOAD_CS      PD3
#define MAX7219_CLK          PD4
```

max7219.h
max7219.c

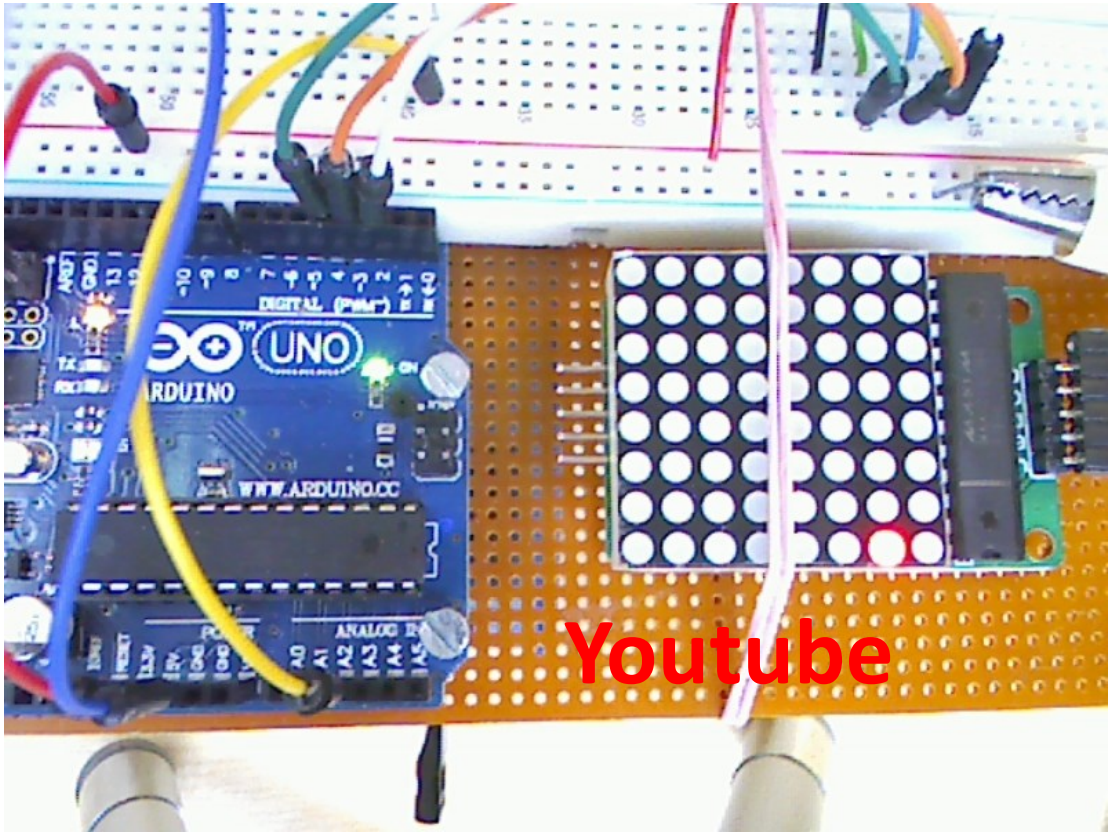
```
extern void MAX7219_Write( unsigned char u8Register, unsigned char u8Data);
extern void MAX7219_Init(void);
extern void MAX7219_Clear (void);
extern void MAX7219_SetBrightness (char brightness)
```

Ab jetzt kann man die 8x8 LED Matrix auch als Byteanzeige gebrauchen.
Sogar **8 Bytes**, wie Status-Flags, div Werte etc. können angezeigt werden:

MAX7219_Write([1..8] , Data);

Youtube: http://youtu.be/7kharpUQNNo
und https://www.youtube.com/watch?v=YIcebSrfGek

MAX7219_Write(**1**, (1<< (u8NN++) % 8)); //Lauflicht



```
/ **********************
void MAX7219_Init(void)
// **********************
{
//INIT_MAX7219_DIR(); //MAKRO
MAX7219_DIN_DIRREG |= (1<< MAX7219_DIN);
MAX7219_LOAD_CS_DIRREG |= (1<< MAX7219_LOAD_CS);
MAX7219_CLK_DIRREG |= (1<< MAX7219_CLK);

MAX7219_Write(REG_SCAN_LIMIT, 7); // set up to scan all eight digits
MAX7219_Write(REG_DECODE, 0x00); // set to "no decode" for all digits
MAX7219_Write(REG_SHUTDOWN, 1); //ShutdownStop
MAX7219_Write(REG_DISPLAY_TEST, 0); //DisplayTestStop
MAX7219_Clear();
MAX7219_SetBrightness(5); // INTENSITY_MAX 0x0F
//MAX7219(
};
```

Anwendung in *.c: *Beispiel:*

u8CC = pgm_read_byte_near(font8x8 + (65*8) + (u8Mod * 8) + u8NN);

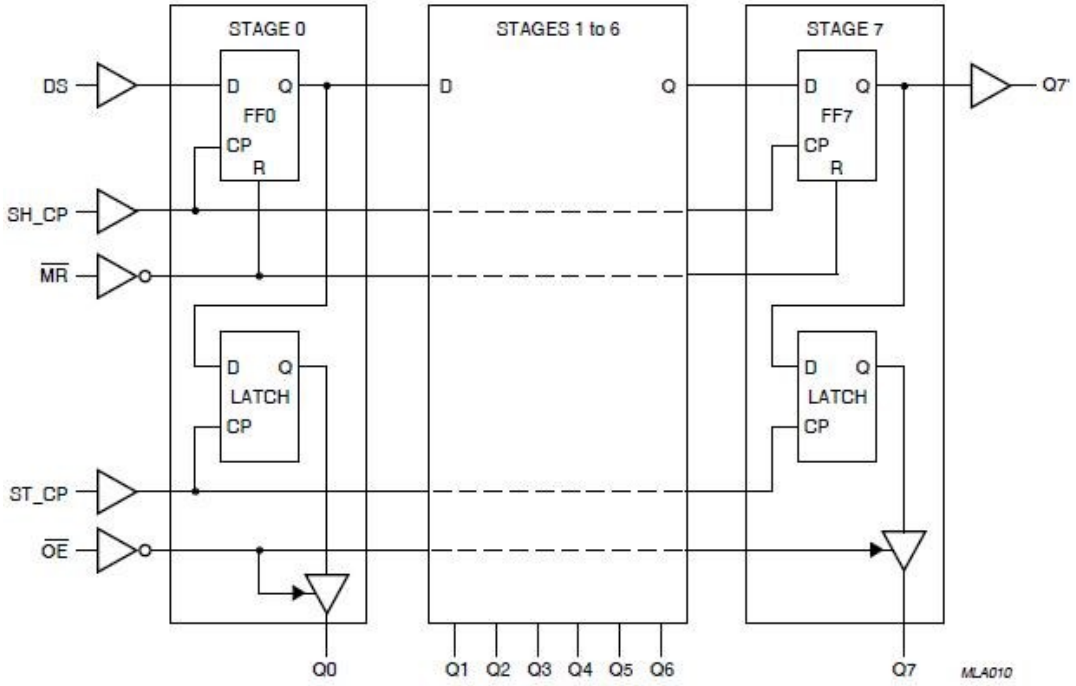
MAX7219_Write(u8NN+1, u8CC);

→ **MAX7219_Write([1..8] , Data[0..255]);**

……..

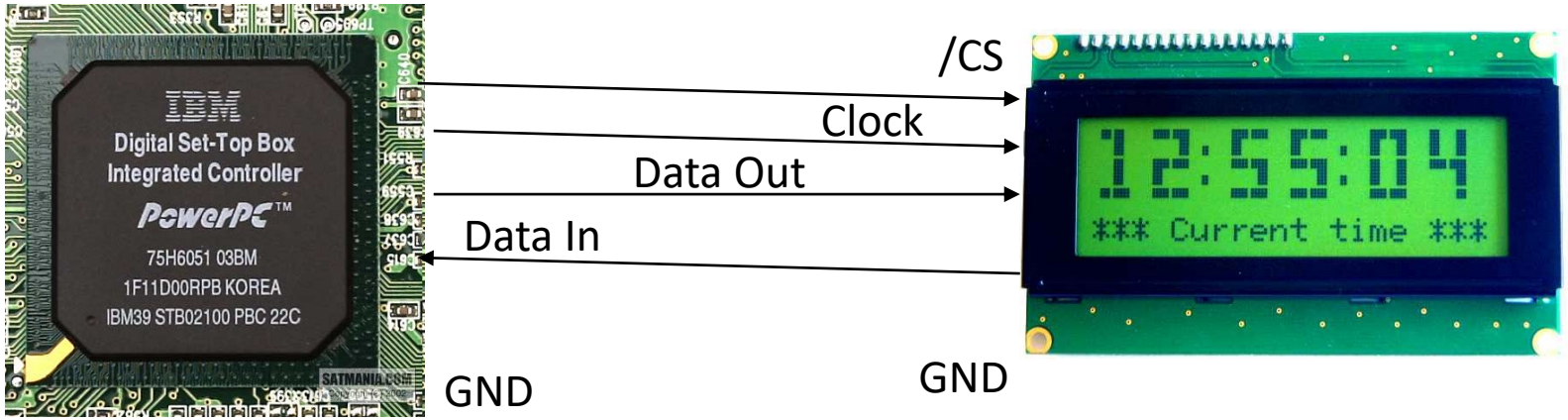
Das führt zu →Protokollgerechte Signalerzeugung

Da Mikrocontroller beliebige Signale ausgeben können, kann man dies benutzen um externe ICs anzusteuern.
So kann **fast jedes IC/Gerät** mit den ihm eigenen Eigenschaften an den µKontroller angebunden werden.
Output/Input-Porterweiterungen, 3/4-Leiter Bus, 2-Leiter-I2C-Bus Bausteine, LCD-Displays, Drehgeber, Tastenmatrix usw. Sensoren…….also ICs aller Art.



Externer OUT-Baustein.
Shift-Out mit 74HC595

SPI = Serial Port Interface
Synchron-Bus



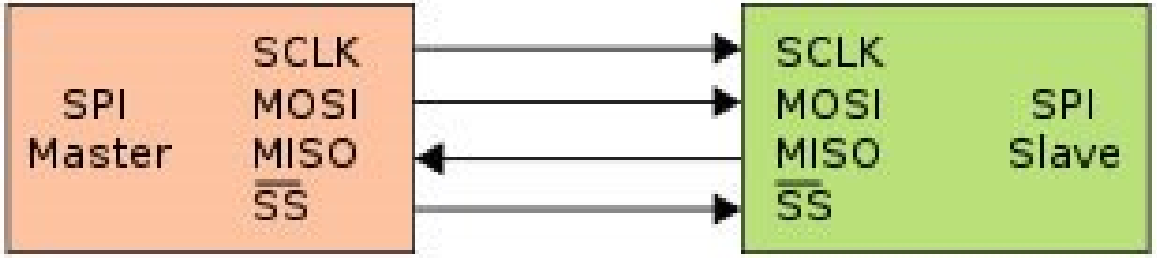
- Einfach zu nutzen und zu verstehen
- Serielles Bus Protokoll
- Meist ein 3 bis 4-Leitungs Protokoll, evtl. auch nur 3 Leiter
  - Clock
  - Data In
  - Data Out
  - Chip Select (/CS) oder Enable oder…….

  - Ach ja, da wäre ja noch die alles verbindende GND =„Ground“.

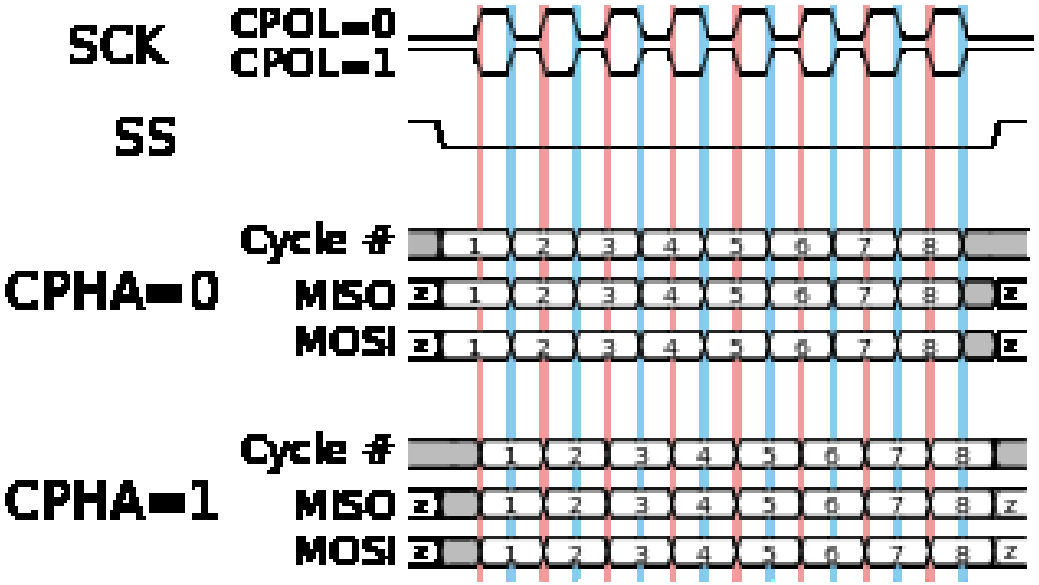
Was ist SPI ?

Serial Peripheral Interface (SPI)

https://de.wikipedia.org/wiki/Serial_Peripheral_Interface



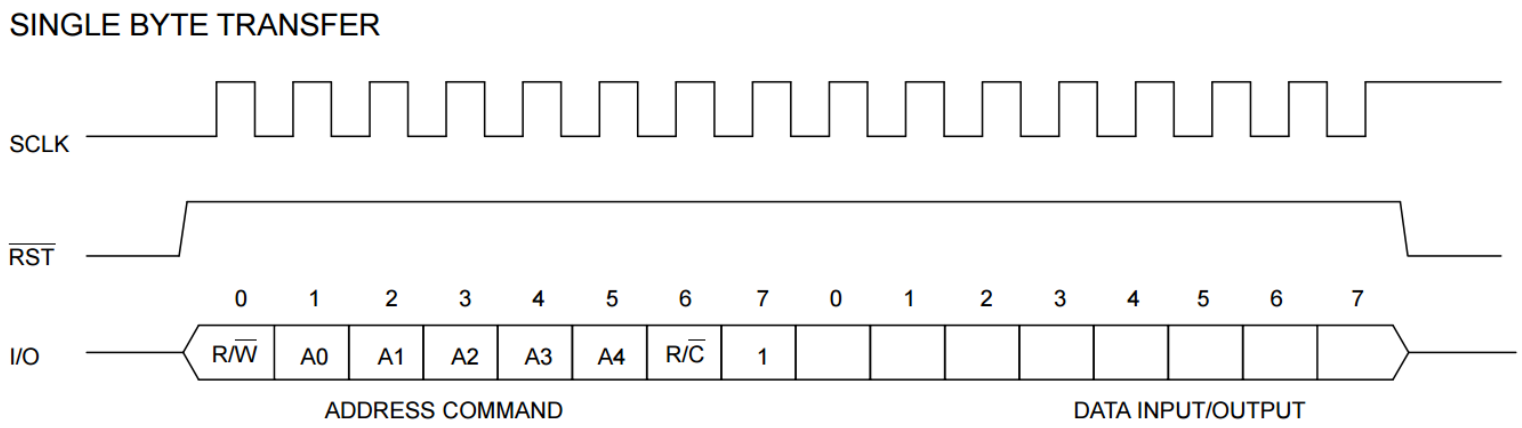
SCLK = Serial Clock
MOSI = Master Out, Slave In
MISO = Master In, Slave Out
/SS = Slave Select,
oder Chip Select /CS



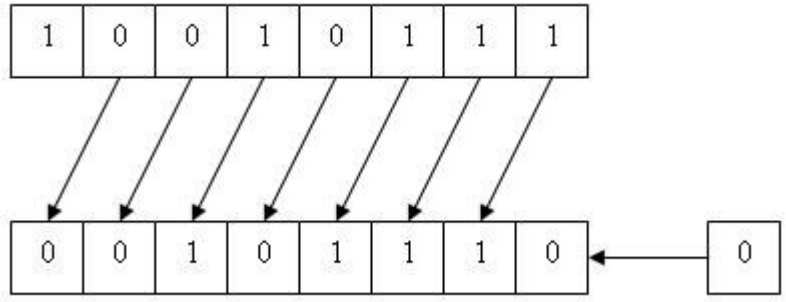
Externe ICs Ansteuern
Kontrolle über Signale

Wir brauchen eine Methode um einzelne Signale zeitlich synchron präzise anzusteuern. Oft ist das mit eine 2, 3, 4 Draht Bus realisiert. SCK= CLOCK, Data_IN, DATA_Out, plus Steuerleitungen wie Enable oder \Reset. Je nach Handbuch

SPI → Serial Port Interface



Schieben von Daten, allgemein…



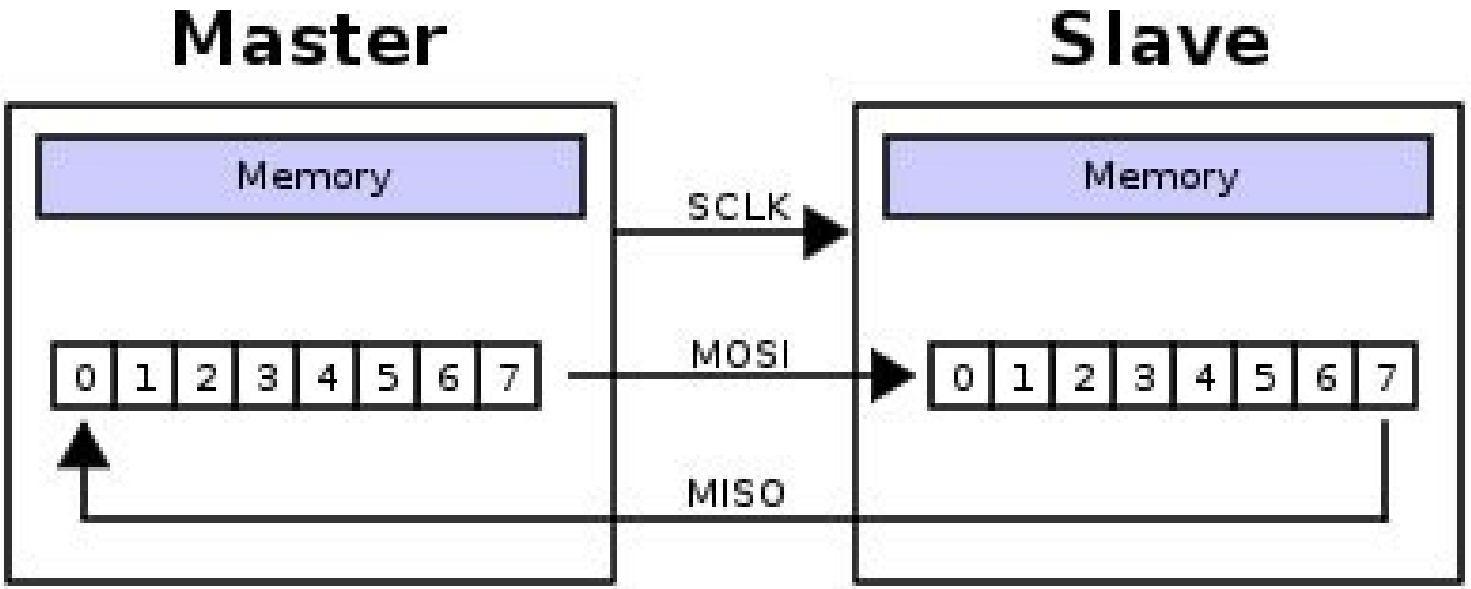
Bits in ein Register **shiften** ist etwa so, wie Eier in einer Schachtel **mit jedem Takt** um eins nach links weiter schieben.

Kommt **ein neues Ei = 1** herein, rutscht dieses mit dem nächsten Takt in das 1. Fach: → „1“

Kommt **KEIN Ei** herein , rutscht ein **Loch=0** in das 1. Fach. → „0“

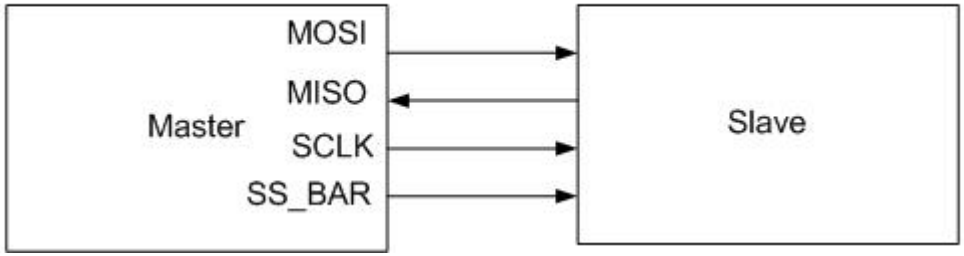
Alle anderen Eier werden immer **mit jedem Takt** eins weiter nach links sortiert.

Schiebe = Shift Protokoll



- Master gibt Takt/Clock vor. → CLK oder SCK
- Master schiebt mit jedem Clock ein Bit als Daten zum Slave → MOSI
- Zugleich schiebt der Slave Daten zum Master. → MISO

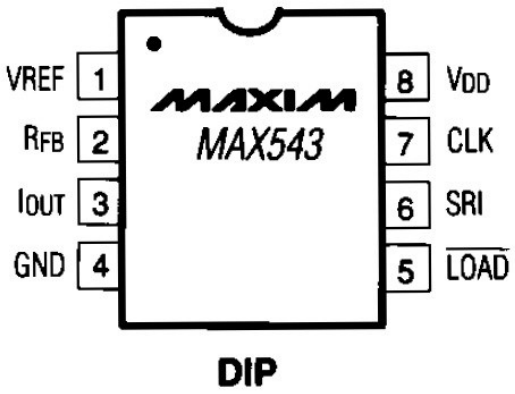
Synchronbus mit integrierten Shift Registern

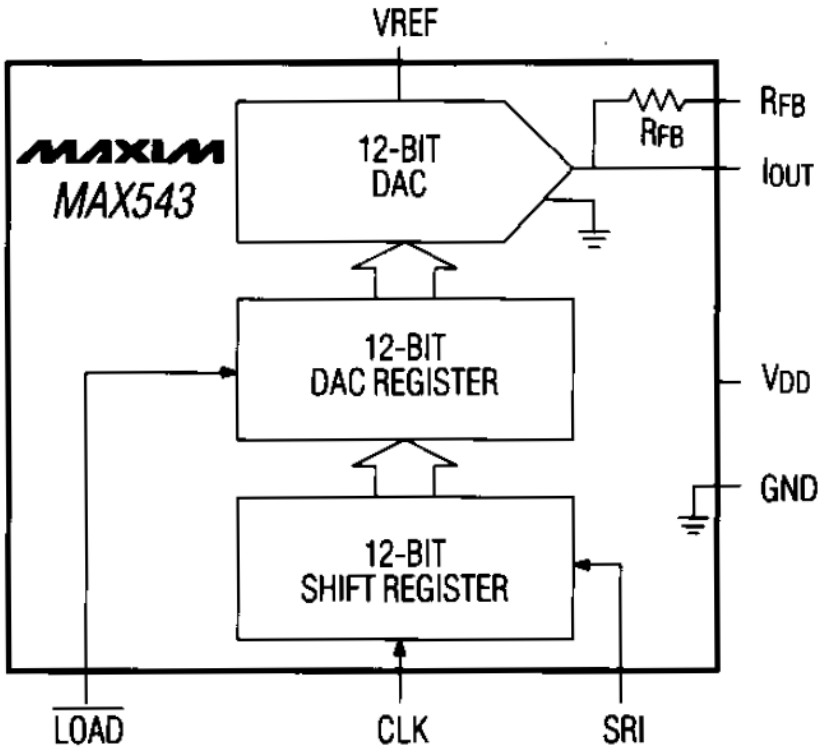


- Wires:
  - Master Out Slave In (MOSI)
  - Master In Slave Out (MISO)
  - System Clock (SCLK)
  - Slave Select 1…N
- Master Set Slave Select low
- Master Generates Clock
- Shift registers shift in and out data

12 BIT DAC mit MAX-543

Netterweise gibt es ICs mit integriertem R2R Netzwerk und als 12 DAC. Diese sind jedoch laserabgeglichen und **richtig teuer ~20€/Chip**





Innen:
Ein **12 Bit Shift Register** um die Daten hineinzuschieben.
Dann wird alles auf einmal „übernommen“
12DAC-Regsiter

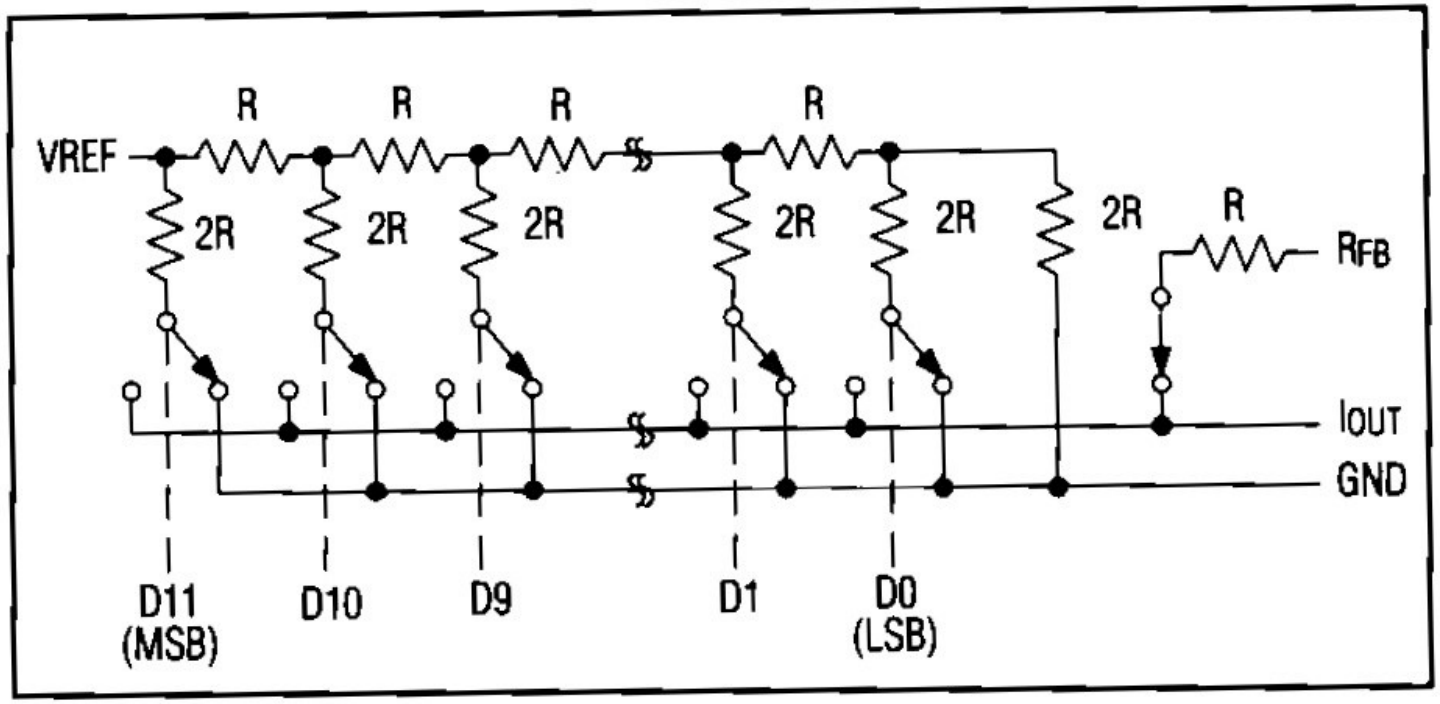
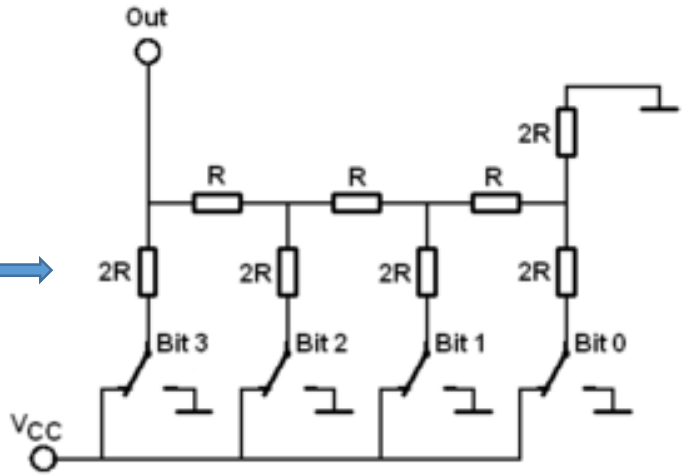


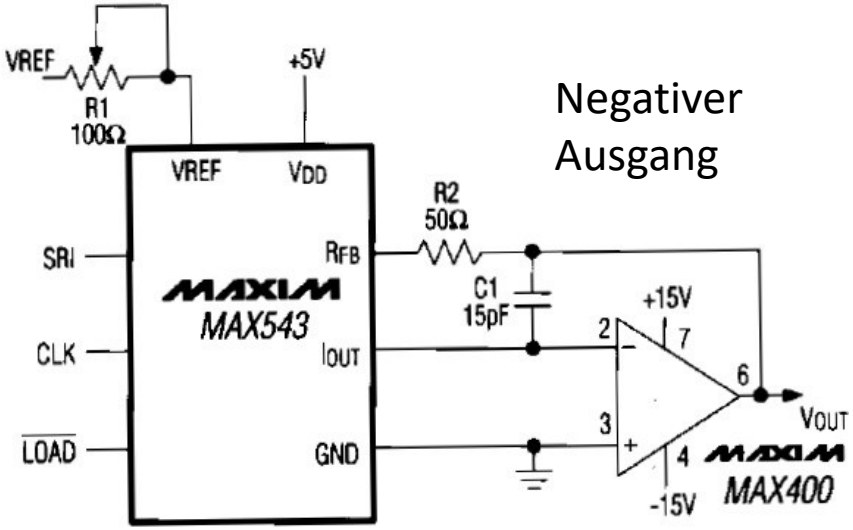
Figure 1. MAX543 Simplified Circuit

Noch braucht es etwas externe Beschaltung, damit das R2R Innenleben dem Prinzip entspricht.

Ein **RFB** (feedback) Eingang und kein Uout, sondern „**IOUT**“ steht da…..



Negativer Ausgang



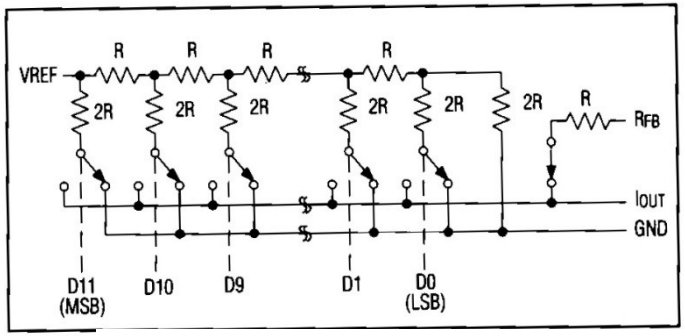
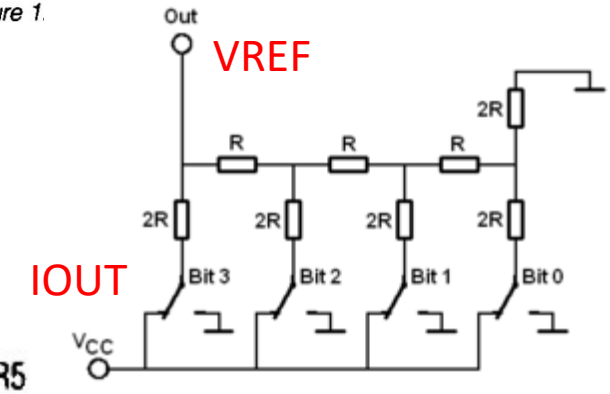
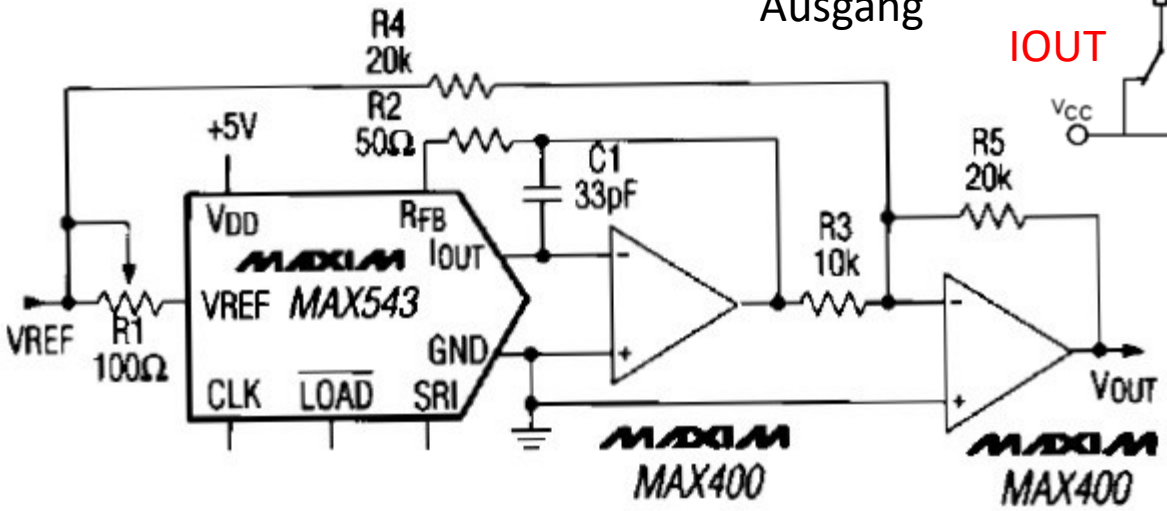


Figure 1.

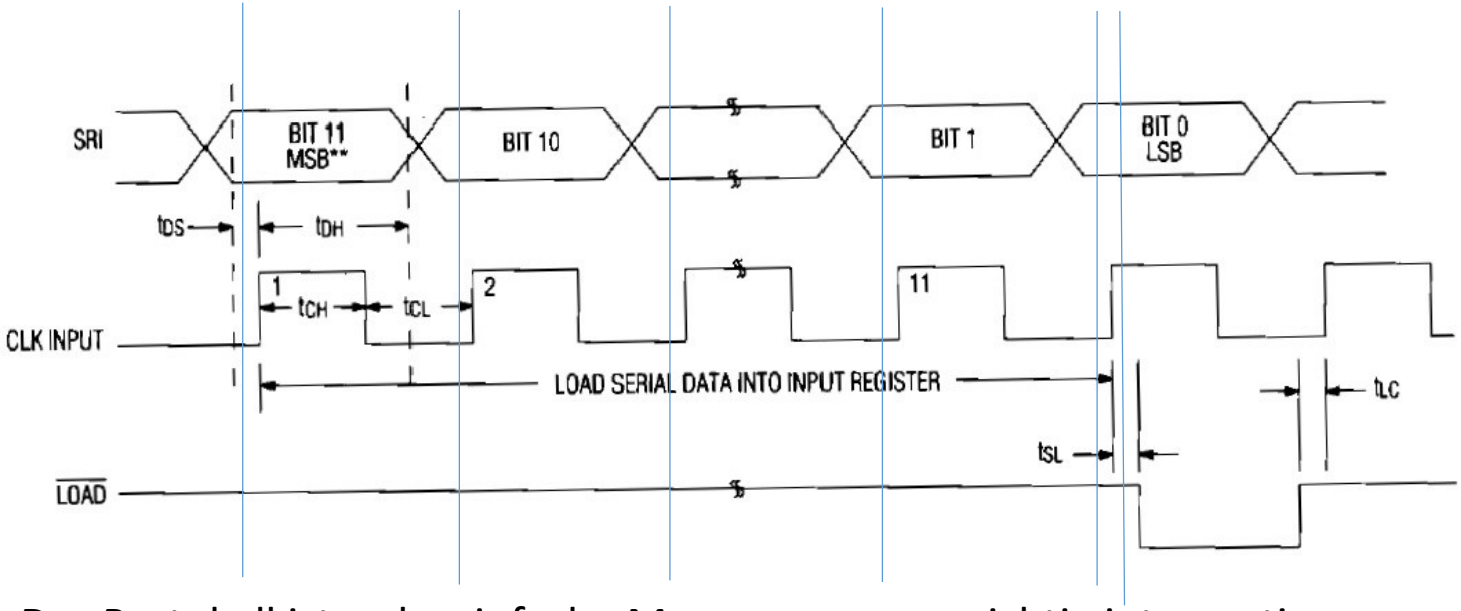


Positiver Ausgang



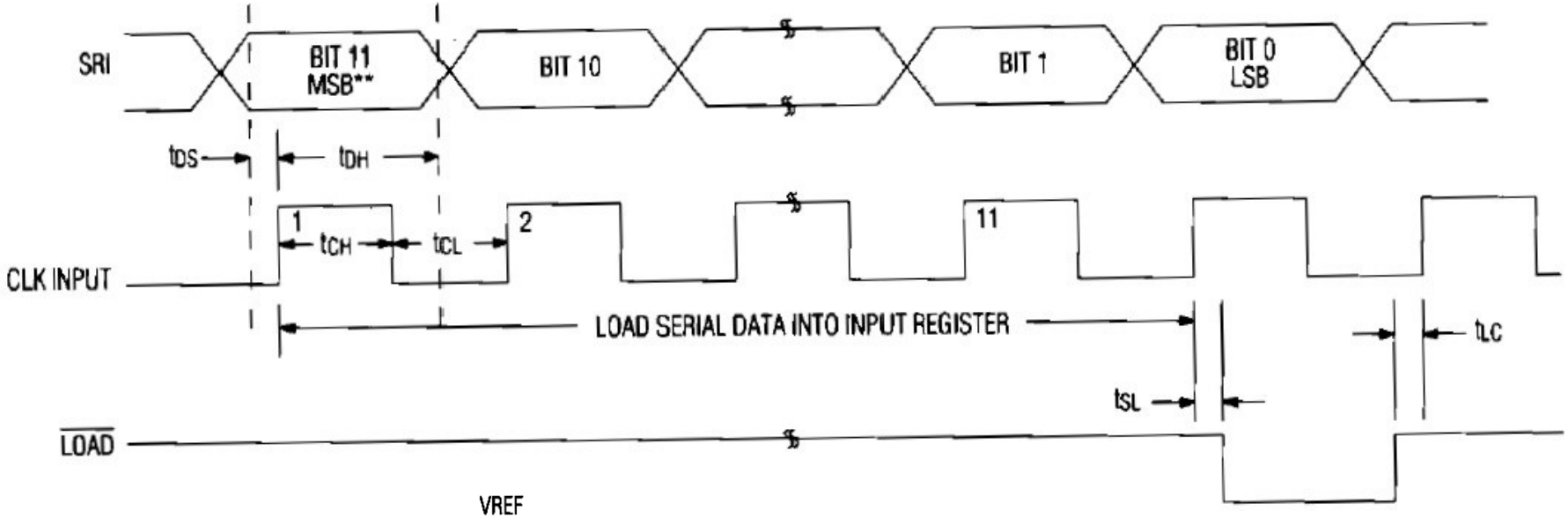
Der OP ist ein I-U Wandler. Etwas kompliziert… Ist also strombasiert, nicht spannungsbasiert. Aber einfach so bauen und es geht…

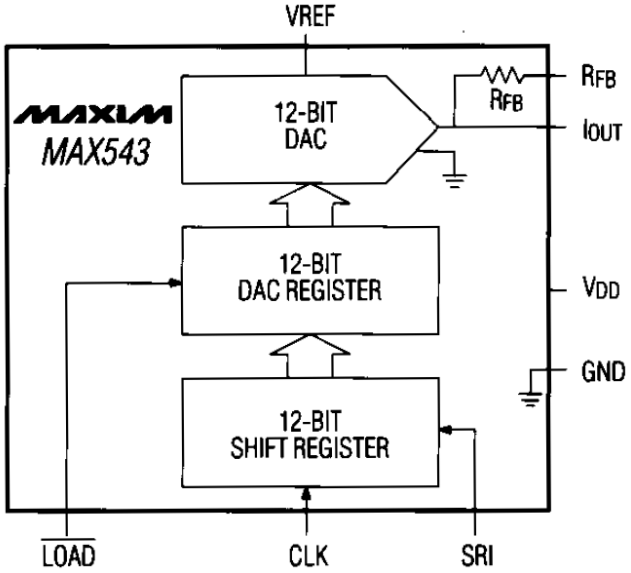
Programmierung des MAX543 12Bit DAC

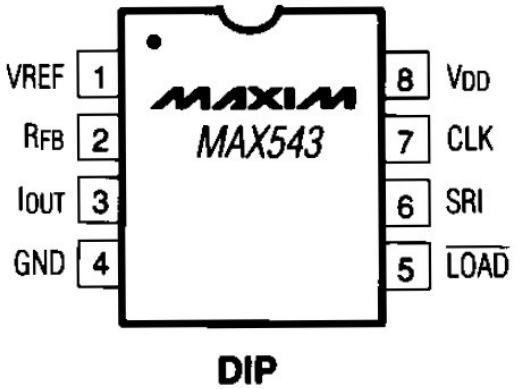


Das Protokoll ist sehr einfach. Man muss es nur richtig interpretieren:
12 Bit sind zu übertragen. →uint16_t u16Val;
Beginnend mit **u16Val Bit N=11** → an den **SRI** (PIN) und **/LOAD** = 1
Do{
if(**u16Val & (1 << N--)**) → wenn **!= 0 SIR=1** setzen, sonst **SRI=0**
CLOCK Pin (**rising edge**) „Hier wird der Wert übernommen!“ → **falling edge**
Nächstes Value Bit diesmal → N=N-1 //N= Bit 10..9..8..7…..0
}while (N)
/Load = 0
/Load = 1 //und nach jeder Bitsetzaktion _delay_us(1);

Sammeln der relevanten Informationen. Pin Belegungen und Zyklen Ablauf.

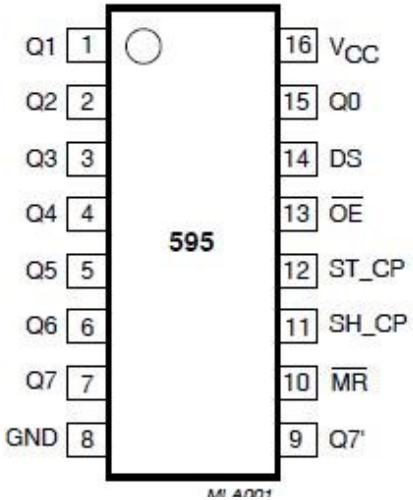






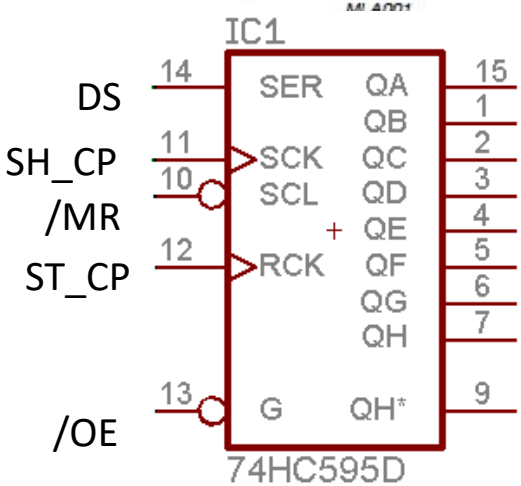
TTL Shift Register **74-595** als **Output Erweiterung**

Ein gutes einfaches Beispiel externer Hardware ist das **74-595** TTL Shift Output Register. …und gleich ein schönes Beispiel, wo unterschiedliche Pin Namen trotz gleicher Bedeutung vorkommen.



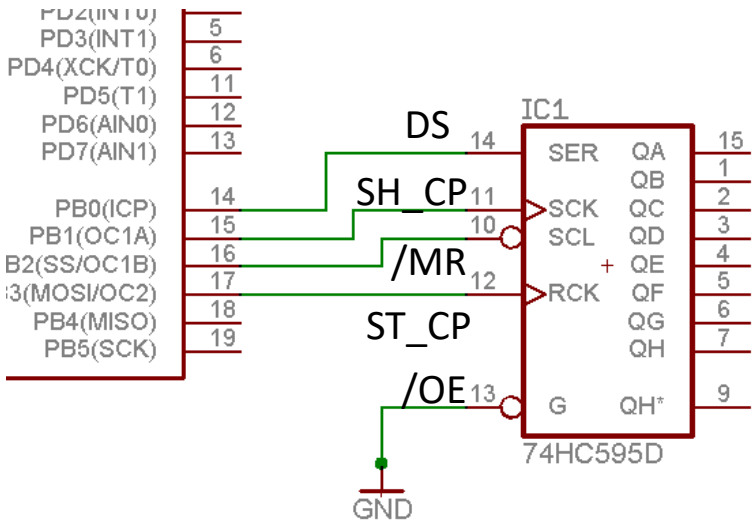
Auszug der Funktionstabelle:

DS v SER = Serial Data Input
ST_CP v RCK = Storage Register Clock Input
/MR v SCL = Master Reset
SH_CP v SCK = Shift Clk Input
/OE v G = Output Enable



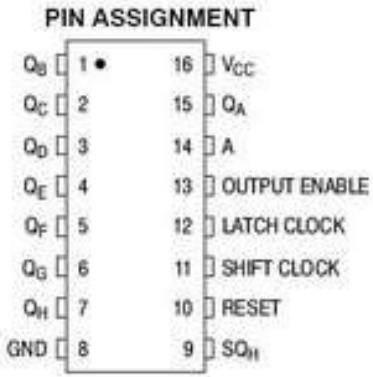
SH_CP & /MR=L EMPTY Shiftreg To Storagereg
SH_CP & /MR=L Ausgang 0 Stellen
SH_CP & /MR=H Data in Shiftreg
ST_CP & /MR=H Shiftreg to Storagereg
ST_CP & ^ ST_CP Pass Throug Shiftreg to Storagereg

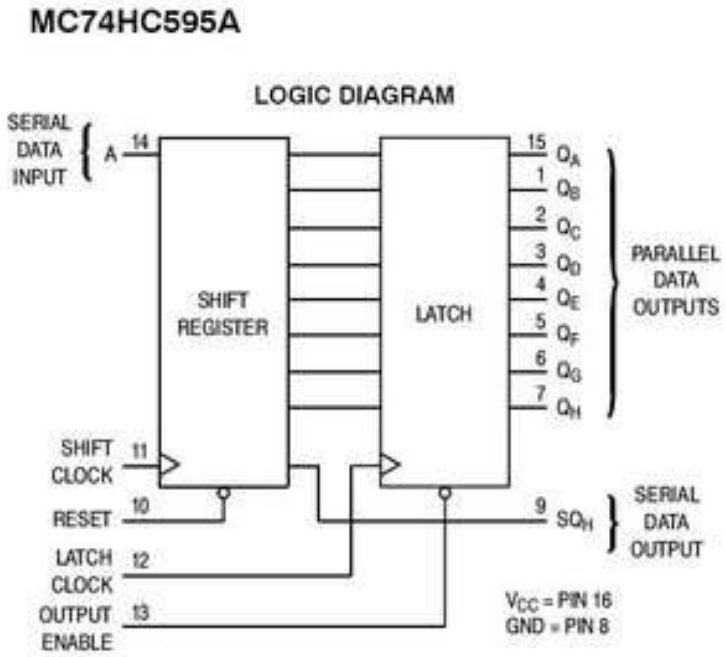
TI-Namensgebung → sn74hc595.pdf

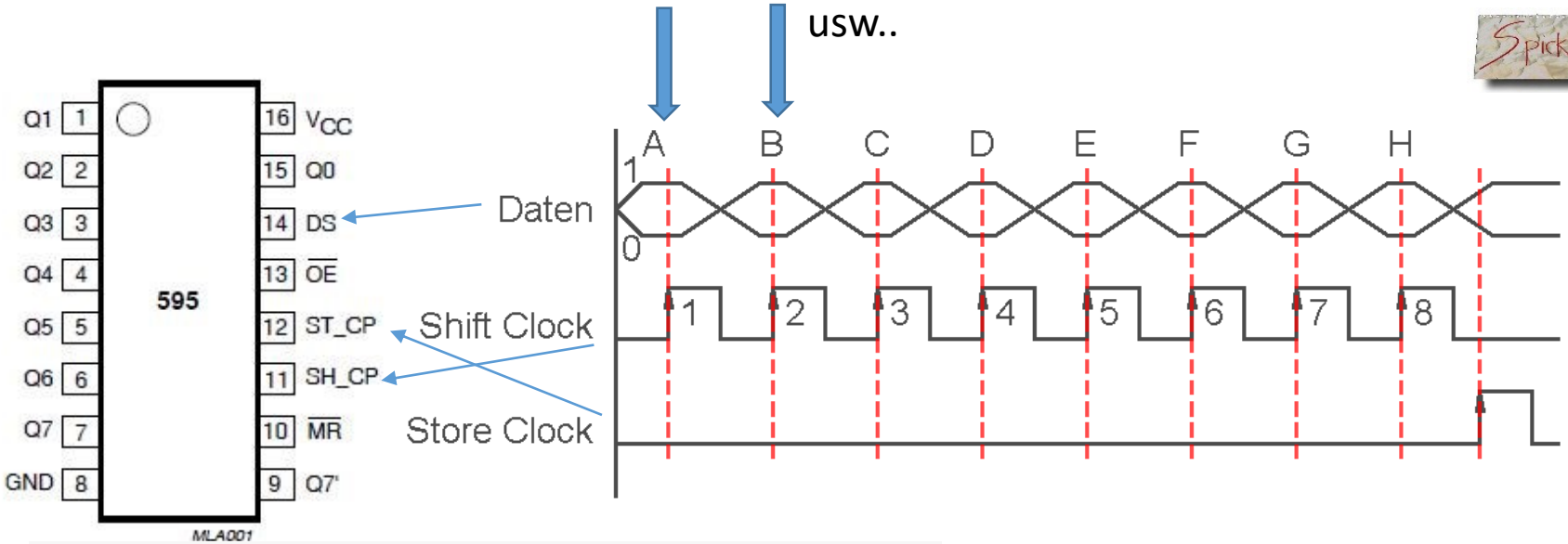


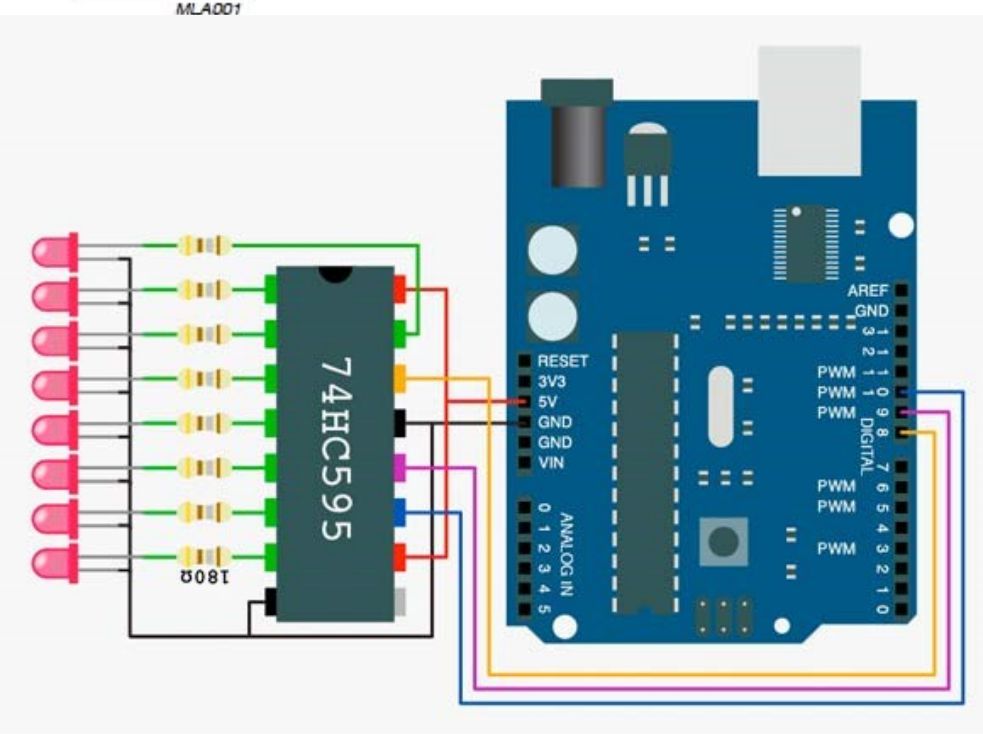
Durch einfache Anbindung an den µC mittels einiger Steuerleitungen kann nun ein externer Chip bedient werden. Nun sollten wir einfach etwas über dessen Innenleben kennenlernen → Manual.pdf

1.) zuerst werden die **Daten 0..7 pulsweise** (CLOCK) in ein Shift-register eingespielt.
2. Wenn alle Daten im Shiftregister sind, werden diese mit einem STORE Signal in den Output übernommen. Ein Überlaufbit kann in ein nachfolgendes IC geschoben werden.



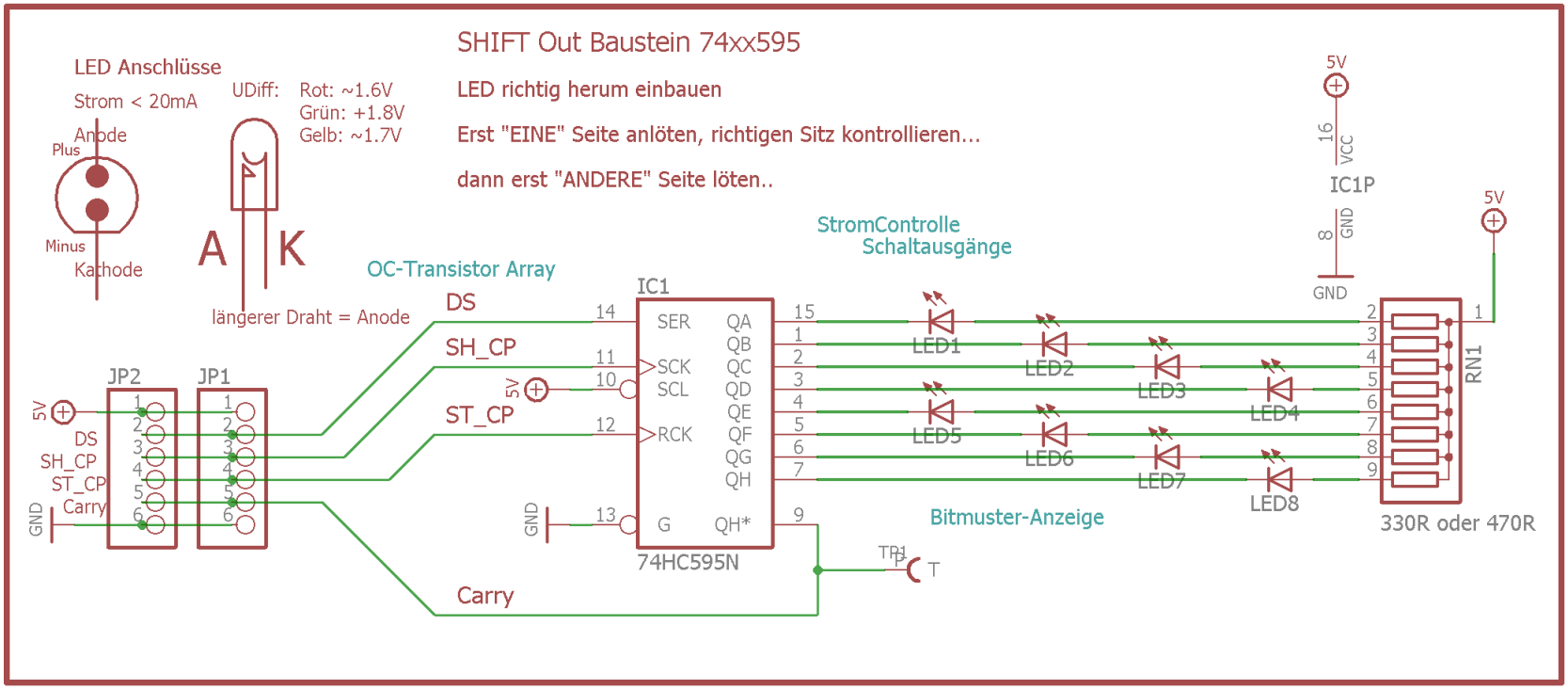


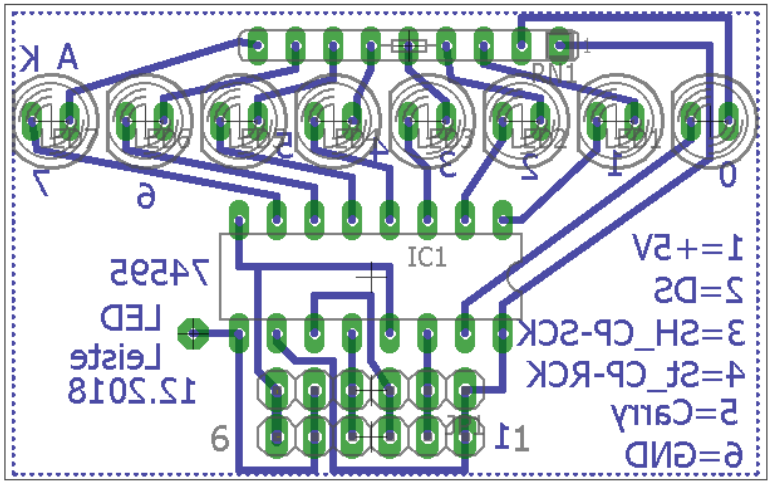





```
#define 74595_DDR      DDRB
#define 74595_PORT     PORTB

#define DS_74595       (1<<PB0)
#define ST_CP_74595    (1<<PB1)
#define SH_CP_74595    (1<<PB2)
```

/OE = 0
/MR = 1
Q7 → DS nächster Baustein
wenn da



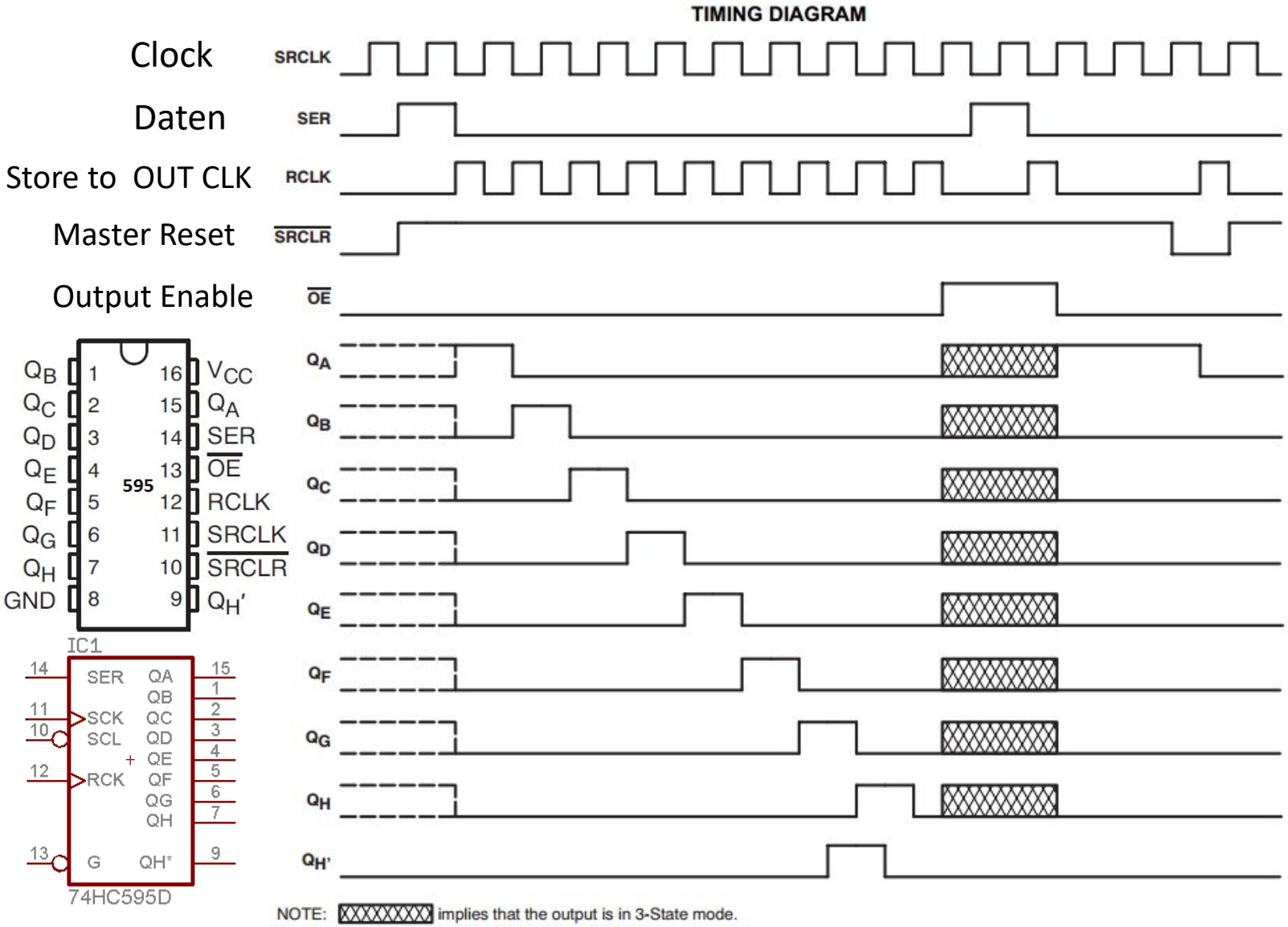


Steckbrett Variante
LED Leiste mit 74HC595 Shift Register

Pins, von rechts kommend: 6 5 4 3 2 1

GND Carry ST_CP_RCK SH_CP_SCK DS +5V

Beispielhafter **Shift-Transport eines** Datenbits durch die Stages



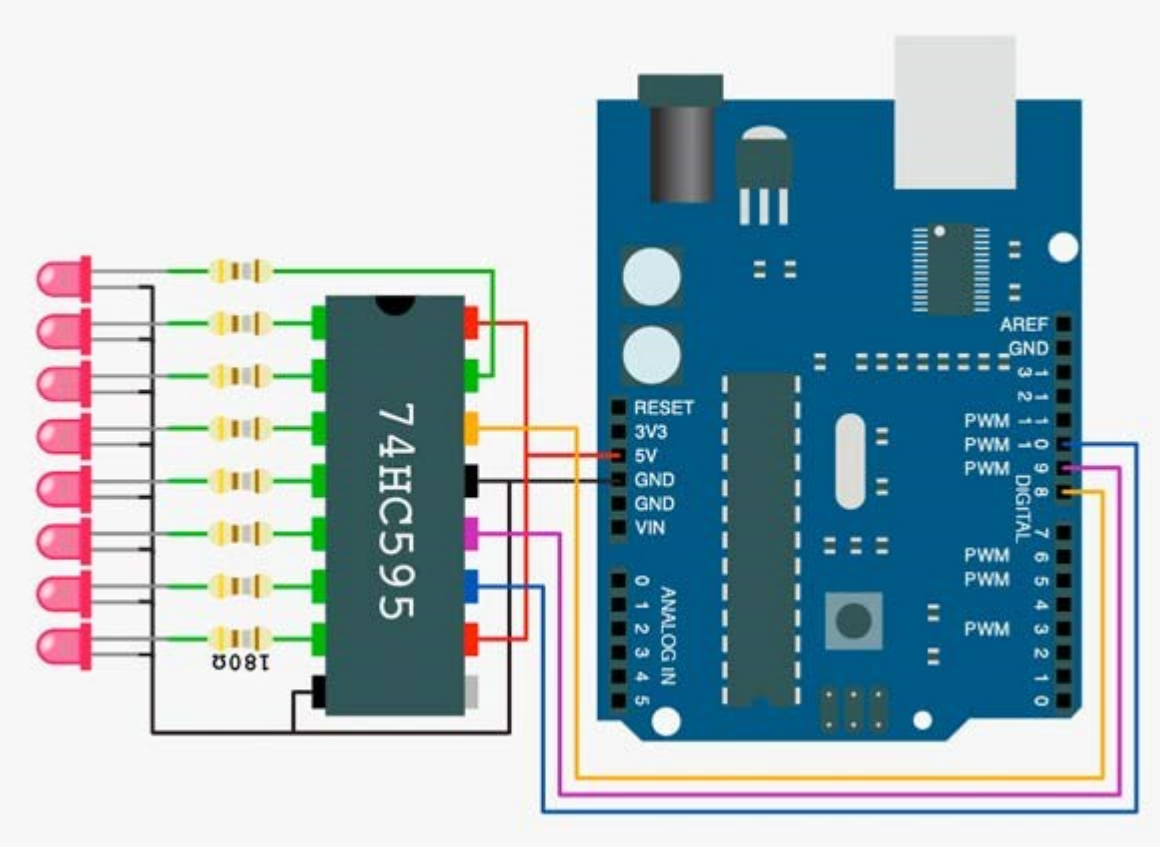
Das sieht nur kompliziert aus, bedeutet aber dass wir **nur 3 Leitungen** brauchen, um **beliebig viele Ausgangsleitungen** zu erzeugen.

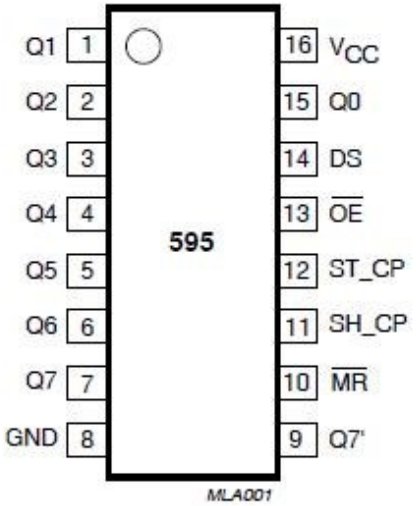
DS = Daten Bits

SH_CP = Shift Clock → 8 * Datenbits von DS reinschieben

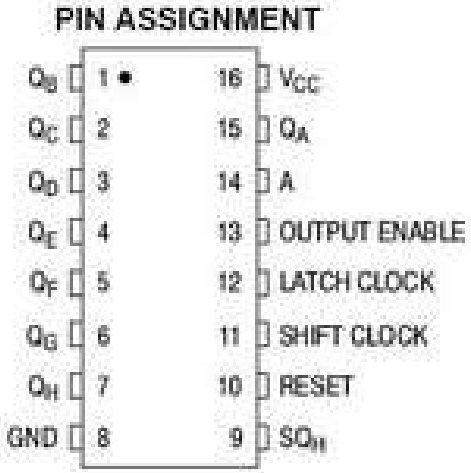
ST_CP = Store Clock → Daten auf den Ausgang legen

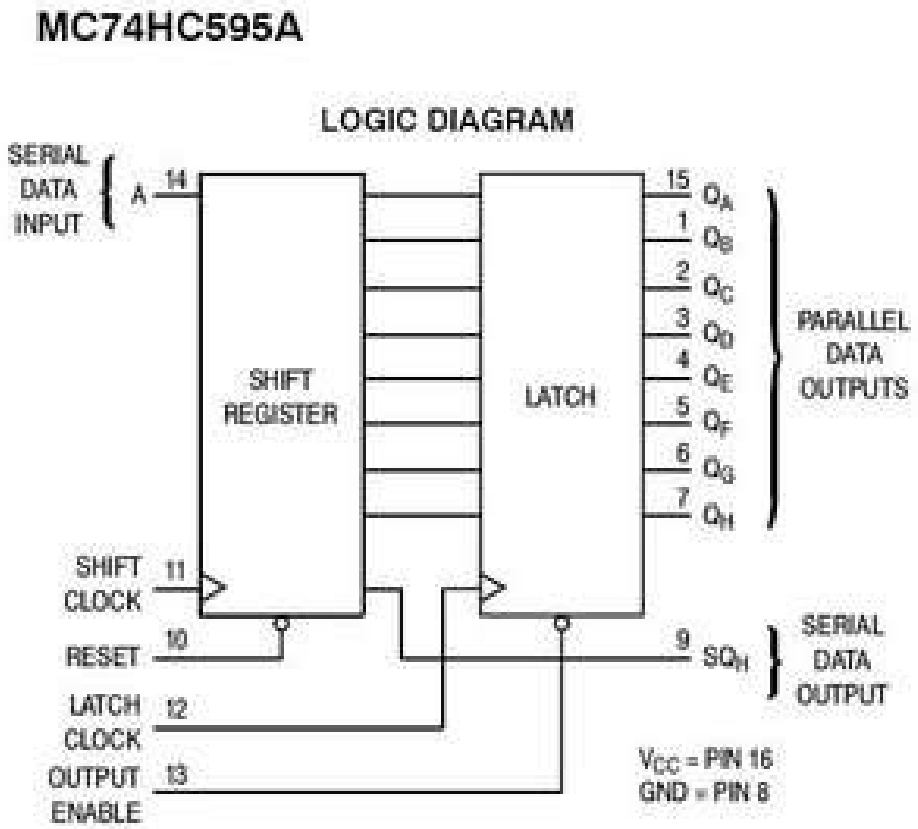
GND, /OR = GND. VCC, /MR=V+. Q7 = Übertrag für den nächsten Baustein.





Erst wenn alle 8 Bits im Shift-Register stehen, werden diese mit einem ÜBERNAHME-SIGNAL in den Ausgang (LATCH) transportiert.
So bleiben alle Bits in der „Außenwelt“ in Ruhe stehen, bis die **Soll-Situation** hergestellt ist.





```
#ifndef OUTSHIFT_74_595_HEADER
        #define OUTSHIFT_74_595_HEADER

#define OUTSHIFT74595_DDR   DDRB
#define OUTSHIFT74595_PORT  PORTB

#define DS_74595     (1<<PB0)     //DATA
#define ST_CP_74595  (1<<PB1)  //STORE
#define SH_CP_74595  (1<<PB2) //SHIFT

#define OUT_74595_INITDIRREG()
        (OUTSHIFT74595_DDR |= DS_74595 | ST_CP_74595 | SH_CP_74595)

#define OUT_74595_SERDATA_LOW() (OUTSHIFT74595_PORT  &= ~ DS_74595 ) //Daten rausshiften
#define OUT_74595_SERDATA_HIGH() (OUTSHIFT74595_PORT |= DS_74595 ) //Daten rausshiften

#define OUT_74595_SCK_LOW()  (OUTSHIFT74595_PORT &= ~ SH_CP_74595 ) //Shift clock
#define OUT_74595_SCK_HIGH() (OUTSHIFT74595_PORT |= SH_CP_74595 ) //Shift clock

#define OUT_74595_RCK_LOW() (OUTSHIFT74595_PORT &= ~ ST_CP_74595 ) // Latch Data to Output
#define OUT_74595_RCK_HIGH()(OUTSHIFT74595_PORT |= ST_CP_74595 ) // Latch Data to Output

#define USED_74595_ICS          1    //POINTER des Datentyps der Funktion ebenfalls anpassen
extern void OUT_74595( uint32_t u32Val ) ;
        #endif  // OUTSHIFT_74_595_HEADER
```

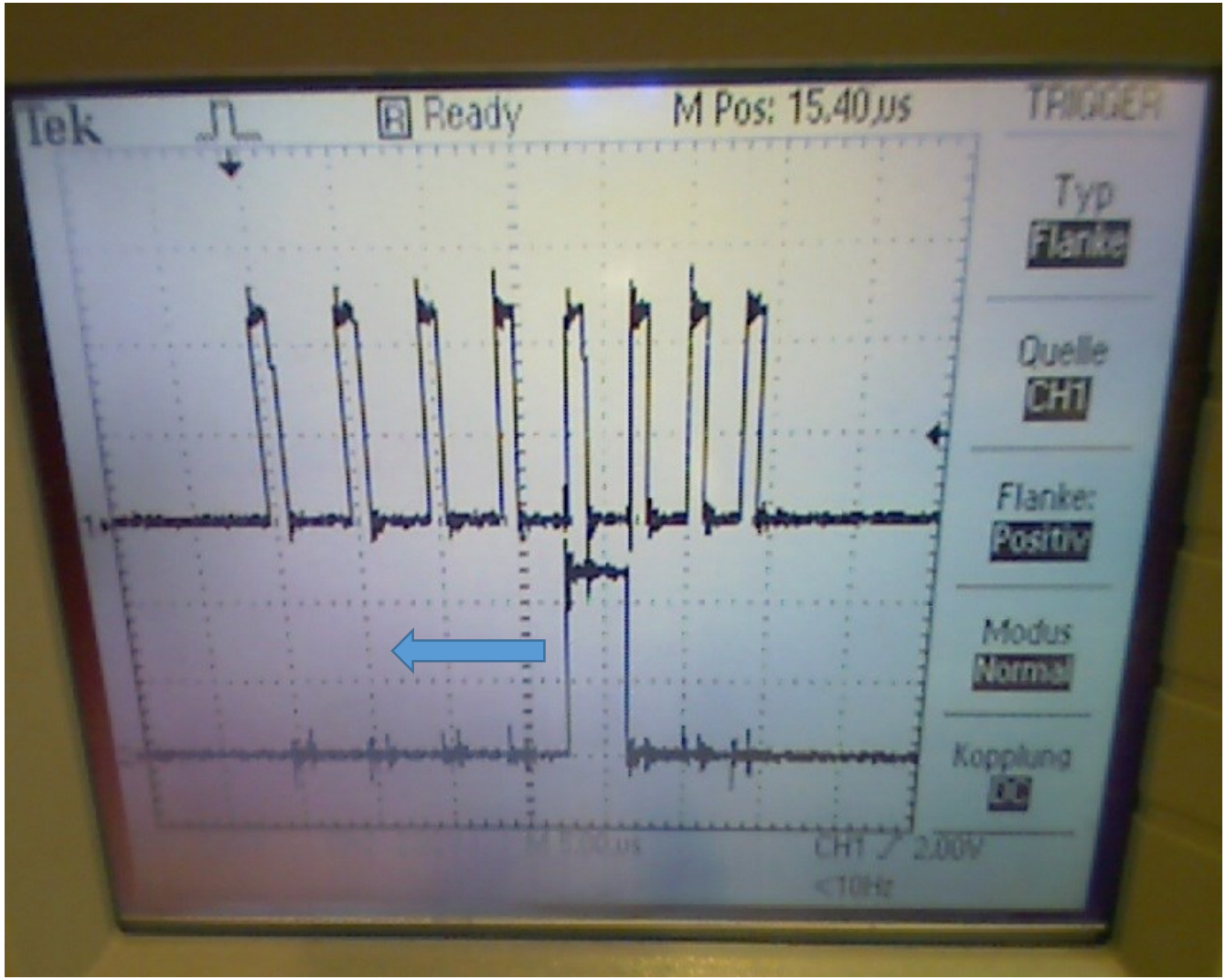
```
void OUT_74595( uint32_t u32Val )  //uint16_t uint32_t
{
uint8_t u8NN;
OUT_74595_InitDDRegs();  // AUSLAGERN IN INIT
OUT_74595_SCK_LOW();
//TWO_CYCLES();
u8NN =  (USED_74595_ICS * 8) - 1;
do
                { // Clock out bits
                if( u32Val &  (1UL << u8NN) )              // Ist es undiert eine EINS ?
                                {
                                OUT_74595_SERDATA_HIGH();
                                }
                 else
                                {
                                OUT_74595_SERDATA_LOW();
                                };                //Shift clock
                TWO_CYCLES(); //_delay_us(1);
                OUT_74595_SCK_HIGH();
                TWO_CYCLES(); //_delay_us(1);
                OUT_74595_SCK_LOW();
                TWO_CYCLES(); //_delay_us(1);

  }while( u8NN-- );

OUT_74595_RCK_HIGH(); // Latch Data to Output
TWO_CYCLES(); //_delay_us(1);
OUT_74595_RCK_LOW();
};
```

Shift Out mit 74HC595:

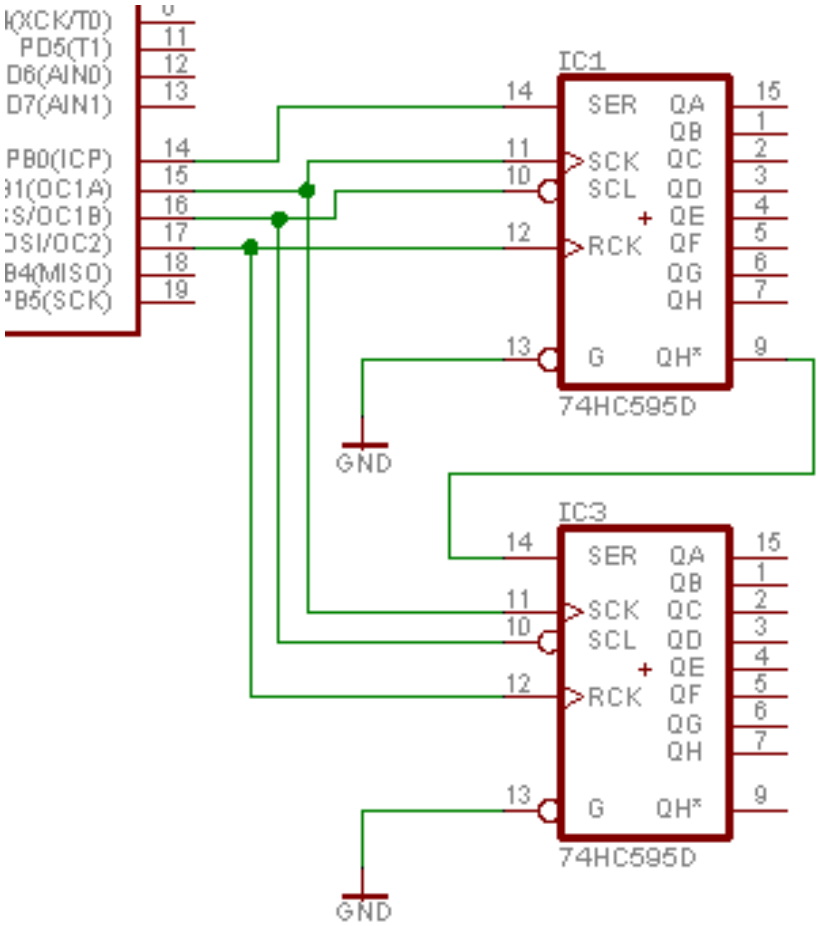
Oben: CLOCK SH_CP_74595 (1<<PB2) //SHIFT

Unten: DATA DS_74595 (1<<PB0) //DATA

→ u8Val= (1 << ((u8NN++) % 8))



Kaskadierung beliebig vieler Out-Shift-Register

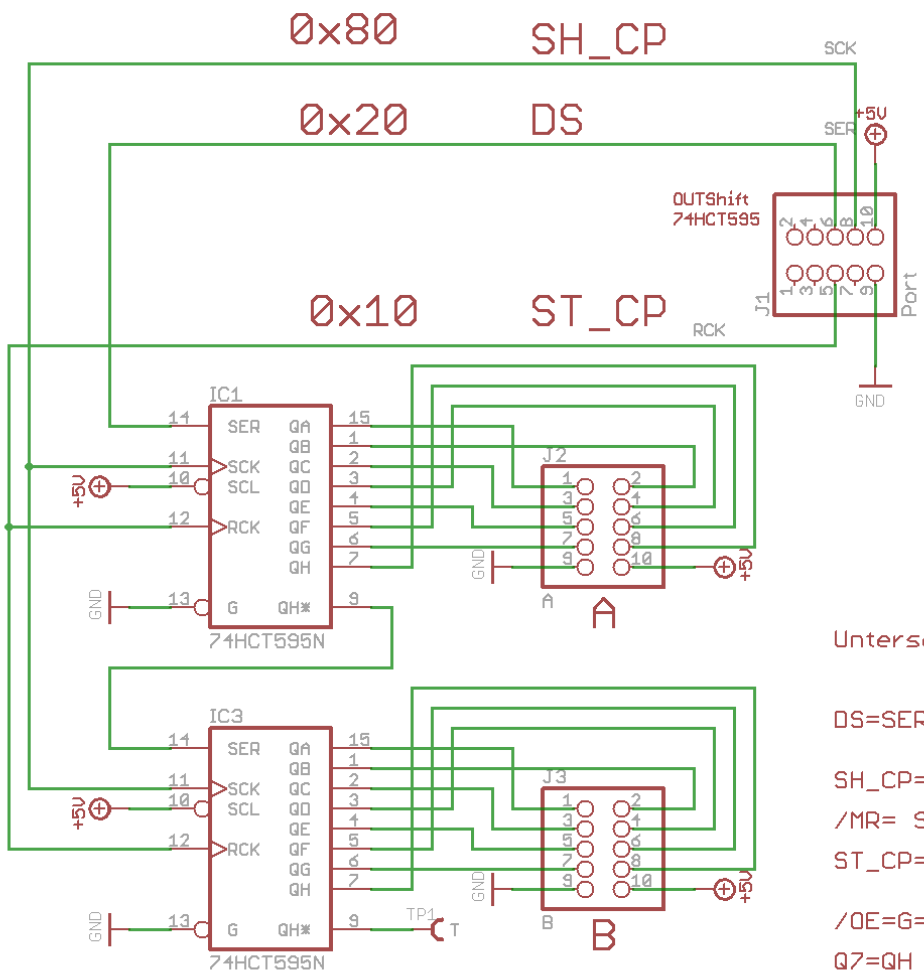


Wenn man 2 oder mehrere Shift-Register hintereinander schaltet, muss man das Programm so gestalten, dass **N*8** getaktet wird. Zugleich muss man bedenken, dass das **letzte Bit** in der Kette **zuerst gesendet** wird.

Hier im Beispiel: Bit 16..0

```
uint16_t  u16OutVal;
uint8_t    u8Parser = 15;
do
{
If(u16OutVal  &  (1 << u8Parser
) )
            set →Serbit
else        clr →Serbit
}while(u8Parser--);
```

Platine mit 2 kaskadierten 74HC595



OUTShift
74HCT595

Unterschiedliche Pin-Namen

DS=SER=Data Ser Input

SH_CP=SCK= Shift Clock Inp

/MR= SCL=Master Reset

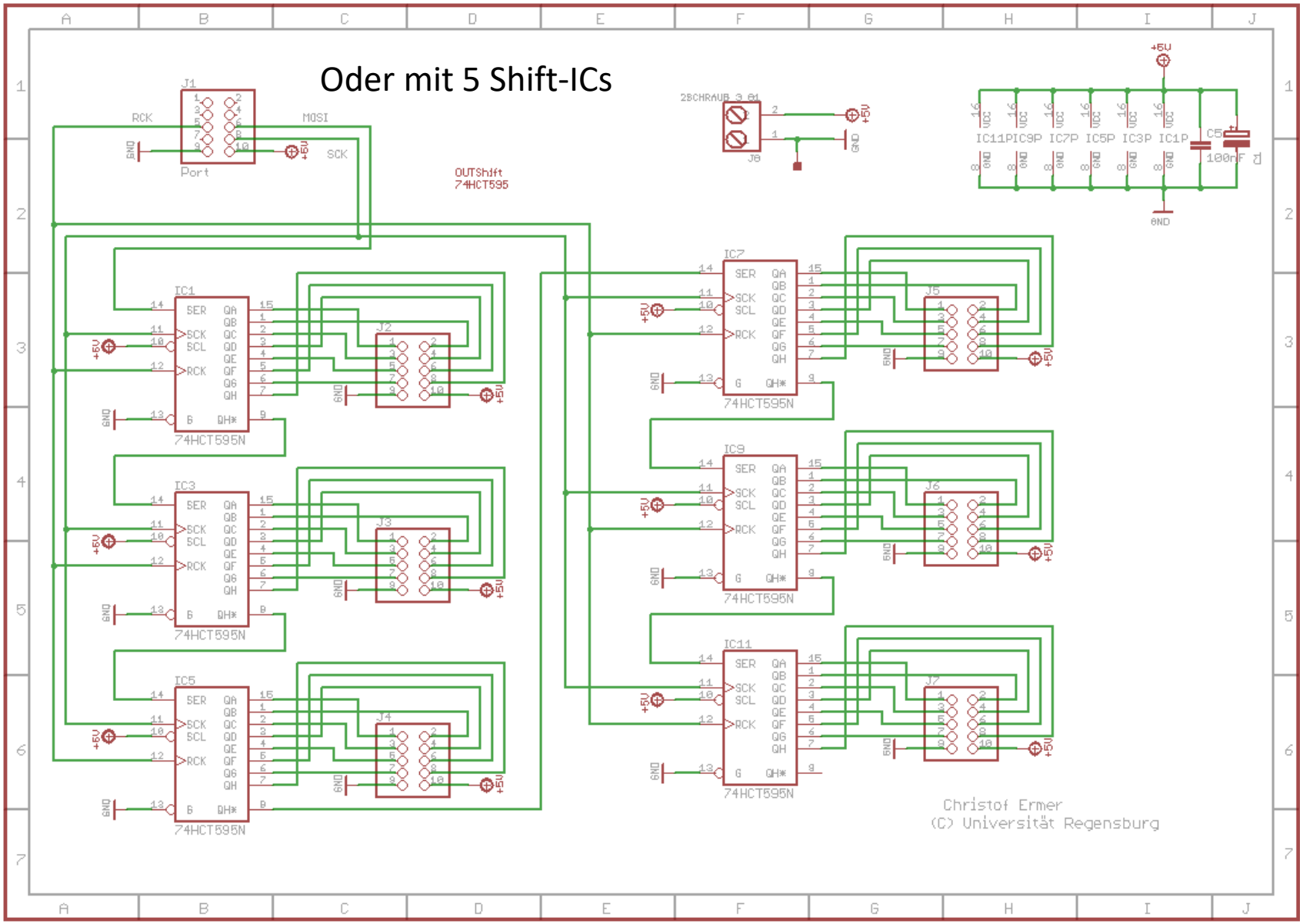
ST_CP=Storage Reg Clk Inp

/OE=G= Ouput enable

Q7=QH = übertrag

Christof Ermer

Oder mit 5 Shift-ICs



J1
IC1
IC3
IC5
J2
J3
J4
OutShift
C5
C1
74HC595
IC7
IC9
IC11
J5
J6
J7
J8
5'2010 CE

SPI Bus → MISO, MOSI, SCK, /SS=Shiftenable

Im Grunde wie ein 8 Bit In/Out Shift Register
Timing: XTAL / 2,4,8,16,32,64,128
Interner 8Bit ReadDataBuffer
SPI Status Register
SPI Control Register

```
void SPI_MasterInit(void)
{
/* Set MOSI and SCK output, all others input */
DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
/* Enable SPI, Master, set clock rate fck/16 */
SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}
void SPI_MasterTransmit(char cData)
{
/* Start transmission */
SPDR = cData;
/* Wait for transmission complete */
while(!(SPSR & (1<<SPIF)));
}
```

WICHTIG !

Makros mit #define erlauben Fehlerarme und sichere Beherrschung einzelner Bits am Port

Dies geschieht mit einem typischen SET an #define....

Einam erstellt kann dies mit copy-paste erweitert werden.

Beispiel am IC MAX543

```
#define MAX543_DDRX          DDRC
#define MAX543_PORTX         PORTC
#define MAX543_CLK           (1<<0)
#define MAX543_DATA          (1<<1)
------- usw   ....noch weitere Pins
#define MAX543_INIT()    (MAX543_DDRX |= MAX543_CLK | MAX543_DATA)
#define MAX543_CLK_HIGH()        (MAX543_PORTX |= MAX543_CLK_BIT)
#define MAX543_CLK_LOW()         (MAX543_PORTX &= ~ MAX543_CLK_BIT
#define MAX543_DATA_HIGH()       (MAX543_PORTX |= MAX543_DATA)
#define MAX543_DATA_LOW()     (MAX543_PORTX &= ~MAX543_DATA)
..... Weitere Pins
```

Typisches Präprozessor MAKRO Set

Es kommen immer wieder die selben Aufgaben vor.

1. Ein Bit nutzen und benennen.
2. Direction Register einstellen für Input oder Ouput.
3. Das Bit selbst, oder mehrere Gleichzeitig, setzen oder löschen

Dazu kann man einfach folgende #defines woanders her mit **Copy-Paste** holen und **umschreiben**.
Das spart Zeit und man hat es dennoch ordentlich.
Hier an einem reduziertem Beispiel mit dem Clock-Pin eines DAC Bausteines MAX543

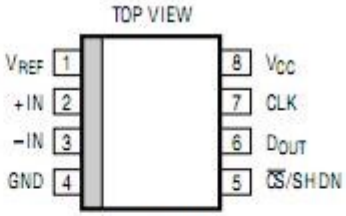
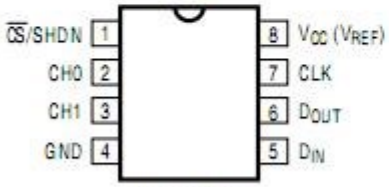
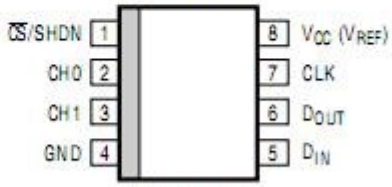
```
#define CLK_MAX543_DDRX      DDRC
#define CLK_MAX543_PORTX     PORTC
#define CLK_MAX543_PIN       3

        //jetzt werden damit die Makros definiert , fast wie echte Funktionen mit ()

#define INIT_CLK_MAX543_DDRX()  (CLK_MAX543_DDRX     |=      (1<< CLK_MAX543_PIN)
#define SET_CLK_MAX543()        (CLK_MAX543_PORTX    |=      (1<< CLK_MAX543_PIN)
#define CLR_CLK_MAX543()        (CLK_MAX543_PORTX    &= ~    (1<< CLK_MAX543_PIN)
```

```
#ifndef MAX543_HEADER
#define MAX543_HEADER
#define MAX543PORTX             PORTB
#define MAX543DIRREG            DDRB

#define MAX543CLK_BIT                   _BV(PB0)
#define MAX543SRI_BIT                   _BV(PB1)
#define MAX543LOAD_BIT                  _BV(PB2)
#define MAX543_CTRL_MASK     (MAX543CLK_BIT | MAX543SRI_BIT|  MAX543LOAD_BIT)

#define MAX543_INITDIR()  (MAX543DIRREG |=  MAX543_CTRL_MASK) //DDRC |= 0b0001110

#define MAX543_SRI_HIGH()               (MAX543PORTX |= MAX543SRI_BIT);
#define MAX543_SRI_LOW()                (MAX543PORTX &= ~MAX543SRI_BIT);

#define MAX543_CLK_HIGH()               (MAX543PORTX |= MAX543CLK_BIT);
#define MAX543_CLK_LOW()                (MAX543PORTX &= ~MAX543CLK_BIT);

#define MAX543_LOAD_HIGH()              (MAX543PORTX |= MAX543LOAD_BIT);
#define MAX543_LOAD_LOW()               (MAX543PORTX &= ~MAX543LOAD_BIT);

#define AUFLOESUNG_SINUS 4096 //0...4095
extern void WriteMAX543(uint16_t u16Val);
#endif
```

```
void WriteMAX543(uint16_t u16Val)
{
uint8_t u8NN=11;
MAX543_INITDIR();
MAX543_CLK_LOW(); //Startcondition
MAX543_LOAD_HIGH(); //Startcondition
do
        {
        if(     u16Val & (1 << u8NN) )
                {
                MAX543_SRI_HIGH();
                }
        else
                {
                MAX543_SRI_LOW();
                }
                MAX543_CLK_HIGH();
        _delay_us(1); //__asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
        if( !u8NN )
                {
                MAX543_LOAD_LOW();
                };
        MAX543_CLK_LOW();
        }while(u8NN--);
_delay_us(1); //__asm__ __volatile__( "rjmp 1f\n 1:" );    // 2 cycles
MAX543_LOAD_HIGH()
};
```

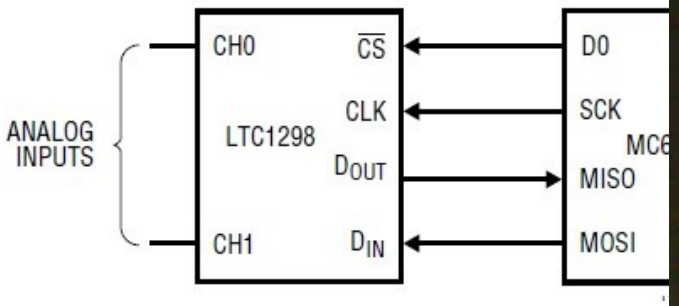
ADC-IC LTC1298 mit Pin-Out

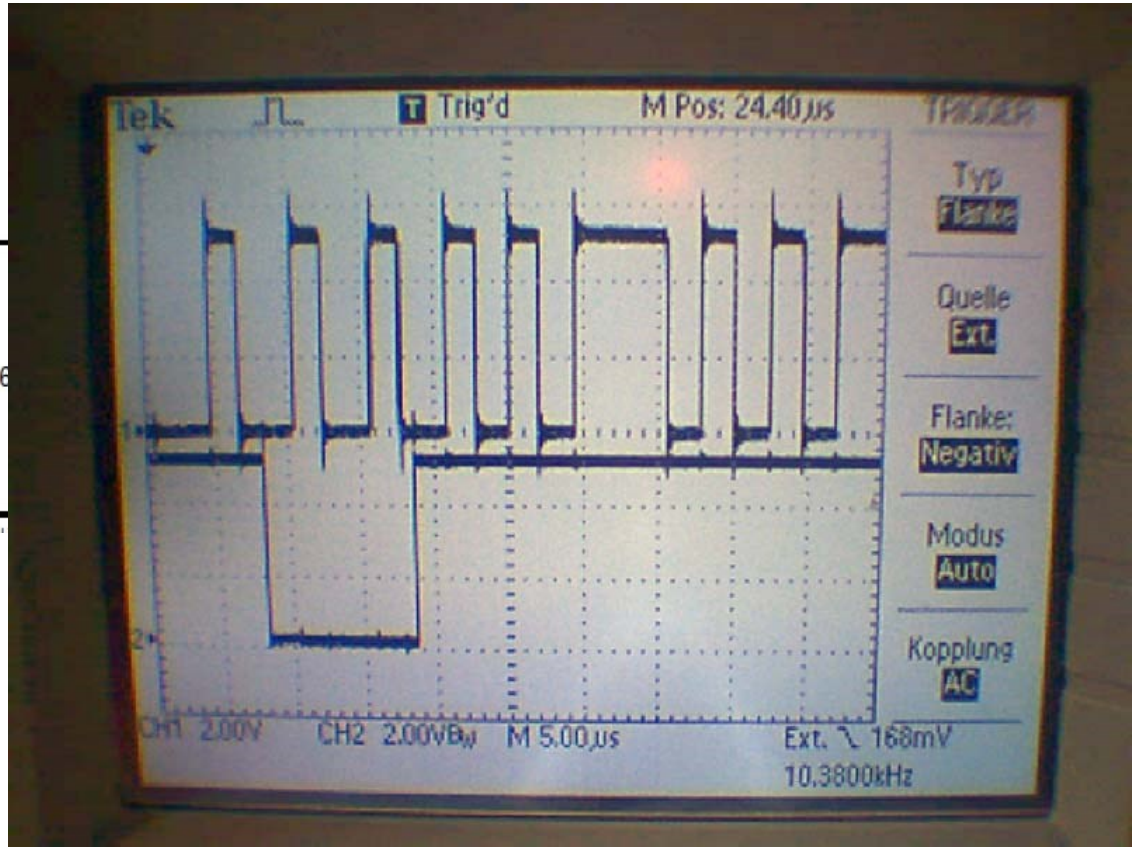
So kommt ein Chip über den Ladentisch. Man besorgt sich die Manual-PDF und pickt sich das Wichtige raus (siehe Bild).
Je nach TYP-Variante machen wir uns mit den Anschlüssen vertraut:
/CS/SHD, CLK, OUT, [DIN], {CHx oder +/-IN} usw.. Vcc=V+, GND=-Pol
Hier erst einmal das P**in-Out**:

 N8 PACKAGE 8-LEAD PLASTIC DIP T_{JMAX} = 150°C, θ_{JA} = 130°C/W	ORDER PART NUMBER LTC1286CN8 LTC1286IN8	 S8 PACKAGE 8-LEAD PLASTIC SOIC T_{JMAX} = 150°C, θ_{JA} = 175°C/W	ORDER PART NUMBER LTC1286CS8 LTC1286IS8 PART MARKING 1286 1286I
 N8 PACKAGE 8-LEAD PLASTIC DIP T_{JMAX} = 150°C, θ_{JA} = 130°C/W	ORDER PART NUMBER LTC1298CN8 LTC1298IN8	 S8 PACKAGE 8-LEAD PLASTIC SOIC T_{JMAX} = 150°C, θ_{JA} = 175°C/W	ORDER PART NUMBER LTC1298CS8 LTC1298IS8 PART MARKING 1298 1298I

Consult factory for military grade parts.

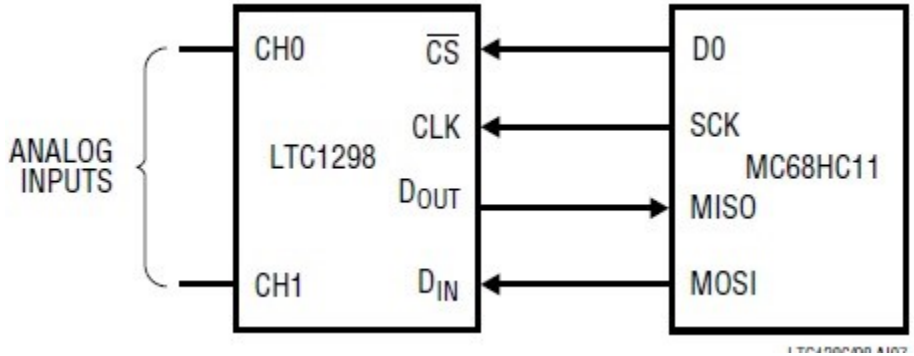
Nun betrachten wir DOUT (LTC1298→µC) relativ zu DIN
Oben = DOUT (LTC1298→µC)
Unten = DIN (µC →LTC1298)
Trigger-punkt für **/CS** (Chip-Select) ist der Pfeil.
Bildschirmbild zeigt ziemlich genau einen ganzen Zyklus.
Interpretation: Während der D-IN Phase ist der Ausgang des Chips **hochomig** (3-State) und unbestimmt. Siehe Gewackel bei der Lowflanke unten...
Nach dem Clk-Up/Down n.d. Initialisierung wird der Ausgang freigegeben.





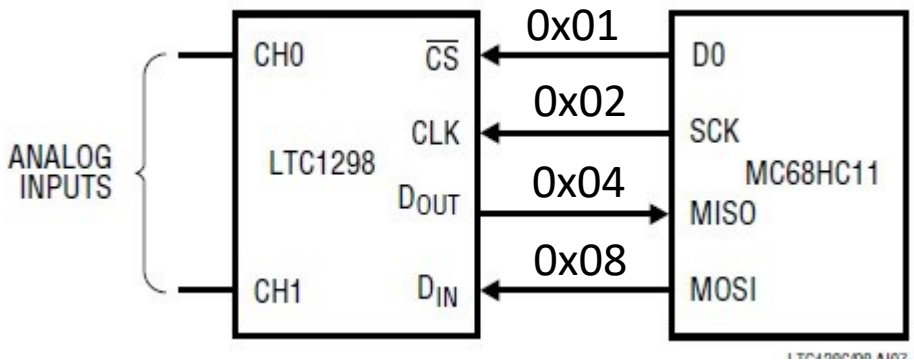
LTC1298 ADC- IC

So will es der Erfinder :
Mit 4 Signalleitungen kann dieser Baustein gesteuert werden.
1.) **/CS** = Chip Select oder Shut Down (bei High Level)
2.) **CLK** = Clock. Das bedeutet normalerweise, dass hier eine µC Output eine steigende und dann eine fallende Flanke hat: **rising/falling edge**
3.) **DIN** = Data In. Ein µC Output sendet Initialisierungsdaten etc. an den Baustein,
4.) **DOUT** = Data Out- Ein ,C Leseeingang holt die Daten (in Zusammenarbeit mit Clock-Out) hier vom Chip ab.



Der µController wird mit dem Baustein L1298 verbunden.
Doch bevor Strom angelegt wird, wird zuerst die Eingangs-Ausgangszuordnung richtig programmiert.

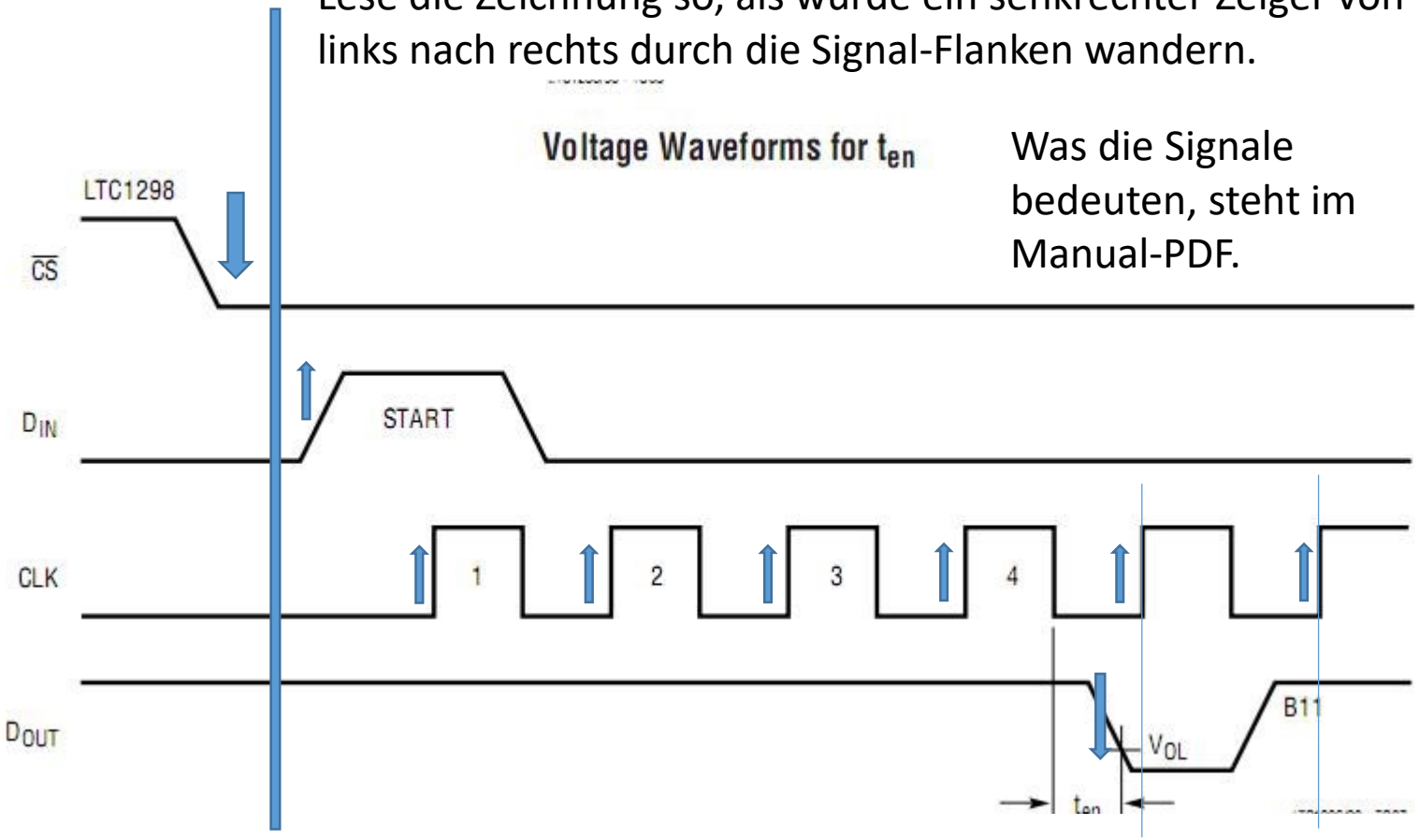
Was ist Input oder Output?
"**Des einen Output ist der Input des Anderen„**
Heißt: Wir brauchen 3 Output-Leitungen und eine als Input
Wir wollen gemischten Portbetrieb, um nur ein Port zu benutzen. Aber es wäre beliebig festlegbar. Die Zuordnung kann frei erfunden werden.




```
#define L1298_CS        0x01
#define L1298_CLK       0x02
#define L1298_DIN       0x04
#define L1298_DOUT      0x08

#define L1298_DDR       PORTB
#define L1298_PORT      PORTB
#define L1298_PIN       PINB
```

```
L1298_DDR &= ~( 1 << L1298_DOUT);
L1298_DDR |=  (1<<1298_CS) | (1<<1298_CLK) | (1<<1298_DIN);
```

Die Signale dienen dazu, den Chip mit dem µController zu synchronisieren. **/CS** „falling edge“ löst eine Antwort des IC aus. **CLK** „rising edge“ ist Datenbit lesen. **DOUT →L**= Load. Lese die Zeichnung so, als würde ein senkrechter Zeiger von links nach rechts durch die Signal-Flanken wandern.



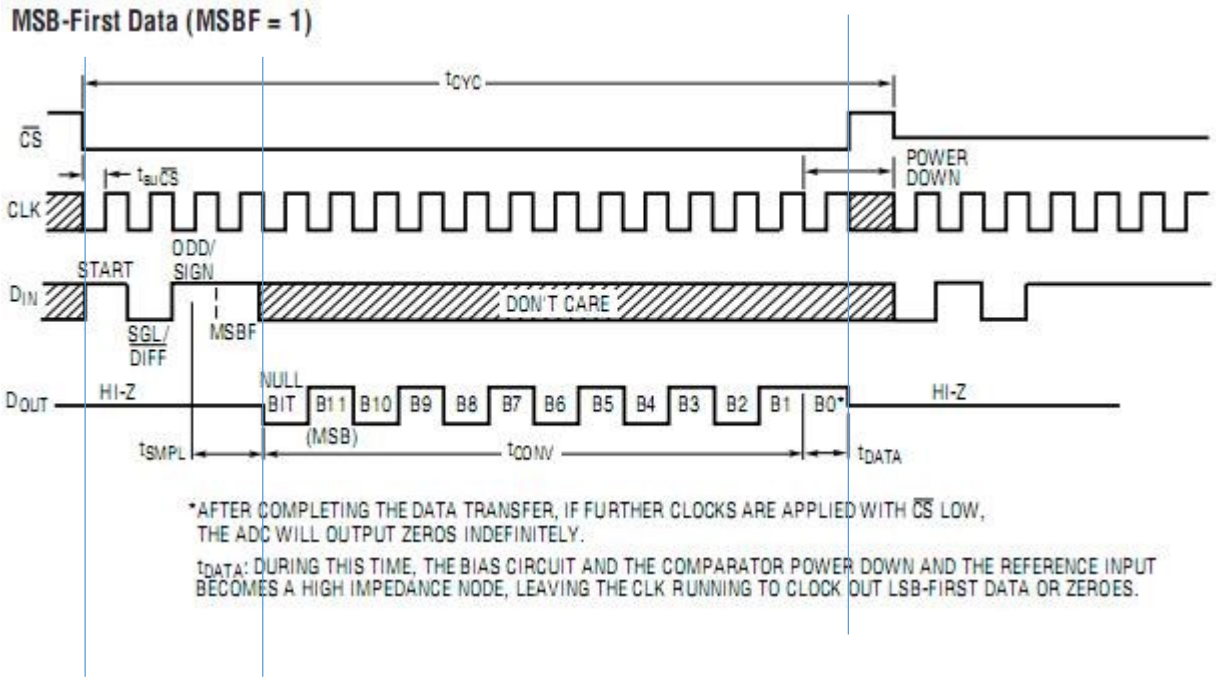
Was die Signale bedeuten, steht im Manual-PDF.

Zuerst werden vom µC, nach /CS ↓ die 4 Initialisierungsbits gesendet. Lesemoment ist ↑ Clk. Dann mit ↑ Clk von 11..0 lesen.

/CS → 0
4 Init-BITs
Rising CLK

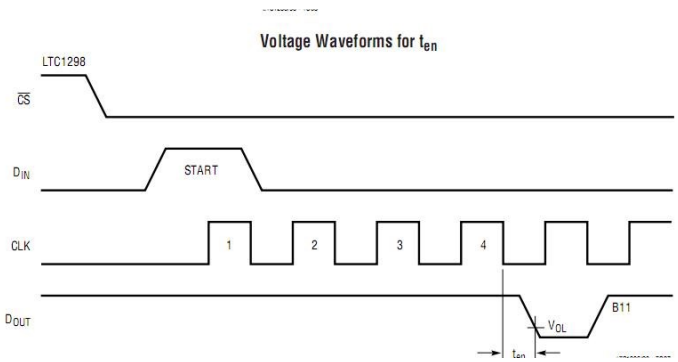
Rising CLK
Read Bit 11
Rising CLK
Read Bit 10
..
..
..
Rising CLK
Read Bit 0

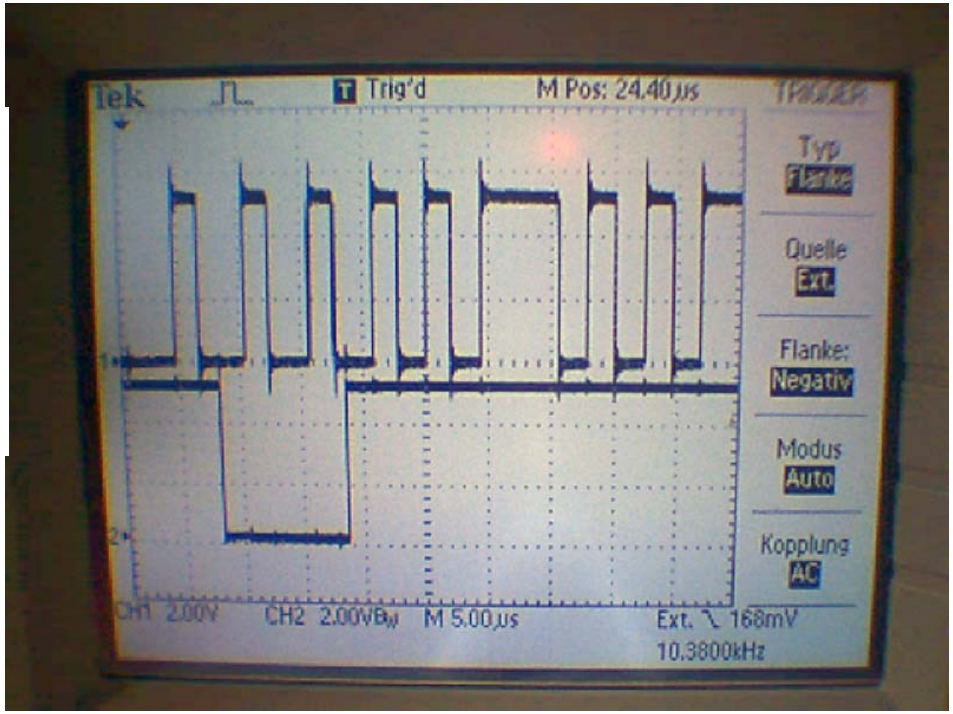
/CS → 1



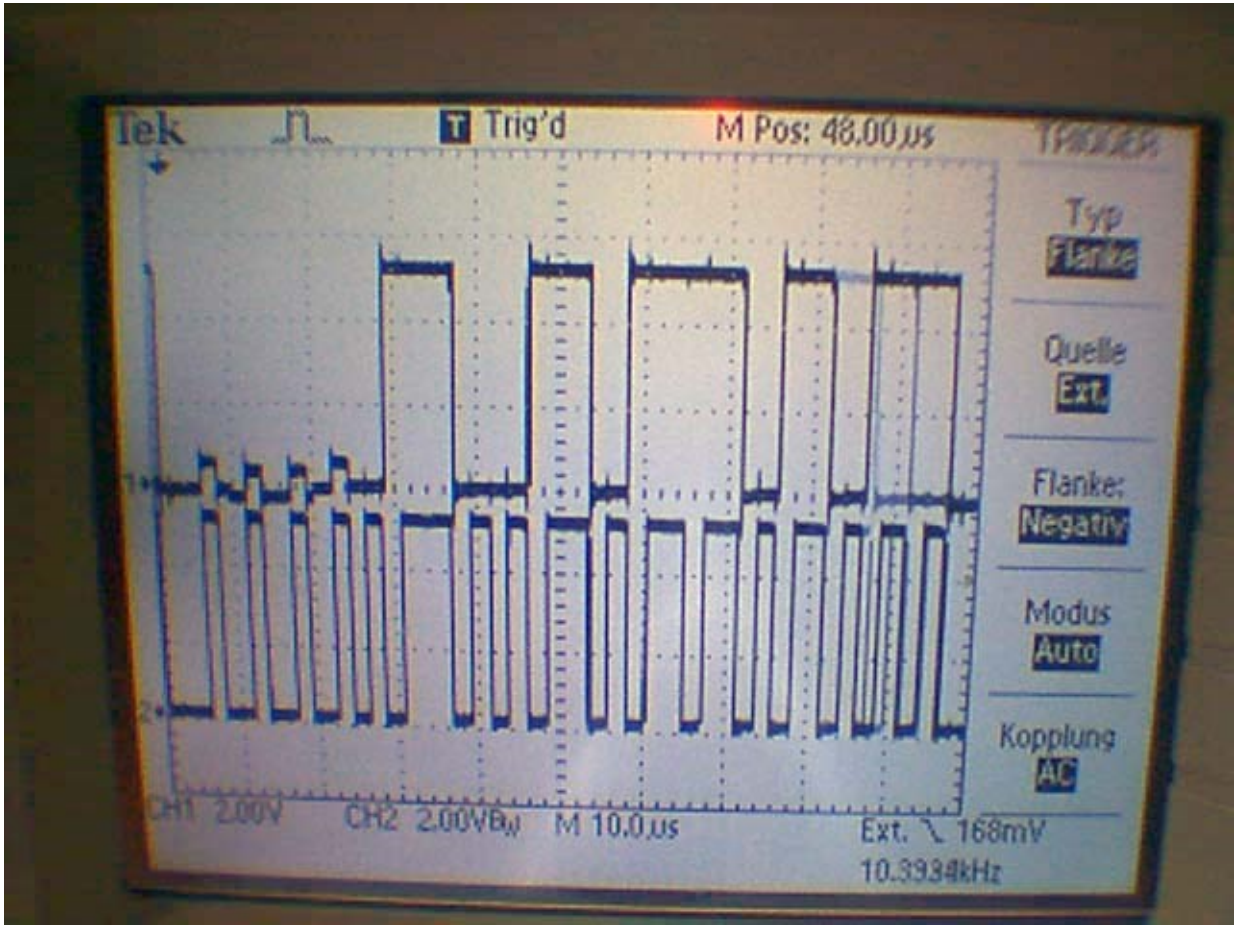
Die Datenbit Information wird mit "Steigender Flanke" des CLKs (oben) übertragen.
Nach 4 Bit "Lesen" DIN wird noch ein CLK (High-Low) übertragen, dann beginnt der Lesezyklus) über einen anderen Draht (bzw. Pin). → DOUT
→Triggerpunkt für das Foto = Low-Flanke des Chip Select = **/CS**

Oberes Signal = **Clock** ~2*2,5µS = 200kHz,
Unteres Signal = **DIN** (µC →LTC1298) (Data-In)Bits 1,2,3,4, und zwar ***1001*** 1, = Startbit, dann kommt 0, 0, 1, was die Modus Programmierwörter für Differential Eingang +/- sind.
.....mal abzählen an der steigenden Flanke des CLKs oben: (die ersten 4 Clocks!)





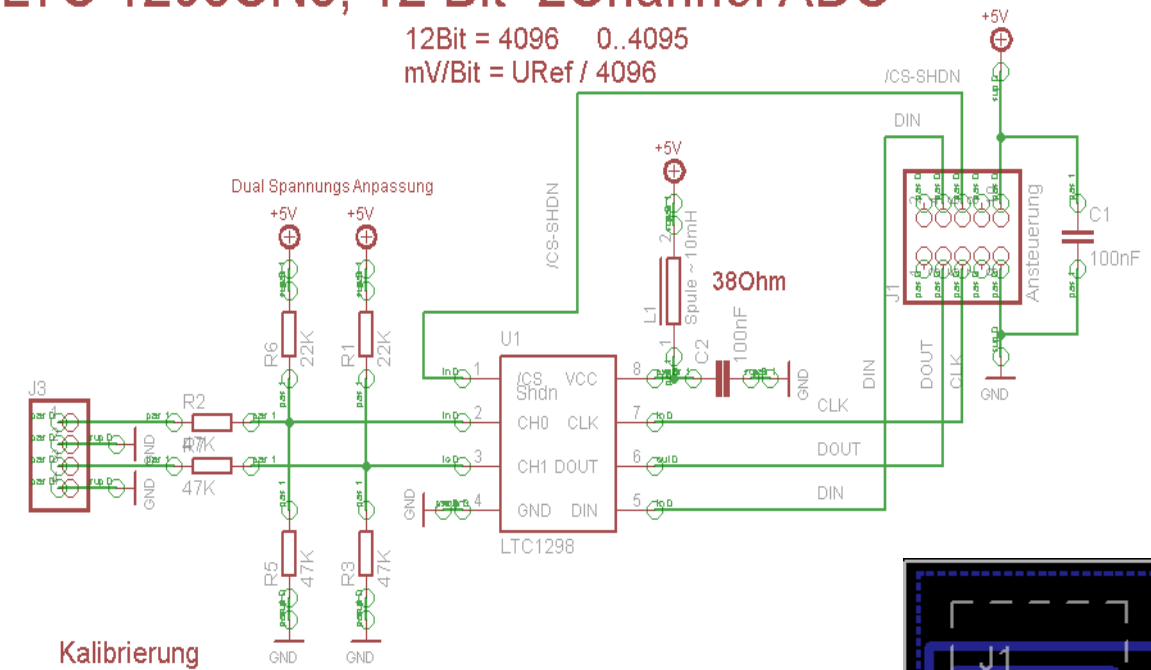
Der Pfeil zeigt auf: Start mit /CS = Low
Oben= wieder **Dout** zu sehen (LTC1298→µC)
Unten = diesmal der **CLK** (Clock)



LTC 1298CN8, 12 Bit 2Channel ADC

12Bit = 4096 0..4095

mV/Bit = URef / 4096

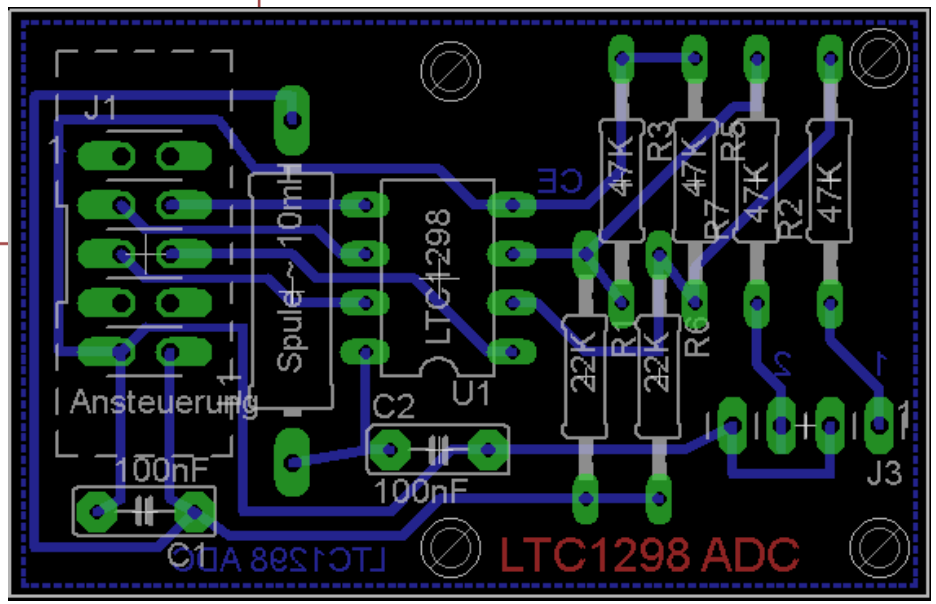


Kalibrierung

1. Eingang auf 0V; = fADC_Zero_Value
2. Eingang auf VREF 5.0V legen; = fADC_Ref_Value
3. Berechne KalibrierungsWert: "Volt / BIT"

fCalibFactor = Meßspannung wird berechnet (fADC_Ref_Value - fADC_Zero_Value

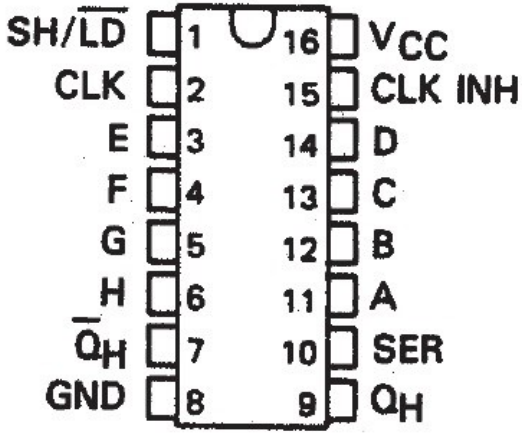
Meßwert = (ADCValue - u16Adc_Zero_Value) * fCalibFactor



Input Shift 74xx165

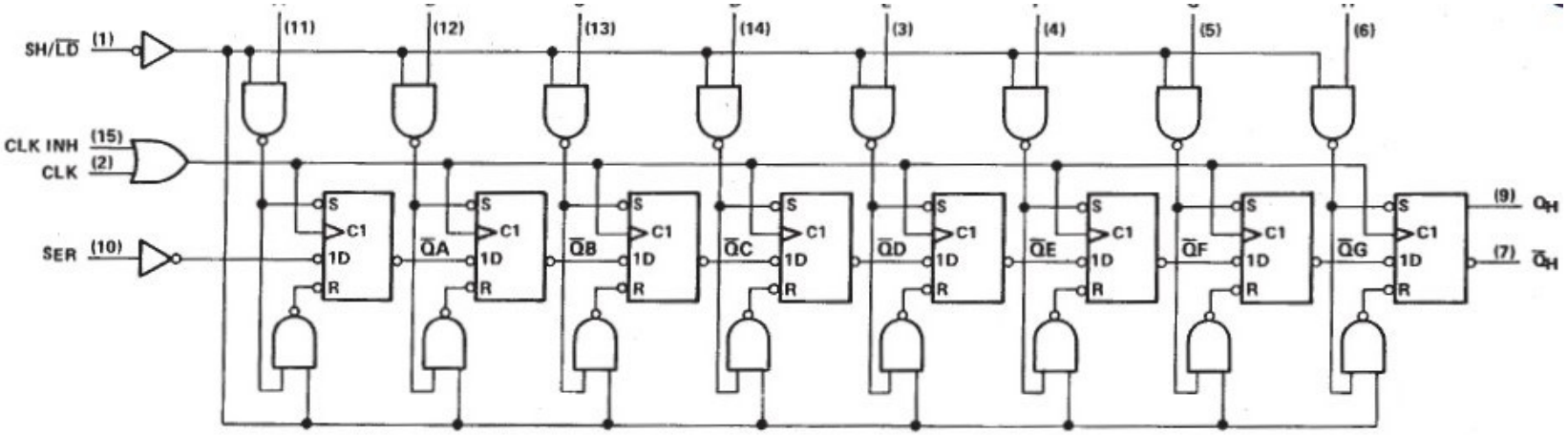
Input Shift Register, Parallel-Load 74xx165

SN54165, SN54LS165A . . . J OR W PACKAGE
SN74165 . . . N PACKAGE
SN74LS165A . . . D OR N PACKAGE
(TOP VIEW)



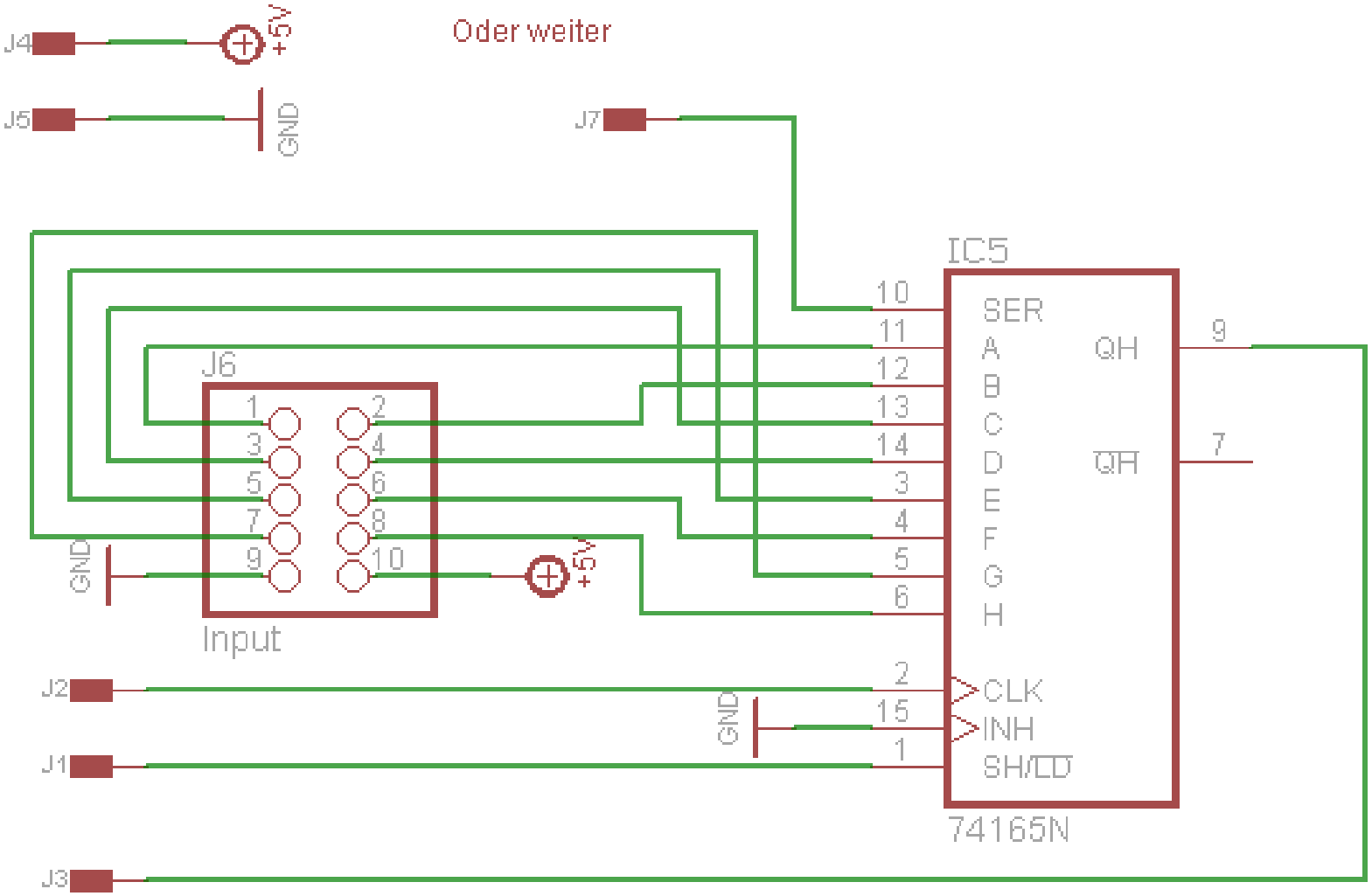
Selbstverständlich können mit Shift-Registern auch die **Eingänge erweitert** werden. Die Technik ist genau dieselbe. Diese ICs sind schon sehr alt und fast obsolet. Dennoch sind sie gut und ideal, um das Prinzip zu zeigen.

Es ist unbedingt erforderlich, die Manual-PDF zu konsultieren.

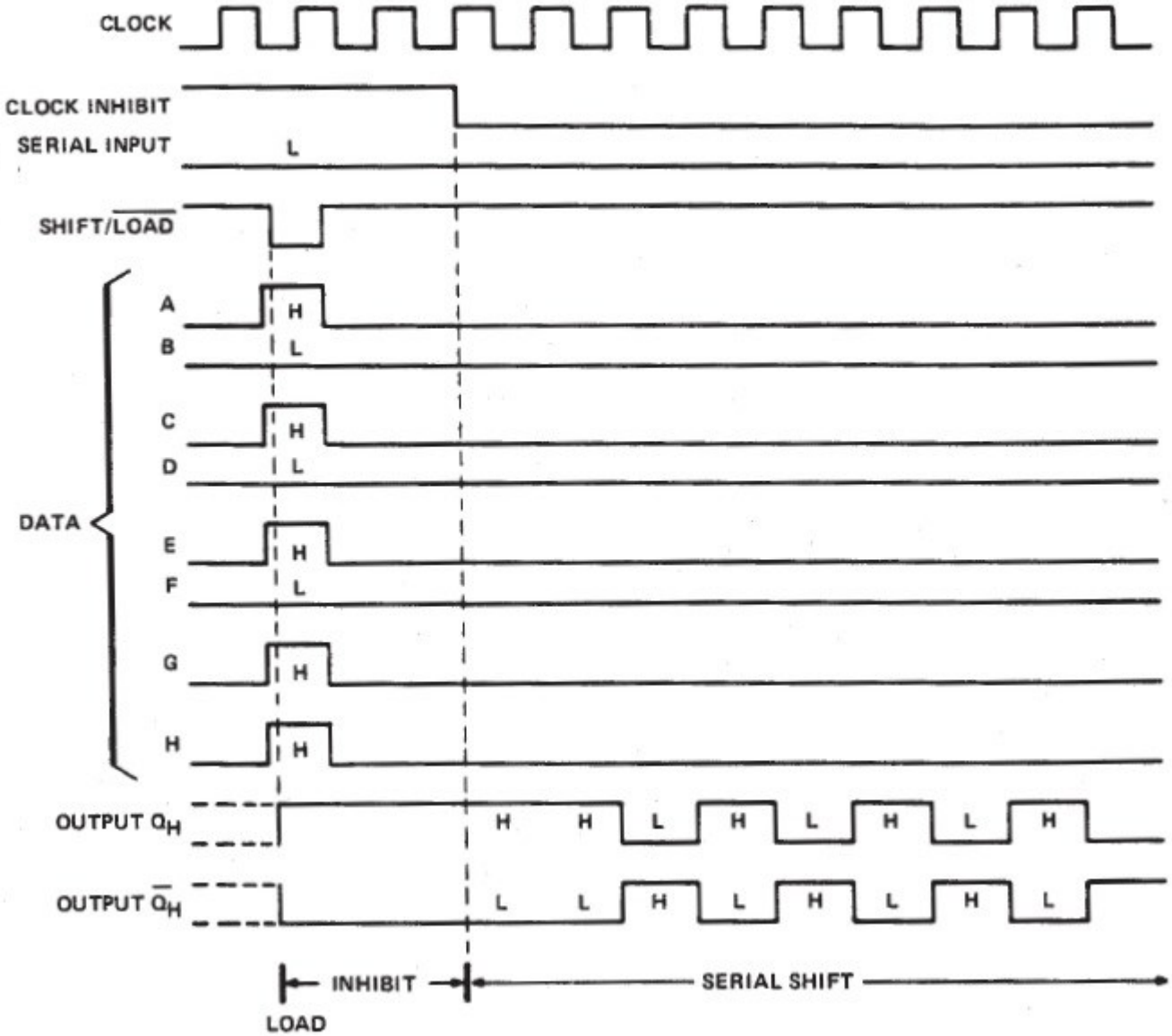


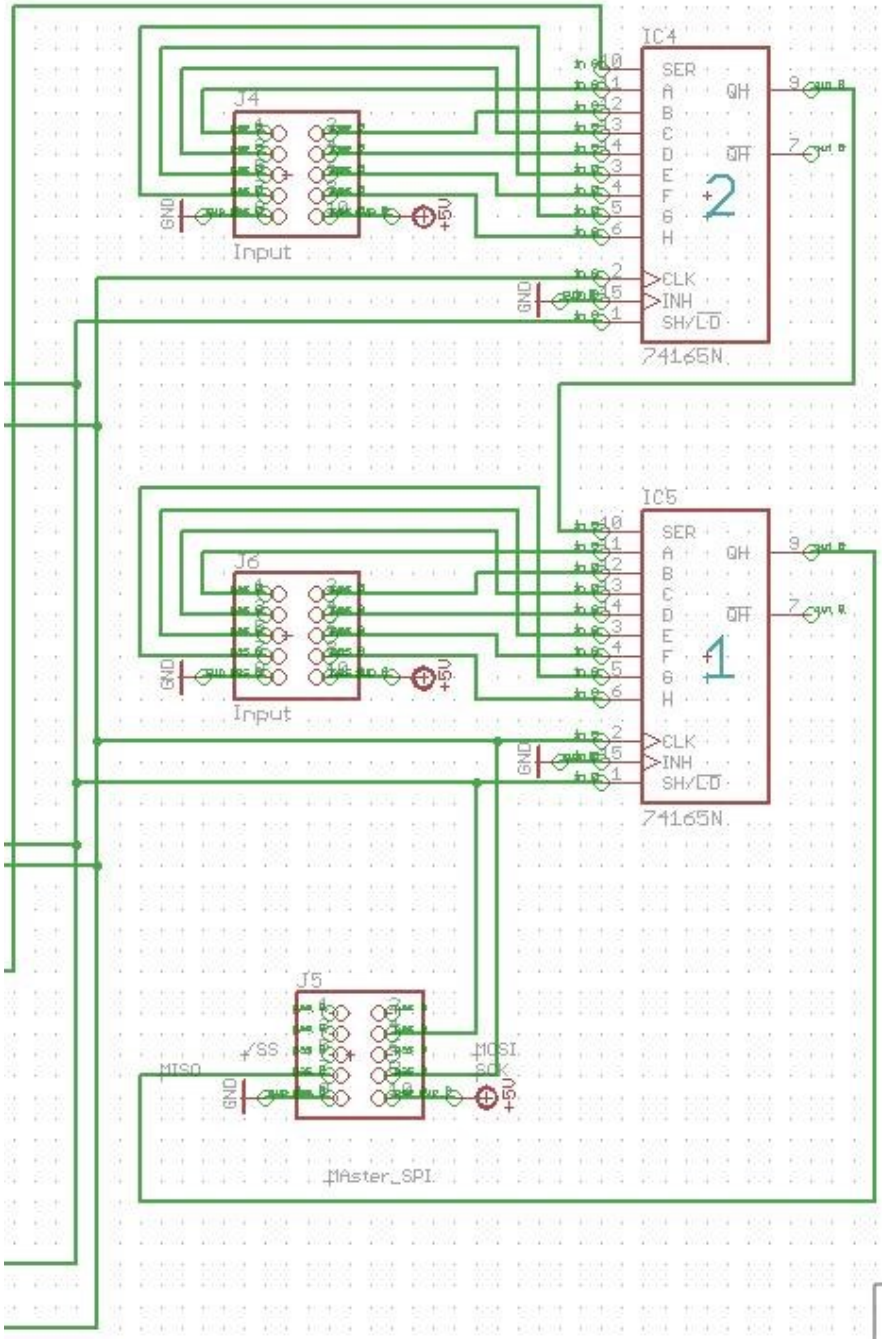
Pin numbers shown are for D, J, N, and W packages.

8 TTL Inputleitungen.
3 Steuerleitungen sind erforderlich..



typical shift, load, and inhibit sequences





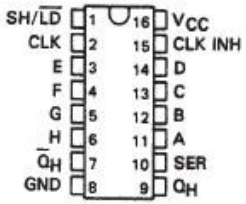
Ausschnitt: Kaskadierung von x * 74HC165.

3 Leitungen reichen, um beliebig viele Eingänge zu erzeugen.

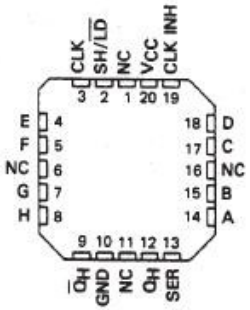
SN54165, SN54LS165A, SN74165, SN74LS165A
PARALLEL-LOAD 8-BIT SHIFT REGISTERS

SDLS062B – OCTOBER 1976 – REVISED JANUARY 2000

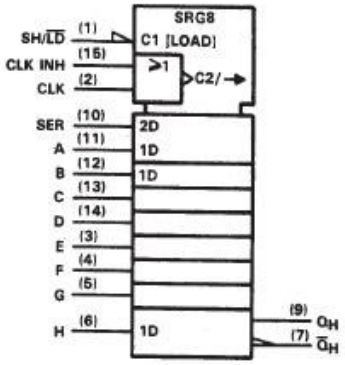
SN54165, SN54LS165A . . . J OR W PACKAGE
SN74165 . . . N PACKAGE
SN74LS165A . . . D OR N PACKAGE
(TOP VIEW)



SN54LS165A . . . FK PACKAGE
(TOP VIEW)



logic symbol†



```
#ifndef IN74165_HEADER
#define IN74165_HEADER
#define IN74165_USED
#define INSHIFT_74165_PORT          PORTD
#define INSHIFT_74165_PIN           PIND
#define INSHIFT_74165_DDRX          DDRD
#define INSHIFT_74165_DDRX_VALUES               0b10001000

#define CHIP_COUNT_74165_MAX    5               // Zahl  Der Shift Chips
#define INSHIFT_74165_CLK_BIT       (1<<PD2)
#define INSHIFT_74165_SHLD           (1<<PD3)   //SH /LD
#define INSHIFT_74165_QH_BIT         (1<<PD4)   //QH  Input

//MAKROS
#define INIT_74165PORTDIR()         (INSHIFT_74165_DDRX = INSHIFT_74165_DDRX_VALUES)
#define SET_74165_CLK()             (INSHIFT_74165_PORT |= INSHIFT_74165_CLK_BIT)
#define CLR_74165_CLK()             (INSHIFT_74165_PORT &= ~INSHIFT_74165_CLK_BIT)

#define SET_74165_SHLD()            (INSHIFT_74165_PORT |= INSHIFT_74165_SHLD)
#define CLR_74165_SHLD()            (INSHIFT_74165_PORT &= ~INSHIFT_74165_SHLD)

#define Read_QH()                   (INSHIFT_74165_PIN & INSHIFT_74165_QH_BIT)

//extern uint8_t u8aInShift[ CHIP_COUNT_74165_MAX ];        //ARRAY
extern uint64_t ScanInShift74165( void );

#endif //IN74165_HEADER
```

```
uint64_t ScanInShift74165( void )
{
uint8_t u8NN;
uint64_t g64InShift;    //ULL zur Aufnahme des Ergebnisses

CLR_74165_CLK();
CLR_74165_SHLD();
_delay_us(1);
SET_74165_SHLD();

g64InShift = 0;
for(u8NN=0; u8NN < (CHIP_COUNT_74165_MAX * 8); u8NN++)
        {
        _delay_us(1);
        if( Read_QH() )         //wenn 1
                {
                g64InShift |= (uint64_t)( 1ULL << u8NN );
                };
        SET_74165_CLK();
        _delay_us(1);
        CLR_74165_CLK();
        };
return g64InShift;
};
```

Zusammenfassung
effektive Software Anwendung

1. **Schritt. Hardware-Situation ‘sprechenden’ Namen zuordnen**: (→headerfile)

```
#define Sh74595_DDRX  DDRC
#define SH74595_PORT  PORTC
#define SH74595_SCK    (1<< 2)
#define SH74595_SCL    (1<< 3)
#define SH74595_RCK    (1<< 4)
#define SH74595_MASK (SH74595_SCK | SH74595_SCL | SH74595_RCK)
```

So kann Hardwaresituation in sprechende Bedeutung gewandelt werden.
Hier leicht global änderbar

2. **Makros erstellen!**

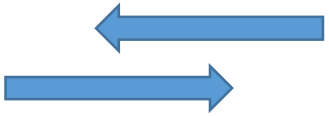
```
#define INIT_Sh74595_SCK()  (Sh74595_DDRX |= SH74595_MASK)

#define SET_Sh74595_SCK()   (SH74595_PORT |= SH74595_SCK)
#define CLR_Sh74595_SCK()   (SH74595_PORT &= ~SH74595_SCK)

#define SET_Sh74595_SCL()   (SH74595_PORT |= SH74595_SCL)
#define CLR_Sh74595_SCL()   (SH74595_PORT &= ~SH74595_SCL)

#define SET_Sh74595_RCK()  (SH74595_PORT |= SH74595_SCL)
```

usw..

Jetzt können die Makros ‚lesbar‘ im Code verwendet werden

Kapitel 6 Externe Peripherie

…nicht die **Datenverarbeitung** ist das Problem, sondern die Daten hinein und wieder heraus zu kriegen.

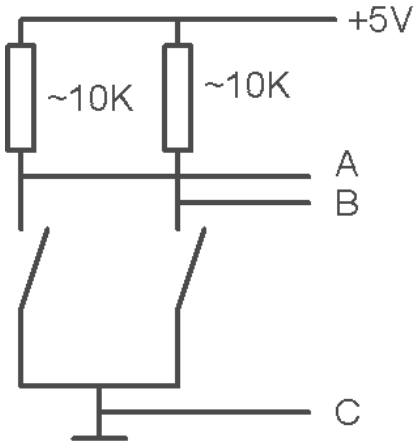


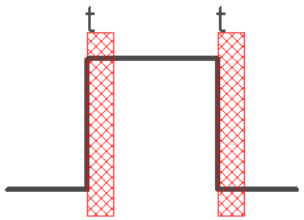
- Signale nach den Anforderungen externer Hardware (ICs etc.) gezielt an die I-O Pins zuweisen.
- Die Signale einem **Protokoll** entsprechend ausgeben, und/oder einlesen und im zeitlichen Kontext koordinieren, interpretieren, und datentypenrichtig auswerten.

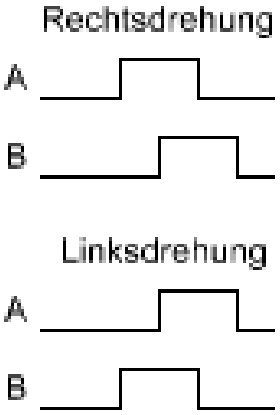
DAC – Digital Analog Converter

Sehr praktisch und irgendwie cool. Rastende anschlagfreie Drehungen der Achse produzieren aufwärts/abwärts zählende Pulse. Es gibt sie auch mit Druckpunkt-Switch (Anwendungen: Senderwahl, Menüauswahl, Modus-wechsel, ergonomische stilvolle Bedienung).
Aber nicht von selbst: ein Algorithmus ist erforderlich und das Prellen muss kompensiert werden.

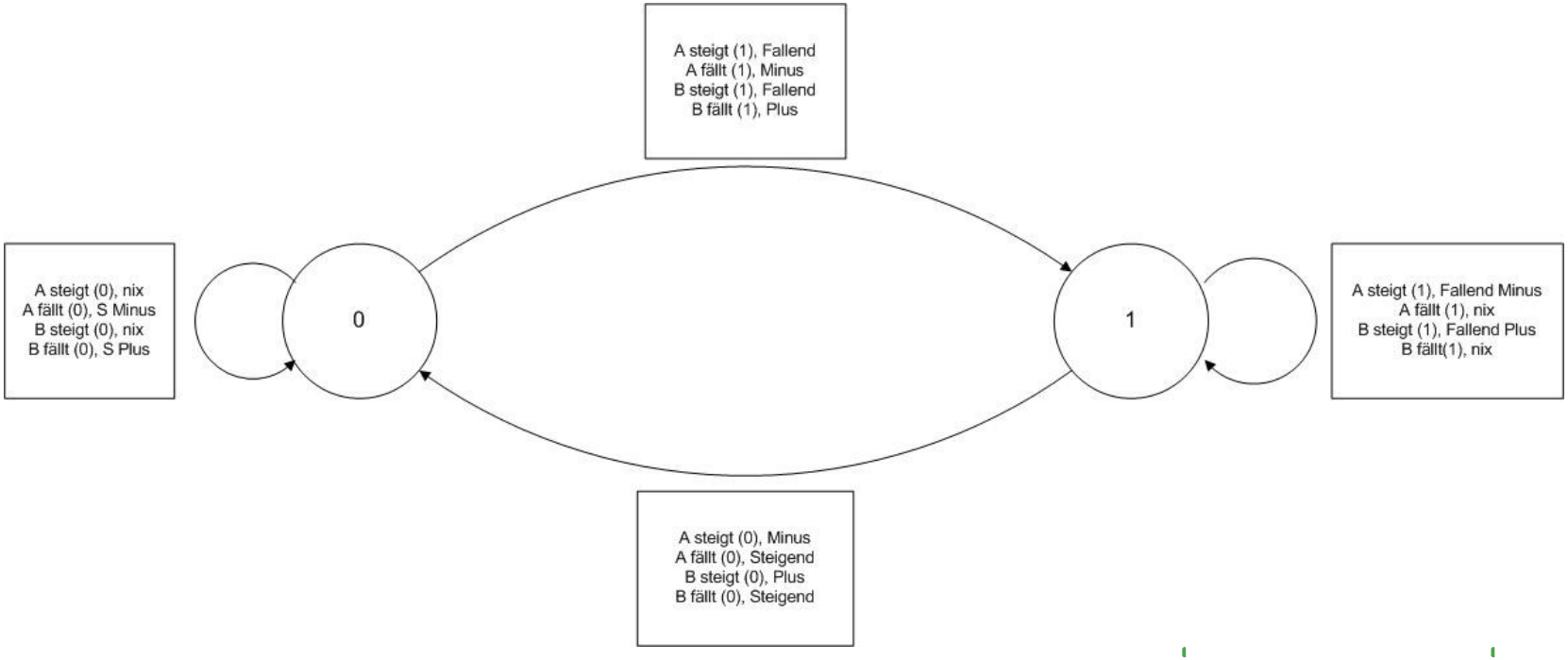
Eagle : piher.lbr
switch-apls.lbr
ALPS_EC12E_SW
ALPS rotary encoder
Typ: STEC12E07
ohne Taster

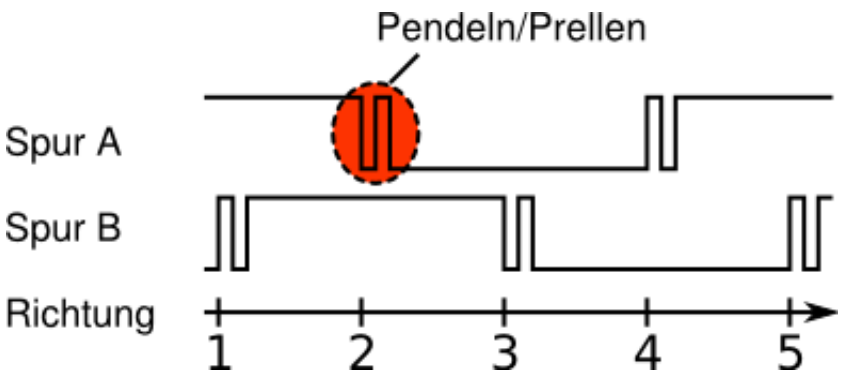


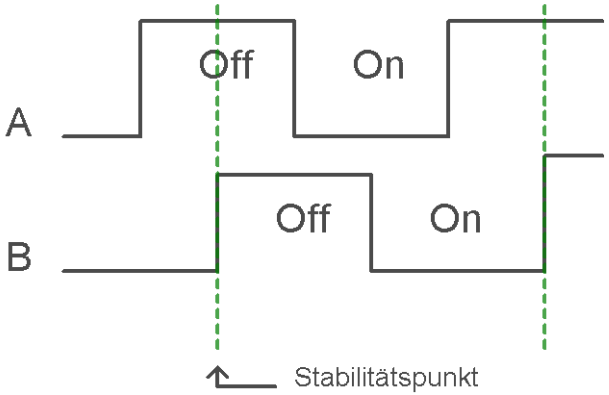




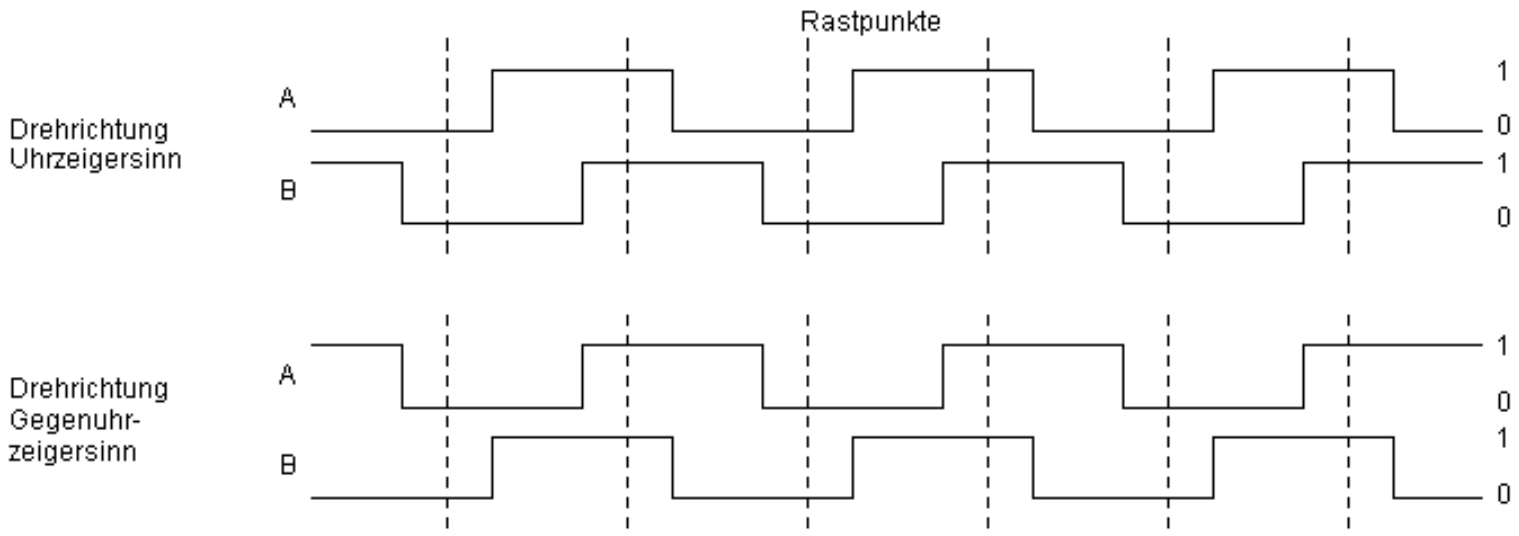
Guter Artikel: http://www.mikrocontroller.net/articles/Drehgeber


A steigt (1), Fallend
A fällt (1), Minus
B steigt (1), Fallend
B fällt (1), Plus
A steigt (0), nix
A fällt (0), S Minus
B steigt (0), nix
B fällt (0), S Plus
0
1
A steigt (1), Fallend Minus
A fällt (1), nix
B steigt (1), Fallend Plus
B fällt(1), nix
A steigt (0), Minus
A fällt (0), Steigend
B steigt (0), Plus
B fällt (0), Steigend


Pendeln/Prellen
Spur A
Spur B
Richtung
1
2
3
4
5


Off
On
A
Off
On
B
Stabilitätspunkt

Anspruchsvoller, als es auf den ersten Blick wirkt:
Die Programmierung sollte für ein sicheres Arbeiten mit Interrupts geschehen. Dazu entprellt. Dabei können die **Rastpunkte** Aufschluss geben *über* die Drehrichtung.
Zustand merken→ vergleichen mit neuem Zustand.



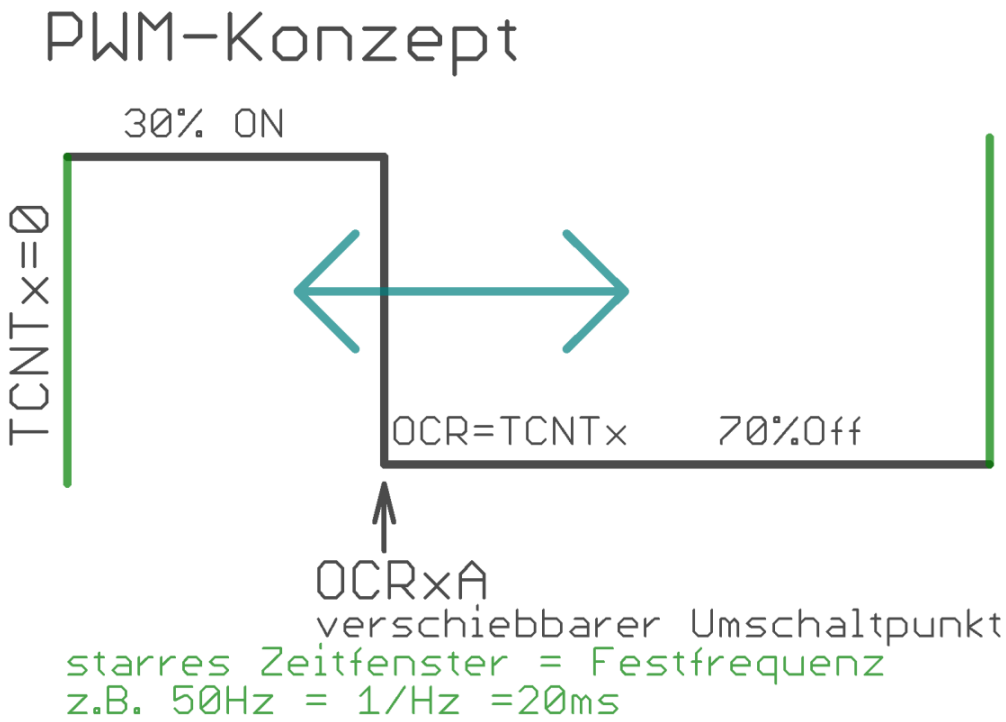
Programmier-Übung !?

Kapitel 7

- Weitere Hardware-Fähigkeiten nutzen.
- Aktoren wie Fernsteuer-Servos ansteuern.
- Schritt und lineare Motoren betreiben.

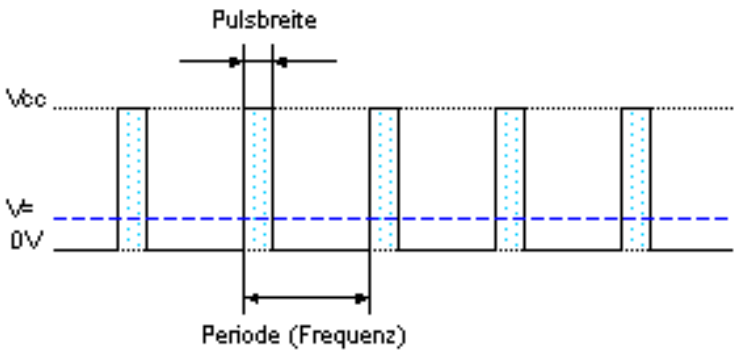
Puls-Weiten-Modulation - PWM

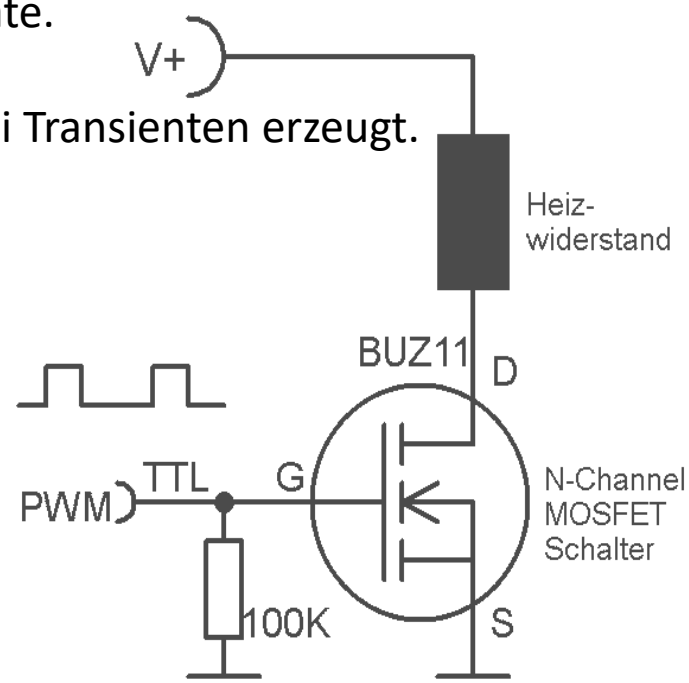
PWM ist einfach eine prozentuale Einschaltzeit mit einem festen Zeitrahmen (). Somit auch eine Leitungsregelung. Zudem kann mit Hilfe der PWM über nur einen Draht komplexe Information transportiert werden.



PWM Anwendungen

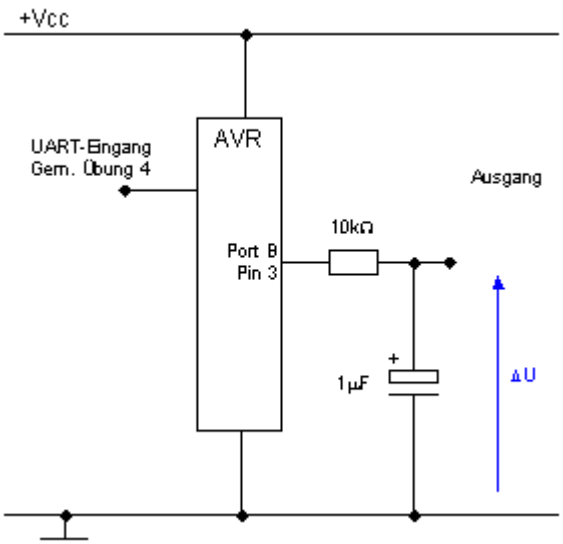
Mit PWM können durch das Pulsbreiten-Verhältnis der Einschaltzeit - ohne großen Leistungsverlust des elektronischen Schalters (MOSFET sind ideal) - große Leistungen gesteuert werden.
Der gezeigte BUZ11 kann **15A** gegen Minus (=Gnd) schalten.
Das sind bei 40V schon 0..600 WATT.. Je nach PWM-Einschaltzeit.
Siehe den Sicherheits-Pulldown-Widerstand am Gate.
Verlustleistung am Schalttransistor:
OFF = 0Watt → ON= ~0 Watt Verlust wird nur bei Transienten erzeugt.

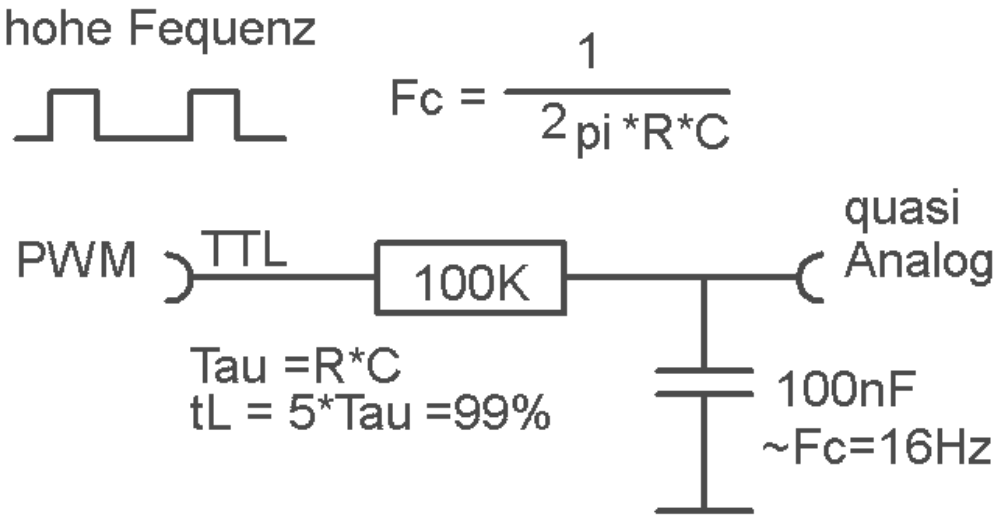




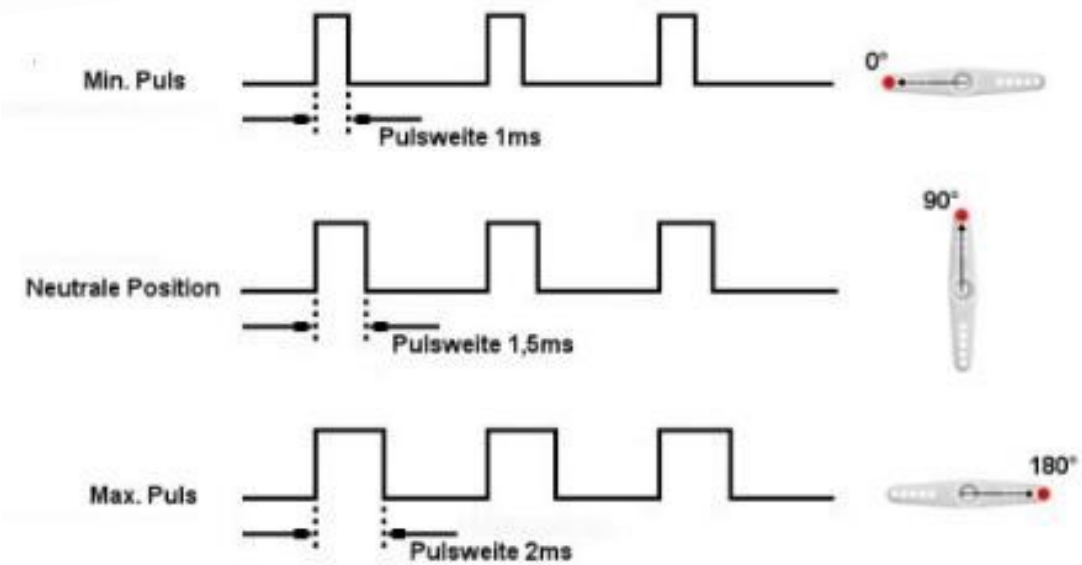
DAC mit PWM

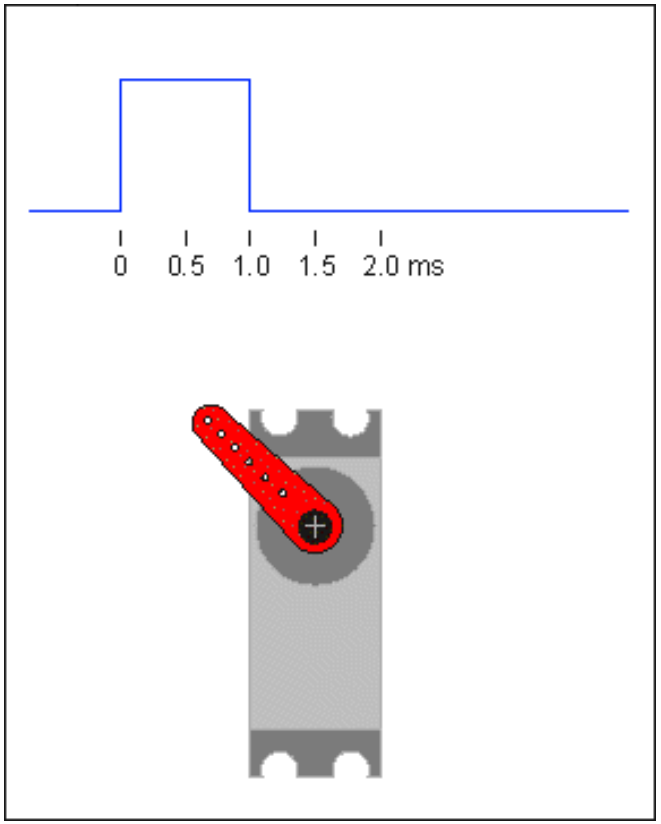
Durch einfache simple Integration kann ein PWM Signal in eine quasi analoge Spannung umgewandelt werden. Dabei sollte dann die PWM nicht zu niedrig gewählt werden, so dass der Integrator sauber glätten kann. *Belastung des Integrators und Filterungsqualität beachten.*





PWM an **SERVO**S aus dem Modellbau

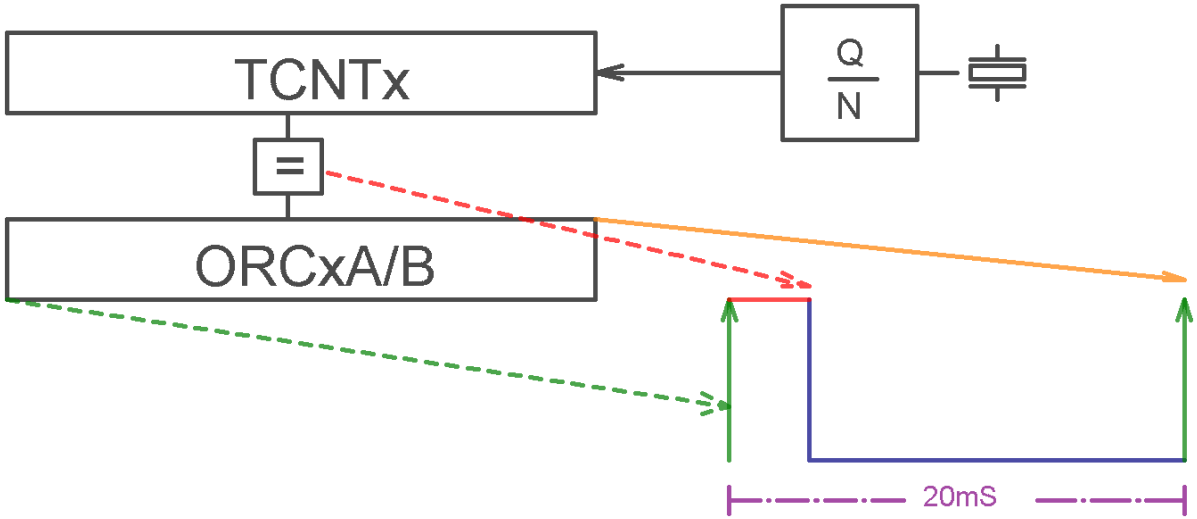




Servos haben eine PWM Steuerung
Dieser folgt mechanisch der PWM-ON Zeit.
1 mSekunde = links
2 mSekunden = rechts → Mitte = ~1.5mS
Die PWM = ~50Hz =1/T = 20 mS
Achtung: Anschläge unbedingt vermeiden
Merke: **Erst PWM a**nlegen, dann **danach erst V+**

Timer x für Servo-PWM Erzeugung nutzen.

Wir wissen, dass ein SERVO **PWM-ON** Zeiten von **~1mS bis ~2mS** *ohne mechanischen Anschlag* hat. Zudem sollte die PWM so um die **~50HZ** betragen.



Natürlich gibt es dafür genau die richtige Methode bzw. Modi.

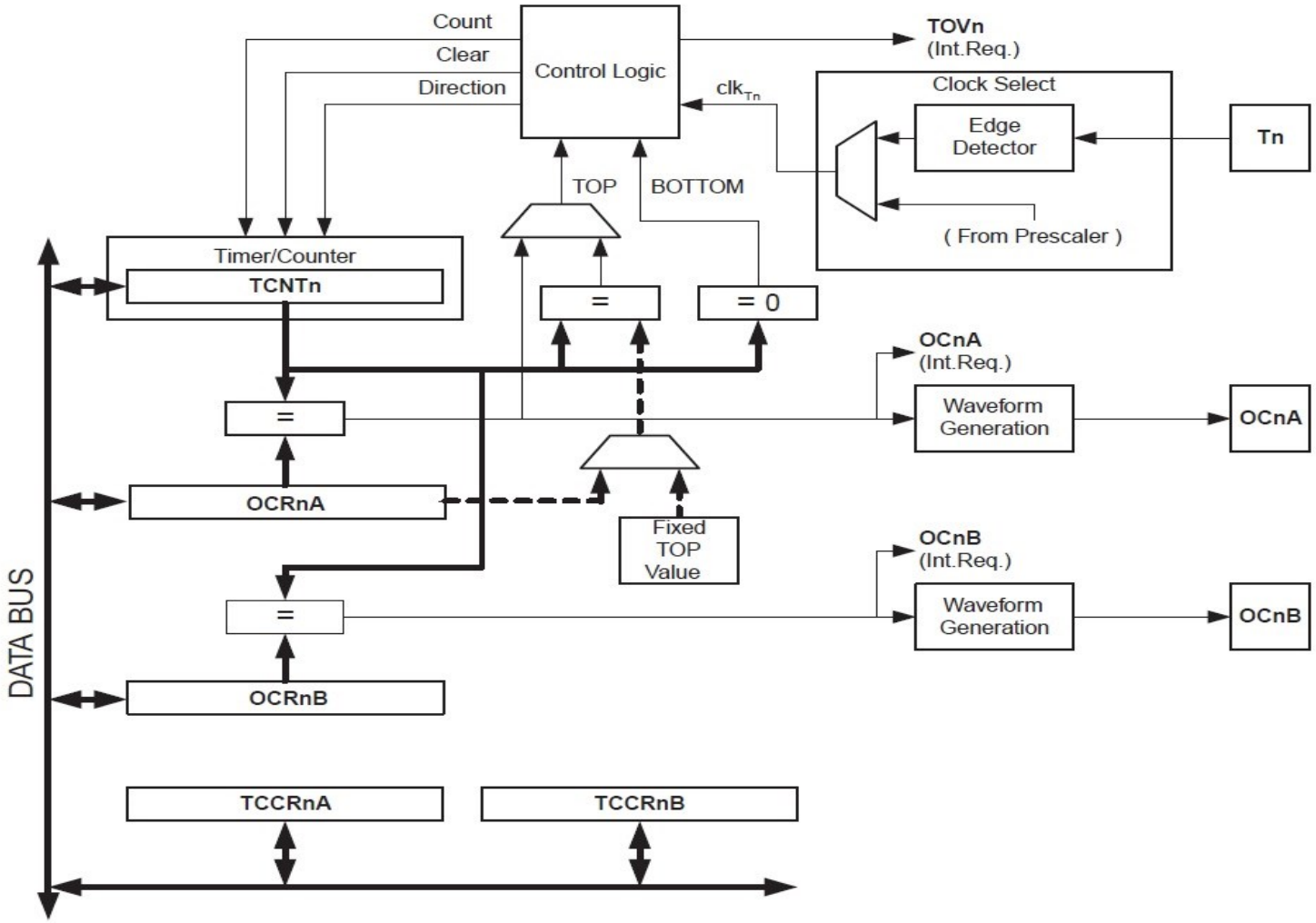
Mit dem OCRxA/B Register wird der 1. Zeitpunkt, der Wechsel am PIN **OCx** von **1→0** eingestellt.

Der **FAST-PWM** Modus sorgt dafür, dass im 2. Zeitpunkt, dem **Overflow-Moment** des Timers, der **OCx** Ausgang wieder von **0→1** wechselt.

16MHZ / 256{Timerdurchlauf} -> 62500/{50Hz} → 1250.
Wir haben einen Vorteiler von 1024 zur Verfügung.
62500 / 1024→61.035 Hz. Das ist genau genug für Servos.
Der **Vorteiler** ist also **1024** → 16.384 mS/Periode.
Servomitte = 1.5mS → 16.384/256= **0.064 mS/Bit**
1mS = 1/0.064 ->15.625 → runden: ~OCRx=**16**
1.5mS = 1.5/0.064 →23.4375 c→ runden: OCRx=**23 (Mitte)**
2mS = 2/0.064 → 31.25 → runden: OCRx=**31**.25
Servo-Range ist also von 16..31 +/- ausprobieren:

Timer 0 Übersicht..

Figure 14-1. 8-bit Timer/Counter Block Diagram



Timer 0 - SFRs für PWM

TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-2. Compare Output Mode, non-PWM Mode

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	Toggle OC0A on Compare Match
1	0	Clear OC0A on Compare Match
1	1	Set OC0A on Compare Match

Erste Recherchen bringen uns wieder einen Satz SF-Register zu Tage……

Auch der Modus ist schnell ausgemacht: → Fast PWM
Action: Comparematch → TNCT0=0, Max Value

Table 14-8. Waveform Generation Mode Bit Description

Mode	WGM02	WGM01	WGM00	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on(1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM ★	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

→ **TCCR0A = _BV(WGM00) | _BV(WGM01) | _BV(COM0A1);**

TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 14-9. Clock Select Bit Description

CS02	CS01	CS00	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/(No prescaling)
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

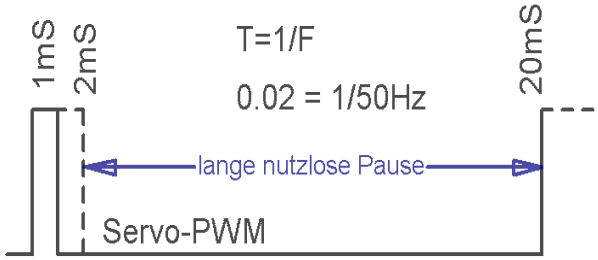
→ **TCCR0B = _BV(CS02) | _BV(CS00);** // 1024

→ Und jetzt noch: **DDRD |= _BV(PD6);** //OC0A

//16Mhz / 1024 / 256 = ~ 61 Herz

Die PWM Funktion in C

```
/ ************************************
void Timer0_PWM( uint8_t u8PwmRange)
// ************************************
{  // 8BIT TIMER T0
OCR1A = u8PwmRange;
// CLK_1024  FAST_PWM OC0_MODE
TCCR0A = _BV(WGM00) | _BV(WGM01) | _BV(COM0A1);
TCCR0B =  _BV(CS02) | _BV(CS00); // 1024
//16Mhz / 1024 / 256 = ~ 61 Herz
DDRD |= _BV(PD6); //OC0A
};
```




```
/*
Statt ewig (1<< xx ) zu schreiben wurde vorweg dies definiert:
 #define  _BV(x) (1<<(x))
*/
```

SOFORT nach Einschalten eine M**indest-PWM** von **> 1mS .. < 2mS** einstellen, sonst läuft der SERVO an den Anschlag.
Das kostet viel Strom und der Servo kann kaputt gehen auf Dauer.

Wie wir sehen, ist bei einem 8 Bit Timer mit ~61 Hz die Auflösung im Bereich 1..2m Sekunden mehr als bescheiden. OCR1A:

16=~1mS

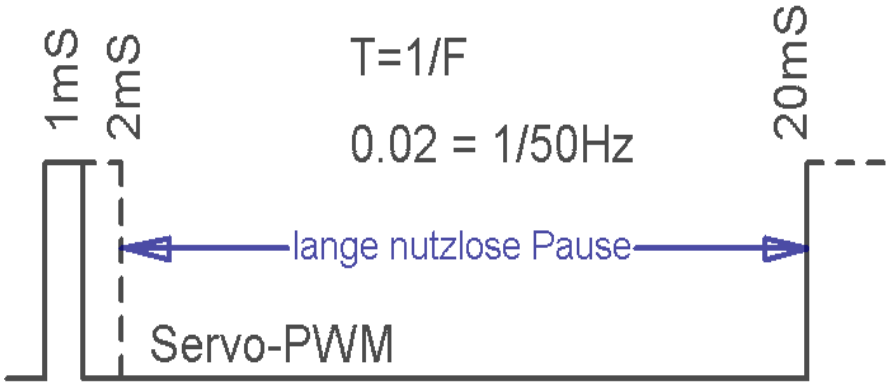
24 für ~1,5mS

32=2mS

Das geht nur besser mit einem 16 Bit Timer im 10Bit PWM Mode

… oder.. Plan B.

etwas Programmiergeschick.



Schön wäre es, die 8 Bit=256 für ~1..2MS zur Verfügung zu haben.

Dann müssen wir nur geschickt die lange Pause überbrücken: 2mS..20mS

Der Timer müsste also für 2mS → f=1/t = 1/0.002=500Hz laufen und dann 9 Perioden pausieren. Wir suchen also mal nach Näherungs-Parametern:

16^6/256 /500 = Teiler **125** ..Pech….. Gibt es nicht!

Mögliche Teiler sind /1 / 8 /64 /256 /1024. Die erste Näherung ist ***/256..***

Also sehen wir mal, wie wir damit leben können. T=16^6/256/256=~244

Es wird sinnvoll sein, darüber nachzudenken, ob wir den Compare Interrupt schlauer nutzen können. Darin könnte man bis **5 z**ählen **(244 / 5 = 48.8Hz)** und bei 0 den Port freigeben.. So etwas in der Art. Einfach mal ausprobieren….

So findet man Lösungen und Erfahrungen.

TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Ovl. und CompA. Interrupt . aktivieren… TIMSK0 = (1<< OCIE0A) | (1<< TOIE0); //OVL + COMPA

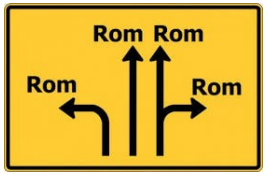

```
ISR( TIMER0_OVF_vect )    //Dieser Vector steht in der „iom328p.h“
{  volatile static uint8_t u8T0_PWM_Cnt=0;
if( !((++u8T0_PWM_Cnt % 5) ) )          // ~244/5=48.8HZ
    { PORTD |= (1<< PD6);               // OC0A=1, Nur  alle 5 MAL
    };
};
ISR( TIMER0_COMPA_vect ) //Dieser Vector steht in der „iom328p.h“
{ PORTD &= ~(1<< PD6);    // OC0A };:

// jedes PORT-BIT kann eigentlich verwendet werden
void Timer0_PWM_Start( uint8_t u8PwmRange) //Normal Mode
{
OCR0A = u8PwmRange;
DDRD |= (1<< PD6); // OC0A
TCCR0A = 0;  // simple Mode
TCCR0B = (1<< CS02); // clk/256 = ~244HZ´→ 16^6/256/256=~244
TIMSK0 = (1<< OCIE0A) | (1<< TOIE0); //OVL  + COMPA
};
```

Jetzt braucht nur **OCR0A** verändert werden:
Tb=Zeit/Bit = 1/ (16^6 /256)

1mS= 0.001 / Tb= 62.5
1.5mS=0.0015 / Tb=92.75
2mS = 0.002 / Tb =125
OCR0A = 93; ist also Servo-Mitte

MERKE: TIMER wird nur im SIMPLE MODE betrieben.
Jedes beliebige Bit könnte genutzt werden.

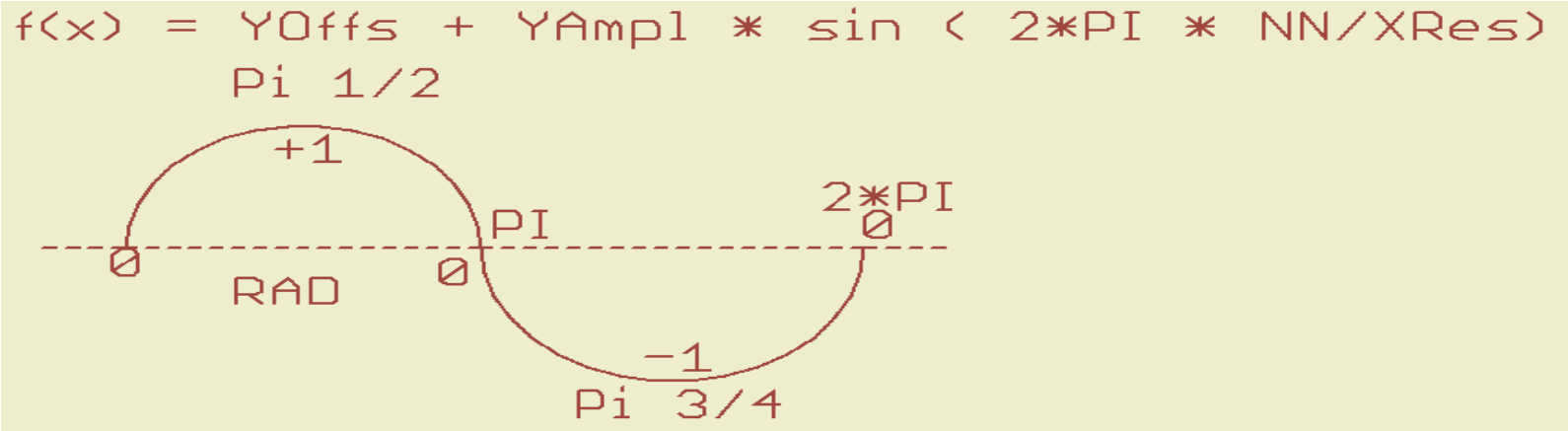
Eine andere Technik.. Tick Interrupt nutzen.
Multi-PWM möglich, zudem völlig freie Portnutzungen möglich.




```
unsigned char gu8PWM_D11_Val = PWM_MITTE;                    // Compare PWM

/* uses timer2 (36kHz for IR communication */
/* counts falling and rising edge => 36kHz*2 = 72kHz */

ISR( TIMER2_COMPA_vect )              //→ 8MHz Quarz
{
gu8Count72kHz++;
//gu8PWM_D11_Val trägt die PWMZeit für Servos

if( (! gu8Count72kHz) && !( gu32_Ticks % 5) ) //281/5 so wird ~55HZ PWM draus.
            {                         //nur alle 5*
            SERVO_D11_HIGH();         //MAKRO PWM PIN +5V
            };

if( gu8Count72kHz == gu8PWM_D11_Val)
            {
            SERVO_D11_LOW();          //MAKRO PWM PIN +0V
            };

If ( !gu8Count72kHz )   // TEILER / 256                //! bedeutet: wenn 0 dann  =
//~282  // Zählt ~8MHz / 111 = 72000/256=281.5315315
            {
            gu32_Ticks++;
            };
};
```

```
#define PWM_1MS     72
#define PWM_2MS     144
#define PWM_MITTE 108
```

Sinus Formel – Prinzip

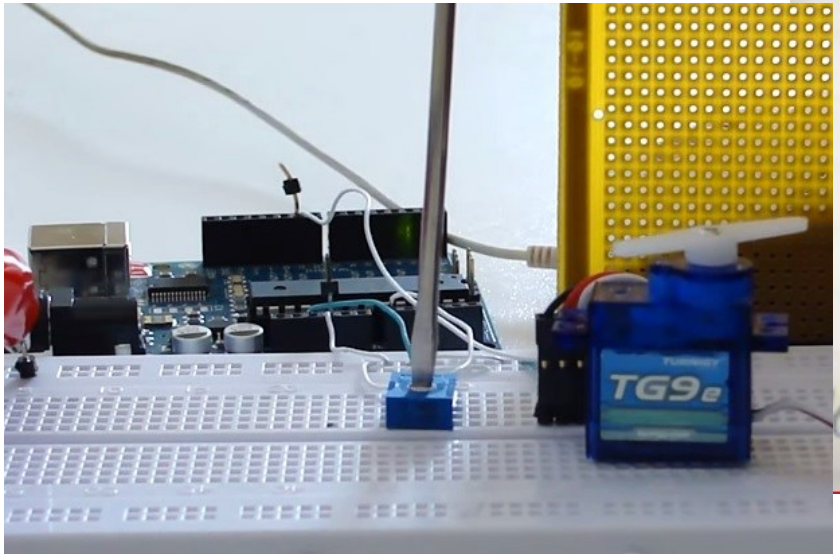


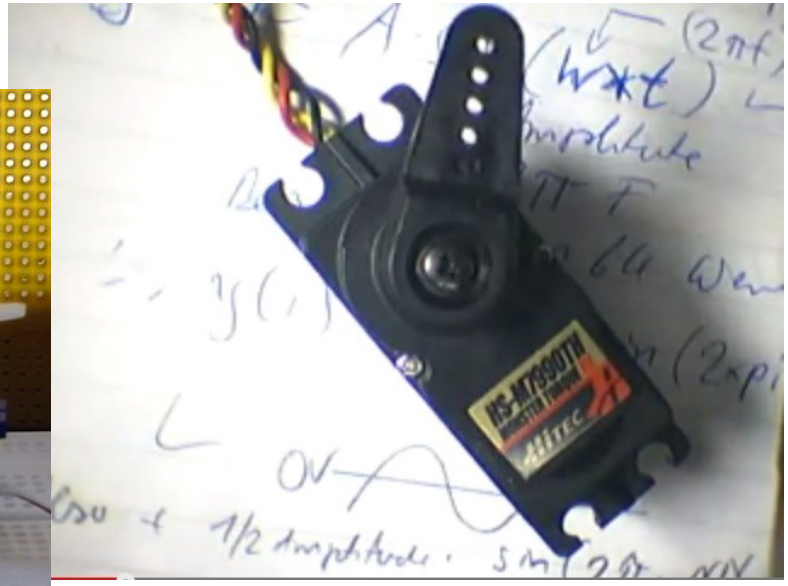
Ein Sinus geht von 0 bis 2Pi zwischen Plus und Minus.
Wir wollen aber ganze positive Zahlen.
Also muss der Offset addiert werden.
yR= resolution Y
xR=resolution X
fx= ½yR + ½yR * sin(2* Pi * n/xR) //n=0, n++, n<=xR

Youtube Video des Beispiels

Sinus Servo
https://www.youtube.com/watch?v=wDE_EwiImvc

Sinus LED
http://youtu.be/zxT49ZjLUkE





Sinus Formel - allgemein

- f(x) = a * sin (2*Pi*x - b) + o
- a ist die Amplitude
- b ist die Phasenverschiebung
- o ist der "Offset"-Wert
- für x setzt du halt n/d ein. (n € N)
- d ist die Schrittanzahl pro Periode

Sinus – Formel,

Erster Lösungsansatz:

$f(x) = Y_{Offset} + Y_{Resolution}/2 * \sin(2 * Pi * n+1/X_{Resoultion})$;

- Xn+1/XResolution ist ein Multiplikator Quotient der 0..2*Pi mit (0… <1) abdeckt.
- Merke: sinus() liefert Faktoren im Bereich -1 bis +1.

Praktisch sieht es für einen Byte-Wertebereich jetzt etwa so aus.

Das prüfen wir mal bei Gelegenheit…..

- F(x) = 128 + 127* sin(2*M_PI * iNN/256)

für iNN → 0..255

// 127, damit die Werte innerhalb des Bytebereichs bleiben.

So realisiert man z.B. eine Sinuswerte-Array-Tabelle

```
void Calc_SinusTable( uint8_t u16OffSet, uint16_t u16Amplitude, uint16_t u16XAufloesung )
{
uint16_t u16NN;
//f(x)= OFFSET + Amplitude * sin ( 2* M_PI * NN/X_AUFLOESUNG )
for( u16NN = 0; u16NN < u16XAufloesung; u16NN++)
        {
        gu8aKurve[u16NN] =  round( u16OffSet + ((u16Amplitude/2.0) - 0.5) *        \
        sin(2*M_PI* u16NN/u16XAufloesung) -0.5 );
        };
};
```


```
//Anzeige der Werte bsp. im mainloop…

if( (gu32_Ticks - u32Now) > PULS_500MS_Tick )
                {
                u32Now = gu32_Ticks; //Zeit merken !!
                PORTA = (1 << gu8aKurve[ gu8KurvePos++ ] );
                };
```

Servo-Sinus Software... Mehrere SW-Module sind erforderlich.

<u>1. Tick-Interrupt:</u> Erweitern für schnellere Berechnung

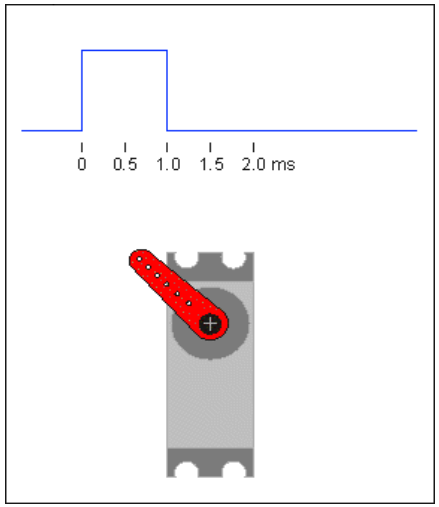
```
 if( ( (gu32_Ticks++) - u32PWM_T) > TICK_100MS )
            {  u32PWM_T = gu32_Ticks;   //neue Zeit merken
             gu8Status |= STATUS_PWM_EVENT;
            };
```

<u>2. Funktion:</u> #define XRES 64.0

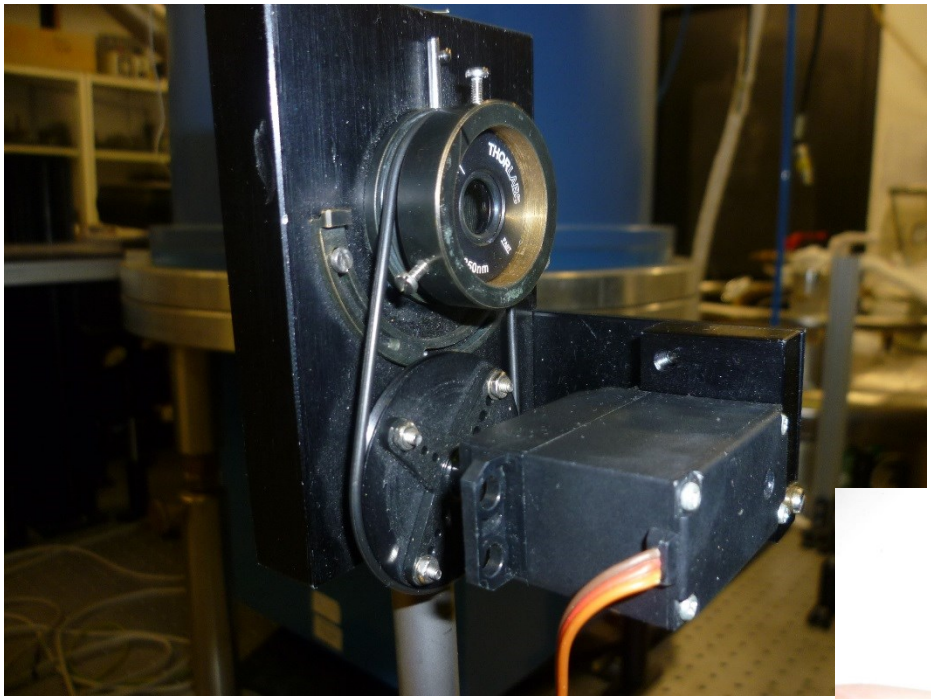
```
uint8_t Sinus_Servo(uint8_t u8XResN )
{return  PWM_OFFSET + HALBE_PWM_RESOLUTION + HALBE_PWM_RESOLUTION
* sin( 2* M_PI * u8XResN/XRES );
};
```

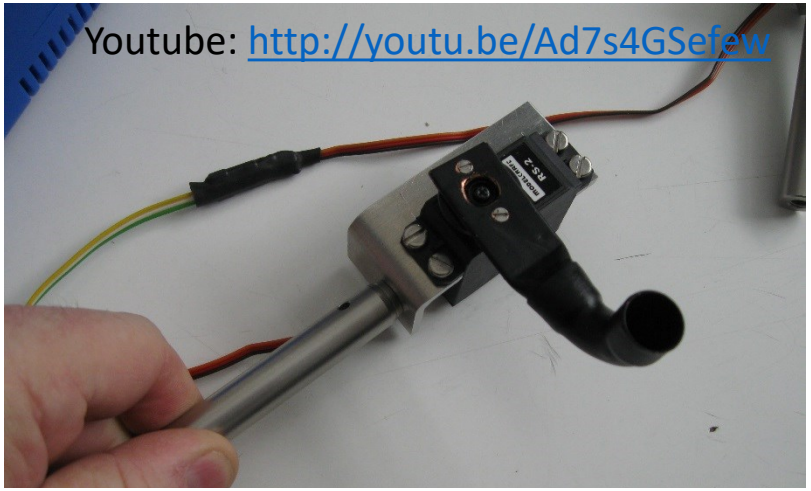
<u>3. MAIN-Loop:</u>

```
if( gu8Status & STATUS_PWM_EVENT)
            {
            gu8Status &= ~STATUS_PWM_EVENT;
                 // Hier die einzelnen X-Schritte berechnen
            OCR0A = Sinus_Servo( u8NN % (uint8_t)XRES );
            u8NN++;
            };
```



Servo Anwendungsbeispiele

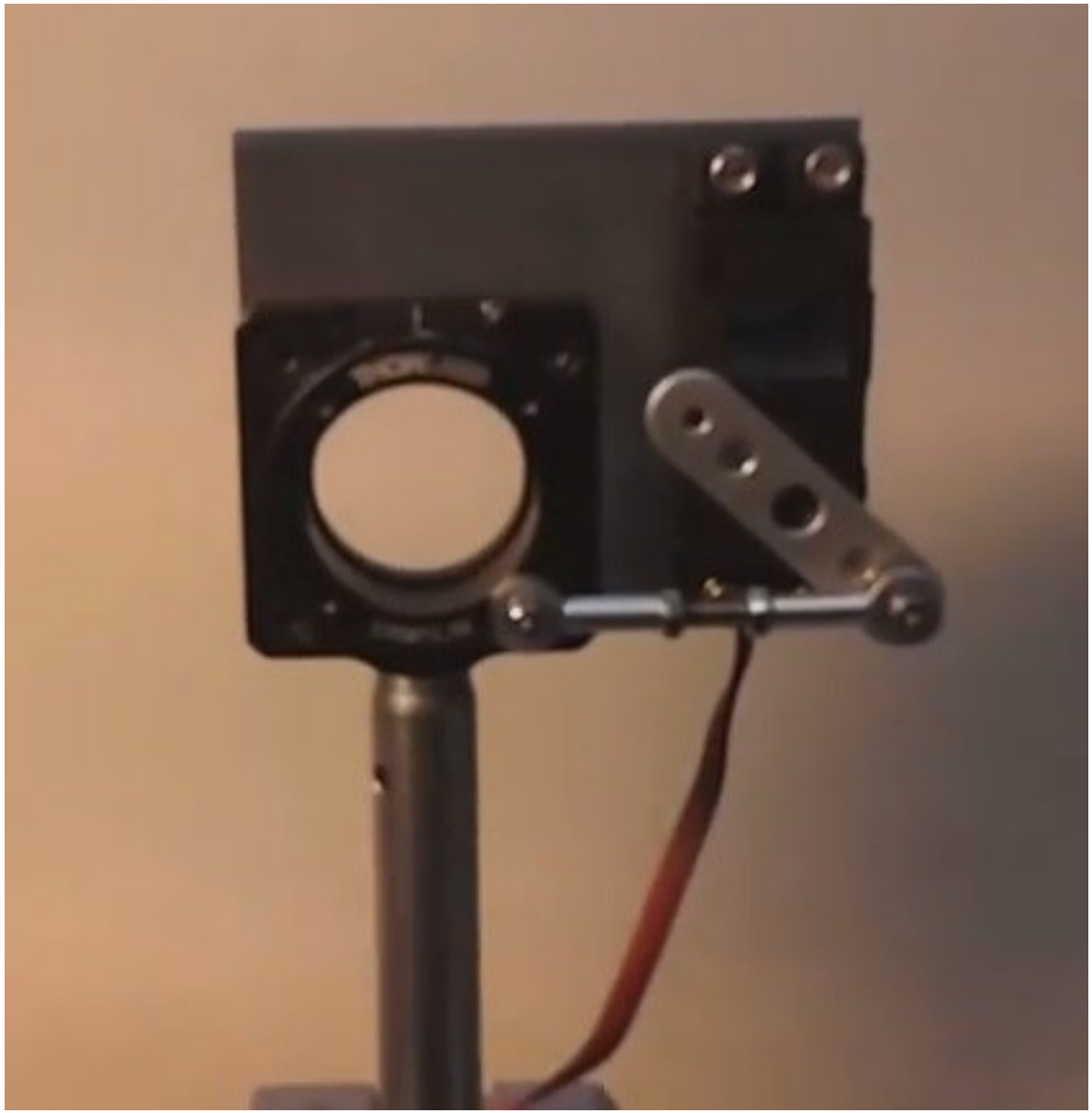




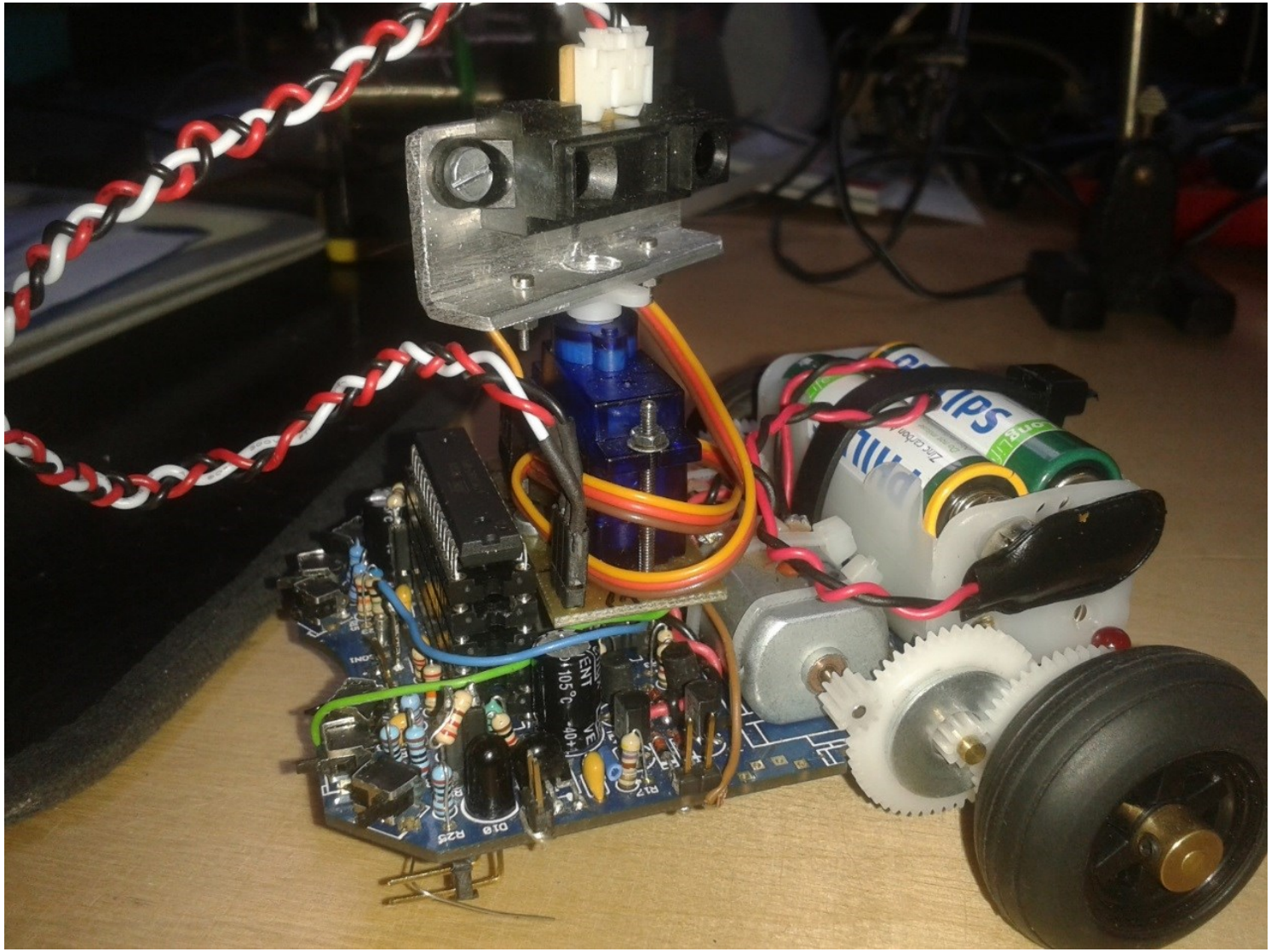
Servo **dreht** im Laserlabor ein Laser Lambda/4 Blättchen.

Laserfangrohr an ein Servo montiert, das sich auf einen textbasierten Befehl so dreht, dass der Laser in das Rohr fällt und sich darin verliert.

Youtube Link: http://youtu.be/yundxoedF1s



Nochmal der ASURO aus dem Mint Girl Projekt 2014
Servo Motor dreht bzw. schwenkt einen IR-Entfernungsmesser wie ein Radar.
(im Asuro DLR Robot) Youtube: http://youtu.be/dstx46pDZzQ



Bitmuster-Ausgabe & const char-Musterspeicherung im Flash-Speicher

Eine wichtige Besonderheit im Umgang mit Mikrocontrollern, die oft nicht bewusst oder verstanden ist und deswegen oft falsch gemacht wird, ist, dass „**Konstanten**“ wie **Strings** „Abcd..“, bzw. beliebige Zeichenarrays (Bitmuster) {0x3A,0x39,0x35,0x36} **nicht im RAM, sondern im Flash** = Programmspeicher abgelegt werden sollen. Dazu wird ein im Hintergrund recht kompliziertes #define namens PROGEM bei der Deklaration mit angegeben.

Macht man das nicht, legt der Compiler Const-Strings nicht nur im Programmspeicher sondern auch im **wertvollen RAM** ab.
<u>Beim PC ist das egal. Im Mikrocontroller jedoch nicht</u>:
Es gilt also in Zukunft:

const char MCA_STARTREK[] **PROGMEM = {**0x81,0x42,0x24,0x18,0x24,0x42,0x00**}**;
const char MCA_Blink[] **PROGMEM = {**0xAA,0x55,0x00**}**;
const char MCA_MYBRAIN[] **PROGMEM = „** I have no brain, use your own !**“**;

„PROGMEN“ Konstanten aus dem FLASH-Memory lesen

Da der Speicherzugriff auf den Flashspeicher ganz anders behandelt wird, als Speicherzugriffe aus dem Arbeitsspeicher (RAM), gibt es so gut wie alle Standardfunktionen für String-Behandlung DOPPELT.
Ein Blick in die Funktions-Dokumentation hilft weiter:
file:///C:/WinAVR-20100110/doc/avr-libc/avr-libc-user-manual/modules.html
Dort finden wir die **<avr/pgmspace.h>: Program Space Utilities**

Hier sind alle Sonder-Definitionen als „Funktions-Spiegelung“ aufgelistet.
Nur mit einem „_P“ angehängt.
Daran gewöhnt man sich schnell…. Man muss es nur wissen..
Bsp.: die RAM-Version von **strcpy()** heißt: **char * strcpy(char * s, char * s)**

Um nun String aus dem PROGMEM zu kopieren, nimmt man:

char * strcpy_P (char *, PGM_P)

So gut wie alle Funktionen haben einfach ein **„_P“** → funkname_P() angehängt.

Wichtige PROGMEM-Defintionen:

Es tauchen in den **<avr/pgmspace.h>: Program Space Utilities** zudem noch **neue**, zugegeben etwas *verwirrende* , **#defines** auf.
Davon jetzt kurz die wichtigsten:

keine

#define PROGMEM __ATTR_PROGMEM__
→ damit werden die const arrays deklariert.
Anwendung: **const char** MCA_RAUMSCHIFF[] **PROGMEM = „NCC 1701-D“**;

#define PSTR(s) ((const PROGMEM char *)(s))
→ damit erzeugt man unverzüglich ein const array und den Zugriffs-Pointer darauf. Also der Programm-Memory-Pointer (***PGM_P***)
→ Bsp: int strcasecmp**_P** (const char *, PGM_P) __ATTR_PURE__
Anwendung: if(!**strcasecmp_P(** gsCmd.ucaCmd, **PSTR(**"PING"**)))** {};

#define PGM_P const prog_char *
→damit wird ein *const char Pointer* erzeugt, der auf den PROGMEM zeigt.
Anwendung: **PGM_P** gppgmPattern; → gppgmPattern = MCA_MUSTER;

#define pgm_read_byte_near(address_short) __LPM((uint16_t)(address_short))
→ zugriff zum byteweisen Lesen aus dem Progmem.
Anwendung: PORTC = pgm_read_byte_near(**(PGM_P)**(gppgmPattern + (gu8Parser % n)));

Übung zu Progmem String & Arrays

Wir wollen **Texte/Zeichen/Muster** zur mehrfachen oder einfachen Verwendung im Flash=Programmspeicher ablegen und gezielt auslesen, evtl. periodisch wiederholt. So kann man stetig sich abwechselnde festgelegte Bitmuster ausgeben.

Wenn man dies kann, kann man alles andere auch!

Ich verspreche nicht zu viel, dass damit grundlegend der Schrittmotorantrieb realisiert werden kann. Aber erst mal ein Beispiel:

const char MCA_STARTREK[] **PROGMEM** = {0x81,0x42,0x24,0x18,0x24,0x42,0};

Wir beobachten, dass zwei „**1**“en von außen nach innen und zurück laufen.

10000001 Lesen:

0**1**0000**1**0 Byte=pgm_read_byte_near(MCA_STARTREK + ((Parser++) % 6));

00**1**00**1**00

000**11**000

00**1**00**1**00 Das ist sehr komprimiert. Letztlich ist MCA_STARTREK ein Pointer zu

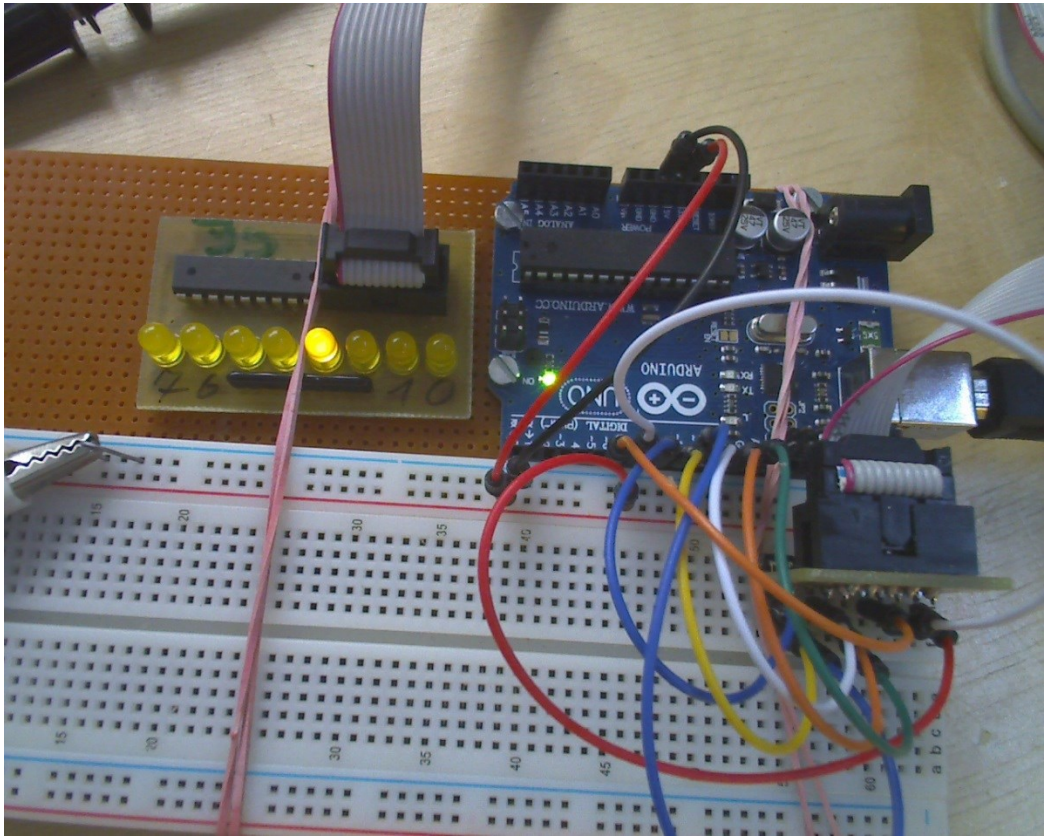
0**1**0000**1**0 dem byteweise ein Offset addiert wird. (**M**emory **C**haracter **A**rray)

10000001 So einen „Fingerzeiger“ nennt man Parser.

Mit dem Modul „% 6“ wird der Wertebereich 0..5 eingeschränkt.

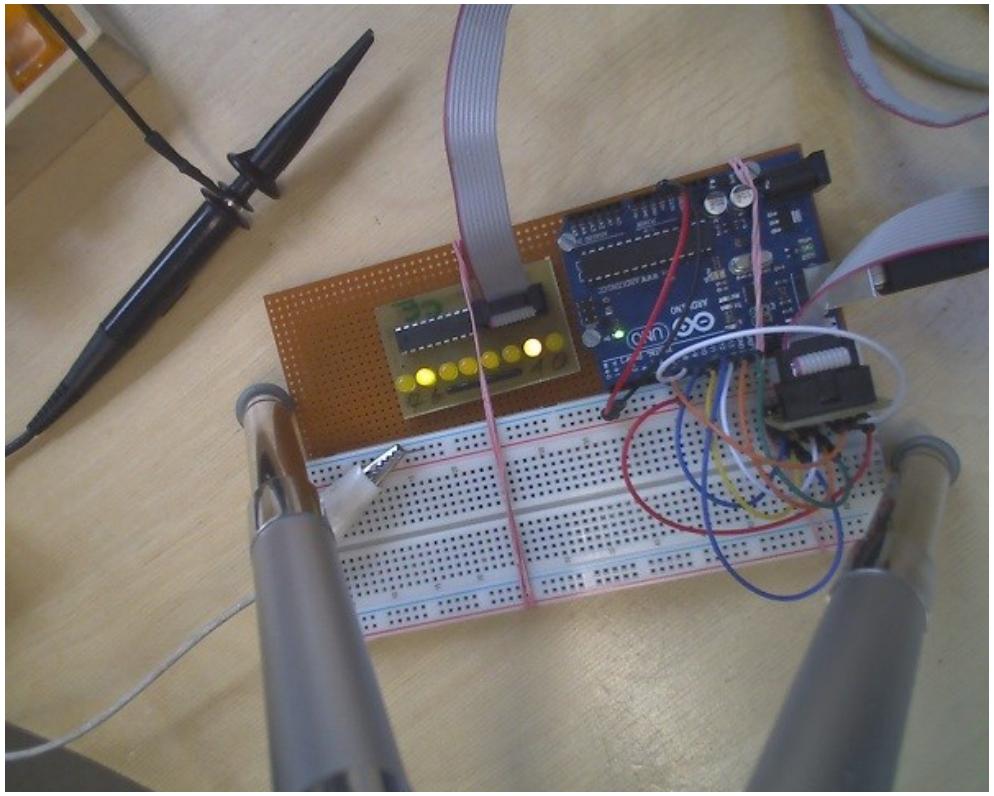
Youtube: http://youtu.be/3ELvAC1iKjg LED Knight Rider:

```
const char MCA_KNIGHT_RIDER[] PROGMEM =
{0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x40,0x20,0x10,0x08,0x04,0x02,0};

PORTx = pgm_read_byte_near( MCA_KNIGHT_RIDER + ( (gu16Parser++) % 6) );
```



Youtube: http://youtu.be/bCqgU04aVAY LED-Lauflicht

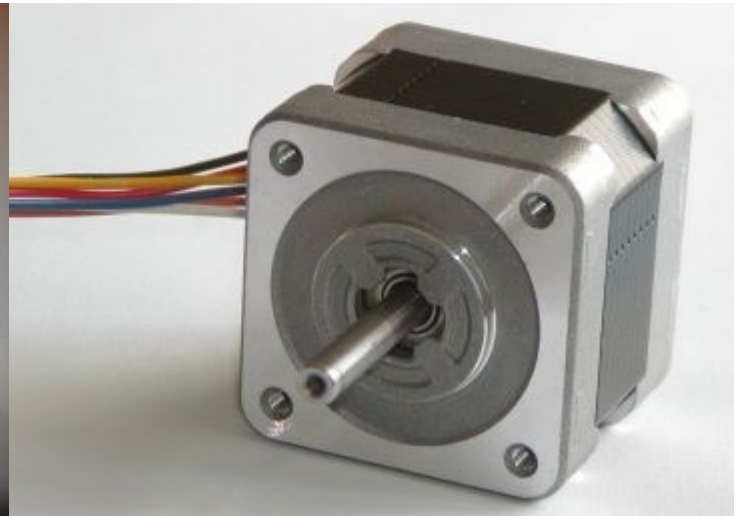
```
const char MCA_STARTREK[] PROGMEM = {0x81,0x42,0x24,0x18,0x24,0x42,0};

PORTx = pgm_read_byte_near( MCA_STARTREK + ( (gu16Parser++) % 6) );
```



Motoren

Bisher hätte man alles auch „zu Fuß“ mit diskreten Bauteilen realisieren können.
Richtig sinnvoll werden µController, wenn man diese für Motorantriebe einsetzt.
Dabei gibt es: ***Polwender-Schleifermotoren***,
BLDC = Brushless DC „Bürstenlose Gleichstrom-Motoren“ und
(Step) bzw. → **Schrittmotoren**.
Damit kann man präziseste Bewegungen realisieren.
Nicht unüblich sind **200 Step/Rotation**.
Und jede einzelne Position kann gezielt angefahren werden. Links wie rechtsherum!

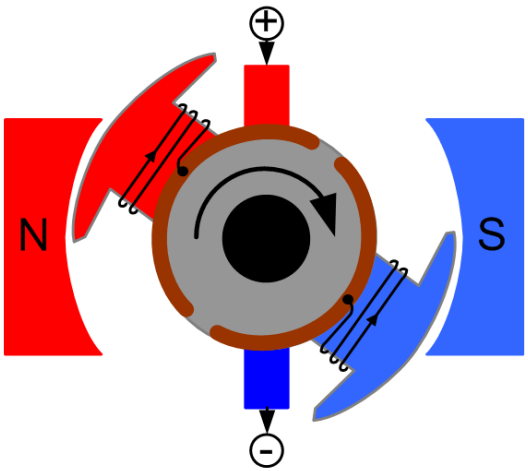
Lineare Kleinmotoren



typischer 6-Draht Schrittmotor

Polwender DC - Motoren

Sie sind die einfachsten Universalmotoren.
Diese sind vorwiegend **2-polig**.
Meist mit Dauermagneten im Stator.
Es gibt aber auch Reihen und Nebenschluss Statoren.
Meist etwas schwach, dafür mit relativ hoher Drehzahl. Diese gibt es aber auch mit recht knackiger Stromaufnahme und Drehzahl.

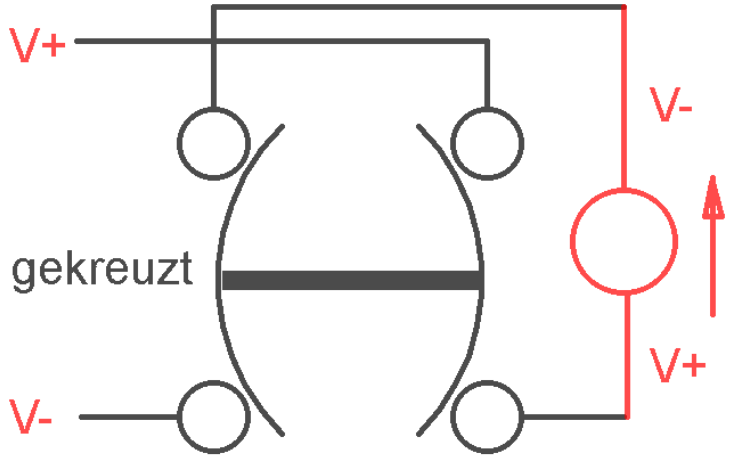


Die **Leistung/Drehzahl** kann hervorragend via **PWM geregelt** werden.
Die **PWM-** sollte nicht zu hoch gewählt werden (Induktionsdrosselung)

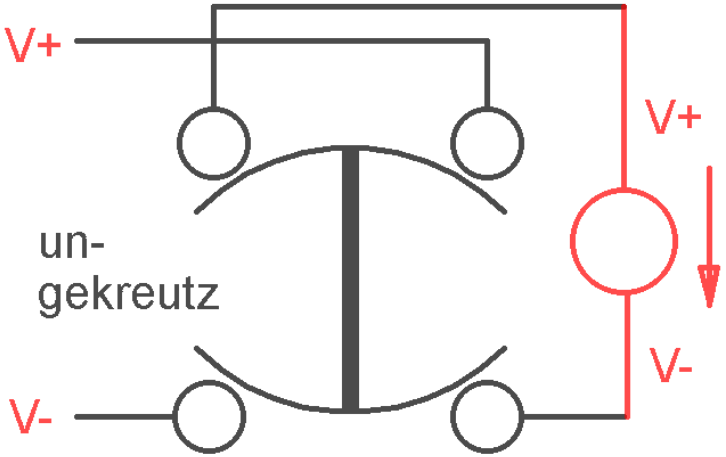
Die Drehrichtung wird von der **Polung des Stromes** bestimmt.
Deshalb braucht man einen „**Polwender**“, wenn man vorhat **links- und rechtsherum** drehen zu lassen. Das kann hervorragend eine **H-Brücke** Leisten.

Modellbau-Motor
Typischer Drohnenmotor

Polwendung / Umpoler der Stromversorgung mit mechanischen Mitteln

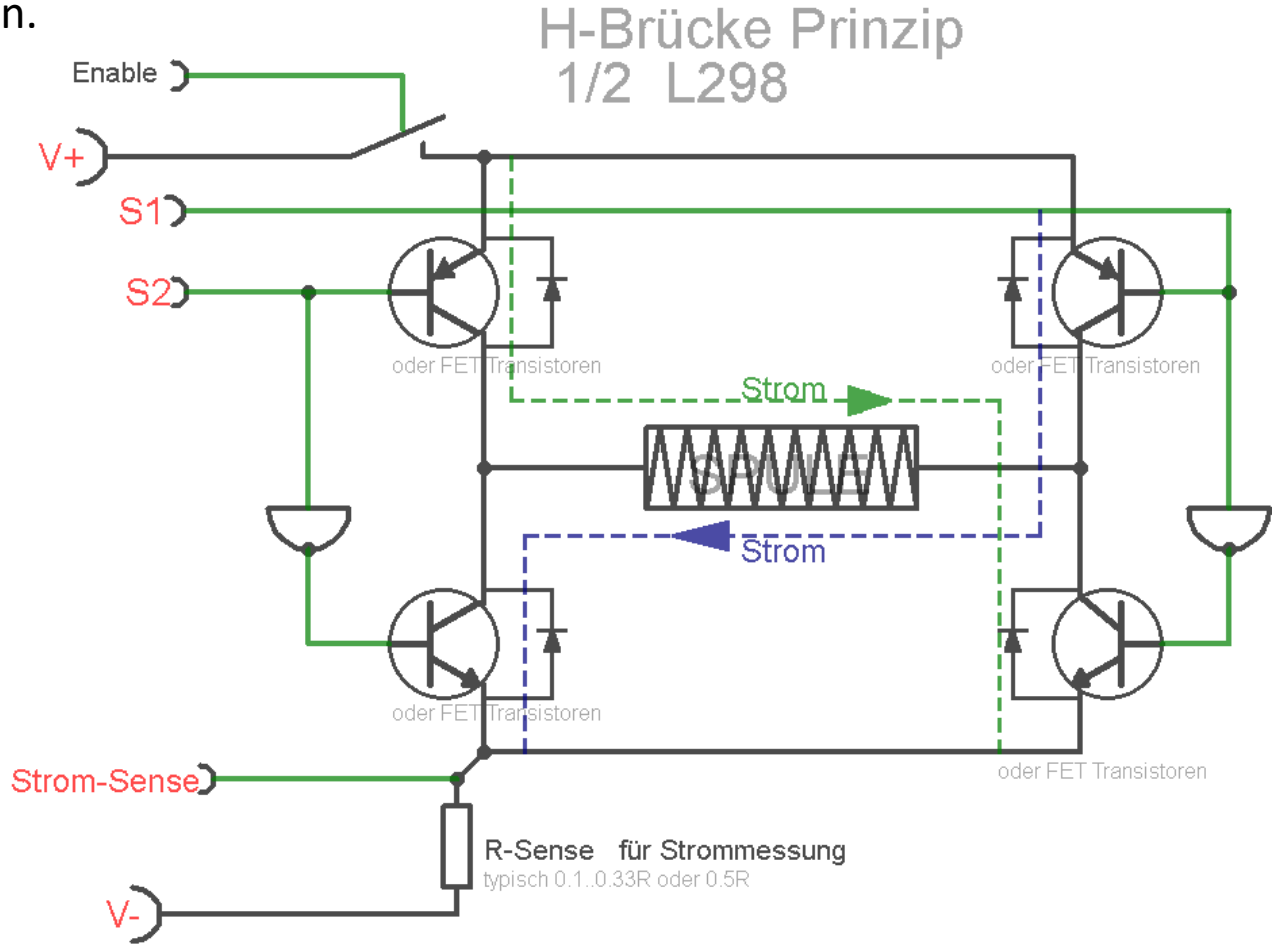


Bsp.: Ein alter mechanischer Bakelit/Porzellan Kreuzschalter aus den 40er Jahren, den ich mal gefunden habe, war sehr schlau gebaut. Mit einfachsten Mitteln wurde ein kurzschlusssicherer zuverlässiger Kreuzschalter (**Polwender**) gebaut.

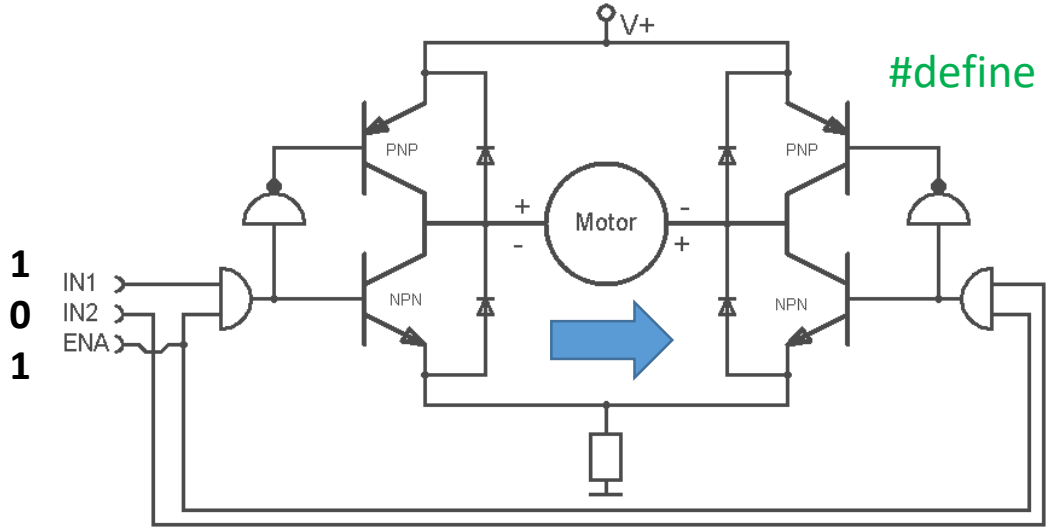


Die H-Brücke

Mit einer **H-Brücke** kann durch **überkreuztes** und **wechselseitiges** Einschalten der Transistoren eine **Spule/Motor** etc. beliebig gepolt be/durchstromt werden.



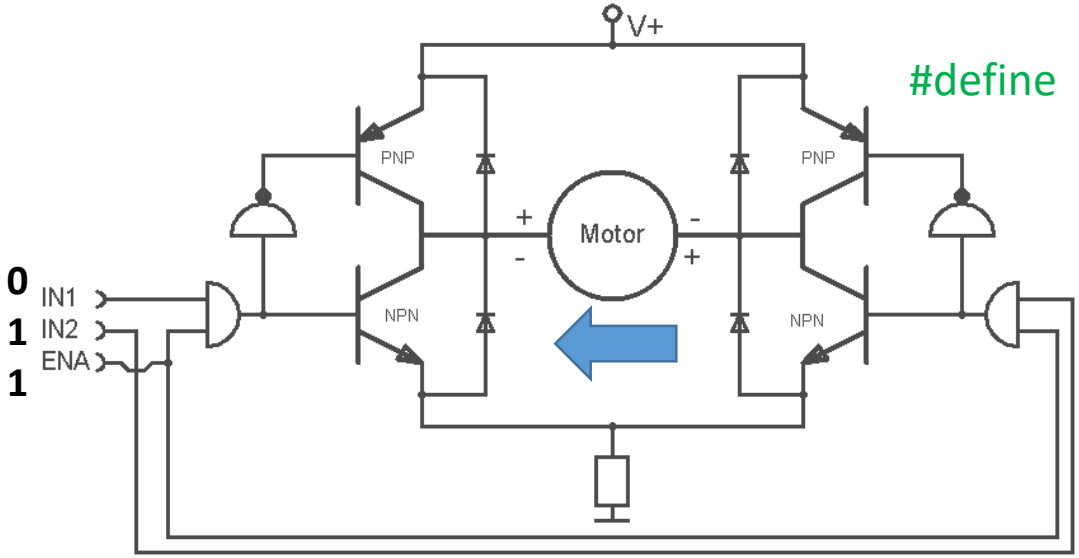
Polwender mit H-Brücke




```
#define   STROM_RECHTS  0x05
```

0b00000101

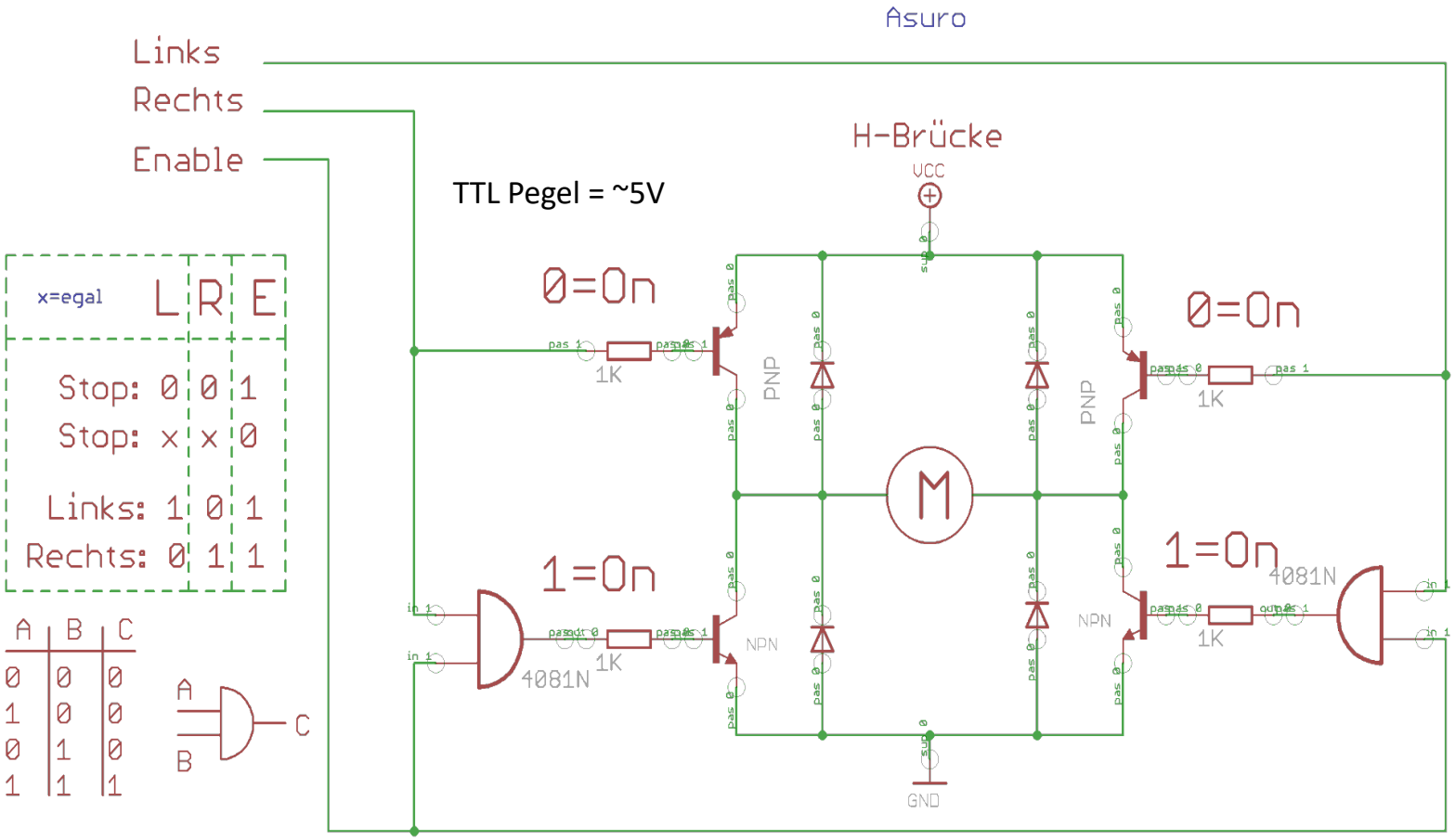
Enable könnte eine PWM Leistungsregelung haben




```
#define  STROM_LINKS  0x06
```

0b00000110

Reales Bsp.: So sieht die **H-Brücke** im Asuro-Robot aus.
Eine versehentliche Fehlschaltung ist durch die UND-Gatter verriegelt.
Die Freilauf-Dioden leiten die hohen Spulen-Schaltspannungen ab.
….meine Meinung: nicht besonders gelungen, aufwändig und nichts dahinter.
+5V Begrenzung. Da wäre ein H-Brücken IC sinnvoller → **L298N**



Hier eine **H-Brücke** aus **diskreten Transistoren**. Man beachte die stromgekoppelte automatische Kreuzung der Schalt-Transistoren, inclusive Freilaufdioden.

Steuerbits erlaubt:

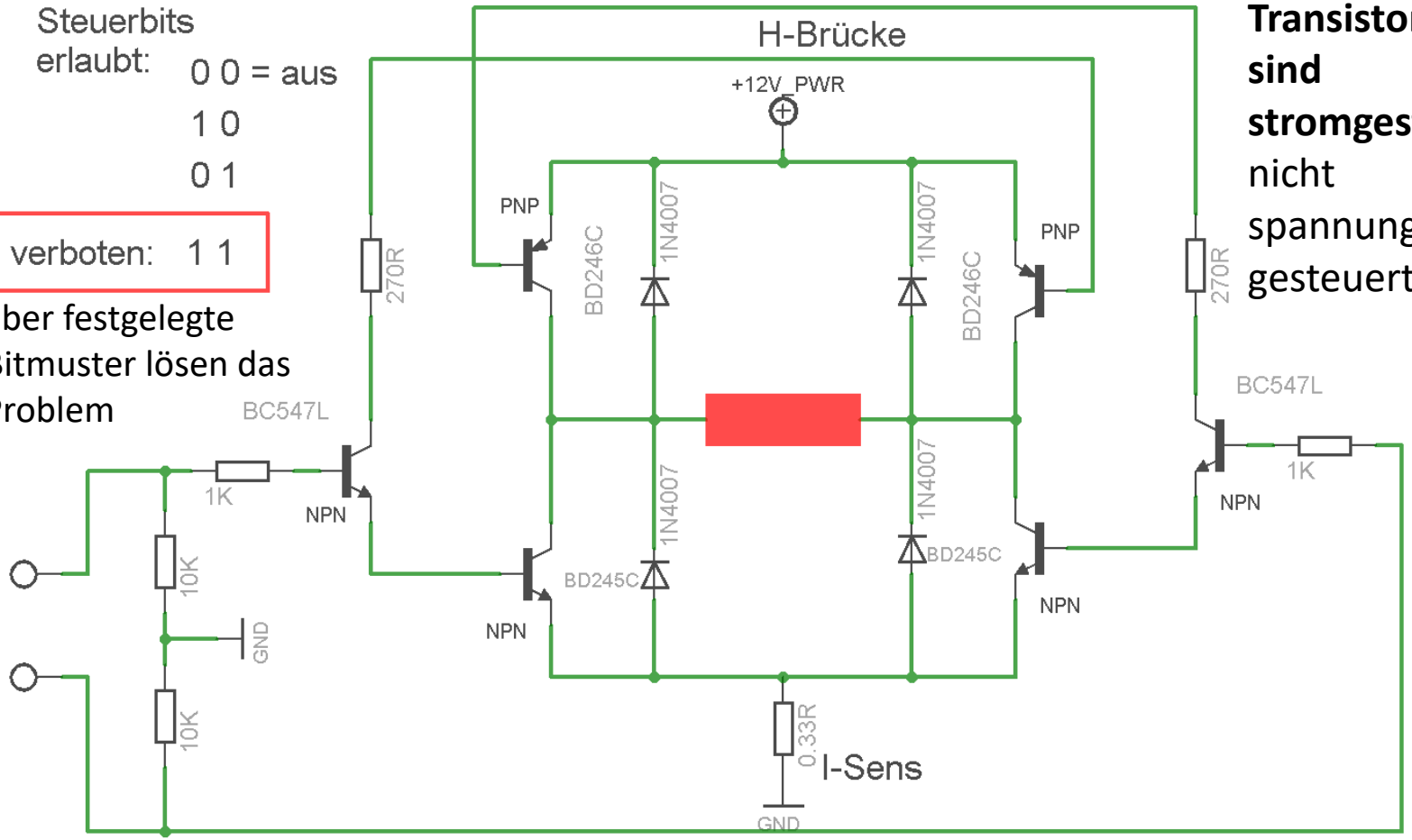
0 0 = aus

1 0

0 1

verboten: 1 1

aber festgelegte Bitmuster lösen das Problem

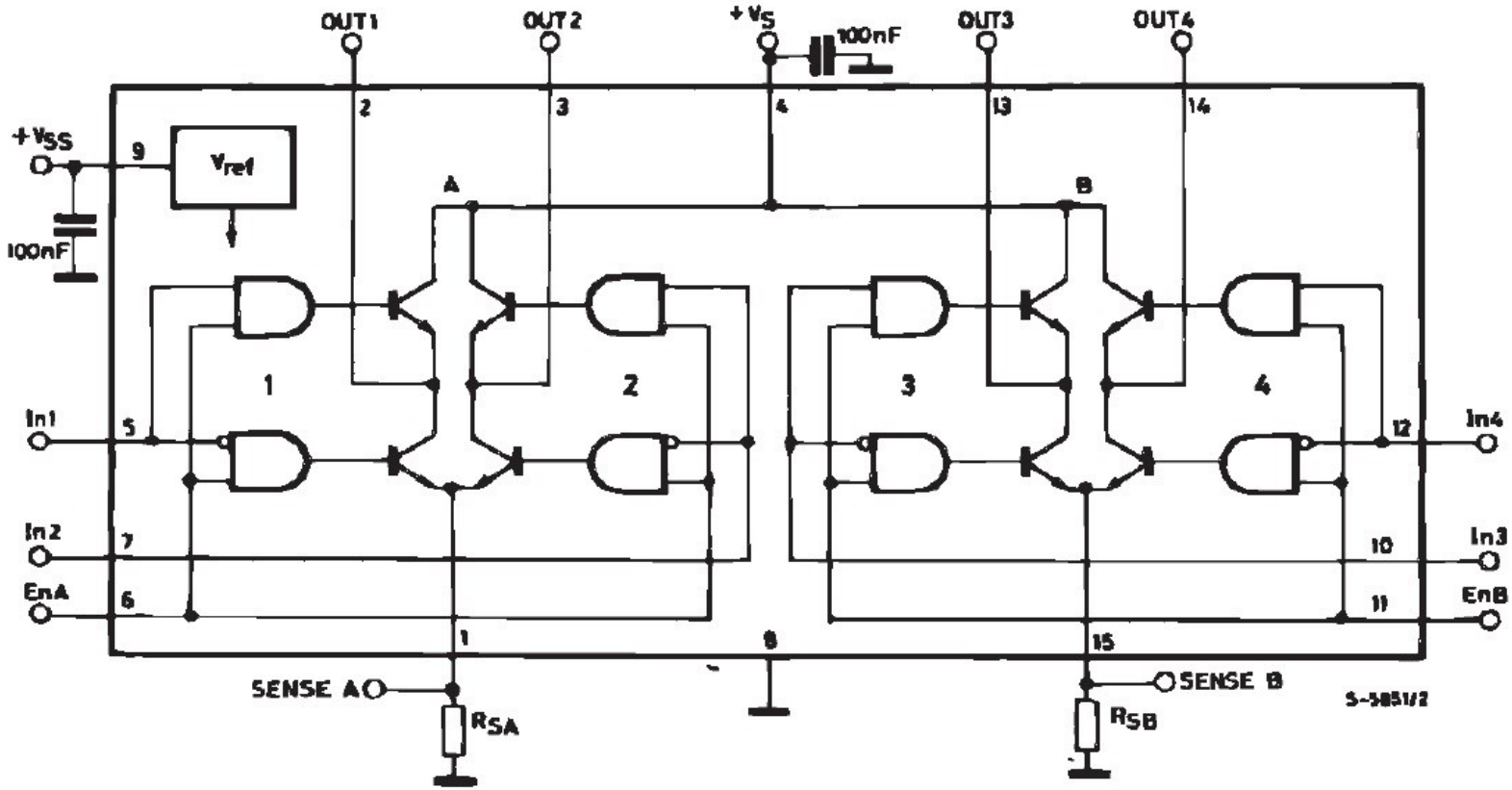


Merke:

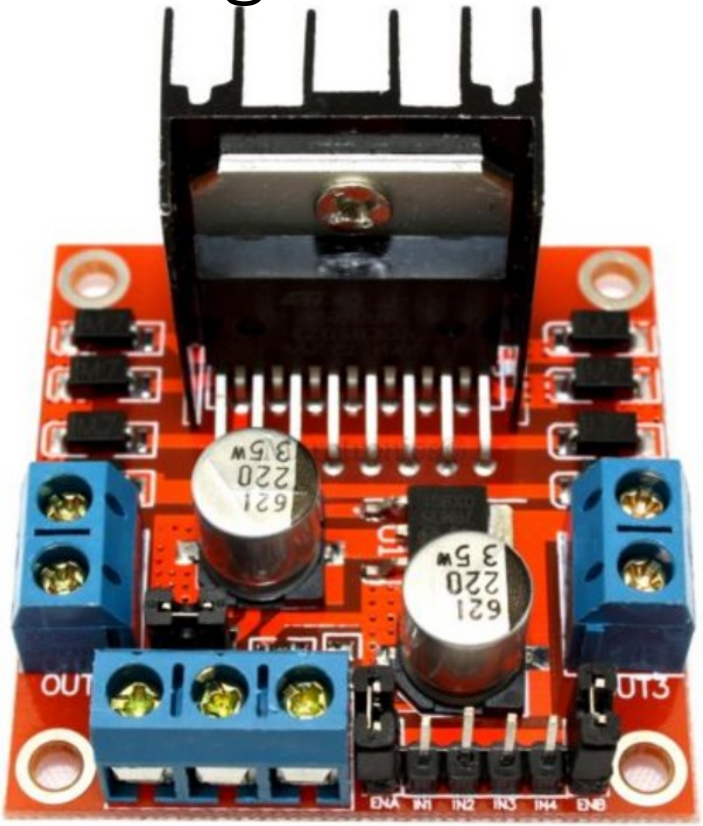
Transistoren sind stromgesteuert, nicht spannungs-gesteuert.

Dieser „verboten“ Fall „11“ könnte mit einem EXOR gelöst werden

Original L298 Dual H-Bridge Blockdiagramm



Fertige H-Brücken

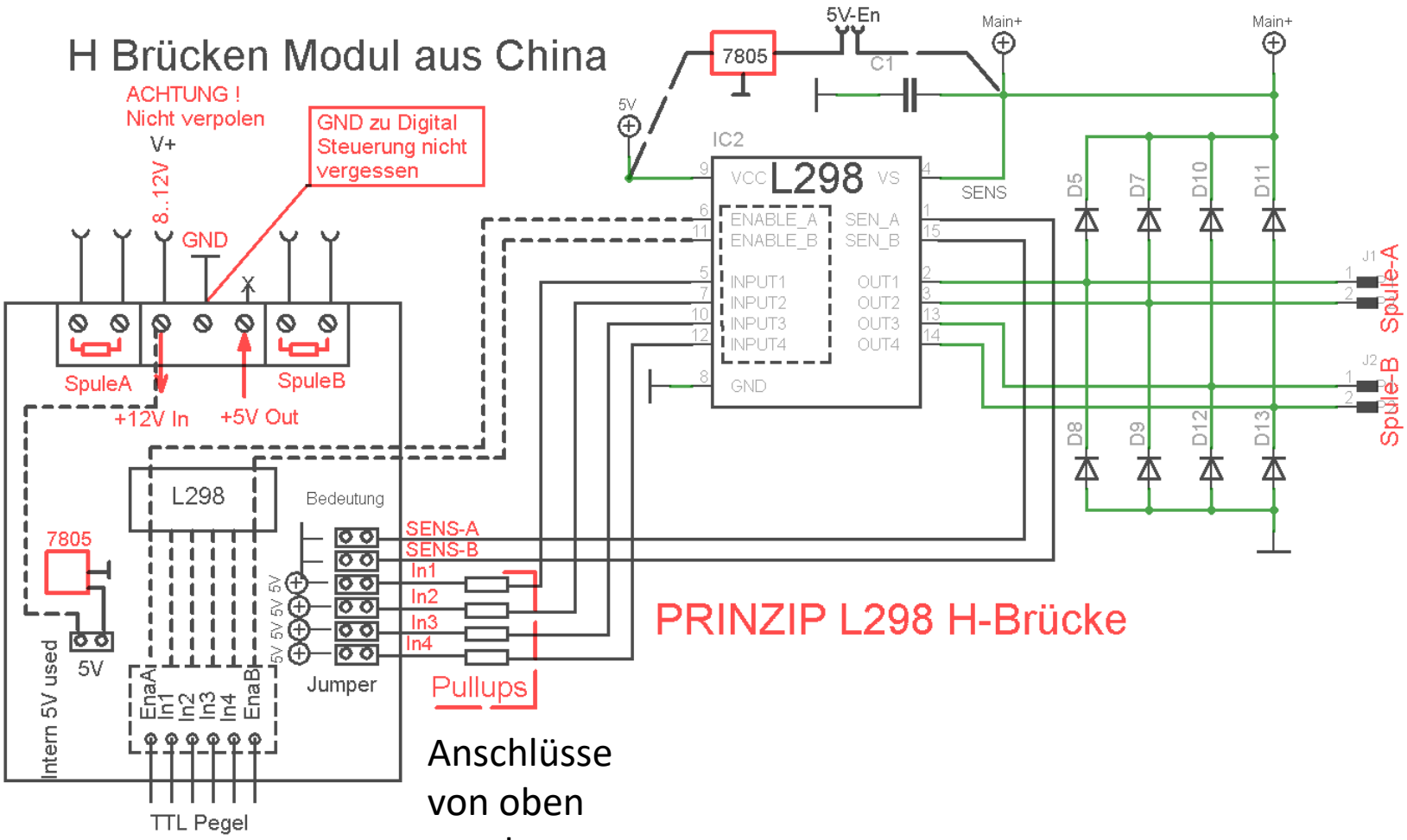


…gibt es Billig im Internet.
Dennoch muss man das Datenblatt zum verwenden H-Brücken IC lesen.

OBEN: Beliebte **L298N** 2* H-Brücke
(preiswert ~5..6€)

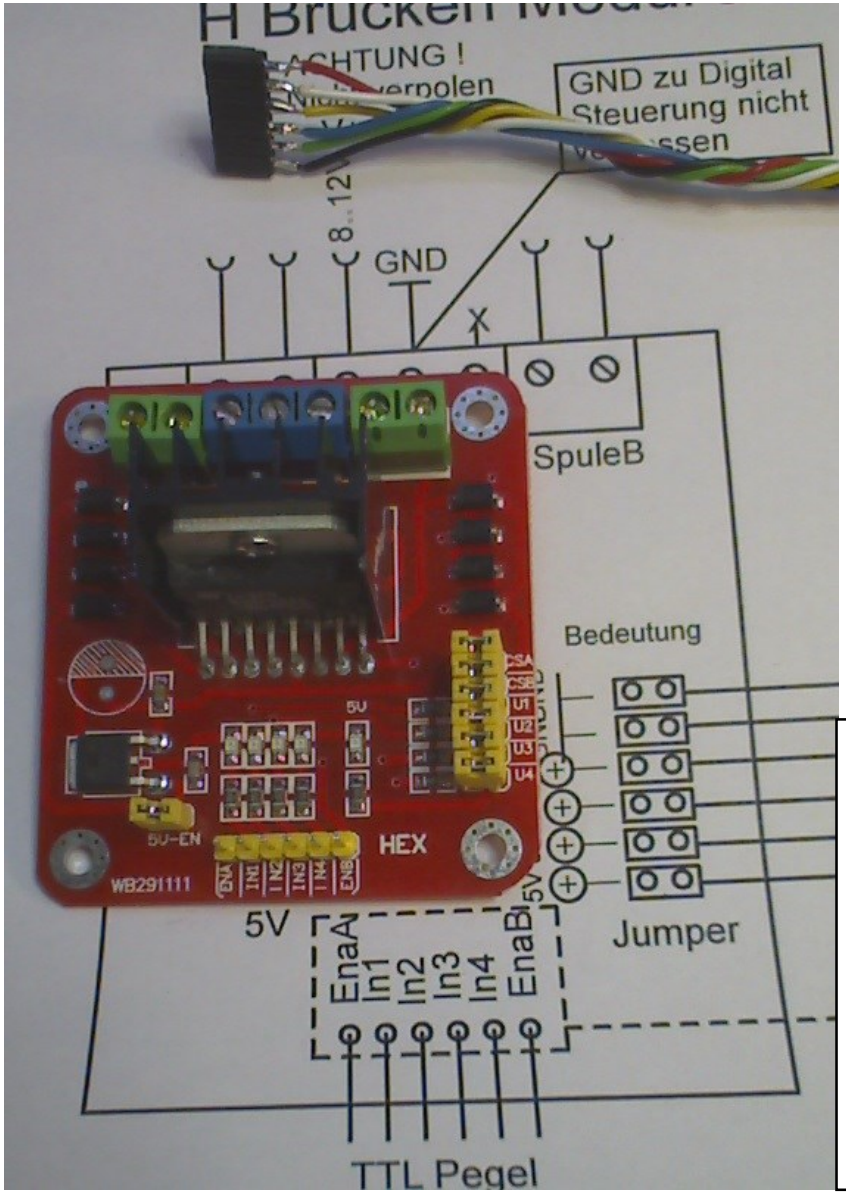
Rechts: Kräftige **BTS796**H 2* H-Brücke (teuerer 18€)

Für wenig Geld, ~5..10€, kann man sich ein fertiges H-Brückenmodul aus China kaufen, welches das **Doppel-H-Brücken-IC L298** enthält.



Anschlüsse von oben gesehen.

L298 H-Brücken Modul aus China, mit Pin-Belegungen. Oben, Spule **A-B**, externe **V+**



Basteln einer hohen Anschluss-Buchsenleiste mit 6 Drähten für die Signale IN[1..4] und Enable [A/B]

SAUBER !!
löten,
verdrillen.
Enden für
Steckbrett
5mm ab-
isolieren

Software für den DC-Motor

Wir können uns nun **modular** die Software zusammenkopieren.

1.) Wir benötigen einen **PWM** für die Motoren..
(beim ASURO habe ich eine 2-kHz PWM gemessen.. Das ist hoch, ging aber. (experimentieren! Aber mit Verstand und Messgerät → Oszi)
2.) Wir benötigen passende Bitmuster, die die H-Brücke, die wir verwenden, richtig schaltet.
Einfacherweise tut es ein #define …..oder doch besser gleich ein PROGMEM-Array . →

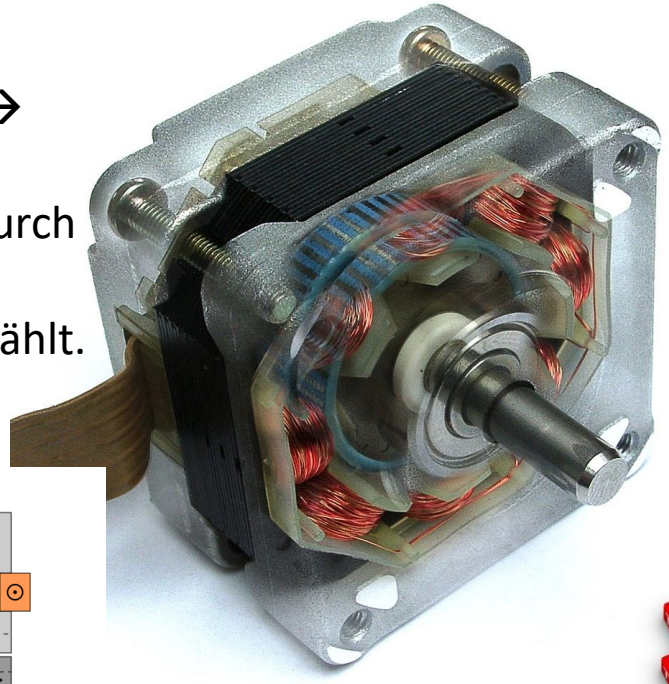
const char MCA_FULLSTEP PROGMEM = {0x35, 0x3A}; ← **11 1010**
11 0101

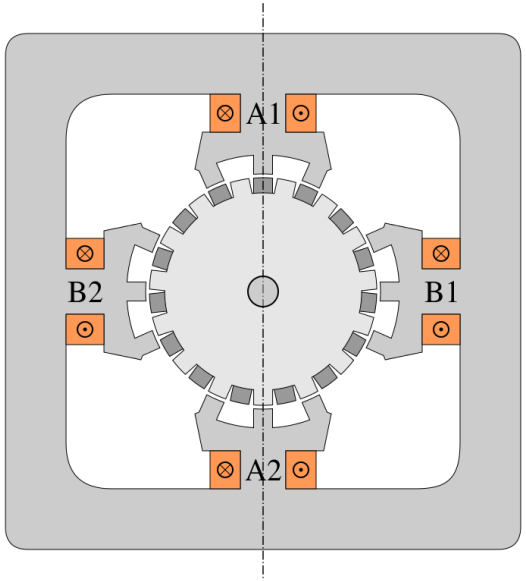
3.) Zudem wollen wir per Kommando:
linksrum, rechtsrum →"LL, RR", die Richtung ändern.

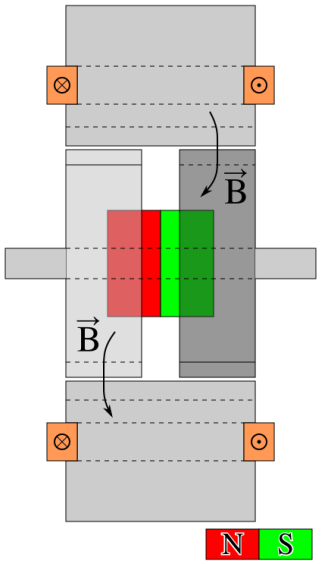
4.) Den 8 Bit PWM geben wir mit genau diesem Kommando vor.
Default: „PWM,150" für den ersten Test:

Schrittmotoren

Schrittmotoren sind sogenannte **Synchron-Motoren**, weil diese sozusagen schrittweise → z**ahnweise** am Steuersignal hängen.
Dabei ist darauf zu achten, dass diese nicht durch Überlastung aus dem Tritt kommen.
Die Positionen (200/Rotation) werden +/- gezählt.
Meist ist eine Indexierung → „0 Position“ erwünscht.

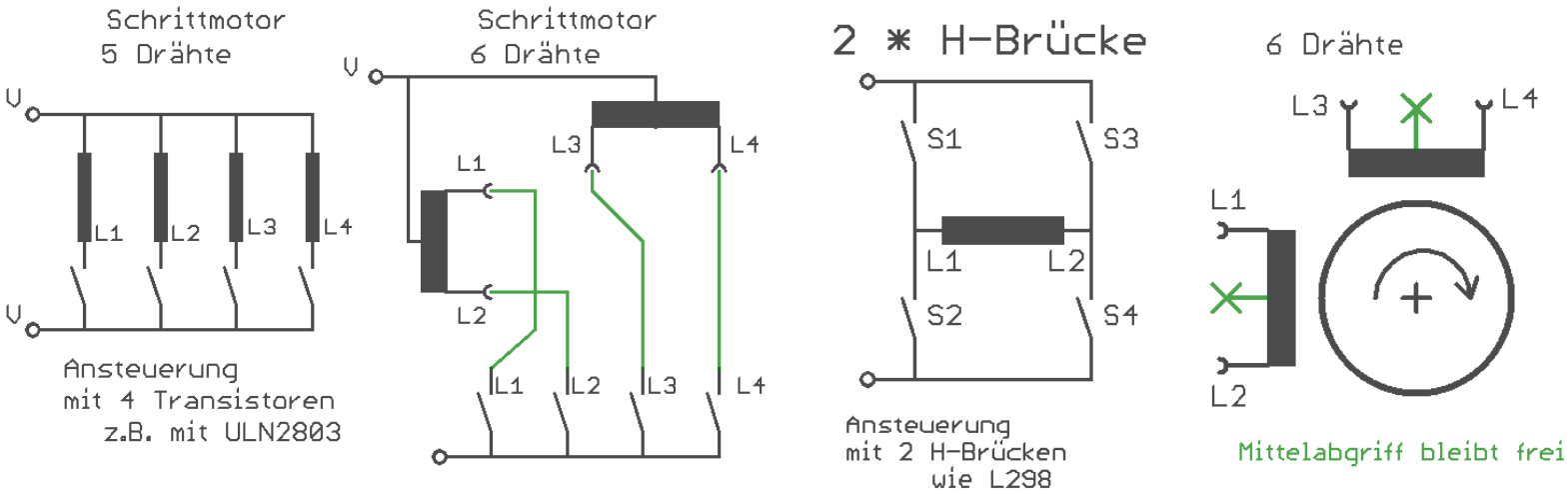






Wenn man alle Drähte des Motors kurzschließt, bremst die Gegen-EMK den Motor stark ab.
Dies beweist, dass ein Step-Motor auch ein Generator sein kann.
→Idee dazu: Pulsgeber.

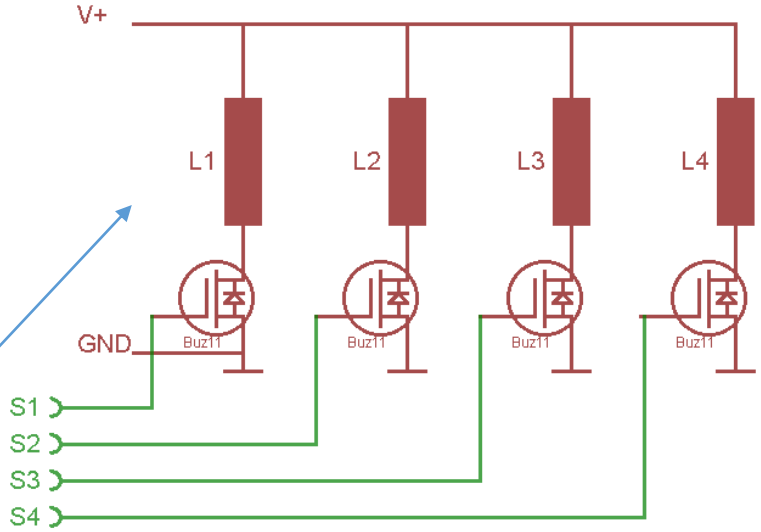
Schrittmotor - Prinzip



Mit der Kenntnis der möglichen Innenbeschaltung lässt sich durch Ohm-Messungen interpretieren mit welcher Motorvariante man es zu tun hat.

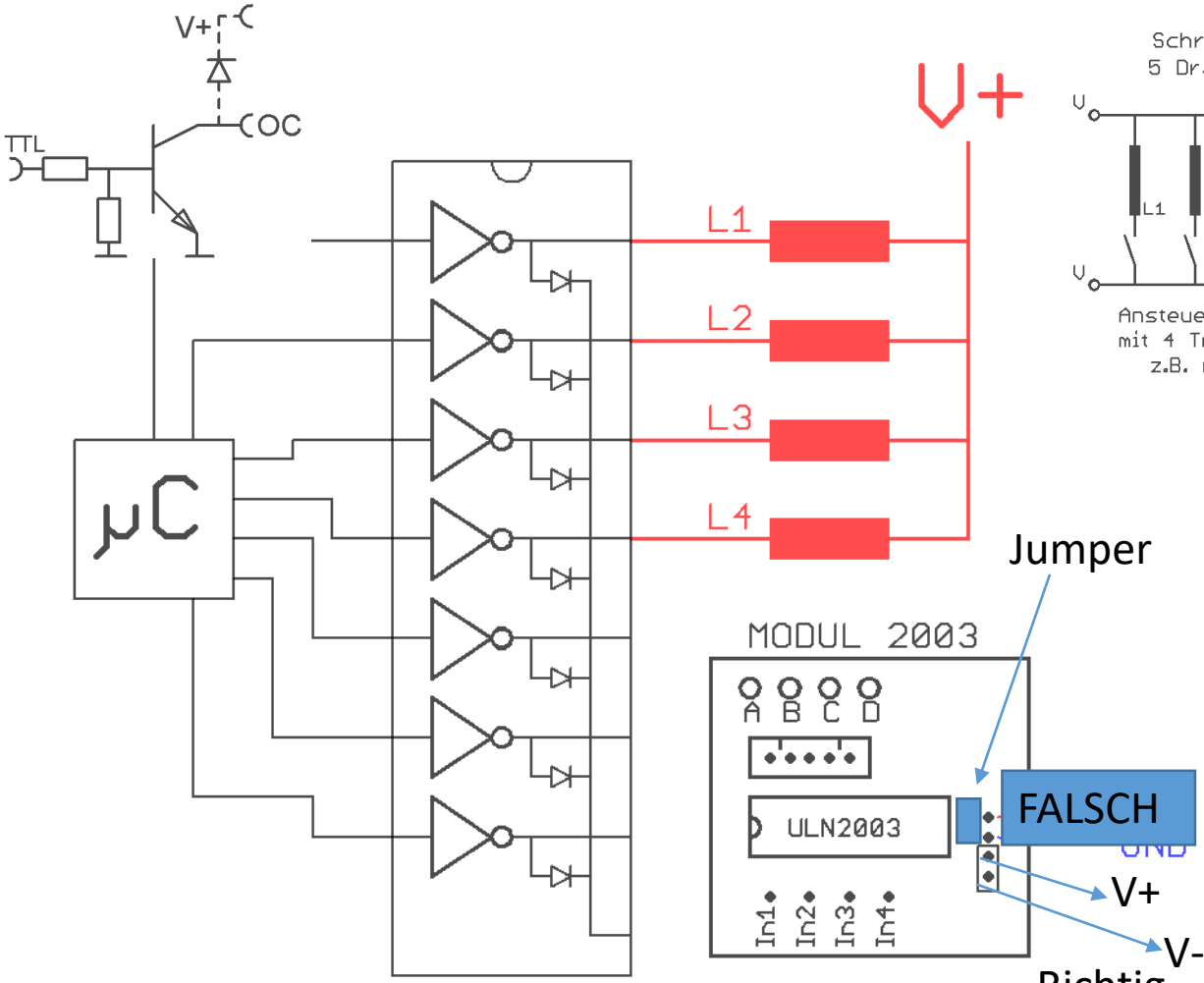
Beim **6-Pol Motor** lässt man z.B. einfach die Innenleiter frei.

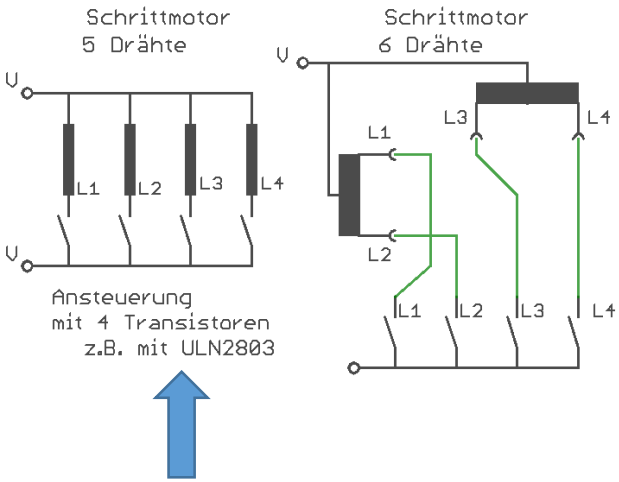
Beim **5-Pol**er ist **EINE** Leitung die gemeinsame Zuleitung.



Step-Motor Platine im Arduino Kit

…ist **keine H-Brücke !!**, sondern einfach ein Sink-Current Treiber → ULN2003
→ Open Collector Transistor Array mit TTL Eingang. Also Bitmuster beachten.





Jumper

Richtig

Die Schaltung ist für den 5 Draht Step-Motor. Wir müssen also GND und V+ zusätzlich zu den 4 **In[1..4]** Signaleingängen anlegen.

const char MCA_ARDUINO_SM[] PROGMEM = {0x08,0x0C,0x04,0x06,0x02,0x03,0x01,0x09};

5-Draht Getriebe Motor, gerade noch mit **5V** direkt versorgbar !
Immer den max. Strom beachten! Bei zu großer Belastung
(> ~200mA) geht das nicht mehr vernünftig über USB.
Dann ist ein externes Netzteil erforderlich !.

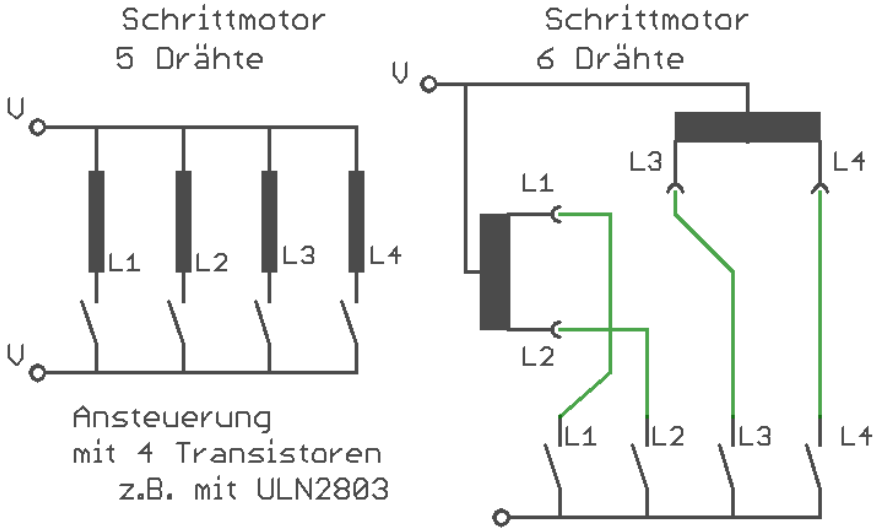
Bitmuster zum Fahren dieses 5Pol Getriebe-Steppmotors

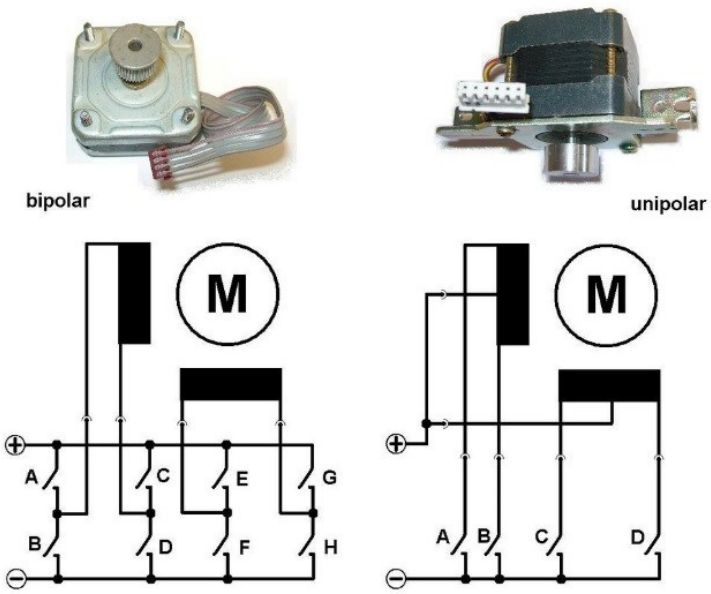

```
const char MCA_ARDUINO_SM[] PROGMEM = {0x08,0x0C,0x04,0x06,0x02,0x03,0x01,0x09};
```

Code:

```
if(gu8SM_Status & MOTOR_DIRECTION)
   { SM_PORTX = pgm_read_byte_near( MCA_ARDUINO_SM + u8NN++ ); }
else
   { SM_PORTX = pgm_read_byte_near( MCA_ARDUINO_SM + u8NN-- ); };
u8NN %= 8;
```

Konsequenzen für den Step-Motor Anschluss für einen reinen Sink-Current Treiber wie ULN2803 oder ULN2004





2-Spulen-4-Pol „bipolar“ Motoren können wir leider NICHT anschließen.
Diese brauchen eine H-Brücke.

Es gehen nur „unipolar“ 5-Poler mit gemeinsamer Common bzw. 6-Poler mit Mittelabgriff.

Programmierung: Schrittmotor

Wir können uns nun modular die Software zusammenkopieren und anpassen:

1.) Wir benötigen einen langsamen 16-Bit Timer mit Interrupt für die Schritte.

2.) Wir benötigen passende Bitmuster, die die H-Brücke die wir verwenden, richtig schalten. Hier am L298 Beispiel
Einfacherweise tut es ein #define oder doch besser gleich ein **PROGMEM-Array** . →

```
const char MCA_FULLSTEP[] PROGMEM = {0x3A, 0x39,0x35,36};
const char MCA_HALFSTEP[] PROGMEM =
        {0x3A,0x28,0x39,0x11,0x35,0x24,0x36,0x12};
const char MCA_ARDUINO_SM[] PROGMEM =
        {0x08,0x0C,0x04,0x06,0x02,0x03,0x01,0x09};
```

Muster:
E2,E1,IN 4..1

11 1010
11 1001
11 0101
11 0110

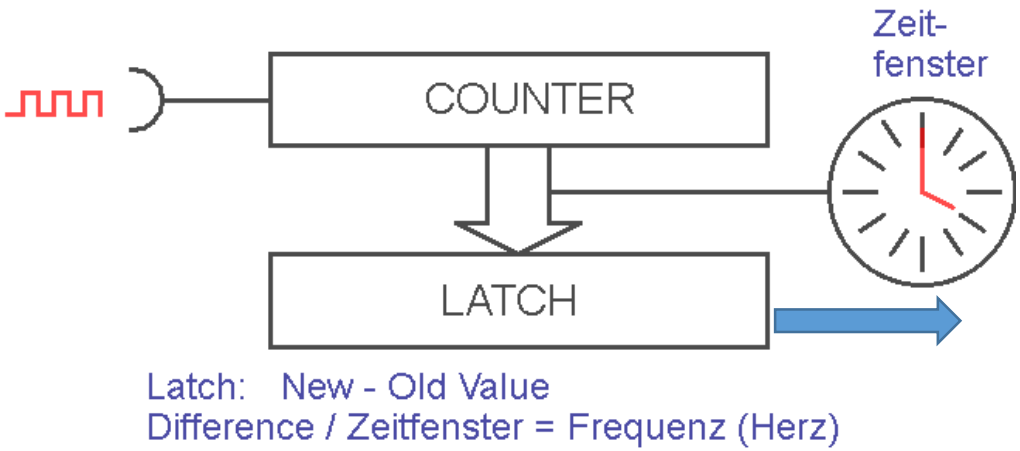
Analysiere die Bitwechsel

3.) Zudem wollen wir per Kommando:
linksrum, rechtsrum → "LL, RR", die Richtung ändern.
Step-FRQ ändern → „FF,0.3..1000" Hz
4.) Einfache On, OFF → „FF,0"
5.) Kombi-Befehl → MM, Speed, Direction → „MM,0.3..1000,[0,1]";

```
ISR( TIMER1_COMPA_vect )// signal handler
{
volatile static uint8_t u8NN;
volatile uint16_t u16IstFrq ;
u16IstFrq = round( F_CPU / 1024.0 / (float)OCR1A );
if( gu8SM_Status & RAMPE_AKTIVE )
   {
                if( gu16SOll > (u16IstFrq + RAMPENSTEIGUNG) )
                {
                OCR1A  = round( F_CPU / 1024.0 / ((float)(u16IstFrq + RAMPENSTEIGUNG)) );
                TCNT1 = 0;
                }
else
                {
                OCR1A = round( F_CPU / 1024.0 / (float)gu16Soll );
                TCNT1 = 0;
                gu8SM_Status &= ~RAMPE_AKTIVE;
                };
                u8NN--;
                };
u8NN %= 4;
STEPM_PORTX = (STEPM_PORTX &  ~STEPM_6BIT_MASK) |
                SET_STEPM_BITS( (pgm_read_byte_near( MCA_FULLSTEPS + u8NN ) ) );
} else  {       TCCR1B = 0; // STOP CLOCK
                TIMSK1 = 0; // DISBALE INTERRUPT T1
STEPM_PORTX &= ~STEPM_6BIT_MASK; gu8SM_Status &= ~MOTOR_ON;      };
```

```
void StartTimer_1( uint16_t u16Frq )
{
TCCR1B = 0;  TCNT1 = 0;   STEPM_INIT();
gu8SM_Status &= ~(MOTOR_ON | RAMPE_AKTIVE);
if( u16Frq )
        {
        gu16Soll = u16Frq;
        if( gu8SM_Status & RAMPE_ENABLED )
                {
                OCR1A = round( F_CPU / 1024.0 ); // Starte langsam /(1sekunde)
                gu8SM_Status |= RAMPE_AKTIVE;
                }
        else
                {
                OCR1A = round( F_CPU / 1024.0 / (float)gu16Soll );
                };

        TIMSK1 = _BV(OCIE1A); //Interrupt
        TCCR1B = _BV( WGM12) | _BV( CS12) | _BV(CS10); //CTC + /1024
        gu8SM_Status |= MOTOR_ON;
        sei();
        }
```

Frequenz Messung mit Timer



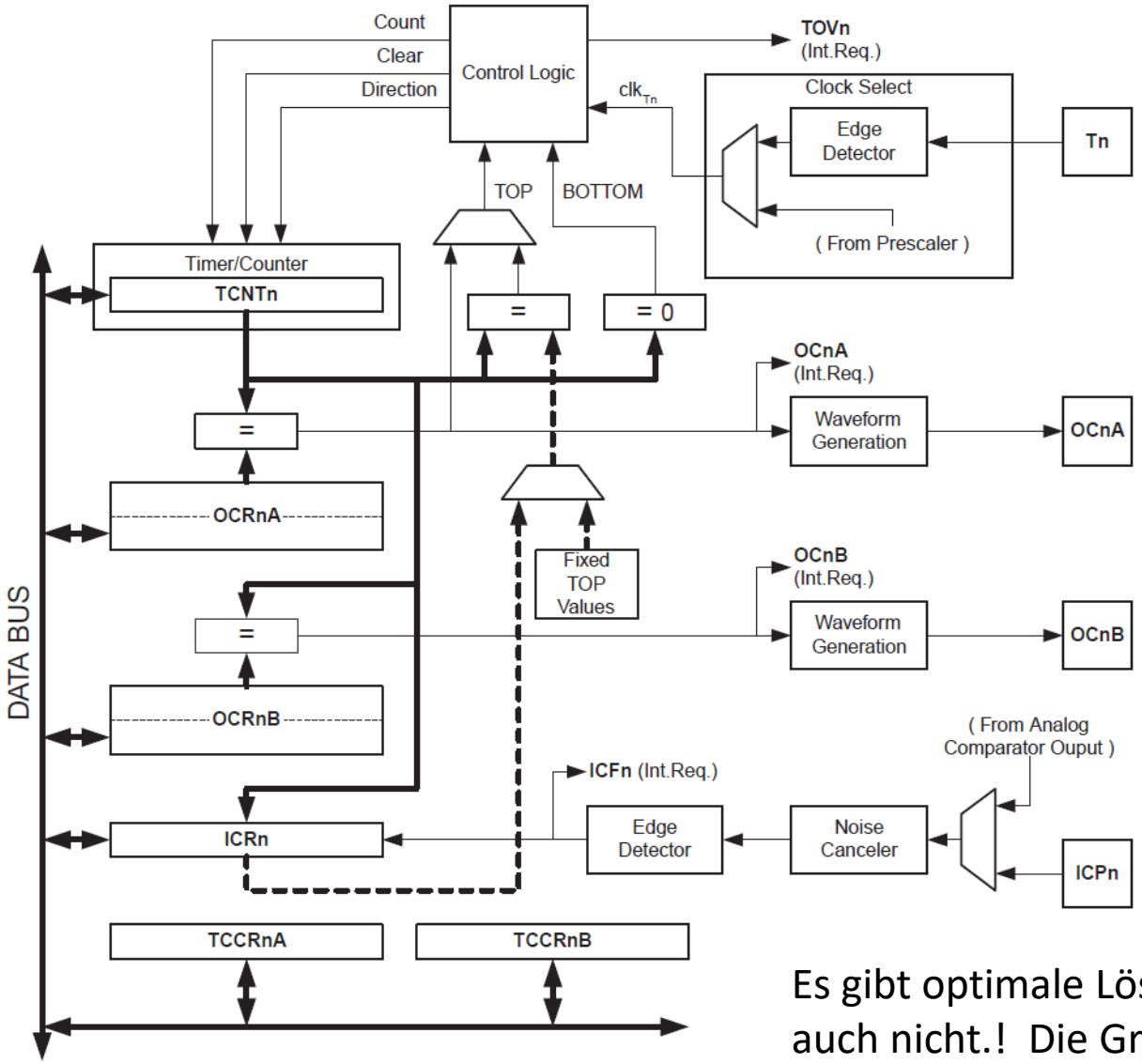
Ein Messung bedeutet → **Pulse/Zeit**.

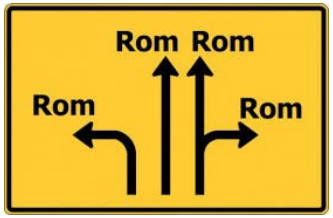
Idealerweise wird ein Zähler in einer genauen Zeitscheibe vor dem Lesen in ein **Latch** kopiert und dieses dann ausgelesen.

→ Difference = Latch - OldVal;
So kann „on the fly“, ohne Störung des Zählers gelesen werden.
Sonst könnte bei 0-Überläufen Blödsinn rauskommen.. !!
= Difference / Zeitscheibe; Dann → OldVal = Latch;
Ein Timer{interrupt} bildet die Zeitscheibe.

Wenn die µC Hardware keinen Latch hat, dann muss der Timer für den Lesemoment angehalten werden… Da ist auf Ungenauigkeiten zu achten

Figure 15-1. 16-bit Timer/Counter Block Diagram[1]

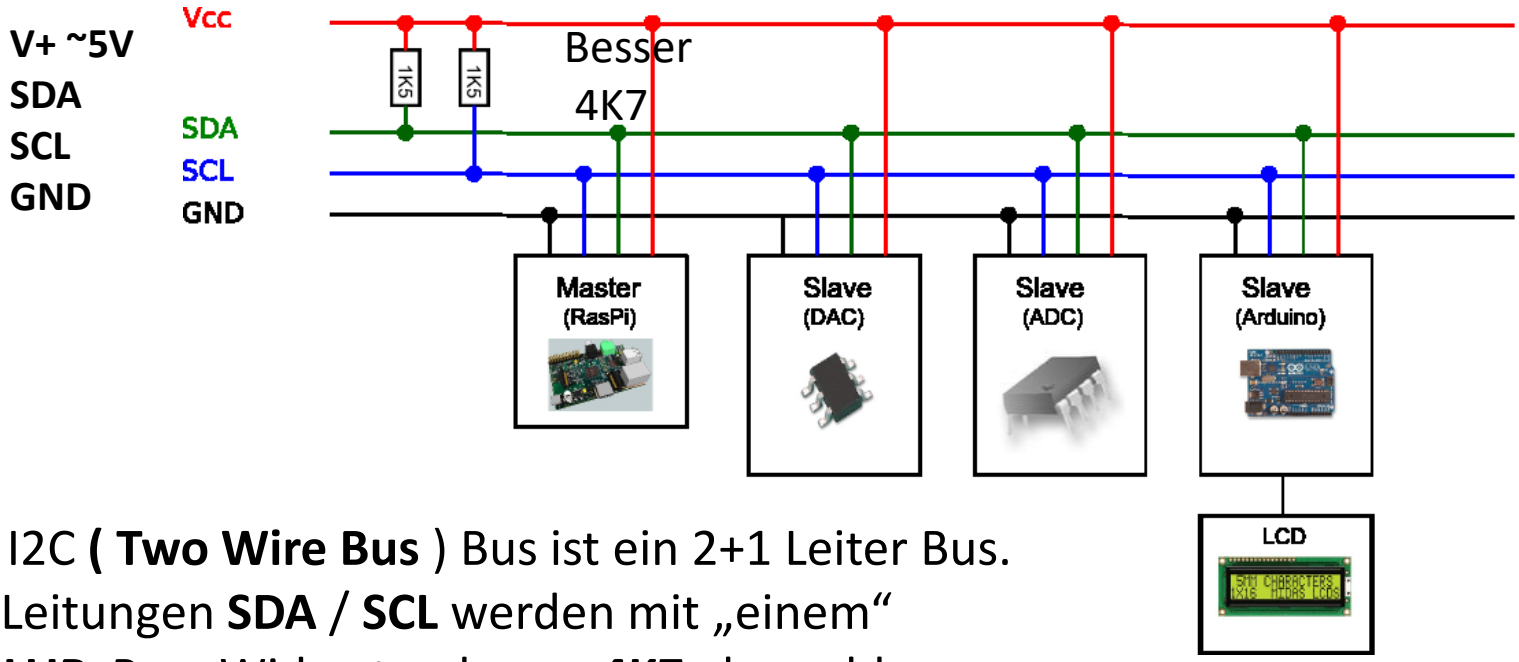




Unser Ziel: Messung eingehender Pulse. Der Weg…. „Irgendwie“ mit der vorhandenen Hardware ‚möglichst‘ präzise innerhalb eines **Zeitfenster**s die eingehenden **Pulse zählen** und interpretieren.

Es gibt optimale Lösungen… oder auch nicht.! Die Grenzen liegen in den Fähigkeiten der Hardware.

I2C oder TWI BUS (Two Wire Bus)



Der I2C **(Two Wire Bus)** Bus ist ein 2+1 Leiter Bus.
Die Leitungen **SDA** / **SCL** werden mit „einem“
PULLUP Paar Widerstand von **~4K7** abgeschlossen.

Es ist ein **MASTER – SLAVE** Bus.
Naheliegenderweise ist der Mikrocontroller der Master.
Die angeschlossene Geräte/ICs verarbeiten die Daten.
Damit lassen sich viele Leitungen im Hardwarelayout sparen.
Viele, nicht übertrieben schnelle sind so, **der Reihe nach**, ansprechbar.

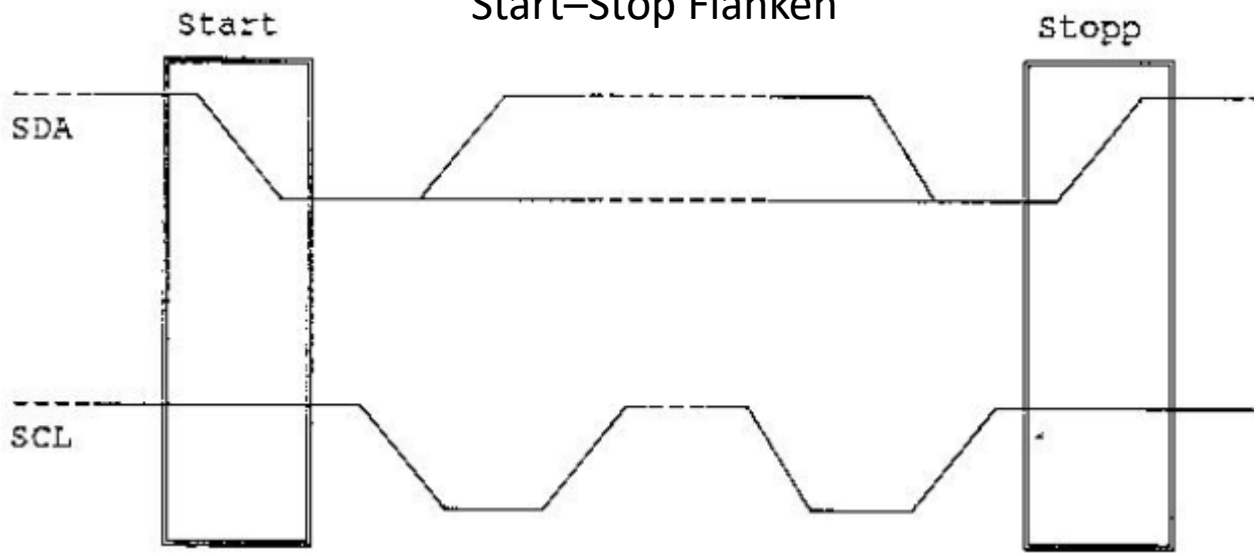
7-Bit Adressierung

BIT 7	6	5	4	3	2	1	BIT 0
I2C - Adresse[6..0]							Read=1 / Write = 0

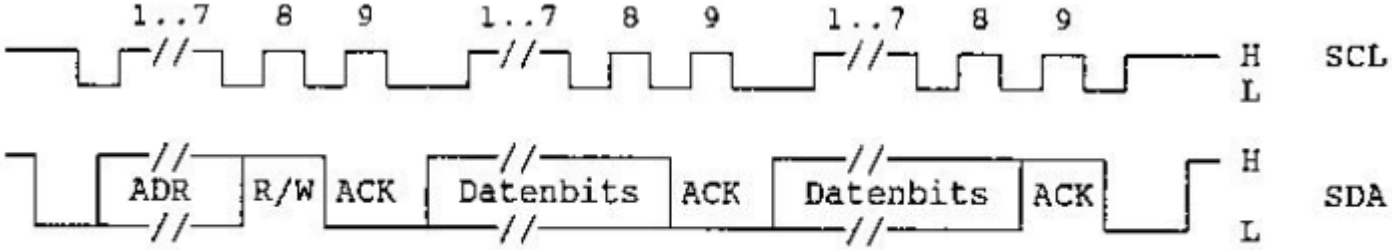
Jedes IC/Gerät/Device hat **eine eigene** Adresse.
Diese Adresse hat **7 BIT** → damit max. **128 Geräte**.
Adressen sind hier starr festgelegte/verdrahtete Bits am IC.
(Bsp. Bits: (**x**)**0100010** = Adr. 34). (die ersten 3 Bist sind evtl. Jumperbar)
Die Daten werden in einem Protokoll übermittelt, das aus Adressen, Steuerbits und Daten besteht.
Zugleich enthält die Signalflanken-Situation Information über den START-STOP Status.

Wichtig: (Adresse << 1), wegen 7 Bit Adresse

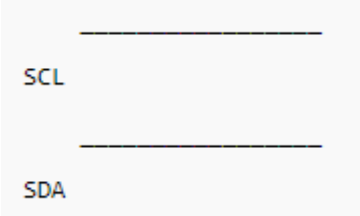
Start–Stop Flanken



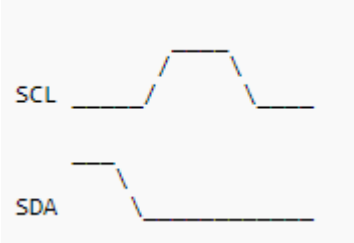
Dann folgen Informationen was beabsichtigt ist,
R/W
ACK = Acknowledge = Bestätigen
Daten Bits, bis zum ACK



Beispiel einer Übertragung (Lesen).



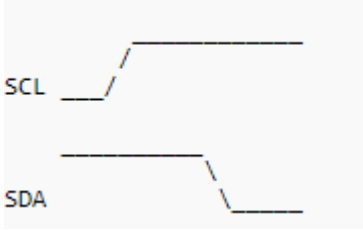
Ruhe Level
SCL=1
SDA =1



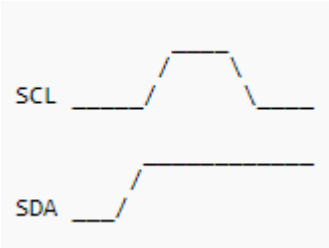
ACK

! Clock von Slave bei Read

ACK: Write to slave, Acknowledge (Bestätigung) Slave quittiert erkennen der Adresse bzw. des Datenbyte. Read to master: Slave quittiert nach erkennen seiner Adresse. Master quittiert nach Empfang. ACK=für weitere Daten bereit

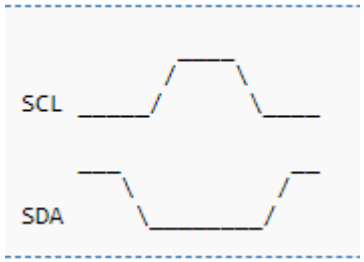


START
SCL=1
SDA 0→1

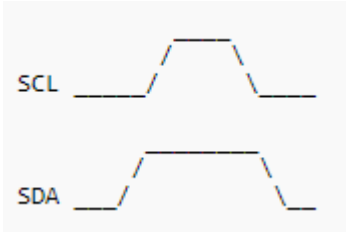


NACK

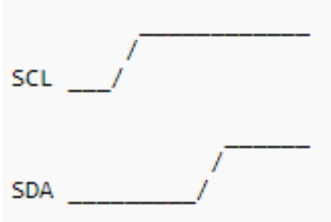
NACK: Bei Master-Readsequenz Nach lesen eines Byte: Master→NACK to Slave. NACK=Keine weiteren Daten mehr. Takt = von Slave



Datenbit
SDA=0 v 1
SCL 0→1
Hier 0



Datenbit
SDA=0 v 1
SCL 0→1
Hier 1



STOP
SCL=1
SDA =0→1

http://www.cc-zwei.de/wiki/index.php?title=I2C_Bus_Hintergrundwissen#Bus_Zust.C3.A4nde

Wir verwenden wieder die Module von Peter Fleury
http://homepage.hispeed.ch/peterfleury/avr-software.html

I2C Master Interface

This library consists of a C include file *i2cmaster.h* and an assembler module *i2cmaster.S.*

It can be used to communicate with I2C devices (serial EEPROM, serial RTC etc) from any AVR device.

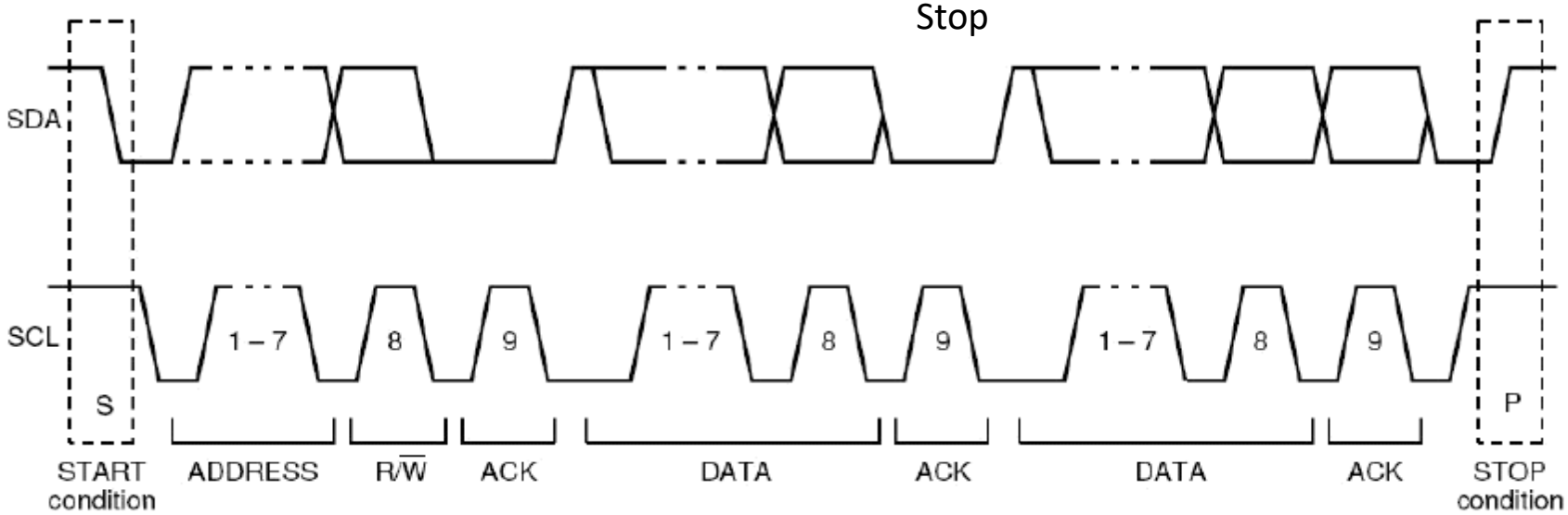
Am ATMega328 liegt:

SCL an PC5

SDA an PC4

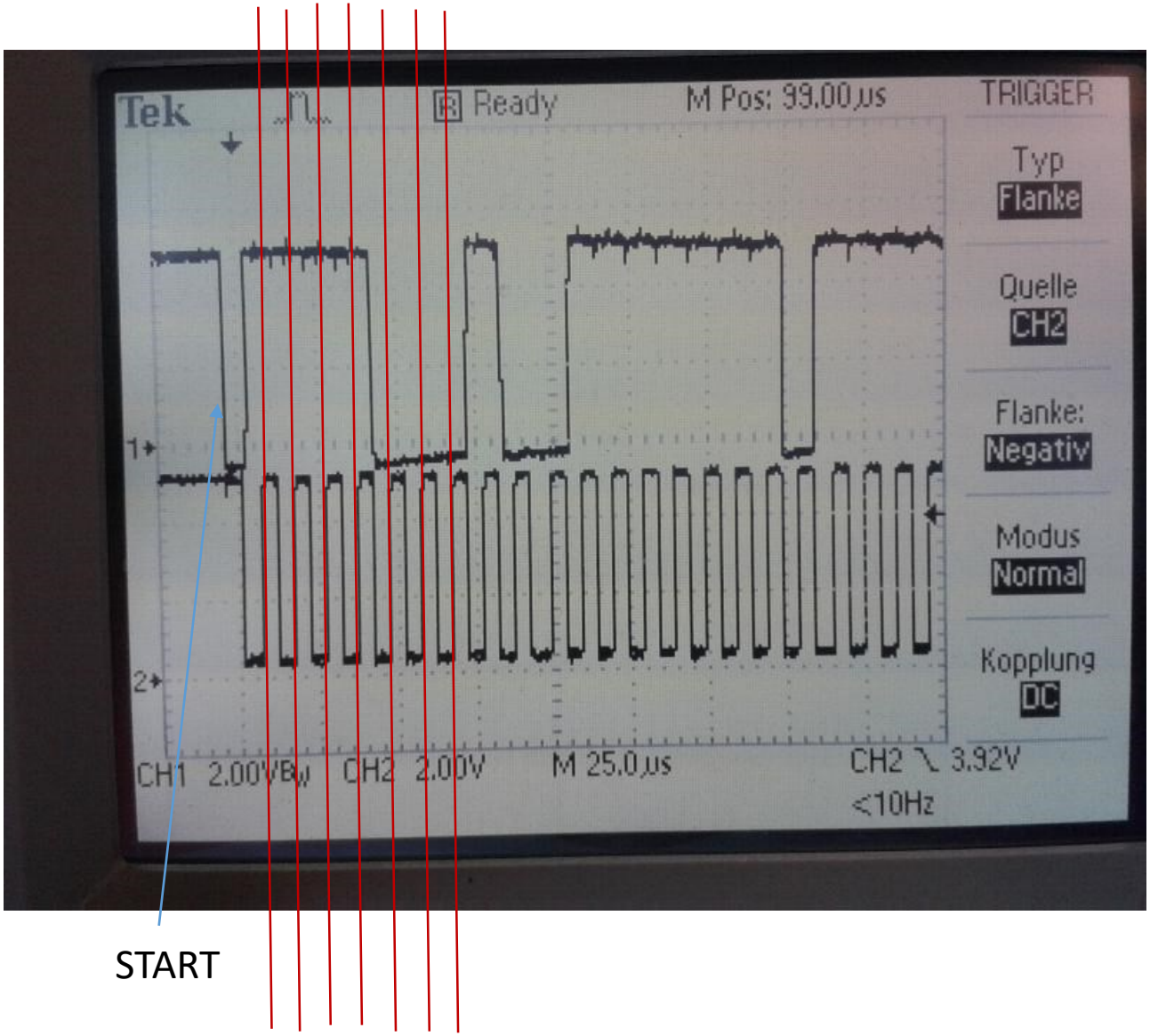
Signale:

Start
Address
R/W
Ack / NACK
Data (Ack..)
Stop



SDA

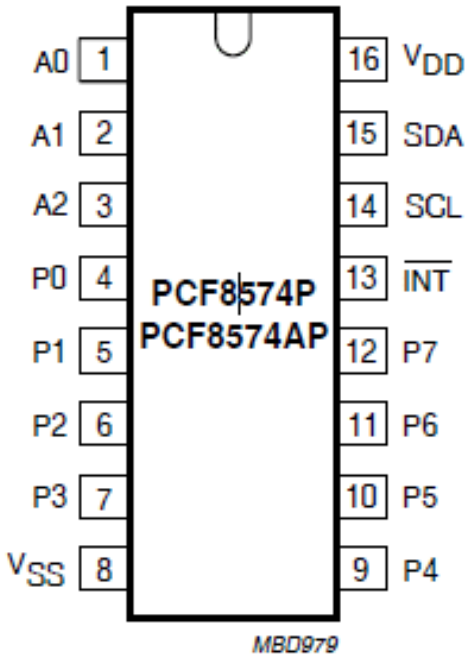
SCL



START

Adresse = x1111000 = 120

I2C PCF8574 Bus Expander



5.1 DIP16 and SO16 packages

SYMBOL	PIN	
A0	1	address input 0
A1	2	address input 1
A2	3	address input 2
P0	4	quasi-bidirectional I/O 0
P1	5	quasi-bidirectional I/O 1
P2	6	quasi-bidirectional I/O 2
P3	7	quasi-bidirectional I/O 3
V_{SS}	8	supply ground
P4	9	quasi-bidirectional I/O 4
P5	10	quasi-bidirectional I/O 5
P6	11	quasi-bidirectional I/O 6
P7	12	quasi-bidirectional I/O 7
$\overline{INT}$	13	interrupt output (active LOW)
SCL	14	serial clock line
SDA	15	serial data line
V_{DD}	16	supply voltage

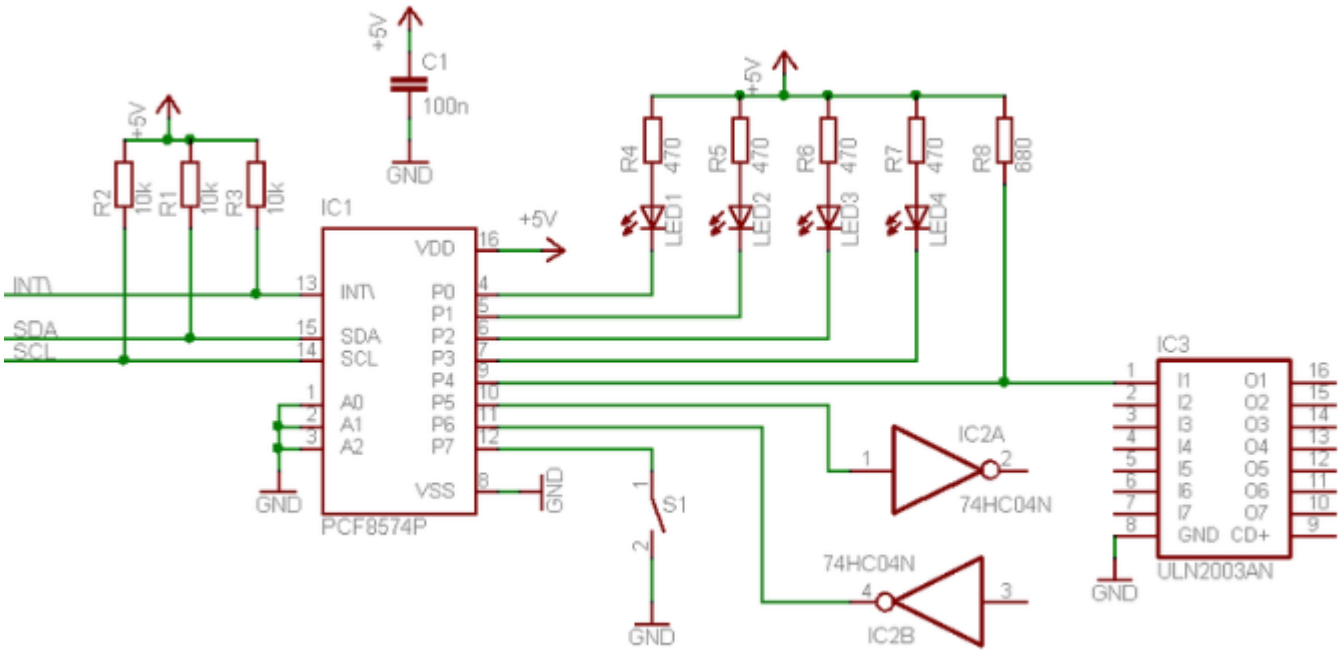
ACHTUNG :
out = open colletor

ADRESSE codiert ->

| 0 | 1 | 0 | 0 | A2 | A1 | A0 | R/W | == 64

DEZ = 0x40

7-Bit codiert als linksbündig 8 Bit + r/W



https://www.mikrocontroller.net/articles/Port-Expander_PCF8574

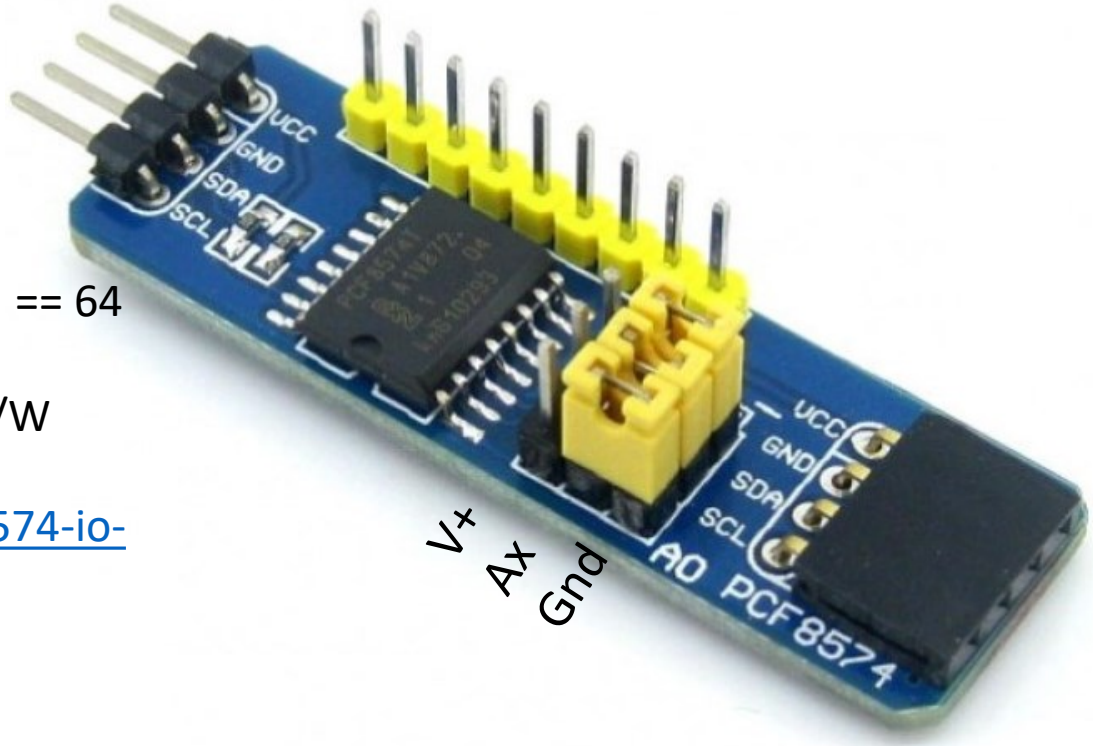
ACHTUNG : out = open colletor

ADRESSE codiert ->

| 0 | 1 | 0 | 0 | A2 | A1 | A0 | R/W | == 64

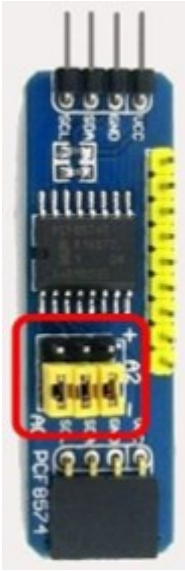
DEZ = 0x40

7-Bit codiert als linksbündig 8 Bit + r/W



https://www.waveshare.com/pcf8574-io-expansion-board.htm

A0	A1	A2
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



0 100 000 0 = 0x40

+ (3Bit)=7 Adressen

0x40..0x47

Alle Jumper Richtung

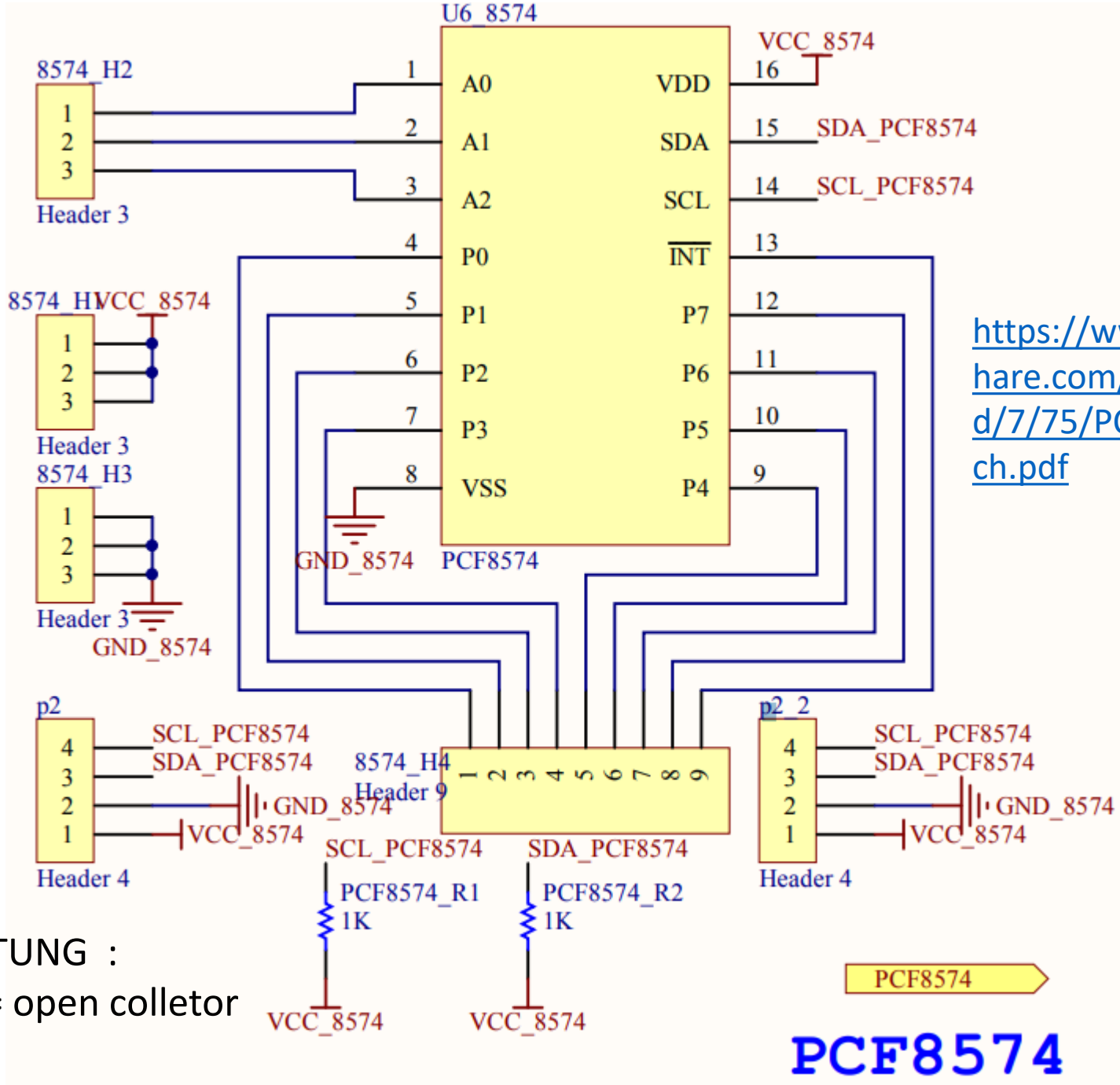
Stecker= 0x47

ACHTUNG :

out = open colletor

=0x40

Wegen Kollision mit LCD auf 0x47 ändern

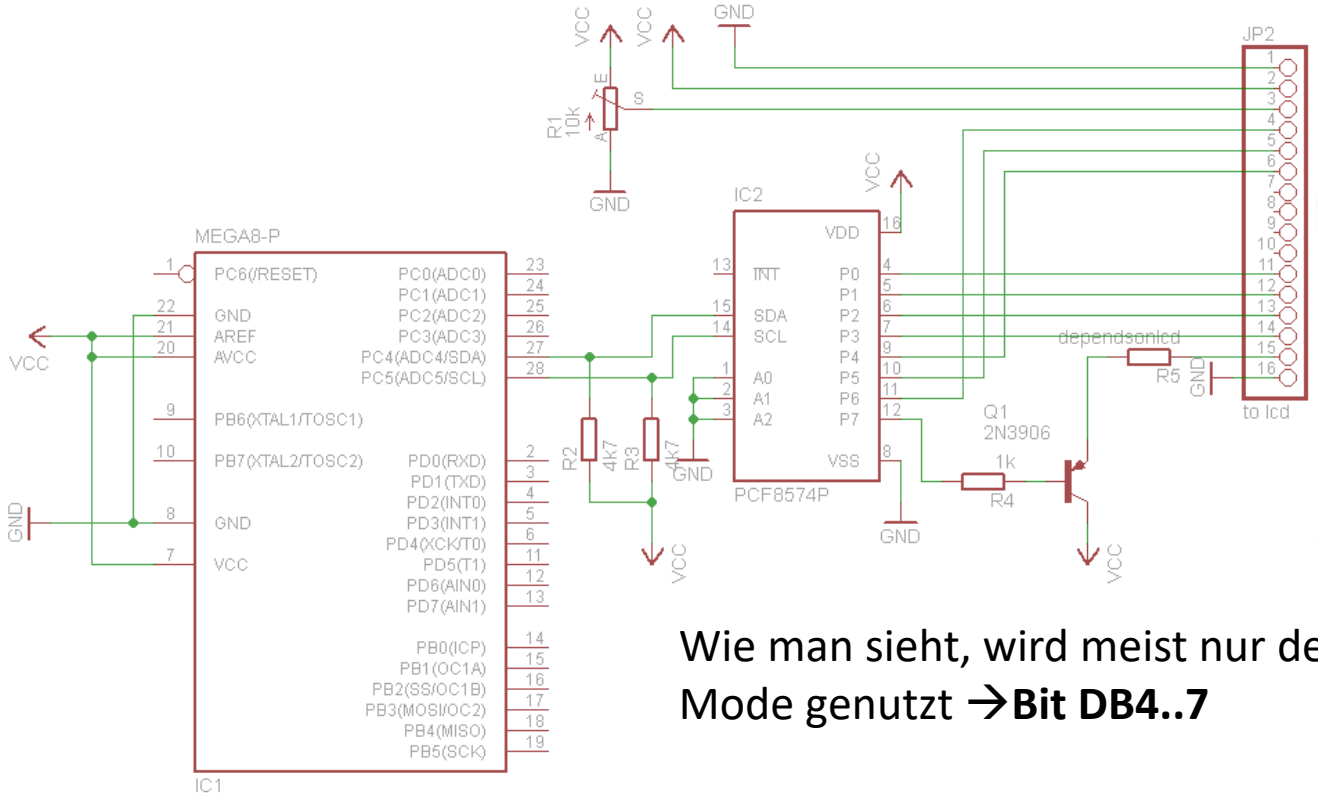


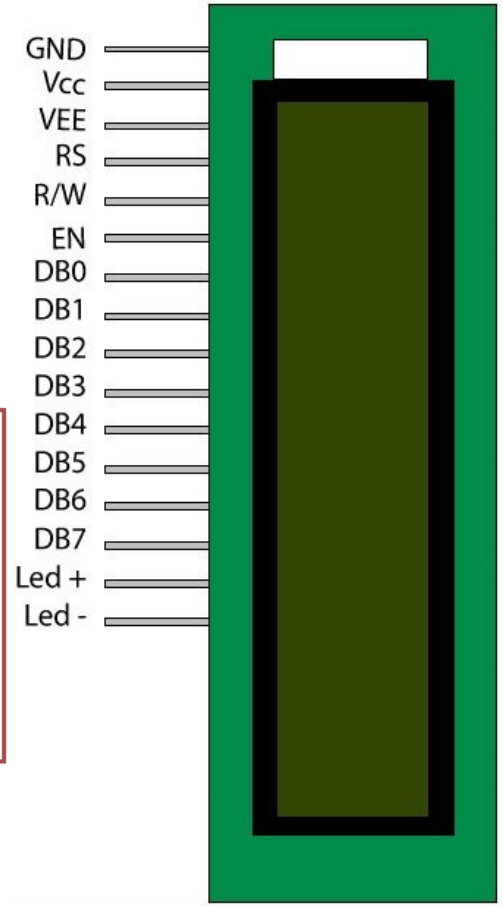
https://www.waveshare.com/w/upload/7/75/PCF8574_sch.pdf

ACHTUNG :
out = open colletor

I2C-LCD Display

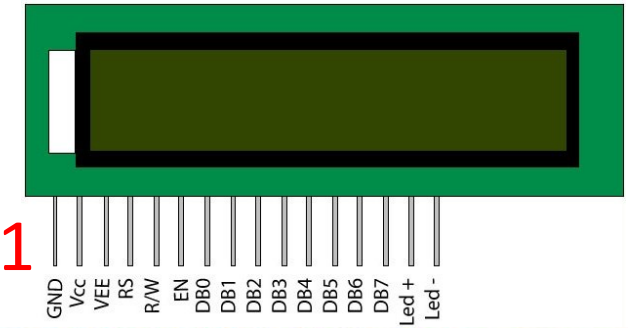
Das I2C-IC **PCF8574** wird auch hier verwendet.
Es stellt die Daten I-O parallel zu Verfügung.
Eine freie Adresse (A0..2) auswählen und in der Software eintragen. Beachte, dass dies eine **7 Bit Adresse** ist →
(Adr << 1),.....weil 8 Bit Übergabe.



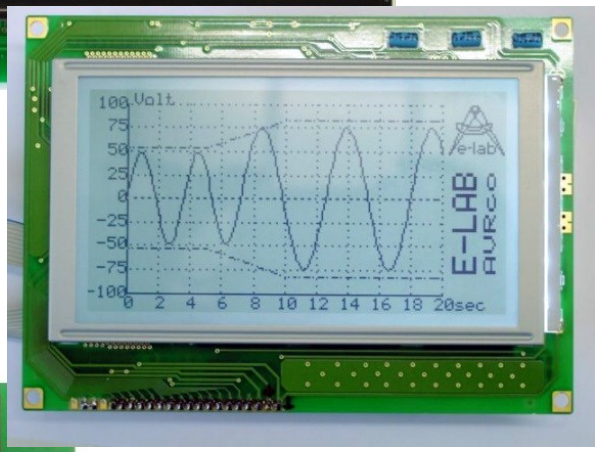


Wie man sieht, wird meist nur der 4bit Mode genutzt →**Bit DB4..7**

Ausführungen LCD-Module



Grafik Display

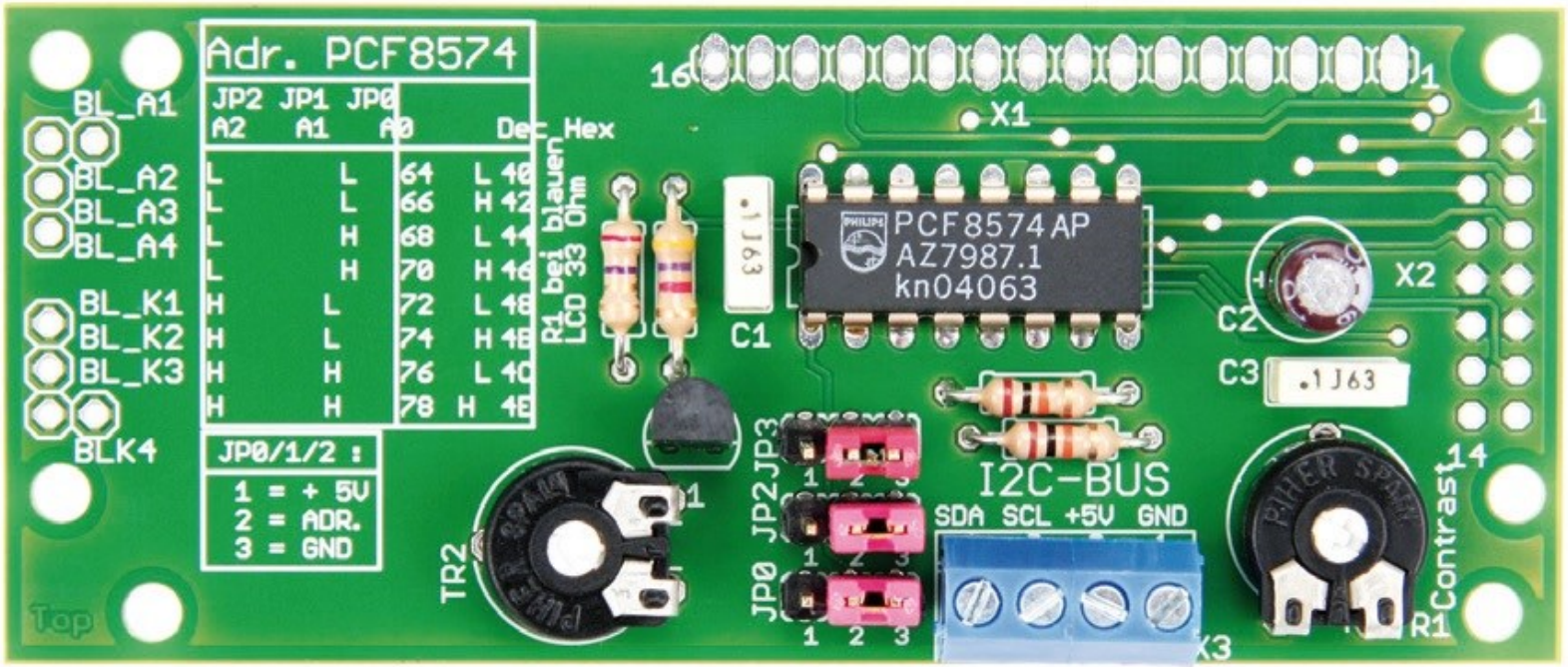


Pins:
14 13
.
.
.
2 1

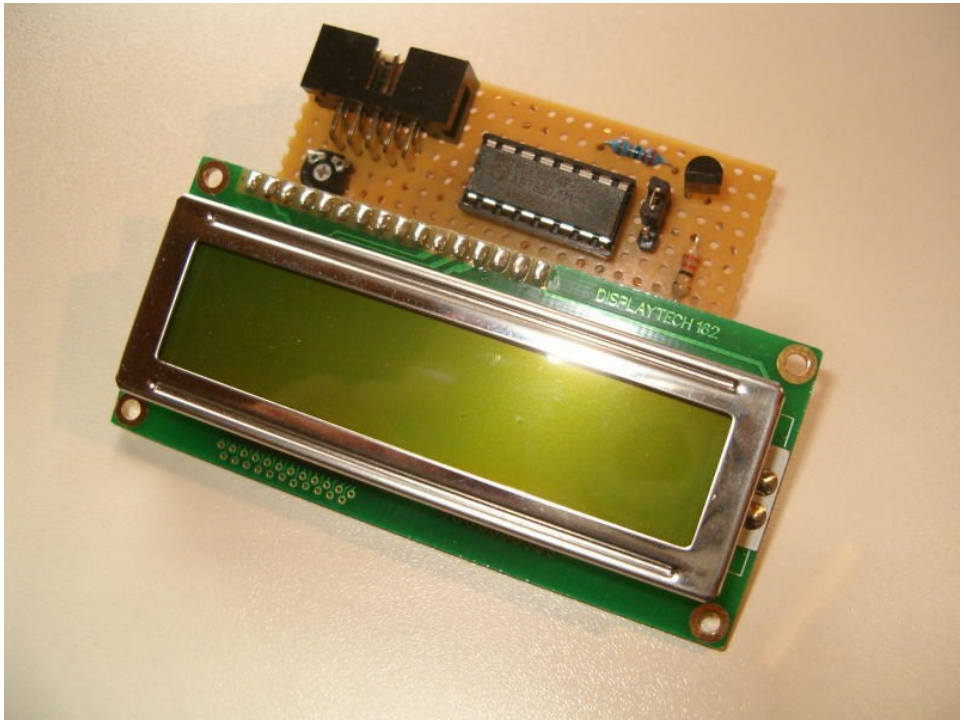
TIP: Achte bei Doppelreihen LCD auf die Vertauschung:
Pin 1=+5V, Pin2=GND
Und
1→ beginnt innen.

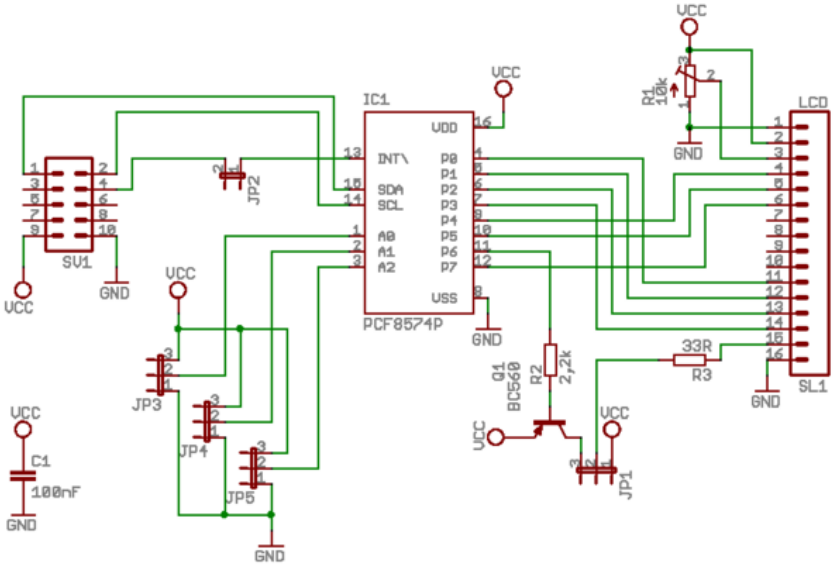
I2C Modul mit PCF8574 für Standard LCD Display

Es bietet eine Montagemöglichkeit für 1-Reihen und Doppelreihen LCD.
TIPP: Achte bei Doppelreihen LCD auf die Vertauschung: Pin 1=+5V, Pin2=GND



Selbstbau auf Lochraster I2C LCD PCF8574

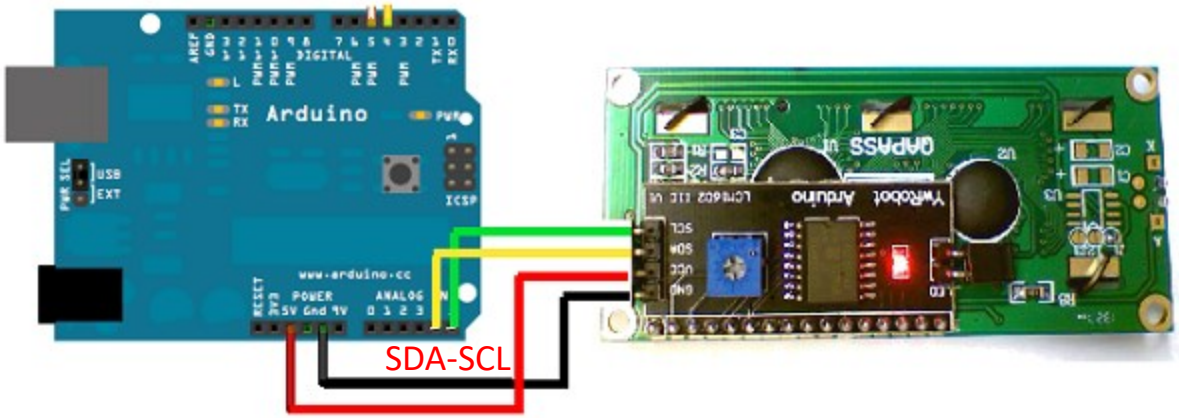




Software und Beispiel, siehe: http://bralug.de/wiki/BLIT2008-Board-LCD

→ Pollin LCD I2C Adapter hat die Adresse 0x40 wenn alle 3 Jumper Low sind
Jumper Position 2-3 = GND = **ADR 0x40 = dezimal 64 = binär 0b0100 0000**

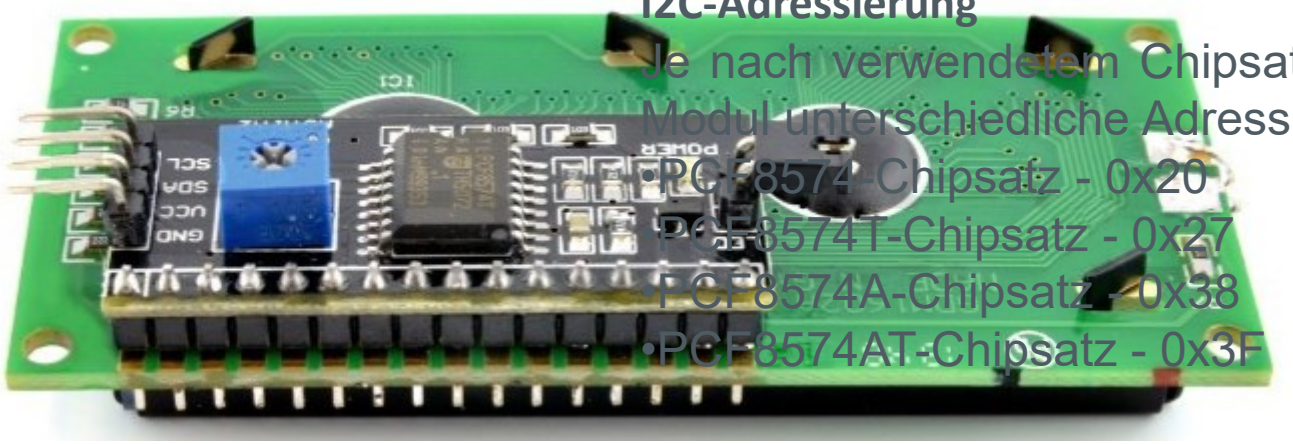
I2C LCD an Arduino Uno
I2C-Konverter für HD44780 LCD-Display



Die Standardadresse ist 0x27

Auch dazu habe ich eine Bibliothek von Peter Fleury gefunden.

→ **i2clcd.c - i2clcd.h** Siehe Beispiel „93-I2C-LCD-Modul_PCF8574“



I2C-Adressierung

Je nach verwendetem Chipsatz kann dieses Modul unterschiedliche Adressräume haben:

- PCF8574-Chipsatz - 0x20
- PCF8574T-Chipsatz - 0x27
- PCF8574A-Chipsatz - 0x38
- PCF8574AT-Chipsatz - 0x3F

LCD-02352**1602 2004 LCD Adapter Platte IIC I2C/Interface LCD1602**

Die Standardadresse ist 0x27

Nein	Name	Beschreibung
1	Masse	Masse.
2	VCC	Stromversorgung +5 V.
3	SDA	Die Datenleitung des I2C-Busses.
4	SCL	Die Taktleitung des I2C-Busses.

I2C-Adressierung

Je nach verwendetem Chipsatz kann dieses Modul unterschiedliche Adressräume haben:

•PCF8574-Chipsatz - 0x20
•PCF8574T-Chipsatz - 0x27
•PCF8574A-Chipsatz - 0x38
•PCF8574AT-Chipsatz - 0x3F

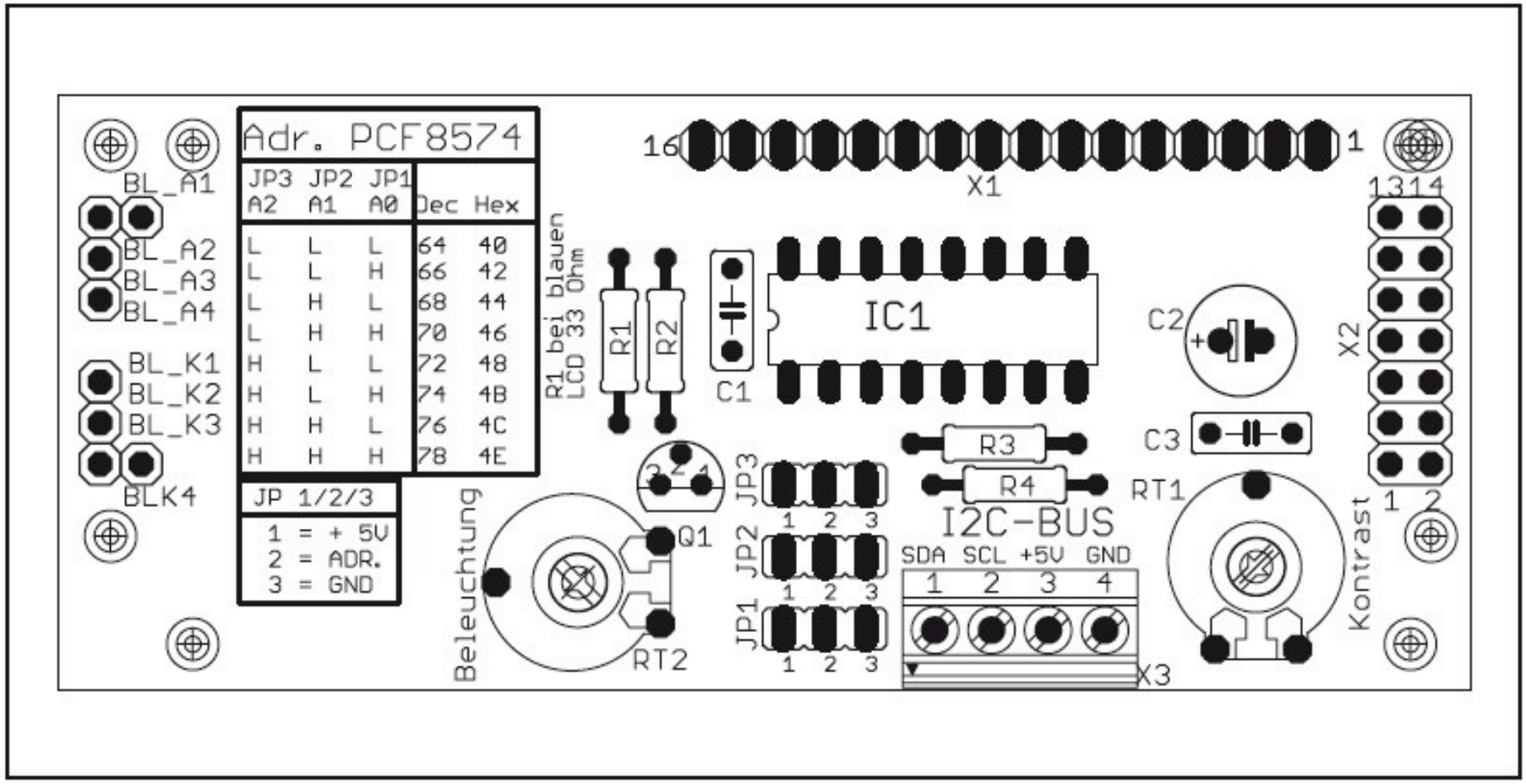
Das System verfügt über vier Stifte und einen Jumper, dessen Entfernung die Hintergrundbeleuchtung des Bildschirms ausschaltet. Auf der Platine befindet sich außerdem ein Goldpin-Streifen, der auf die Anschlüsse gängiger Displays auf Basis des HD44780-Treibers abgestimmt ist. Alle Leitungen sind angelötet.

IIC I2C Serielles Schnittstellenmodul LCD1602 Adressenwechselbar, 5 Stück
Arduino 1602/2004 I2C Interface 4-Draht 1602/2004 Bildschirm
Arduino IIC/I2C Interface LCD1602/2004 Adapterplatte ohne LCD-Bildschirm
Arduino Steuerplatine IO-Port ist nur 20, plus einige Sensor, SD-Karte Han, Relais-Module und mehr, IO-Port ist nicht genug,
Der originale 1602/2004 Bildschirm benötigt 7 IO-Anschlüsse können hochfahren. Wir haben dieses Modul entwickelt, um Ihnen helfen, fünf IO-Ports zu speichern, wir senden eine Arduino-Bibliothek
Produktparameter:
1. Maße (L x B x H): 41,5 x 19 x 15,3 mm.
2. Gewicht: 5 g.
3. Leiterplatten-Farbe: Schwarz.
4. Versorgungsspannung: 2,5–6 V.
5. Unterstützt das I2C-Protokoll.
6. Mit Hintergrundbeleuchtung Power Control, kann über Jumper Einstellungen angeschlossen werden und Hintergrundbeleuchtung Power. Schließen Sie den Jumper an, um die Hintergrundbeleuchtung auszuschalten, ziehen Sie den Jumper, um die Hintergrundbeleuchtung zu trennen.
7. Der Kontrast kann durch Drehen des blauen Potentiometers im Uhrzeigersinn eingestellt werden, gegen den Uhrzeigersinn geschwächt. Potentiometer-Design auf der Vorderseite, sodass Kunden jederzeit frei einstellen können.
8. Module können kaskadiert werden, um bis zu acht zu kaskadieren. Durch Kurzschluss A0/A1/A2 ändern Sie die Geräteadresse. **Die Standardadresse ist 0 x 27**.

I2C LCD + 8x8 Matrix mit MAX7219

Software: 93-I2C-LCD-Modul_PCF8574 MAX7219

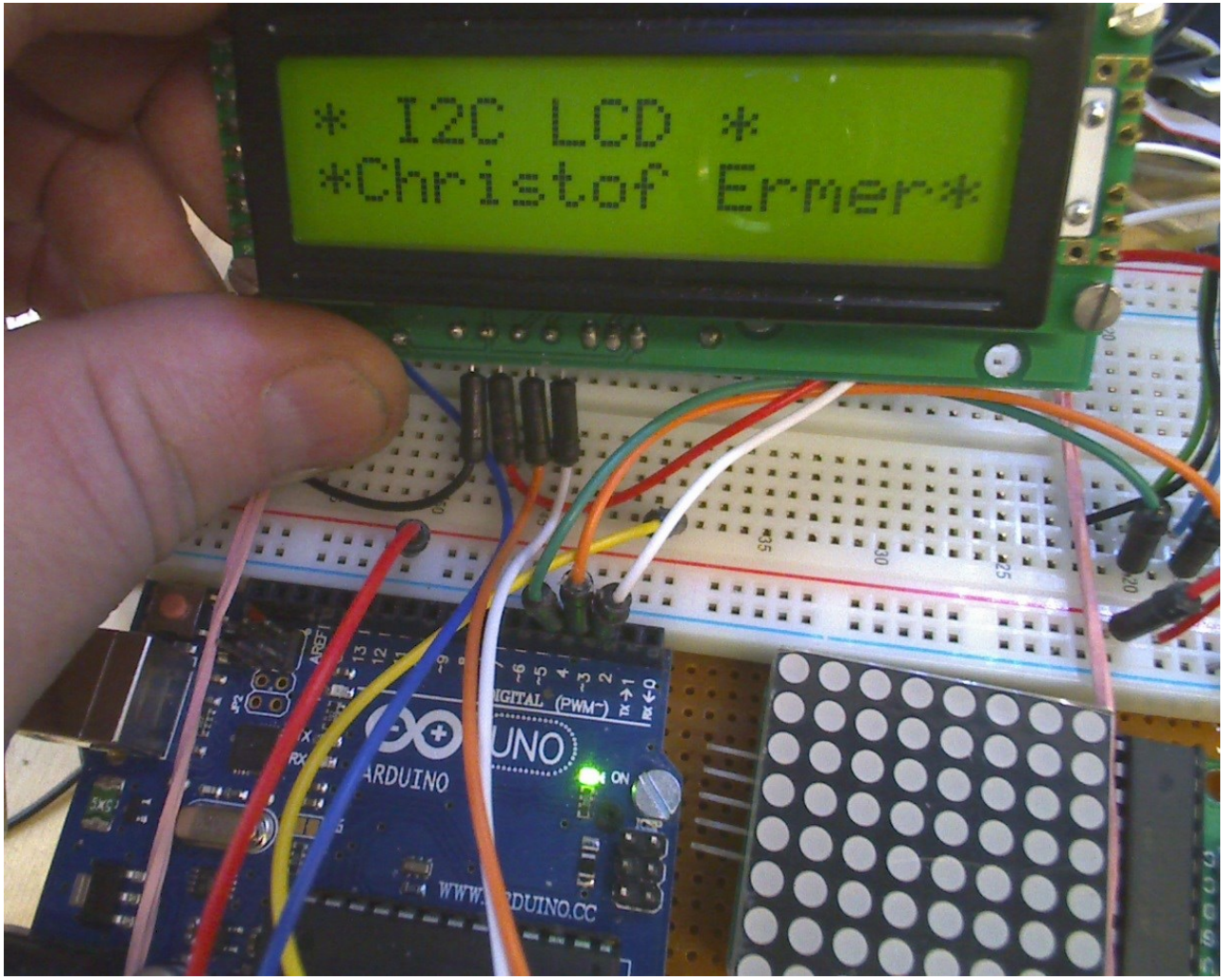
```
#include "i2cmaster.h"
#include "i2clcd.h“
const char MCA_LCD_16SPACE[] PROGMEM = "                ";  // 16 Leer Stellen
void main(void)
{
lcd_init();                                  // LCD initialisieren
lcd_command(LCD_CLEAR);          // LCD-Anzeige loeschen
lcd_backlight(ON);                        // Hintergrundbeleuchtung an

strcpy_P(gcaStr, MCA_LCD_16SPACE);
lcd_printlr(1,1,(unsigned char *)gcaStr);
lcd_printlr(2,1,(unsigned char *)gcaStr);

strcpy_P( gcaStr, PSTR("* I2C LCD *") );
lcd_printlc(1,1,(unsigned char *)gcaStr);

strcpy_P( gcaStr, PSTR("*Christof Ermer*"));
lcd_printlc(2,1,(unsigned char *)gcaStr);
strcpy_P( gcaStr, PSTR("loop/s:"));
lcd_printlc(2,1,(unsigned char *)gcaStr);
ultoa( u32MainloppCnt, gcaStr, 10 );
lcd_printlr(2,8,(unsigned char *)gcaStr);
};
```


Adr. PCF8574
JP3 JP2 JP1
A2 A1 A0 Dec Hex
L L L 64 40
L L H 66 42
L H L 68 44
L H H 70 46
H L L 72 48
H L H 74 4B
H H L 76 4C
H H H 78 4E
JP 1/2/3
1 = + 5V
2 = ADR.
3 = GND
BL_A1
BL_A2
BL_A3
BL_A4
BL_K1
BL_K2
BL_K3
BLK4
16
1
X1
1314
X2
1 2
R1 bei blauen LCD 33 Ohm
R1
R2
C1
IC1
C2
C3
R3
R4
RT1
Kontrast
JP3
JP2
JP1
1 2 3
I2C-BUS
SDA SCL +5V GND
1 2 3 4
X3
Q1
3 2 1
Beleuchtung
RT2

I2C LCD + 8x8 Matrix mit MAX7219

Software: 93-I2C-LCD-Modul_PCF8574 MAX7219

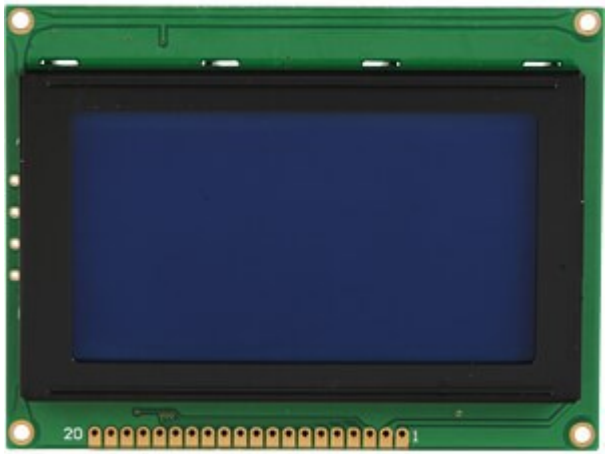
Youtube: http://youtu.be/KoqKPxGFp78

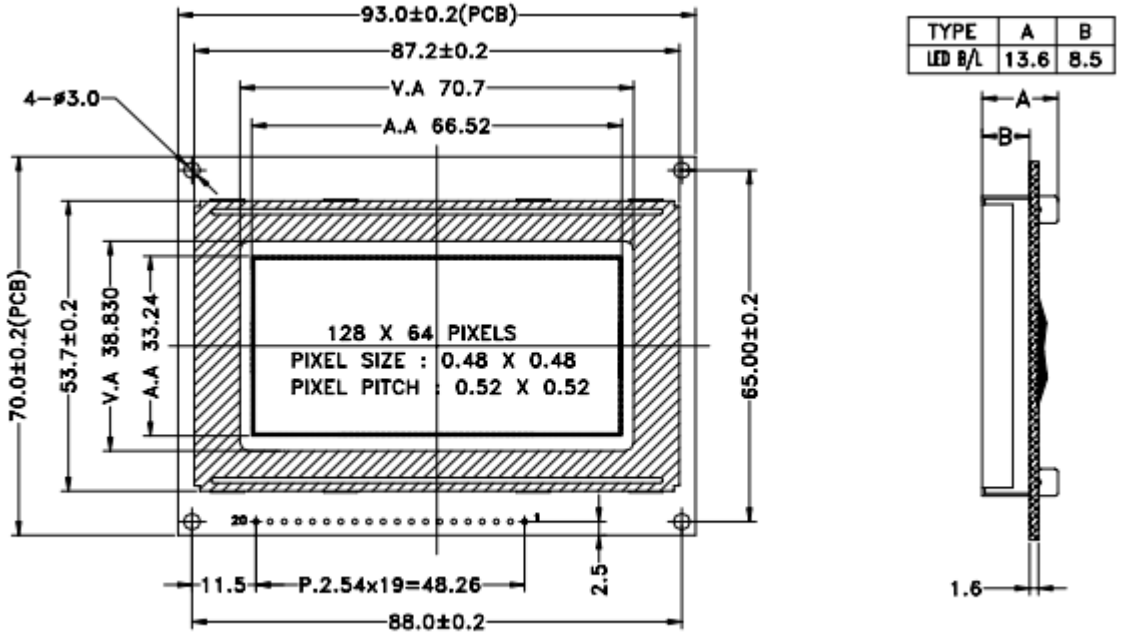


Graphic LCD 128x64 Display Type:12864.

Gibt e sin In allen Formen. Ich habe gerade ein Standardteil da:
DATA VISION PHICO D-0 94V-0 P188 **128x64** 20-Polig ~30€
https://datavision.com/wissen.de/wiki/index.php/LCD-Modul_am_AVR
Identifizieren, Auffinden diverser LCD Module:
http://www.sprut.de/electronic/lcd/
Bsp: TG**12864**B-03 https://www.pollin.de/productdownloads/D120424D.PDF

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
NAME	VSS	VDD	VO	D/I	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	CS1	CS2	/RST	VEE	K	A





PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
NAME	VSS	VDD	VO	D/I	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	CS1	CS2	/RST	VEE	K	A

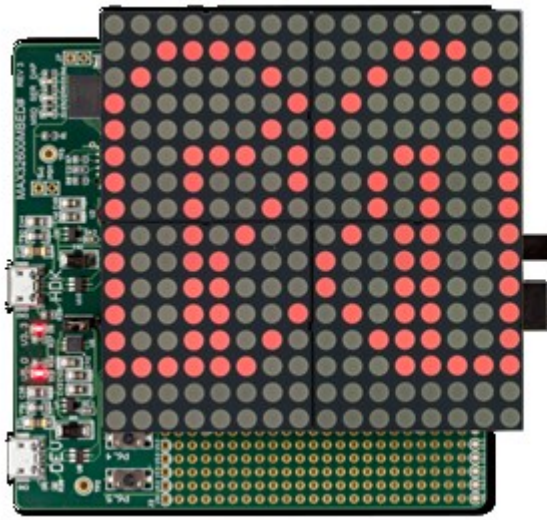
SKETCH Quick Test →LiquidCrystal440.h und LiquidCrystal440.cpp im Verzeichnis /libraries/LiquidCrystal440

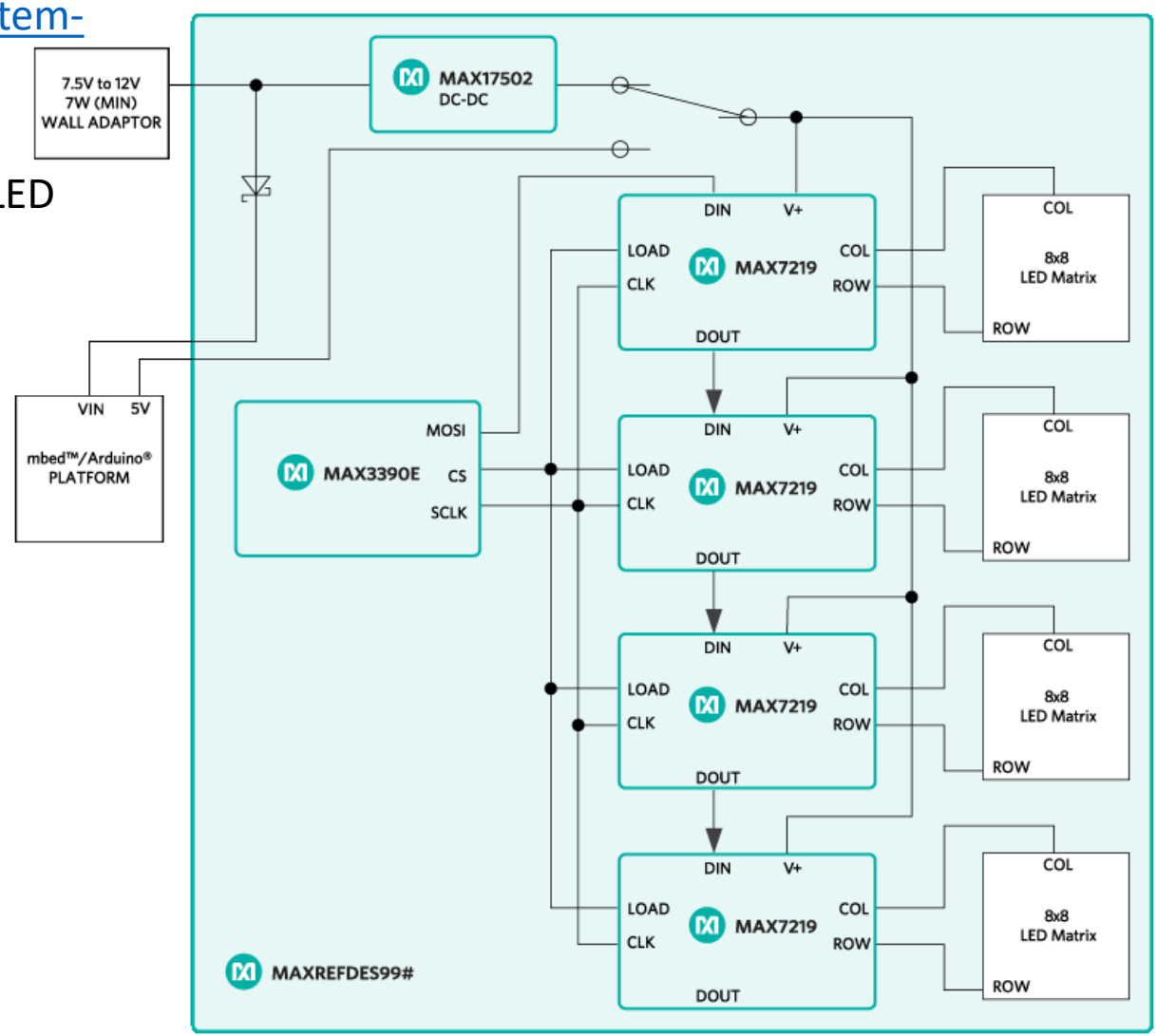
```
#include <LiquidCrystal440.h>
LiquidCrystal lcd(13,12,11,10,5,4,3,2); //RS, RW, E2, E1, D4, D5, D6, D7

void setup() {
  lcd.begin(40,4);
  lcd.setCursor(0,0);
  lcd.print("Zeile 1");
  lcd.setCursor(0,1);
  lcd.print("Zeile 2");
  lcd.setCursor(0,2);
  lcd.print("Zeile 3");
  lcd.setCursor(0,3);
  lcd.print("Zeile 4");
}
void loop() …..
```

Arduino Bus - 4fach 8x8 Matrix Shield MAXREFDES99# https://youtu.be/5n3Nx3rOnx8

https://www.maximintegrated.com/en/design/reference-design-center/system-board/6251.html/tb_tab0

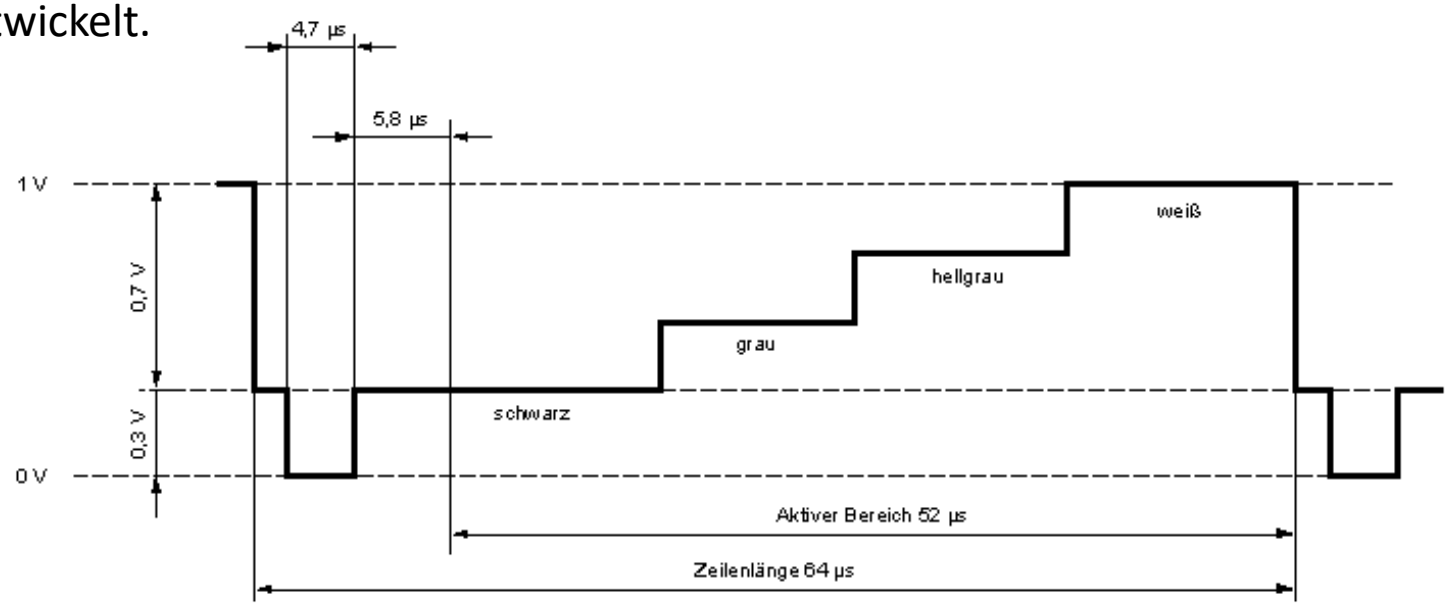
4 * MAX7219 Ics Steuern 4 LED Matrixen via Levelshifter IC MAX3390E und die Signale:
MOSI
SCK
CS



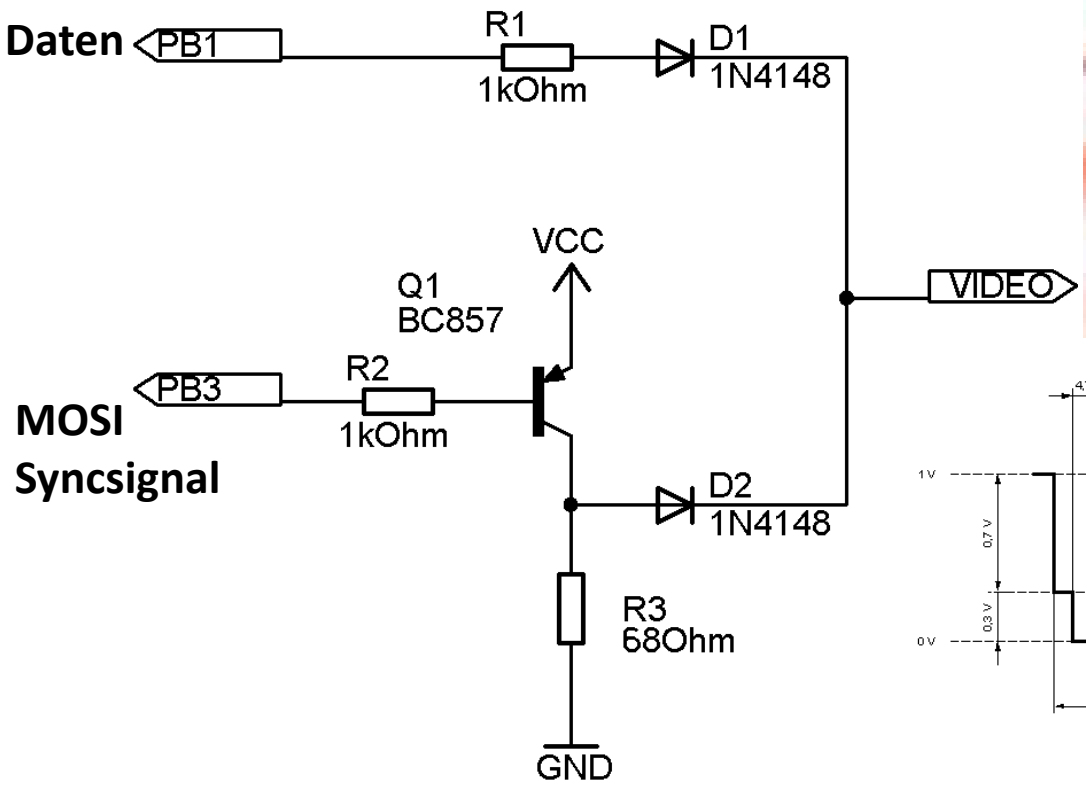


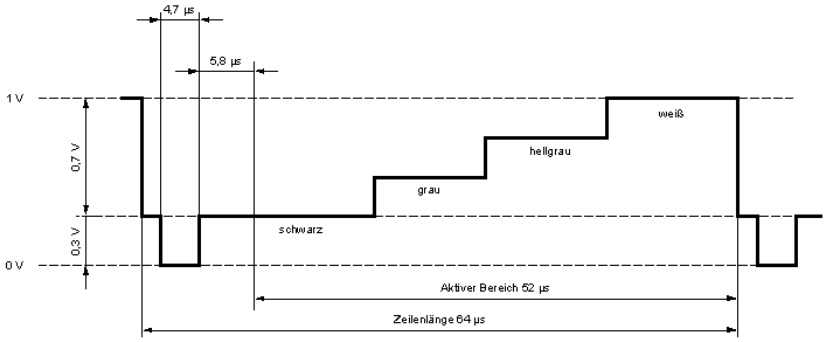
TV/VIDEO „BAS“ Signalsynthese

Ein klassisches altes TV (Fernseher) Signal (Gelber Stecker) nannte man **F-BAS Signal**. (Farb-BAS)
Alte Röhren-Fernseher hatten **625 Zeilen** * **25 Bilder**/50 Halbbilder = **15625 Hz** Zeilenfrequenz. Man konnte die ZF als Kind pfeifen hören.
1/15625Hz = **64µS**/Zeile. Dies können wir mit dem µC erzeugen! Nur stabil muss das Signal sein. Dazu kann man die In-Build Synchron-Shiftregister nutzen.
→ **MOSI**. Das TV-Signal wurde schon in den 50ern für das Schwarz-Weiß TV entwickelt.



Eine simple Transistor-Dioden Schaltung mischt Synchronisier- und Daten-Signale und sorgt für die notwendige Impedanz Anpassung.
(~5MHZ Bandbreite
Ein kleiner Aufbau ist zu empfehlen.





Etwas Flair der 80er Jahre eines ZX81. Aber ein normales TV kann nun als Anzeigemonitor dienen. In einem Kurs hatte ein Teilnehmer ein Tetris Spiel auf dieser Basis programmiert! Klasse.

Youtube: http://youtu.be/f7NBdHZe4Mo

In einem Kurs hatte ein Teilnehmer ein Tetris Spiel auf dieser Basis programmiert! Klasse.

Bedenke: Dies ist kein PC-Monitor Signal sondern ein TV Signal.

Sowas hatte ich nicht erwartet..

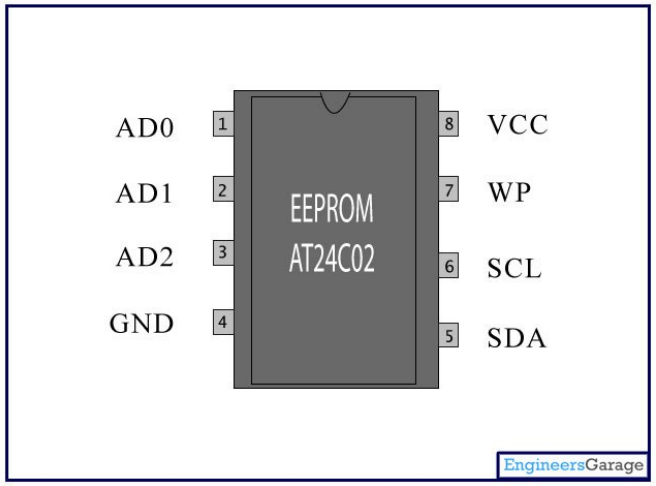
Kapitel 7

Praxis Erfahrungen im Einsatz innerer Komponenten wie EEPROM. Verschiedene Datenbus Methoden. Weitere Anwendungen externer Bibliotheken/Module und externer Hardware

EEPROM = electrically erasable programmable read only memory

Das EPROM ist ein kleiner innerer resistenter Speicher oder auch über **I2C** oder Synchronbus angeschlossenes externes IC. Bsp. Typ: 24C02 Internally Organized 128 x 8 (1K), 256 x 8 (2K), 512 x 8 (4K), 1024 x 8 (8K) or 2048 x 8 (16K)

24C02=Internally organized with 32 pages of 8 bytes each, the 2K requires an 8-bit data word address for random word addressing.



Dazu stellt die WINAVR Funktionsbibliotheken zur Verfügung:

NIEMALS zyklisch schreiben. Max 1000000 Schreibzugriffe / EEPROM = Lebensdauer.

PRAXIS-Erfahrung: unbedingt einen Supervisor = Spannungswächter Chip verwenden

Ein **Spannungseinbruch** (Auto Anlasser etc.) **produziert einen Datenverlust**! (Das war ein schwer rauszufindendes Problem in einem kommerziellen Gerät.

Tipp: Bekannte konstante Sequenz (Präfix) in den Datensatz schreiben z.B. 0x5AA5 → Erster Zugriff nach Einschalten:

Präfix fehlt → neuen default Datensatz schreiben, →lesen.

Präfix vorhanden → Nur lesen, in RAM Struktur schreiben bei Bedarf.

Änderung der Datenstrukturinhalte → „Präfix + Datensatz“ schreiben.

```
#include <avr/eeprom.h>

Defines
#define    EEMEM    __attribute__((section(".eeprom")))
#define    eeprom_is_ready()
#define    eeprom_busy_wait()   do {} while (!eeprom_is_ready())

Functions
uint8_t    eeprom_read_byte (const uint8_t *__p) __ATTR_PURE__
uint16_t   eeprom_read_word (const uint16_t *__p) __ATTR_PURE__
uint32_t   eeprom_read_dword (const uint32_t *__p) __ATTR_PURE__
float      eeprom_read_float (const float *__p) __ATTR_PURE__
void       eeprom_read_block (void *__dst, const void *__src, size_t __n)
void       eeprom_write_byte (uint8_t *__p, uint8_t __value)
void       eeprom_write_word (uint16_t *__p, uint16_t __value)
void       eeprom_write_dword (uint32_t *__p, uint32_t __value)
void       eeprom_write_float (float *__p, float __value)
void       eeprom_write_block (const void *__src, void *__dst, size_t __n)
void       eeprom_update_byte (uint8_t *__p, uint8_t __value)
void       eeprom_update_word (uint16_t *__p, uint16_t __value)
void       eeprom_update_dword (uint32_t *__p, uint32_t __value)
void       eeprom_update_float (float *__p, float __value)
void       eeprom_update_block (const void *__src, void *__dst, size_t __n)
```

```
#include <avr/eeprom.h>
#define EEPROM_SECTION  __attribute__ ((section (".eeprom")))
//  ** this global variables are stored in EEPROM  **
uint16_t  dummy       EEPROM_SECTION  = 0;      // avoid using lowest addresses
uint8_t   eeprom_var1  EEPROM_SECTION  = 1;      // EEPROM address 0002
uint8_t   eeprom_var2  EEPROM_SECTION  = 2;      // EEPROM address 0003
uint16_t  eeprom_var3  EEPROM_SECTION  = 1027;    // low byte = 0003, high = 0004
float     eeprom_var4  EEPROM_SECTION  = 1.3456;  // four byte float
int main(void)
{
   uint8_t   state1;      uint16_t  state2;     float     floatVar;
   DDRB   = 0xff;              // use all pins on port B for output
   PORTB  = 0xff;
   state1 = eeprom_read_byte( &eeprom_var1 );  //read variable from EEPROM addr 0002
   PORTB = ~state1;                                          //and output value (1) to port B
   eeprom_write_byte( &eeprom_var2, state1 );  //and write value to EEPROM addr 0003
   state2 = eeprom_read_word( &eeprom_var3 );//read 16 bit variable from EEPROM
   PORTB = ~(state2 & 0x00FF);   //output lower byte (3) to port B
// read float value (1.3456) from EEPROM
   eeprom_read_block( &floatVar, &eeprom_var4, sizeof(eeprom_var4) );
   if ( floatVar == 1.3456 ) PORTB = ~1;
```

Anhang unsortiert

ich weiß..
Kein Inhaltsverzeichnis!
Also ein Merkzettel Bereich

Nachschlagseite:

„makefile“ secrets für Arduino

```
MCU = atmega328p
F_CPU = 16000000
#---------------- Programming Options (avrdude) ----------------
AVRDUDE_PROGRAMMER = arduino
AVRDUDE_LOADERBAUD = 115200
AVRDUDE_PORT = com8          # programmer connected to serial device

AVRDUDE_WRITE_FLASH = -D -Uflash:w:$(TARGET).hex:I
AVRDUDE_VERBOSE = -v –v

AVRDUDE_FLAGS = -p$(MCU) -P$(AVRDUDE_PORT) -b$(AVRDUDE_LOADERBAUD) -c
$(AVRDUDE_PROGRAMMER)

        # ist nichts anderes als:
        #AVRDUDE_FLAGS = -p$(atmega328p) -P$(com8) -b$(115200) -c $(arduino)

#AVRDUDE_FLAGS += -C avrdude.conf
AVRDUDE_FLAGS += $(AVRDUDE_NO_VERIFY)
AVRDUDE_FLAGS += $(AVRDUDE_VERBOSE)
AVRDUDE_FLAGS += $(AVRDUDE_ERASE_COUNTER)
……
program: $(TARGET).hex $(AVRDUDE) $(AVRDUDE_FLAGS) $(AVRDUDE_WRITE_FLASH)
```

Nur zur Info: AVRDUDE „Ausgabe“ nach der Programmierung

"make.exe" programavrdude -patmega328p -Pcom8 -b115200 -c arduino -v -v -D -Uflash:w:main.hex:i

avrdude: Version 5.11, compiled on Sep 2 2011 at 19:38:36

Copyright (c) 2000-2005 Brian Dean, http://www.bdmicro.com/

Copyright (c) 2007-2009 Joerg Wunsch

System wide configuration file is "C:\WinAVR-20100110\bin\avrdude.conf"

Using Port : com8

Using Programmer : arduino

Overriding Baud Rate : 115200

AVR Part : ATMEGA328P

Chip Erase delay : 9000 us

PAGEL : PD7

BS2 : PC2

RESET disposition : dedicated

RETRY pulse : SCK

..Es gibt bei ATMega µCs noch etwas

FUSES

Ein schwieriges Kapitel.!
UND VORSICHT. Falsche Einstellungen der Fuses können den teuren Mikrocontroller „**bricken**“. (= ***in einen nutzlosen Ziegelstein verwandeln***)

Fuses sind Betriebsmodus Schalter die unabhängig vom Programm die Hardware und Funktionen des Mikrocontrollers beeinflussen,
Fuses können:
die Speichernutzung voreinstellen
die Funktionen einzelner Komponenten enable/disable
Wachtdog, Brownout Detection etc….
die **Taktquelle umschalten** auf Intern/extern Quarz VORSICHT !!!
Startupdelay Time auswählen
wiederbeschreiben/flashen sperren (Lock) !!! VORSICHT !!!
Auslesen sperren !!! VORSICHT !!!
Bootsector Sperren etc..
Es hilf nichts, nur ein „genauer“ Blick ins Handbuch schafft Klarheit…
Dort steht jedoch merkwürdiges.
Bit gesetzt → „1“ bedeutet: NICHT ausgewählt, oder kein Häkchen.
Bit gelöscht → „0“ bedeutet: Ausgewählt, oder Häkchen gesetzt.. Seltsam…!!

Die Bedeutung kann durch das Handbuch oder Googel erfahren werden. (kleine Kommentare sind ja schon da)

Manual fuse bits configuration

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.

Note: ☐ means unprogrammed (1); ☑ means programmed (0).

Bit	*Low*	*High*	*Extended*
7	☐ **CKDIV8** Divide clock by 8	☐ **RSTDISBL** External reset disable	
6	☐ **CKOUT** Clock output	☐ **DWEN** debugWIRE Enable	
5	☐ **SUT1** Select start-up time	☑ **SPIEN** Enable Serial programming and Data Downloading	
4	☐ **SUT0** Select start-up time	☐ **WDTON** Watchdog Timer Always On	
3	☐ **CKSEL3** Select Clock Source	☐ **EESAVE** EEPROM memory is preserved through chip erase	
2	☐ **CKSEL2** Select Clock Source	☐ **BOOTSZ1** Select boot size	☐ **BODLEVEL2** Brown-out Detector trigger level
1	☐ **CKSEL1** Select Clock Source	☐ **BOOTSZ0** Select boot size	☑ **BODLEVEL1** Brown-out Detector trigger level
0	☐ **CKSEL0** Select Clock Source	☑ **BOOTRST** Select reset vector	☐ **BODLEVEL0** Brown-out Detector trigger level

Interessante Bauteile

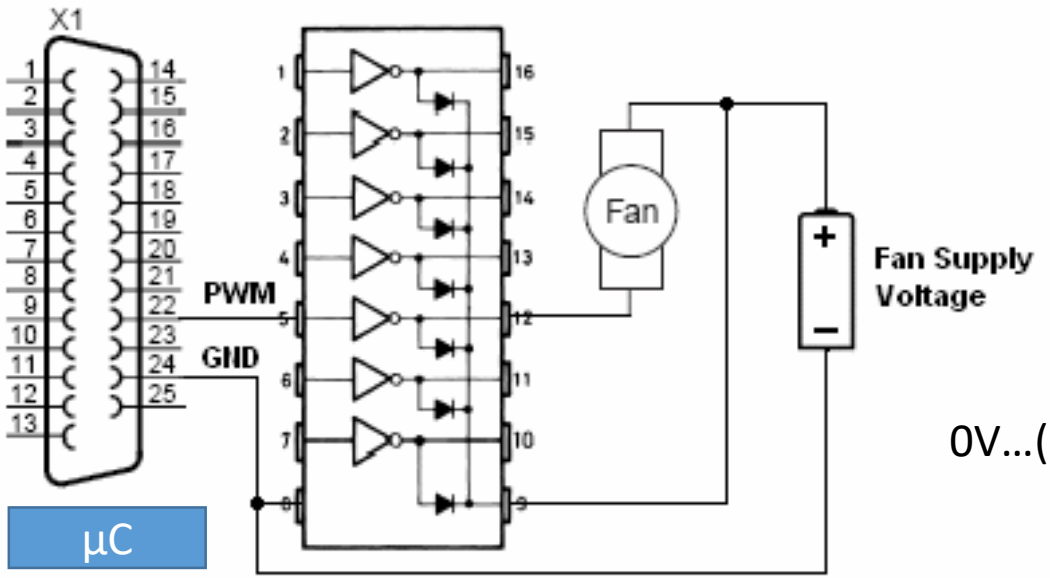
Aus der Welt der Robotik und der Industrie

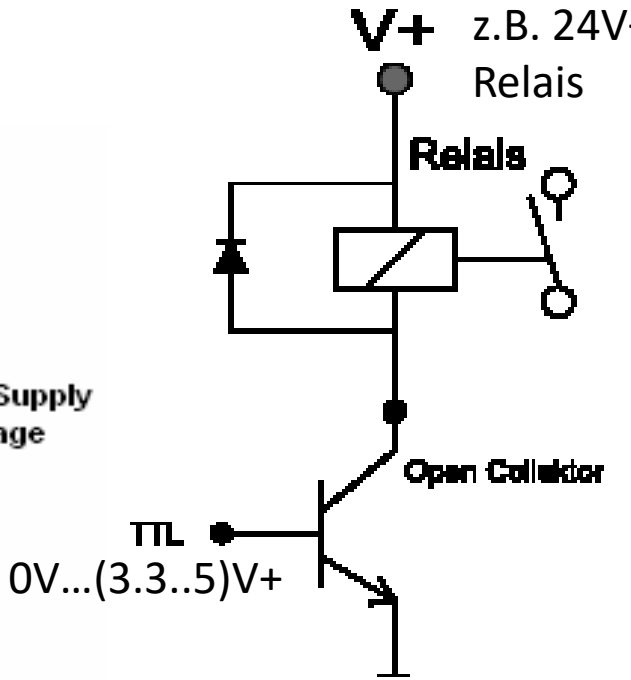
„Open Collector“ → OC Konzept

Höhere Spannung möglich.
z.B. 24V+ am Relais

ULN2003

Transistor Array IC





Der **Trick** : eine direkte TTL (+5V) Schaltspannug vom µController, die den **µC**ontroller so gut wie NICHT belastet, kann einen hohen Strom an einer viel höheren Spannung schalten. Das ohne jeglichen Widerstand in der Verbidnung von µC zum ULNxxxx IC.

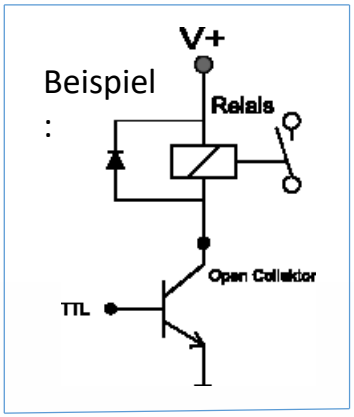
Pro Transistor kann durchaus ~500mA im Schaltbetrieb genutzt werden.
Das reicht auch für kleine Motoren durchaus aus.

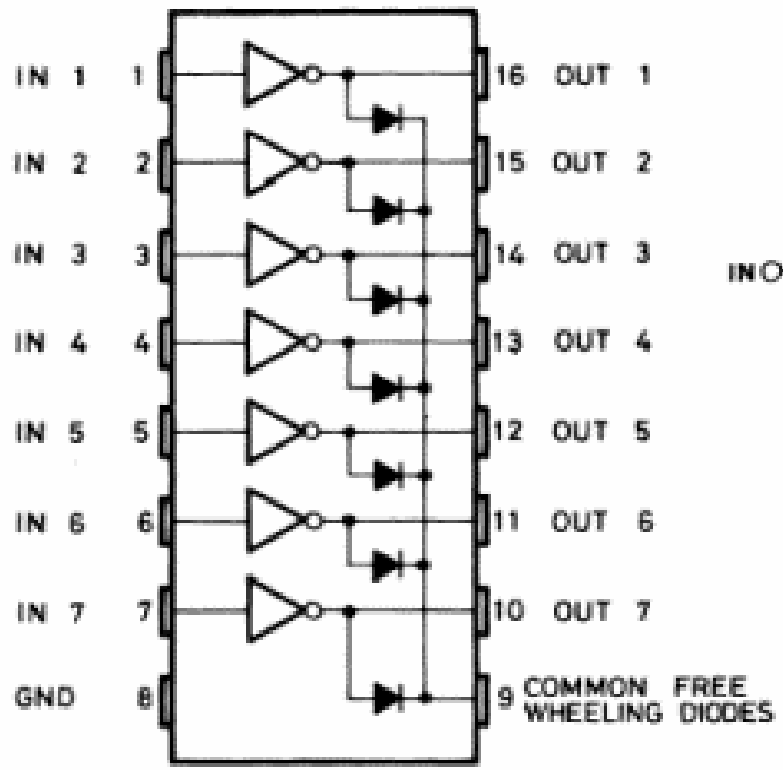
ULN2003 = 7 NPN Transistor Array IC

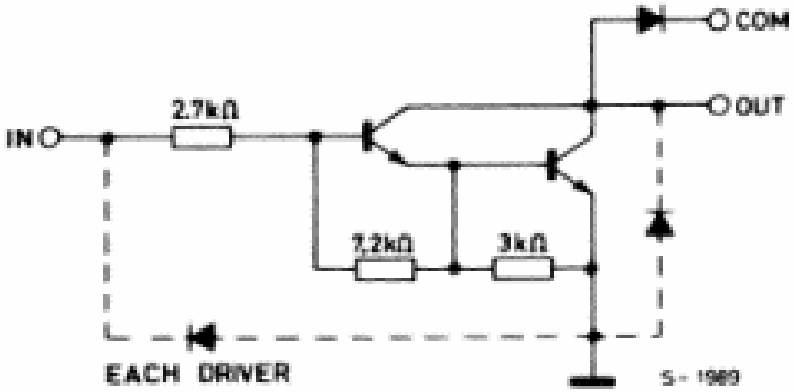
….wie auf der Motorplatine verbaut

Man nennt es auch „**low side switch array** “

„Open Collector“ **mit 7 Darlington Transistoren**
und integrierten Freilauf Dioden.
Damit kann man 7 Bauteile, die
1.) hohen Strom fordern und
2.) die an V+ liegen mit einem „s**chwachen**“ µC
Steuersignal auf GND, und somit AN schalten.





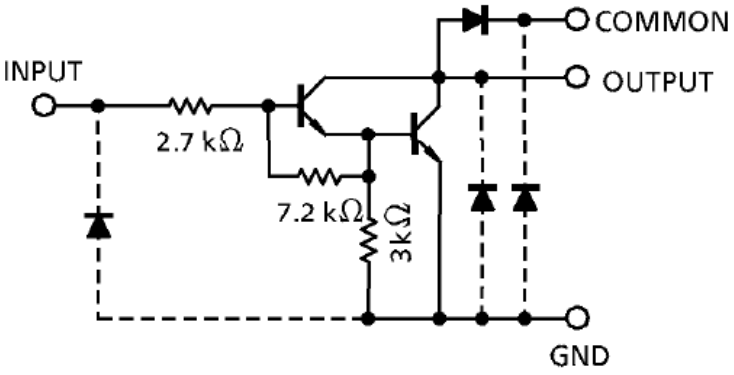


ULN2003 (each driver)

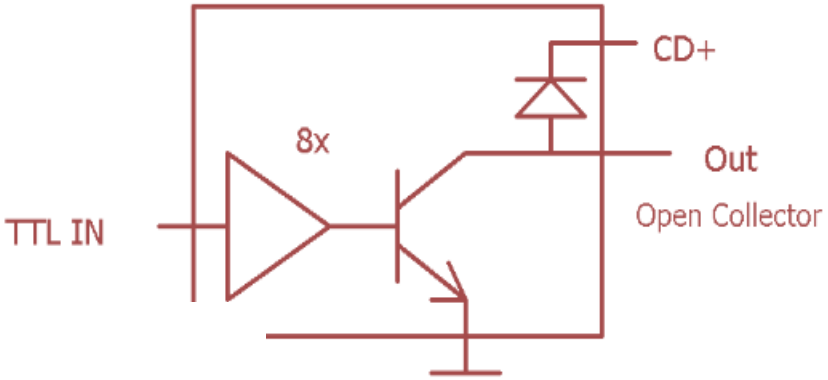
ULN2803 = 8 NPN Transistor Array IC

Man nennt es auch „**low side switch array** “

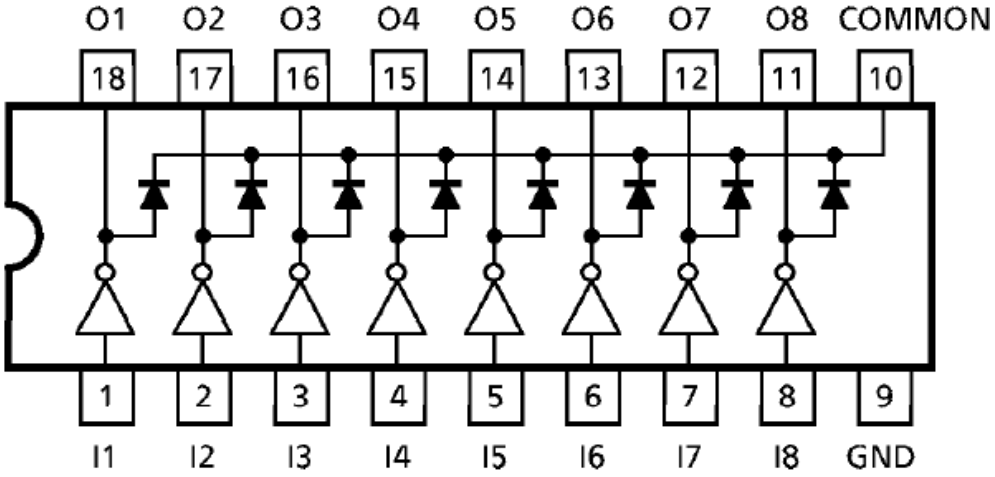
ULN2803AP / AFW



Im ULN2803 sind an allen Ausgängen Dioden in Sperrichtung.
Die Kathoden sind zusammengeführt und gemeinsam an CD+ verfügbar.
Damit können dann induktive Lasten (Relais) angeschlossen werden.

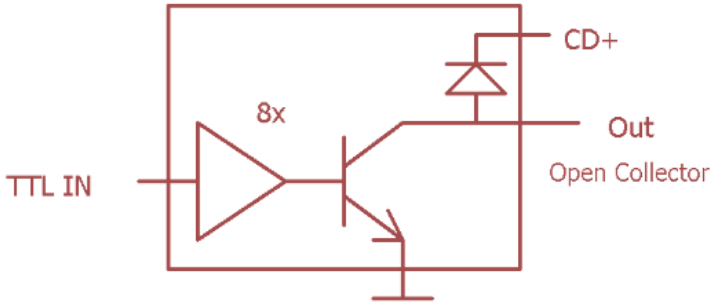


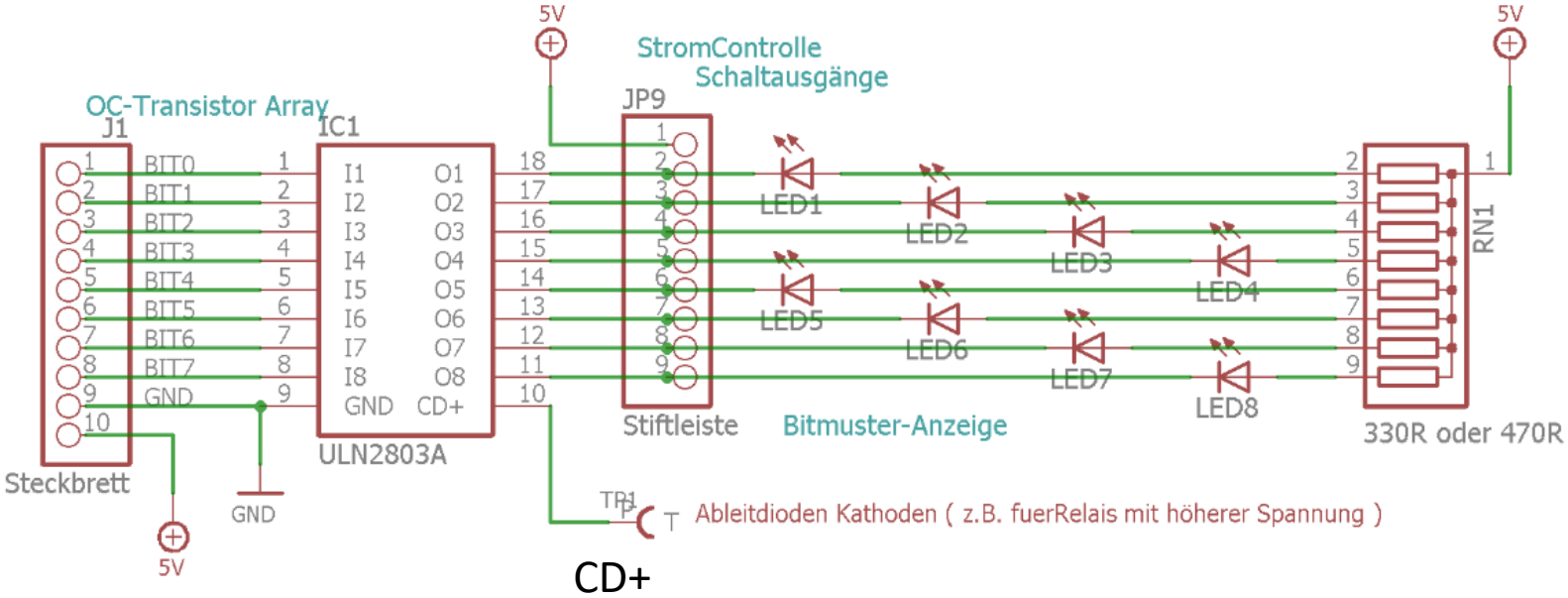
PIN CONNECTION (TOP VIEW)



Anwendung als LED Treiber. So kann der µController für somit in gefahrlos, **viele stromhungrige Aktoren**, auch Motoren betreiben

Im ULN2803 sind an allen Ausgängen Dioden in Sperrichtung.
Die Kathoden sind zusammengeführt und gemeinsam an CD+ verfügbar.
Damit können dann induktive Lasten (Relais) angeschlossen werden.



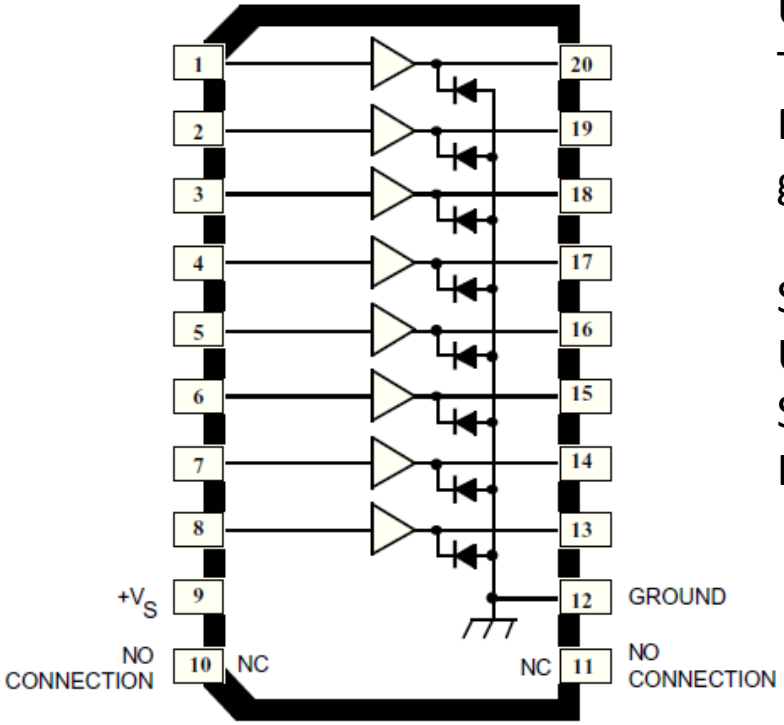


Das gibt es auch **als Positiv Spannungs Treiber** (z.B. für Matrixen)

UDN2981 = **8 PNP** Transistor Array IC

Man nennt es auch „**high side switch array** “

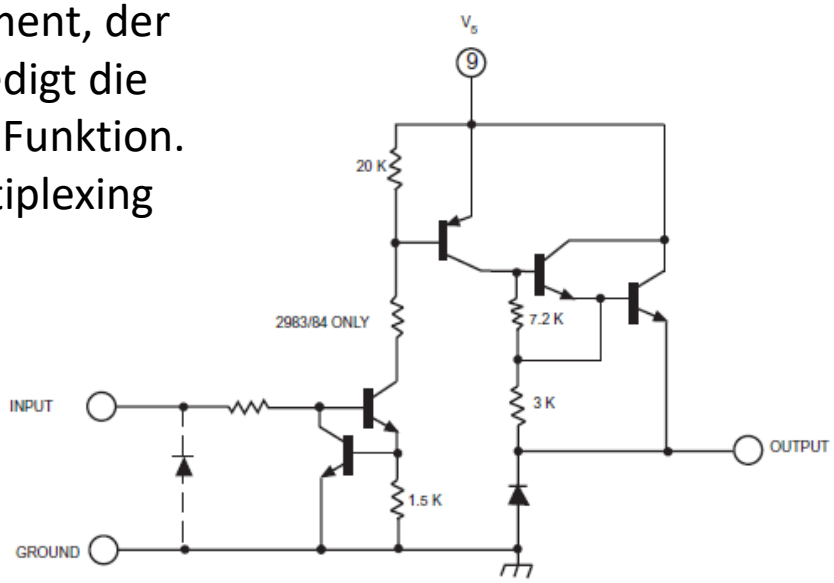
A2982SLW and A2984SLW



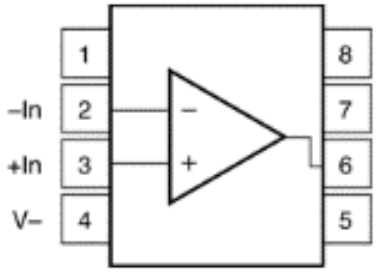
Der etwas teurere ~2€ UDN2801 POSTIV Treiber wird gerne für Matrix Ansteuerungen gebraucht.

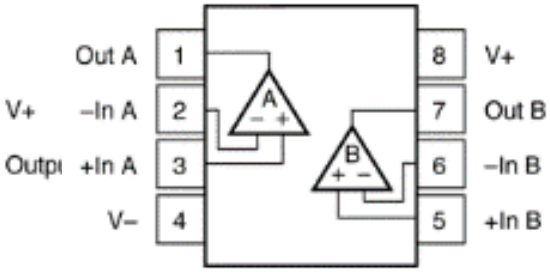
Sein Komplement, der ULN2801 erledigt die Sink-Current Funktion. Ideal für Multiplexing

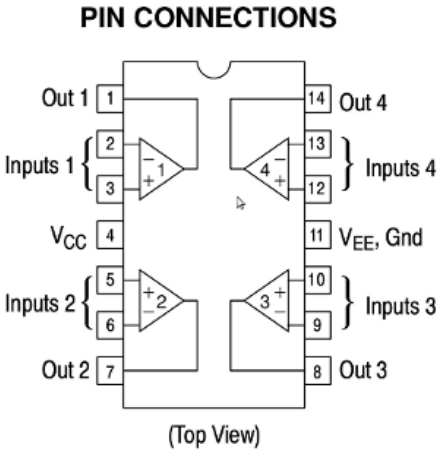
One of Eight Drivers

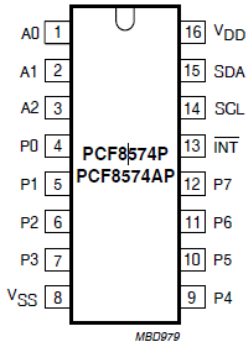


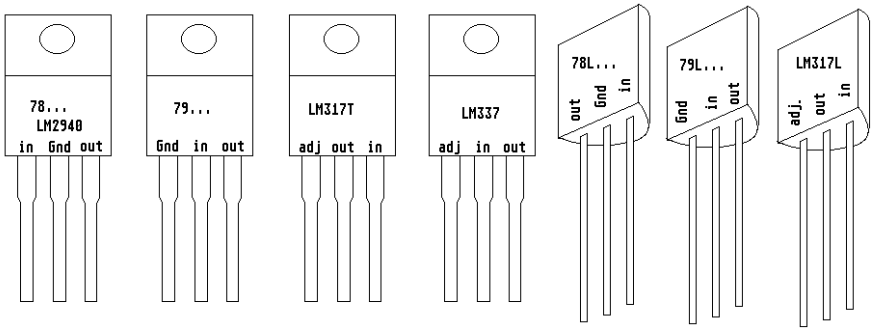
Pinbelegungen gängiger ICS





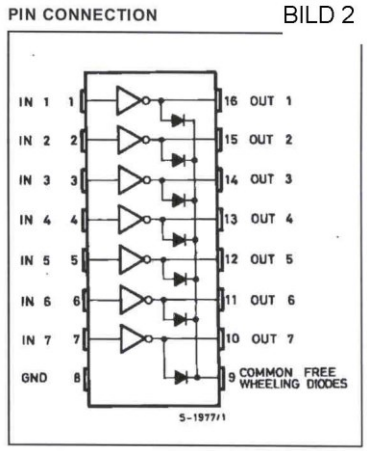




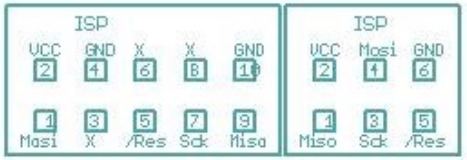


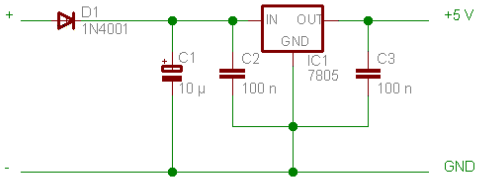
Anschlußbelegung der TO220-Gehäuse

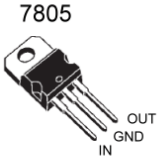
Anschlußbelegung der TO92-Gehäuse

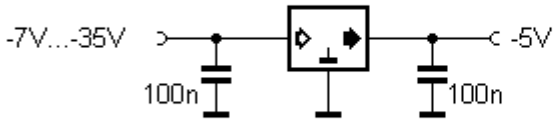


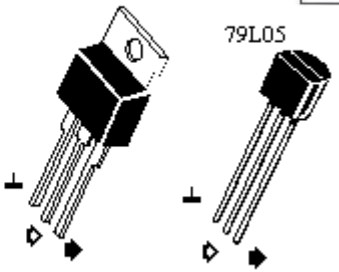
ULN2803



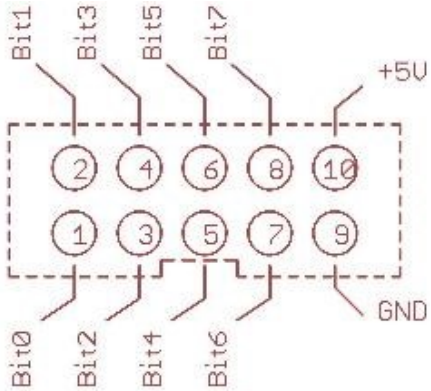








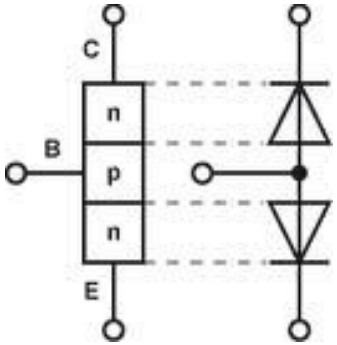
Pfostenreihe

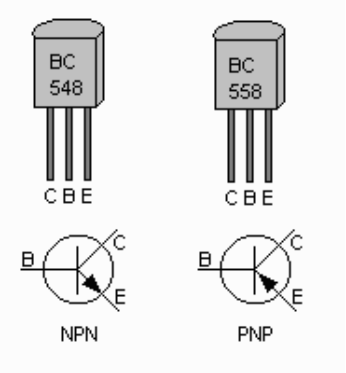


Eine kleine Erwähnung der Eigenschaften von Transistoren sei hier erlaubt.

Transistoren haben drei Anschlüsse, entsprechend drei gepolten Schichten

Die Reihenfolge ist NPN oder PNP. Man kann sich diese wie gegenpolig geschaltete Dioden vorstellen und wie Dioden genau ‚So' testen.





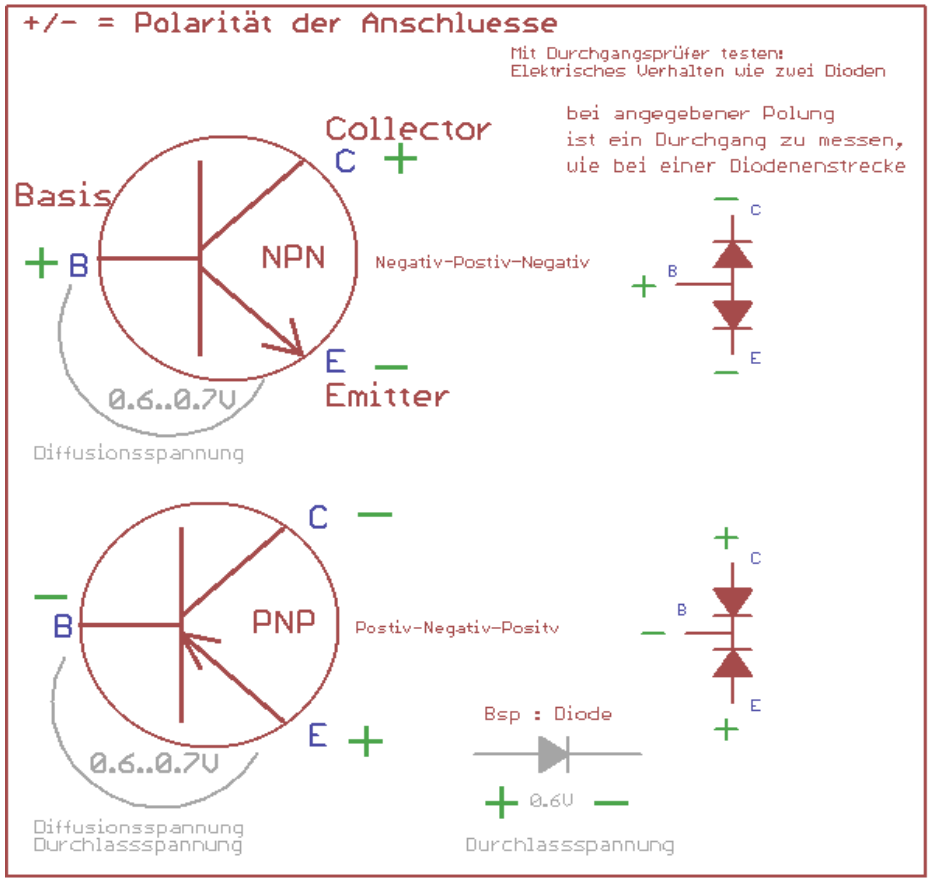
Zum Glück interessiert uns in der Digitalwelt nur der AN und AUS Zustand.

Dieser wird meist durch TTL Pegel:

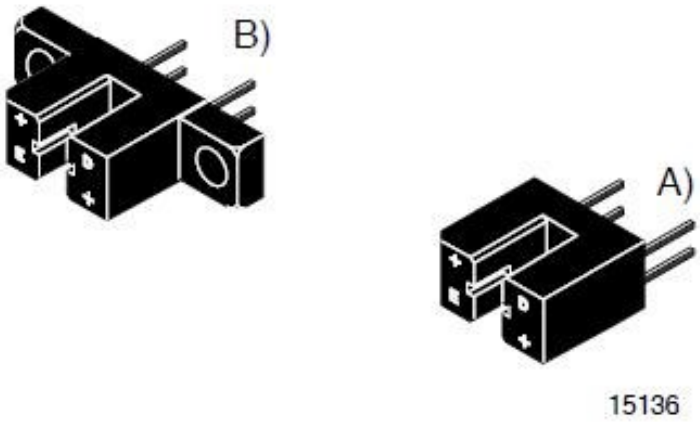
1 = 5 Volt Spannung

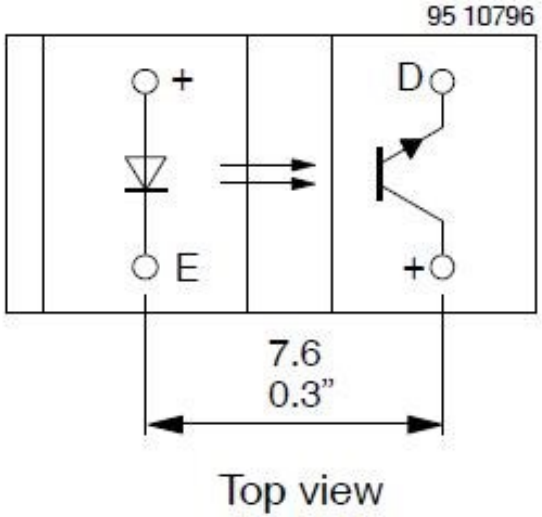
0 = 0 Volt Spannung

repräsentiert.

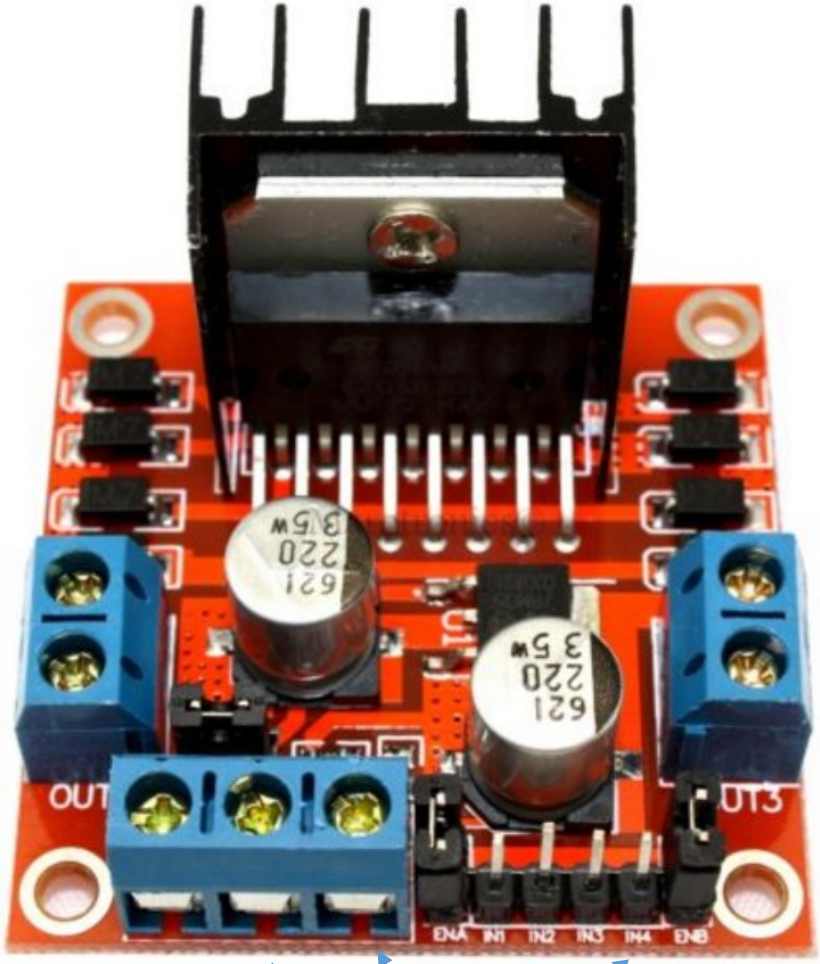


Gabellichtschranke





Neueres L298N H-Brückenmodel (China)

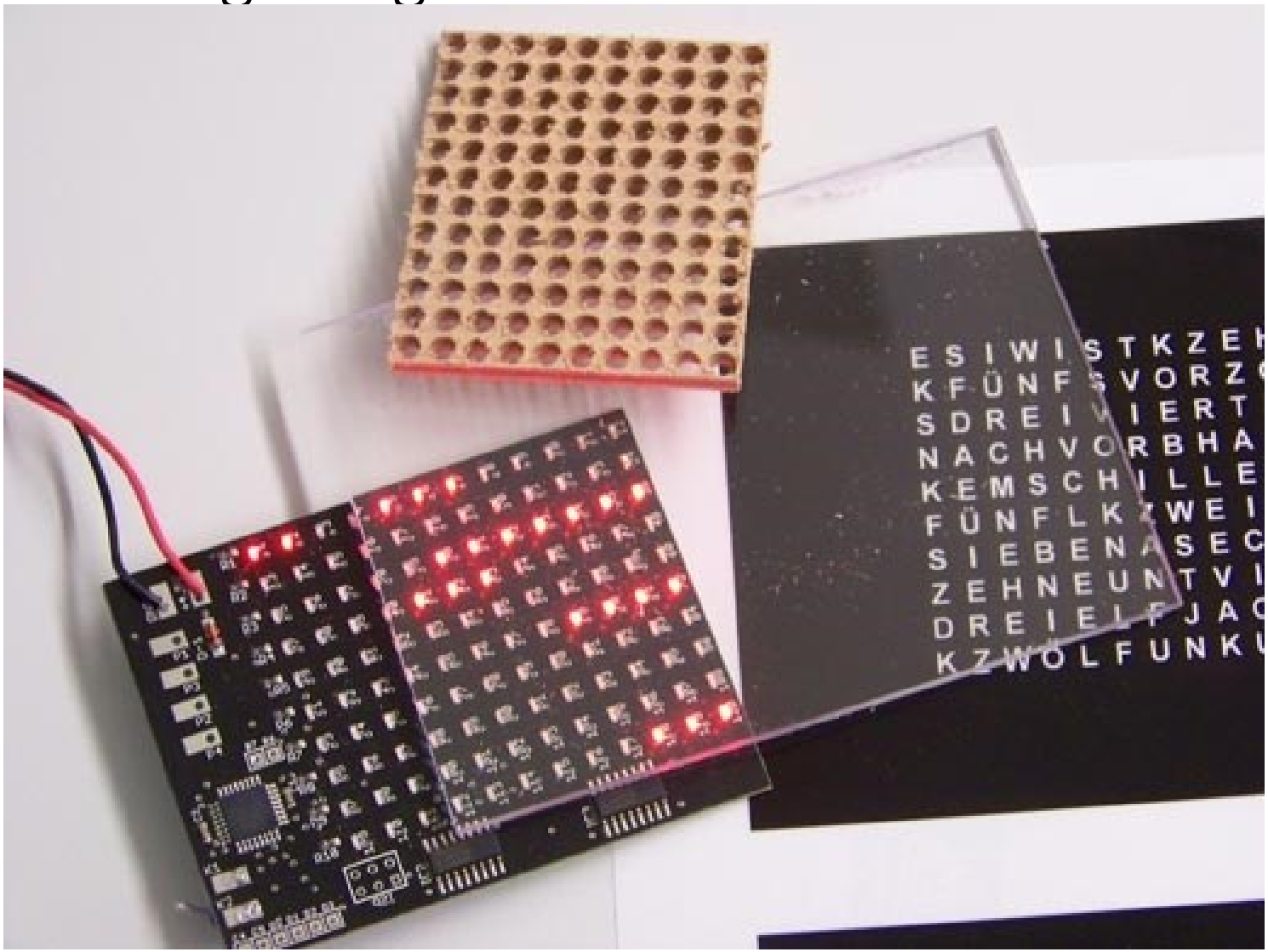


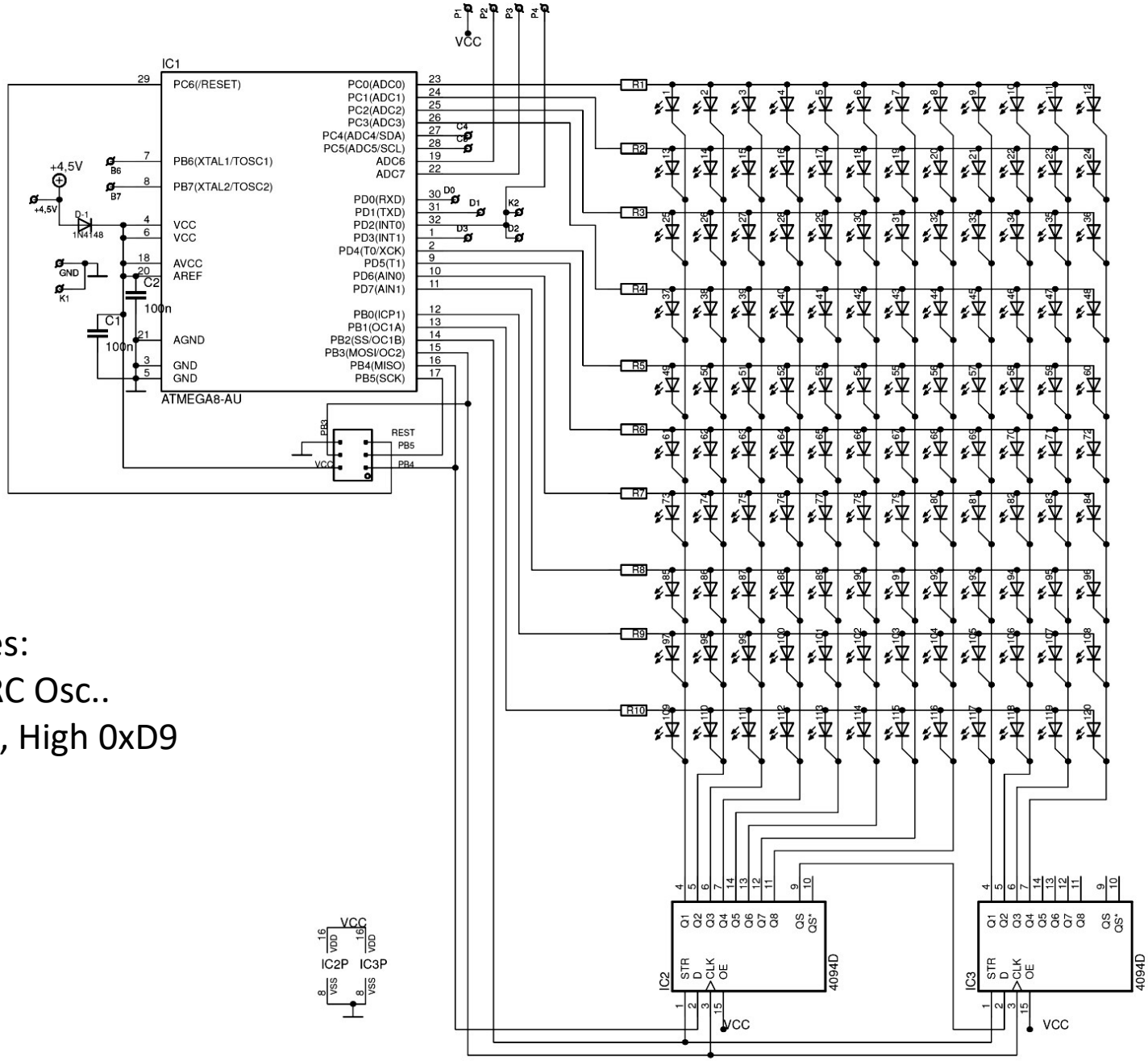
Versorgung: 8..~24V
VORSICHT: Verlustleistung beachten (Kühlkörper ist klein).
5V Klemme = Output !! KienEingang

+12V-IN GND +5V-OUT

ENA In1, IN2, In3, IN4, ENB

Retro Ping-Pong Board



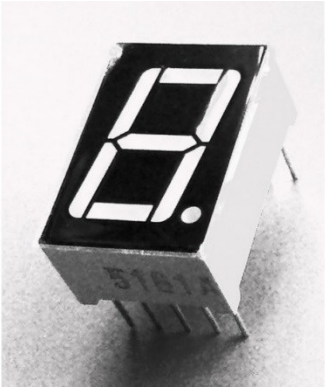


ORG Fuses:

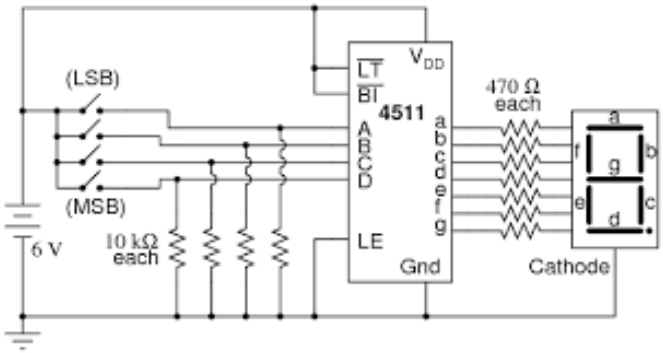
8MHZ – RC Osc..

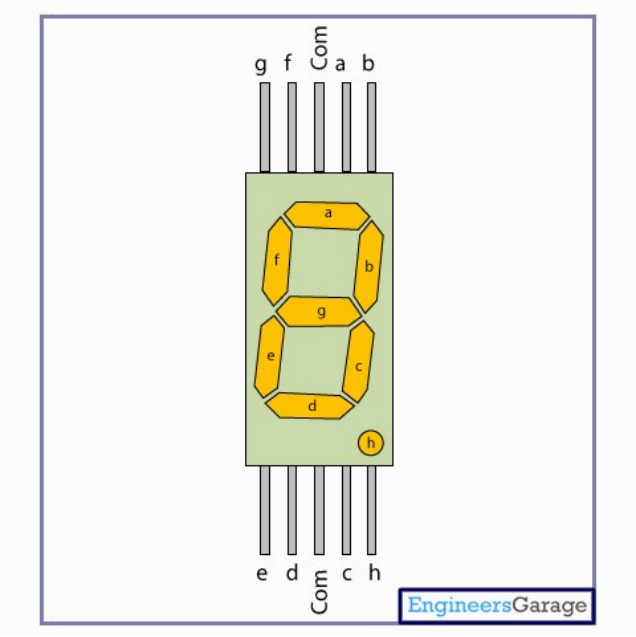
Low 0xE4, High 0xD9

7 Segment 5161AS

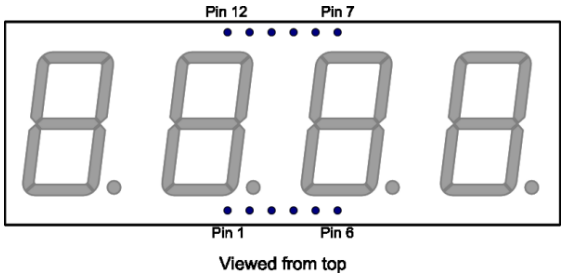


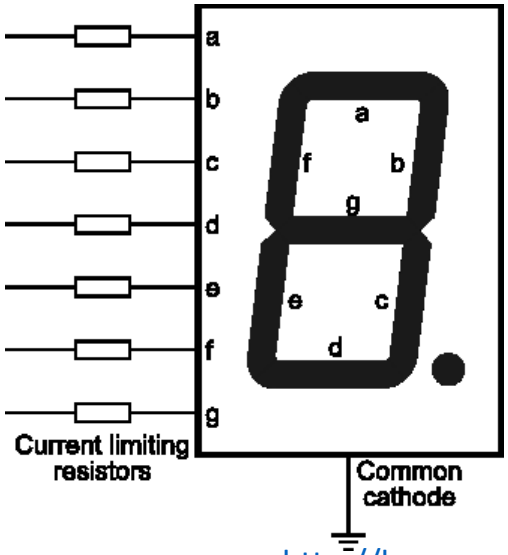
Es gibt 7 Segment Decoder ICs.
TLL 7447 mit Anoden Common
TTL 7446 mit Kathoden Common
CMOS 4511 mit Kathoden Common

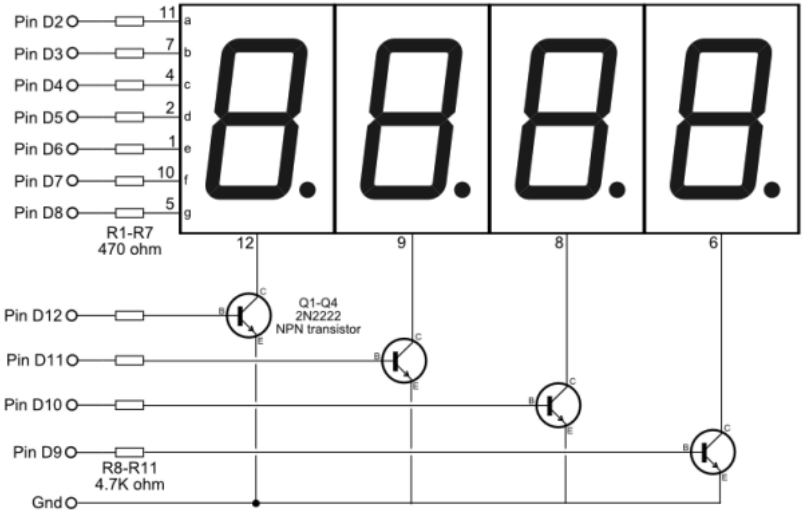




HS420561K-32 4fach Nummernanzeige



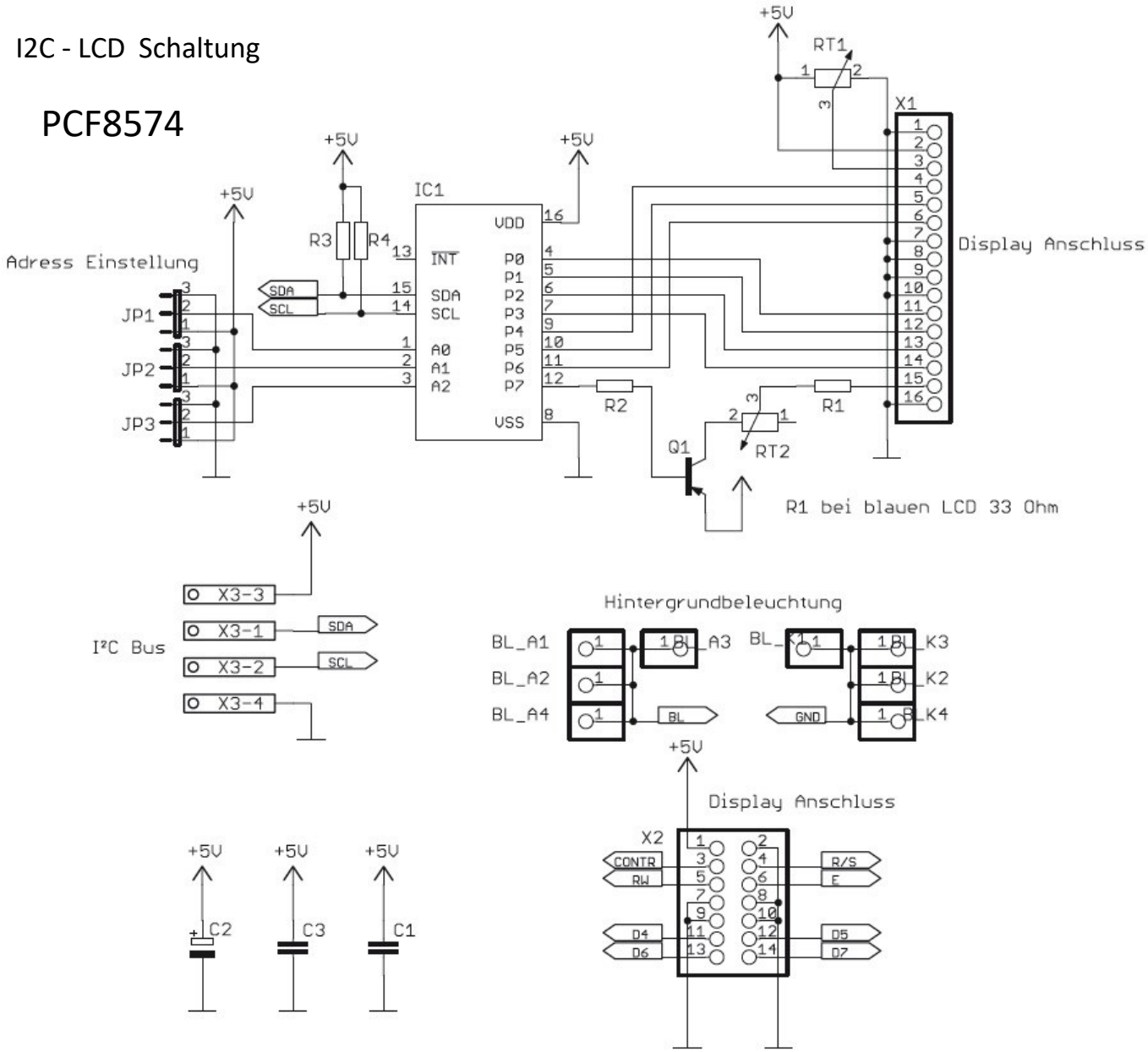




http://learn.parallax.com/4-digit-7-segment-led-display-arduino-demo

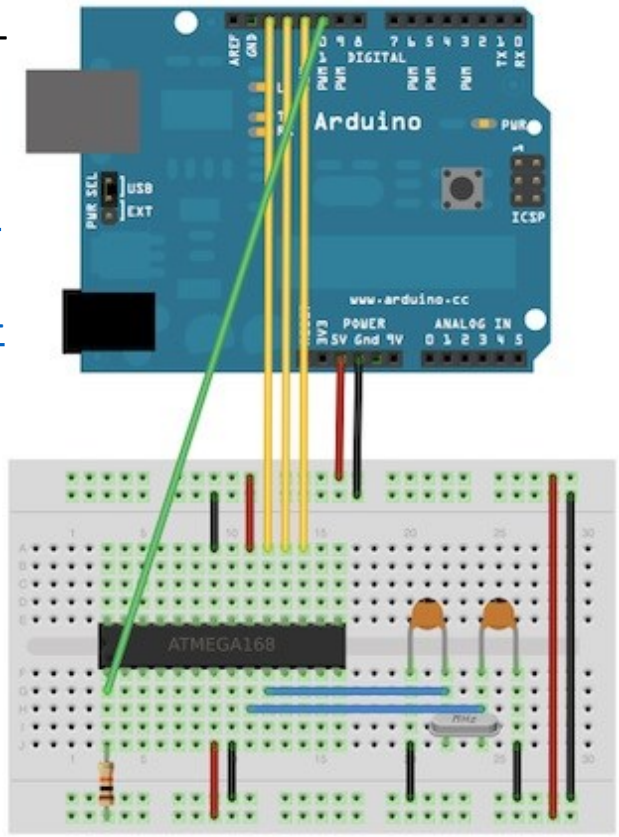
I2C - LCD Schaltung

PCF8574



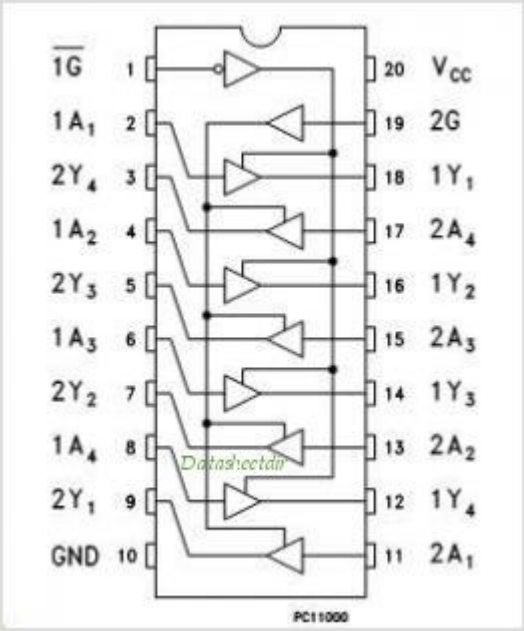
Plan B: ARDUINO als ISP umfunktioniert

Trick. Einen zweiten Arduino kann man als USB-Programmer missbrauchen, indem man das Sketch-Programm **Arduino als ISP** auf-flasht. https://learn.sparkfun.com/tutorials/installing-an-arduino-bootloader oder http://www.avrfreaks.net/forum/using-winavr-arduino



BUS Buffer with 3 State Outputs (non Inverted)

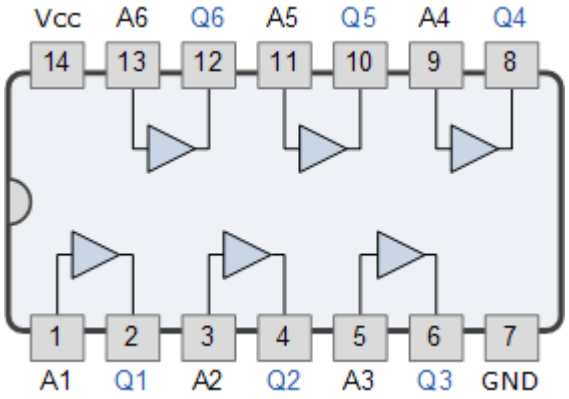
74HC241



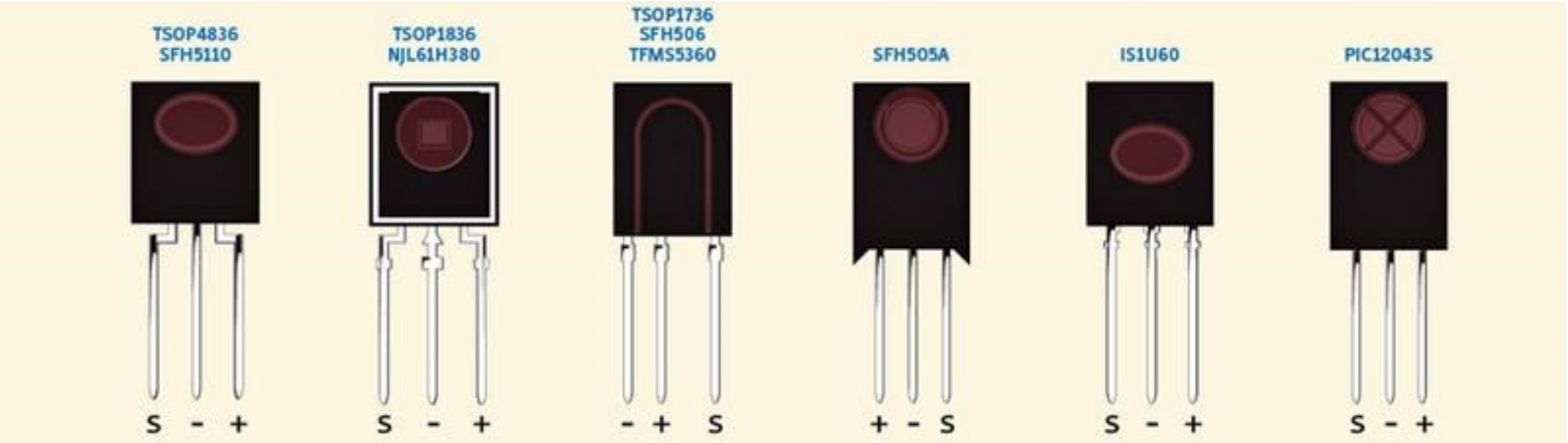
3 State = High, Low, Hochohmig

Für den Betrieb ist dann die Freigabe erforderlich

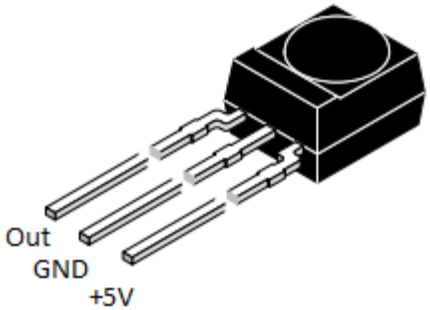
74HC07 OC

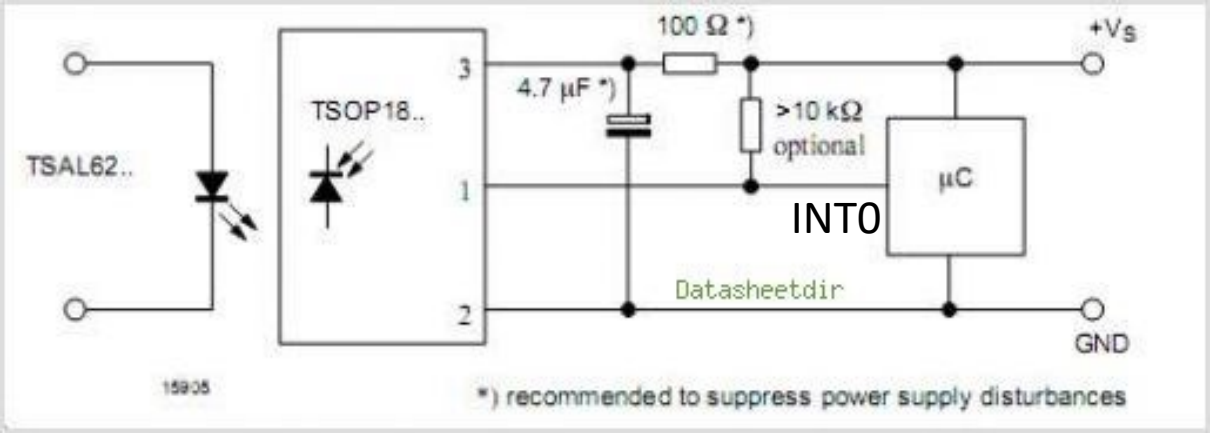


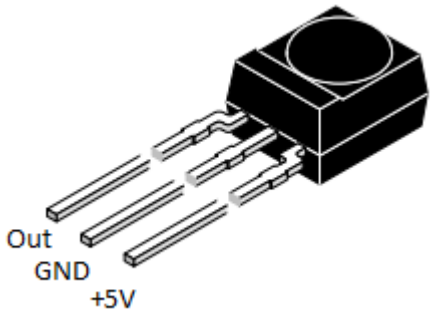
https://www.mikrocontroller.net/articles/74xx

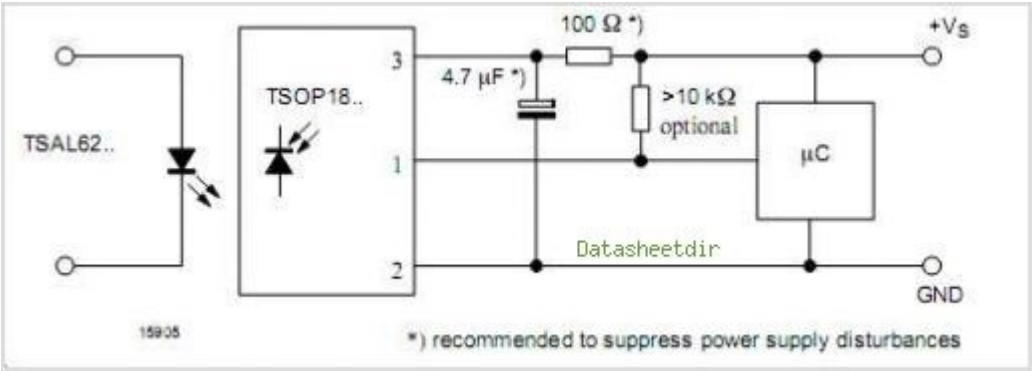


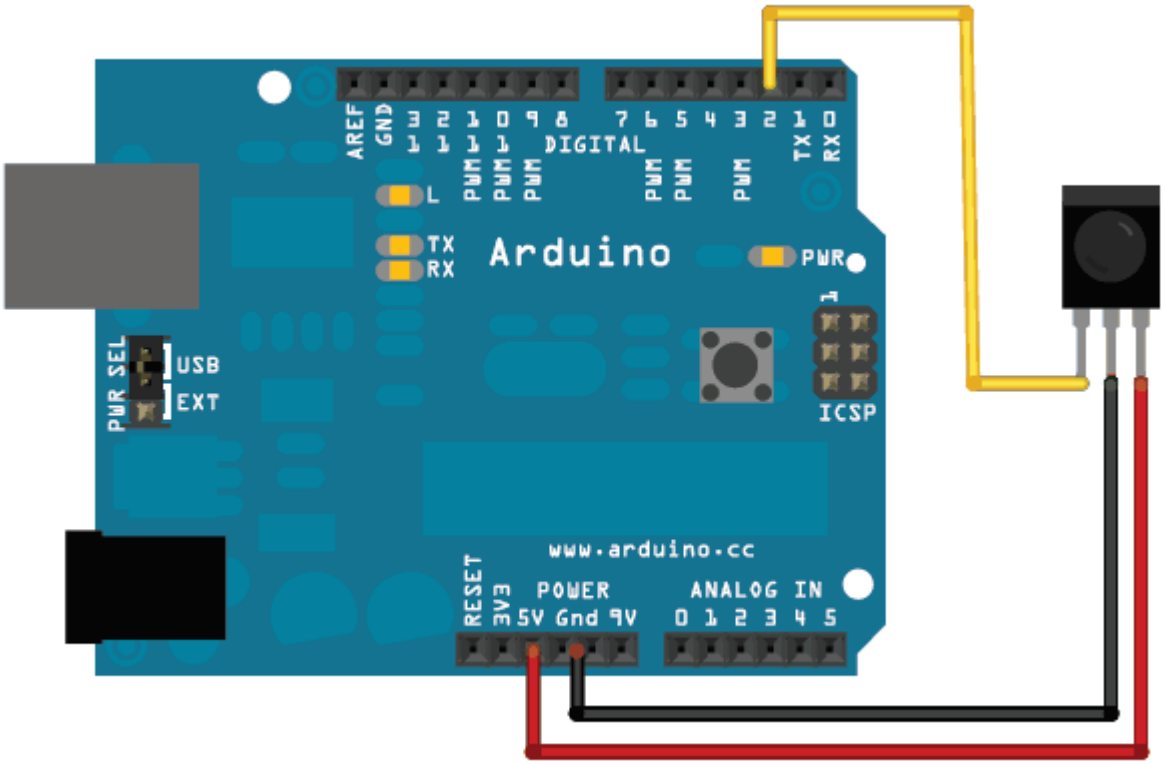
TSOP1838T liegt dem Kurs-Arduino Kit bei











TSOP1838 an
INT0 = PD2

2-Achs-Schwenkkopf FPV Kamera w / Dual Servo 9g / Lenkgetriebe für Roboter / r / c Auto - Schwarz + Blau #03890008

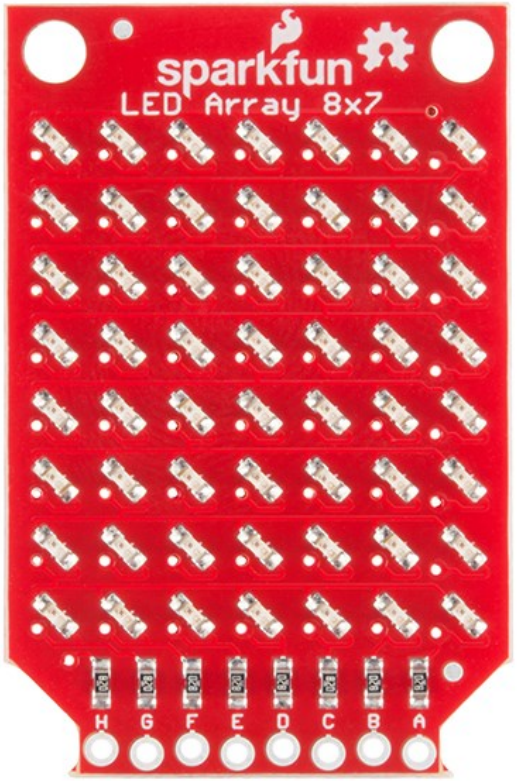
diverse Quellen wie China:
http://www.miniinthebox.com/

Modell LDTR-393879
~10€ mit Servos

LED ARRAY 8x7

Vorwiderstände integriert.
Aber , Niemals ohne MULTIPLEXING betreiben.
Vorwiderstände sind für Multiplexing dimensioniert.
Immer nur eine Lampe gleichzeitig,.
Oder „gemultiplext“ eine Reihe gleichzeitig

1
2
3
4
5
6
7
8



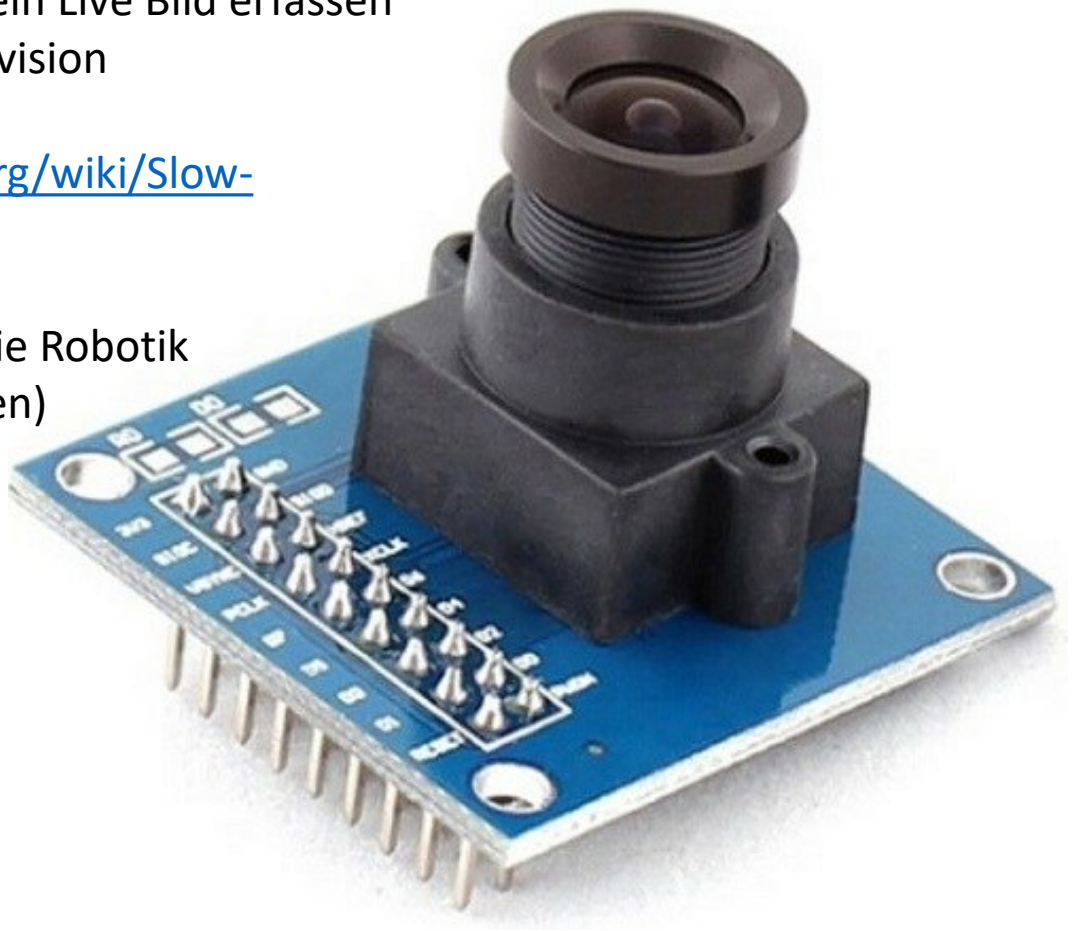
A=+5V → (A)..B..H= 0V → oberste Zeile rechts nach links
B=+5V → A..(B)..H= 0V → 2. Zeile rechts nach links
C=+5V → A..(C)..H= 0V → 3. Zeile rechts nach links
D=+5V → A..(D)..H= 0V → 4. Zeile rechts nach links
E=+5V → A..(E)..H= 0V → 5. Zeile rechts nach links
F=+5V → A..(F)..H= 0V → 6. Zeile rechts nach links
G=+5V → A..(G)..H= 0V → 7. Zeile rechts nach links
H=+5V → A..(H)..H= 0V → 8. Zeile rechts nach links

xxxxx01 = LED A-1
xxxxx01 = LED B1
xxxx0x1Zeilenweise mit 1 Selektieren
0 Aktiviert die LED. E
EINE 1 aktiviert LED an der falschen Stelle !
DAHER --Vorsicht Lösung:
Tristate (das zugehörige DDRx Bit = 0, Pullup Off = LED AUS

I2C Bus & parallel Bus
VGA Kamera Modul (Bucht ~3€)

Hier könnte man z.B. ein Live Bild erfassen und via Slow-scan Television Übertragen.
https://en.wikipedia.org/wiki/Slow-scan_television

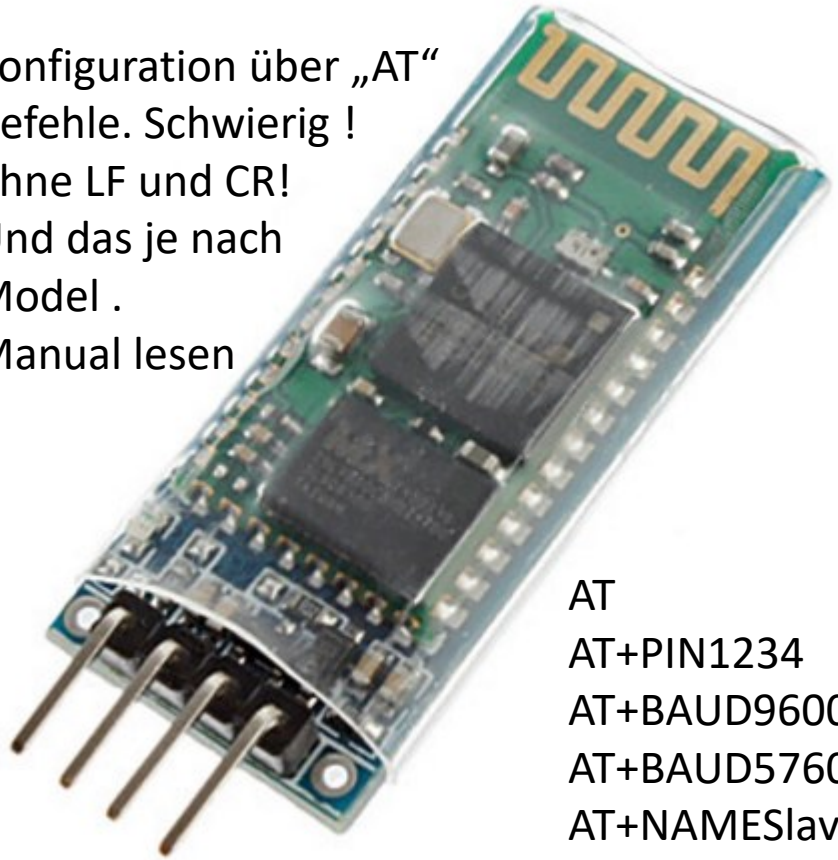
Ebenso denke ich an die Robotik (Erfassung von Objekten) usw.



2-Axis FPV Camera Cradle Head + OV7670 Camera Set for Robot / R/C Car - Black + Blue #03890009

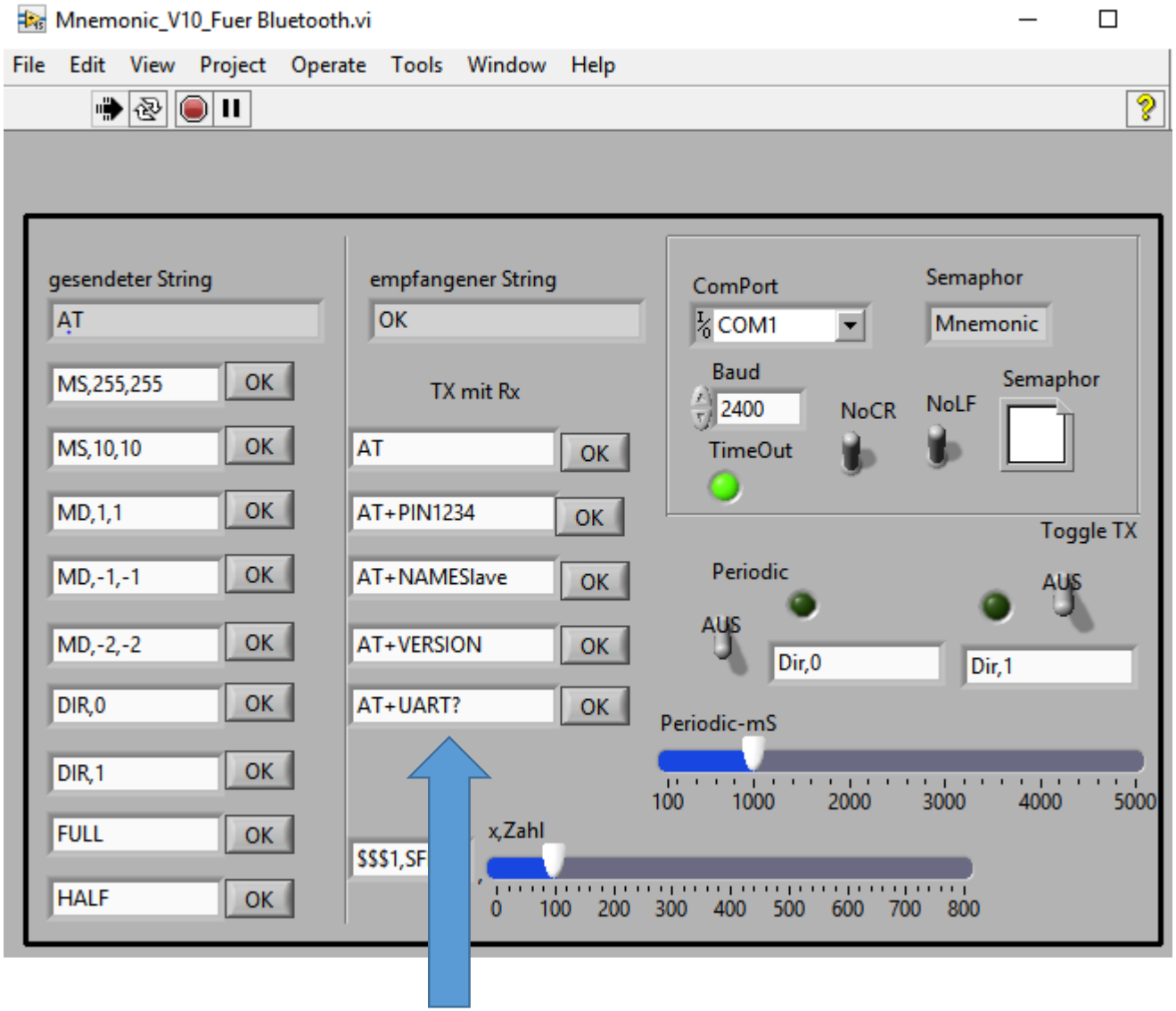
HC-06 Wireless Bluetooth Transceiver RF Main Module Serial #00903460

Konfiguration über „AT“ Befehle. Schwierig ! ohne LF und CR! Und das je nach Model . Manual lesen



AT
AT+PIN1234
AT+BAUD9600
AT+BAUD57600
AT+NAMESlave
AT+VERSION
AT+ROLE=0

„AT“ Kommando:
LabView für Bluetooth Konfiguration



Extra VI das „Schnell“ via RS232 TTL 3.3V ohne CR und LF die Kommandos sendet.

AT
AT+PIN1234
AT+VERSION
AT+NAMESlave
AT+ROLE=0

Basic AT commands

Command Return Parameter Description
AT OK None Test
AT+VERSION? +VERSION:<Param> OK Param: Version number
Get the soft version
AT+ORGL OK None Restore default status
AT+ADDR? +ADDR: <Param> OK Param: Bluetooth address Get module Bluetooth address
AT+NAME=<Param> OK Param: Bluetooth device name Set device's name
AT+NAME? +NAME:<Param> OK Param: Bluetooth device name Inquire device's name
AT+ROLE=<Param> OK Param:0=Slave role; 1=Master role; 2=Slave-Loop role
Set module role
AT+ ROLE? + ROLE:<Param> Param:0=Slave role; 1=Master role; 2=Slave-Loop role
Inquire module role
AT+UART=<Param>,<Param2>,<Param3> OK Param1: baud rate(bits/s); Param2: stop bit; Param3: parity bit Set serial parameter
AT+ UART? +UART=<Param>,<Param2>,<Param3> OK Param1: baud rate(bits/s); Param2: stop bit; Param3: parity bit Inquire serial parameter

To change HC-05 baud rate from default 9600 to 115200, 1 stop bit, 0 parity enter: "AT+UART=115200,1,0“ oder AT+UART=2400,3,1

Sensoren

aller Art

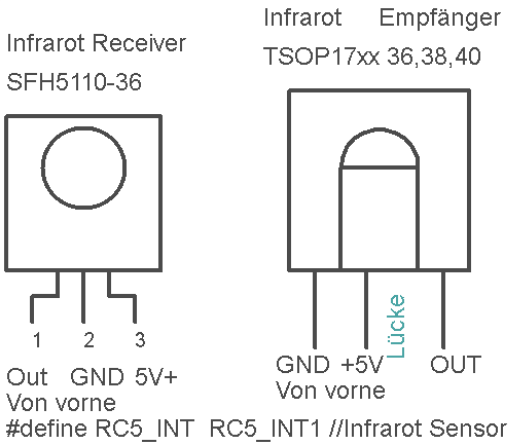
Register INTx und IR-Remote

+

RC5 Philips Protokoll:

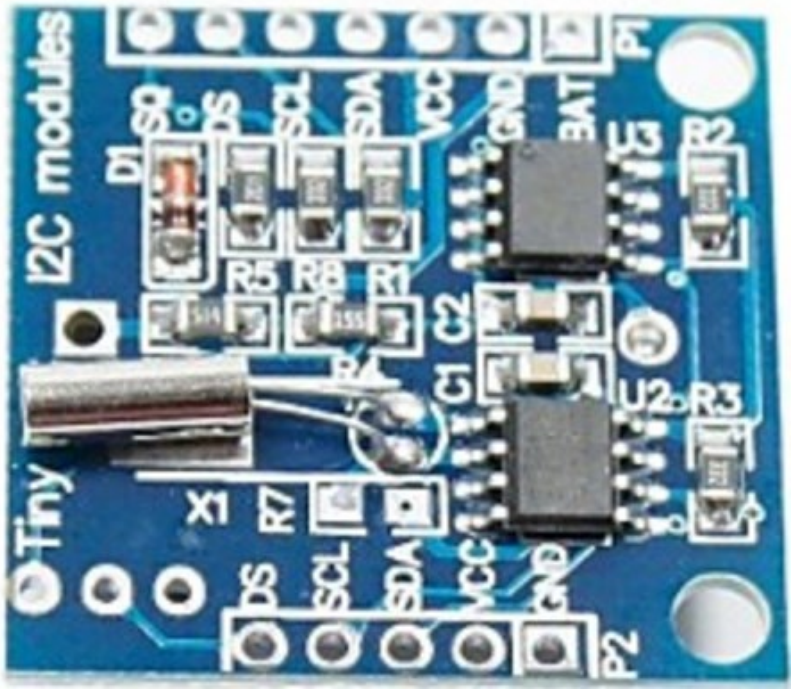
INT0 Pin am ATMega x für den Empfänger
ist durch Umbau schon frei.

#define RC5_INT RC5_INT1
„rc5.c rc5.h“



Eine einfache Switch-Case Liste setzt Kommandos in Aktionen um..
Version beachten für ATM328- Austausch der Register ATM16 → ATM328
GIFR → EIFR, GICR →EIMSK, TIFR→ TIFR0, TIMSK =→TIMSK0, MCUCR → EICRA

2c RTC DS1307 Echtzeituhr-Modul für (für Arduino) (1 x LIR2032) #01141502

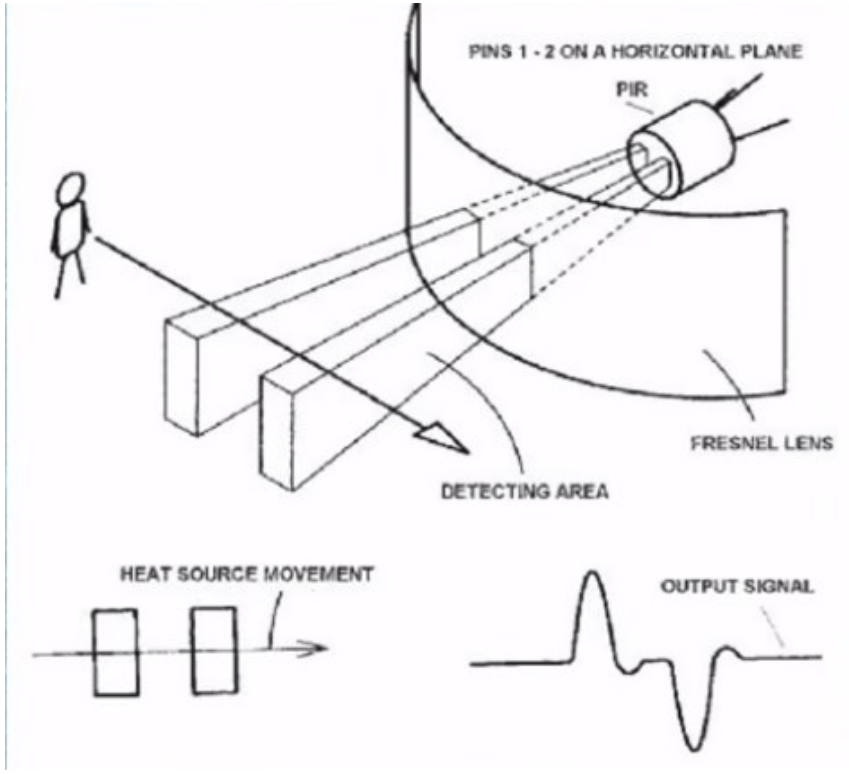


Modell: d1207003
Stückzahl: 1
Farbe: blau
Material: FR4 Leiterplatten
Features: zwei Stahl i2c-Schnittstelle; Stunden: Minuten: Sekunden am / pm Tag Monat, Tag - Jahr, Schaltjahr Kompensation; genaue Kalender bis zum Jahr 2100, basierend DS1307 RTC mit LIR2032 Batterie (Batterie enthalten), 56 Byte von nicht-flüchtigen Speicher für Anwender zur Verfügung, Arbeitszeit: 1 Jahr
Anwendung: für DIY-Projekt

Infrarot „PIR“ -Sensor

Körperwärme-Detektor:

Entweder eigens Modul bauen oder gekauftes Modul verwenden.
Erkennung: einfach
Richtungsortung … nicht ganz so einfach..
(Wie Vektor bestimmen ?)
Probleme: Abschirm-Blickwinkel?

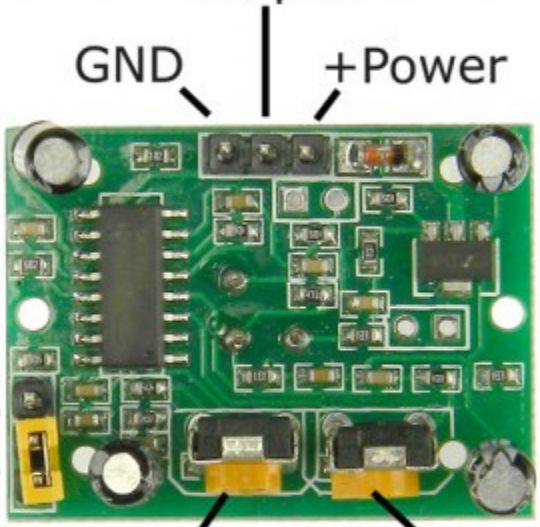


D-SUN HC-SR501 **PIR** MOTION DETECTOR

https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/testing-a-pir

(Quelle China Bucht)

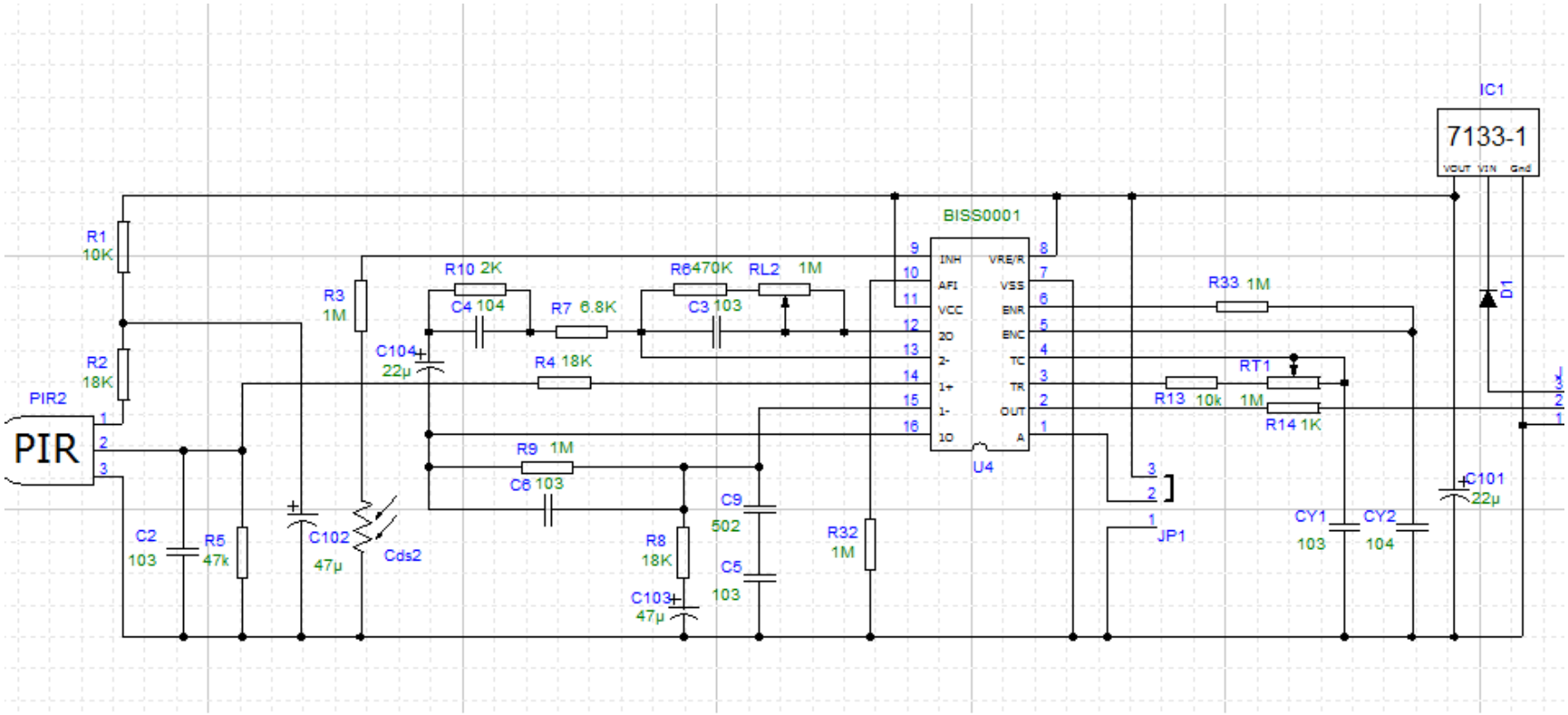
Supply 4.5..20V
3.3V Out



Sense Regler
3..to7 Meter
clockwise makes it more sensitive.

Time Regler 5 to 200Sec
Linksrum = kürzer
Mitte=120sec
1Min= ¼ von Links

PIR DSNFIR800b Schematic



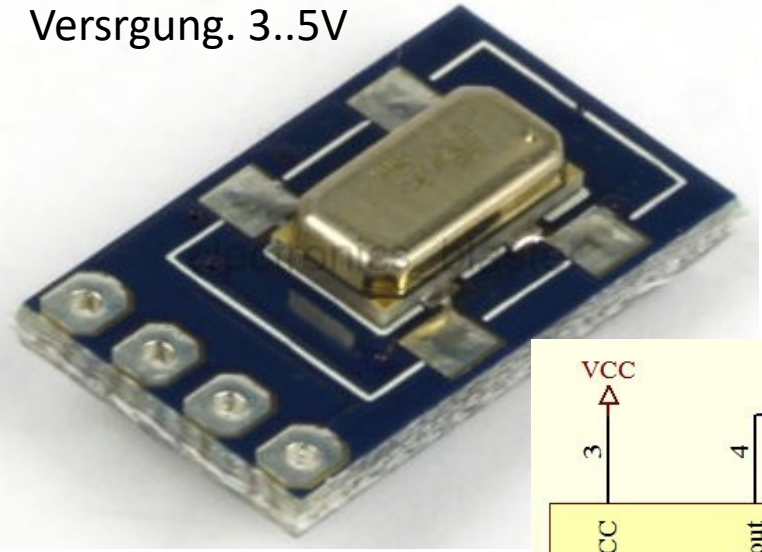
digitale Temperatur / Feuchte-Messtestmodul für Arduino Modell LDTR-0022

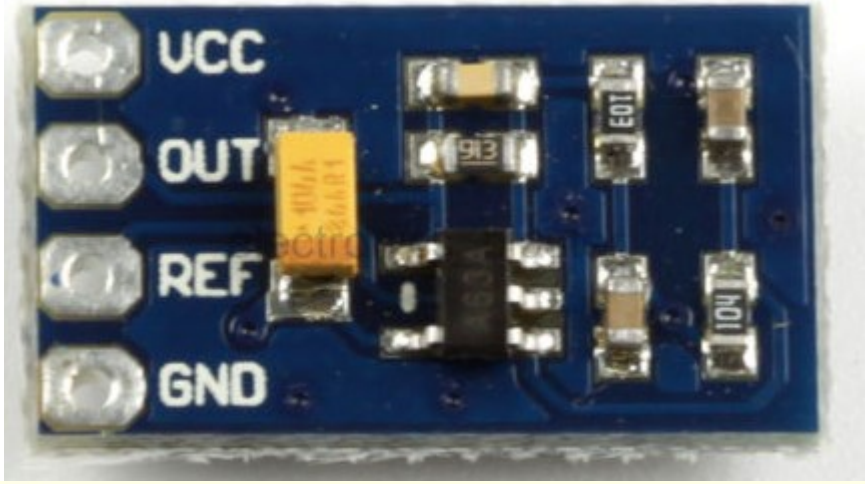
Haupt

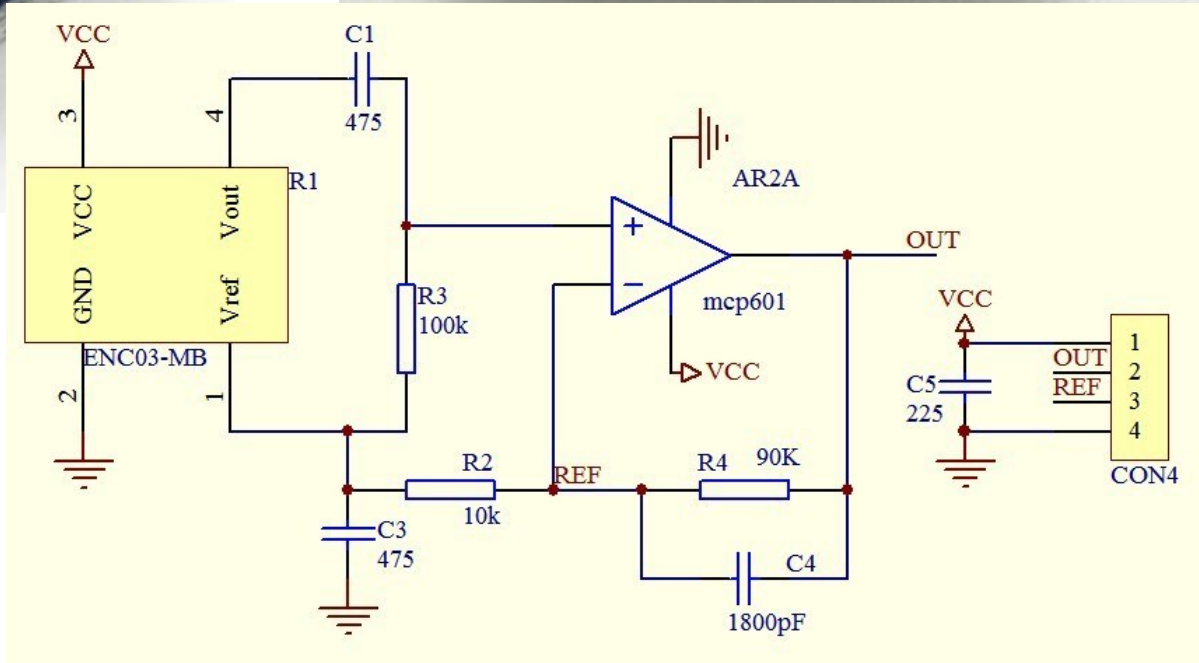
Anwendung: Temperatur und Luftfeuchtigkeit und Ardino Test erkennt
Ausgang: single / Bus-Digitalsignal;
Messbereich: Luftfeuchtigkeit: 20 ~ 90% rF, Temperatur: 0 ~ 50'c; Genauigkeit: Luftfeuchtigkeit: + / 5% relative Luftfeuchtigkeit, Temperatur: +/- 2'c; Auflösung: 1% rF, Temperatur: 1'c;
Installation Loch Platz: 2.7cm; Installation Loch-Durchmesser: 0,3 cm
Betriebsspannung: DC 3.3 ~ 5 v

GY-35-RC Single Axis Gyroscope Analog Output Gyro Module 3..5V for Arduino

Ein-Achs Sensor mit
Analogem Datenausgang
Versrgung. 3..5V







Schaltung auf der
kleinen Platine..

F04878 MPU-6050 Modul 3-Achsen-Gyro-Sensor Analog + Beschleunigungsmodu

Dieser 3_Achs Gyrosensor wird über ein µController **I2C BUS** Protokoll angesteuert. Der I2C Bus hat die Signale: **SDA** und **SCL**. Das Manual PDF zu lesen ist unbedingt erforderlich. Googele nach:

„MPU-6050_DataSheet_V3 4“

VCC = VDD = 2.375V-3.46V → **3.3V**

GND = 0V

SCL = I2C Bus Clock

SDA = I2C Bus Data

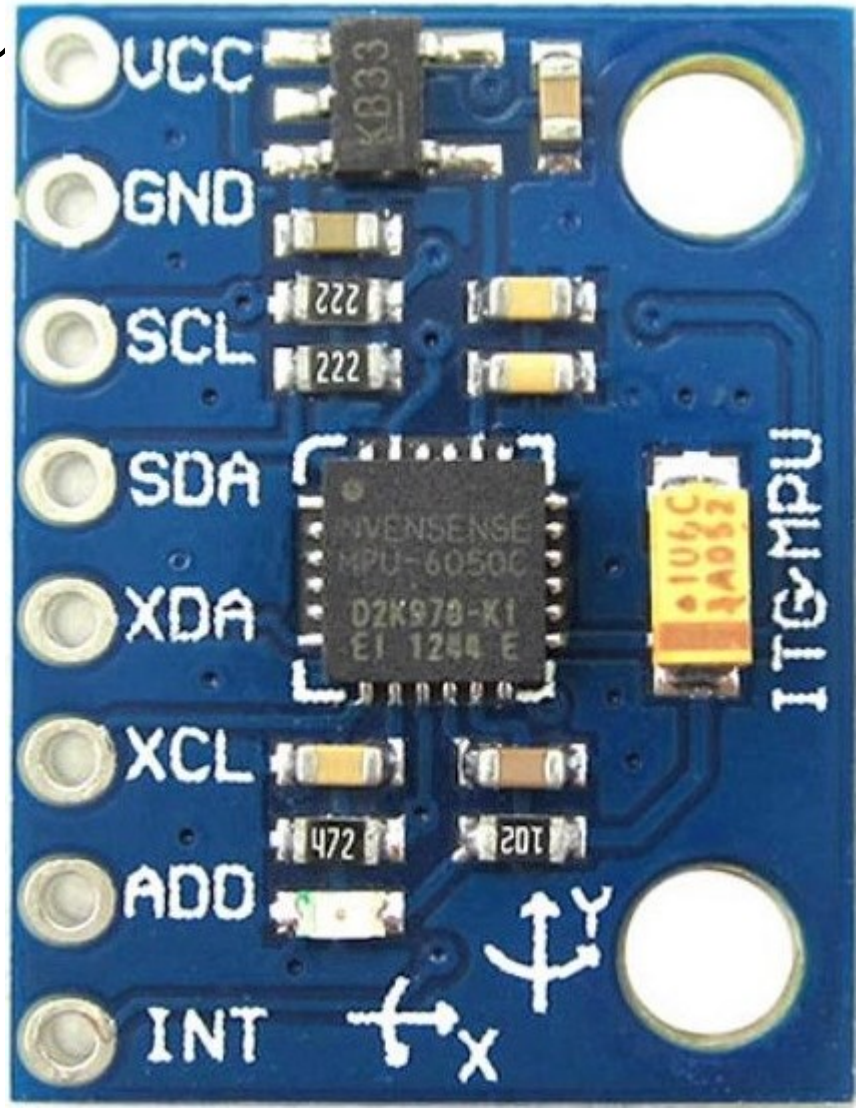
XCL = Aux CL

XDA = Aux DA

AD0 = AD0/SD0

Siehe Pinbelegung und Bedeutung

→ Datenblatt **MPU-6050**

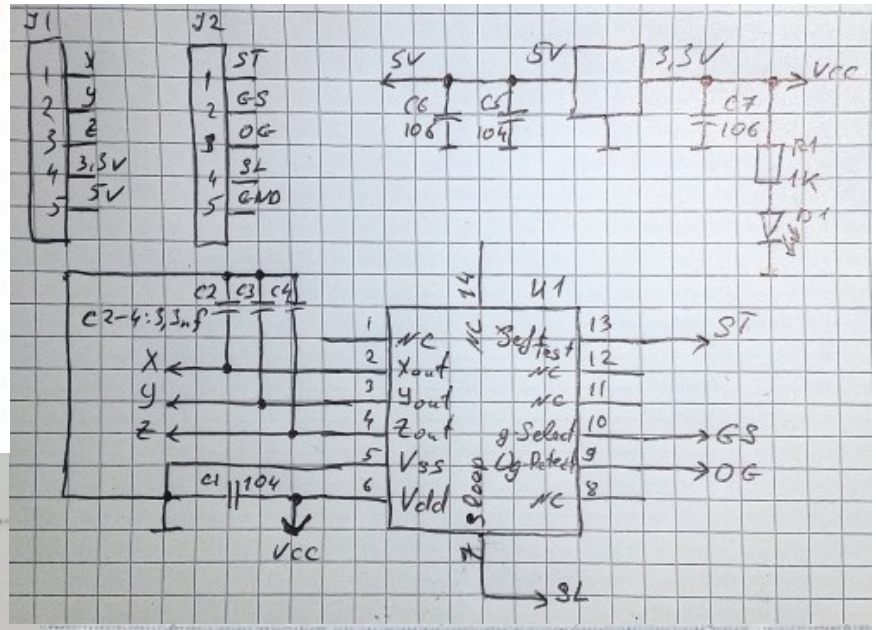


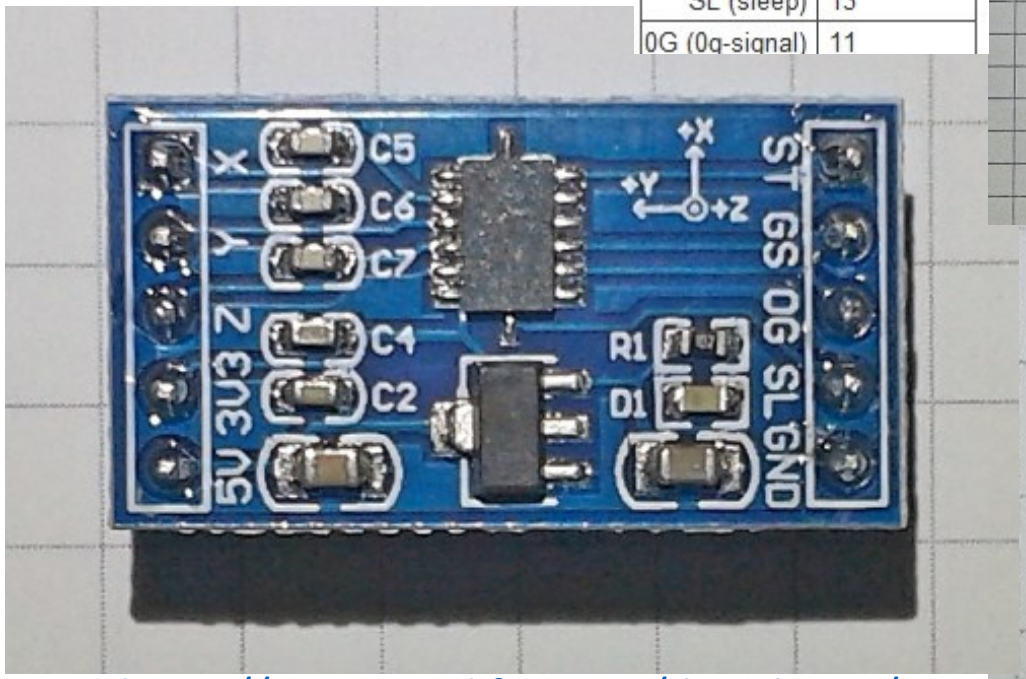
Beschleunigungssensor MMA7361

Dieser gibt einfach 3 Analogspannungen aus.
Handbuch lesen !!
Spannungen beachten
PEGELANPASSUNG

Billig in China zu bekommen

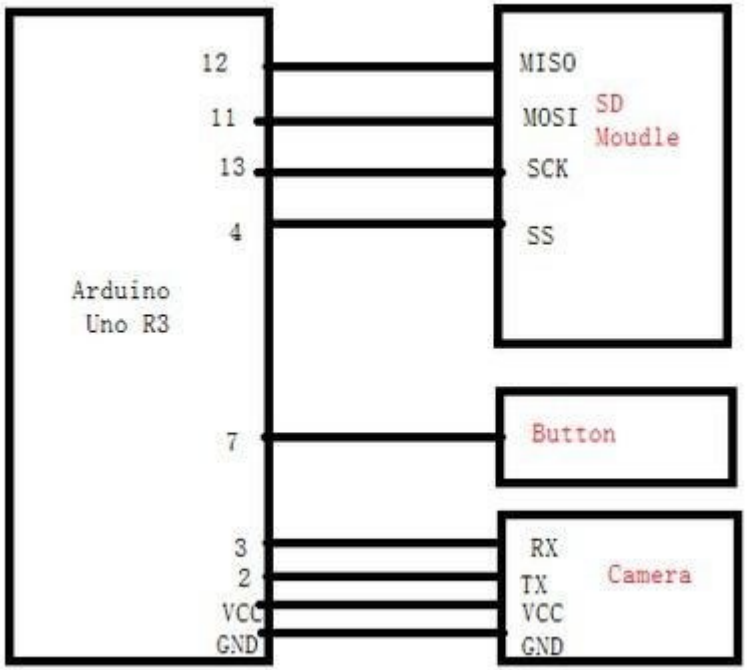
MMA7361	Arduino PIN
5V	NC
3,3V	3,3V
GND	GND
GS (g-Select)	10
ST (self.test)	12
x	A0
y	A1
z	A2
SL (sleep)	13
0G (0g-signal)	11





https://www.sparkfun.com/datasheets/Components/General/MMA7361L.pdf.

OV7670 300KP VGA Kamera Modul für Arduino

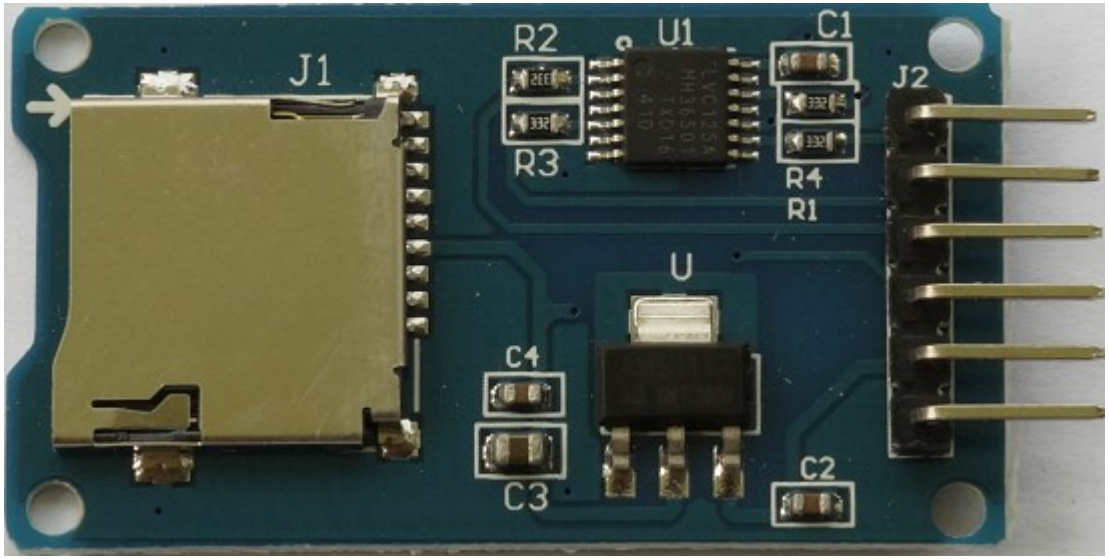

Arduino
Uno R3
12
11
13
4
7
3
2
VCC
GND
MISO
MOSI
SCK
SS
SD
Moudle
Button
RX
TX
VCC
GND
Camera

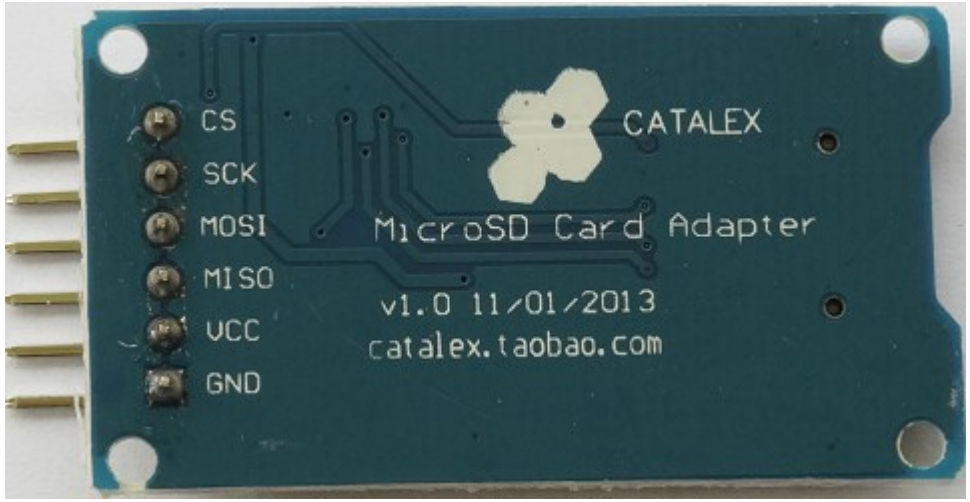

OV7670
2
1
16
15


UC-246
D0
D2
D4
D6
XCK
HS
SDA
GND
D1
D3
D5
D7
PCK
VS
SCL
3V3

Micro-SD-Card-Reader-f-Arduino-ESP8266-SD-Karte-Memory-Shield-Module-6p-SPI

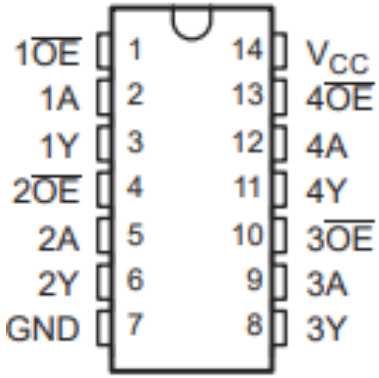
catalex.taobao.com





SD Card →
benötigt Filesystem
ISP Synchron Bus

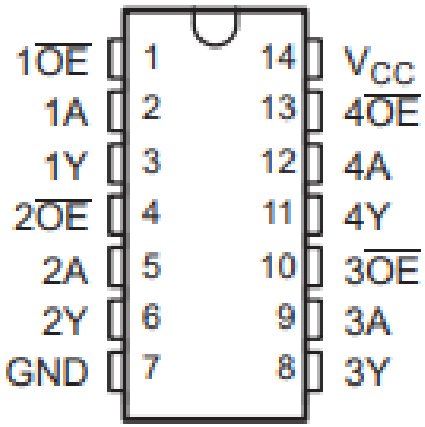
CHIP:
LVC125A
MY20706
TXD16
480

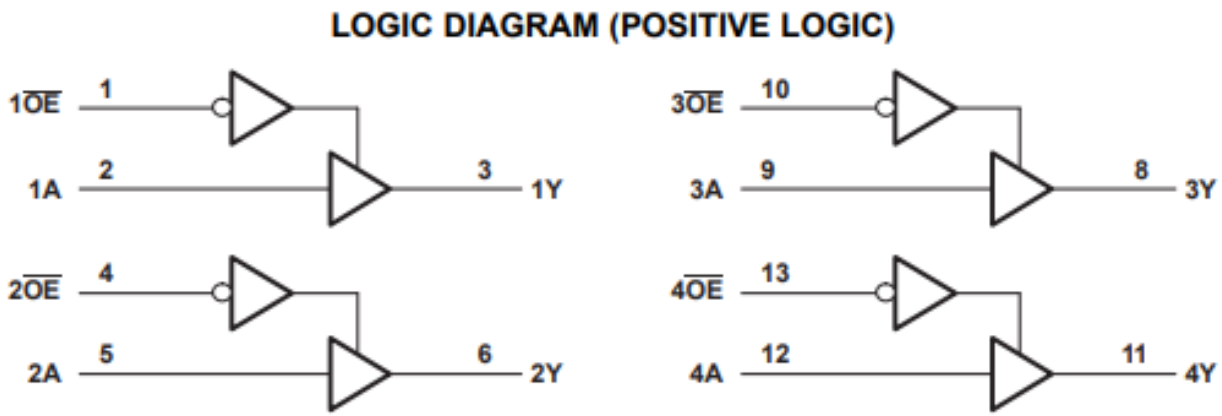


QUADRUPLE BUS BUFFER GATE WITH 3-STATE OUTPUTS
Also ein Dummerchen.
Alles zufuß zu programmieren

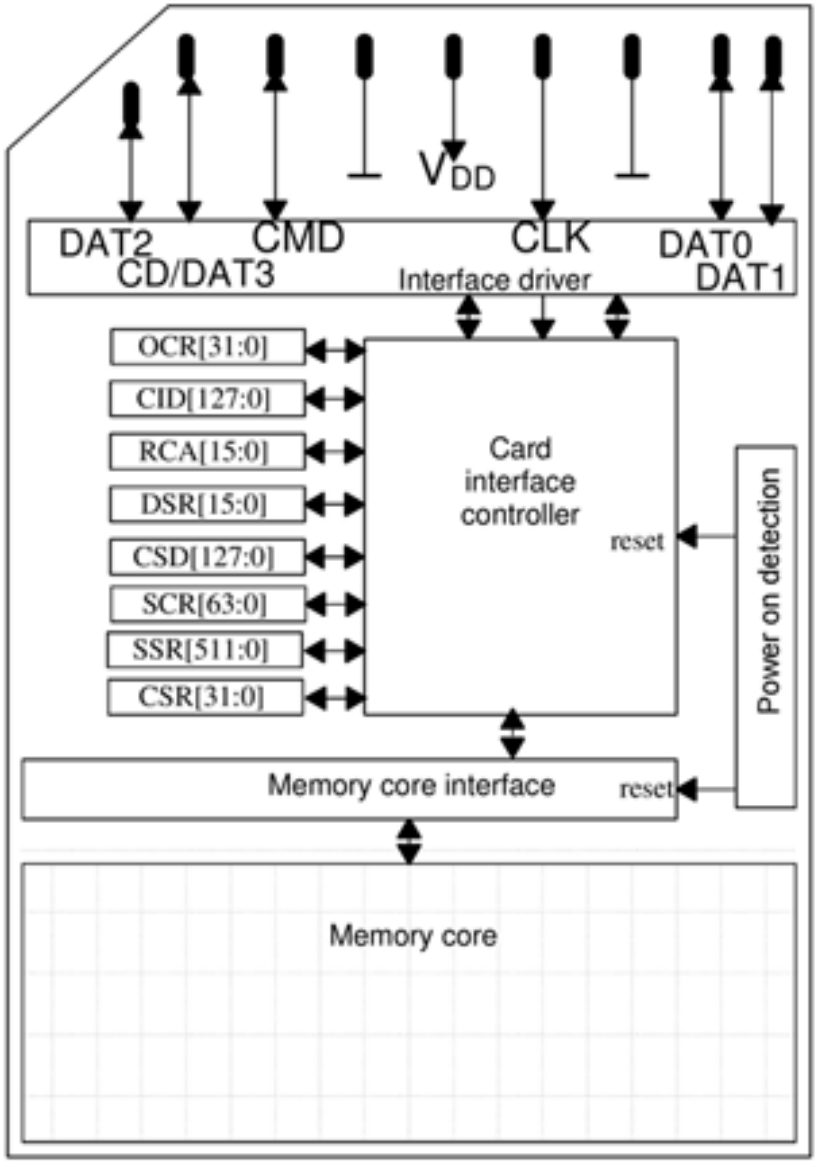
https://www.ebay.de/itm/Micro-SD-Card-Reader-f-Arduino-ESP8266-SD-Karte-Memory-Shield-Module-6p-SPI/162555979945?ssPageName=STRK%3AMEBIDX%3AIT&_trksid=p2057872.m2749.l2649

QUADRUPLE BUS BUFFER GATE WITH 3-STATE OUTPUTS

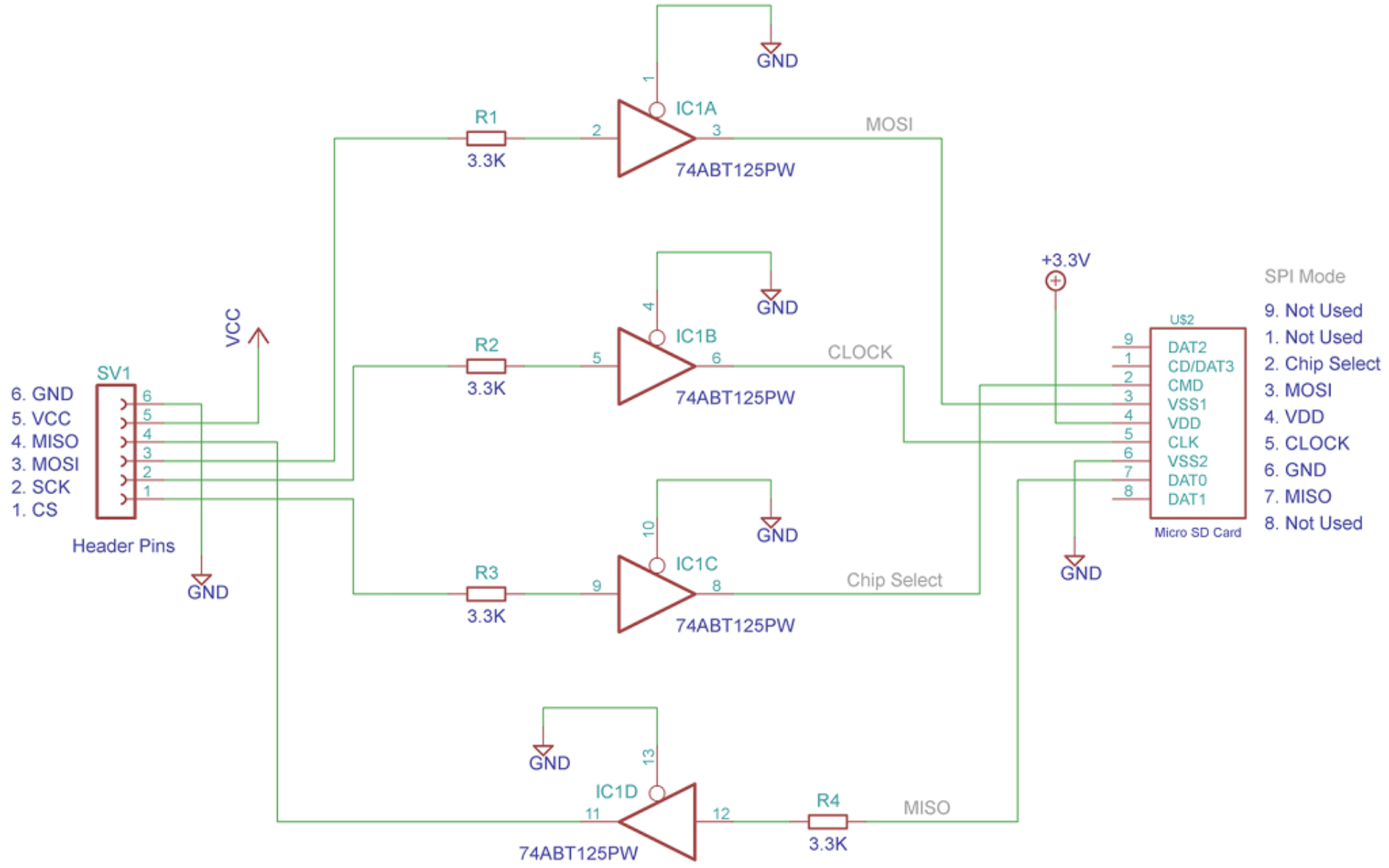




Catalex Micro SD Card Reader Schematic



SD Card Architecture


GND
IC1A
R1
3.3K
74ABT125PW
MOSI
GND
IC1B
R2
3.3K
74ABT125PW
CLOCK
GND
IC1C
R3
3.3K
74ABT125PW
Chip Select
GND
IC1D
R4
3.3K
74ABT125PW
MISO
VCC
SV1
6. GND
5. VCC
4. MISO
3. MOSI
2. SCK
1. CS
Header Pins
GND
+3.3V
U$2
DAT2
CD/DAT3
CMD
VSS1
VDD
CLK
VSS2
DAT0
DAT1
Micro SD Card
GND
SPI Mode
9. Not Used
1. Not Used
2. Chip Select
3. MOSI
4. VDD
5. CLOCK
6. GND
7. MISO
8. Not Used

Filesystem für den SD Card Reader :
https://github.com/greiman/SdFat

https://www.vishnumaiea.in/projects/hardware/interfacing-catalex-micro-sd-card-module

Beim Testen ist mir aufgefallen, dass die Initialisierung des Sensors recht lange gedauert hat (gefühlte 10 Sekunden). Der Sensor hat eine einstellbare Empfindlichkeit (±1.5g, ±6g; 800 mV/g bei 1,5g) und eine Erkennung des freien Fall (0g-detect).

http://s6z.de/cms/index.php/arduino/sensoren/22-beschleunigungssensor-mma7361

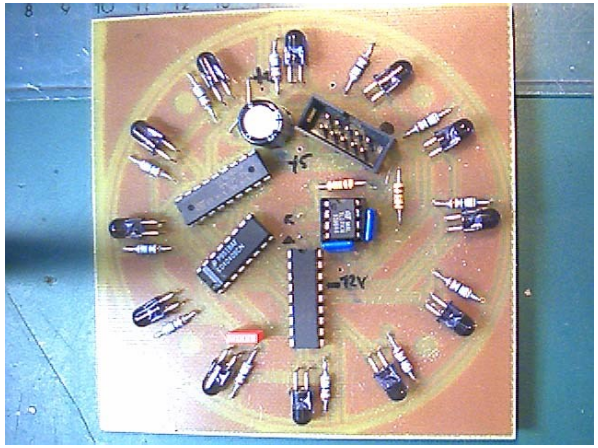
IR Radar

Alternative zum Ultraschall-Verfahren.

Niveau: Aufwendig. Eigen Platine.
Software. Mittelschwer..
Verhalten: unsicher, Kinderkrankheiten

Coolnessfaktor: hoch. Neuartig.
Hat keiner gemacht.
Was zum angeben bei Geldgebern.
Realisation.:
Kein Orts-Mapping mit ATMega 8
Mit mehr Platz oder Telemetrie alles möglich.
Ausbaubar je nach Aufwand.

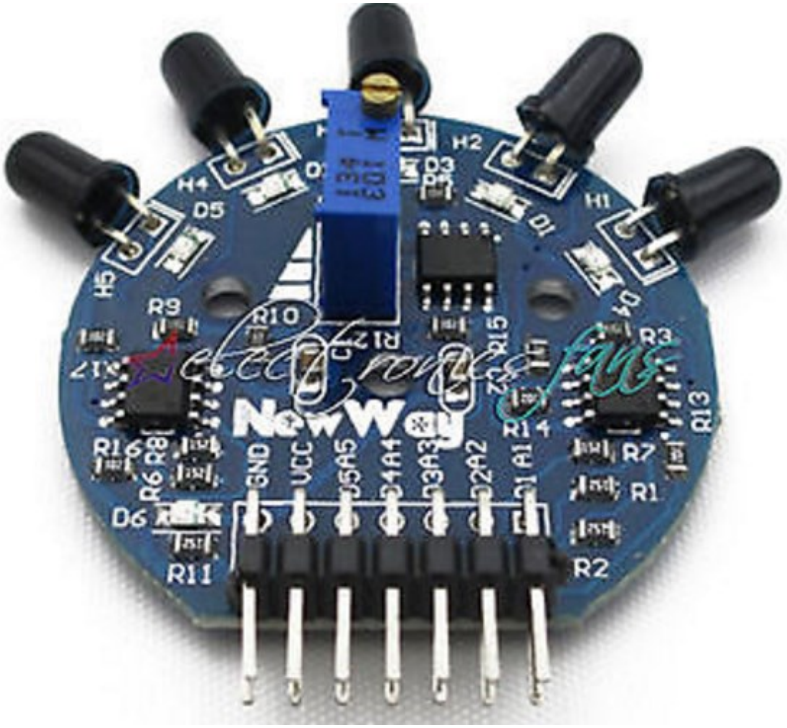
Ergebnis: Unbestimmt,
Experimentierbedarf, aber spannend.

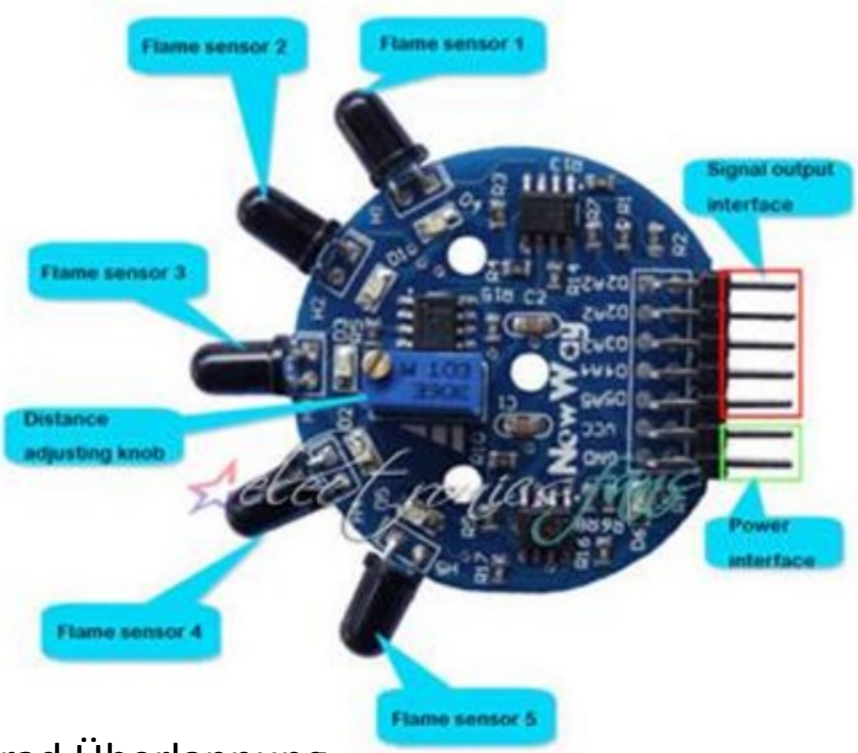


IR-Sender-
Uhrkranz-Sender
mit PWM 36KHz

IR-Empfänger
36KHz

Flame (Flammen) Detector





5 Kanal 120 Grad/Sensor mit 30 Grad Überlappung
3.3..9V
Detektiert 700nM ... 1100nM
Analog und Digital Output

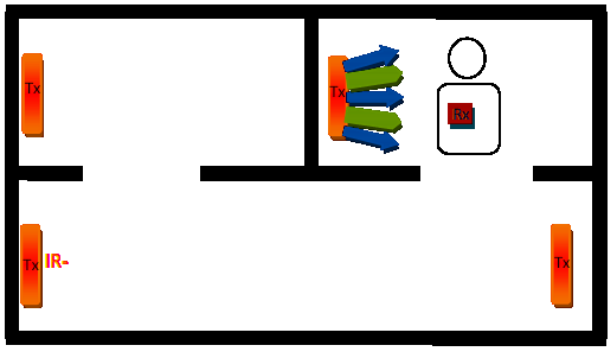
http://www.ebay.de/itm/172322439654?_trksid=p2057872.m2749.l2649&ssPageName=STRK%3AMEBIDX%3AIT

Indoor-Ortungsverfahren nach MIT

http://indoor-ortung.de/systeme/cricket/

Steht und fällt mit der Qualität der Ultraschallortung.
Versuch aufbauen und dann Machbarkeit entscheiden.

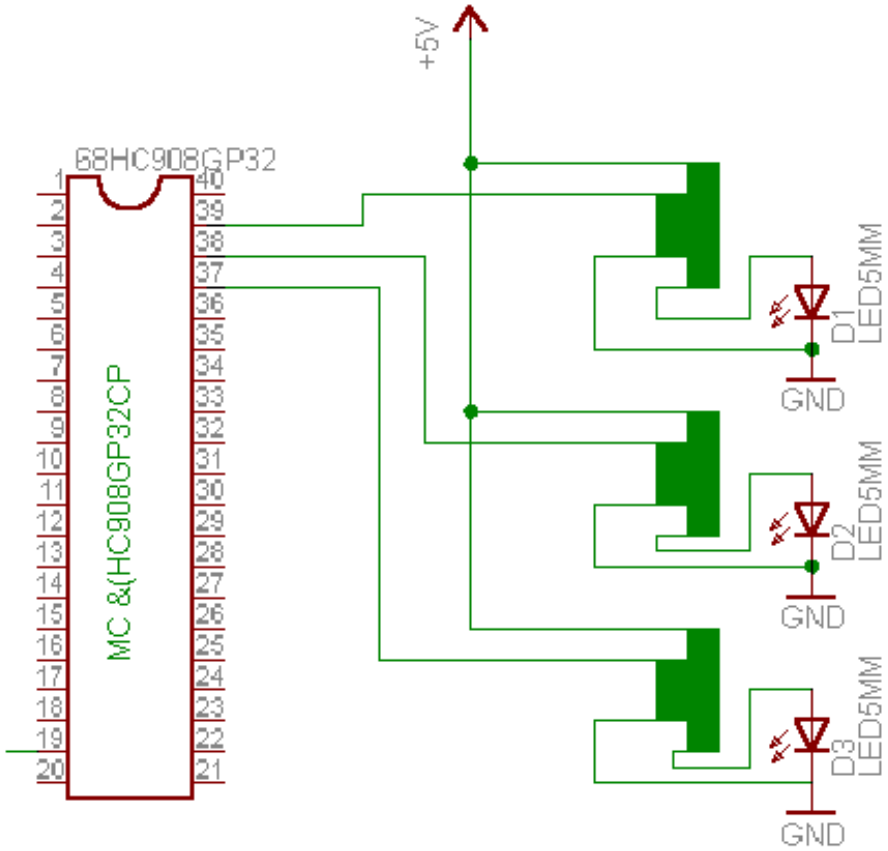
Status: Ultrasonic ist grad im Bau.
Keine Erfahrungswerte.
U-Sonic- Verfahren noch unbestimmt, was sich
bewährt., Timer oder Amplitudenauswertung

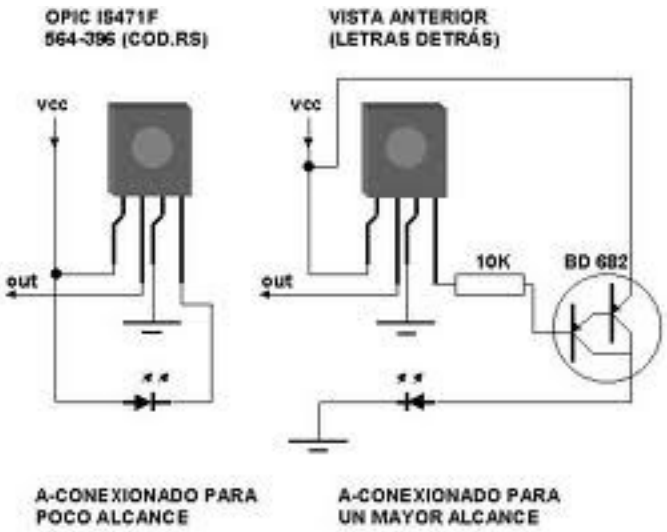


Alternative: Abstandssensor IS471

Der IS471F ist ein sehr einfach zu verwendender Infrarot Abstandssensor. Alles was man zusätzlich zum IS471F braucht ist eine Infrarot LED. Der IS471F reagiert auf Gegenstände die in ca. 7 bis 10 cm Entfernung sind. Das Ausgangssignal des IS471F an dem die LED angeschlossen wird ist bereits moduliert, wodurch die Einflüsse von Fremdlicht minimiert werden. Er treibt einen Strom von ca. 50 mA durch die LED, das ist kein Problem für 20mA LEDs, weil sie nur gepulst werden. Wenn man einen Transistor als Verstärker dahinter schaltet kann man die LED mit einem noch viel höherem Strom betreiben und so die Reichweite erhöhen. Der IS471F arbeitet Digital, das bedeutet er liefert nur eine 0 oder 1 am Ausgang. Die Pegel sind TTL kompatibel. Man kann also nur erkennen ob etwas da ist oder nicht. Es ist also keine Aussage darüber wie weit der Gegenstand entfernt ist möglich. Ich habe den IS471F bei meinen beiden Robotern benutzt und bin sehr zufrieden damit. Man muss aufpassen, das man die LED gegen den IS471F abschirmt, damit keine Streustrahlung der LED auf den Sensor trifft. Die IS471F arbeiten mit unterschiedlichen en um das IR Licht zu modulieren. Manchmal kann es aber vorkommen das die en sehr dicht zusammenliegen oder gleich sind. Dann können sich zwei zu dicht nebeneinander liegende Sensoren sich gegenseitig beeinflussen. Der Anschluss ist recht einfach wie auf dem folgendem Bild zu erkennen.

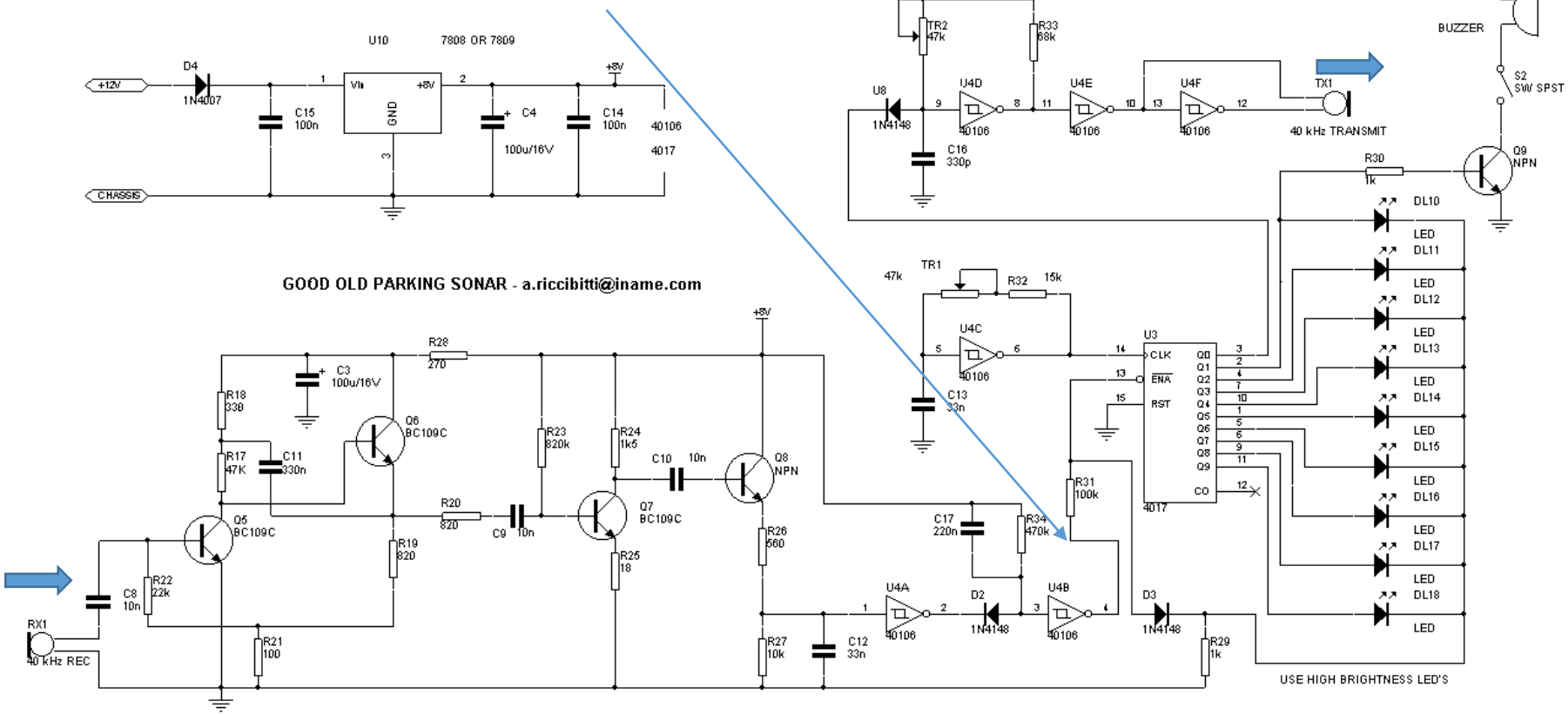
Anschluss des IS471F Infrarot Distanz Sensors





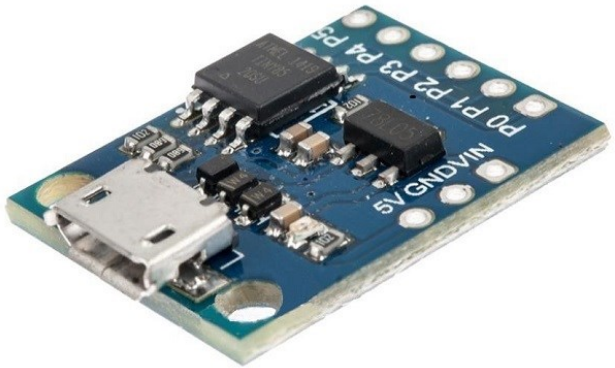
Sonar bzw. Ultraschall

Robotik: Mit Ultraschall lässt sich einfach, wenn auch mit nicht allzu großer Auflösung die unmittelbare Umgebung untersuchen. Der Grundsätzliche Aufbau: Sender-Oszillator und ein Empfänger. Anstelle der LED-Leiste mit 4017 kann nun auch ein **µC** sitzen:



ATTiny85 Minimodul für wenige €

Für wenige €(2..3.) bekommt man schon einen kleinen µController der AT Familie.



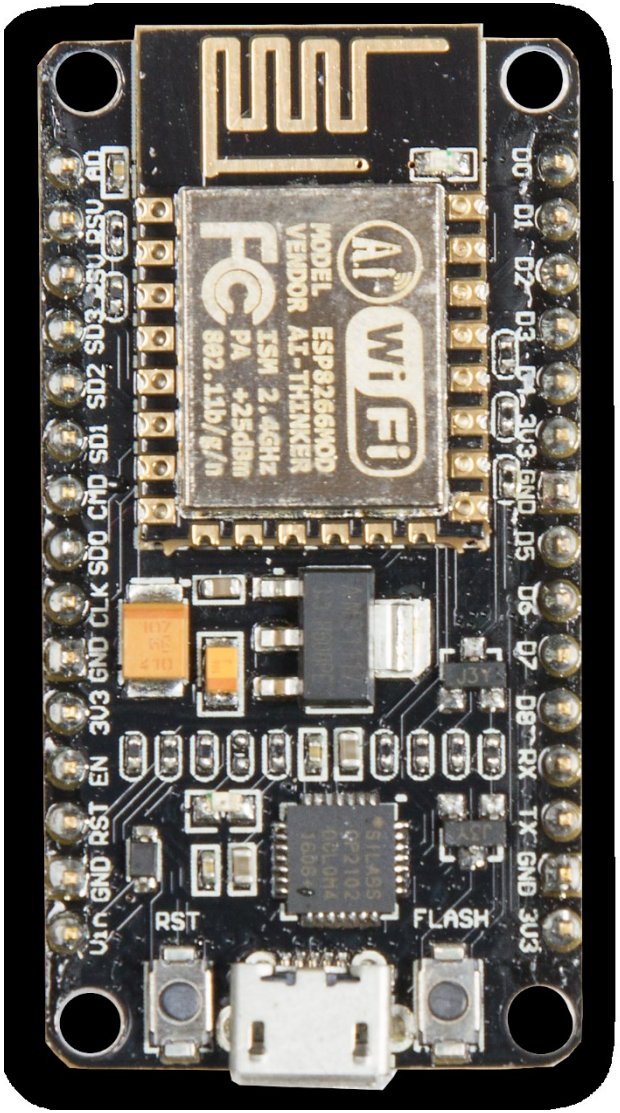
Mini USB Micro Development Board
- Condition: New
- Support for the Arduino IDE 1.0+ (OSX/Win/Linux)
- Power via USB or External Source - 5v or 7-16v)
- On-board 150ma 5V Regulator
- Built-in USB (and serial debugging)
- 6 I/O Pins (2 are used for USB only if your program actively communicates over USB, otherwise you can use all 6 even if you are programming via USB)
- 8k Flash Memory (about 6k after bootloader)
- I2C and SPI (vis USI)
- PWM on 3 pins (more possible with Software PWM)
- ADC on 4 pins
- Power LED and Test/Status LED
- Color as pictures
- PLS NOTE that due to lighting effects, monitor's brightness/ contrass settings ect, there could be some slight differences in the color tone of the pictures and the actual item!

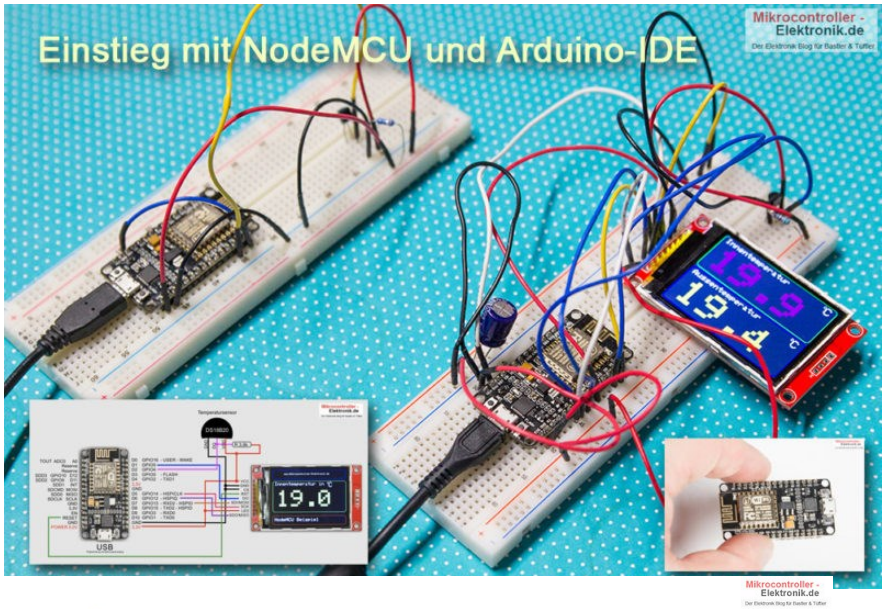
Package Included:1 x Digispark Kickstarter Attiny85 Mini USB Micro Development Board for Arduino(With out header pin)

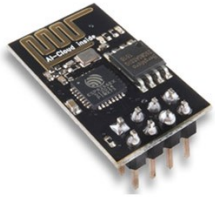
NodeMCU und ESP8266

Sehr gute und umfangreiche Anleitung - WLAN Modul und ARDUINO IDE

http://www.mikrocontroller-elektronik.de/nodemcu-esp8266-tutorial-wlan-board-arduino-ide/

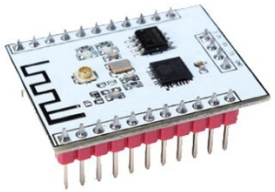






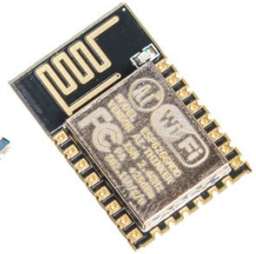
ESP-01

ESP-07



ESP201

ESP-12

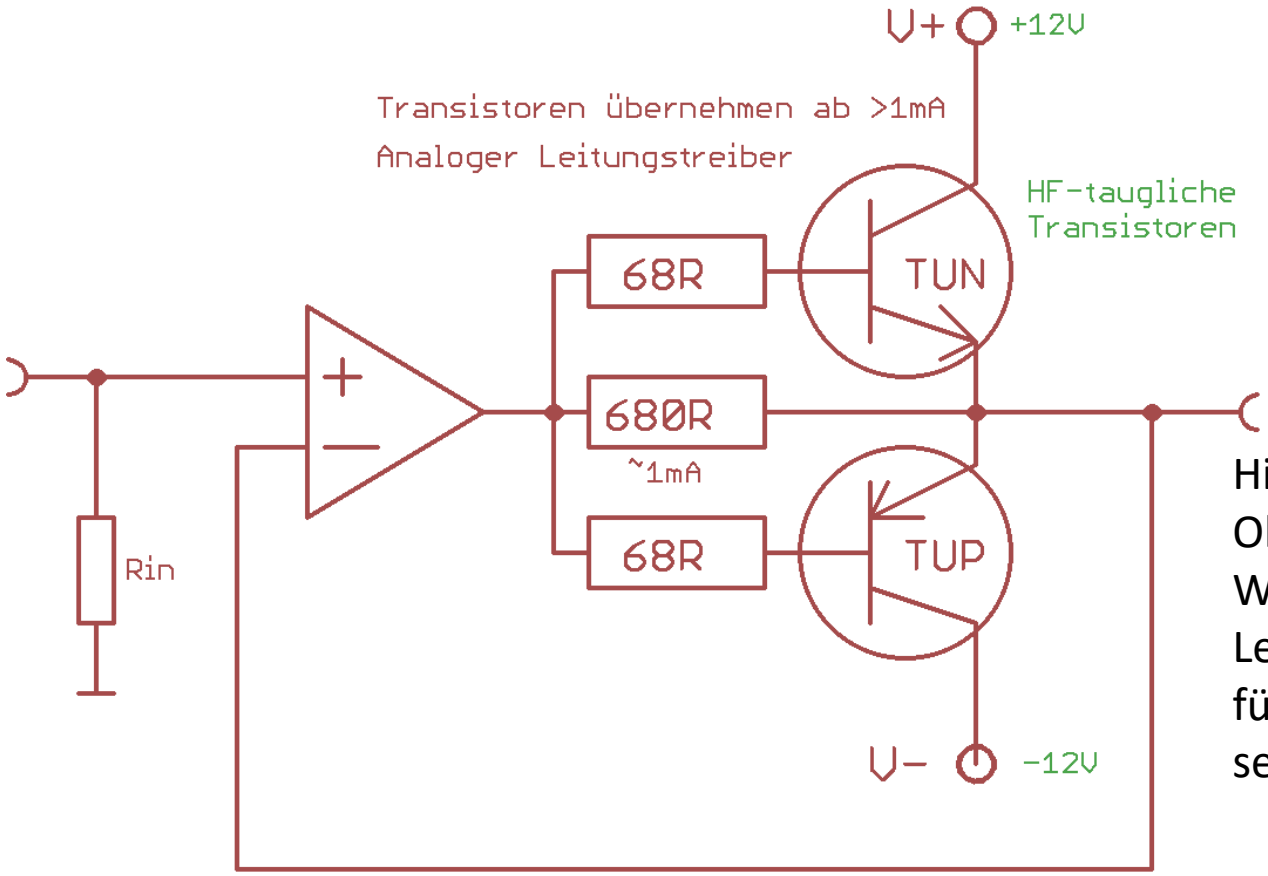


ESP-12E

ESP-12F

Elektronik:
Interessante ICs
und Schaltungskonzepte.

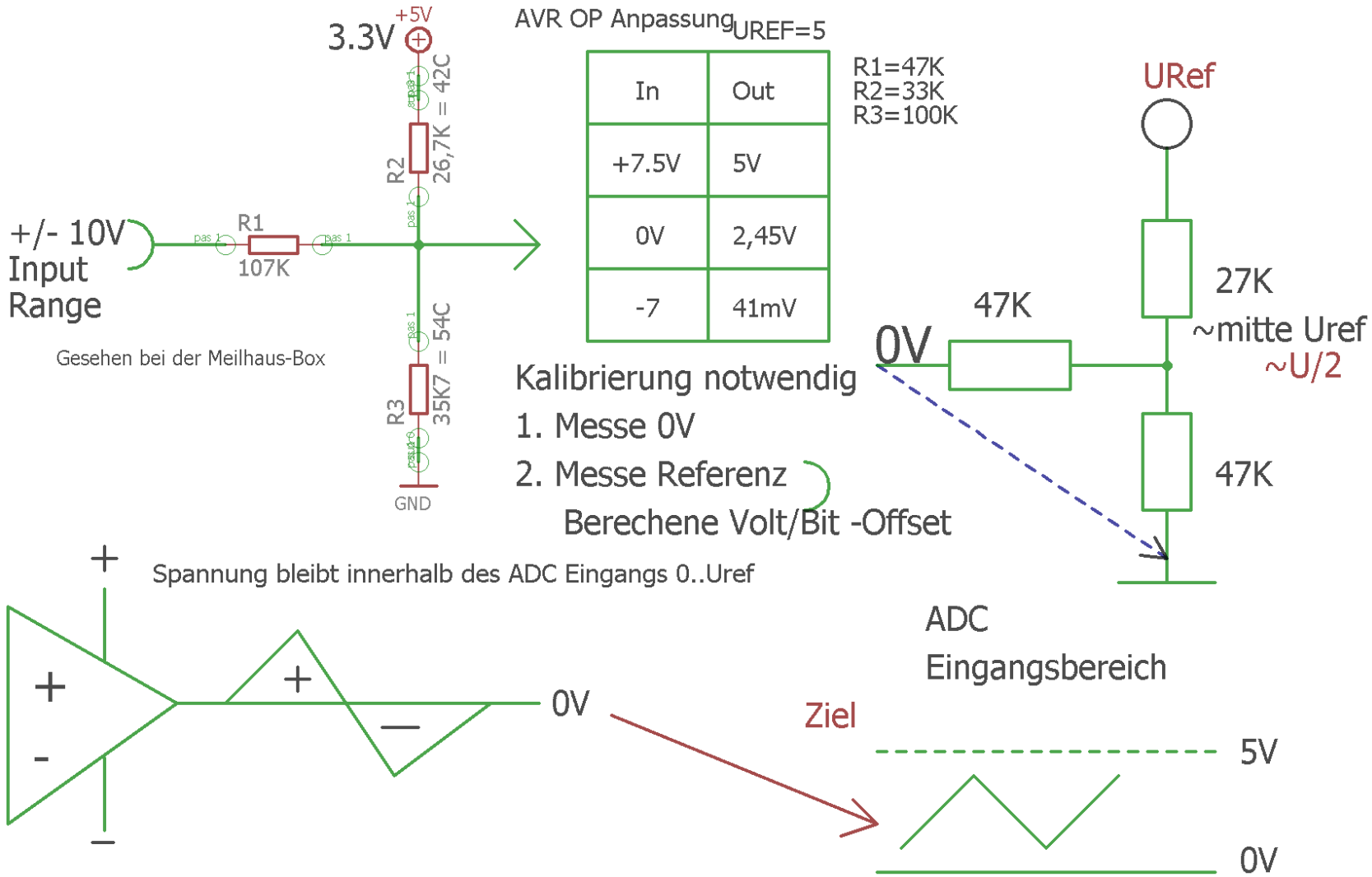
Einfache Stromaufstockung für OP



Hier kann ein 50 Ohm Abschluss-Widerstand als Leitungstreiber für BNC Kabel sein.

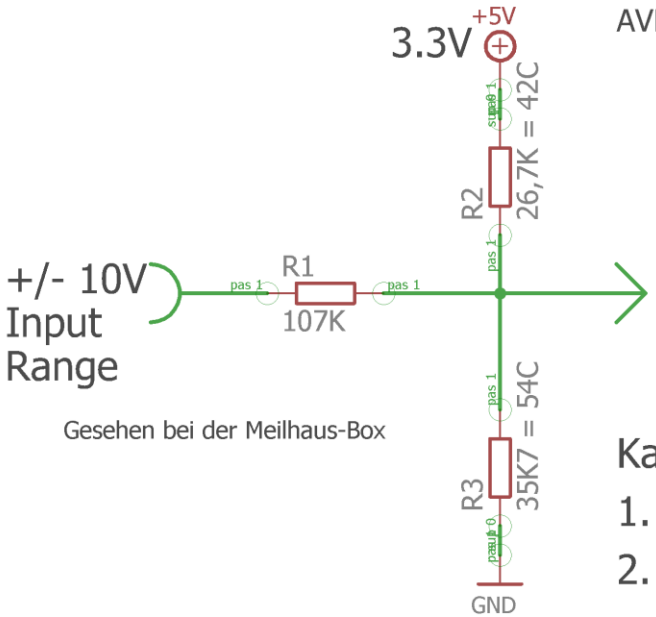
TUN TUO = Transistor Universal NPN oder PNP

So wird ‚einfach' aus einer +/- Spannung „nur eine positive" Spannung für den ADC !



ADC Pegelanpassung

TRICK! Anpassung: Duale = Plus/Minus Spannung an einen ADC mit einfachem Eingang von 0V.. Uref (Bsp. 5V oder 2.54V)



Gesehen bei der Meilhaus-Box

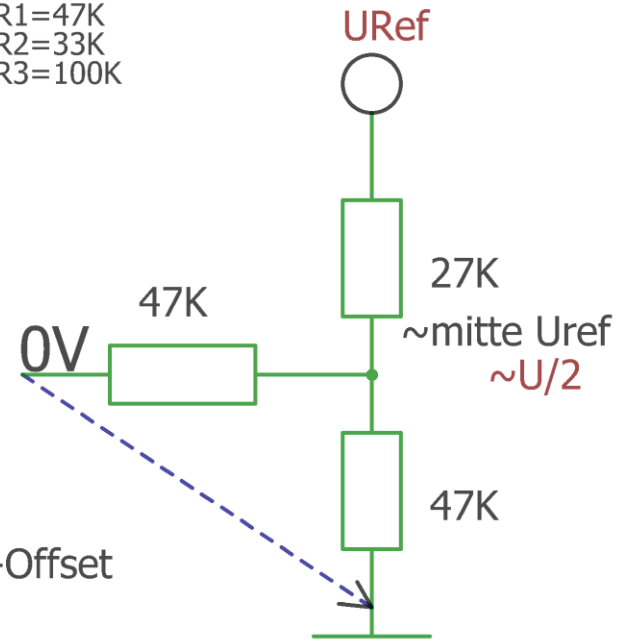
AVR OP Anpassung UREF=5

In	Out
+7.5V	5V
0V	2,45V
-7	41mV

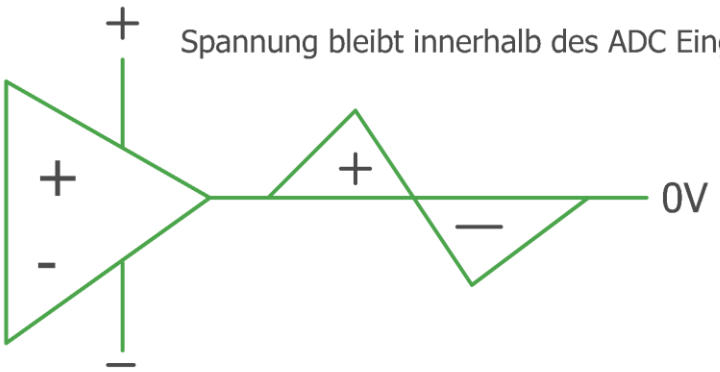
R1=47K
R2=33K
R3=100K

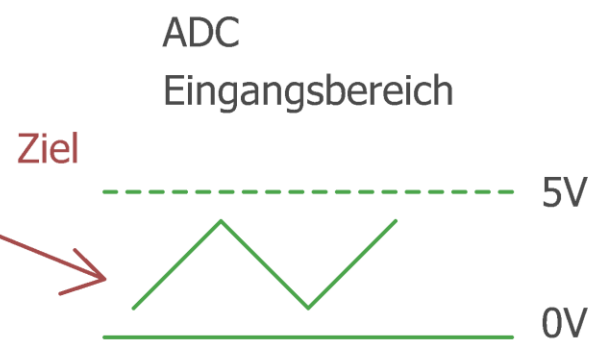
Kalibrierung notwendig

1. Messe 0V
2. Messe Referenz
   Berechene Volt/Bit -Offset



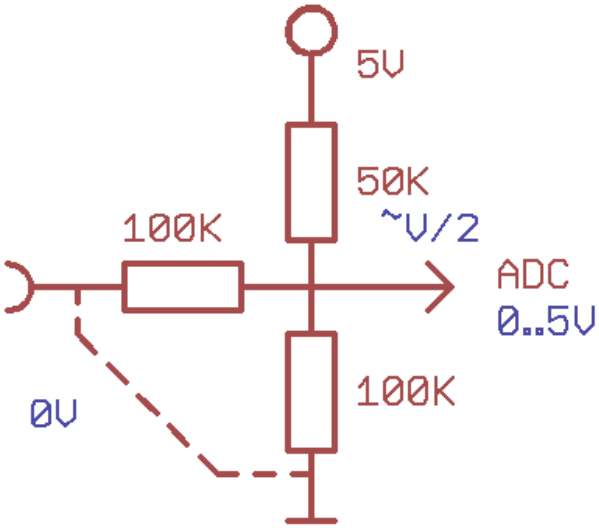
Spannung bleibt innerhalb des ADC Eingangs 0..Uref





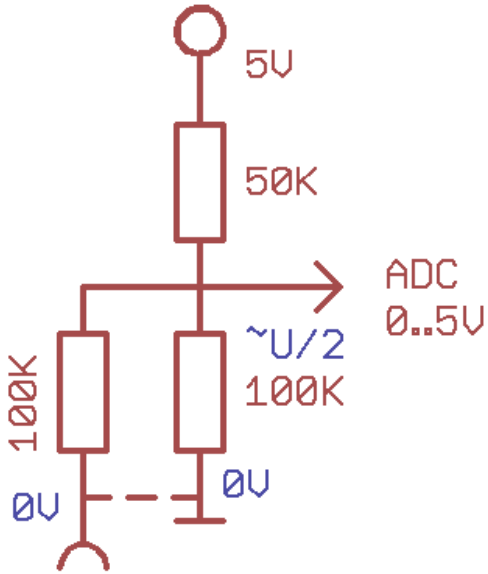
Kalibrierungsprozedur des ADC Pegelanpassers

Kalibrierung

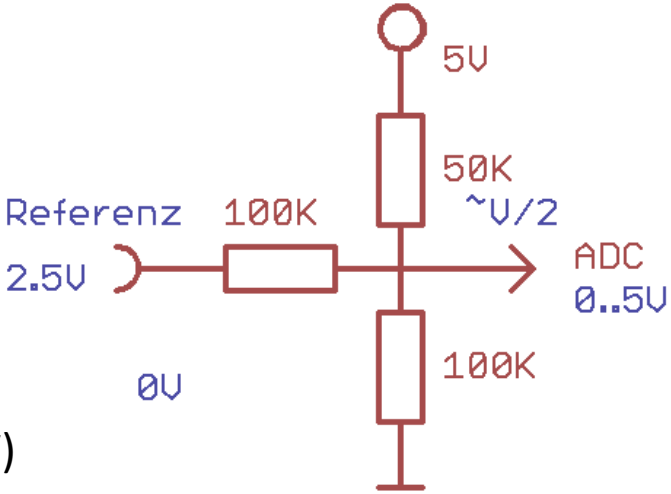


=

Eichen 0V



Eichen mit Referenz



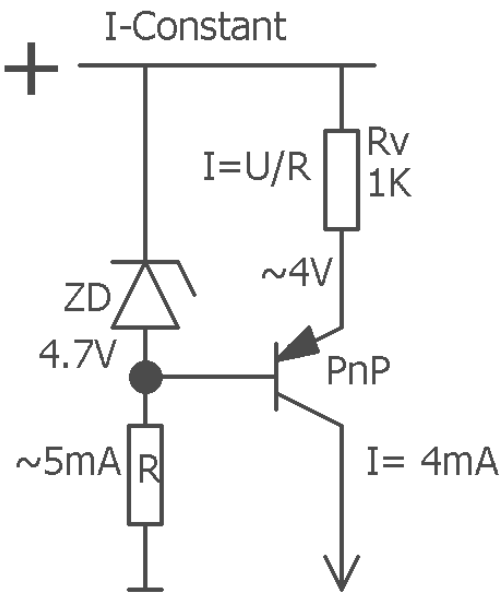
1. Eingang gegen Masse. ADC Wert merken
2. Eingang gegen Referenz. ADC Wert merken.

Rechnen: Volt/Bit = Uref / (ADCval_RefV – ADCval_0V)

ADCval_0V = Offset.

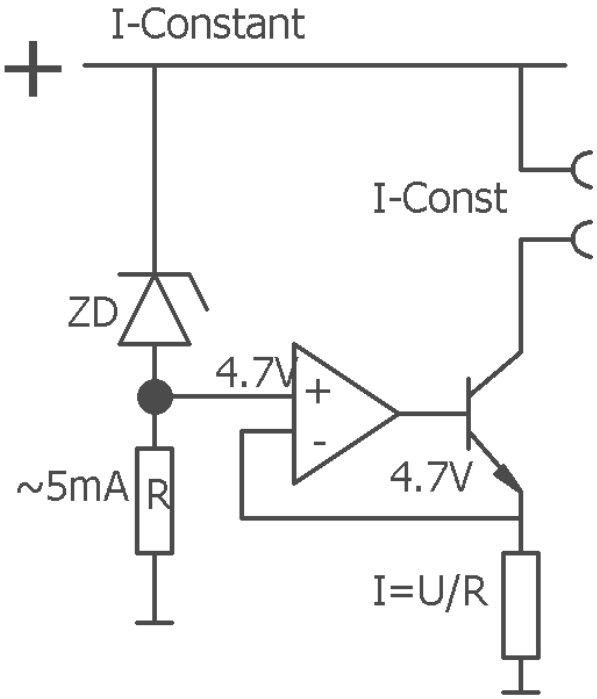
Gemessene Spannung = (ADC_Value - Offset) * (Volt/Bit)

Konstant Strom Schaltungen

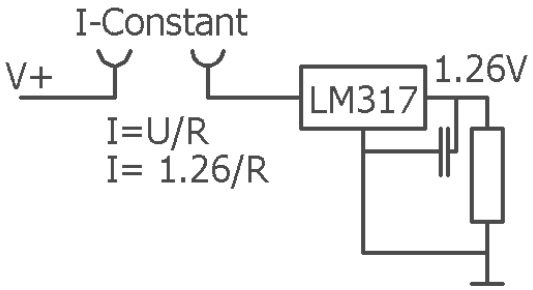


I=U/R
4V/1K=4mA

Konstanten Strom bekommt man durch geregelte konstante Spannung am konstanten Widerstand



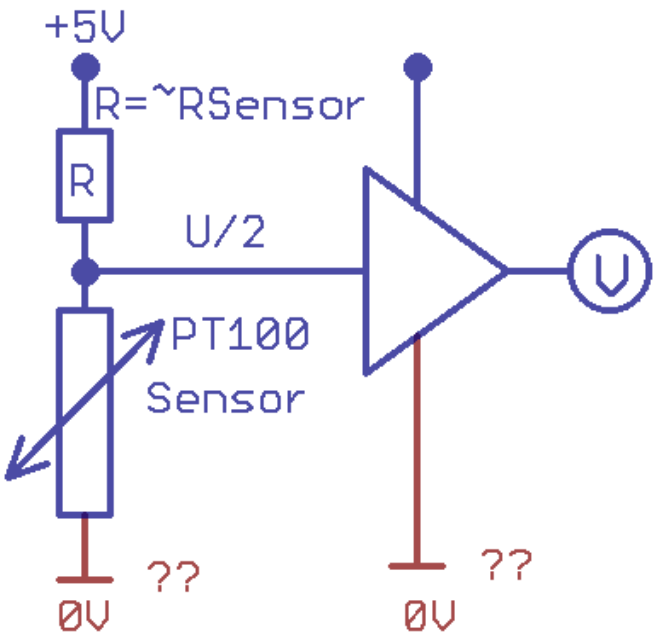
1.26V



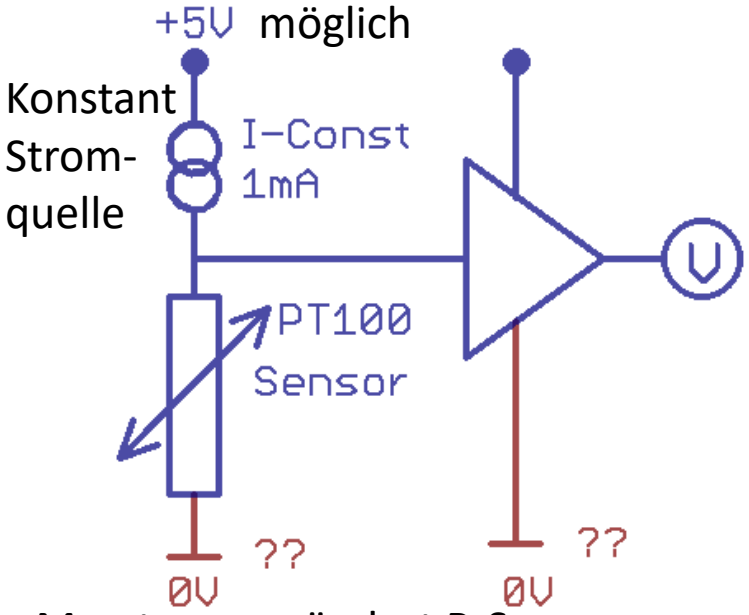
Festspannungsregler

Spannungsteiler mit Sensor versus 1-Punkt Messung mit Konstant-Stromquelle.

Sensor verändert I-Gesamt des Spannungsteilers
Da Igesamt= Ugesamt/(R1+R2)



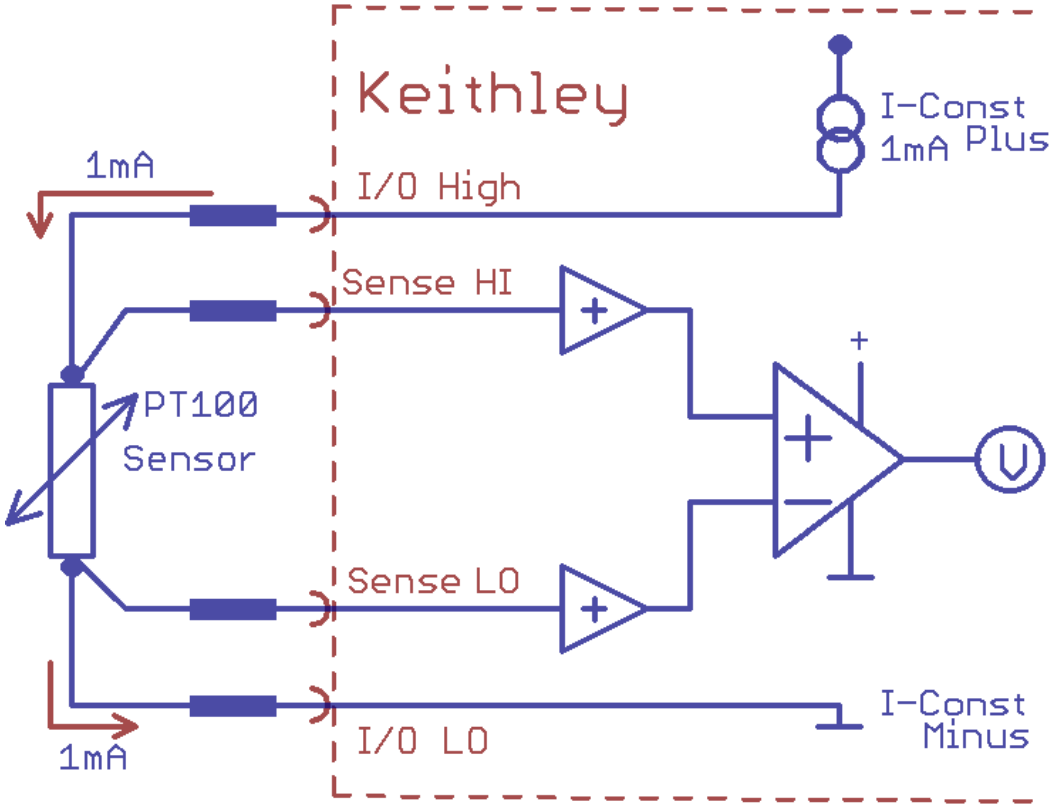
Exakte lineare Abbildung des Mess-R durch U=Iconst * R
→(1mA) ist viel.
Daher Imess so klein wie möglich



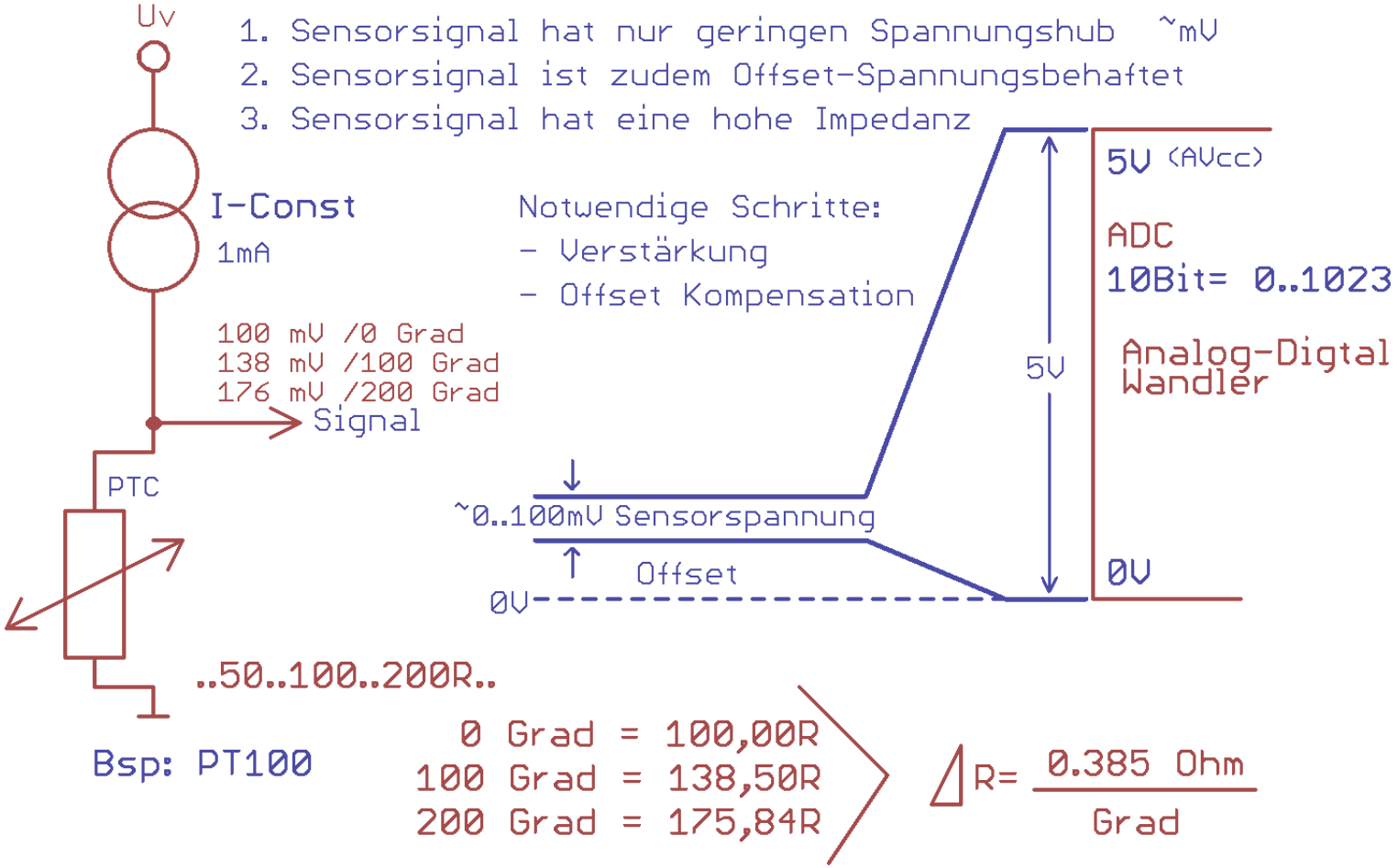
Messtrom verändert R-Sensor.
TRICK: **Messtrom nur im MESSMOMENT einschalten**. Dann sofort wieder aus.

Prinzip 4-Punkt Messung

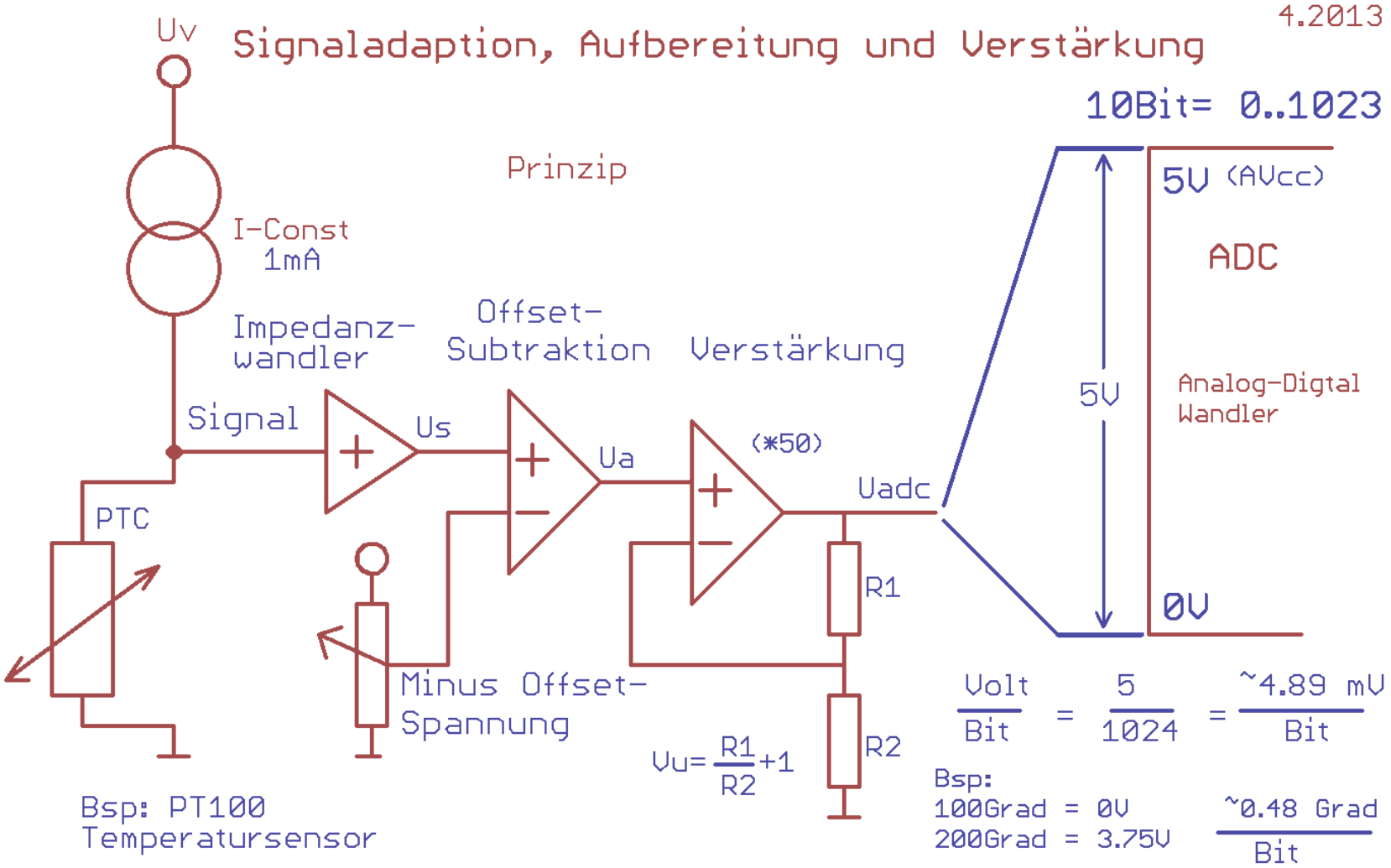
4 Leitungen zum Messpunkt. Dort treffen Die 2 Messleitungen werden so NICHT strombelastet und haben so auch keinen Spannungsabfall.



Sensor Signal verstärken



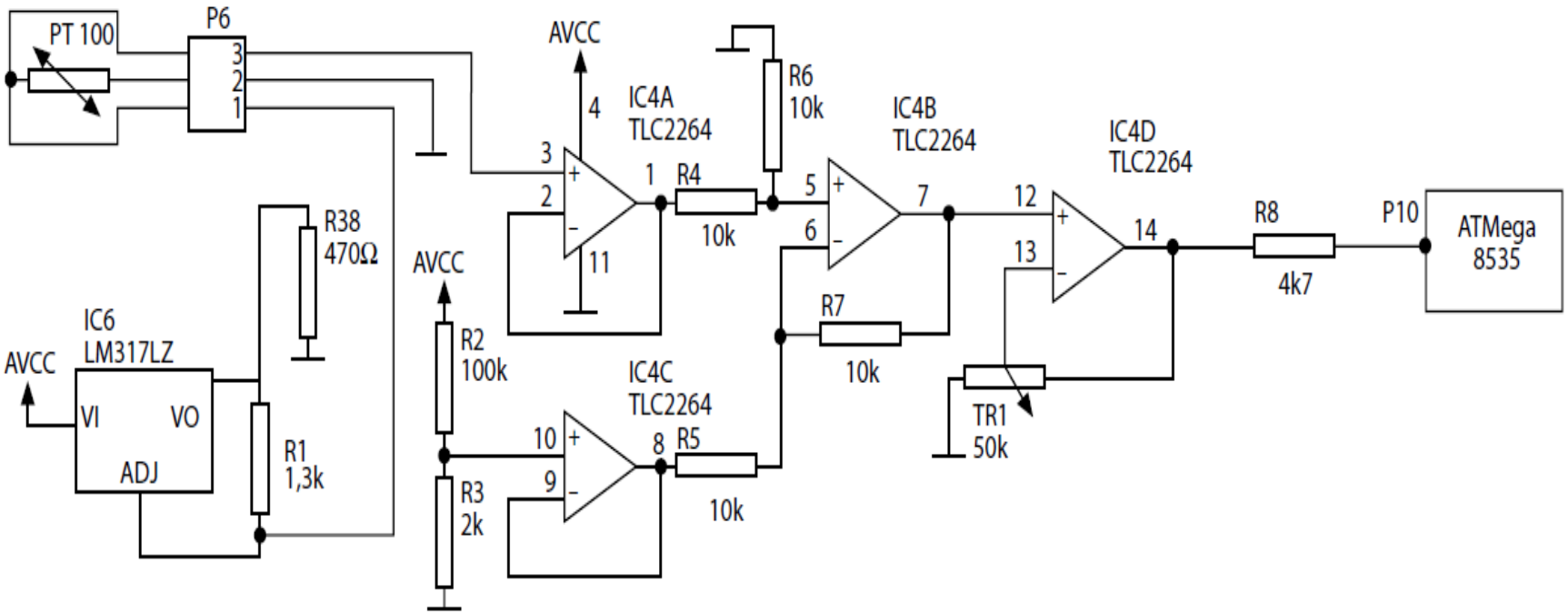
Sensor Signal Aufbereitung für ADC Eingang 0..5V

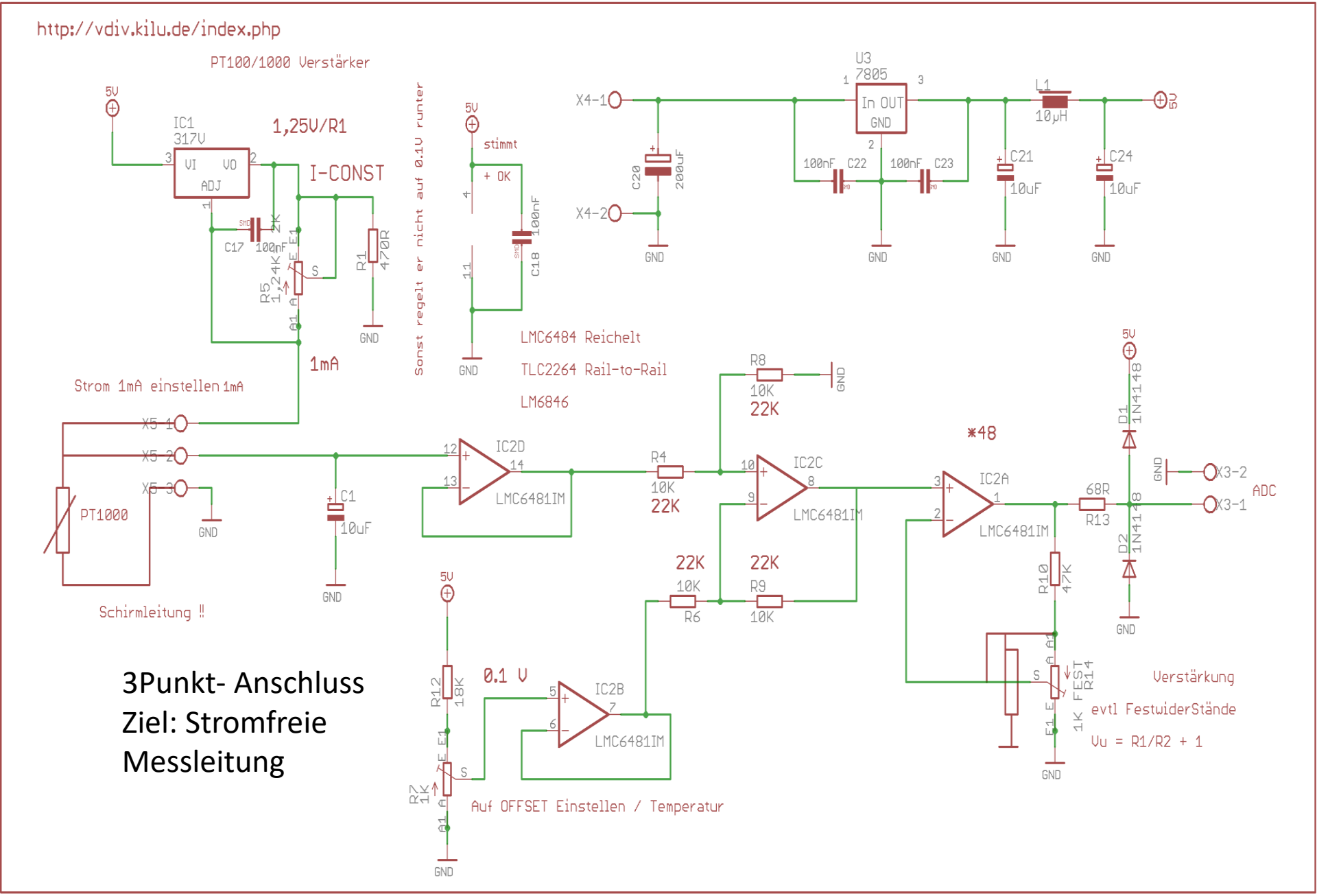


ADC Bereichs-Anpassung

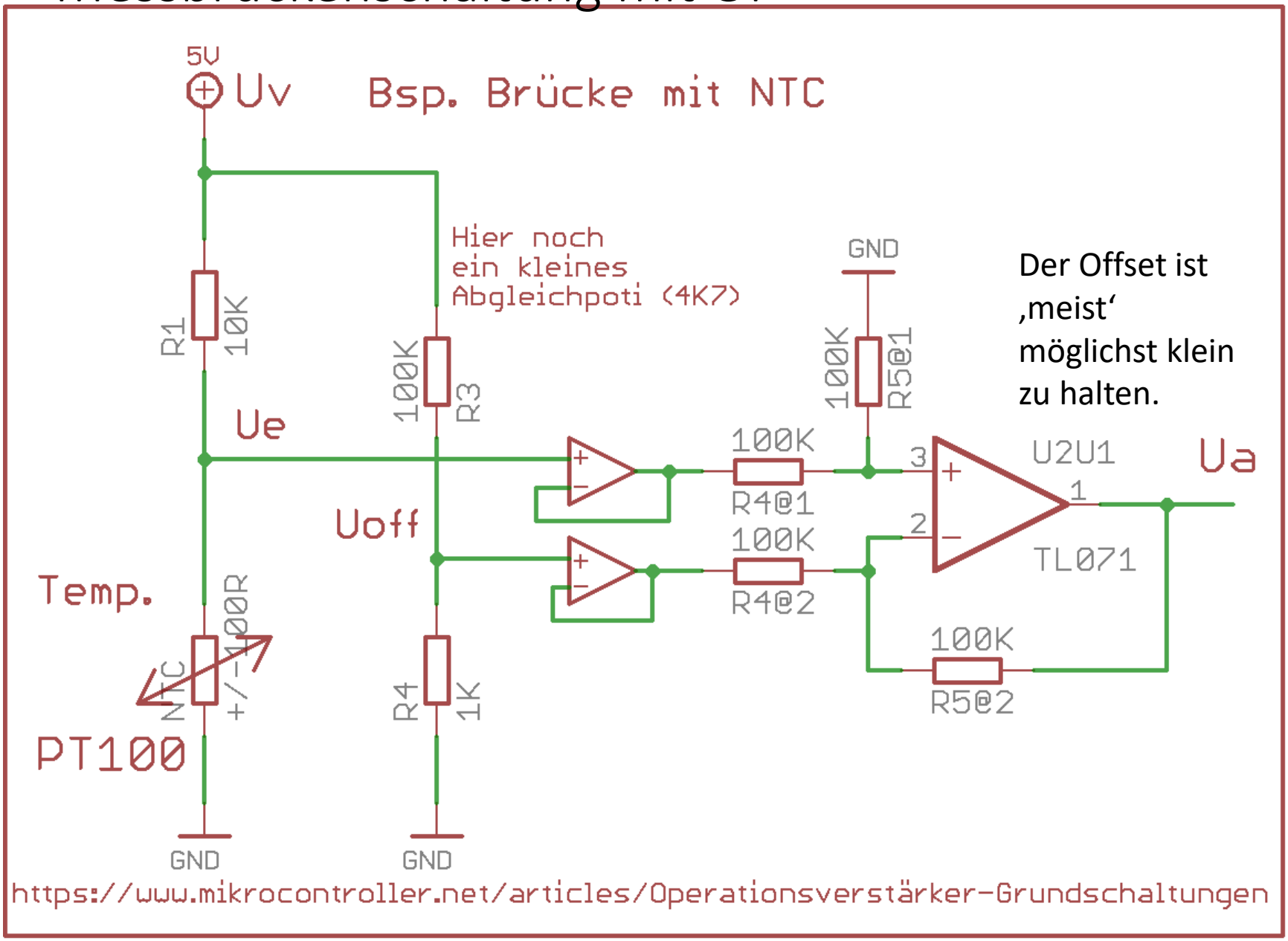
Wünschenswert ist es, den ADC immer mit optimaler Auflösung arbeiten zu lassen.
Das heißt, möglichst viele **Bit/Volt**. Merke: „der Simple Weg ist der gute Weg.“
Es hängt einfach vom Fall ab, und welche Ressourcen man ‚übrig‘ hat´.
10Bit = 2^10 = Reference/1024 = 1024/2.54V = ~2.48mV pro Bit.

Beispiel, ein PT100 hat nur eine kleine Widerstandsänderung. Diese soll jedoch möglichst den gesamten Auflösungsbereich des ADC über die Referenz ausnutzen.




http://vdiv.kilu.de/index.php
PT100/1000 Verstärker
5V
IC1
317V
VI
VO
ADJ
1,25V/R1
I-CONST
C17
100nF
2K
R5
1,24K
R1
470R
GND
1mA
Strom 1mA einstellen 1mA
X5-1
X5-2
X5-3
PT1000
Schirmleitung !!
C1
10uF
Sonst regelt er nicht auf 0.1V runter
stimmt
+ OK
C18
100nF
LMC6484 Reichelt
TLC2264 Rail-to-Rail
LM6846
IC2D
LMC6481IM
R4
10K
22K
R8
10K
22K
IC2C
LMC6481IM
22K
10K
R6
22K
R9
10K
IC2B
LMC6481IM
0.1 V
R12
18K
R7
1K
Auf OFFSET Einstellen / Temperatur
X4-1
X4-2
C20
200uF
U3
7805
In OUT
GND
100nF
C22
100nF
C23
C21
10uF
C24
10uF
L1
10µH
*48
IC2A
LMC6481IM
R10
47K
68R
R13
D1
1N4148
D2
1N4148
X3-2
X3-1
ADC
R14
FEST
1K
Verstärkung
evtl FestwiderStände
Vu = R1/R2 + 1
3Punkt- Anschluss
Ziel: Stromfreie
Messleitung

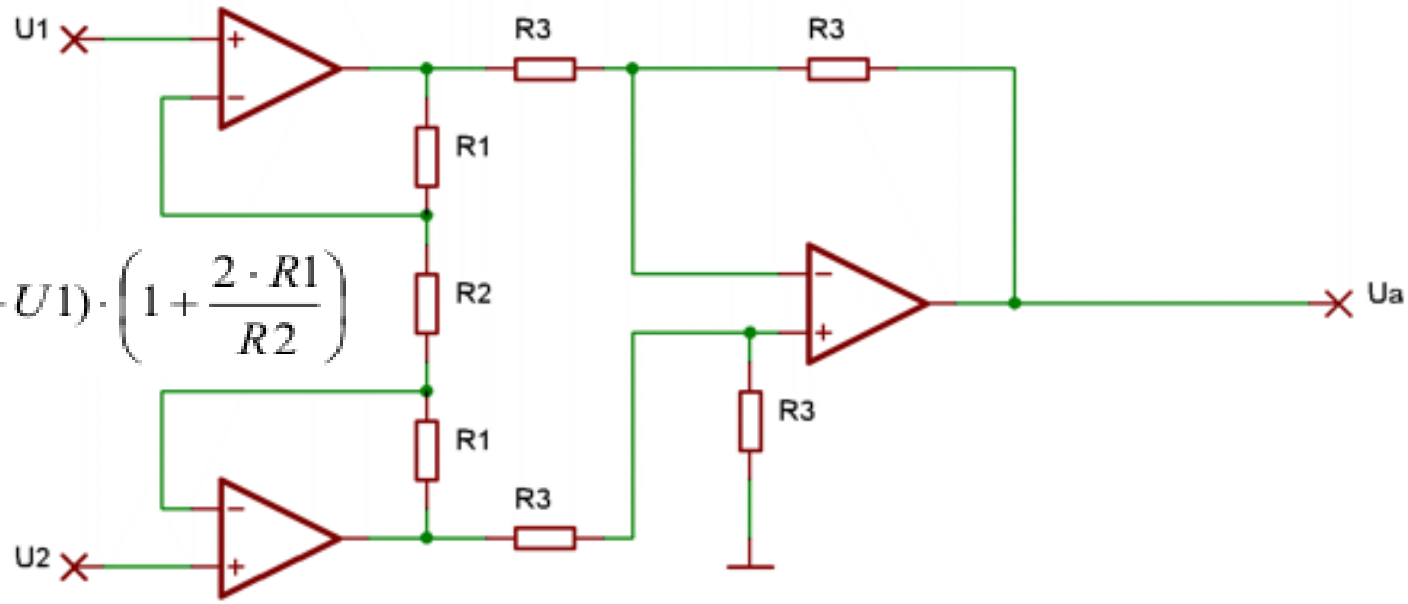
Messbrückenschaltung mit OP



Subtrahierer mit Verstärkung in der Vorstufe

Diese Schaltung ist gut geeignet, um Potentialdifferenzen ohne Belastung der Spannungsquelle zu verstärken. Ein noch besseres Verhalten erreicht man durch das Verstärken der Eingangssignale in den OPs direkt an den Eingängen durch Spannungsverstärker.

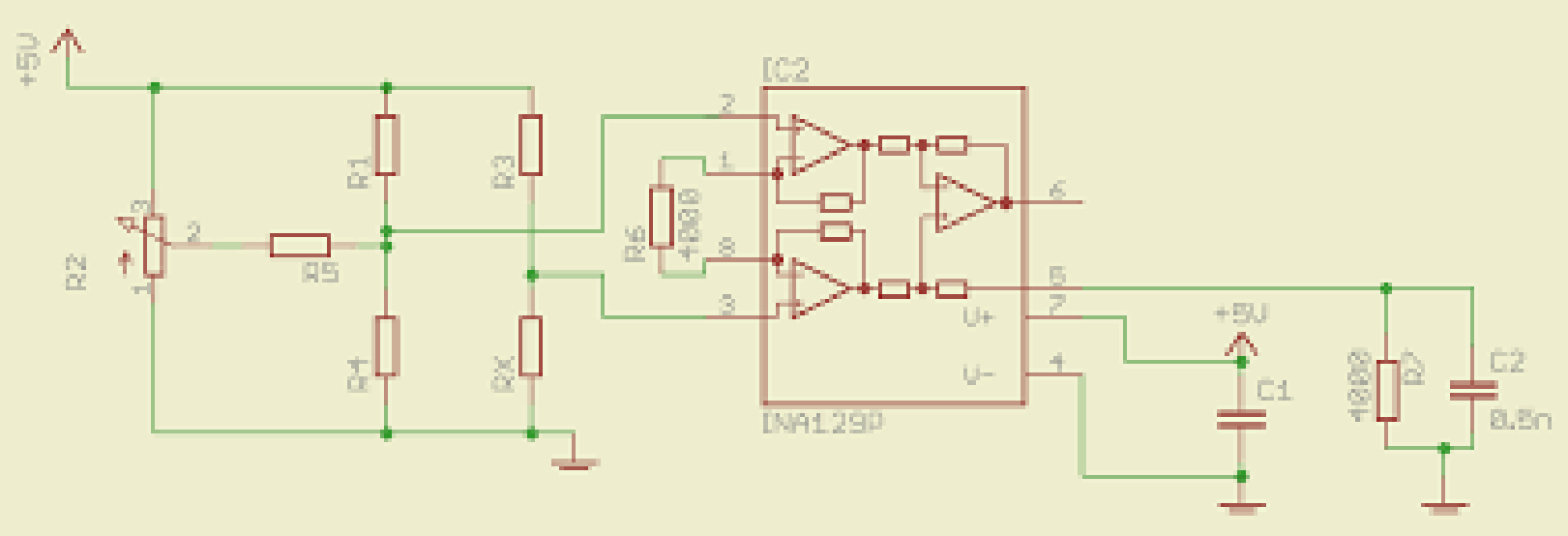
$$Ua = (U2 - U1) \cdot \left(1 + \frac{2 \cdot R1}{R2}\right)$$



Diese Schaltung wird Instrumenten Verstärker genannt. Da am Differenzverstärker alle Widerstände den selben Wert besitzen, ist dessen Verstärkungsfaktor 1. Die Verstärkung wird ausschließlich vom Verhältnis der Widerstände R1 zu R2 bestimmt. Da sich R2 nur ein mal in der Schaltung befindet, kann anstelle eines Widerstandes ein Potentiometer verwendet, und die Schaltung damit genau kalibriert werden.

http://www.loetstelle.net/grundlagen/operationsverstaerker/opamp_4.php

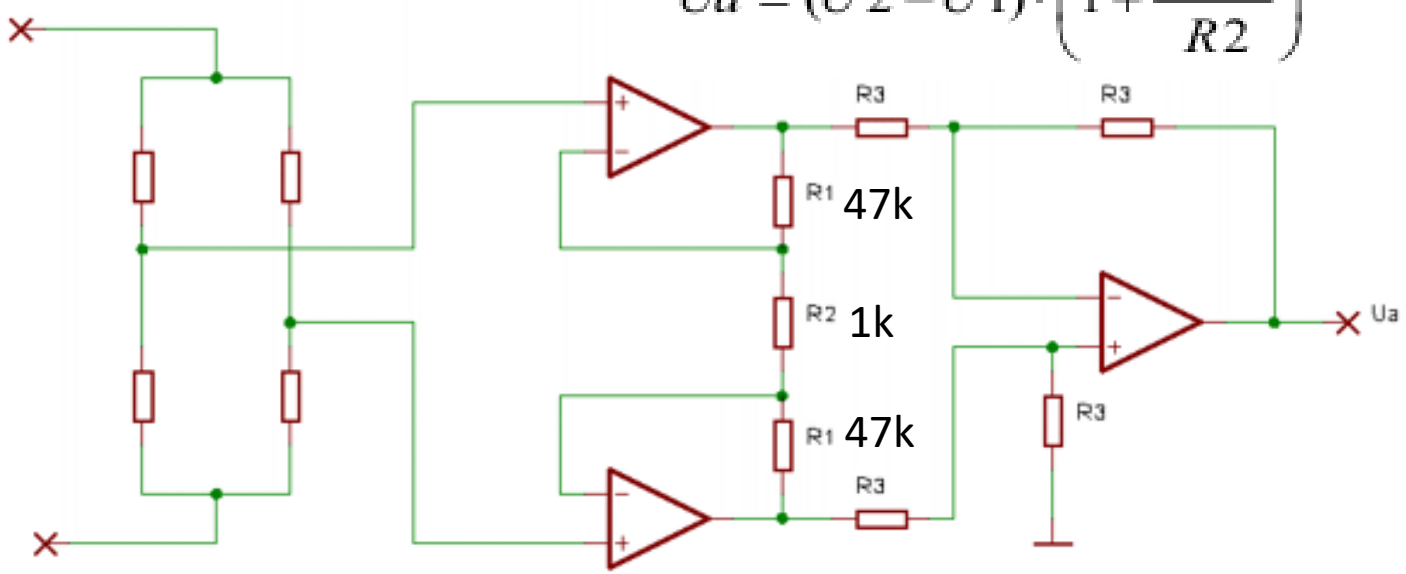
Fertige OP Schaltung INA129P



Differenzverstärker in Brückenschaltung.

Es soll die Verstimmung einer Brückenschaltung bestimmt werden. Da Diese nicht belastet werden darf, muss ein Instrumenten Verstärker verwendet werden. Die Potentialdifferenz beträgt maximal 0,1V soll um den Faktor 100 verstärkt werden. Zunächst die Schaltung:

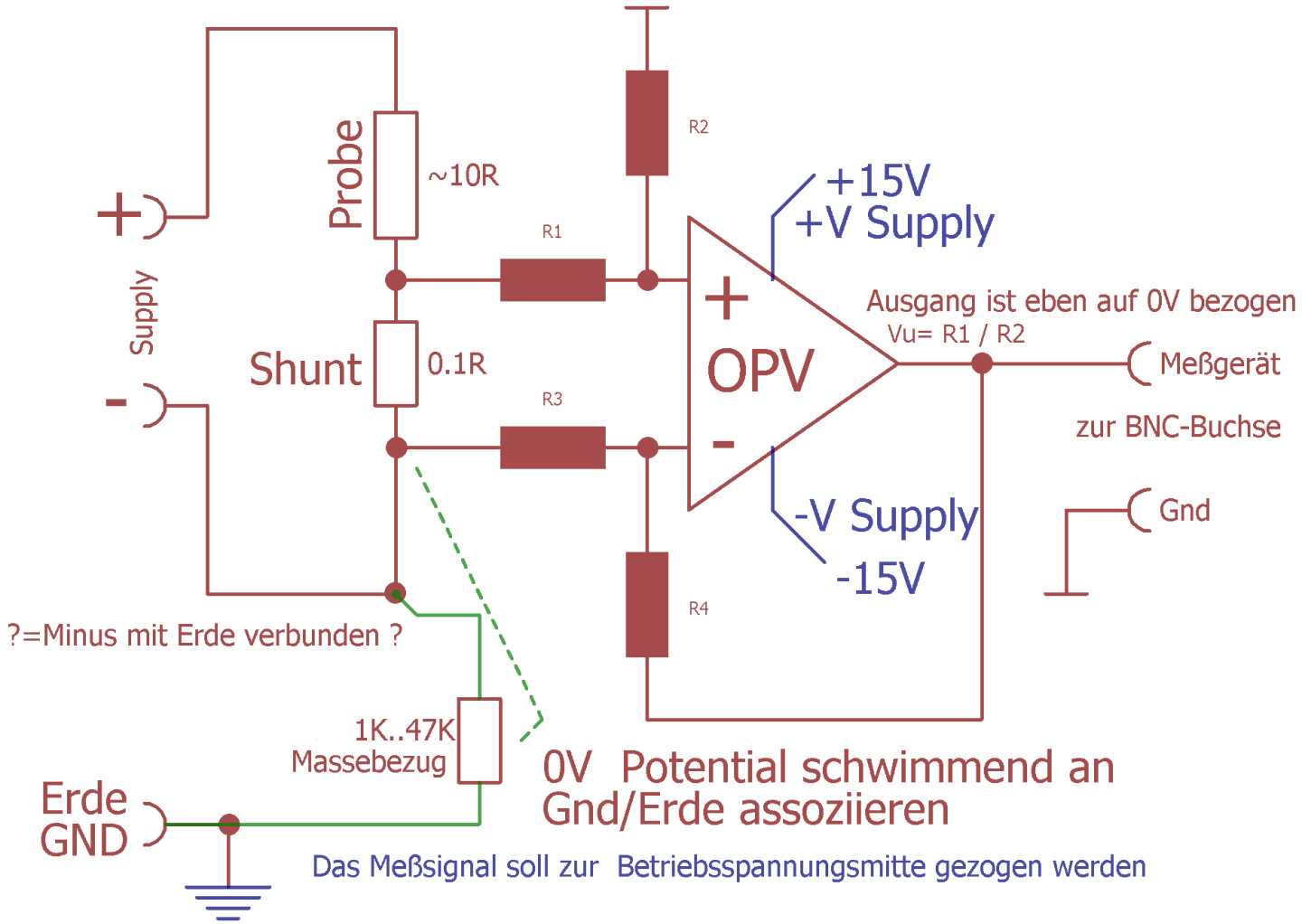
$$Ua = (U2 - U1) \cdot \left(1 + \frac{2 \cdot R1}{R2}\right)$$



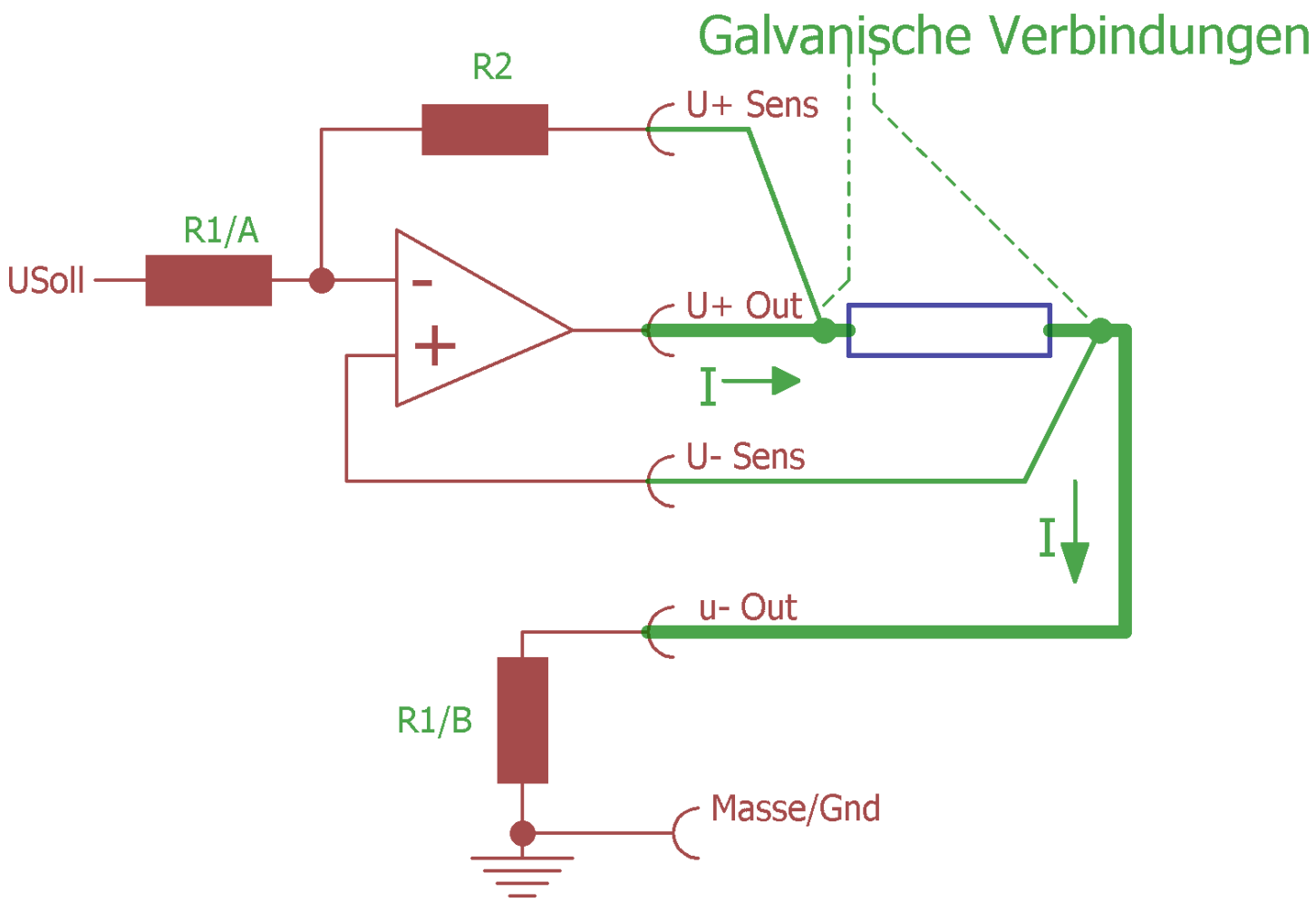
Jetzt müssen nur noch die Werte der Widerstände R1 und R2 bestimmt werden. Durch auflösen der Formel erhält man das benötigte Verhältnis: R1/R2 = 44,5. Jetzt kann z.B. für R2 ein Widerstand von 1 kOhm und für R1 ein in der E96 Reihe enthaltener 44,2 kOhm Widerstand

Potentialbezug hochohmiger Messsituationen

Da OP Betriebsspannungen haben, muss das Messsignal für den OP innerhalb dieser Betriebsspannung „SCHWIMMEN“



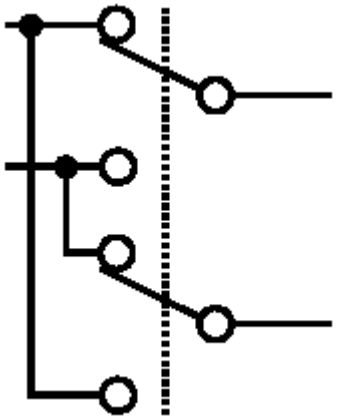
4-Punktmessung
Signalweg mit Verstärker



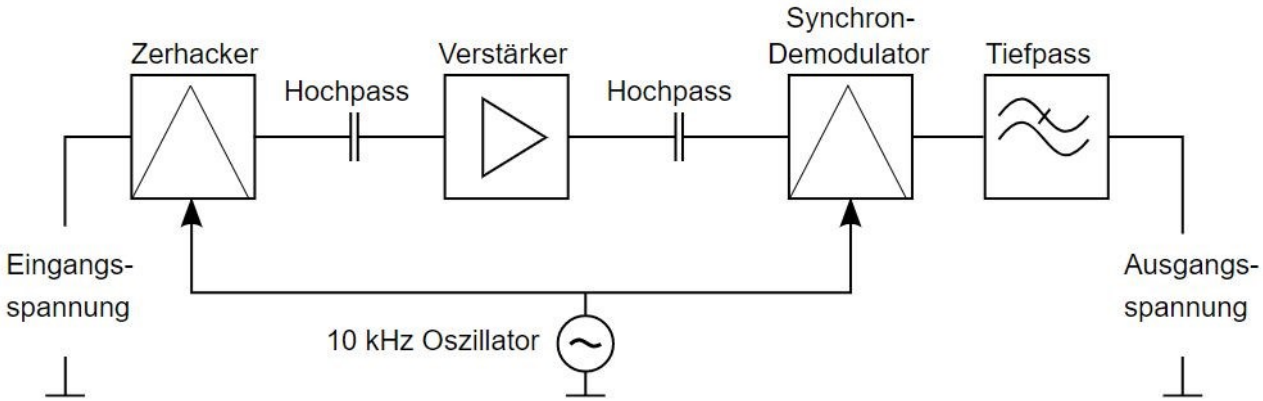
Chopper Verstärker

Um niedrige Pegel, die Offsetbelastet sind, zu messen z.B. Dehnungsmessstreifen in einer Brückenschalten, gibt es den wenig bekannten Chopper Verstärker. Dieser eliminiert effektiv 1/F Rauschen, das bei hoher Verstärkungen proportional auftritt und kompensiert auch Offset-Spannungen.
https://de.wikipedia.org/wiki/Chopper-Verst%C3%A4rker
Dies ist verwandt mit dem Lock-IN Prinzip

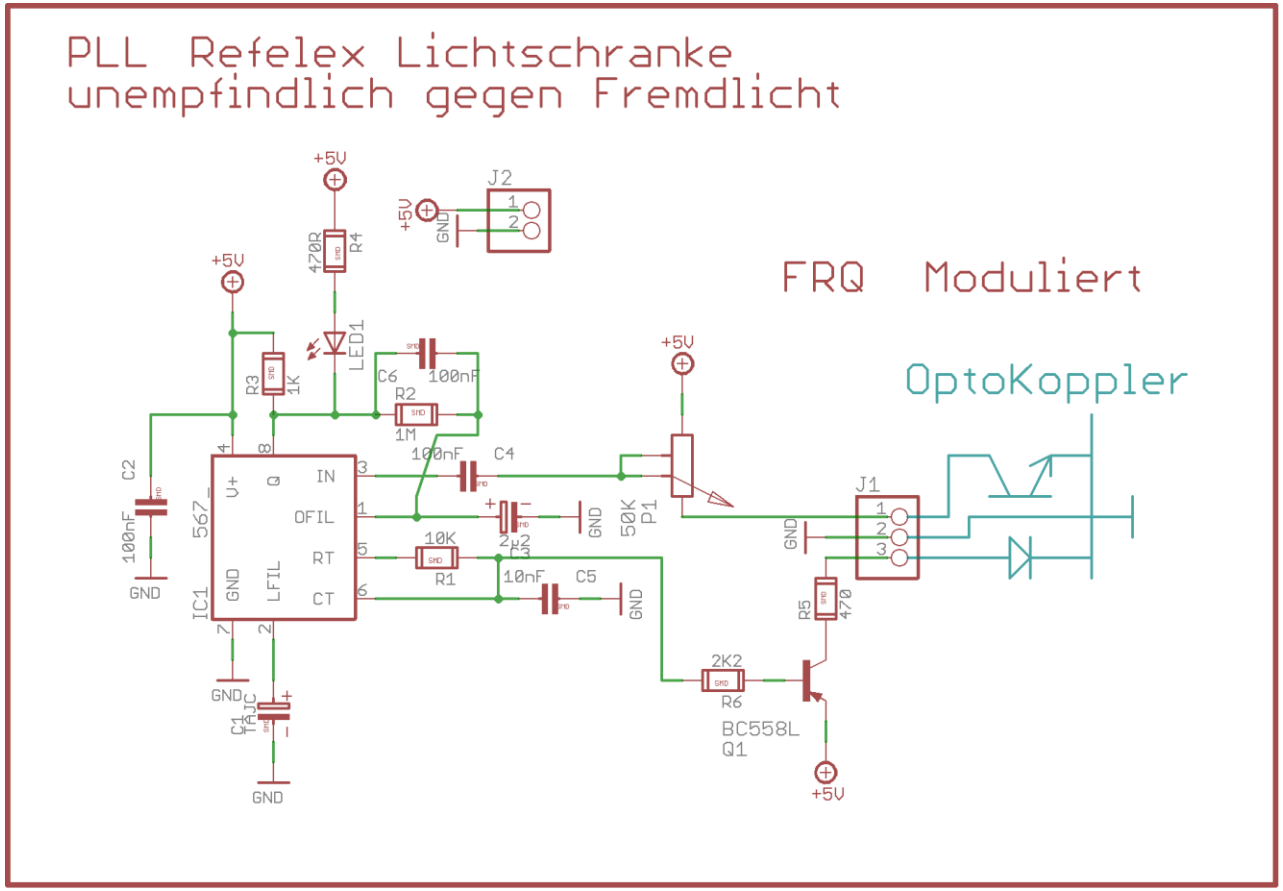
Ein typisches IC ist der ***ICL7650S.***



„Umpoler“ als multiplikative Mischstufe des Eingangssignals mit einem bipolaren Rechtecksignal



Lichtschranke mit moduliertem IR-Licht



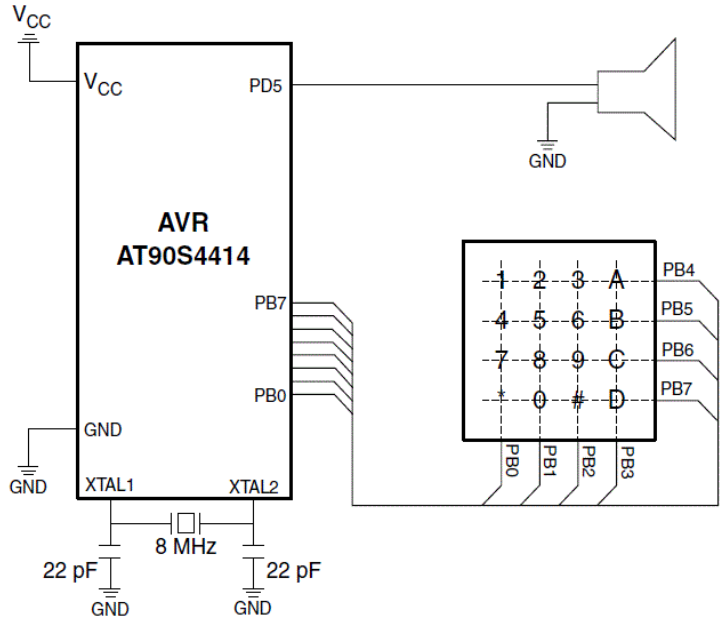
Trick.
Ein PLL Baustein **ICL567** erzeugt eine auf die er selbst als PLL reagiert.

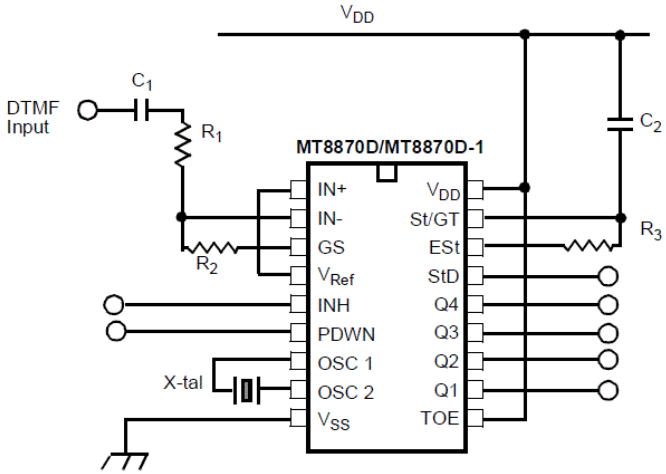
Nur wenn er diese sieht, gibt es einen Schaltpuls.
Somit ist der Optokoppler unempfindlich gegen Fremdlichteinstreung

DTFM = Telefon Ton Decoder

AVR als DTFM Generator
https://www.mikrocontroller.net/topic/238161
Und auch wieder per Software decodieren. Oder mit Decoder IC
Datenblatt des MT8870, hier:
http://pdf1.alldatasheet.com/datasheet-pdf/view/77085/MITEL/MT8870.html

Figure 1. DTMF Generator





NOTES:
R_1, R_2=100KΩ ± 1%
R_3=300KΩ ± 1%
C_1, C_2=100 nF ± 5%
X-tal=3.579545 MHz ± 0.1%

Figure 10 - Single-Ended Input Configuration

FUSES

Ein schwieriges Kapitel.!
UND VORSICHT. Falsche Einstellungen der Fuses können den teuren Mikrocontroller „**bricken**“. (= in einen nutzlosen Ziegelstein verwandeln)

Fuses sind Betriebsmodus Schalter die unabhängig vom Programm die Hardware und Funktionen des Mikrocontrollers beeinflussen,

Fuses können:
die Speichernutzung voreinstellen
die Funktionen einzelner Komponenten enable/disable
Wachtdog, Brownout Detection etc….
die Taktquelle umschalten auf Intern/extern Quarz VORSICHT !!!
Startup delay Time auswählen
Wiederbeschreiben, flashen sperren (Lock) !!! VORSICHT !!!
Auslesen sperren !!! VORSICHT !!!
Bootsector Sperren etc..
Es hilf nichts, nur ein „genauer“ Blick ins Handbuch schafft Klarheit…
Dort steht jedoch merkwürdiges.
Bit gesetzt → „1“ bedeutet: NICHT ausgewählt, oder kein Häkchen.
Bit gelöscht → „0“ bedeutet: Ausgewählt, oder Häkchen gesetzt.. Seltsam…!!

Die Bedeutung kann durch das Handbuch oder googeln erfahren werden. (kleine Kommentare sind ja schon da)

Gute Webseite: **http://www.engbedded.com/fusecalc/**

Manual fuse bits configuration

This table allows reviewing and direct editing of the AVR fuse bits. All changes will be applied instantly.

Note: ☐ means unprogrammed (1); ☑ means programmed (0).

Bit	*Low*	*High*	*Extended*
7	☐ **CKDIV8** Divide clock by 8	☐ **RSTDISBL** External reset disable	
6	☐ **CKOUT** Clock output	☐ **DWEN** debugWIRE Enable	
5	☐ **SUT1** Select start-up time	☑ **SPIEN** Enable Serial programming and Data Downloading	
4	☐ **SUT0** Select start-up time	☐ **WDTON** Watchdog Timer Always On	
3	☐ **CKSEL3** Select Clock Source	☐ **EESAVE** EEPROM memory is preserved through chip erase	
2	☐ **CKSEL2** Select Clock Source	☐ **BOOTSZ1** Select boot size	☐ **BODLEVEL2** Brown-out Detector trigger level
1	☐ **CKSEL1** Select Clock Source	☐ **BOOTSZ0** Select boot size	☑ **BODLEVEL1** Brown-out Detector trigger level
0	☐ **CKSEL0** Select Clock Source	☑ **BOOTRST** Select reset vector	☐ **BODLEVEL0** Brown-out Detector trigger level

Addendum

Wie gesagt, es soll kein C-Crash Kurs sein. Einige, von der Sprache eigentlich unabhängige, Konzepte und Strukturen haben sich jedoch bewährt.

Zuerst eine kleine Merkzettel Sammlung

Konventionen zur Schreibweise von Konstanten und Variablen

- Dies ist eine Beschreibung der von mir benutzen Technik in der Programmierung.
- Aber keine Vorschrift !! Jeder kann das machen, wie er selbst will. Klar.
- Letztlich gelten nur die Syntaxvorschiften, 1978 erschienen von Kerningham und Ritchie in dem Buch "Programmieren in C".

- Jedoch.... ist dies der Unterschied zwischen einem "Hauruck-Basic-Programmierer" und einem "näher am Profi"..
- Also eine freundliche Empfehlung:: die ich in meinen Beispielen verwende:
- So kann man diese besser verstehen, und !! Fehler schnell finden. Weil man "sieht", was was bedeutet...!

- Es ist nur eine Handvoll simpler Regeln, die auch MS/Linux Profis verwenden (mehr oder weniger).

1. Es gibt **keine EIN-Buchstaben Variablen**.

Das ist unter „Profis“ verpönt. Aus gutem Grund.

Bsp: for(**i** = 0;….

"*ist stümperhaft, kein sprechender Name, ein Datentyp erkennbar etc.*".

Besser: typisiert und ein "sprechender Name" wie "u8ByteCount".

Was das bedeutet gleich mehr.

2. **Konstanten**, und **Definitionen**, **Makros** werden immer nur mit „**GROßBUCHSTABEN“** geschrieben

also typischerweise alles im Headerfile wie

```
#define    MEINEERFINDUNG 654321UL
```

3. aber auch alle anderen Konstanten und Makrodefines,

Bsp: *alles **Großgeschrieben***:

```
const char MCA_DELIMTERSTR[] PROGMEM = { ',',0 }
#define MOTOR_ENABLE_FLAG          0x02
```

4. Makros

```
#define LED1_ON() (LED_PORT |= LED1_MSK)
#define LED1_OFF()            (LED_PORT &= ~LED1_MSK)
```

3a
Variablen werden am Wortanfang mit Kleinbuchstaben mit dem Variablentyp gebrandmarkt. "Wie ? kommt gleich"
3b
Variablen-Namen werden möglichst "sprechend", nach der 3a-Regel mit den Typenkleinbuchstaben,
dann mit jedem "Silbenanfang" einem weiteren "GroßBuchStaben,
gefolgt von den "Kleinbuchstaben" der Silbe, geschrieben.

Das ist alles !!
am Besten ein Beispiel:

```
char          gcaUartRXBuffer[80+1];    //lese : global char array Uart RX (empfangs) Buffer (speicher)
unsigned char   *  pu8Parser        //lese (lokaler) Pointer auf unsigned 8Bit mit Namen Parser
uint32_t   gu32Ticks    //lese unsigned int mit 32 Bit , Ticks → TickCounter
u..sw..
```

Liste der Typen-Kleinbuchstaben-Kürzel: (in der Reihenfolge, die ich verwende)

g= global definierte Variable. // Ohne "g" ist Var. also eine lokale
Funktionsvariable.
p = Pointer. // Var, ist ein Pointer
s = Struct Variable
u = unsigned // signed ist default
c = char
a = Array; //Var ist ein Array (bei Strings die zusätzliche 0 nicht vergessen)
8, 16, 32, 64 //Bitbreite der Variable... praktischer als char integer word... Und eindeutiger

in GNU-C von WinAVR wird anstelle von char integer word usw. eine einheitliche Form verwendet, die die Bitbreite angibt.

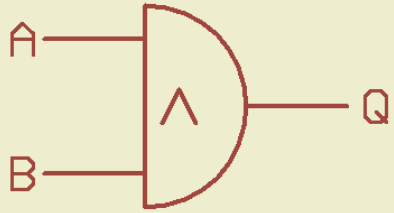
int8_t ist short, char;
uint8_t = quasi byte;
uint16_t = unsigned int; int16_t = integer usw..
uint32_t

Das war schon alles! Die paar Buchstaben zu lernen ergibt sich automatisch am Type-namen .. c=char usw..
Man gewöhnt sich schnell daran, und möchte es irgendwann nicht mehr anders..
Die Zeiten von for(i = ..) sind bald vorbei.

Spickzettel

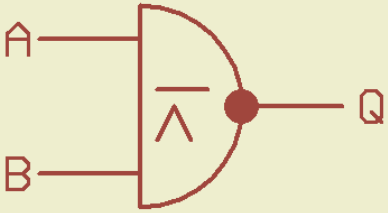
Binäre Grundelemente und deren Logik Tabellen

Alte amerikanische Symbole



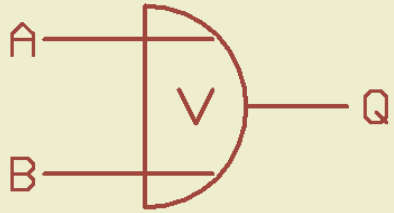
AND

A	B	Q
0	0	0
1	0	0
0	1	0
1	1	1



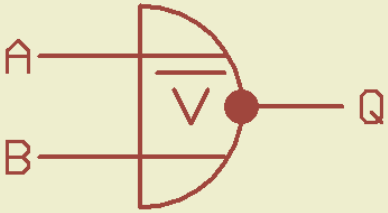
NAND

A	B	Q
0	0	1
1	0	1
0	1	1
1	1	0



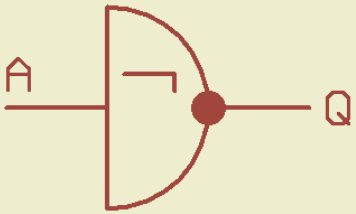
OR

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	1



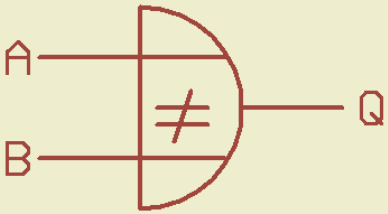
NOR

A	B	Q
0	0	1
1	0	0
0	1	0
1	1	0



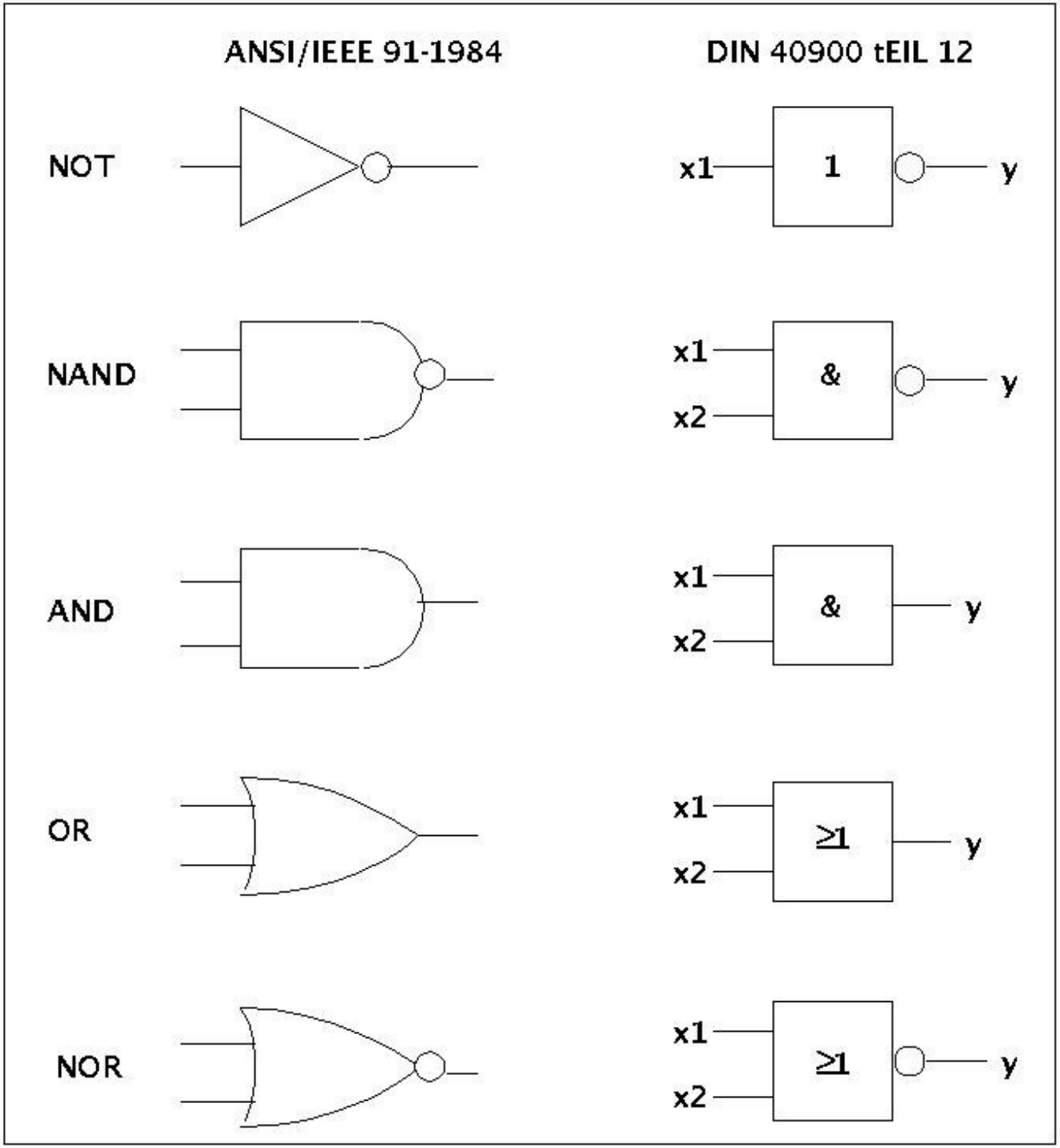
NOT

A	Q
0	1
1	0



EXOR

A	B	Q
0	0	0
1	0	1
0	1	1
1	1	0



Links
Amerikansiche
zeichen

Rechts
Europäsiche
DIN Zeichen
(ich mag die nicht)

Hier hängen noch ein paar Übersichtstabellen etc. an

Byte Darstellung mit Werten der Bitposition

Position	8	7	6	5	4	3	2	1
Wertigkeit	2^7 (128)	2^6 (64)	2^5 (32)	2^4 (16)	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
Beispiel	1	0	1	1	1	0	0	1

(1* (2^7))+(0*(2^6))+(1*(2^5))+(1*(2^4))+(1*(2^3))+(0*(2^2))+(0*(2^1))+(1*(2^0))

Wie übersetzt man zu fuß eine Binärzahl in seinen Dezimalwert?

Nur eine Übung !
Und zum Nachschlagen

Bits zählt man von 0 bis 7, nicht von 1 bis 8. ! 0..7
Das 7.Bit ist als das "letzte" Bit von 8 Bit

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 256 = 2^8

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
2^7	**2^6**	**2^5**	**2^4**	**2^3**	**2^2**	**2^1**	**2^0**
128	64	32	16	8	4	2	1

Trick:
Man stellt die Bits, die man ausrechnen will, einfach über diese Tabelle:
Dann addiert man einfach die 2^n-te Stelle, wo eine EINS ist.
Wo ein NULL ist, nichts addieren.
Bsp: Binär **1010 0100** = dezimal 164 = hexadezimal 0xA4

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	**1**	0	0	**1**	0	0
128 +	0	32 +	0	0	4	0	0

Ergebnis: 128 + 32 + 4 = 164

CRC Prüfungen der Daten
Cyclic Redundancy Check

Zyklische Redundanzprüfung → ein Prüfsummenverfahren.
Da bei der Datenübertragung Fehler passieren können, braucht man ein Protokoll mit einer Prüfung.
Dazu gibt es viele Links im Web;
https://www.mikrocontroller.net/articles/CRC
Die Berechnung des CRC-Werts beruht auf Polynomdivision: :
Bsp: Die Bitfolge 1,0,0,1,1,1,0,1 entspricht dem Polynom:

$$1 \cdot x^7 + 0 \cdot x^6 + 0 \cdot x^5 + 1 \cdot x^4 + 1 \cdot x^3 + 1 \cdot x^2 + 0 \cdot x^1 + 1 \cdot x^0 = x^7 + x^4 + x^3 + x^2 + 1$$

Die Bitfolge der Coderepräsentation der Daten wird durch ein vorher festzulegendes Generatorpolynom (das CRC-Polynom) Modulo mod(2) geteilt, wobei ein Rest bleibt. Dieser Rest ist der CRC-Wert. Bei der Übertragung des Datenblocks hängt man den CRC-Wert an den originalen Datenblock an und überträgt ihn mit.
http://de.wikipedia.org/wiki/Zyklische_Redundanzpr%C3%BCfung
http://de.wikipedia.org/wiki/Secure_Hash_Algorithm

https://cpp.hotexamples.com/examples/-/-/CRC16_Calc/cpp-crc16_calc-function-examples.html

Anwendung CRC16.c

```
#define CRC16_INIT_VALUE    0xFFFF    //!< initial value for CRC algorithem
#define CRC16_GOOD_VALUE    0x0F47    //!< constant compare value for check
#define CRC16_POLYNOM       0x8408    //!< 16-BIT CRC CCITT POLYNOM
```

UINT16 CRC16_Calc (UINT8* data, UINT16 length, UINT16 initVal);

Diese Funktion berechnet das Einerkomplement der Norm

16-BIT CRC CCITT-Polynom G(x) = 1 + x^5 + x^12 + x^16

Bool CRC16_Check (UINT8* data, UINT16 length, UINT16 initVal);

This function checks a data block **with** attached CRC16

```
//Testdaten     4 BYTE
u8aDatas[0] = 0xA5;
u8aDatas[1] = 0x5A;
u8aDatas[2]= 0xA5;
u8aDatas[3]= 0x5A;
u8aDatas[4]= 0;
u8aDatas[5]= 0;
u8Lenght = 4;   //DATENBYTES
```

```
            //Erzeuge CRC Wert
u16Crc = CRC16_Calc(u8aDatas, u8Lenght, CRC16_INIT_VALUE);
            printf("\r\nCRC Calc Init = %d  0x%x", u16Crc, u16Crc);
for(u8NN=0; u8NN < u8Lenght; u8NN++)
            { printf("\r\nVal 0x%x", u8aDatas[u8NN] ); };
```

1.

```
u16Crc = ~u16Crc; // Mache Komplement.. Warum? Ist so !!
```

2.

```
u8aDatas[u8Lenght++]= u16Crc & 0xFF;  //LOW BYTE
u8aDatas[u8Lenght++]= (u16Crc >> 8);  //HIGH BYTE

u16Erg =  CRC16_Check(u8aDatas, u8Lenght,  CRC16_INIT_VALUE);
            printf("\n\r\n  u16Check OHNE Fehler in 1 = %d  0x%x", u16Erg, u16Erg);
// Liefert 1 zurück wenn alles stimmt

//MACHE FEHLER
u8aDatas[1] = 0x5B;
u16Erg =  CRC16_Check(u8aDatas, u8Lenght,  CRC16_INIT_VALUE);
```

Liefert 0 zurück, wenn Fehler.

CRC Calc Init = 4938 0x134a
Val 0xa5
Val 0x5a
Val 0xa5
Val 0x5a

CRC = ~CRC; //komplement
dann
(mit Angehängter CRC)

Val 0xa5
Val 0x5a
Val 0xa5
Val 0x5a
Val 0xb5
Val 0xec

u16Check OHNE Fehler in 1 = 0 0x0
Val 0xa5
Val 0x5b
Val 0xa5
Val 0x5a
Val 0xb5
Val 0xec

u16Check Fehler in 1 = 0 0x0
Val 0xa5
Val 0x5b
Val 0xa5
Val 0x5a
Val 0xb5
Val 0xec

Die State Machine

Eine vereinfachte Form der Programmsteuerung ist die State-Machine
Einfache #defines (als Nummern) legen Zustände fest.

```
//STATE MACHINE
//(Fortlaufende Nummern von 0..255, frei erfindbar..
)
#define STATE_MACH_NIX                  0
#define STATE_IN_ACTION                 1
#define STATE_IR_AUTOSTOP               5
#define STATEMACH_BLINK_GREEN_ON        10
#define STATEMACH_BLINK_GREEN_OFF       11
```

```
void StateMachine(void)
{ /// --------------STATE MACHINE  ----------------------------------------
switch(  gu8StateMachine  )
    {
    case STATE_MACH_NIX:          //blödsinn aber so gehts
        {      break;     };
   case STATE_IN_ACTION:
        {        break;        };
   case STATE_IR_AUTOSTOP:
        {
     if( GetIR_Sens() > gu16IR_Sensibility )
        };
    } // Switch
```

State Machine Codes

Aus „statmach.h“

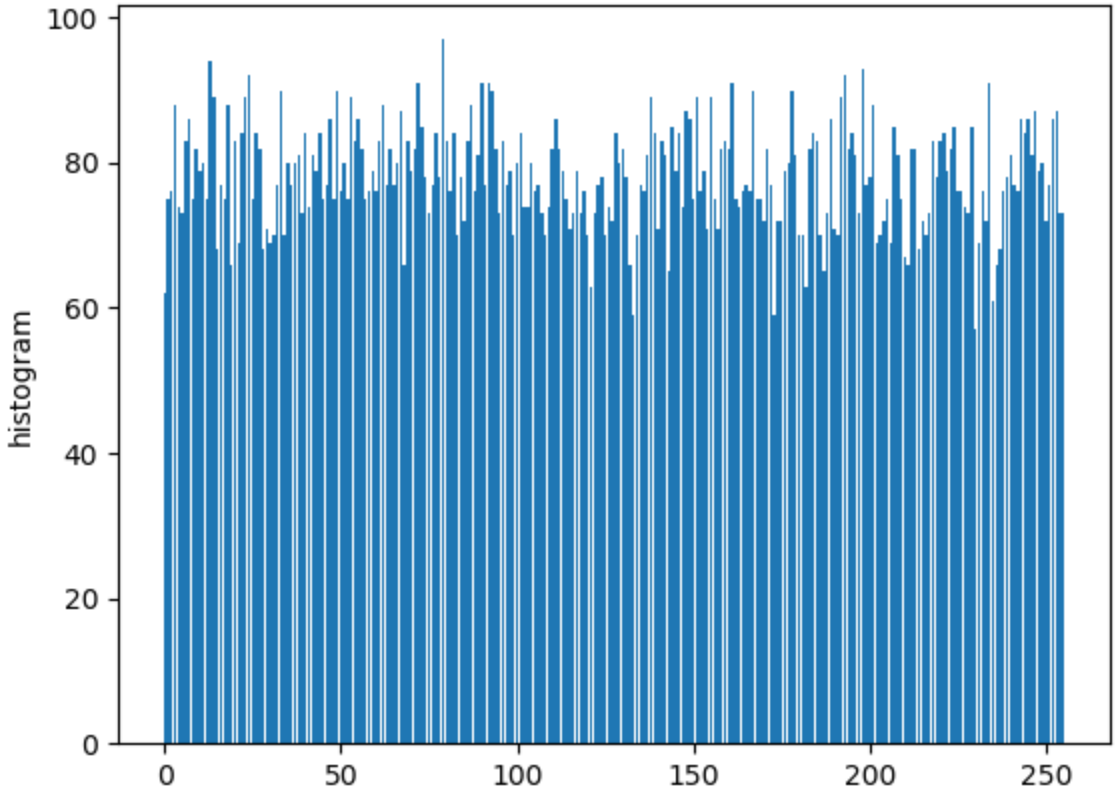
```
#define STATE_MACH_NIX                          0
#define STATE_IN_ACTION      = IRControlle      1
#define STATEMACH_BLINK_GREEN_ON                10
#define STATEMACH_BLINK_GREEN_OFF               11

#define STATE_FREE                              20
#define STATE_STOP                              21
#define STATE_FAHRE_VORWAERTS                   22
#define STATE_FAHRE_RUECKWAERTS                 23
#define STATE_DREHE_RECHTS                      24
#define STATE_DREHE_LINKS                       25

#define STATE_SCHWENKE_DAS_IR_RADAR             222
#define STATE_FAHREGERADE_MIT_IR_RADAR          233
```

FUNDUS

Diskussion über Pseudozufall
Aufhebenswert..
Von Jörg W. dl8dtl https://www.mikrocontroller.net

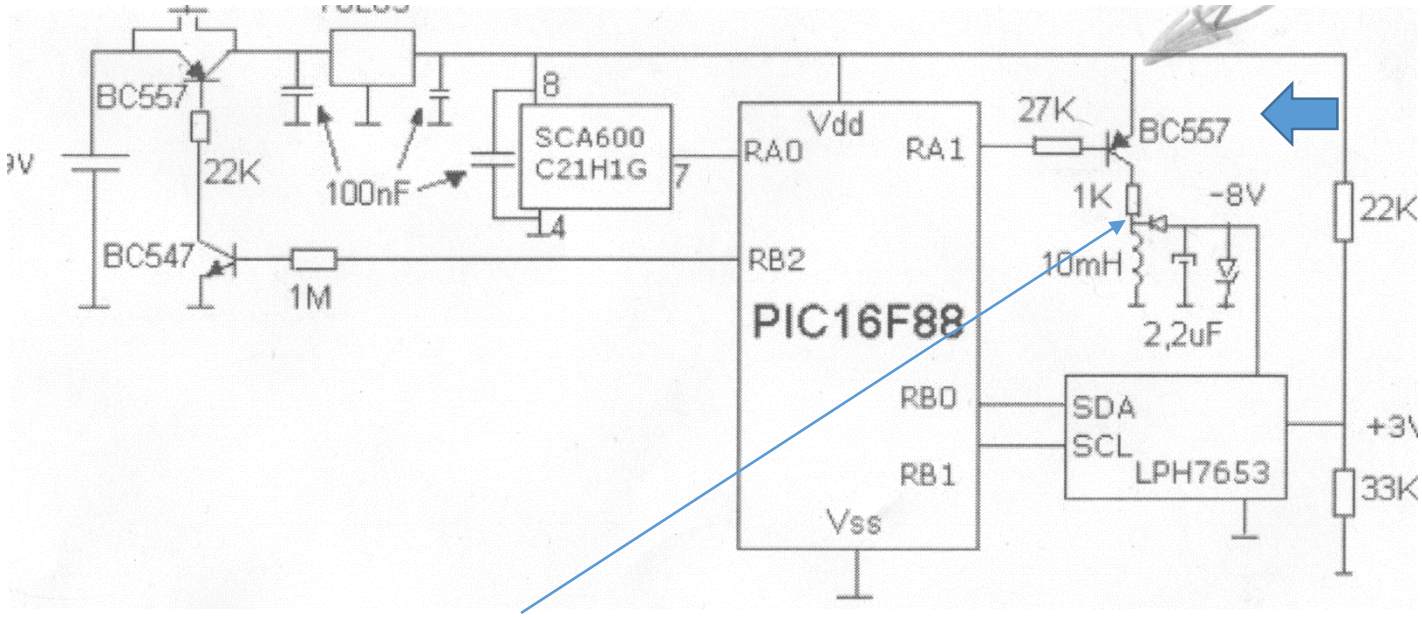
Er schrieb:
Die Funktion lsfr1() habe ich mir aus dem englischen Wikipediaartikel:
https://en.wikipedia.org/wiki/Linear-feedback_shift_register

```
uint8_t lfsr1(void)
{
   static uint16_t lfsr = 0xe1;
   uint8_t bit;
   unsigned period = 0;

   /* taps: 8 6 5 4; feedback polynomial: x^8 + x^6 + x^5 + x^4 + 1 */
   bit = ((lfsr >> 0) ^ (lfsr >> 2) ^ (lfsr >> 3) ^ (lfsr >> 4)) /* & 1u */;
   lfsr = (lfsr >> 1) | (bit << 15);

   return lfsr;
}
```


100
80
60
40
20
0
histogram
0
50
100
150
200
250

Gleich zwei Tricks. 11.2018

Einschalten mit Taster, der den Transistor kurz überbrückt. Eine Selbsthaltungsrückführung hält den Transistor auf.
Der Controller kann die Schaltung selbst deaktivieren



Erzeugung einer „negativen, oder auch positiven höheren Hilfsspannung mit Abschaltpulsen an der Induktivität. ~10KHz (experimentieren)

“ ~ungeregelt -8V, und mehr!“, viel mehr möglich.

Ich empfehle Siebung, mindestens mit Zehnerdiode.

Es gibt eine gute Webseite um die Werte für Fuses sicher zu ermitteln, sofern man die Parameter versteht. Tipp: „Read your fucking Manual“
http://www.engbedded.com/fusecalc/

Die Webeseite erlaubt die Parameter auf verschiedene Weise einzugeben Funktionen mit Kästchen auswählen.

AVR part name: ATmega328P ▾ Select (141 parts currently listed)

Feature configuration

This allows easy configuration of your AVR device. All changes will be applied instantly.

Features

Ext. Crystal Osc.; Frequency 8.0- MHz; Start-up time PWRDWN/RESET: 16K CK/14 CK + 65 ms; [CKSEL=1111 SUT=11]

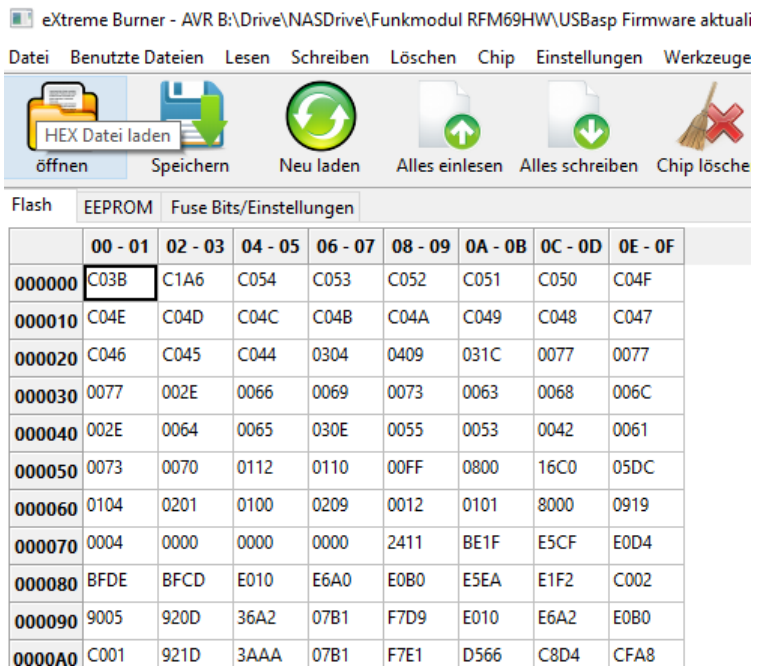
- [ ] Clock output on PORTB0; [CKOUT=0]
- [ ] Divide clock by 8 internally; [CKDIV8=0]
- [x] Boot Reset vector Enabled (default address=$0000); [BOOTRST=0]

Boot Flash section size=256 words Boot start address=$3F00; [BOOTSZ=11] ▾

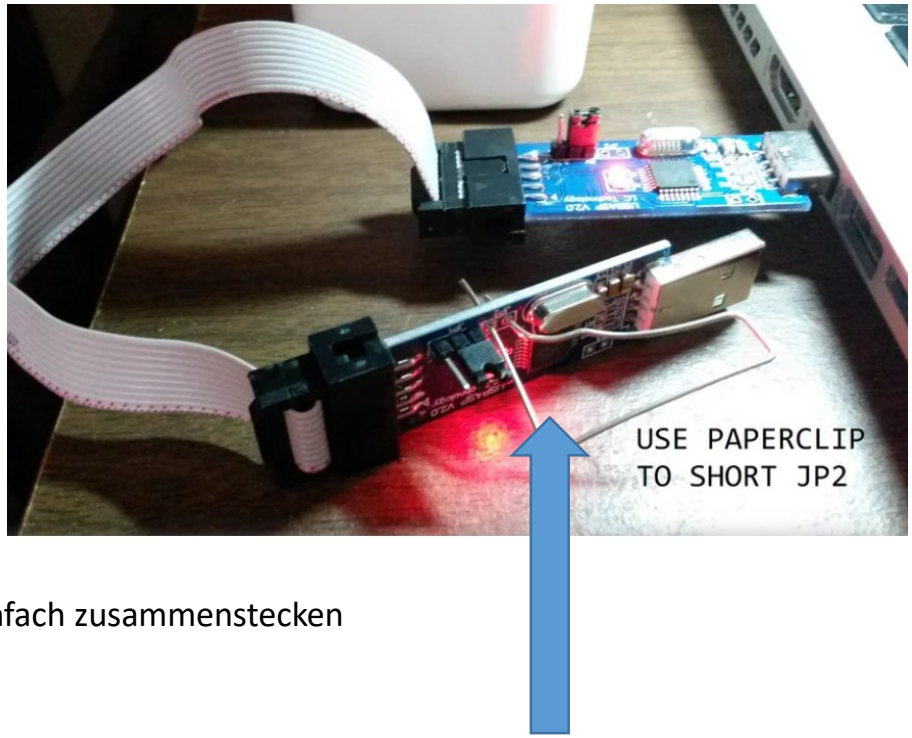
- [ ] Preserve EEPROM memory through the Chip Erase cycle; [EESAVE=0]
- [ ] Watch-dog Timer always on; [WDTON=0]
- [x] Serial program downloading (SPI) enabled; [SPIEN=0]
- [ ] Debug Wire enable; [DWEN=0]
- [ ] Reset Disabled (Enable PC6 as i/o pin); [RSTDISBL=0]

Brown-out detection level at VCC=2.7 V; [BODLEVEL=101] ▾

USBasp Firmware update - **Bessere Methode, aber man braucht ZWEI USBasp.**
https://www.instructables.com/Firmware-Upgrade-for-USBASP-Clone-Fixing-Error-Set/
Mit eXtreme Burner
Firmware: https://www.fischl.de/usbasp/



	00 - 01	02 - 03	04 - 05	06 - 07	08 - 09	0A - 0B	0C - 0D	0E - 0F
000000	C03B	C1A6	C054	C053	C052	C051	C050	C04F
000010	C04E	C04D	C04C	C04B	C04A	C049	C048	C047
000020	C046	C045	C044	0304	0409	031C	0077	0077
000030	0077	002E	0066	0069	0073	0063	0068	006C
000040	002E	0064	0065	030E	0055	0053	0042	0061
000050	0073	0070	0112	0110	00FF	0800	16C0	05DC
000060	0104	0201	0100	0209	0012	0101	8000	0919
000070	0004	0000	0000	0000	2411	BE1F	E5CF	E0D4
000080	BFDE	BFCD	E010	E6A0	E0B0	E5EA	E1F2	C002
000090	9005	920D	36A2	07B1	F7D9	E010	E6A2	E0B0
0000A0	C001	921D	3AAA	07B1	F7E1	D566	C8D4	CFA8



Einfach zusammenstecken

Noch ein Link dazu: https://kopterforum.at/viewtopic.php?t=825
GUI → AVRDUDESS https://blog.zakkemble.net/avrdudess-a-gui-for-avrdude/

UASBasp Installieren
http://stefanfrings.de/avr_tools/libusb.html

USBasp Firmware update (ohne zweiten USBasp) mit Arduino UNO Board

http://www.rogerclark.net/updating-firmware-on-usbasp-bought-from-ebay/

Wenn die Meldung kommt:
avrdude: warning: cannot set sck period. please check for usbasp firmware update

Download the latest version of the USBASP firmware from
http://www.fischl.de/usbasp

NOTE. Attempting to reprogram your USBASP may damage or erase it. Only follow the process described below if you are willing to take this risk 😉

Program an Arduino UNO as an ISP.

This is a fairly simple process.

Plug in your Arduino.

From the Examples (on the File menu). Select “Arduino ISP”

Select “Upload” and confirm that the new firmware has been uploaded. Note down the Comm port that the Arduino board is connected via i.e from the Tools->Serial port menu

Download the latest version of the USBASP firmware from http://www.fischl.de/usbasp

I used usbasp.2011-05-28.tar.gz

unzip the file and note the full path of the the bin/firmware directory for the firmware you have just downloaded and unzipped

Open a command prompt (windows cmd), and Cd to your Arduino program folder and then into the hardware\tools\avr\bin folder

This folder should contain the avrdude.exe that will be used to flash the new firmware.

Link the Self Program jumper on the USBASP – on my board this was labeled J2, some other boards are different.

I had to solder the pins onto the PCB as they were missing, and used a jump link, but any way to reliably connect the Self Program pins should work fine.

Connect the Arduino to the programming pins of the USBASP

Make sure you disconnect the USBASP from your computers USB My connections were

Arduino USBASP

5V ——— 2
GND ——— 10
13 ——— 7
12 ———- 9 (MISO)
11 ———- 1 (MOSI)
10 ——— 5 (RESET)

Check that avrdude can connect to the USBASPIn the windows command window, type
avrdude -C ../etc/avrdude.conf -c avrisp -P COM8 -b 19200 -p m8 -v

Note if your Arduino is not on COM3 you need to change this to whatever your Arduino IDE used.

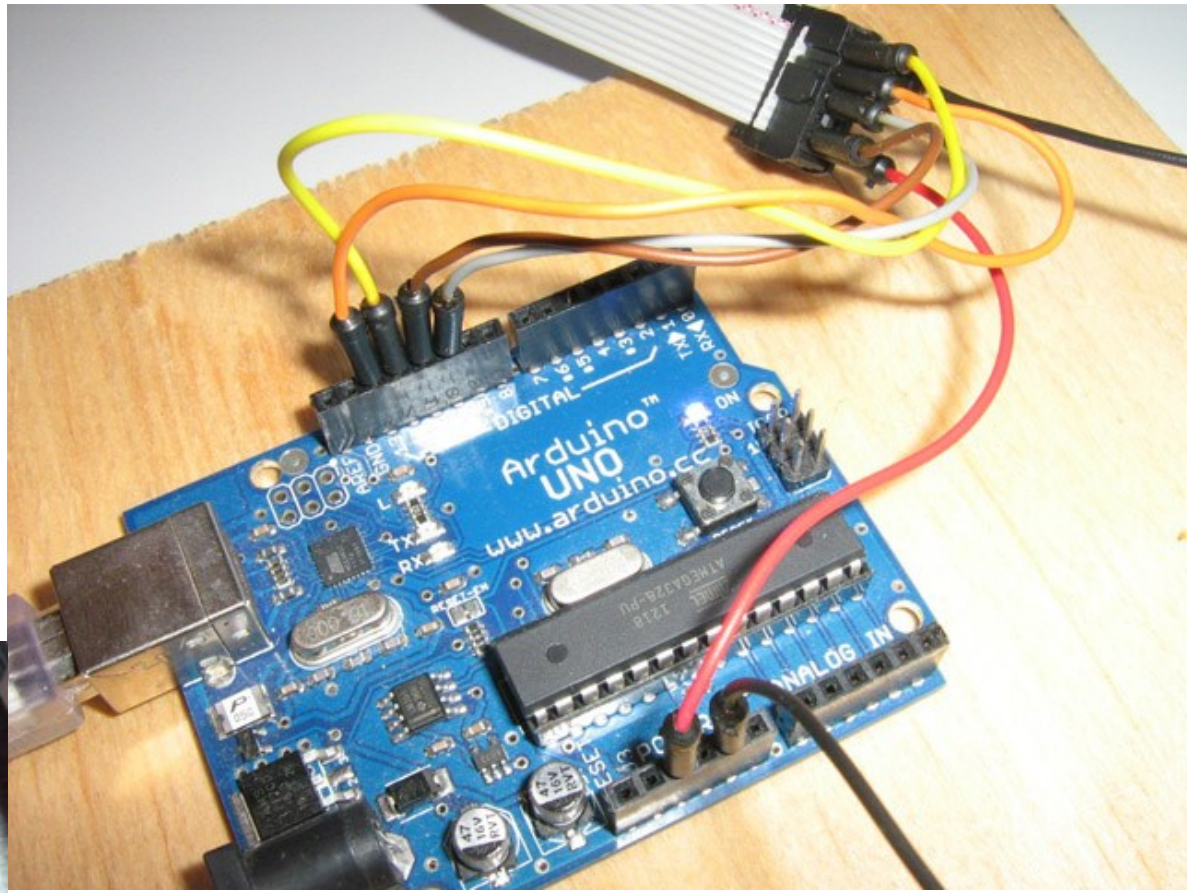
If everything is connected correctly you should see a load of information about the USBASP board that you are about to program, like this

Update Firmware am USB-ASP mit Arduino als ISP
Zuerst:

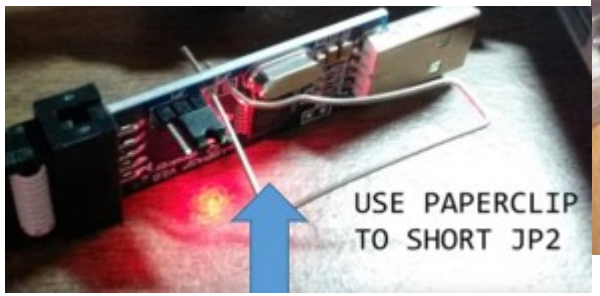
Arduino als ISP USBasp Verbinden wie Plan:

Make sure you disconnect the USBASP from your computers USB My connections were

Arduino USBASP
5V ——— 2
GND ——— 10
13 ——— 7
12 ——— 9 (MISO)
11 ——— 1 (MOSI)
10 ——— 5 (RESET)



Lade in Sketch Programm Arduino ISP auf den Arduino



```
Using Port                    : COM8
Using Programmer              : avrisp
Overriding Baud Rate          : 19200
AVR Part                      : ATMEGA8
Chip Erase delay              : 10000 us
PAGEL                         : PD7
BS2                           : PC2
RESET disposition             : dedicated
RETRY pulse                   : SCK
serial program mode           : yes
parallel program mode         : yes
Timeout                       : 200
StabDelay                     : 100
CmdexeDelay                   : 25
SyncLoops                     : 32
ByteDelay                     : 0
PollIndex                     : 3
PollValue                     : 0x53
Memory Detail                 :
```

```
Programmer Type : STK500
Description     : Atmel AVR ISP
Hardware Version: 2
Firmware Version: 1.18
Topcard         : Unknown
Vtarget         : 0.0 V
Varef           : 0.0 V
Oscillator      : Off
SCK period      : 0.1 us

avrdude: AVR device initialized and ready to accept instructions
```

If you get an error message about invalid signature, your board is may not use an ATMega8, and you will need to change the -p option on the AVRDUDE command to match you board. If you omit the -p option, AVRDUDE will display a list of compatible devices and their codes which you can use to select the correct device.If you have the correct device, but still get a signature error, check that you are using -c avrisp and NOT -c arduino.

Using -c arduino tells AVRDUDE that you want to program the Arduino board and not the USBASP

If you can’t connect at all, check your wiring, as its possible that some boards don’t have the same pinout as my board.

Assuming that AVRDUDE can connect to the USBASP, you can backup the original firmware using the commandavrdude -C ../etc/avrdude.conf -c avrisp -P COM3 -b 19200 -p m8 -U flash:r:original_firmware.bin:r
This should read the firmware and save it to a file called original_firmware.bin (in the same folder as avrdude.exe)

As a double check you can attempt to write the original firmware back to the device, I'm not sure if this is a foolproof way of testing whether you can flash the USBASP but I did it as a test, as previously I'd had issues writing / flashing the device but was able to read it OK.avrdude -C ../etc/avrdude.conf -c avrisp -P COM3 -b 19200 -p m8 -U flash:w:original_firmware.bin
If this appears to work and AVRDUDE verifies OK, then its probably safe to attempt to burn the new firmware.

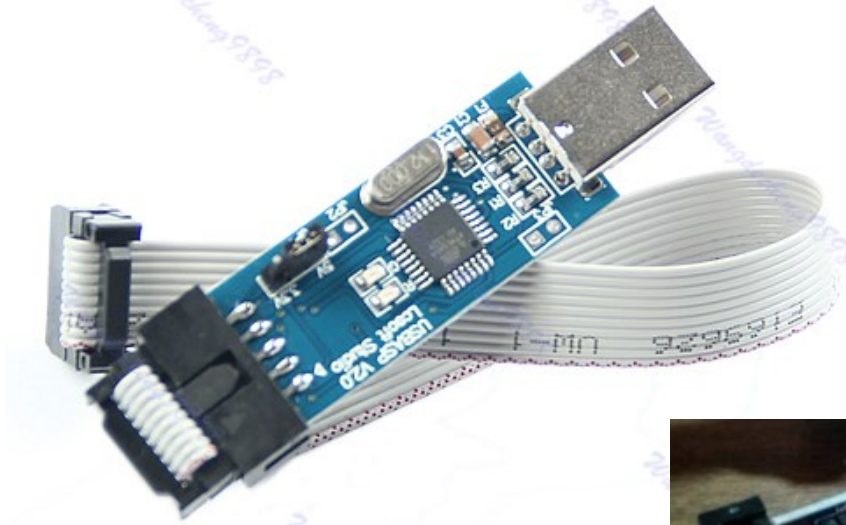
In my case the new firmware was in c:\usbasp so the command was

avrdude -C ../etc/avrdude.conf -c avrisp -P COM3 -b 19200 -p m8 -U flash:w:c:\usbasp.atmega8.2011-05-28.hex

Note. If the chip on your USBASP is not a MEGA8, you need to use the correct / matching file e.g.

avrdude -C ../etc/avrdude.conf -c avrisp -P COM3 -b 19200 -p m8 -U flash:w:c:\usbasp.atmega88.2011-05-28.hex

If the process worked ok and AVDRUDE verified the new firmware, you can disconnect the USBASP from the Arduino and connect it via USB to the PC and attempt to program your target device (using the Arduino IDE or AVRDUDE).



Verbinde J2 am USB-ASP

Check that avrdude can connect to the USBASP
In the windows command window, type

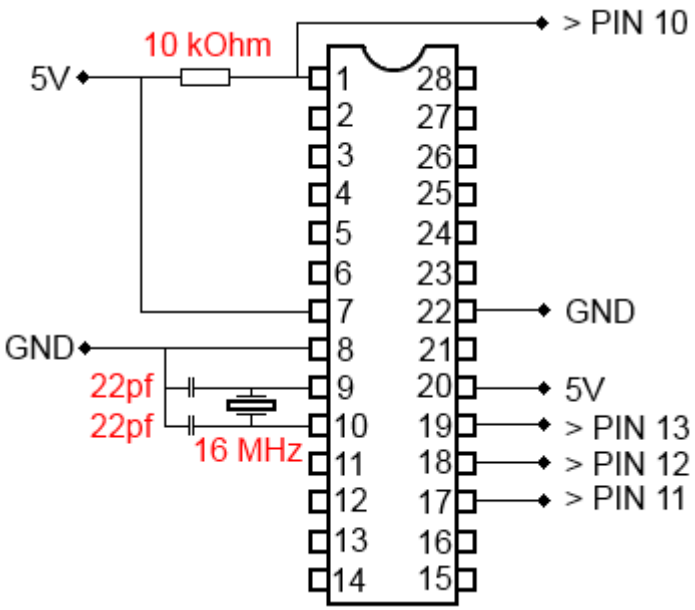
open cmd
goto cd C:\Program Files (x86)\Arduino\hardware\tools\avr/bin
C:\Program Files (x86)\Arduino\hardware\tools\avr/bin
dann
avrdude -C ../etc/avrdude.conf -c avrisp -P COM8 -b 19200 -p m8 -v
flash:w:c:\usbasp\usbasp.atmega8.2011-05-28.hex

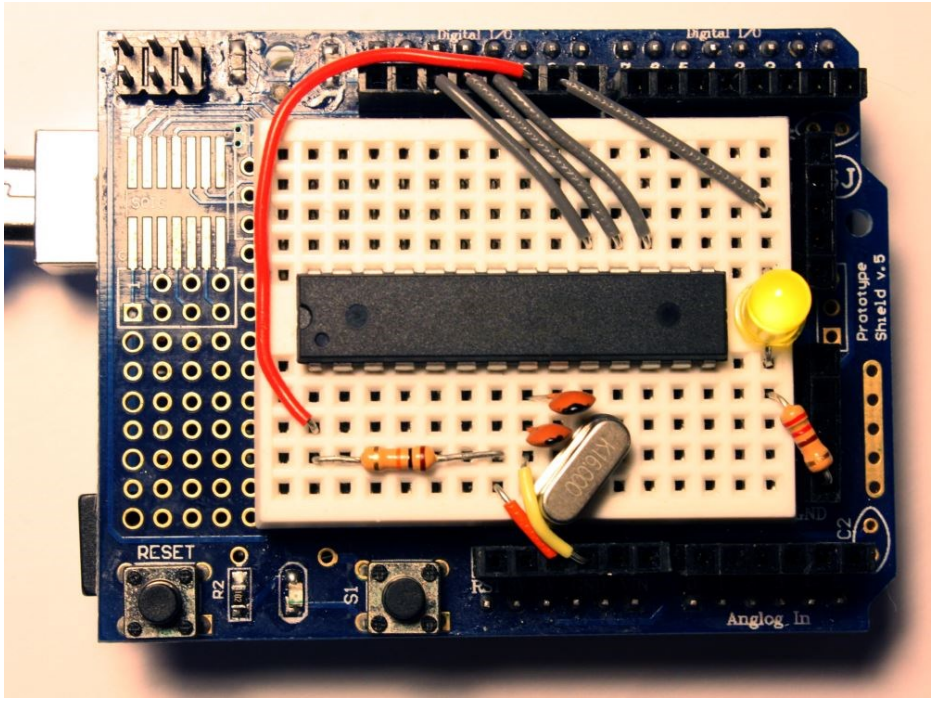
firmware ist in C:\usbasp\

SIEHE:
http://www.rogerclark.net/updating-firmware-on-usbasp-bought-from-ebay/
https://www.das-labor.org/wiki/Usbasp
http://blog.lincomatic.com/?p=1480

Arduino Bootloader auf ATMega328p laden

Dazu diese Schaltung aufbauen



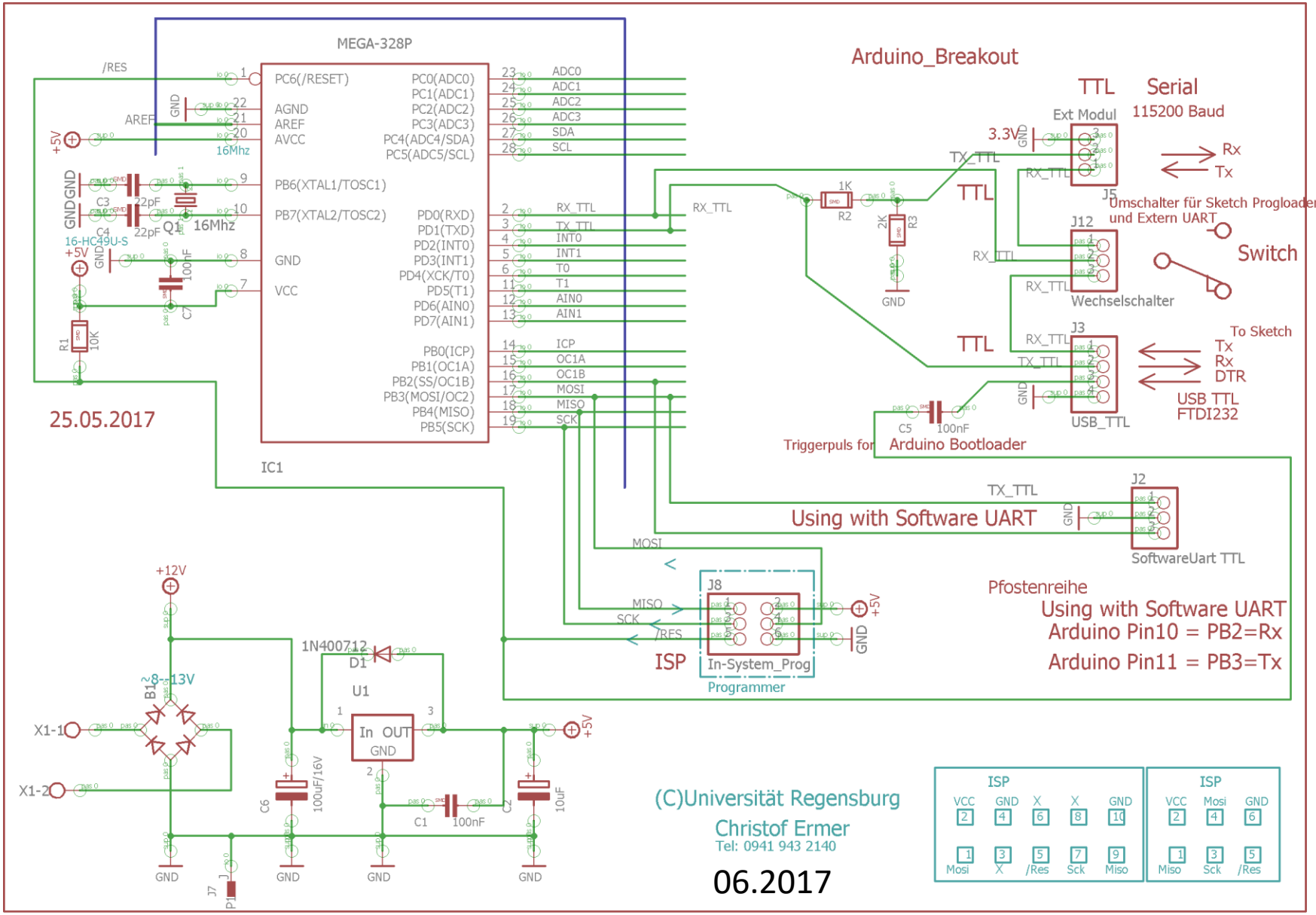


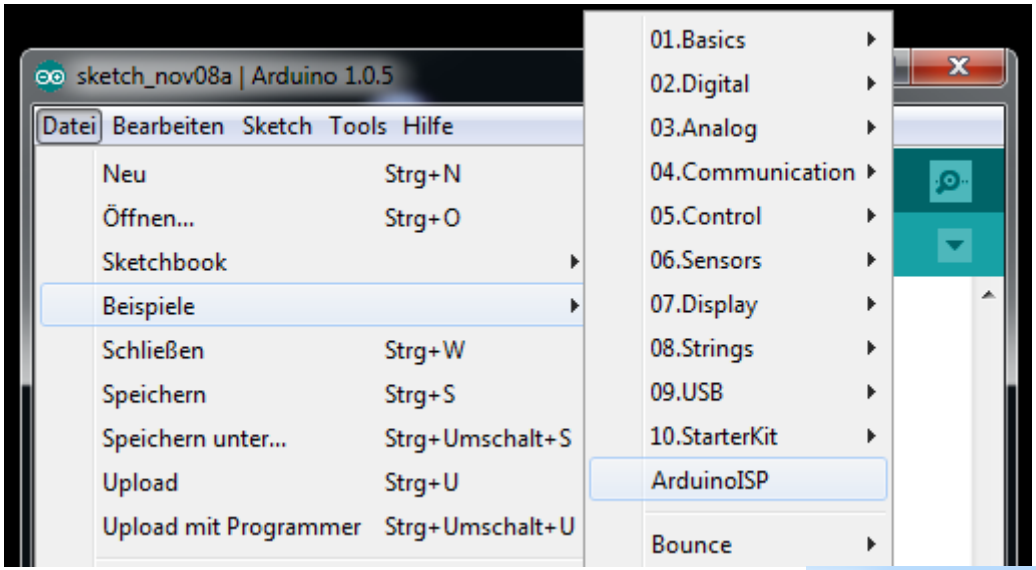
https://www.frag-duino.de/index.php/maker-faq/35-programmieren-eines-atmel-atmega-328p-mit-dem-arduino-uno-bootloader

Ich habe mir eine kleine Platine dazu gebaut.

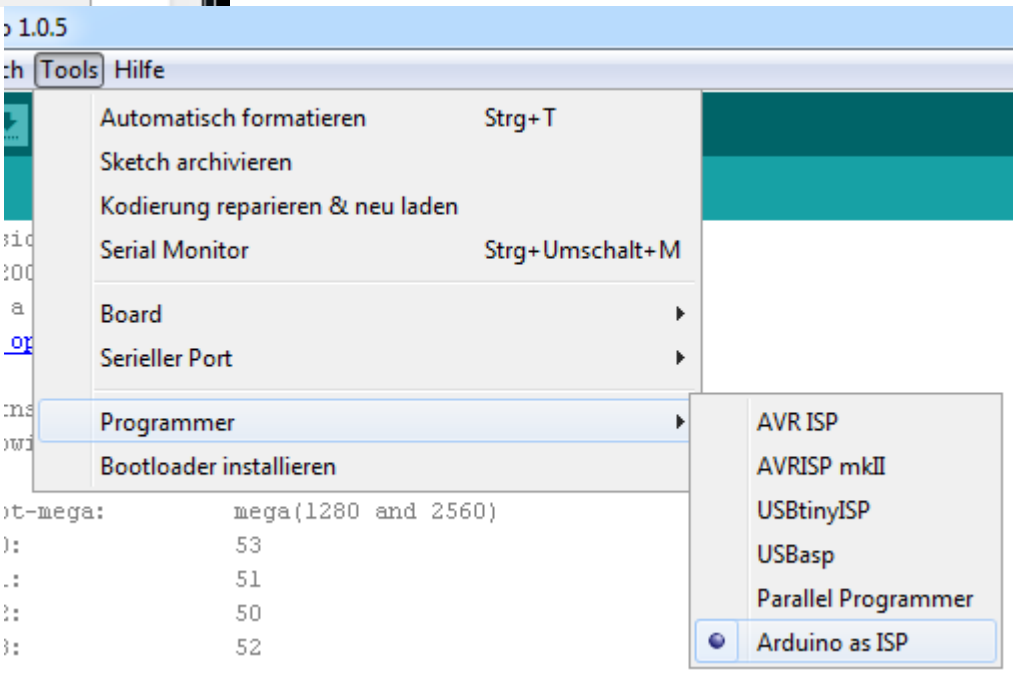
Dann unter Sketch und einem ARDUINO-UNO das Programm laden mit dem der Bootloader aufgebrannt wird. Sieh Anleitung:

Arduino like ATMega328p Breakout mit 2 freien USARTs. Build-in und Software USART



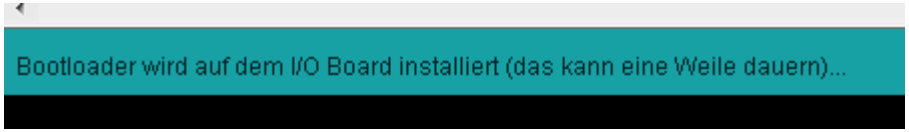


Als nächstes müsst ihr das Arduino Board mit dem Arduino ISP Beispiel flashen. Dazu öffnet ihr es über das Menü und Uploadet es:



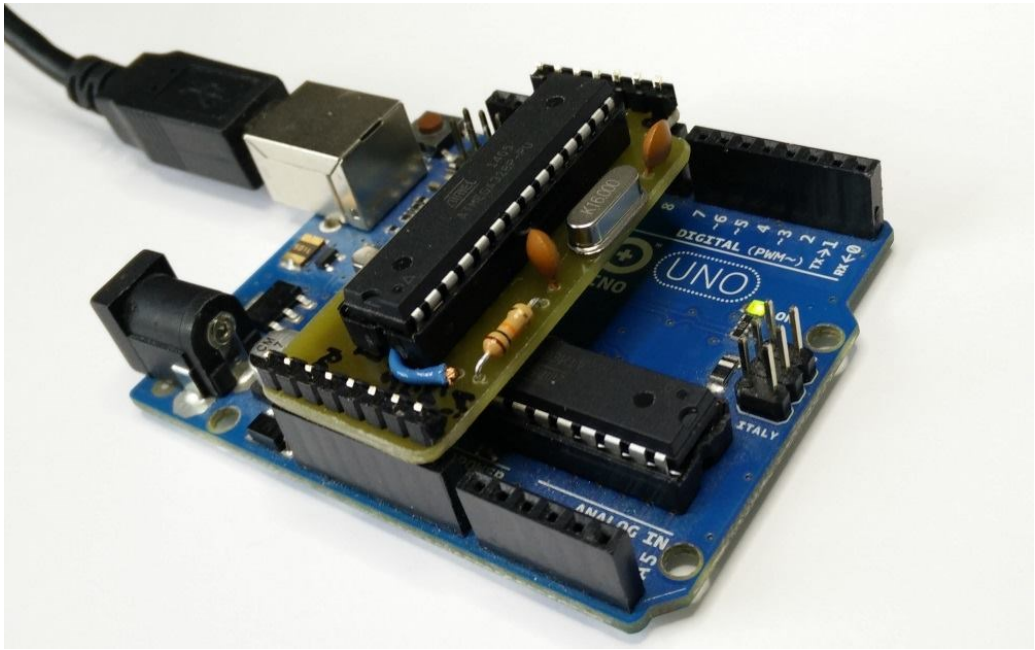
Dann stellt ihr den Programmer unter "Tools - Programmer - Arduino as ISP" ein

Als nächstes müsst ihr über "Tools - Bootloader installieren" klicken. Die Statusanzeige sollte dann so aussehen:



Extra gefertigte Bootloader flash Platine

und später dann so



Falls ihr folgende Fehlermeldung bekommt, habt ihr euch beim Aufbau der obigen Schaltung vertan oder habt irgendwo kalte Lötstellen:

Als nächstes empfehle ich, den neu programmierten Chip zu testen. Dazu könnt ihr ihn mit dem auf dem Arduino befindlichen Chip tauschen und z. B. mit dem Blink-Beispiel programmieren.

Software Artefakte Merkzettel (lose Sammlung)

```
struct bits {
  u8 b0:1;
  u8 b1:1;
  u8 b2:1;
  u8 b3:1;
  u8 b4:1;
  u8 b5:1;
  u8 b6:1;
  u8 b7:1;
} __attribute__((__packed__));
define SBIT_(port,pin)            ((*(volatile struct bits*)&port).b##pin)
#define    SBIT(x,y)  SBIT_(x,y)
//Optimization improvements  // remove useless R18/R19
#define    OPTR18 __asm__ volatile (""::);// it helps, but why ?
                                  // always inline function x:
#define AIL(x)   static x __attribute__ ((always_inline)); static x
// never inline function x:
#define NIL(x)   x __attribute__ ((noinline)); x
```

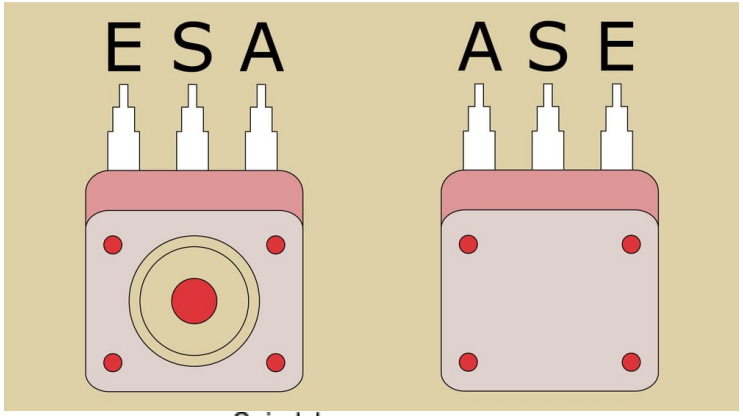
```
// volatile access (reject unwanted removing access):
#define vu8(x)  (*(volatile u8*)&(x))
#define vs8(x)  (*(volatile s8*)&(x))
#define vu16(x) (*(volatile u16*)&(x))
#define vs16(x) (*(volatile s16*)&(x))
#define vu32(x) (*(volatile u32*)&(x))
#define vs32(x) (*(volatile s32*)&(x))
```

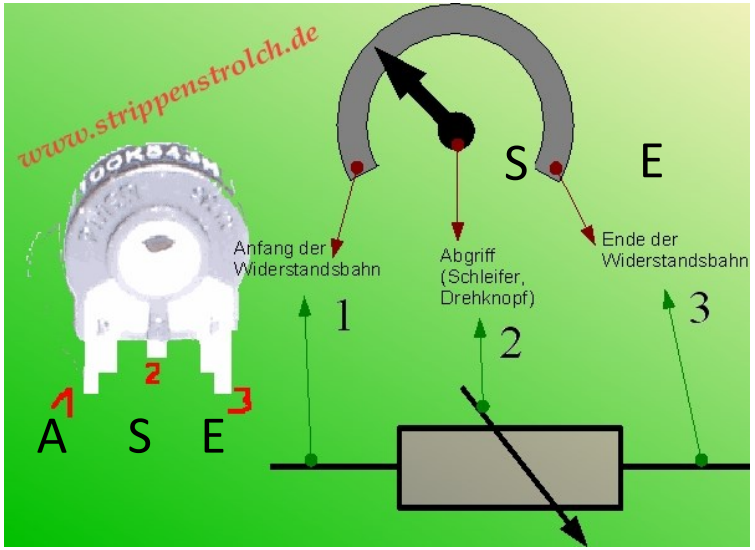
Praxis

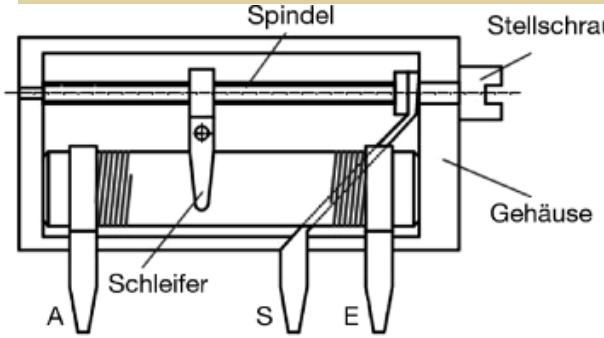
Aus der Praxis:

Hardware.

Potentiometer Drehsinn





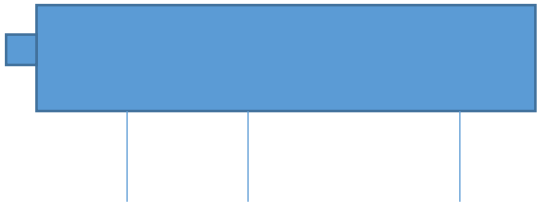


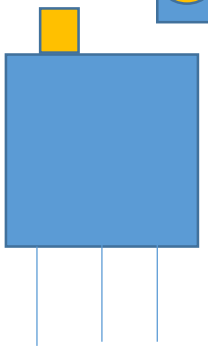
Drehung Linksrum

Drahtebene

1 2 3

Drehung Linksrum





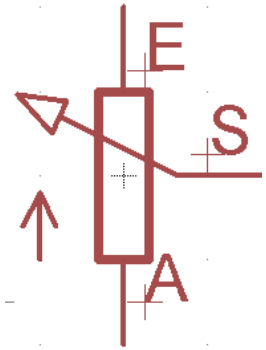
R > R <

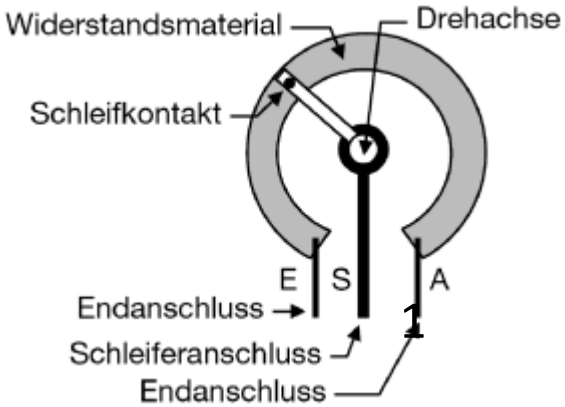
3 2 1

E S A

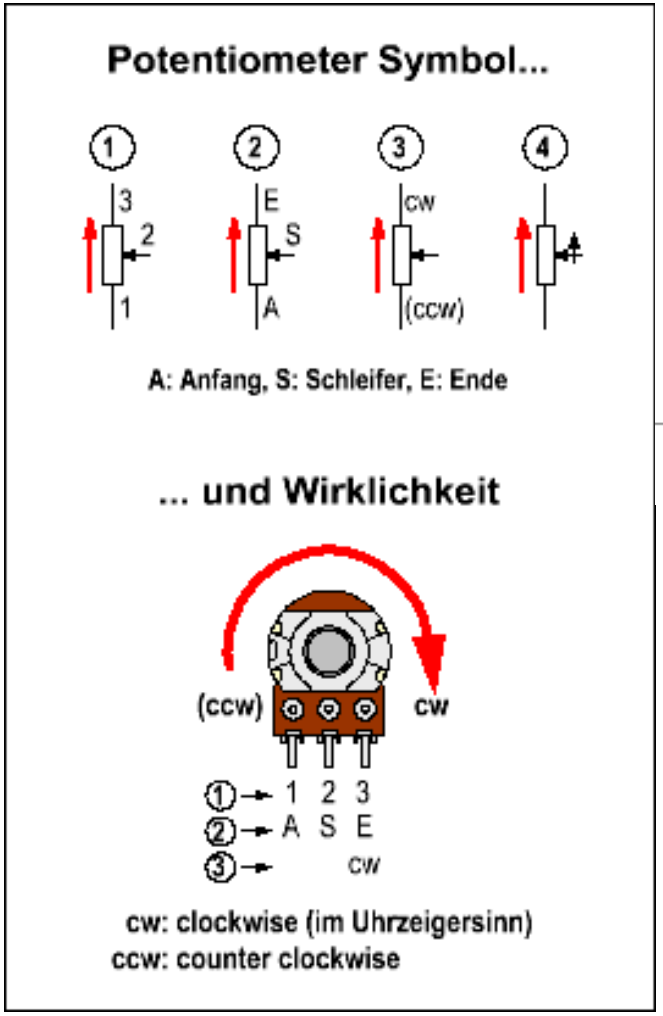
R < R >

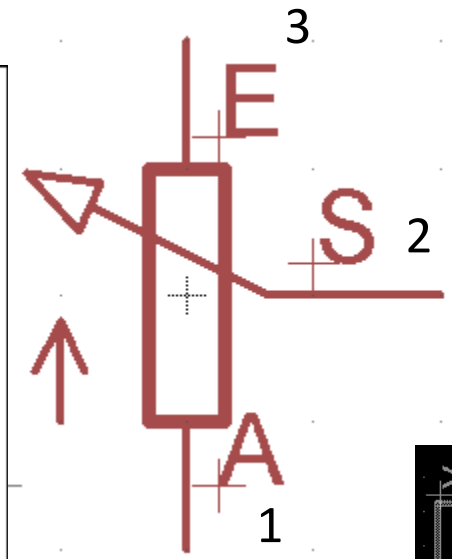
A S E

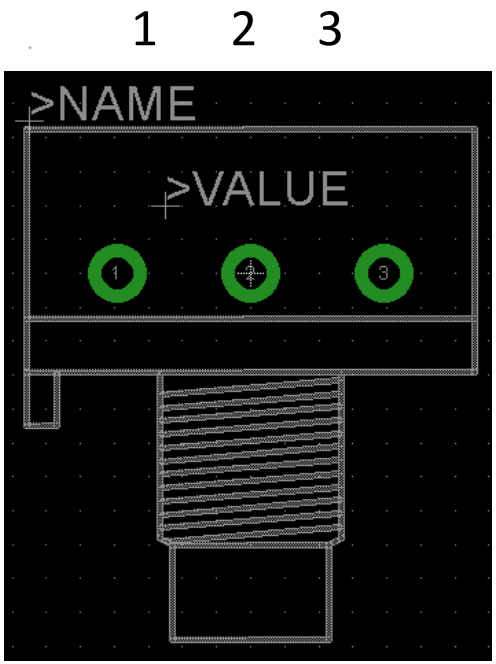




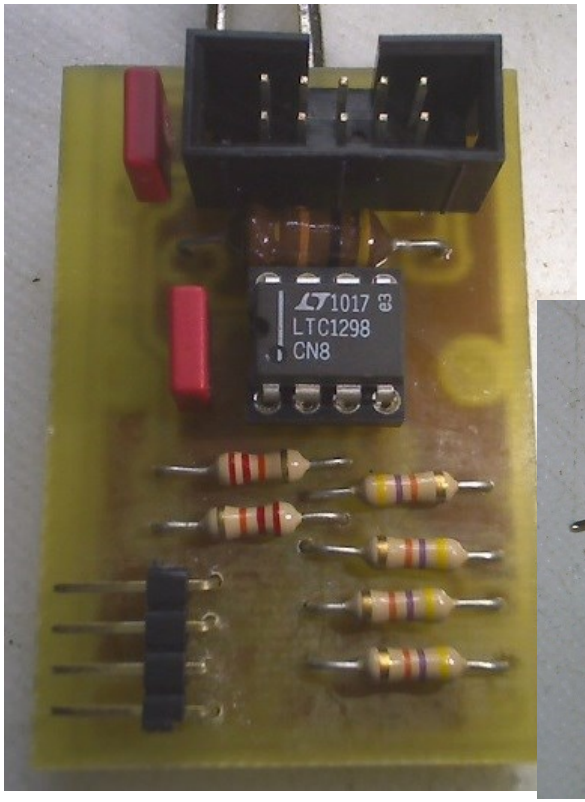
Das scheint Ansicht von hinten zu sein


Potentiometer Symbol...
1
2
3
4
3
2
1
E
S
A
cw
(ccw)
A: Anfang, S: Schleifer, E: Ende
... und Wirklichkeit
(ccw)
cw
1 2 3
A S E
cw
cw: clockwise (im Uhrzeigersinn)
ccw: counter clockwise

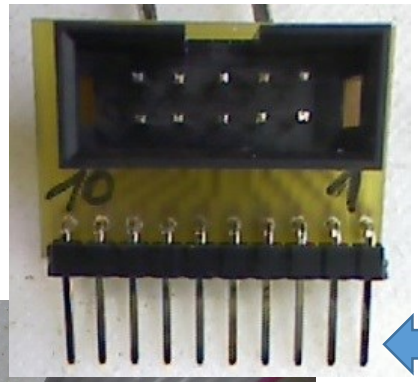

3
E
S
2
A
1

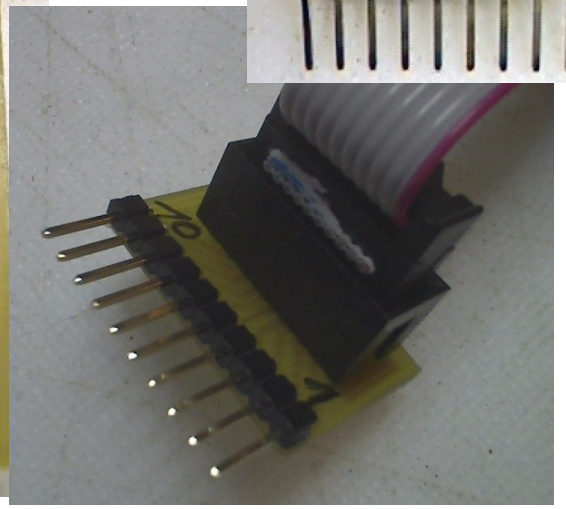

1 2 3
>NAME
>VALUE

Praktische Flachbandkabel-Anbindungen externer kleiner Platinen an ein Steckbrett:



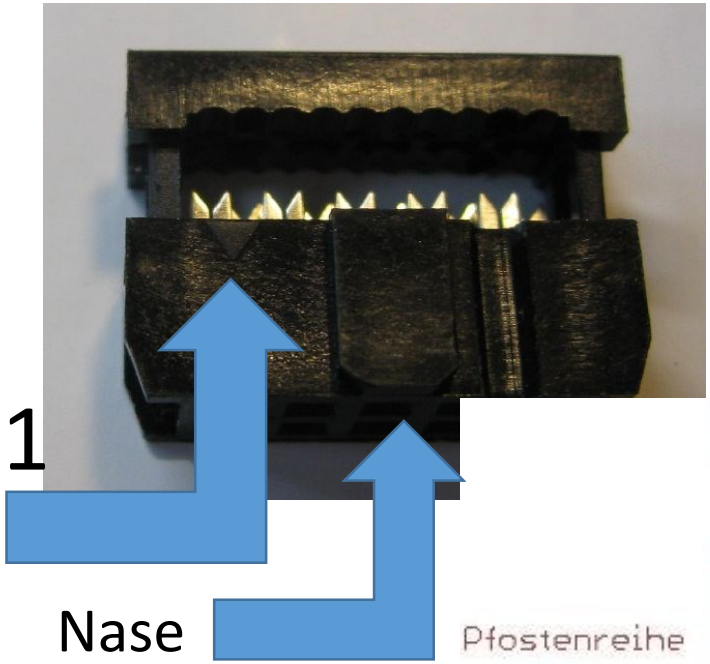
Immer auf die „**1**“ und GND=(9) +5V=(10) achten.

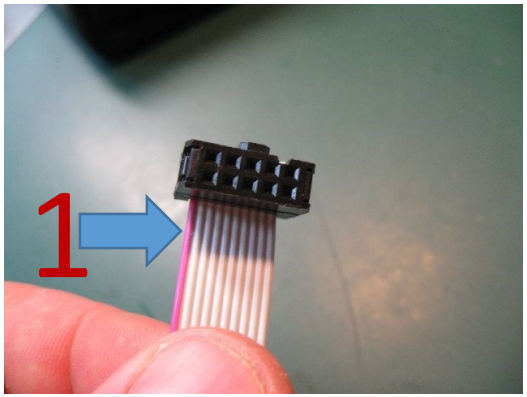


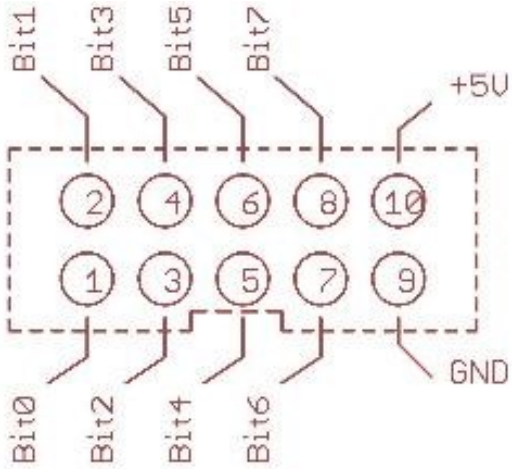


Dazu brauchen wir passende Flachbandkabel

Praktisches → Flachbandkabel



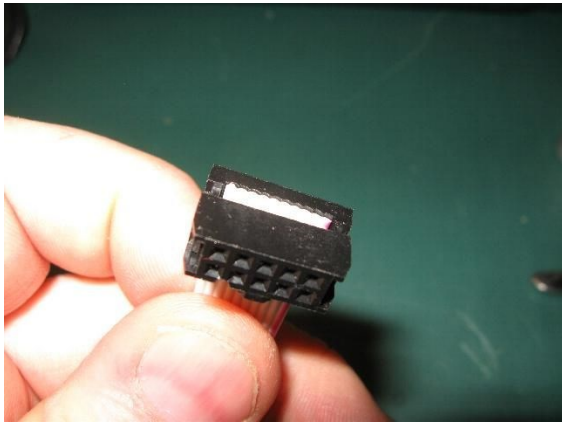




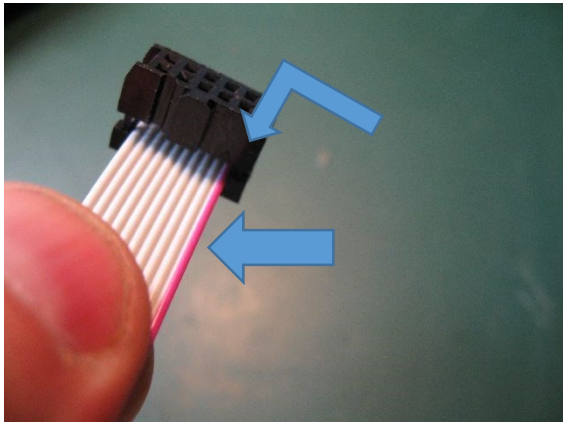
Die **Pfostenstecker Pin-Belegung**, die ich vor Jahren einmal festgelegt habe, enthält

Bit (0..7) und **GND →(9)** , **+5V →(10)**

Kabel **gerade** schneiden

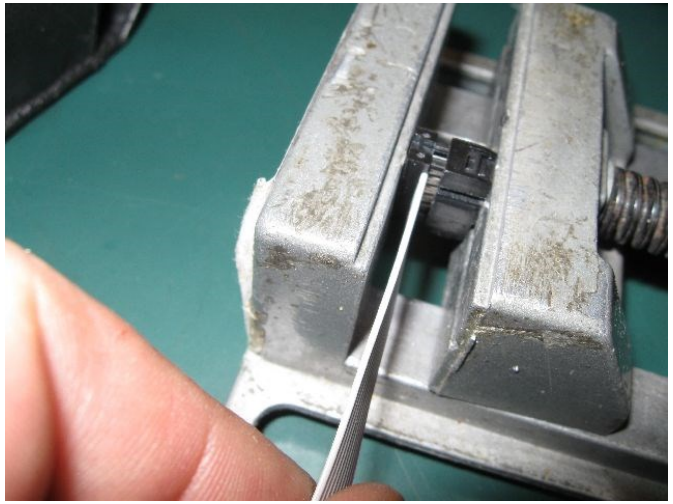


Ende des Kabels genau auf Kante.

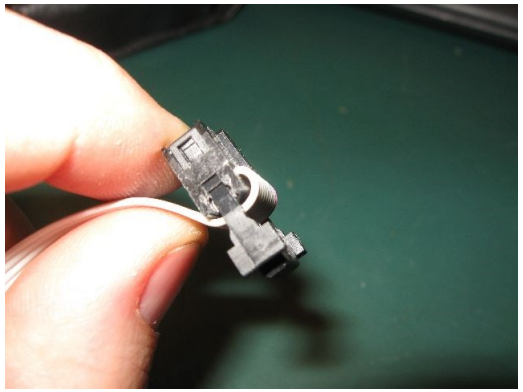


Rote Markierungen auf „1“
2*Prüfen! (wird gerne falsch gemacht)
Beliebige Seite auswählen, wo das Kabel sinnvoll sein soll.
Es ist **egal von welcher Seite** man das Flachbandkabel einführt.
Entscheidend ist, dass die **ROTE ADER → 1** auf der Seite der **Markierung** liegt.

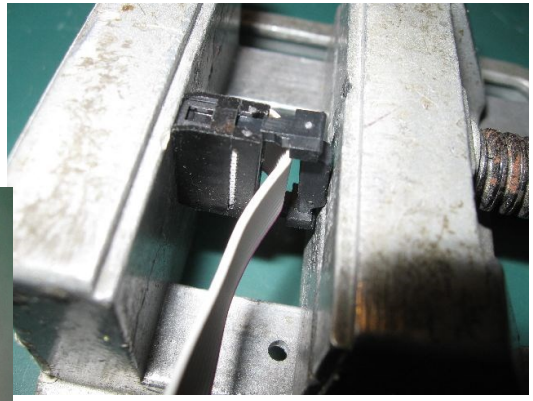
ROT →



In einem kleinen Schraubstock „**gerade**“ quetschen. Keine schiefe Zange!
Mit „**sanfter Gewalt**“ zudrehen!!

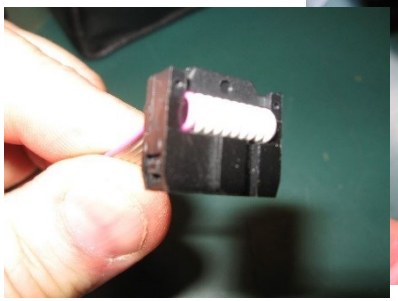


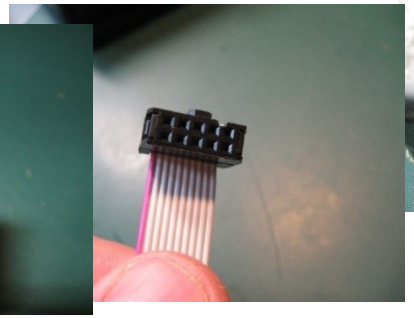
Umfalten und Zugentlastungsbügel aufstecken.



Zugentlastung sanft aufquetschen

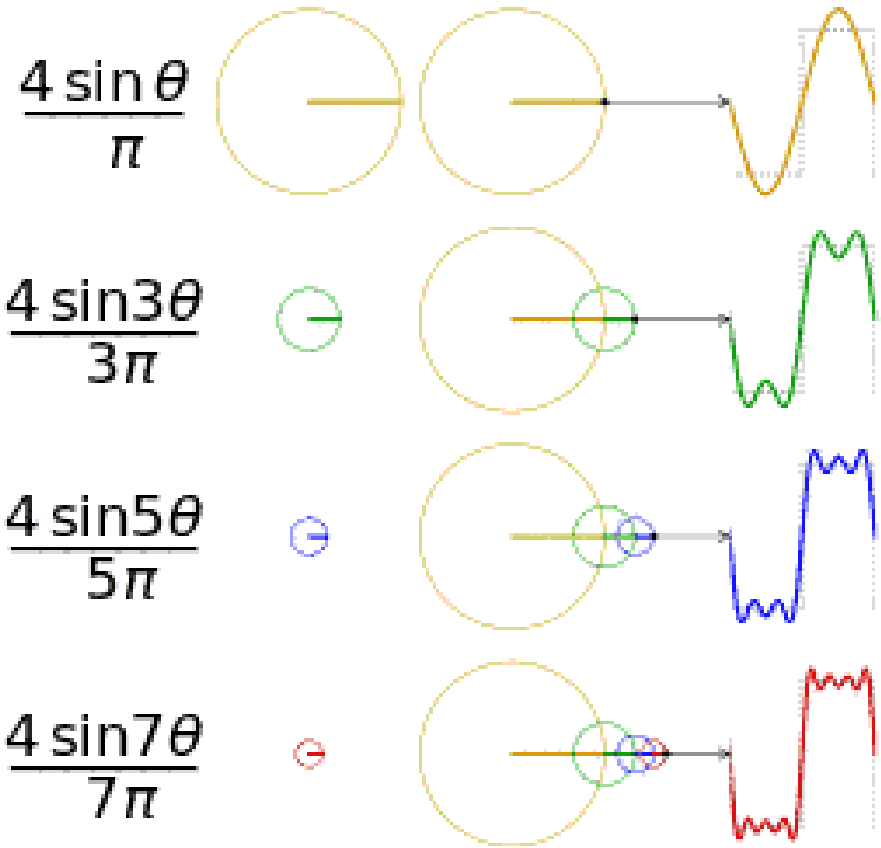
Fertig.





Schaltflanke, betrachtet mit einer Fourier Zerlegung.

→Schnelles digitales Schalten ist immer HF belastet.



Analogieschluss. Eine Rechteckschwingung (0an/aus) ist die Summe aller ungeradezahligen. Sinusschwingungen.. Geradezahlig wäre ein Dreieck.

Die Flanke wird immer steiler, …ergo
→ **schnelle** Schaltvorgänge produzieren hohe ***Oberwellenfrequenzen***.

..das war es vorerst… aber alle diese Techniken werden in einem C-Programm, speziell im Mikrocontroller angewendet
Doch jetzt haben wir eine Wiedererkennungswert.
Jetzt gibt es nur eines zu tun , lesen , erkennen, selber schreiben, üben.

Ab hier wird’s Chaotisch und ist eher eine Merkzettel
und ein Fundus
Allerlei was mir unter die Finger kommt,
Gedanken Anschubser, usw..

YouTubes:

Arduino LED Lauflicht - Knight Rider - mit Source Code
https://www.youtube.com/watch?v=Z_MvWpcwWao

BAS Video-Signal Synthese mit µC, Bsp: Tetris
(Programmiert von Armin Shaukat) http://youtu.be/f7NBdHZe4Mo

machine learning: Schulung µC: das sind die Vorlesungen zu dem Fach
https://www.youtube.com/channel/UCKHYFGVBavKAKY9ADFBcxSw

div. Links zur Hardware und Schematics

WICHTIG → Neuer Speicherplatz der **AVR Universal Bibliotheken** von Peter Fleury :
http://homepage.hispeed.ch/peterfleury/avr-software.html

Italienische Arduino Seite: Sensoren: http://tallerarduino.com/category/sensores-2/
ARDUINO Schaltplan: http://arduino.cc/de/uploads/Main/Arduino_Uno_Rev3-schematic.pdf

4x7 Segment Digit: http://electronics.stackexchange.com/questions/34815/using-4-digit-7-segment-led

Universal Robots: http://www.universal-robots.dk/DK/Presse/Multimedia/Products.aspx

CMOS: http://en.wikipedia.org/wiki/List_of_4000_series_integrated_circuits

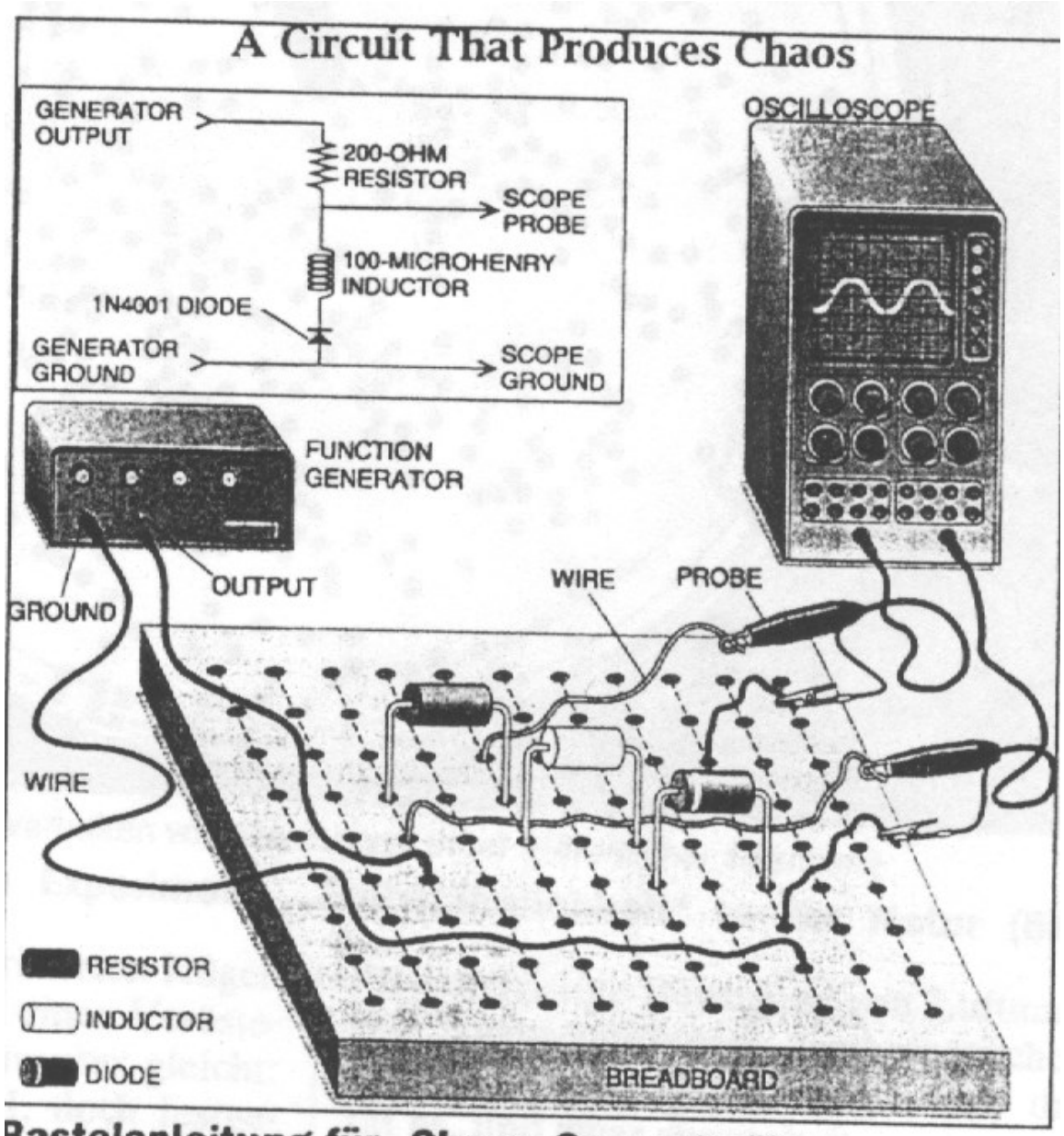
FTDI CHIP:
http://www.heise.de/make/meldung/Treiber-gegen-gefaelschte-Chips-FTDI-ruft-Fake-Killer-zurueck-2435079.htmlhttps:/www.lab-nation.com/

List of CMOS 40xx IC Circuits
ROS = Robot Operating System: ros.org
smartscoope
https://www.lab-nation.com/

Sehr viele schöne und lehrreiche Seiten gibt es im Netz. Stöbern lohnt sich:
Bsp:
https://www.heise.de/developer/artikel/Timer-Counter-und-Interrupts-3273309.html

Handmade.
Chaos Generator
Ideal für „echte“
Zufallszahlen.

Macht Lust zum
Experimentieren



Bastelanleitung für „Chaos-Generator“*
‚Chaos sehen entspannt“

Anhänger Prigogines für diese Ordnung den Namen „Anti-Chaos" gebrauchen.

★

Die Chaosmode animiert Naturforscher und Mathematiker zu interdisziplinären Ausschweifungen in die Zeitgeschichte, die Kunstbewertung, die Soziologie, die Ökonomie. Etwas, worin Renate Mayntz, Direktorin des Kölner Max-Planck-Instituts für Gesellschaftsforschung, „naturwissenschaftliche Hegemonialansprüche" sieht, wird von einem Chaosagitator wie Prigogine imperial geltend gemacht.

Wer in der Firma ist ein „Chaos-Killer"?

Seine Ordnungstheorie stützt sich insbesondere auf Computersimulationen, eine überaus künstliche anorganische Erzeugung von Oszillationen in der Petrischale und den Stoffwechsel von Hefepilzen. Verwegen weitet er sie aus auf Himmelsmechanik. Doch damit nicht genug, besorgt er sich daraus Perspektiven auch für die Beurteilung der US-Wirtschaft und des Befundes der GUS.

Zufolge solcher Allzuständigkeit ist er Berater der EG. Seinen Bühnenauftritt in Unna nutzte er schließlich dazu, „unserer Industrie" dringend eine „positivere Einstellung zum Chaos" anzuempfehlen.

So etwas ist Musik in den Ohren ungezählter Chaosjünger in den Chefetagen der Wirtschaft. Sie haben sich frei gemacht für die guten Seiten der nun zum neuen Paradigma ausgerufenen „schöpferischen Instabilität".

Das zu erlernen, gibt es teure Seminare und eine Fortbildungsliteratur, die viele, so auch den Leitartikler der *Münchner Medizinischen Wochenschrift*, so weit bringt, sogar noch „die Tohuwaboiker" zu „Bannerträgern der wahren Kreativität" zu ernennen.

Das Chaos wird von der Mehrzahl seiner gesellschaftlichen Animateure niemals deutlich genug erklärt. Genährt wird die Illusion, die Selbstorganisation des Neuen, der schöpferische Umschwung („Kreativität") laufe im Geschäft ähnlich dynamisch wie in ein paar modellhaften Experimenten im Labor.

Einer der prominentesten Verfechter des „Paradigmas der Selbstorganisation", der Biochemiker Bernd-Olaf Küppers, spricht zwar von der strengen Hierarchie, „die sich von den Zellen . . . bis hin zu den Individuen, Gruppen und Gesellschaften erstreckt".

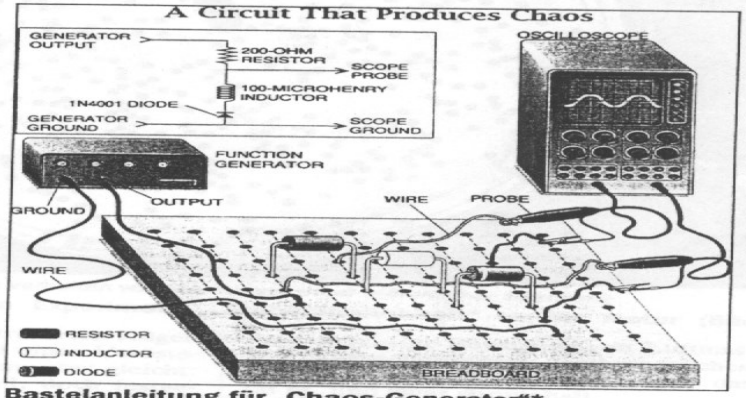
In den verbreiteten Chaosübungen für Betriebswirte hingegen müssen die noch an festen Hierarchien im Unternehmen Interessierten sich das neue Schmähwort „Chaos-Killer" anhängen lassen. Besondere Fragebogen sollen aufdecken helfen, in wem sich so ein Killer verbirgt.

★

Ein bißchen Chaos ist schnell inszeniert. Ein Schuß Sahne in den heißen Kaffee, und es ist ein paar Sekunden lang scheinbar da: Die spiralige Verteilung in der Tasse vermittelt eine Ahnung davon, was an der Sache determiniert und was chaotisch abläuft.

Die Vermengung der Moleküle vollzieht sich, auch vom Rauch über der Zigarette kennt man es ja, in einer voraussehbaren Struktur. Myriaden von Teilchen bewegen sich da so geordnet, als wären sie Mitwirkende eines Massenballetts. Vom unsichtbaren Zusammenstoß zwischen Teilchen rührt es her, daß die Formation sich erst vorhersehbar, dann unvorhersagbar abwandelt und verliert.

Einem wohl öffentlichen Interesse entsprechend hat die Zeitschrift *Scientific American* eine Bastelanleitung herausgegeben, nach der zu Hause mit etwas Aufwand elektrisches Chaos herzustellen ist. Für umgerechnet 300 Mark liefert der Handel das Nötige: Widerstände, Schaltbrett, Oszilloskop. Damit kann jeder durch spielerisches Herumstöpseln Rückkopplungen anfachen, die jählings aus den regelmäßigen Spannungsamplituden im Fenster des Oszilloskops ein erratisches Furioso machen. Das zu verfolgen sei „entspannend", lobt die Zeitschrift. Da sehe man die „Schönheit der Natur".



Bastelanleitung für „Chaos-Generator"*

„Chaos sehen entspannt"

In München haben sich an der TU junge Physiker, Mathematiker und Informatiker zu einem Chaosverein verbunden. Wie sie sagen, soll der eine „Lobby für das Chaos" bilden und möglichst vielen Zeitgenossen möglichst einfach nahebringen, wie sehr es sie angeht. Der Klub wird von der Industrie gefördert. Das Interesse am Chaos ist mittlerweile steuerbegünstigt.

Einmal im Monat üben die jungen Experten eine Art Volksaufklärung im Schwabinger „Cafe Ignaz" aus. Sozialarbeiter, Notärzte, Aktienspekulanten und Besitzer von Heimcomputern suchen da Orientierung in allen erdenklichen Fragen, die sie verunsichern. Es geht um Asylantenzahlen, Herzflattern, Börsenkurse und sonstig Instabiles. Faschistische Schwärmer möchten hören, wie sich, naturgesetzlich gesehen, wieder Ordnung schaffen ließe im Land.

Die jungen Leute vom Verein sind zu allen lieb. Stolz nennen sie sich „Chaoten". Daß dies vor kurzem noch ausschließlich als Bezeichnung für Typen galt, die ihre Anschauungen mit Hilfe von Molotowcocktails und Pflastersteinen geltend machen, berührt die neuen Chaosfans nicht mehr.

Chaos zu zeigen ist ihre Spezialität. Einer von ihnen tut das mit so viel unterhalterischer Bravour, daß ihn ein Software-Produzent vor einem Firmenempfang im Münchner Zwei-Sterne-Restaurant „Tantris" hat auftreten lassen – Chaos zum Horsd'œuvre.

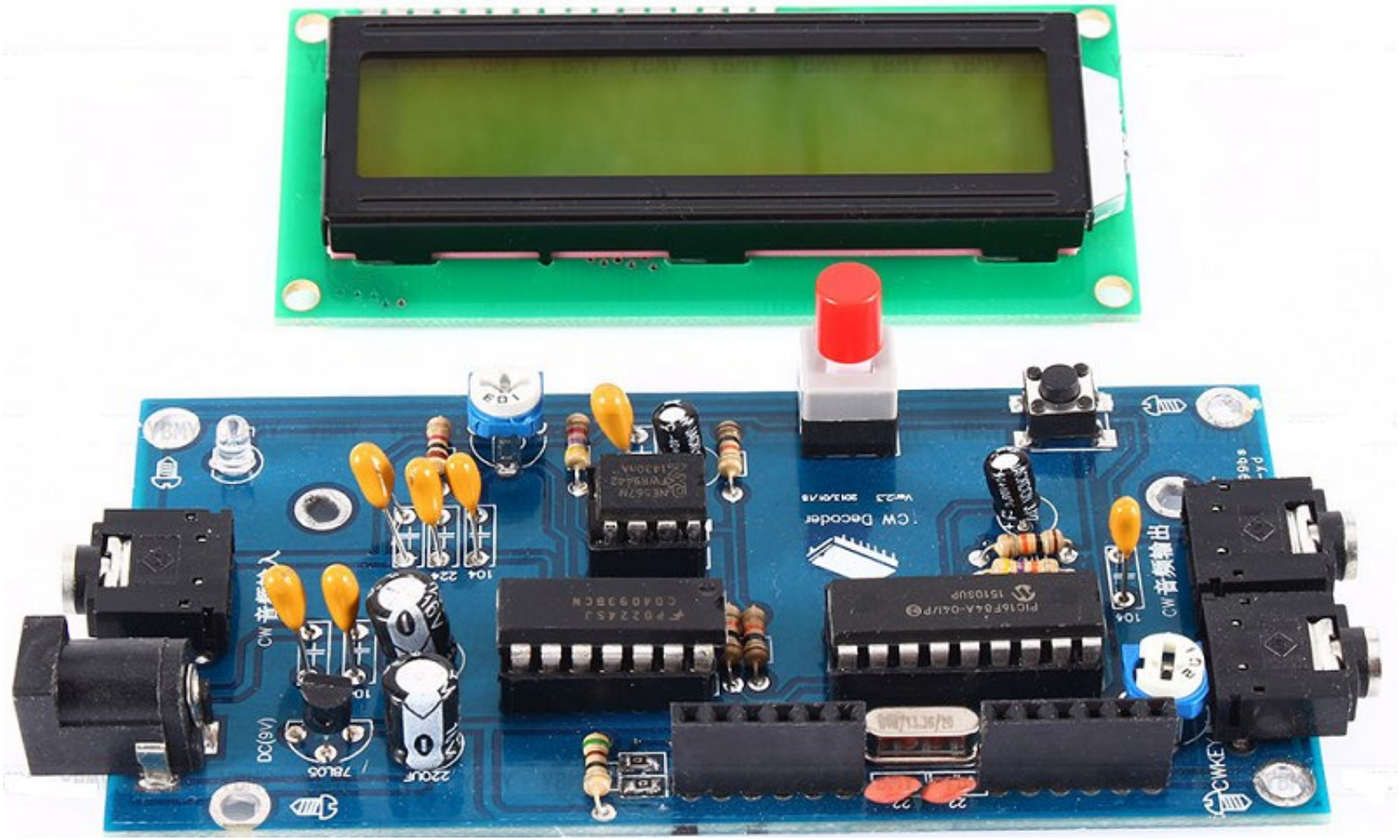
Wo immer der Chaosklub seine einführenden Experimente zur Show macht, ob in regelmäßigen Symposien für jedermann oder in den Physiksälen von Fachschulen: Es herrscht Andrang wie bei einer Filmpremiere.

Mit Pendeln fängt es immer an. Pendel, scherzte James Gleick, der Chaosexperte der *New York Times*, die seien so etwas wie die Labormäuse der Chaosforschung. Das liegt daran, daß sich mit nichts so direkt und einfach vorführen läßt, wieviel Unberechenbares mit Schwingungen passieren kann.

Unglaublich viel in der Natur, ihren Systemen und Zellen ereignet sich in Form einander überlagernder Schwingungen, die kaum zu analysieren sind. Zu Pendeln gibt es eher Zugang, ob sie nun in Gestalt kinetischen Spielzeugs auf dem Schreibtisch stehen, verbunden mit subtilsten Meßgeräten in Forschungsstätten oder als Kinderschaukel

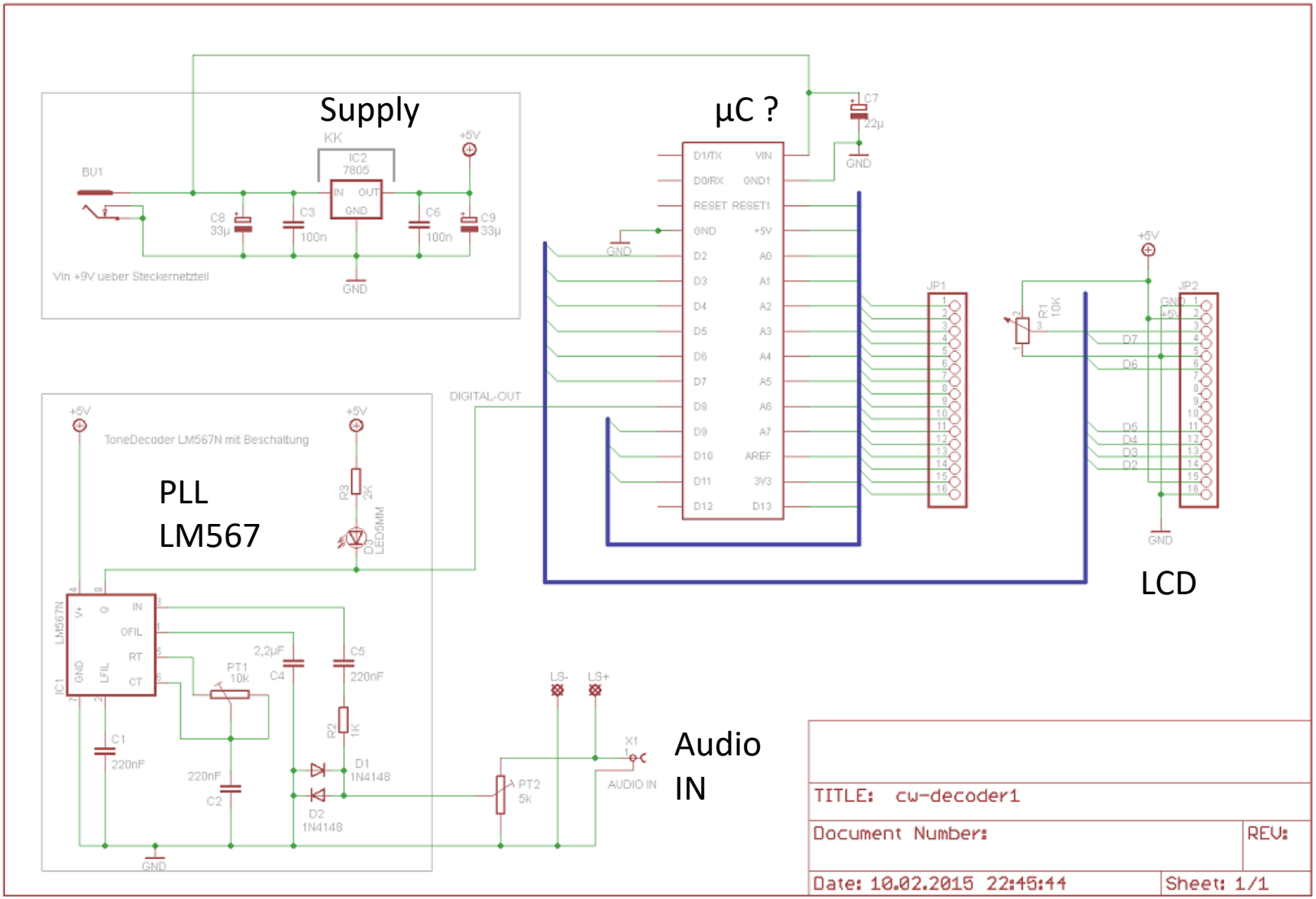
* Aus dem *Scientific American*.

Morse Dekoder aus der Bucht. ~15€
Mit PIC16F64A und NAND Schmitt Trigger CD4039.
Der Analogteil könnte besser sein, genauso wie der Frankenwein.



Ext. Projekt Morsedekoder: http://gemander.org/2015/03/31/cw-decoder-nach-wb7fhc/

http://gemander.org/wp-content/uploads/2015/03/CW-DecoderSCH.png



Morsen mit dem Controller

Projekt Idee:
Senden (Tx)
Timings mit Interrupt und Flags für das Senden.
Einstellbare Geschwindigkeit.

Empfangen (Rx) ist viel schwieriger.
Dynamische Zeit-(Timer) Anpassung.
Ereignis - Level Detektor.
Fehlerkorrektur usw.

International Morse Code

- 1 dash = 3 dots.
- The space between parts of the same letter = 1 dot.
- The space between letters = 3 dots.
- The space between words = 7 dots.

A	• —	V	• • • —
B	— • • •	W	• — —
C	— • — •	X	— • • —
D	— • •	Y	— • — —
E	•	Z	— — • •
F	• • — •	.	• — • — • —
G	— — •	,	— — • • — —
H	• • • •	?	• • — — • •
I	• •	/	— • • — •
J	• — — —	@	• — — • — •
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —
U	• • —		

Internationaler Morsecode:

Lateinische Zeichen:

A	·—		N	—·	
B	—···		O	———	
C	—·—·		P	·——·	
D	—··		Q	——·—	
E	·		R	·—·	
F	··—·		S	···	
G	——·		T	—	
H	····		U	··—	
I	··		V	···—	
J	·———		W	·——	
K	—·—		X	—··—	
L	·—··		Y	—·——	
M	——		Z	——··	

Ziffern:

0	—————
1	·—————
2	··———
3	···——
4	····—
5	·····
6	—····
7	——···
8	———··
9	————·

Sonderzeichen:

À, Å	·——·—
Ä	·—·—
È	·—··—
É	··—··
Ö	———
Ü	··——
ß	···——··
CH	——
Ñ	——·——

Satzzeichen:

. [AAA]	·—·—·—
, [MIM]	——··——
:	———···
;	—·—·—·
? [IMI]	··——··
-	—····—
_	··——·—
(	—·——·
)	—·——·—
'	·————·
=	—···—
+	·—·—·
/	—··—·
@ [AC]	·——·—·

Signale:

KA	(Spruchanfang)	—·—·—
BT	(Pause)	—···—
AR	(Spruchende)	·—·—·
VE	(verstanden)	···—·
SK	(Verkehrsende)	···—·—
SOS	(internationaler Notruf)	···——— ···
HH	(Fehler; Wiederholung ab letztem vollständigen Wort)	····

Kyrillische Zeichen:

А	·—		Й	·———		У	··—	
Б	—···		К	—·—		Ф	··—·	
В	·——		Л	·—··		Х	····	
Г	——·		М	——		Ц	—·—·	
Д	—··		Н	—·		Ч	———·	
Е	·		О	———		Ш	————	
Ё	·		П	·——·		Ь	—··—	
Ж	···—		Р	·—·		Э	··—··	
З	——··		С	···		Ю	··——	
И	··		Т	—		Я	·—·—	

Morsecodierung:

TRICK: Möglichst nur EIN Byte/Zeichen.

Methode: ‚.‘ = 0, ‚-‘ = 1; Wenn eine „1“ übrig bleibt ist der Code fertig.
BSP: „F“ = . . - . → Byte 00010100 (von rechts kommend parsen)

Beispiel: Morse „F“ Code ist von rechts nach links zu interpretieren

```
u8Code = 0b00010100;      // = „F“  Lade hier das zu sendende Zeichen
                          // 0100 von rechts  = Morsecode von links „..-.“
                          // Sprich: 0010 = ..-.

do {
TxFlag = (u8Code & 0x01);    // Prüfe letzte Stelle ob Punkt=0, oder Strich=1
      → Sende TxFlag;        // wie immer das realisiert wird.
u8Code = u8Code >> 1 ;   // Shift alles eine Stelle nach rechts schieben
    }while (u8Code  >  1 );
```

Schwierig. Das Ganze aber dann in einem Interrupt erledigen (Flag-Steuerung

Eine Headerdatei mit den Codes erstellen.

```
static const uint8_t PROGMEM MCA_8BIT_SIMPELMORSE[]= \
// Ascii 48 - 57  ->  0123456789    für Zahlen! “ → Pos =  Ascii Zahl - 48
 {
 0b00111111          //0          -----
, 0b00111110         //1          .----
, 0b00111100         //2          ..---
, 0b00111000         //3          ...--
, 0b00110000         //4          ....-
, 0b00100000         //5 .....
, 0b00100001         //6 -....
, 0b00100011         //6          --...
, 0b00100111         //8          ---..
, 0b00101111         //9          ----.
```

„A“ = ASCII 65, also Position 10
im Header: POS = ASCII - 55

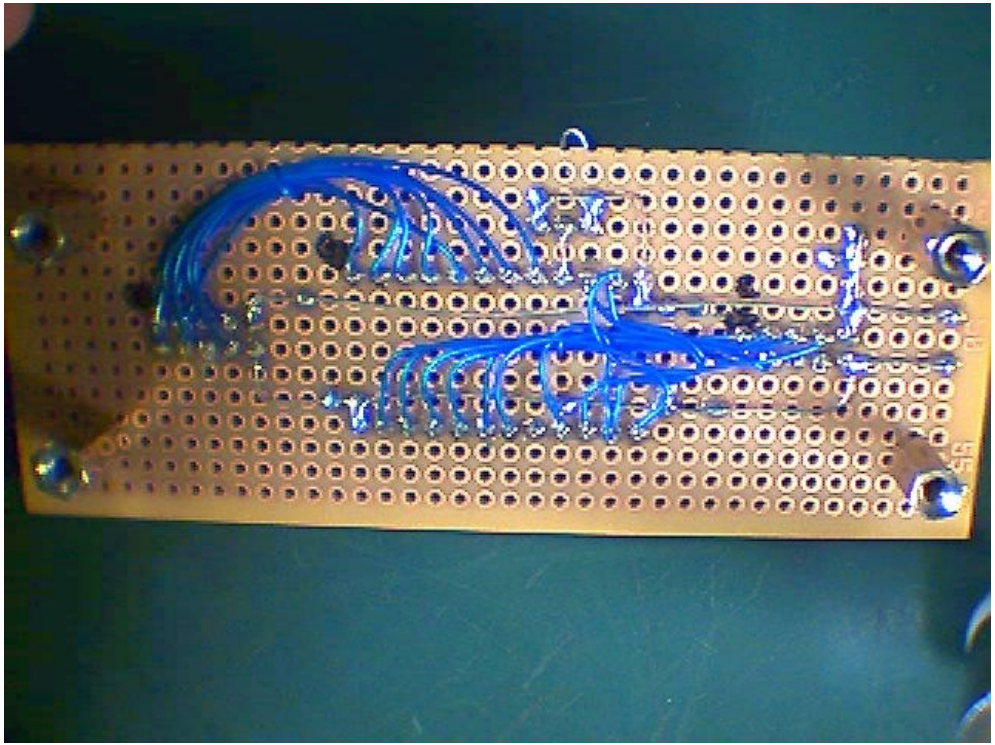
```
//10
, 0b00000110      //A .-
, 0b00010001      //B -...
, 0b00010101      //C -.-.
, 0b00001001      //D -..
, 0b00000010      //E .
, 0b00010100      //F ..-.
, 0b00001011      //G --.
, 0b00010000      //H ....
, 0b00000100      //I          ..
, 0b00011110      //J          .---
```

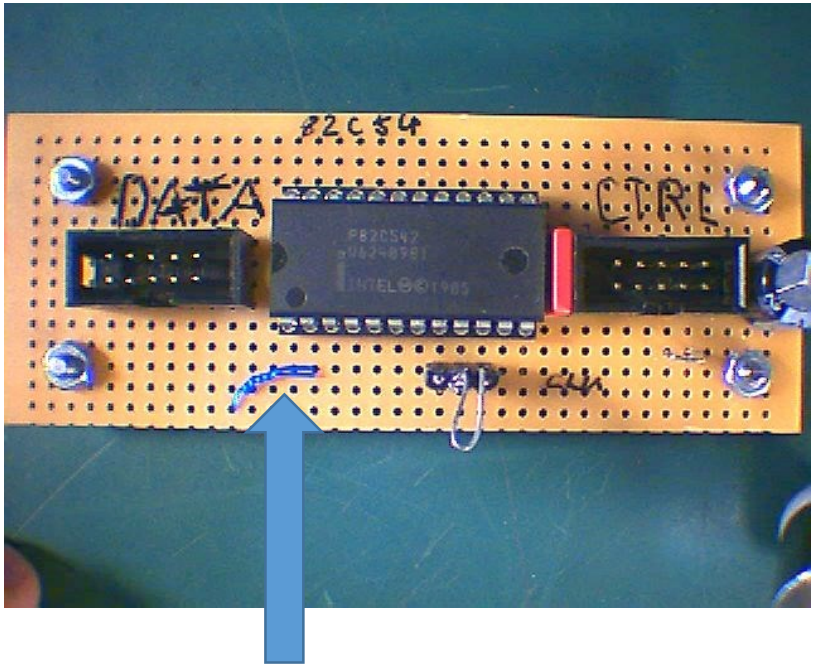
```
//20
, 0b00001101      //K        -.-
, 0b00010010      //L        .-..
, 0b00000111      //M        --
, 0b00000101      //N -.
, 0b00001111      //O        ---
, 0b00010110      //P        .--.
, 0b00011011      //Q        --.-
, 0b00001010      //R        .-.
, 0b00001000      //S        ...
, 0b00000011      //T        -
//30
, 0b00001100      //U        ..-
, 0b00011000      //V        ...-
, 0b00001110      //W        .--
, 0b00011001      //X        -..-
, 0b00011101      //Y        -.--
, 0b00010011      //Z --..
```

```
// SPACE
//36
, 0b00000001        //SPACE = WORDSPACE"1"
{keine Zeichen}
, 0b00011010        //Ä .-.-
, 0b00010111        //Ö ---.
, 0b00011100        //Ü ..--..
, 0b10011000        //ß ...--..
, 0b00011111        //CH ----

 };
```

Sehr zu empfehlen.
Aufbau von Prototyp Platinen

Manuelle Verdrahtung **mit Wire-Wrap** Drähten erlaubt SAUBERE und wirklich dauerhaft wie professionelle Prototypenerstellung.
Dieser **schrumpft beim Löten nicht zurück**. !!
Kann gebündelt werden. (fast wie Bus-Leitungen)
Liegt sauber an und wird gelegentlich an der Oberseite verknotetet.

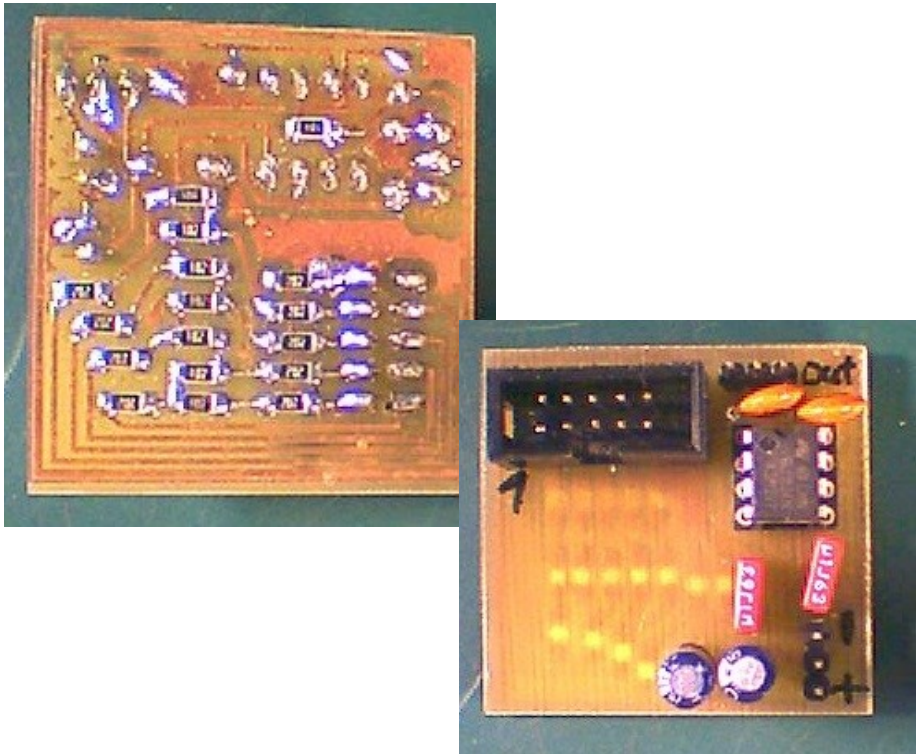


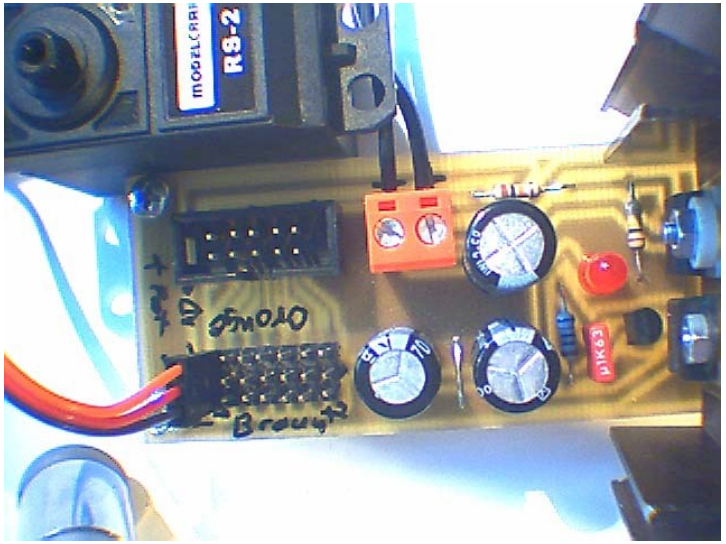


…besser gleich Platine machen

Ab einem gewissen Aufwand z.B. mit SMD Teilen lohnt sich eine Platine unbedingt.
Mit Eagle cadsoft.de steht ein gutes Werkzeug zur Verfügung
Vorteil: geht genau so schnell, nur sauberer.
Und man hat Spass bei der Entwicklung

Hier z.B. ein R2R Netzwerk mit SMD Widerständen.



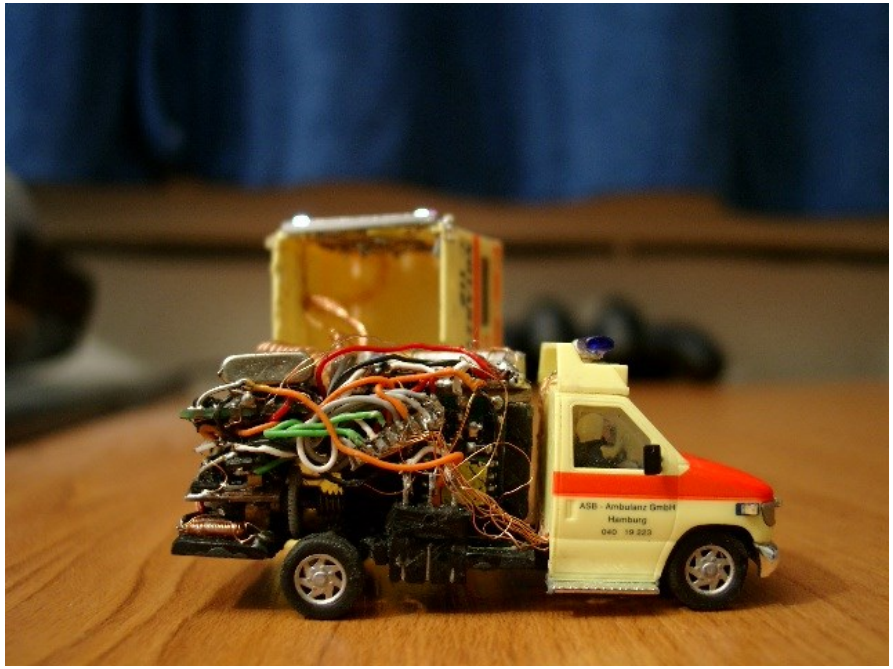


Oder Multi-Servo Ansteuerung

GUTHUB.com hat eine enorme Sammlung Projekte, Quellcodes

Sehr schicke Ziffernanzeige mit Morving
DOT MATRIX RG → Siehe Youtube:
https://www.youtube.com/watch?v=uxMf4LmE7n4&t=764s
Quellcode
https://github.com/dragondaud/myClock
+ https://github.com/hallard/WeMos-Matrix-shield

Projekte & Ideen



Mustergültiges Beispiel:
Der Aufwand scheint übertrieben zu sein, doch ein stringent sauberer, wie übersichtlicher Aufbau, und eine präzise Dokumentation sind der Garant für ein erfolgreiches Projekt.

STICHWÖRTER für Projekte (blose Sammlung)

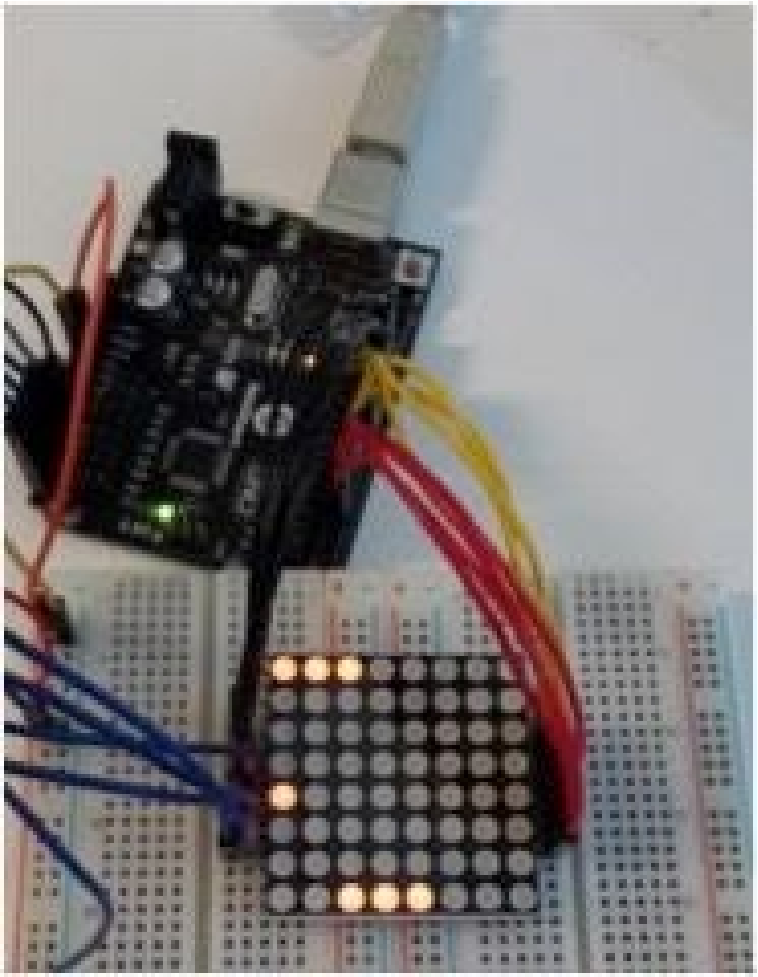
Selbständige Softwareexperimente. Z.B.

* Interrupt Anwendung: LED Ausgabe, blinken mit ^=(Exor) und Timer-FLAG gesteuert.
* Software Taster Entprellen , mit Flags und mehrfachen PINx-Reading Übereinstimmung.
* R2R – Funktionsgenerator,
* PWM für Servo (1mm..2ms mit ~50Hz)
* ADC alg., „automatische PGA?“ und mit Ausgabe an Display
* ADC mit Analogcomparator realisiert? (time-E-Funktion des Integrators)
* DAC-COMPARE… sucsessive aproximatzion:
* Zufalls Musik mit echten Tönen → Timer Synthesizer.
* Morse Sender – Empfänger
* Daten-TON Umsetzer (DTMV Dual Tone Multi Frequency oder MFV-Wähltöne)

http://de.wikipedia.org/wiki/Mehrfrequenzwahlverfahren

* https://www.instructables.com/id/8-x-8-LED-Pong-with-Arduino/

PONG am 8x8 Display



Zwei Potis,
oder besser:
Rotary encoder,
Array Management,
Vektor Generierung,

Gar nicht so ohne…

https://www.youtube.com/watch?v=lvAlKH5SzYk
Oder https://www.youtube.com/watch?v=Mf6buDkbyAQ

Funk Thermometer Protokoll

Gefunden auf :
http://thomaspfeifer.net/funk_temperatur_sensor_protokoll.htm

Die Bedeutung der übermittelten Bits wurde durch mehrfaches probieren und genaues hinsehen herausbekommen. Die Daten werden auf 433Mhz mit AM-Modulation übertragen. Ein langer Impuls entspricht einer "0", ein kurzer Impuls einer "1". Die ersten 16 Bits enthalten eine ID, welche beim Reset zufällig gewählt wird. Die restlichen Bits enthalten die Temperatur. Des weiteren ist anscheinend noch eine Checksumme vorhanden

ID	?	A+5	B	C	A+5	B	?	AB,C°C
0000101000001110	0001	0111	0101	0000	0111	0101	0001	25,0
0000101000001110	0000	0111	0100	0101	0111	0100	0011	24,5
0000101000001110	0001	0111	0011	1001	0111	0011	0110	23,9
0000101000001110	0000	0111	0011	0111	0111	0011	0011	23,7
0000101000001110	0001	0111	0011	0101	0111	0011	0010	23,5
0000101000001110	0000	0111	0100	0101	0111	0100	0011	24,5
0000101000001110	0001	0111	0011	0011	0111	0011	0000	23,3
0000101000001110	0000	1000	0001	1001	1000	0001	0011	31,9
RESET								
0000101000001111	1101	0110	1001	0111	0110	1001	1011	19,7
RESET								
0000101000000111	0000	0110	0000	0011	0110	0000	0000	10,3
0000101000000111	0000	0110	0000	0110	0110	0000	0011	10,6
0000101000000111	0000	0110	0001	0001	0110	0001	0000	11,1
0000101000000111	0001	0110	0001	0101	0110	0001	0101	11,5
0000101000000111	0001	0110	0010	0000	0110	0010	0010	12,0
0000101000000111	0001	0111	0010	0100	0111	0010	1000	22,4
0000101000000111	0001	0111	0001	0001	0111	0001	0011	21,1
0000101000000111	0000	0111	0000	0010	0111	0000	0001	20,2
RESET								
0000101000001100	0100	0111	0000	0111	0111	0000	1111	20,7
0000101000001100	0101	0111	0000	1001	0111	0000	0010	20,9
RESET								

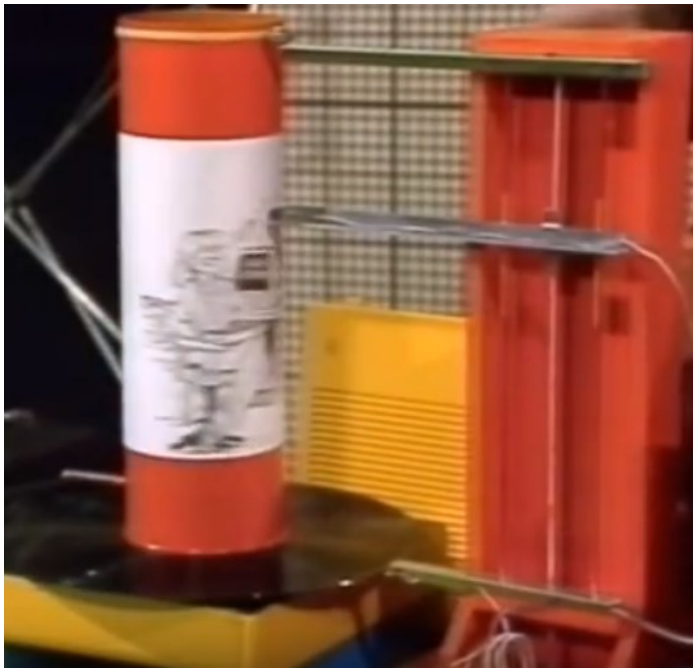
Reales Projekt.
Mit µControllern kann man Maschinen bauen.
Und gutes Geld verdienen

Hier ein Praxisbeispiel aus der Forschung. Ein Probenhalter, mit elastischen Boden und 6 Kammern, wird Sinusförmig gedehnt. Der Antrieb ist ein kräftiger Titangetriebe **Servo** ….kurzes Youtube: https://www.youtube.com/watch?v=naDMfTPF6WI

Coole Projekte

Ich sammle bei Gelegenheit hier Links & Ideen

http://www.instructables.com/



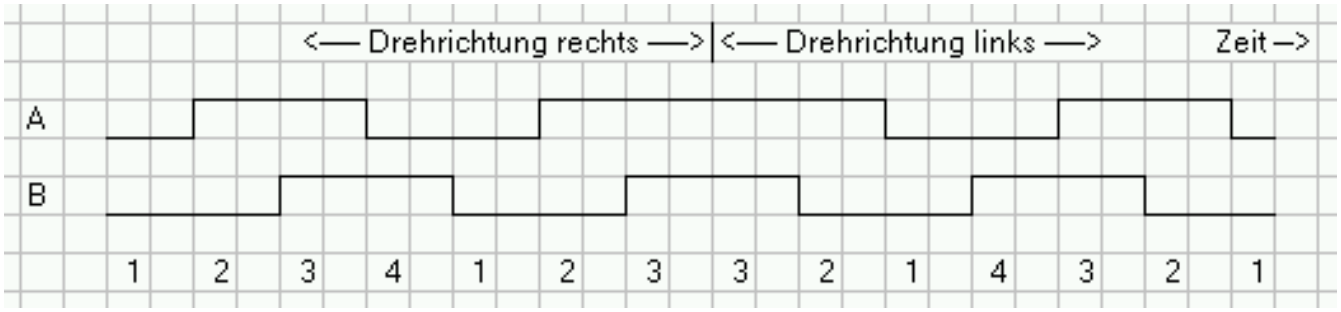
Projektidee:
Trommelscanner mit Einpunkt Sensor →
Realität: Wetterfax für die Seefahrt.
https://www.youtube.com/watch?v=C_Qn5X6jDOk time: 14:24

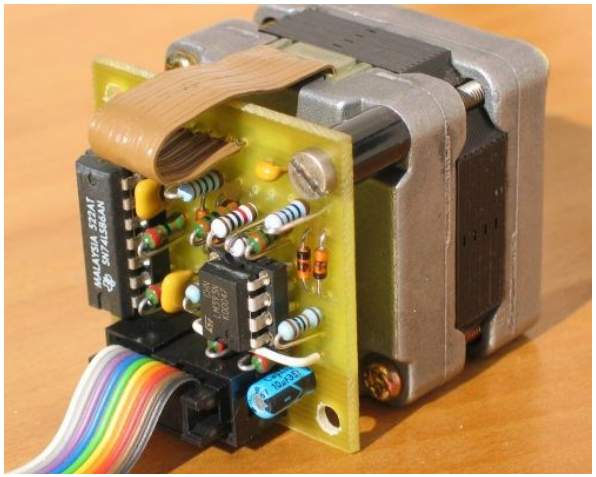
Mit alten CD Laufwerken: https://www.youtube.com/watch?v=PD-tPfSrG2g
(Musik im Hintergrund ist Schrott)

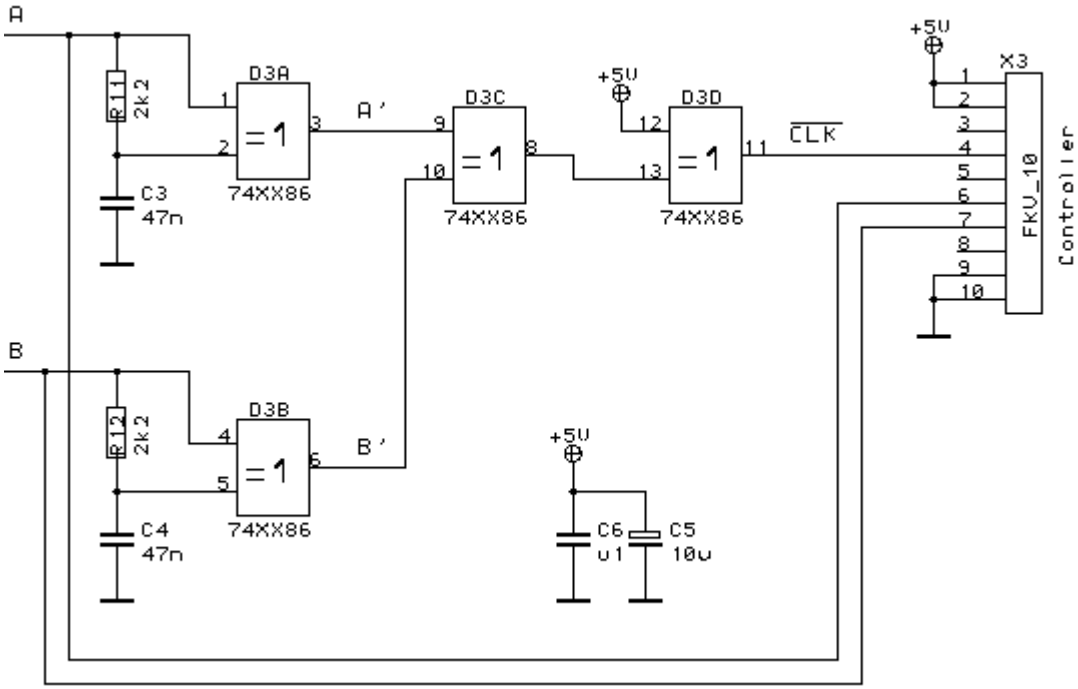
PID Regelung:
Siehe: http://homepages.uni-regensburg.de/~erc24492/PID-Regler/PID-Regler_mit_uC.html

Linearantrieb: http://homepages.uni-regensburg.de/~erc24492/Labor-Projekte/Labor-Projekte.html

Schrittmotor als verschleißfreiher, inkrementaler Drehgeber





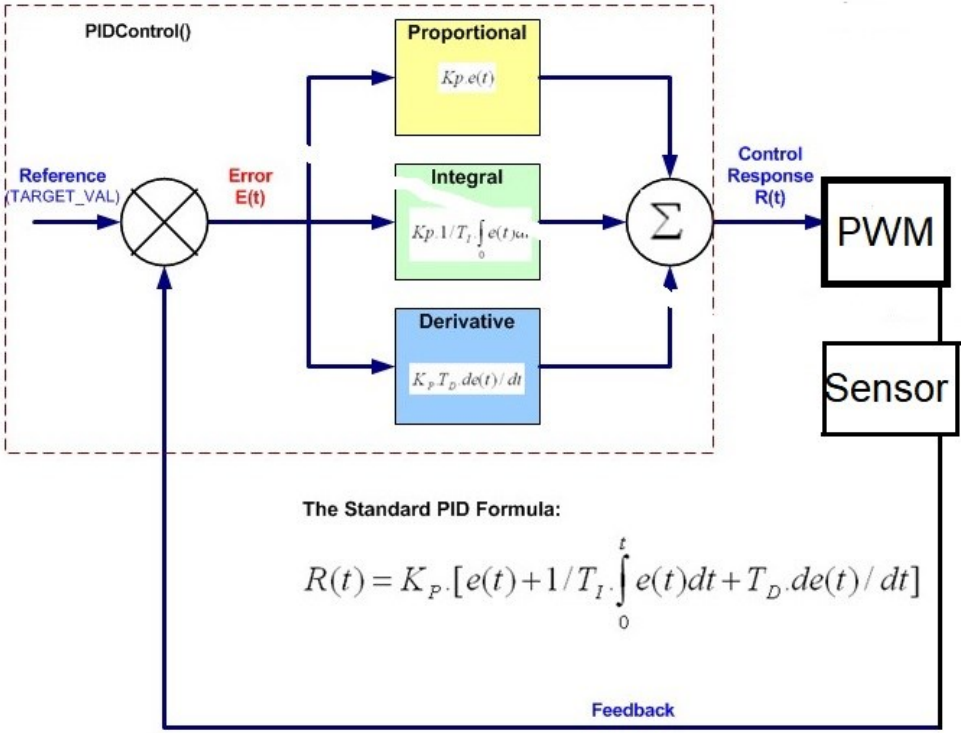


Die Idee hatte ich schon vor 2..3 Dekaden……

Auf der Seite von Ing. Michael Schmidt ist das schön beschrieben.

https://www.elektrik-trick.de/sminterf.htm

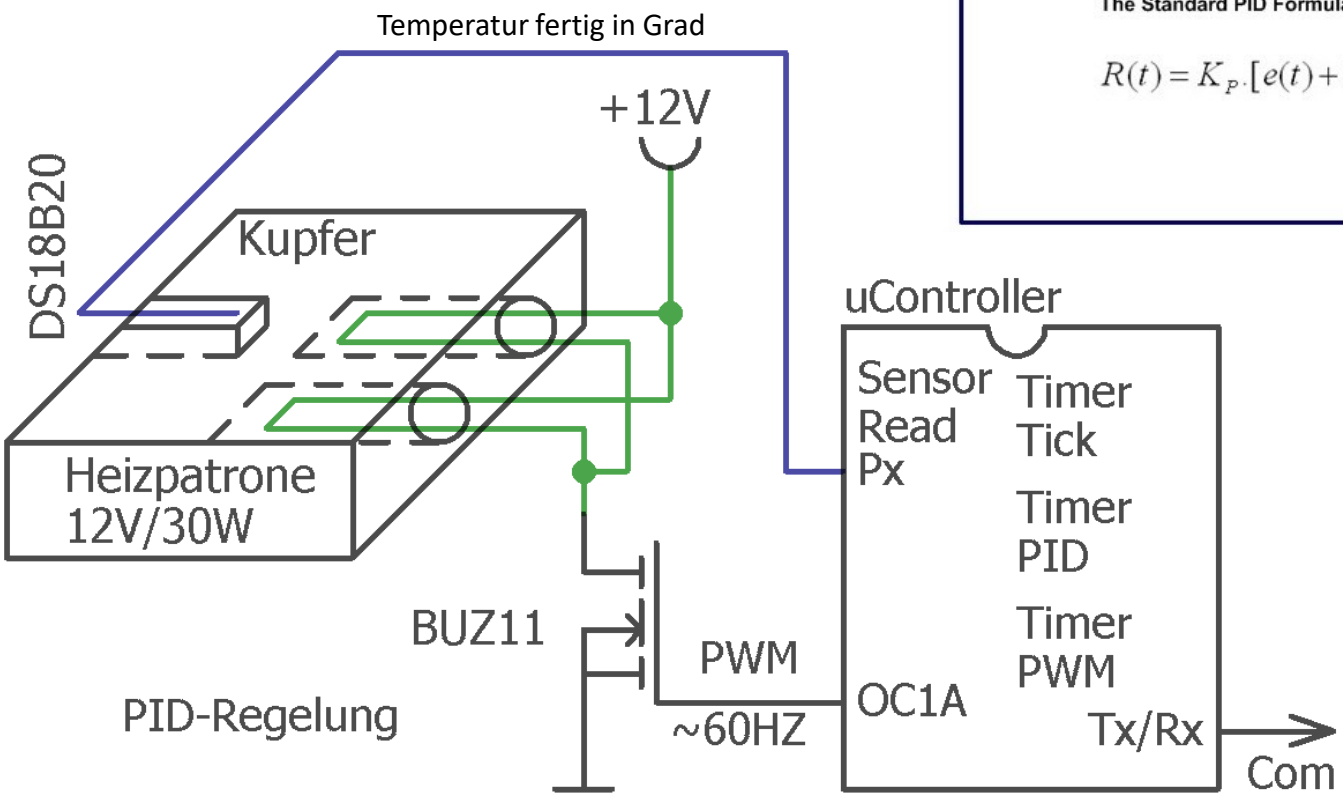
PID →
Proportionale
Integrations und
Differentiationsregelung

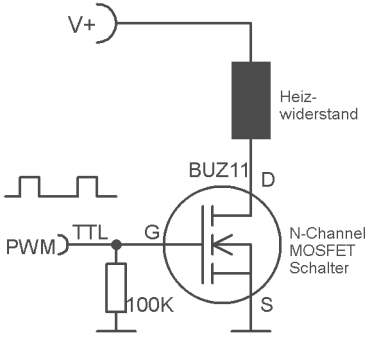


The Standard PID Formula:

$$R(t) = K_P.[e(t) + 1/T_I.\int_0^t e(t)dt + T_D.de(t)/dt]$$

Thermisches Regelsystem



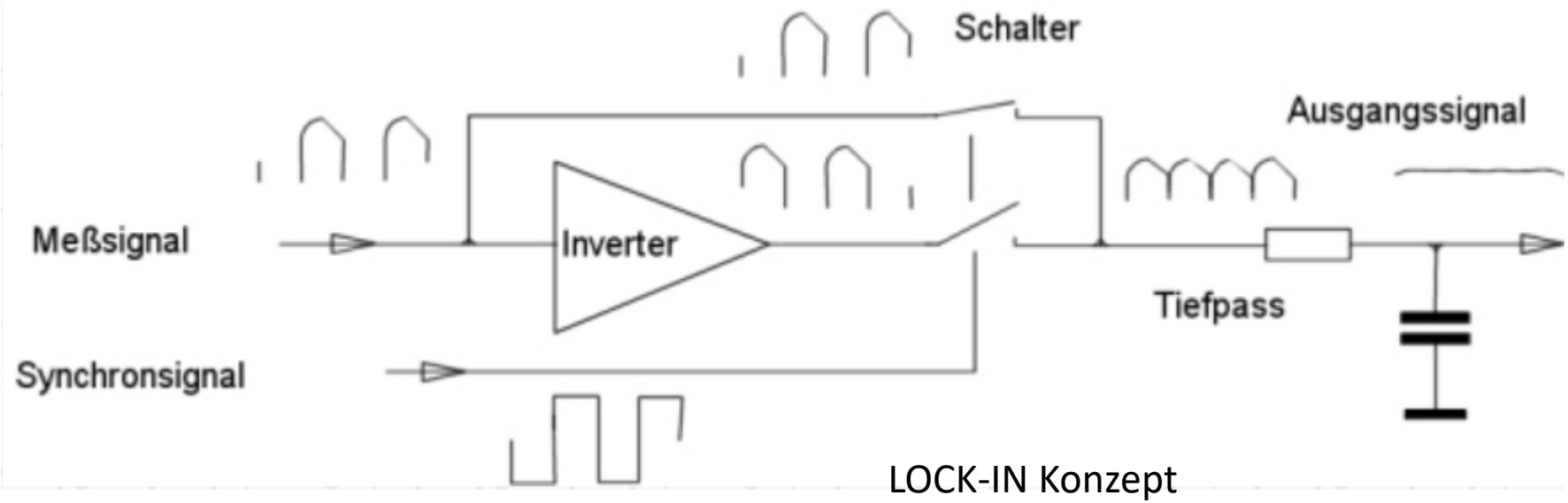


Projekt Idee **Zeitzeichensender** Decoder (77KHz) mit **LOCK-IN** Verstärker (selbst gurgeln), zur Signalverbesserung des Empfängers.

Meine Seite dazu:
https://homepages.uni-regensburg.de/~erc24492/DCF_Pollin/DCF_Pollin.html
Der µC kontrolliert und erzeugt das Reference Signal 77Khz, die negative Spannung (switcher) für einen Analog-Wechselschalter (CMOS 4066), der das Signal + 180Grad switcht, und liest den Ausgangsintegrator

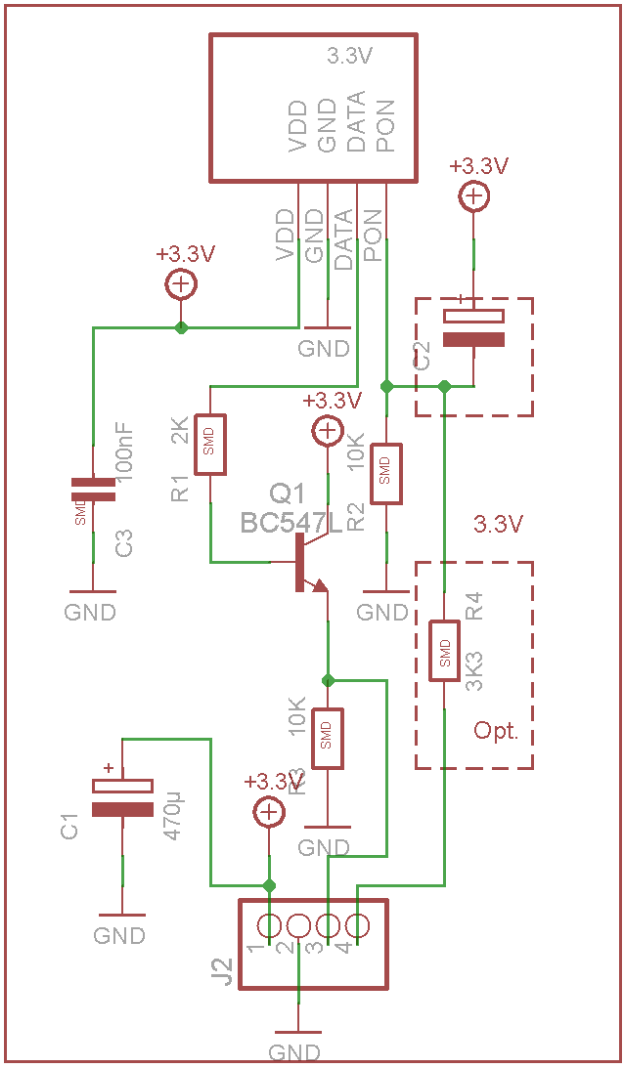
Quelle: http://www.elektronik-labor.de/Elo/Lockin.html

KONZEPT:



LOCK-IN Konzept

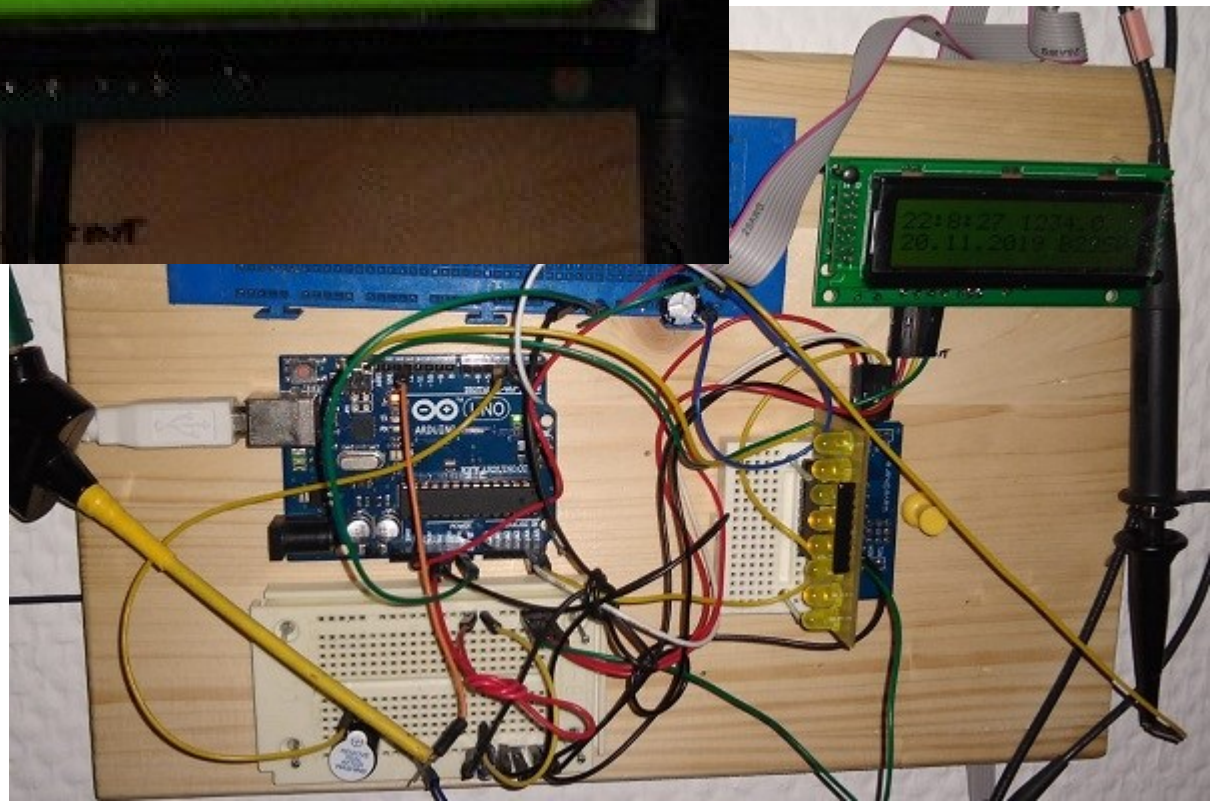
Kleine Adapterschaltung zum Pollin DCF Modul ~6€



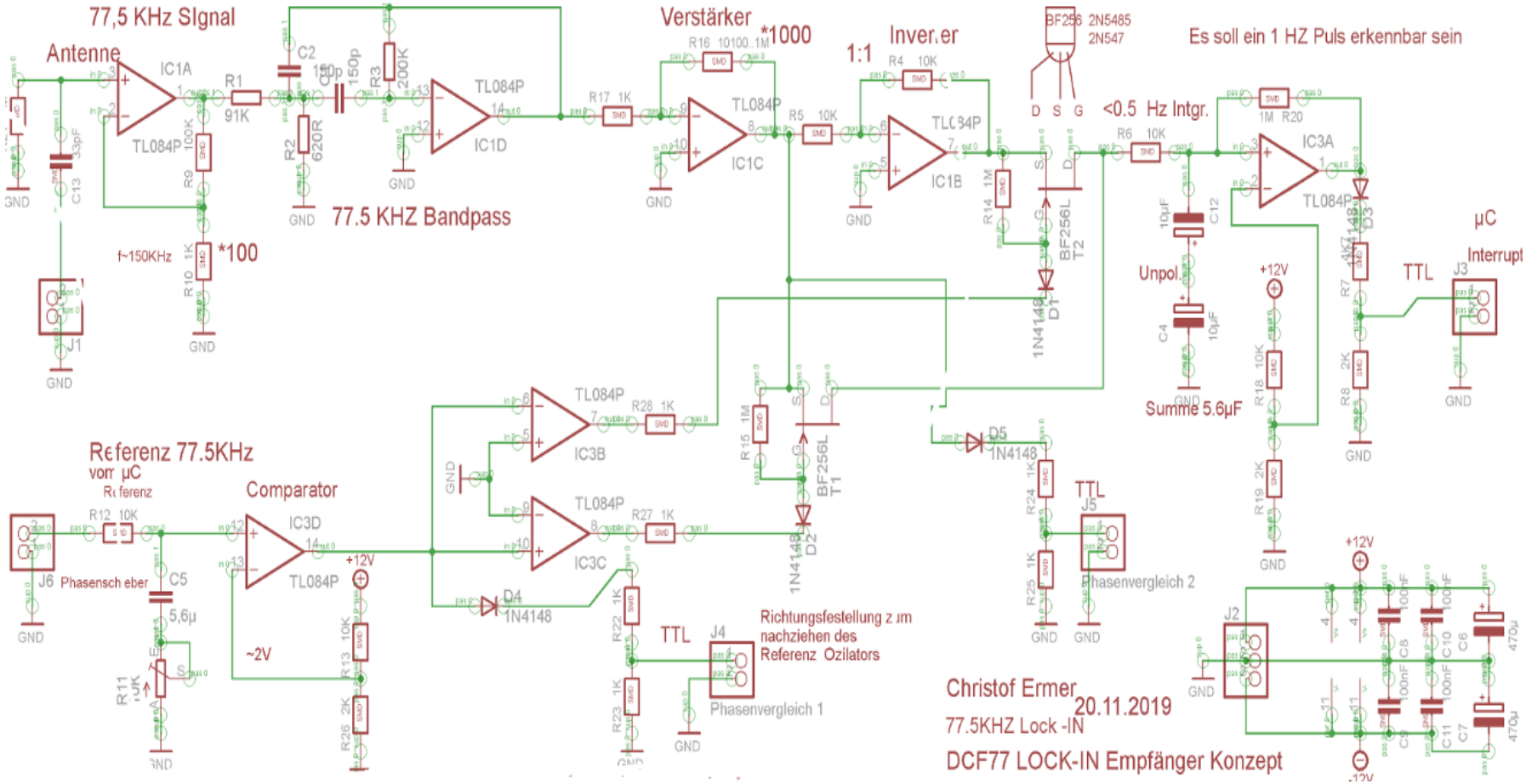
3.3V
PON bekommt High-Puls beim Einschalten→ dann Low (0V)
Emitterfolger puffert schwaches Signal.
+ Filterkondensatoren, dann kann man eine lange Leitung von der Antenne weglegen

DCF Uhr Anzeige,

Und, im Wortsinn,
Steck-Brett Aufbau.
Hängt an der Wand

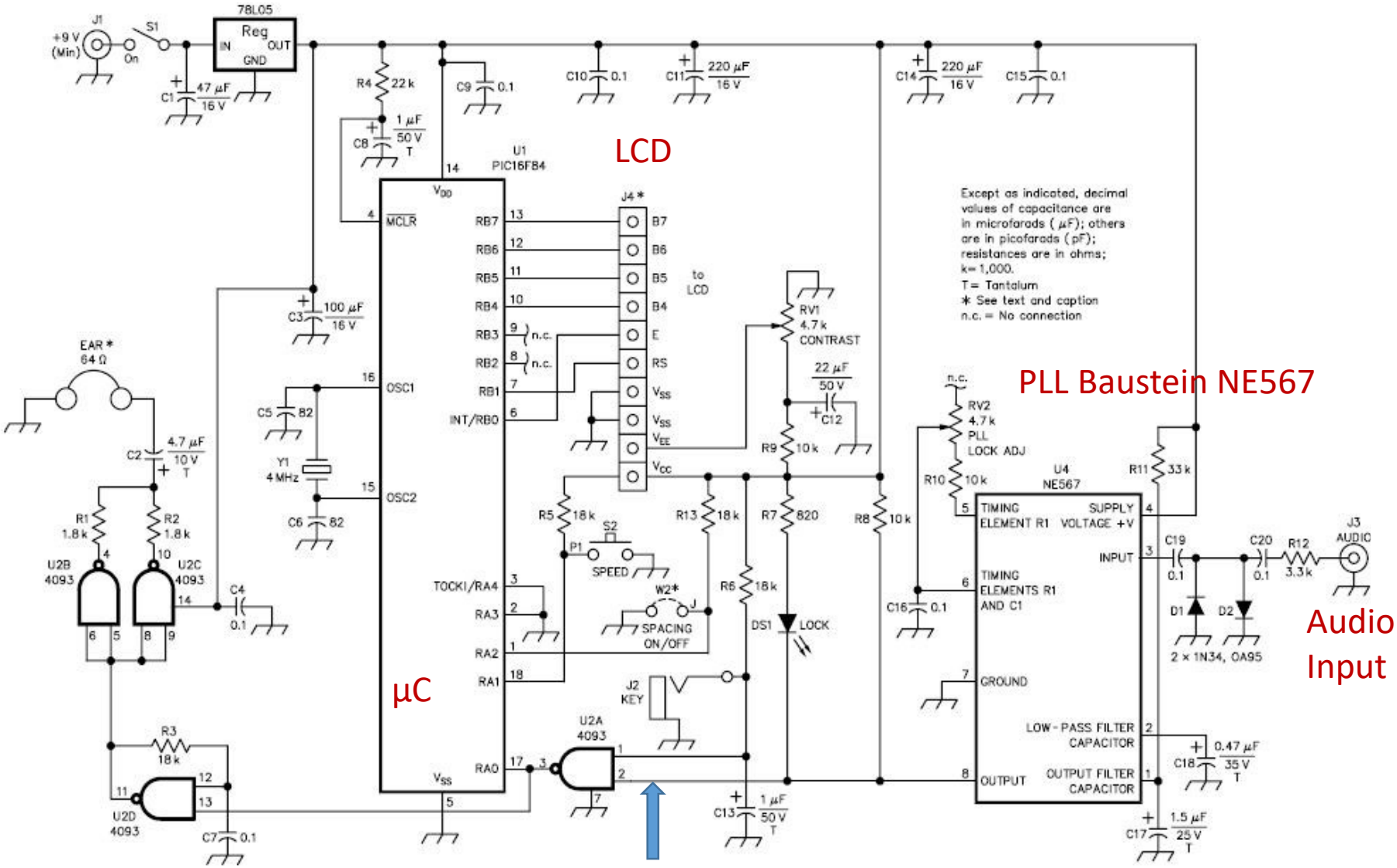


Hier sind noch jede Menge Fehler darin, (nicht nachbauen !!) .
Es ist eine Designstudie

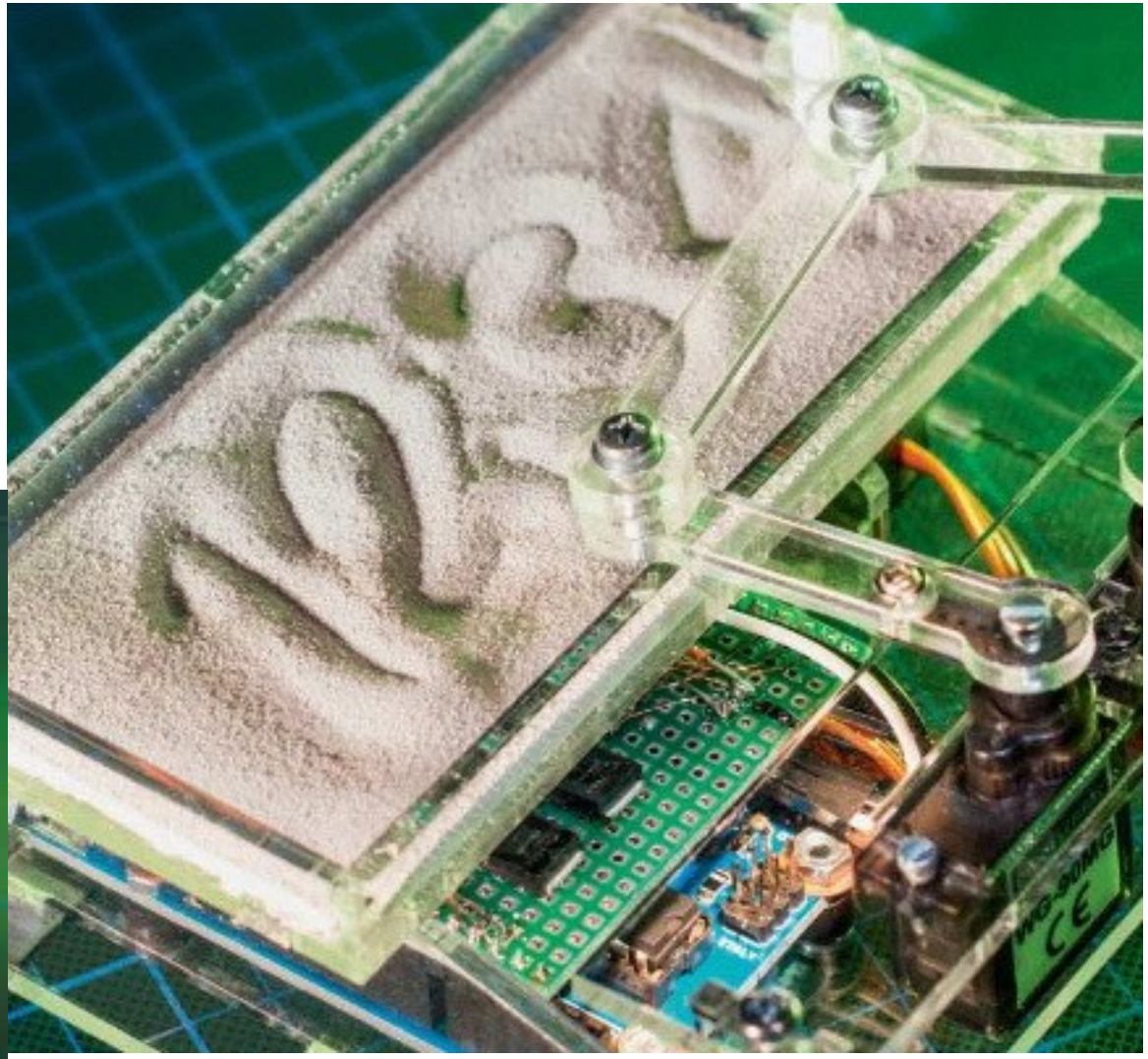


CW MORSE Decoder

mit Hilfe eines PLL Tondedetctors NE567 (**1602 led CW decoder Morse code translator Ham Radio Accessory)**



Der µC erledigt die zeitgetreue Dekodierung



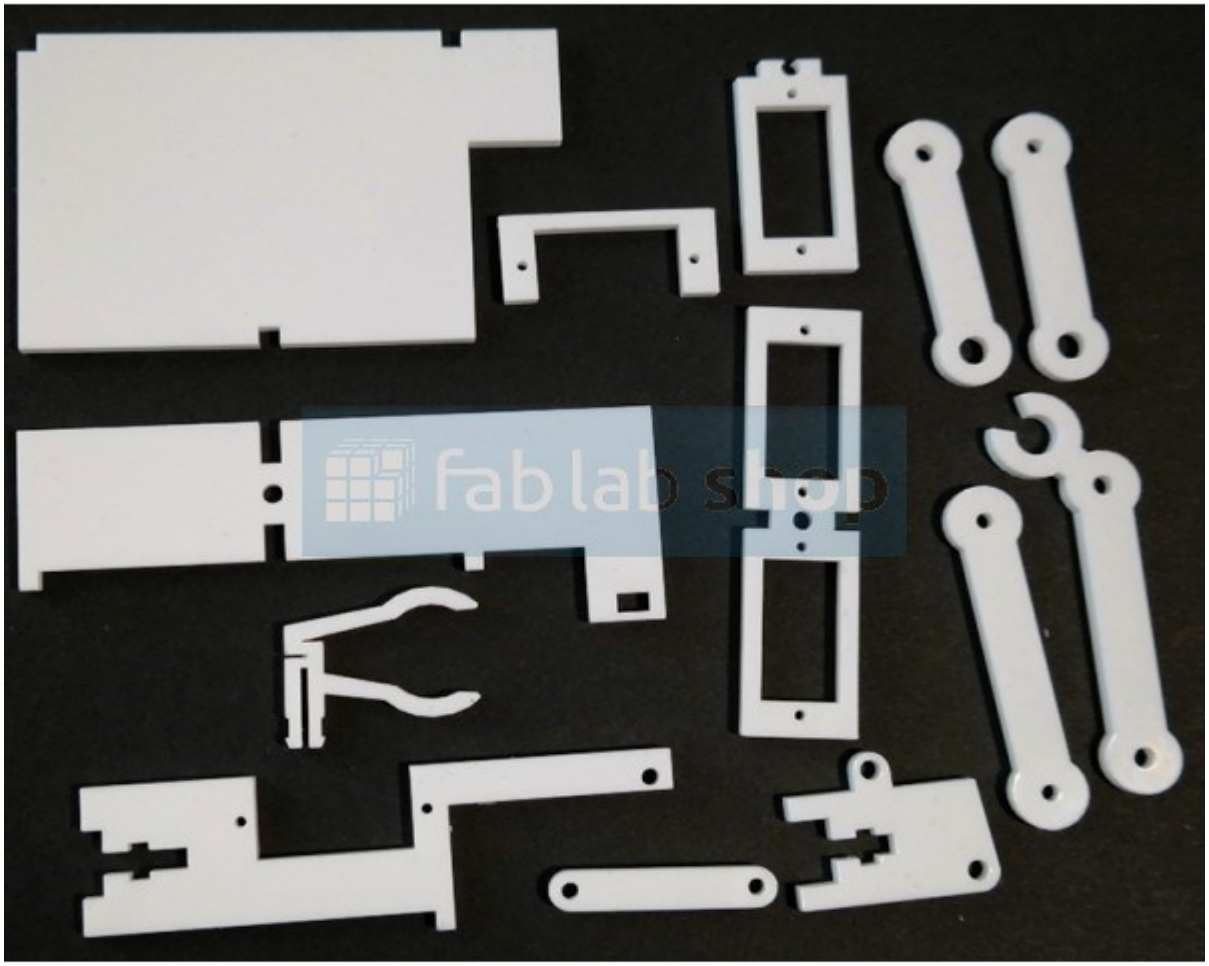
Facebook Video Hashtag „**#PlotClock:"** aus→ **"Project-K"**

The #clock that writes the #time!

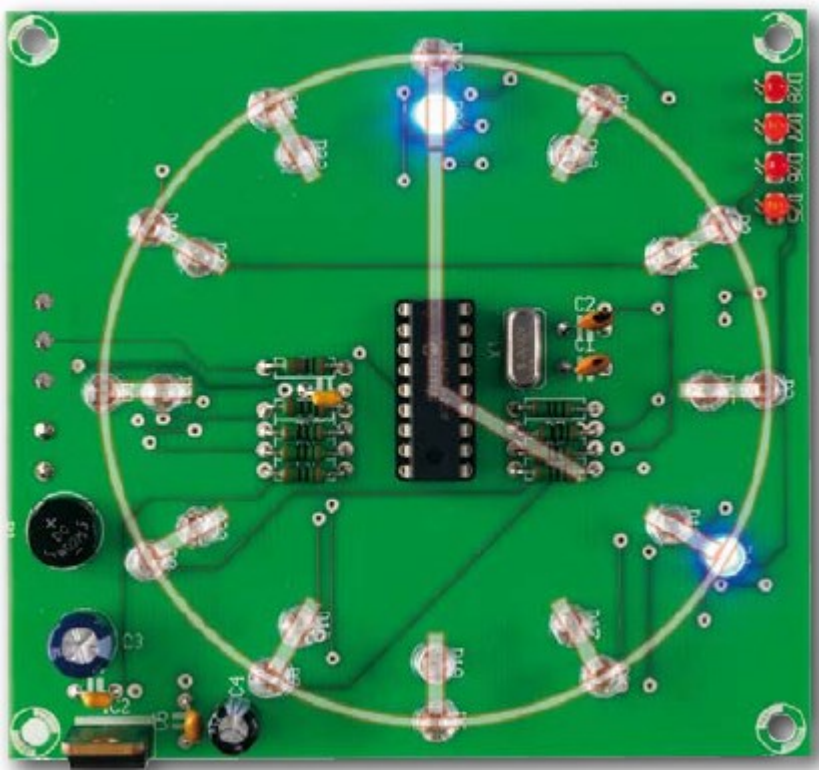
https://www.facebook.com/on.zeroground/videos/886994771369015/

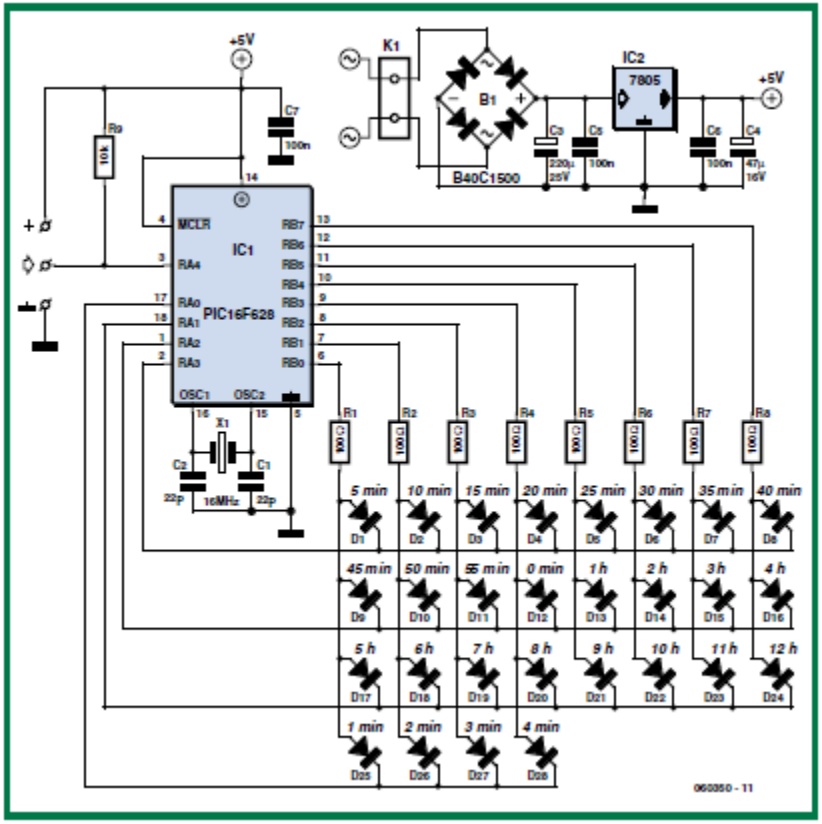
Plotclock

Bausatz: http://www.fablabshop.de/plotclock-bausatz.html
Video: https://www.facebook.com/on.zeroground/videos/886994771369015/



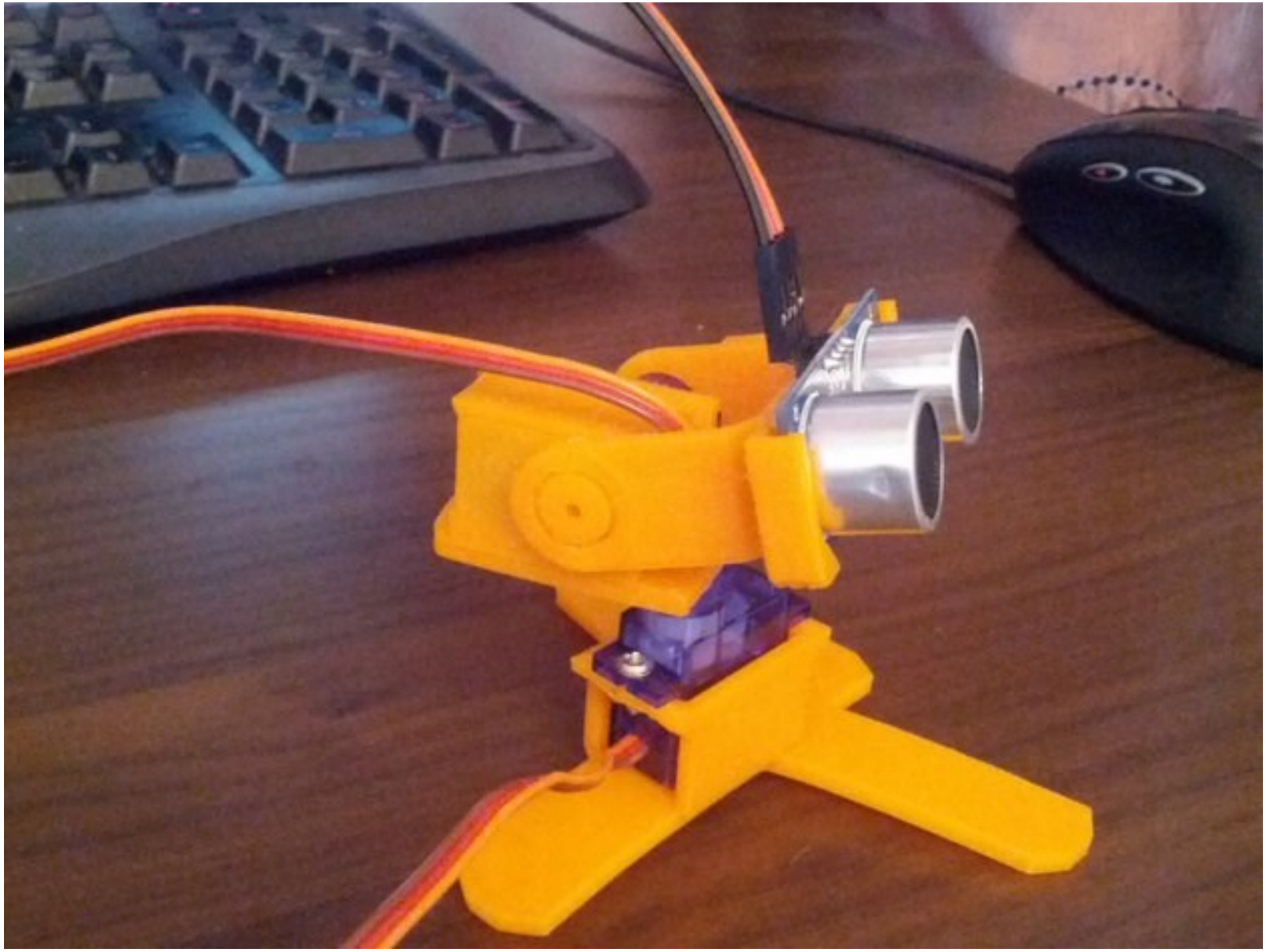
Projekt Idee Simple Clock





Quarz 2^15 = 32768 Hz
Batteriepuffer.
Leichtes einstellen.
RTC IC oder Funkuhr Modul oder keines von beiden. ?

Ultraschall-Auge mit XY-Servo

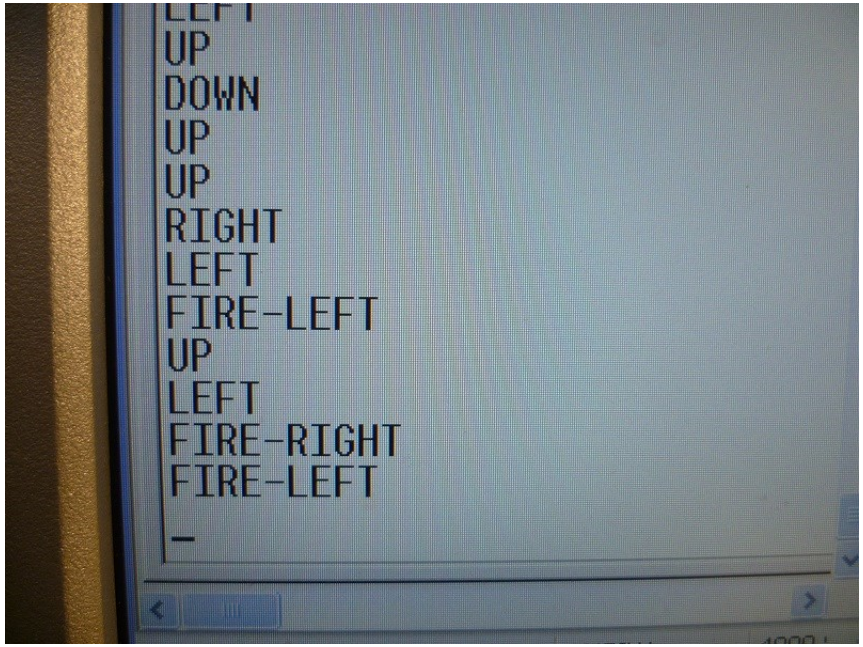


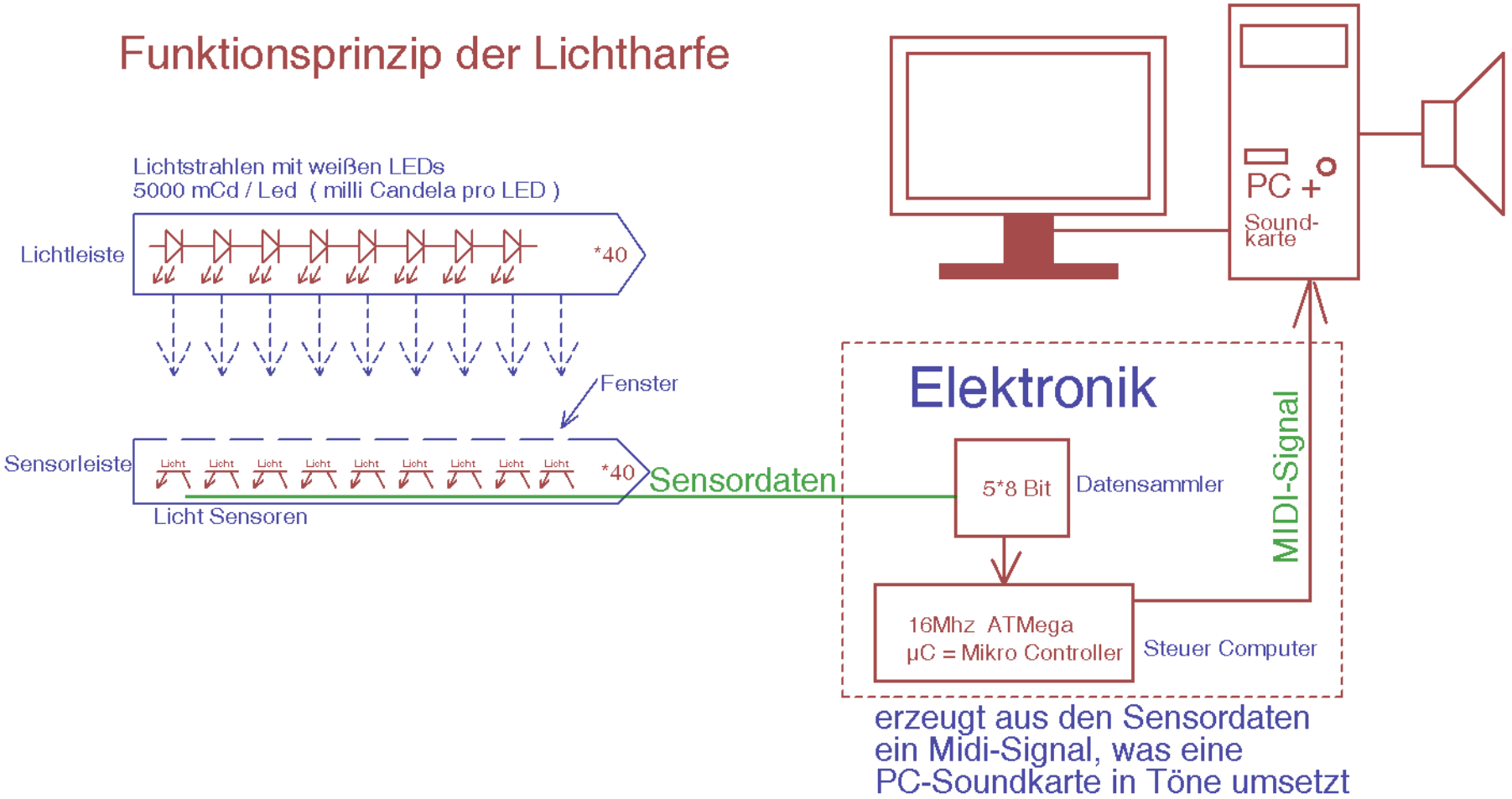
Text Kommando Sender

Mnemonic Text Sender zur LabView Programmsteuerung via serielle Schnittstelle.

Realer Laborbedarf zur Laserspiegeljustierung

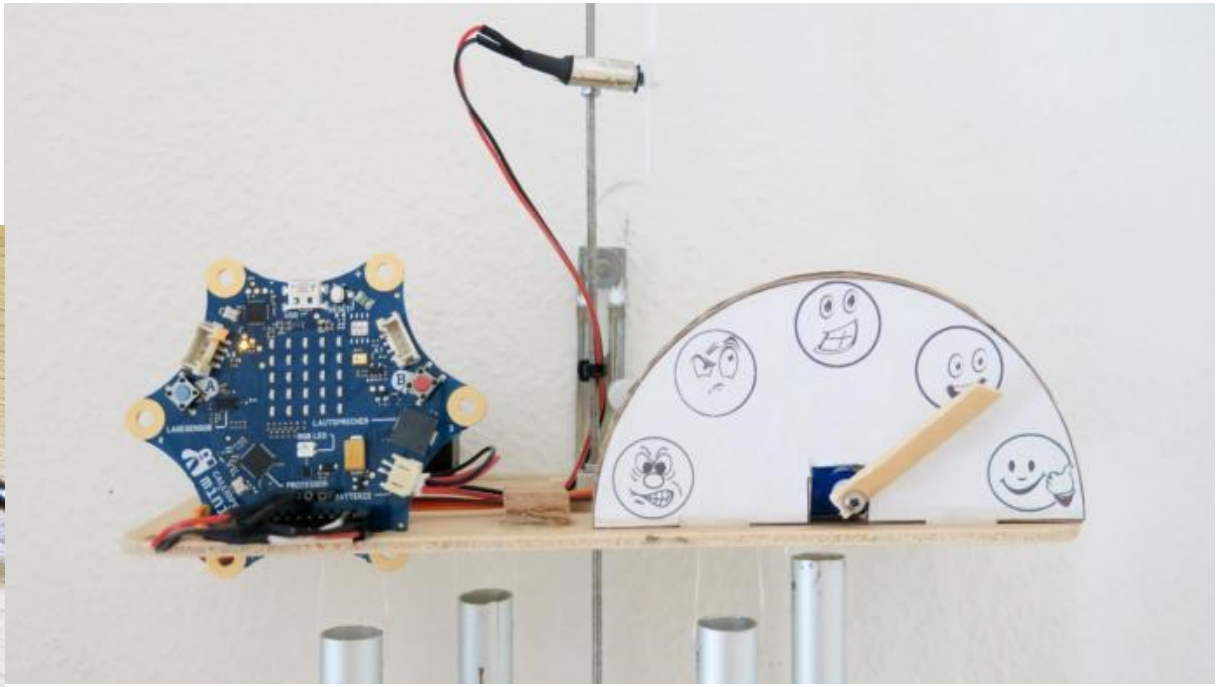
Text Kommando Format: **FIRE_LEFT<13>**

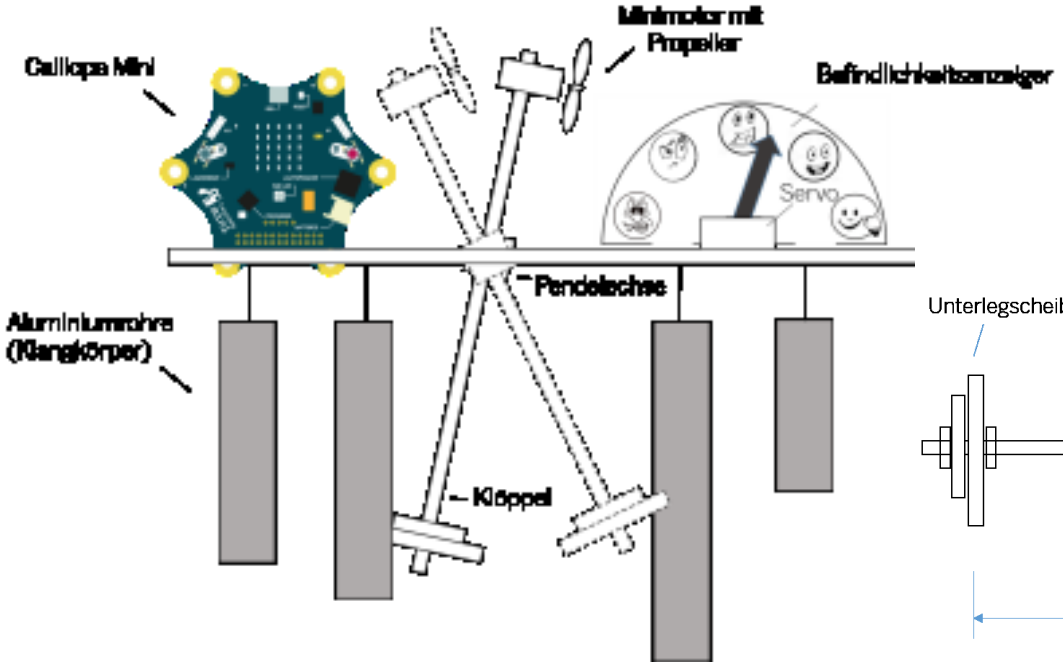




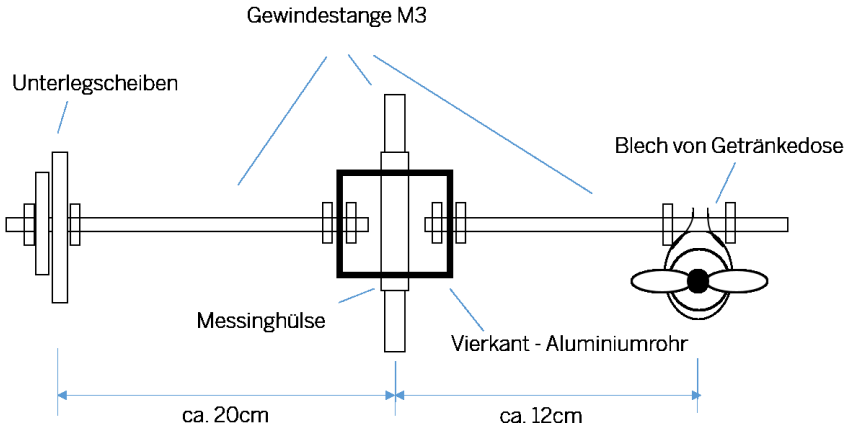
Eine „Software UART“ Schnittstelle sendet das vom AVR generierte MIDI Signal an den MIDI Eingang einer ‚alten‘ Soundkarte. Über die „echte“ UART wird die Steuersoftware, welche in LabView realisiert ist, bedient.

Lärm-O-Meter

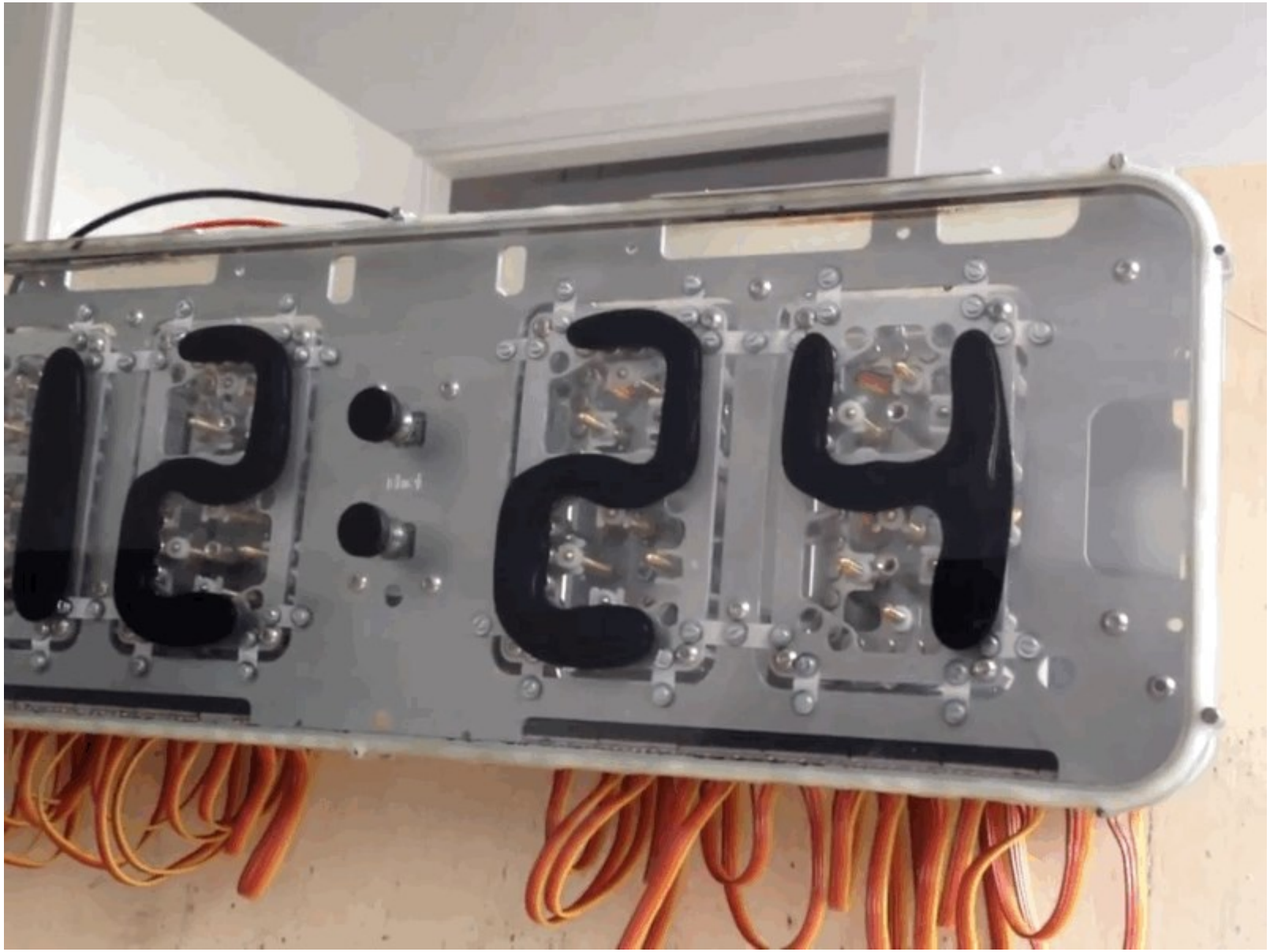




https://www.heise.de/make/artikel/Laerm-O-Meter-fuers-Klassenzimmer-3879235.html

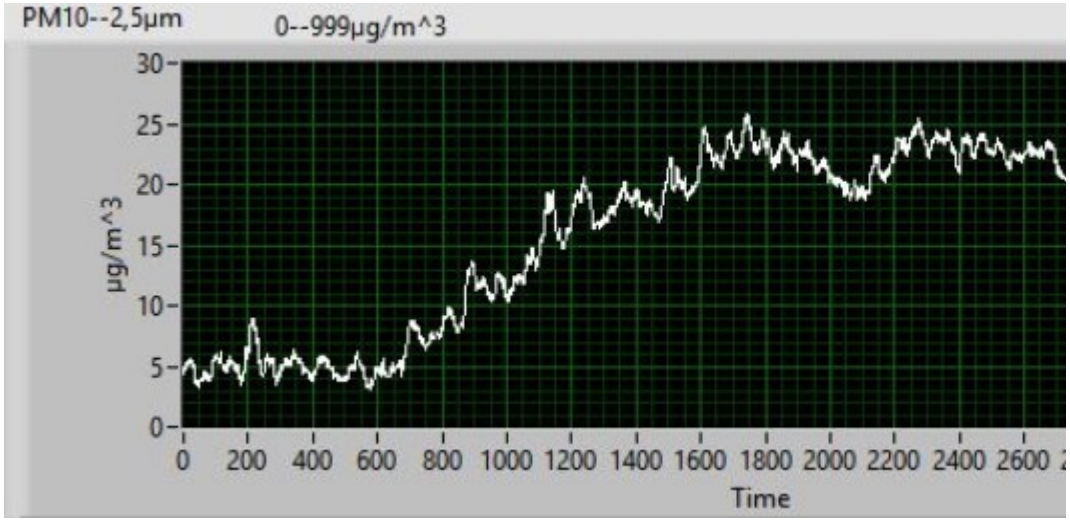


Magnete ziehen magnetisierbares Wasser in die Ziffern.. BCD Treiber



Partikel bzw. Feinstaub Sensor

Mein Artikel dazu: https://homepages.uni-regensburg.de/~erc24492/Feinstaub_Sensor/Feinstaub_sensor.html
Für billiges Geld ~30 gibt es ein Feinstaub Sensor Modul mit ***seriellem Daten***ausgang. **SD011 Feinstaubsensor-nova-fitness.** Ideal um es mit dem µController auszulesen und die Daten in lesbare Formen zu transformieren, zu logen, Langzeitmessungen durchzuführen.



Aber, da ich ja mein COM TxRx selber brauche, muss/kann ich noch einen 2. „**Software Uart**“ für Rx Daten tricksen.
Aber auch ein Grafik taugliches LCD wäre jetzt wunderbar.
So entwickelt man letzlich ein eigenes, verkaufbares Gerät.

Anschluss des **SD011 Feinstaubsensor-nova-fitness**

TTL Signal Ebene (Prüfen ob nichct doch 3.3V. Dann Rx mit Spannungsteiler 1K2K

Von Links:

Pin 3= +5V

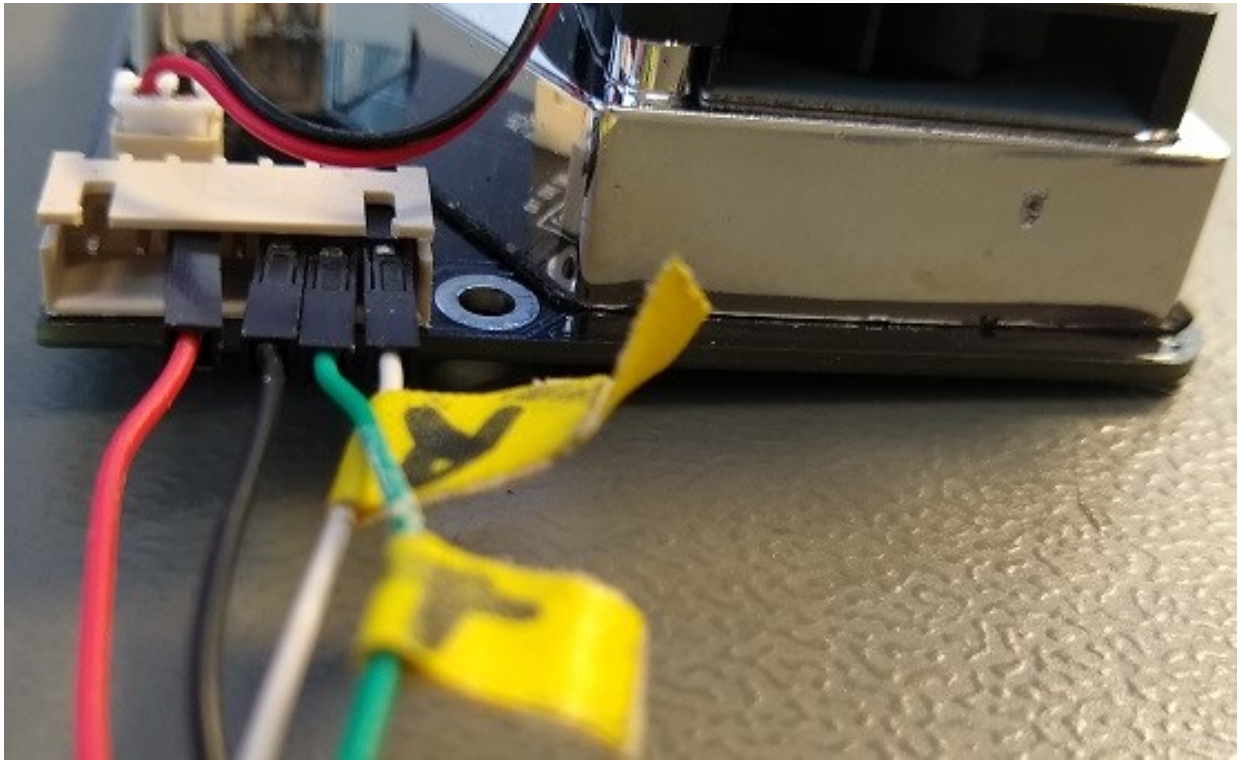
Pin 5=GND

Pin6=Rx --> Tx vom µC

Pin7=Rx --> Rx vom µC

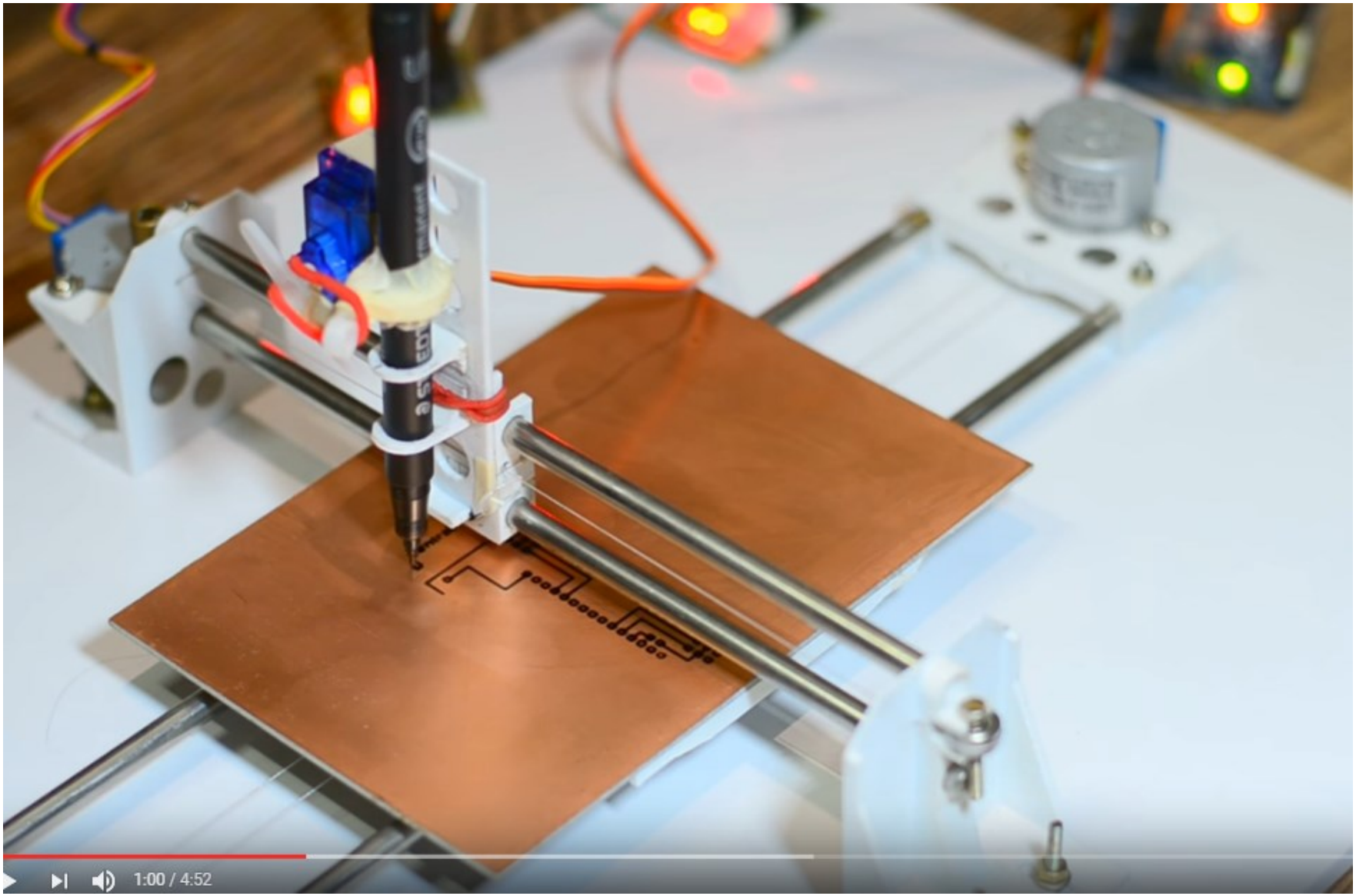
*, *, +5V, *, Rx, Tx

…..(RxTx kreuzen !)

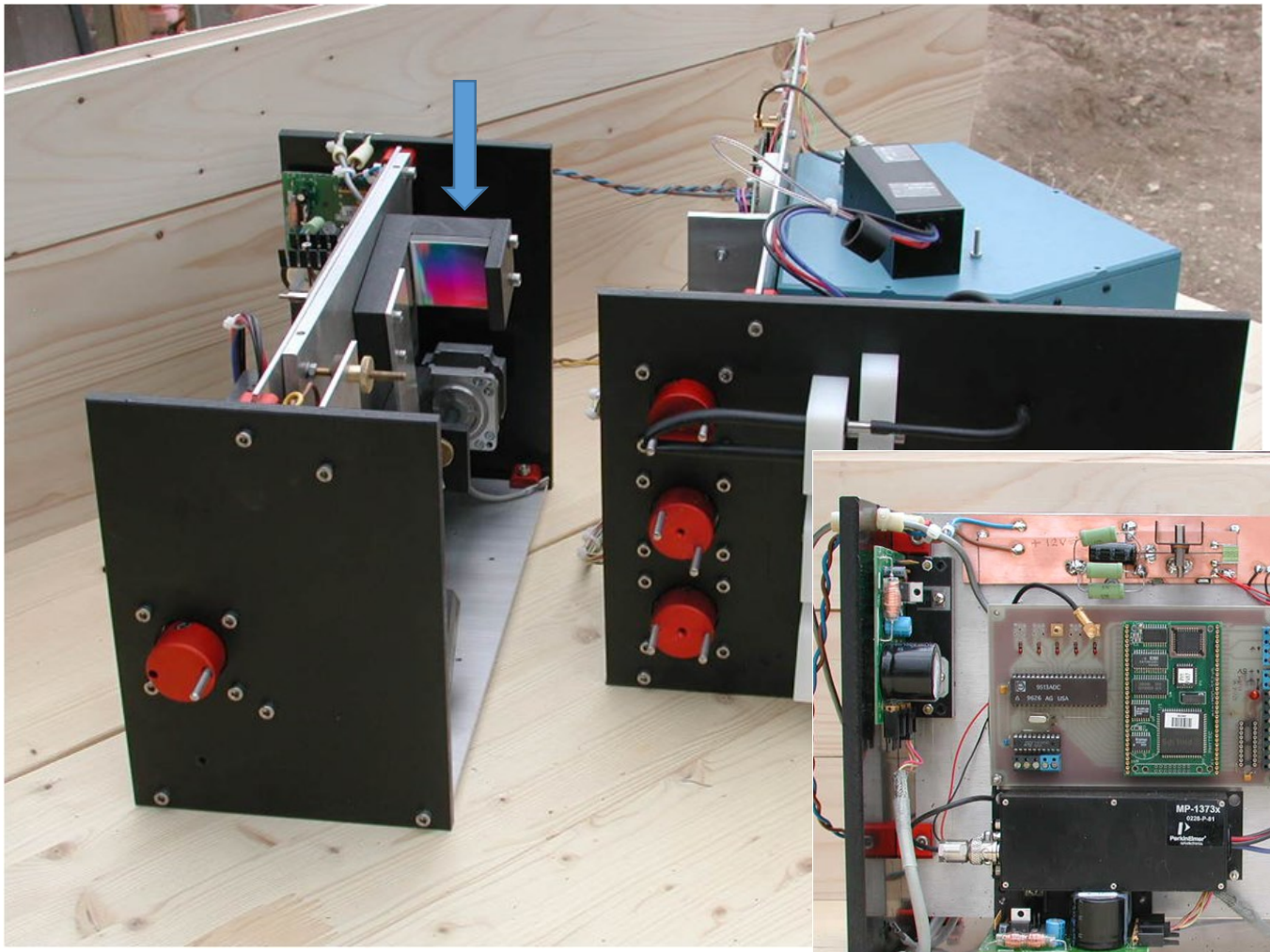


DIY PCB Ink Plotter using Arduino and GRBL CNC

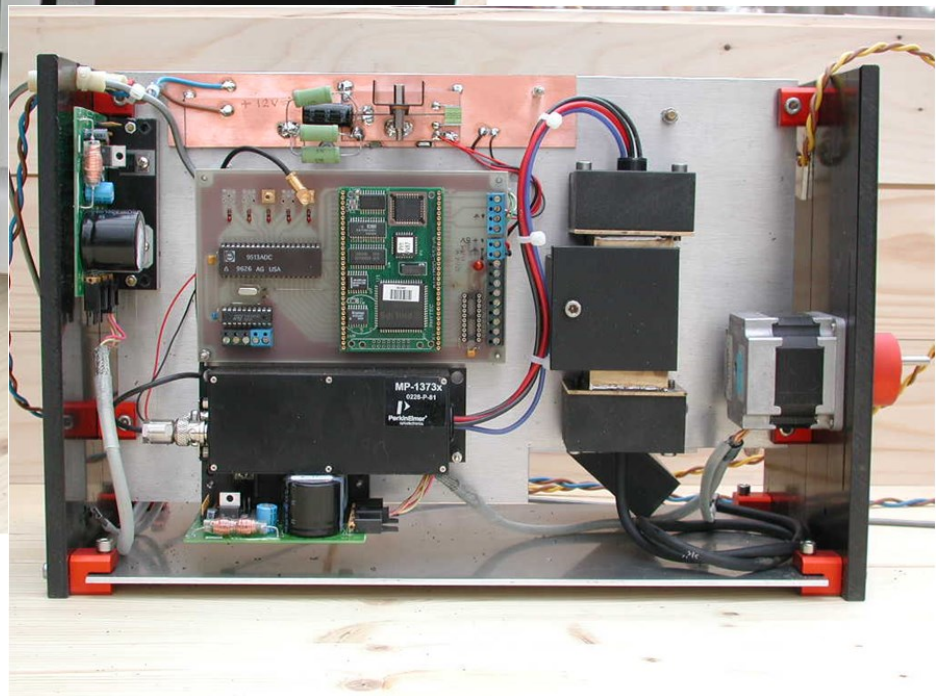
https://www.youtube.com/watch?v=oPVc9ixj0iU



Ein eigenes Projekt eines früheren Kursteilnehmers war einen Getränkeautomaten zu bauen.
Ein AT16 bediente den Geldschlitz, den Motor, die Öffnungsklappe etc.



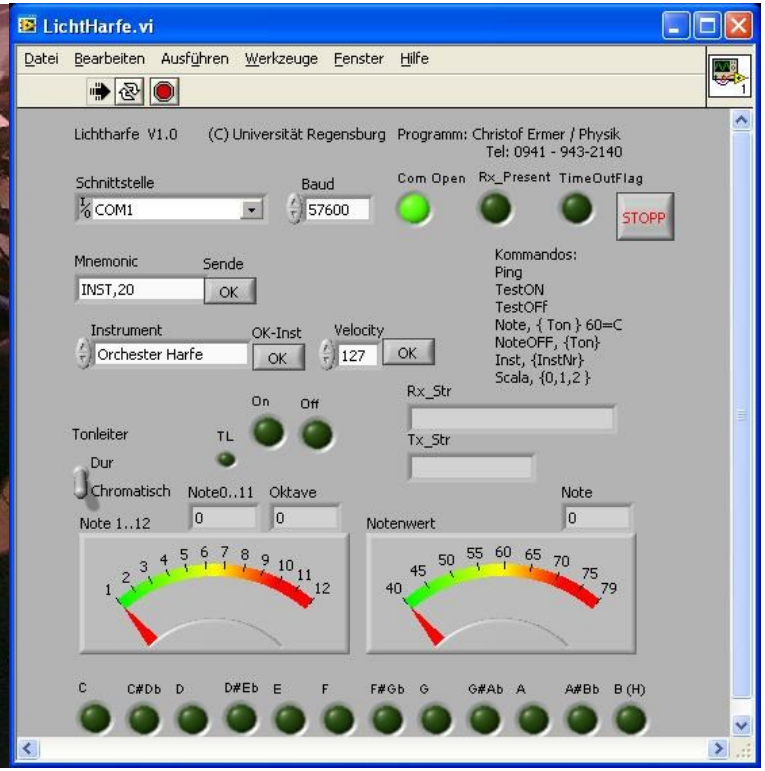
Linke Seite:
mit **µC**, der die -
Zählung
des Photomulitplier
übernimmt.
Einfache Mnemonic
Steuerung.

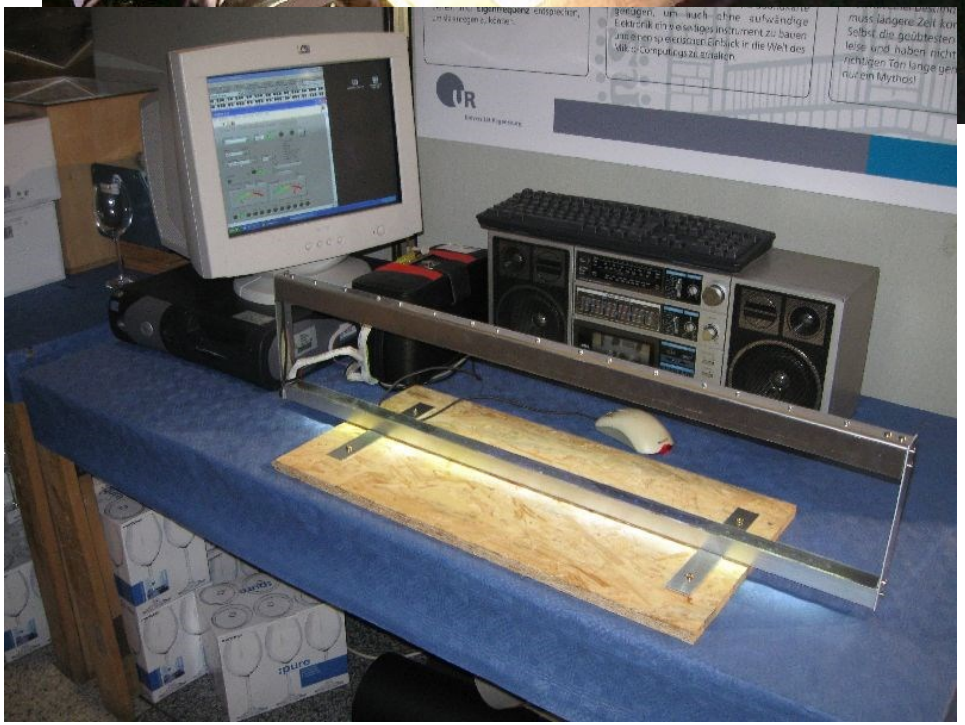


Selbstbau Spektrometer mit
SINUS-Antrieb für das Gitter.
Schrittmotoren treiben die Schlauchpumpen.
(…die roten 2-Achs-Pumpen haben sich bewährt.)

Projekt.. Weißbierglas einschenken

Weißbier:https://www.facebook.com/video.php?v=1032990406720691&pnref=story





Lab-View Frontend mit Mnemonic Steuerung

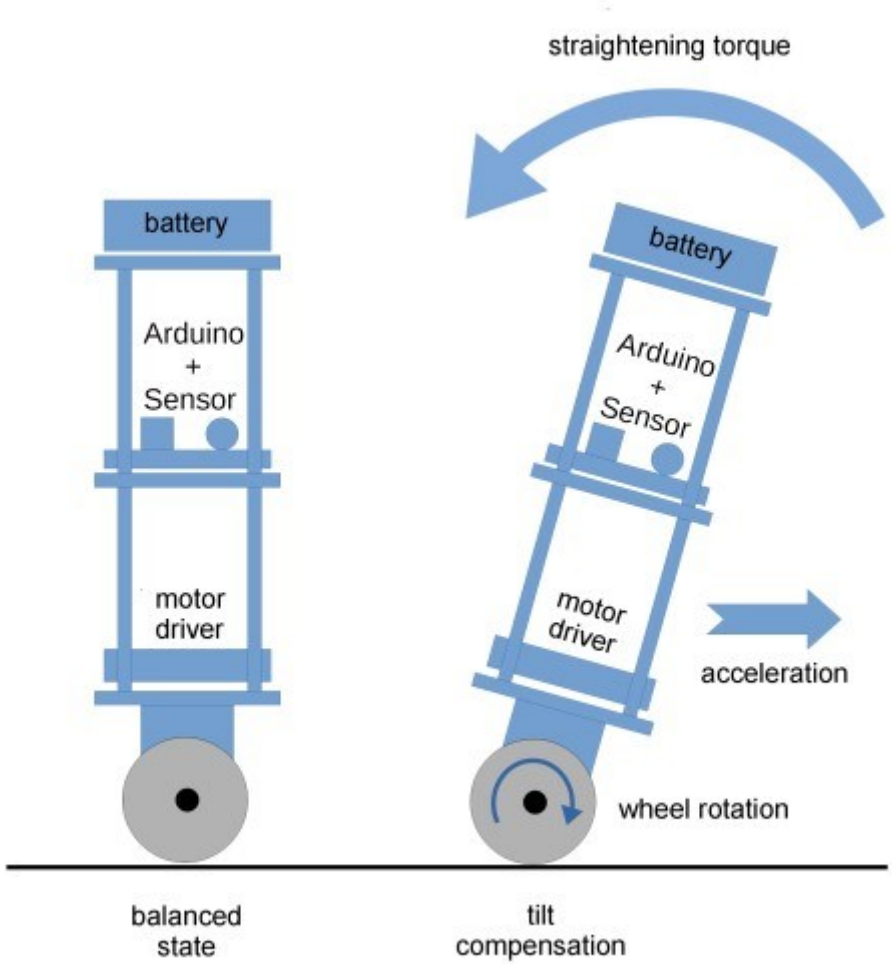
http://www.youtube.com/watch?v=FXU-4QchS4U
und
https://www.youtube.com/watch?v=dy4YgmcelMQ

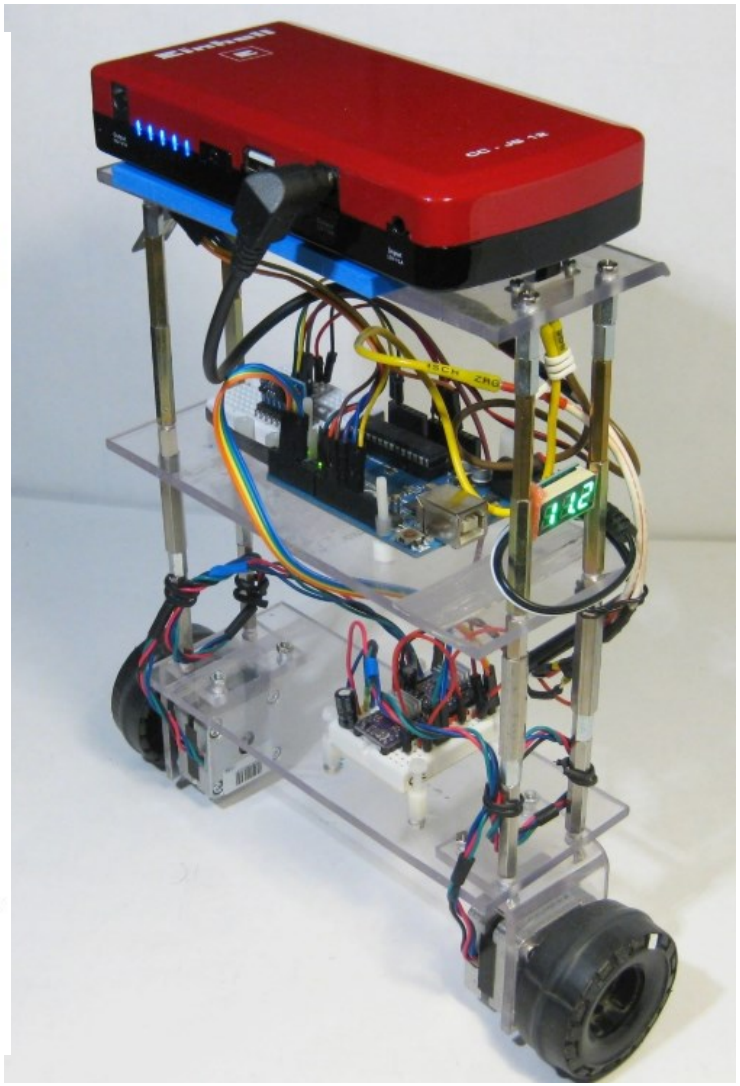
Projektbeispiel. µC Organist. Midi Dateien steuern via µC Aktoren (Magnetstempel)

Mechanischer Aufsatz für reale Orgel ersetzt notfalls den Organisten. Ein kommerzielles Beispiel wie man mit µControllern durchaus einträglich Geräte bauen kann.

https://www.youtube.com/watch?v=5UQyNSARVho&t=133s

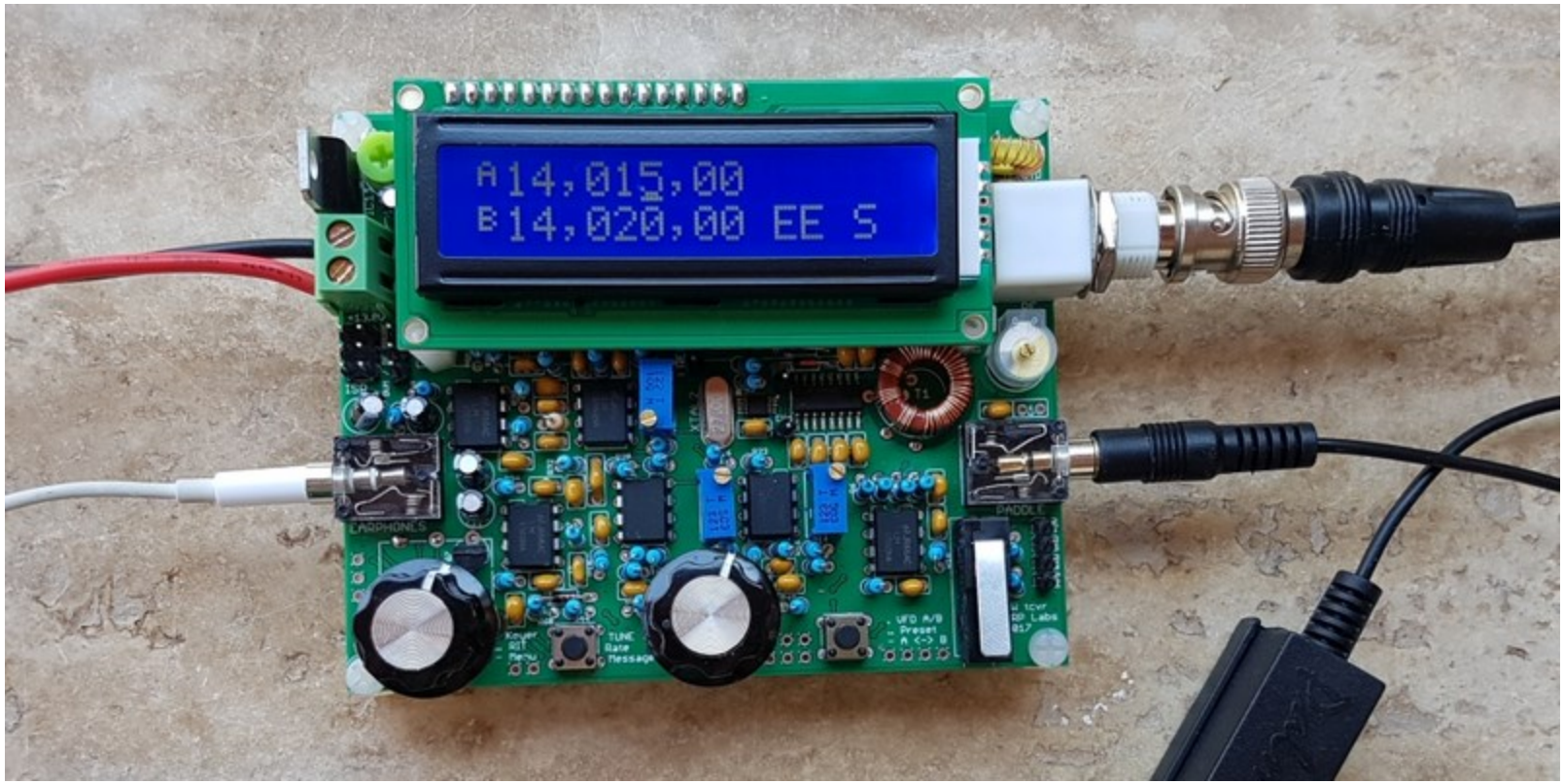
https://www.elektormagazine.de/articles/balbot-ein-selbstbalancierender-roboter

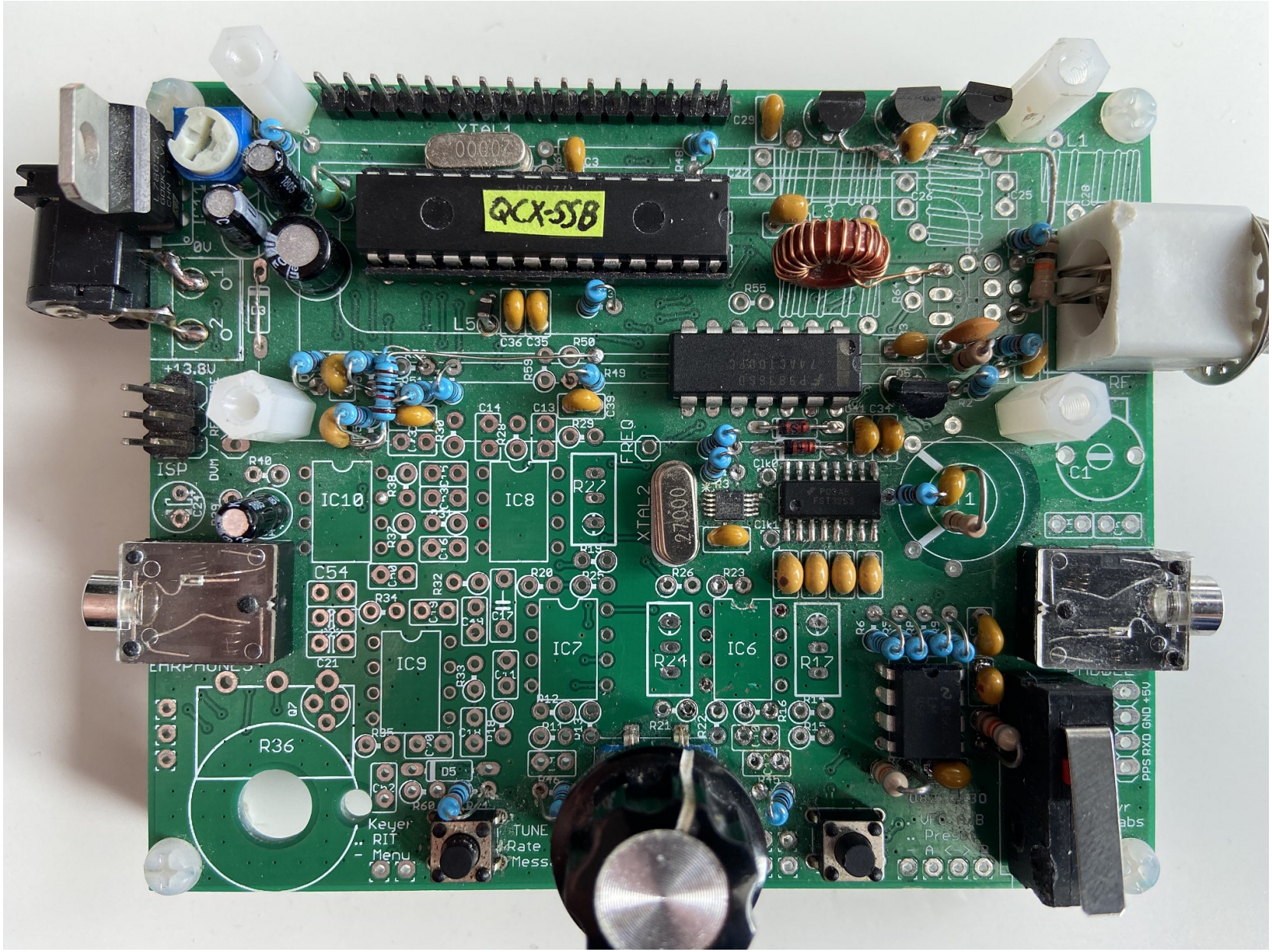




QRP CW Shortwave Transmitter / Decoder → CW Morsen
https://hackaday.com/2020/05/27/atmega328-ssb-sdr-for-ham-radio/

ATMega328 für Amateurfunk als SSB (softwarebasierte Single Side Band Tranceiver/Decoder)
https://hackaday.com/2017/09/13/a-fully-featured-fifty-dollar-qrp-radio/
49-Dollar-Einzelband-CW-HF-Transceiver-Kit (Morsecode)




QCX-SSB
XTAL1
20000
XTAL2
27000
FREQ
+13.8V
ISP
IC10
IC8
IC7
IC6
IC9
C54
R36
R22
R24
R17
L1
L3
L5
C1
RF
PPS RXD GND +5V
Keyer
RIT
Menu
TUNE
Rate
Mess
VFO
Preset
A

Robotik mit ROS

Tipp: ROS fähige Leute sind gesucht.
Phyton C++ Netzwerk
Simulator umschaltbar auf Echtbetrieb

http://reemc.pal-robotics.com/en/

http://www.ros.org/

http://www.amazon.de/Programming-Robots-ROS-Practical-Introduction/dp/1449323898/ref=sr_1_1?ie=UTF8&qid=1458817416&sr=8-1&keywords=ROS

MOOCs sind kostenlose Online Kurse,
https://www.coursera.org/
https://www.edx.org/
https://www.udacity.com/
Sehr gut : https://www.coursera.org/learn/machine-learning
https://www.coursera.org/course/ggp

An dieser Stelle zum „schnell auffinden“

Nützliche gute wie wichtige Adressen

C-Programmierung mit AVR-GCC
https://de.wikibooks.org/wiki/C-Programmierung_mit_AVR-GCC

Library Reference im WINAVR Ordner:
file:///C:/WinAVR-20100110/doc/avr-libc/avr-libc-user-manual/index.html

Alle Header Dateien in WINAVR zum AVR:

C:\WinAVR-20100110\avr/include\avr/iom328p.h

Sehr gute Mikrocontroller Forum Seite:
https://www.mikrocontroller.net/ und
https://www.elektronik-kompendium.de

Berechnen von RC LC Filtern: https://electronicbase.net/de/hochpass-berechnen/ oder http://www.sengpielaudio.com/Rechner-RCglied.htm

Ganzzahlige Datentypen:
#include <stdint.h>

Compiler
Versionsabhängig

Integertypen mit Vorzeichen

Typname	Bitbreite	Wertebereich	Alias (unter AVR-GCC)
int8_t	8 Bit	-128..127	signed char
int16_t	16 Bit	-32768..32767	signed int
int32_t	32 Bit	-2147483648..2147483647	signed long int
int64_t	64 Bit	-9223372036854775808..9223372036854775807	signed long long

Integertypen ohne Vorzeichen

Typname	Bitbreite	Wertebereich	Alias (unter AVC-GCC)
uint8_t	8 Bit	0..255	unsigned char
uint16_t	16 Bit	0..65535	unsigned int
uint32_t	32 Bit	0..4294967295	unsigned long int
uint64_t	64 Bit	0..18446744073709551615	unsigned long long

Fließkommazahlen:

Datentypen (Dezimalzahl)	Typische Größe in Bits	Wertebereich	Genauigkeit
float	32	$+ / - 10^{38}$	7 Nachkommastellen; einfache Genauigkeit
double	64	$+ / - 10^{308}$	16 Nachkommastellen, doppelte Genauigkeit
long double	128	$+ / - 10^{3000}$	16 Nachkommastellen, sehr hohe Genauigkeit

Die echte Bitbreite ist evtl. auch von der Compiler Version abhängig:
Also selber erst mal Testen und Compilerbeschreibungen lesen.
Leider altert Software Wissen auch.
Aber so in etwa kann man es schon mal nutzen

```
#include <stdint.h>
```

Exact-width integer types
Integer types having exactly the specified width

typedef signed char	int8_t
typedef unsigned char	uint8_t
typedef signed int	int16_t
typedef unsigned int	uint16_t
typedef signed long int	int32_t
typedef unsigned long int	uint32_t
typedef signed long long int	int64_t
typedef unsigned long long int	uint64

AVR-GCC
https://gcc.gnu.org/wiki/avr-gcc
https://gnutoolchains.com/avr/

Von C:\WinAVR-20100110 zu ??

https://www.mikrocontroller.net/topic/390253 Da habe ich das her

> Kann Ich denn ne Toolchain von 2015 in WinAVR reinpacken?
Ja, kannst Du. Dazu folgendermaßen vorgehen:

1. avr-gcc-4.7.2 toolchain herunterladen
   https://sourceforge.net/projects/mobilechessboar/files/avr-gcc%20snapshots%20%28Win32%29/avr-gcc-4.7.2-mingw32.zip/download

   Die ist zwar auch schon von Ende 2012, aber ist hinreichend frisch
   und läuft sehr stabil.
2. Verzeichnis C:\WinAVR anlegen
3. Die obige zip-Datei in C:\WinAVR entpacken
4. Das Verzeichnis
   von
      C:\WinAVR\avr-gcc-4.7.2-mingw32
   in
      C:\WinAVR\AVR-GCC-5-3-0
   umbenennen

5. Eine Datei namens C:\WinAVR\avr-gcc-5-3-0.txt erstellen mit folgendem Inhalt:

(Anmerkung habe neuere Version „ C:\WinAVR\AVR-GCC-5-3-0“)

```
-----------------
Windows Registry Editor Version 5.00

[HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall\WinAVR]
"DisplayVersion"="20100110"
"UninstallString"="C:\\WinAVR\\AVR-GCC-5-3-0 \\WinAVR-20100110-uninstall.exe"
-----------------
```

6. Datei C:\WinAVR\avr-gcc-4.7.2.txt in C:\WinAVR\avr-gcc-4.7.2.reg umbenennen

7. Datei C:\WinAVR\avr-gcc-4.7.2.reg mit Doppelklick ausführen

8. Datei ....\WinAVR-20100110\avr-gcc\bin\avr-size.exe nach C:\WinAVR\avr-gcc-4.7.2\bin\avr-size.exe kopieren

9. Ordner ....\WinAVR-20100110\avr-gcc\utils nach C:\WinAVR\avr-gcc-4.7.2\ kopieren

10. Systemsteuerung -> System -> Erweiterte Systemeinstellungen -> Erweitert -> Umgebungsvariablen -> Systemvariablen:

Path bearbeiten, und ersetzen:
Alt:
....\WinAVR-20100110\avr-gcc\bin
Neu:
C:\WinAVR\avr-gcc-4.7.2\bin

11. Fertig. AVR-Studio 4.18 starten und dann unter Projekt -> Configuration Options -> Custom Options kontrollieren, ob folgendes drinsteht:

[Haken gesetzt] Use WinAVR

avr-gcc: C:\WinAVR\avr-gcc-4.7.2\bin\avr-gcc.exe
make: C:\WinAVR\avr-gcc-4.7.2\utils\bin\make.exe

Wenn ja, hast Du alles richtig gemacht, wenn nein, kontrolliere bitte die REG-Datei.,

PLAN B:

https://forum.arduino.cc/index.php?topic=556557.0

„C“ Trick

TimeOut im while-Loop

Was passiert, wenn die Abbruchbedingung nie erfüllt wird?

Hier wird ein Timeout benötigt:

Trick: Counter mit herunter Zählen. Mit && logisch verknüpfen

```
ulTimer = F_CPU /2;   // 16*10^6  / 2 = 8000000
while (   ( Bedingung ) && --ulTimer )….
```

Bsp:

```
while ( !(rfm_cmd(0x27FF, 0) & RF_IRQFLAGS1_MODEREADY ) && --ulTimer)
{
 wdt_reset();  //WATCHDOG!   Damit kein Watchdog reseted.
};
```

Fast Modulo

Der Modulo (Rest) Operator **%** wird sehr oft benötigt.
Nachteil: Resource und Zeit intesiv.
Schneller ist ein Trick mit einfacher Binärer VerUNDung.
Immer dann, wenn vielfache von 2^n Ziel eiern ModuloOperation sind , ist dies zu bevorzugen.

Beispiel. Zähler von 0 bis 7
Modulovariante: NN++; NN %= 8; PORTC = (1<<NN)

Binäre UND Variante: **&**
Nehme vom gewünschten Modulowert im 2 hoch n Rythmus
→ N-1. Also statt Modulo 8 nehmen wir 8**-1** = 7 = 0b0111.
Ziel ist ein UND Vergleich mit Wert & Einsen → 0b0111

Binär VerUNDen Variante:
NN++; NN **&** 0x7; PORTC = (1 << NN) ;
Noch kürzer:
NN++; PORTC = (1 << (NN **&** 0x7));

Was mir gerade so einfällt. **RING BUFFER**. Immer wieder gebraucht. Bei der Analyse der UART Software ist mir ein Softwaretrick aufgefallen.
Einer Art Modulo mit binären zahlen in 2^n
Man **UND**ed einen „**Zähler-Parser“** mit (**2^n -1**)
Bsp:32 = 2^5 = 0b100000 -1 = 0b011111 = lauter EINSEN.
Ist nun ein Wert > als der **binäre Überlauf 0..31**, wird dieser abgeschnitten. Ein Resourcensparender schneller Modulo
33 = 0b100001
& 0b011111 (Maske)

0b000001 = 1 was ja Modulo: 33 % 32 = 1 wäre
Geht aber nur in der 2^n Reihe,.

```
//im Ringbuffer Init
pRBuf->uxRingBufMask = uxArraySize - 1;  //32-1
```

Anwendung: Einfügung im Buffer
pRBuf->uxHead = {Einfügezeichen}: ‚A‘
pRBuf->uxHead = (pRBuf->uxHead + 1) & pRBuf->uxRingBufMask;

```
*(pRBuf->pau8RingBuf) =  0;   // Null voranschieben (Terminierung)
```

```
typedef struct
//
**************gcaStr******************
{
#ifdef BIT8_RINGBUF_USED   //else 16Bit
uint8_t uxHead;
uint8_t uxTail;
uint8_t uxRingBufMask;
};

//always N^2 Size
void RingBuf_Init(RINGBUF_TYPE * pRBuf, uint8_t * pu8BufferStr, uint16_t uxArraySize)
{
pRBuf->uxHead=0;
pRBuf->uxTail=0;        //uxHead = uxTail;
//U16ArraySize Immer N^2 BUF_STRMAX never >256,  or change the Datatype from  HEAD & TAIL to "unsigned int".

                // test if the size of the circular buffers fits into SRAM
pRBuf->uxRingBufMask = uxArraySize - 1;     // 0b11111111  IST MASK
if( (pRBuf->pau8RingBuf = (uint8_t*) malloc( uxArraySize ) ) == NULL )  // auf Byte
pRBuf->pau8RingBuf =  pu8BufferStr;
*(pRBuf->pau8RingBuf) =  0;
};
```

```
void RingBuf_WriteC(RINGBUF_TYPE * pRBuf, uint8_t u8Rx)
//Rückgabeflag: Ok--> 0
{           // & uxRingBufMask ist Limit Size
pRBuf->uxHead = (pRBuf->uxHead + 1) & pRBuf->uxRingBufMask;
*(pRBuf->pau8RingBuf + pRBuf->uxHead) = u8Rx ;
};
            int RingBuf_ReadC( RINGBUF_TYPE * pRBuf )
            {
            if( pRBuf->uxHead == pRBuf->uxTail )
                        {
                        return RING_NO_DATA;
                        };
            /* calculate /store buffer index */
            pRBuf->uxTail = (pRBuf->uxTail+1) & pRBuf->uxRingBufMask;
            /* get data */
            return *(pRBuf->pau8RingBuf + pRBuf->uxTail);
            };
                                    void RingBuf_Flush(RINGBUF_TYPE * pRBuf)
                                    {
                                    pRBuf->uxHead=0;
                                    pRBuf->uxTail=0;     //uxHead = uxTail;
                                    *(pRBuf->pau8RingBuf) =  0;
                                    };
```

```
int RingBuf_Check_Overload(RINGBUF_TYPE * pRBuf)
// Overload State? RingBuf ? {
if( ((pRBuf->uxHead  + 1) & pRBuf->uxRingBufMask) == pRBuf->uxTail )
        {
        return RING_OVL; // returnflag: RXSTRBUF_OVL --> !=0
        };
return RING_OK;
};
```

Hilfsfunktionen

```
 #define LOCNUMSTRSIZE  11
char gcaLocalNumStr[LOCNUMSTRSIZE+1];
// ************************
char* TicksToTime(uint32_t u32Ticks)
// ************************
{  //  "65535H.59M.59S"
u32Ticks /= TICKS_PER_SECOND; // Ticks to Sekunden
uint16_t u16Hours = u32Ticks / 3600;
uint8_t u8Minutes = (u32Ticks- (u32Ticks / 3600)*3600 ) / 60;
uint8_t u8HSecs = u32Ticks % 60;
strcpy(gcaLocalNumStr, UIntToNumStr(u16Hours) );
strcat_P(gcaLocalNumStr, PSTR("H") );
strcat(gcaLocalNumStr, UIntToNumStr(u8Minutes) );
strcat_P(gcaLocalNumStr, PSTR("M") );
strcat(gcaLocalNumStr, UIntToNumStr(u8HSecs) );
strcat_P(gcaLocalNumStr, PSTR("S") );
return gcaLocalNumStr;
}
```

```
float Round(float fIn)
{
return floor(fIn+0.5);
};

//*****************************************************************
uint16_t ADC_10Bit(uint8_t ucChan)
//*****************************************************************
{
DDRC &= ~ _BV(ucChan);   //port A Chanel Bit Input
PORTC &= ~ _BV(ucChan); //Wegen Pullup abschalten
  // Activate ADC with Prescaler 16 --> 16Mhz/128 = 125 khz
ADCSRA = _BV(ADEN) | _BV(ADPS2) | _BV(ADPS1) |_BV(ADPS0);
 // Select pin ADC0 using MUX
ADMUX = ucChan;
    //Start conversion
ADCSRA |= _BV(ADSC);
while (ADCSRA & _BV(ADSC) ) // wait until converstion completed
            {
            wdt_reset();
            };
return ADC;// get converted value
};
```

```
#ifdef TOLERANZ_USED
// ****************************
int Toleranz(int32_t Val1, int32_t Val2, uint16_t Toleranz)
// ****************************
{  // 1 = 0K
  if (abs(Val1 - Val2) > Toleranz) { return 0; };
  return 1;
}
#endif
```

```
char*  IntToNumStr(int16_t lVal)
{
return itoa(lVal, gcaLocalNumStr, 10);
};

char* UIntToNumStr(uint16_t lVal)
{
return utoa(lVal, gcaLocalNumStr, 10);
};

char*  LongToNumStr(int32_t lVal)
{
return ltoa(lVal, gcaLocalNumStr, 10);
};

char* ULongToNumStr(uint32_t ulVal)
{
return ultoa(ulVal, gcaLocalNumStr, 10);
};
char*  FloatToNumStr(double fVal)
{//[-]d.ddd
return dtostrf(fVal, FLOAT_PRAEDIGITS_MAX, FLOAT_DIGITS_MAX, gcaLocalNumStr );
};
```

Spickzettel An dieser Stelle zum „schnell auffinden“

Binär-Hex-Dezimal Übersicht

2^BitNr = Wert

BYTE	Bit 7	BIT 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Hex :	0x**8**0	0x**4**0	0x**2**0	0x**1**0	0x0**8**	0x0**4**	0x0**2**	0x0**1**
Dez:	128	64	32	16	8	4	2	1

Bitposition

0x0**1** = 0000 000**1**
0x0**2** = 0000 00**1**0
0x0**4** = 0000 0**1**00
0x0**8** = 0000 **1**000
0x**1**0 = 000**1** 0000
0x**2**0 = 00**1**0 0000
0x**4**0 = 0**1**00 0000
0x**8**0 = **1**000 0000

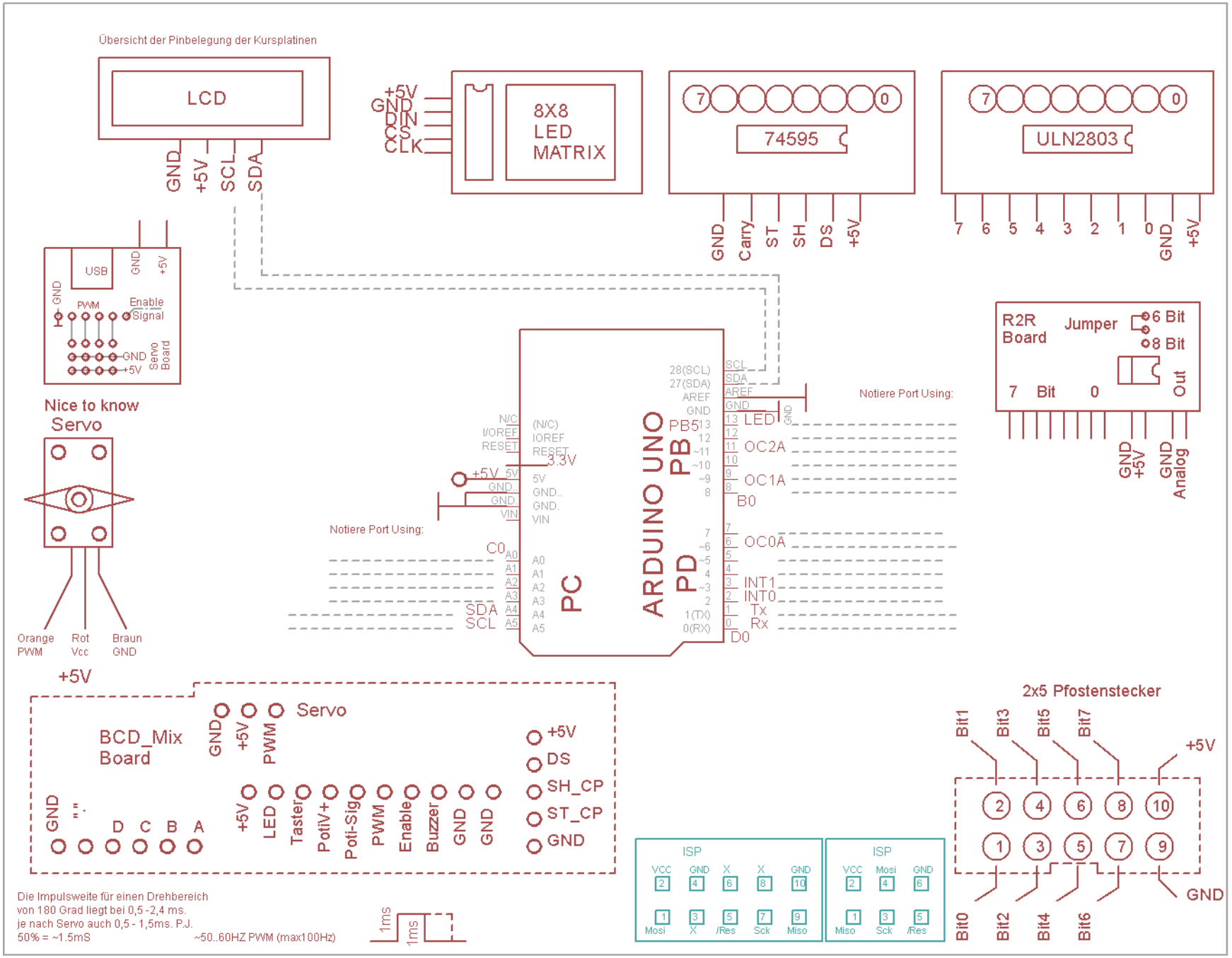
Dezimal	HEX	Binär
1	0x01	0000 0001
2	0x02	0000 0010
4	0x04	0000 0100
8	0x08	0000 1000
16	0x10	0001 0000
32	0x20	0010 0000
64	0x40	0100 0000
128	0x80	1000 0000

Bin	Hex
0000	0
000**1**	**1** 0x01
00**1**0	**2** 0x02
0011	3
0**1**00	**4** 0x04
0101	5
0110	6
0111	7
1000	**8** 0x08
1001	9
1010	A-10
1011	B-11
1100	C-12
1101	D-13
1110	E-14
1111	F-15

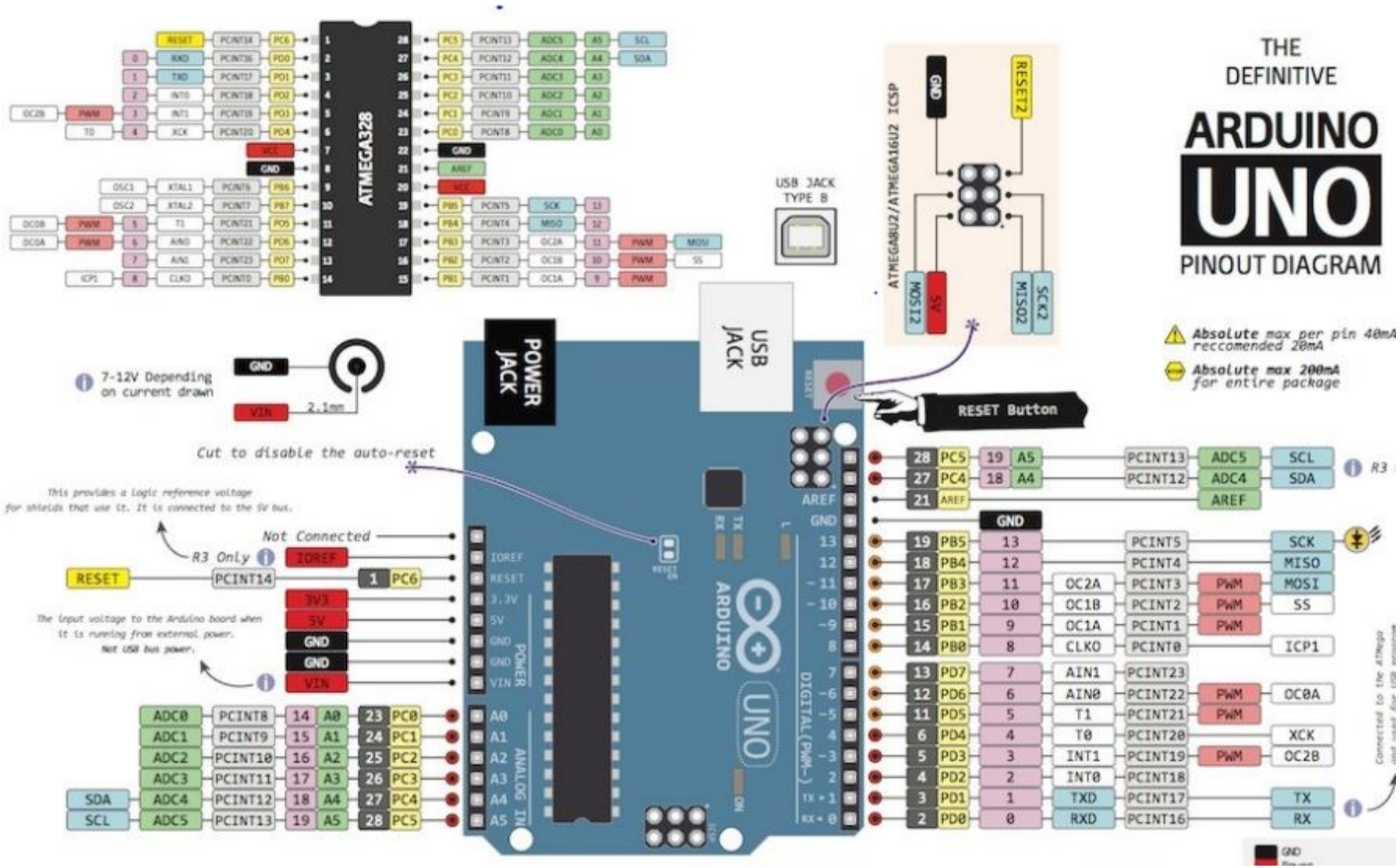
Dezimal:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15** = 16 Stellen
Hex:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**A**	**B**	**C**	**D**	**E**	**F** = 16 Stellen

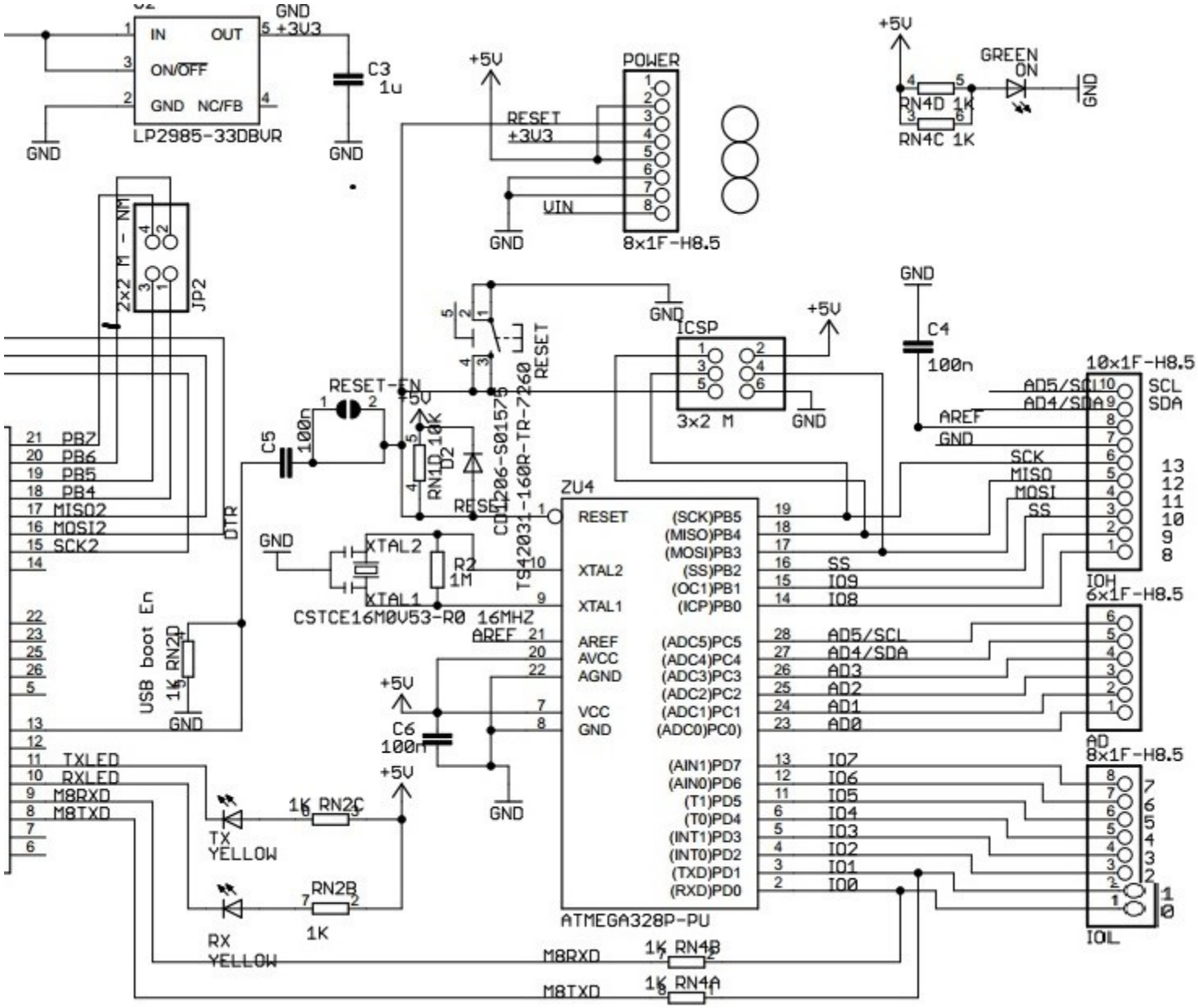
Übungsplatinen
Aussehen,
Pinbelgungen
Schaltplan, Auszüge
Kontaktbeschriftungen.

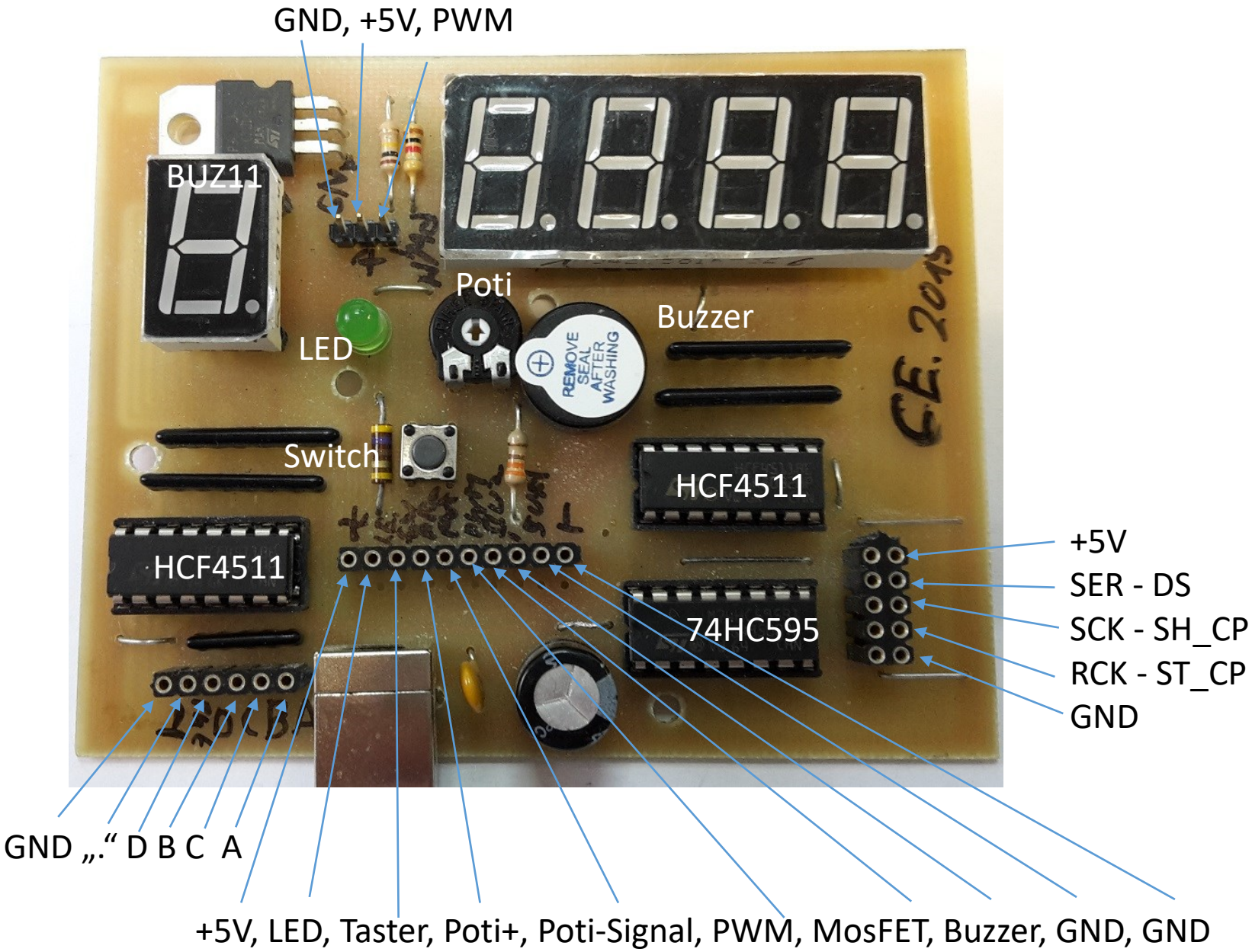
Übersicht der Pinbelegung der Kursplatinen

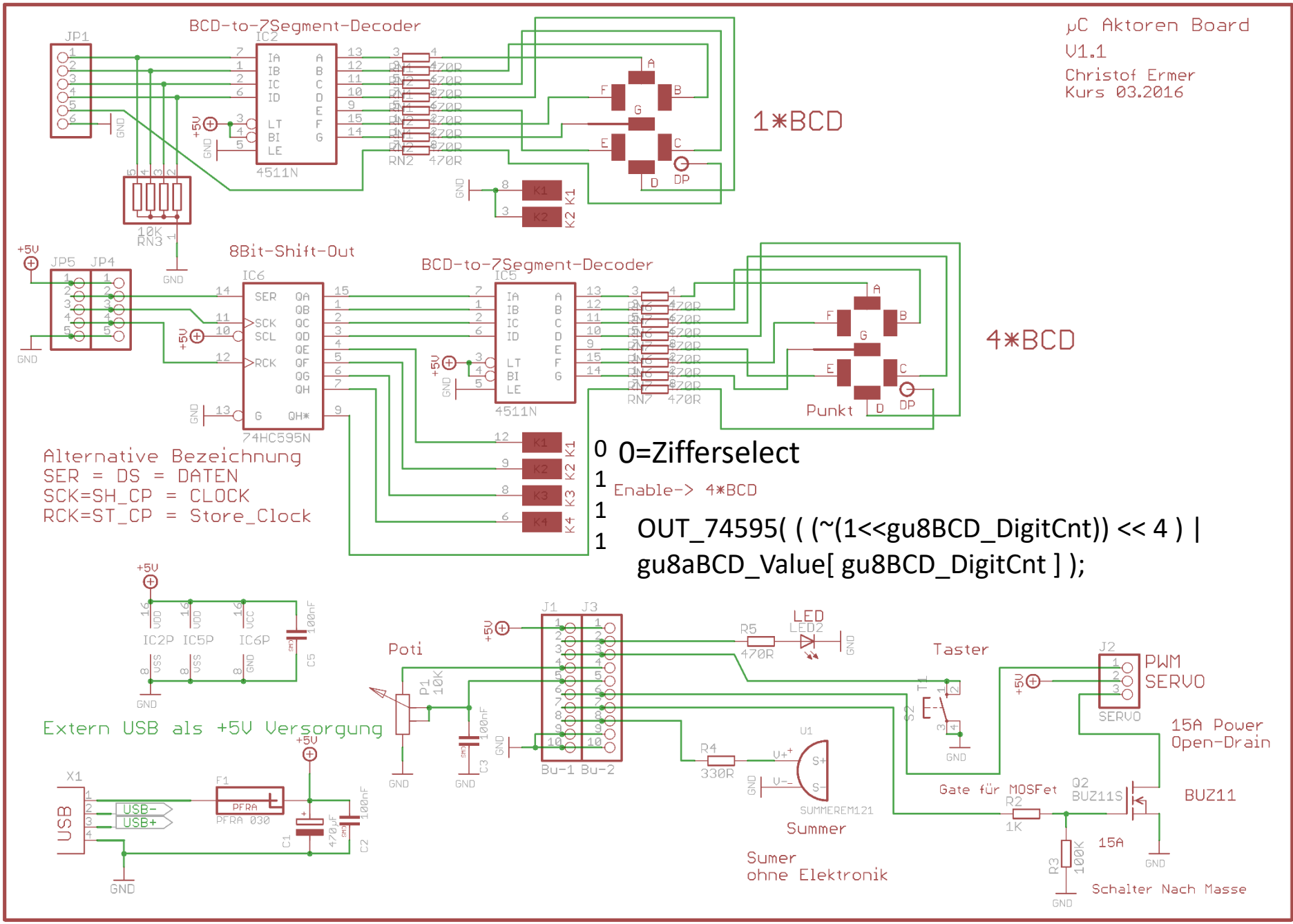


Die Impulsweite für einen Drehbereich
von 180 Grad liegt bei 0,5 -2,4 ms.
je nach Servo auch 0,5 - 1,5ms. P.J.
50% = ~1.5mS ~50..60HZ PWM (max100Hz)

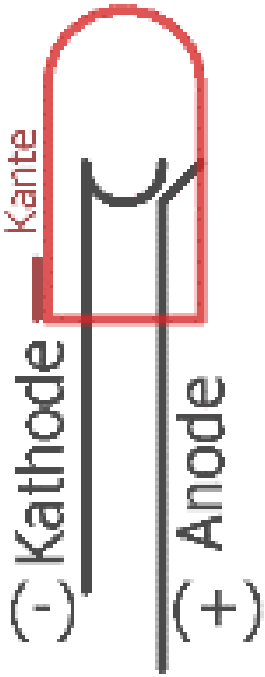

THE DEFINITIVE
ARDUINO UNO
PINOUT DIAGRAM
ATMEGA328
ATMEGA8U2/ATMEGA16U2 ICSP
USB JACK TYPE B
USB JACK
POWER JACK
7-12V Depending on current drawn
2.1mm
Cut to disable the auto-reset
RESET Button
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
This provides a logic reference voltage for shields that use it. It is connected to the 5V bus.
Not Connected
R3 Only
The input voltage to the Arduino board when it is running from external power. Not USB bus power.
ARDUINO UNO
DIGITAL (PWM~)
ANALOG IN
POWER


LP2985-33DBVR
IN
OUT
ON/OFF
GND
NC/FB
GND +3V3
C3 1u
POWER
8x1F-H8.5
RESET
+3V3
VIN
GREEN ON
RN4D 1K
RN4C 1K
JP2
2x2 M - NM
ICSP
3x2 M
C4 100n
10x1F-H8.5
AD5/SCL
AD4/SDA
AREF
SCK
MISO
MOSI
SS
IOH
6x1F-H8.5
AD
8x1F-H8.5
IOL
RESET-EN
C5 100n
RN1D 10K
D2
CD1206-S01575
TS42031-160R-TR-7260
ZU4
XTAL2
XTAL1
R2 1M
CSTCE16M0V53-R0 16MHZ
USB boot En
RN2D 1K
DTR
C6 100n
TXLED
RXLED
M8RXD
M8TXD
TX YELLOW
RX YELLOW
1K RN2C
RN2B 1K
1K RN4B
1K RN4A
ATMEGA328P-PU
(SCK)PB5
(MISO)PB4
(MOSI)PB3
(SS)PB2
(OC1)PB1
(ICP)PB0
(ADC5)PC5
(ADC4)PC4
(ADC3)PC3
(ADC2)PC2
(ADC1)PC1
(ADC0)PC0
(AIN1)PD7
(AIN0)PD6
(T1)PD5
(T0)PD4
(INT1)PD3
(INT0)PD2
(TXD)PD1
(RXD)PD0
AD5/SCL
AD4/SDA
AD3
AD2
AD1
AD0
IO7
IO6
IO5
IO4
IO3
IO2
IO1
IO0
IO9
IO8


GND, +5V, PWM
BUZ11
Poti
Buzzer
LED
Switch
HCF4511
HCF4511
74HC595
+5V
SER - DS
SCK - SH_CP
RCK - ST_CP
GND
GND „.“ D B C A
+5V, LED, Taster, Poti+, Poti-Signal, PWM, MosFET, Buzzer, GND, GND

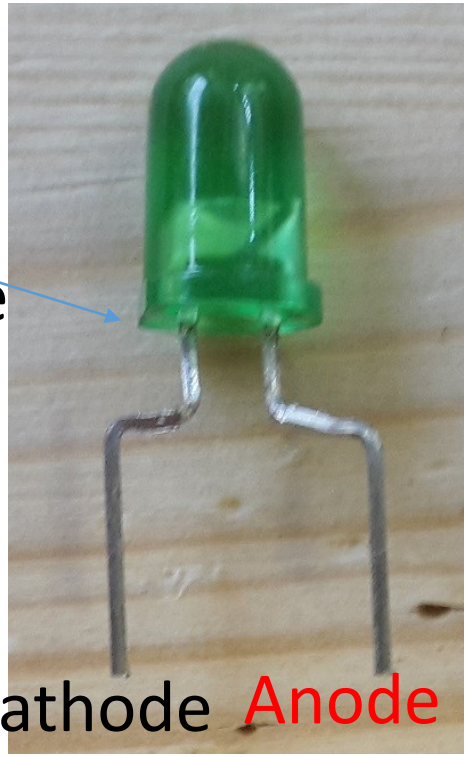

µC Aktoren Board
V1.1
Christof Ermer
Kurs 03.2016
BCD-to-7Segment-Decoder
IC2
4511N
JP1
RN2 470R
RN3 10K
1*BCD
8Bit-Shift-Out
IC6
74HC595N
JP5 JP4
BCD-to-7Segment-Decoder
IC5
4511N
RN7 470R
Punkt
4*BCD
Alternative Bezeichnung
SER = DS = DATEN
SCK=SH_CP = CLOCK
RCK=ST_CP = Store_Clock
0 0=Zifferselect
1 Enable-> 4*BCD
1
1
OUT_74595(((~(1<<gu8BCD_DigitCnt)) << 4) |
gu8aBCD_Value[gu8BCD_DigitCnt]);
IC2P IC5P IC6P
C5 100nF
Extern USB als +5V Versorgung
X1
USB
USB-
USB+
F1
PFRA
PFRA 030
C1 470µF
C2 100nF
Poti
P1 10K
C3 100nF
J1 J3
Bu-1 Bu-2
R5 470R
LED
LED2
Taster
T1
S2
R4 330R
U1
SUMMEREM121
Summer
Sumer
ohne Elektronik
J2
PWM
SERVO
15A Power
Open-Drain
Gate für MOSFet
R2 1K
Q2
BUZ11S
BUZ11
R3 100K
15A
Schalter Nach Masse

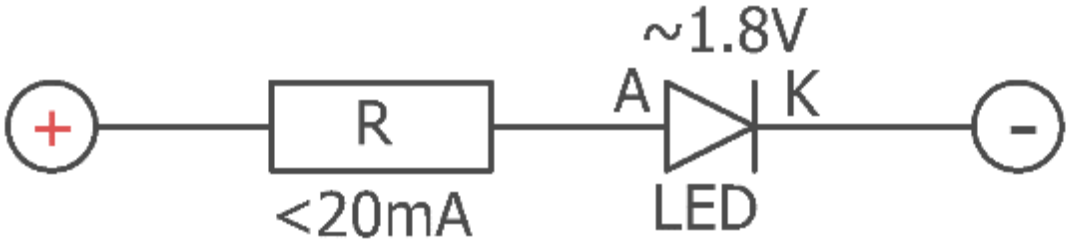
Vorbereitung Steckbrett mit LED **und** Widerstand



Berechnung des Vorwiderstandes.
Teile zusammensuchen und vorbereiten.
LED Anschlüsse **nicht am Gehäuse** biegen. ZANGE VERWENDEN und an Raster ausrichten. (Hinhalten abschätzen, kürzen „>=1cm“ einstecklänge.
Nicht kürzer, sonst langt es nicht

Rasterabstand beachten

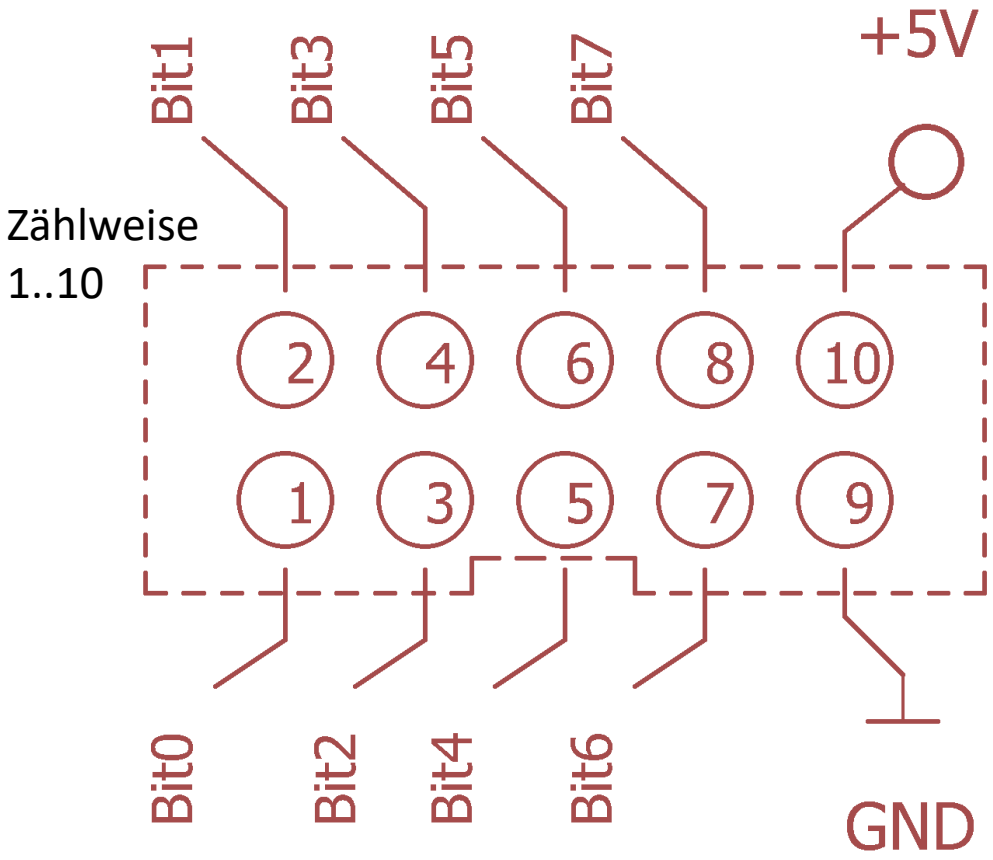


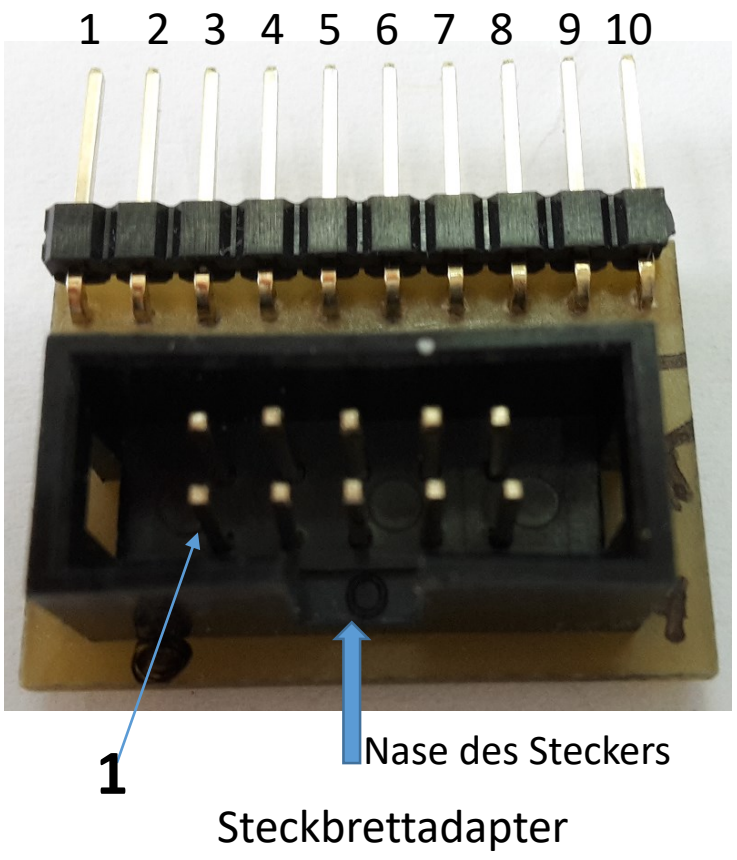


R=U/I = (5-1,8)V / 0.01A = 320--> ~330R

470R bis 560R tun es auch

Pinbelegung eines Pfostensteckers für Flachbandleitungen.





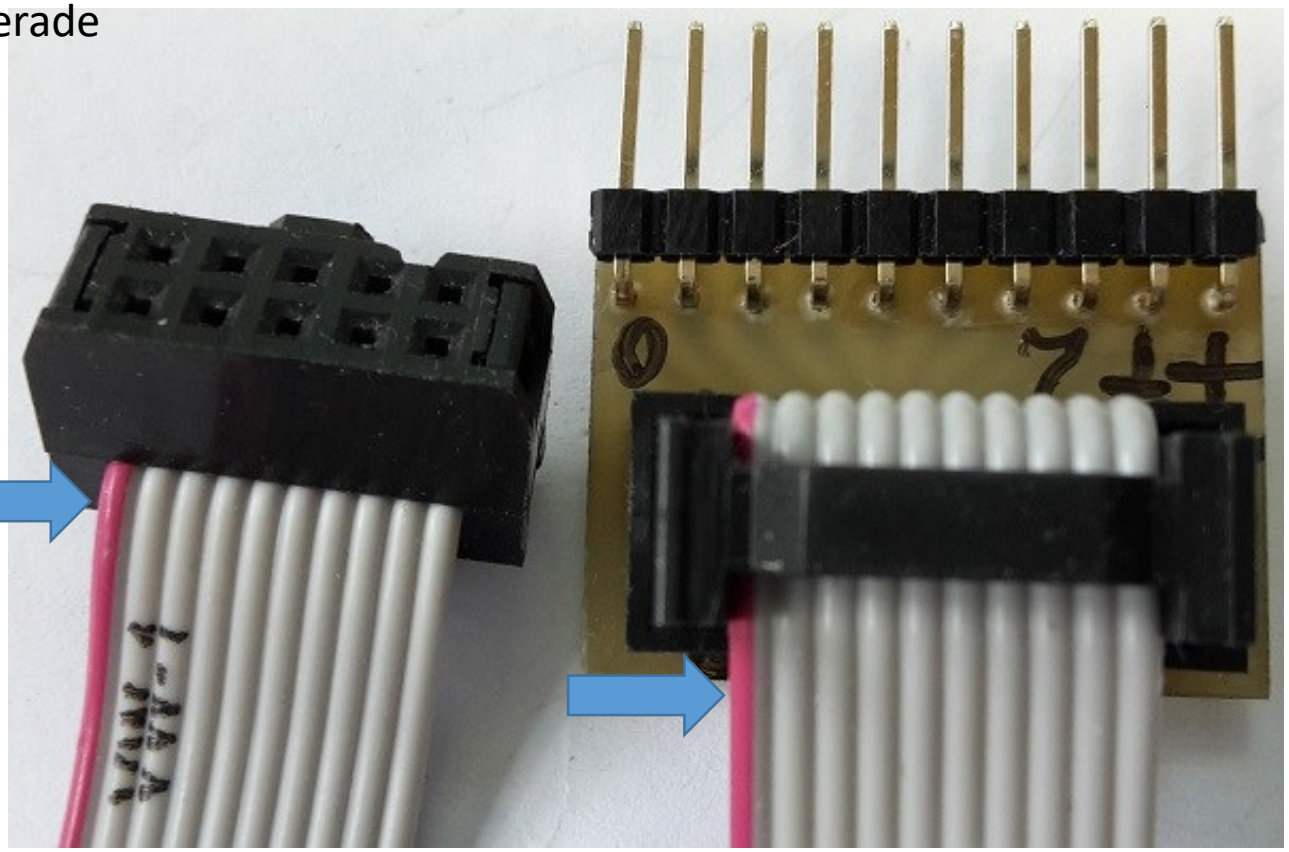
10er Flachbandkabel für 2x5 Postenstecker mit Steckbrettadapter

Pins –
2x5 Stiftleiste und
Stiftleiste gerade

1= Bit 0
2= Bit 1
3= Bit 2
4= Bit 3
5= Bit 4
6= Bit 5
7= Bit 6
8= Bit 7

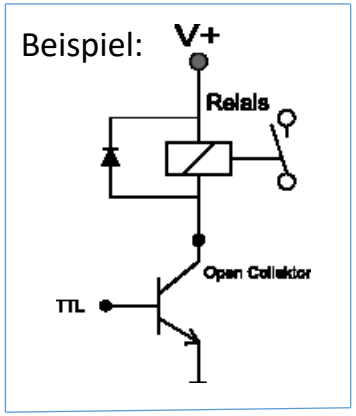
9= GND

10= +5V

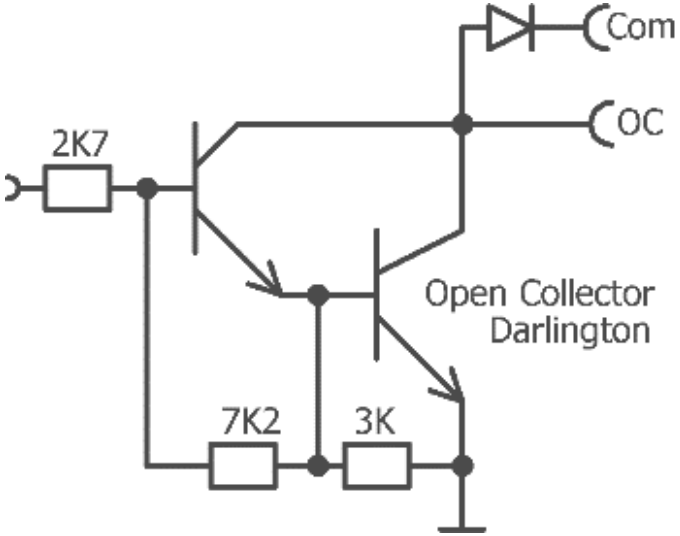


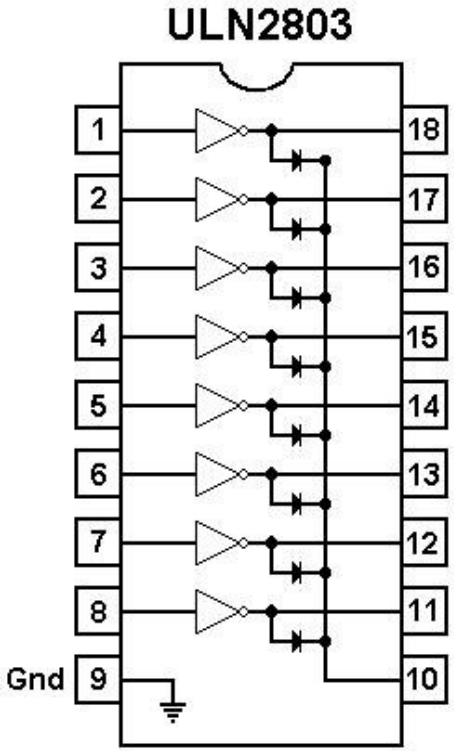
Ein beliebter **Transistor Array Baustein** **ULN2803**

ULN2803 Transistor Array IC
„Open Collector“ Hier **mit 8 Darlington Transistoren** und 8 **integrierten Freilauf-Dioden** an einem gemeinsamen **Common** Anschluss



Damit kann man 8 Bauteile, die
1.) hohen Strom fordern (<=500mA) und
2.) die an V+ liegen mit einem „s**chwachen**“ µC Steuersignal auf GND, und somit AN schalten.



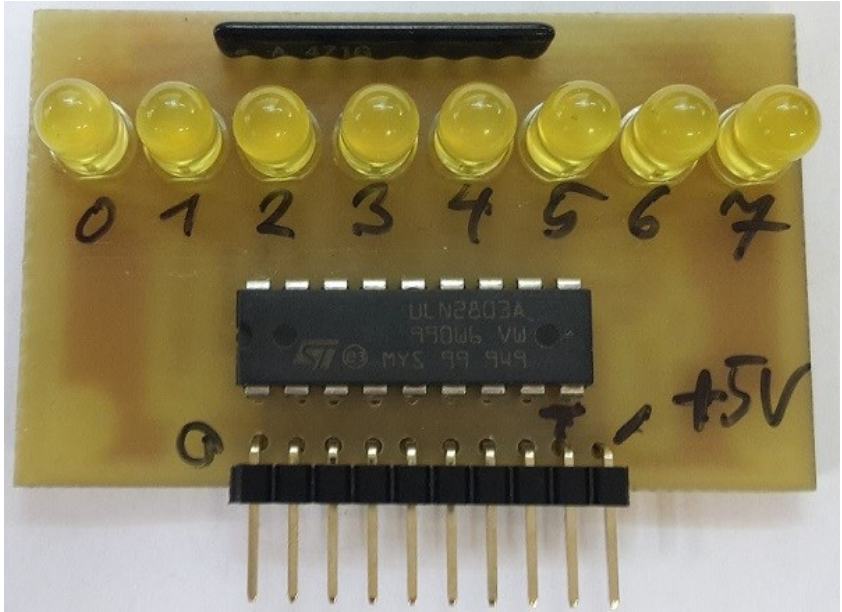


LED-Anzeige für ein Byte

1= Bit 0
2= Bit 1
3= Bit 2
4= Bit 3
5= Bit 4
6= Bit 5
7= Bit 6
8= Bit 7

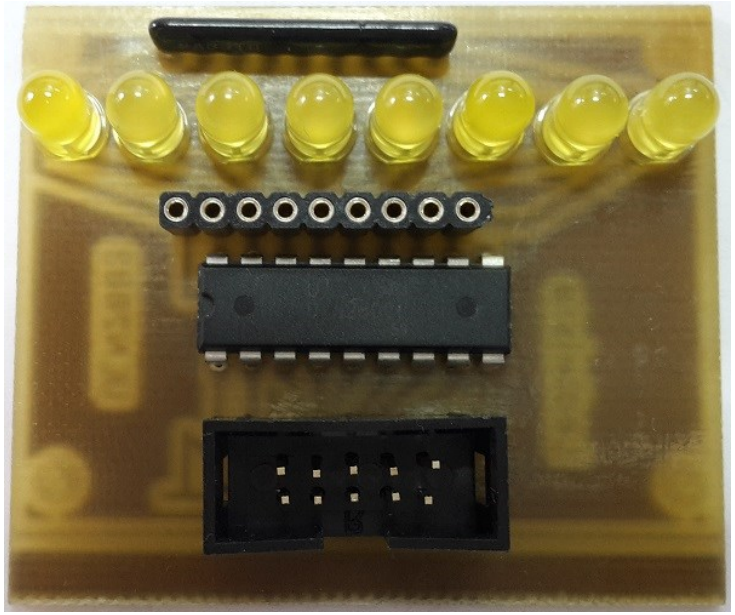
9= GND

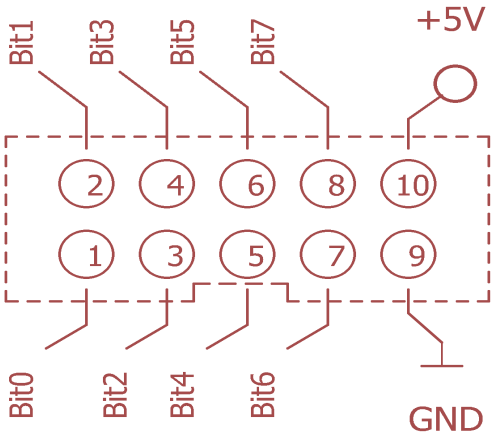
10= +5V



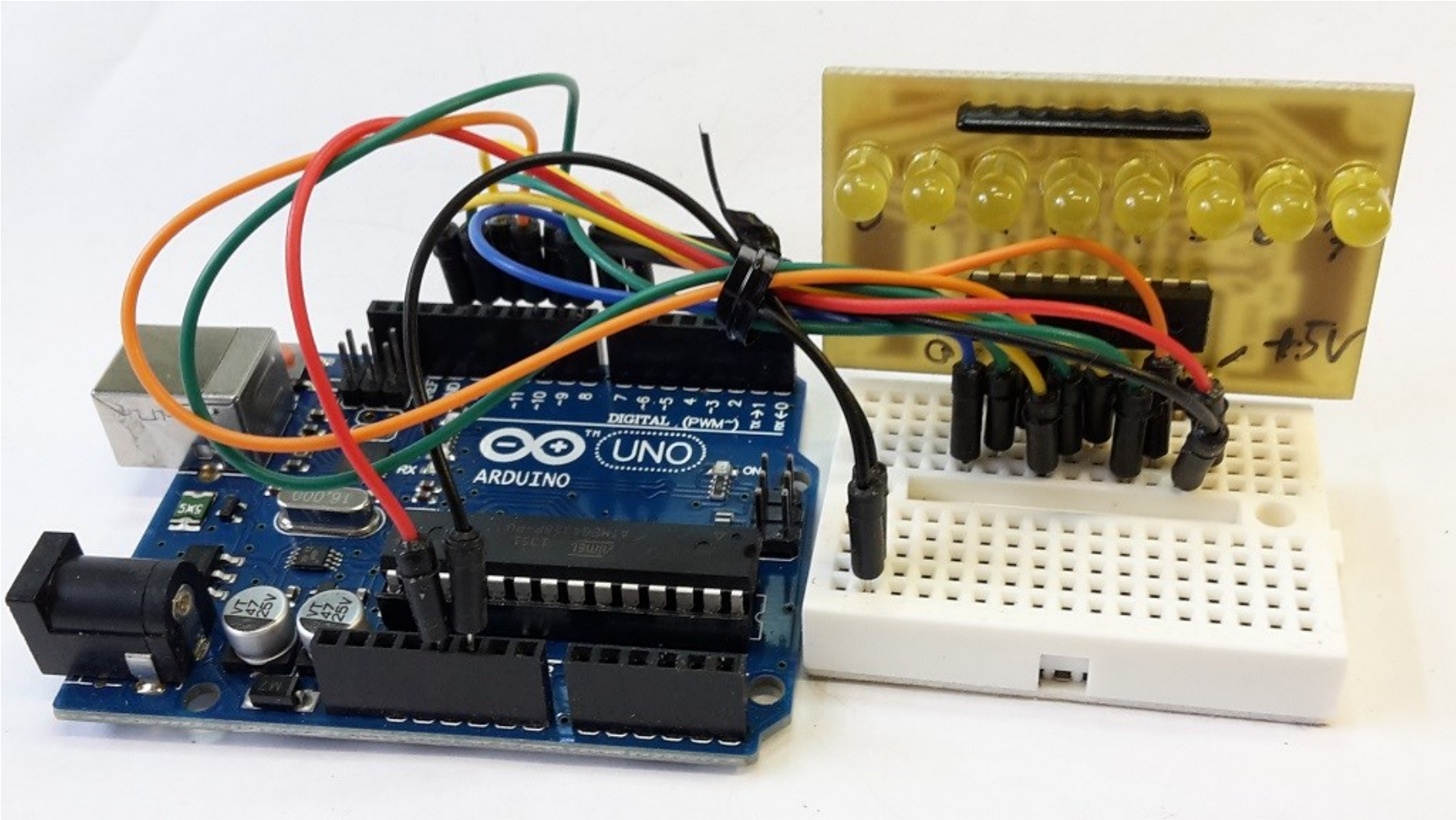
Pin 1 2 3 4 5 6 7 8 9 10

G N D +5V

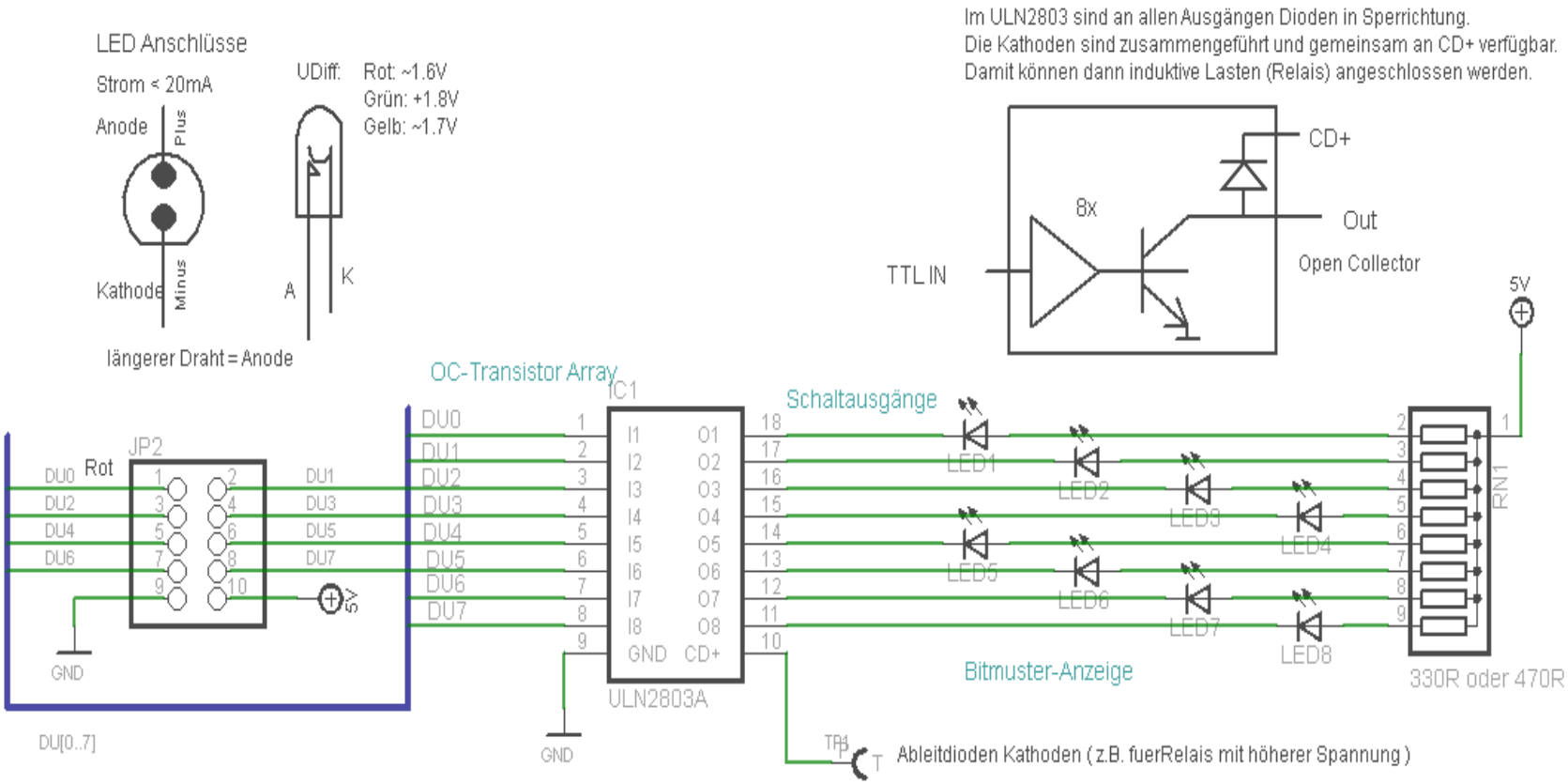




8x LEDs an Arduino mit gemischter Portnutzung.
Beachte GND an PIN 9 und +5V an PIN 10 „richtig“ anschließen

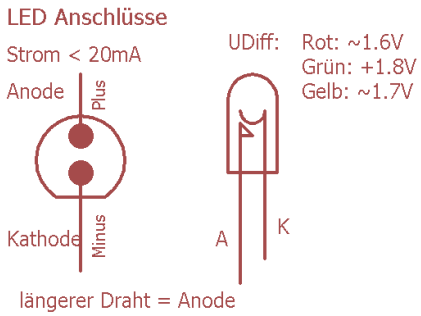


Schaltung für 8 LED mit dem 8x Open Collector Leistungstreiber **ULN2803A**

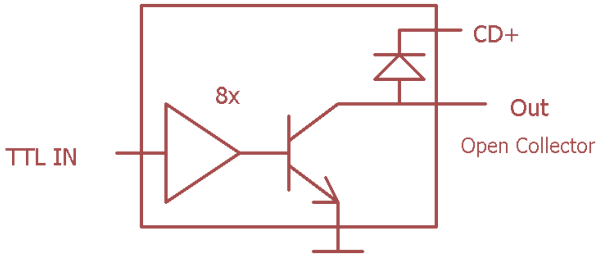


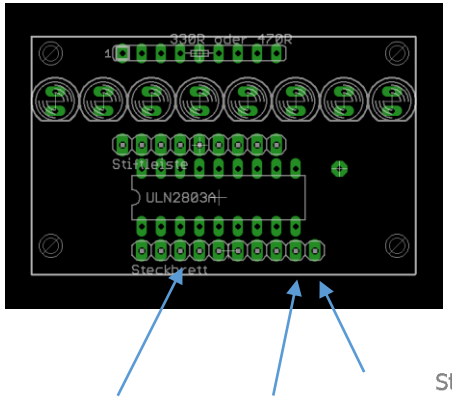
Hier mit 10 poligen
Flachband Stecker

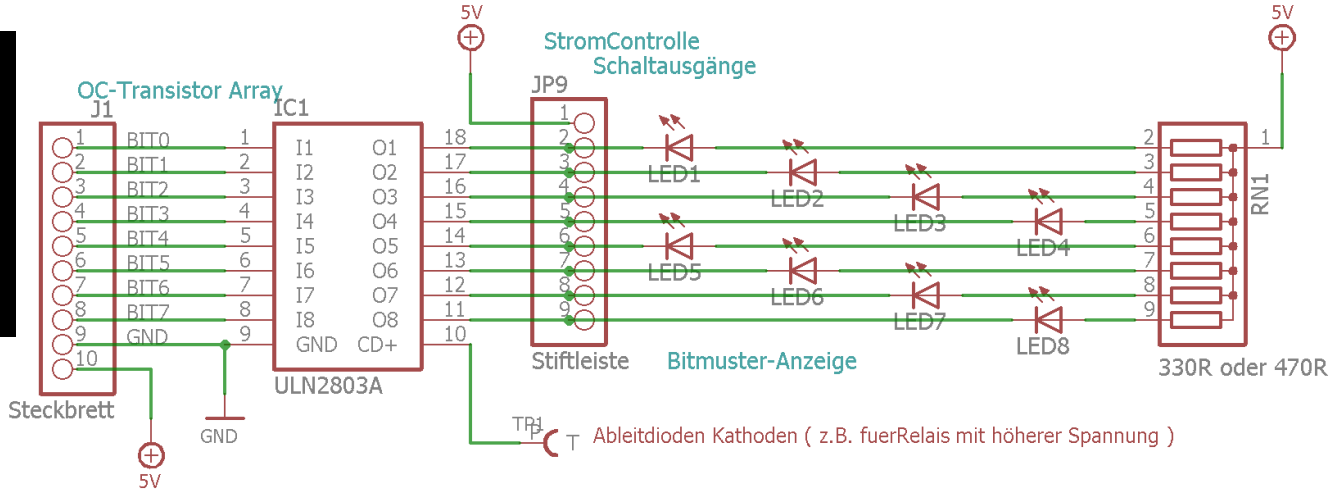
8x LED Leiste Steckbrett Version



Im ULN2803 sind an allen Ausgängen Dioden in Sperrichtung.
Die Kathoden sind zusammengeführt und gemeinsam an CD+ verfügbar.
Damit können dann induktive Lasten (Relais) angeschlossen werden.







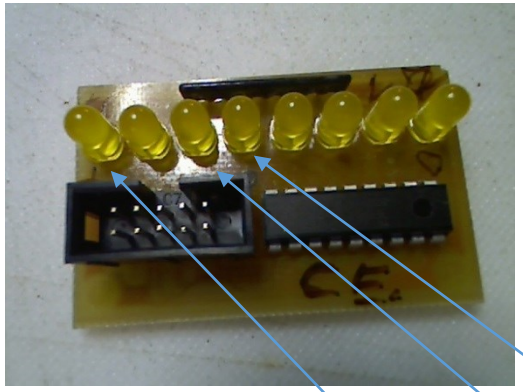
D0…D7 , GND, +5V

Hier mit geradem
Stiftleistenstecker

...weil es ab jetzt hilfreich sein kann.

Einfache Analyse und Fehlersuch/Debug Methoden

Es gibt selbstverständlich gute Debug-Techniken **wie JTAG**. Leider erfordert dies dazu fähige **Extra-Hardware** (Dragonboard etc.), und einen **µC** der eine **JTAG** Schnittstelle hat, und/oder ein **1-Wire Debug,** und eine Software der man sich unterwerfen muss, *und..und..und..* Und es geht dennoch eher zäh! Es geht auch mit Methode und Hirn, und oft viel einfacher und besser…..:



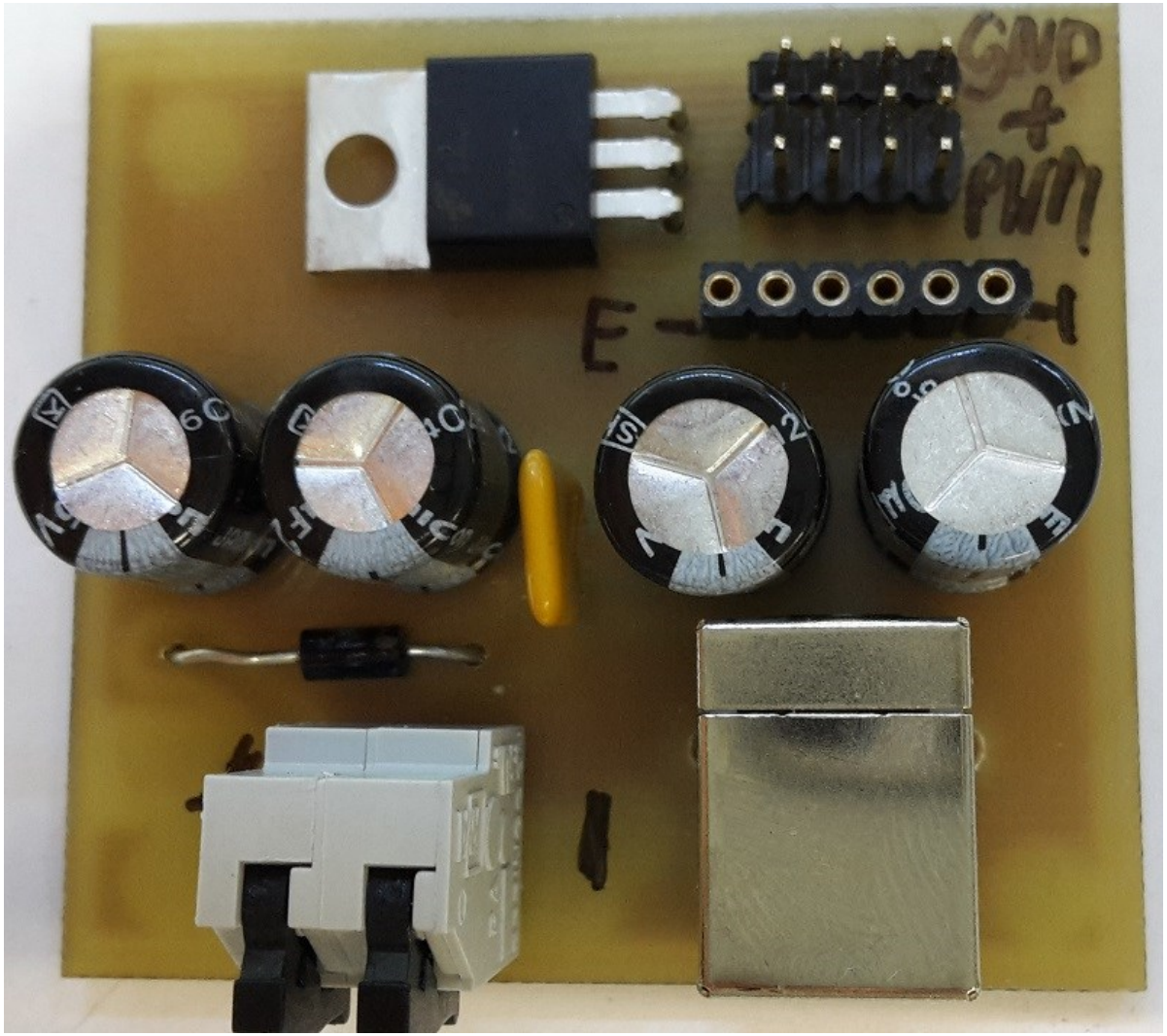
Meine Lieblingsmethode ist eine einfache **8-Bit LED Leiste**.
Ab sofort kann **Online in Echtzeit** jedes Byte angezeigt und somit überwacht werden
PORTx (LEDPort) = **MCUCR**;
oder PORTx = g**u8StatusFlag**s;
oder PORTx = (1<< (u8NN++) % 8); //Lauflicht

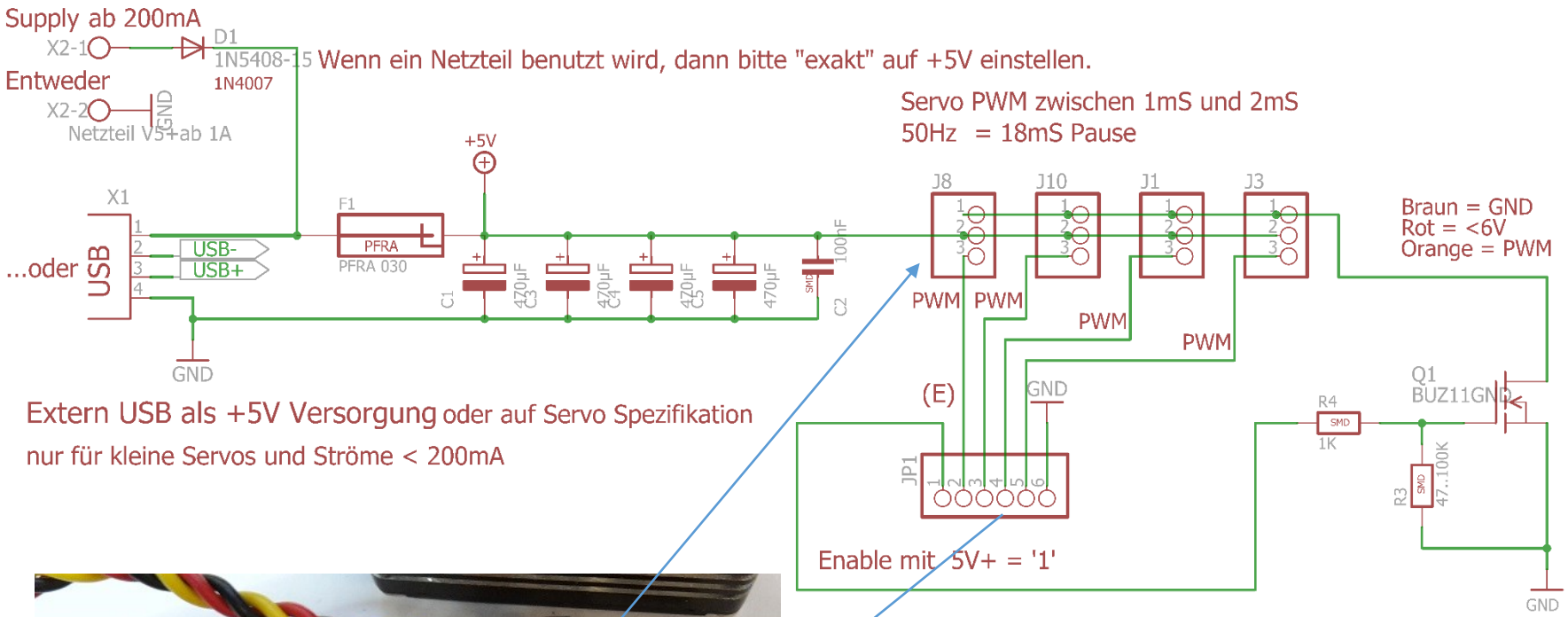
MCU Control Register – MCUCR

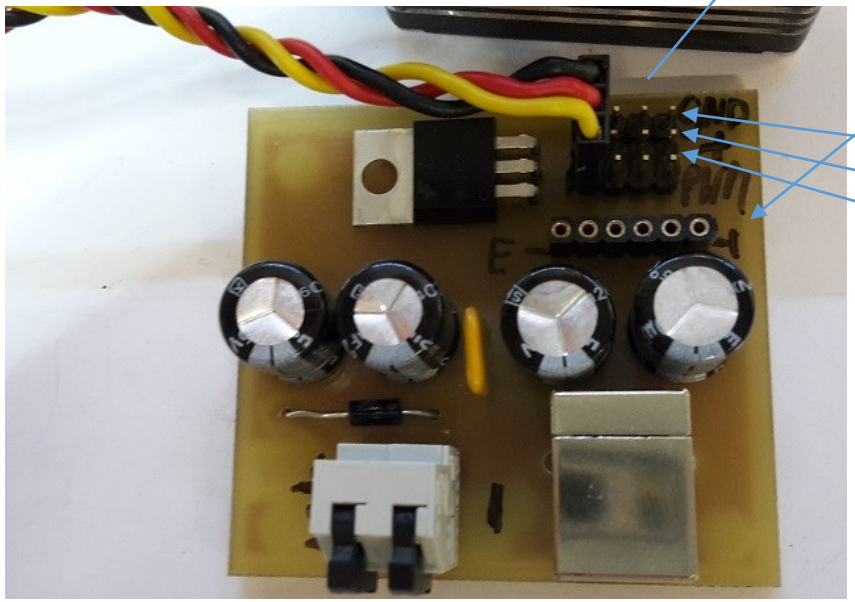
The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

Bit	7	6	5	4	3	2	1	0	
	SM2	SE	SM1	SM0	ISC11	ISC10	ISC01	ISC00	MCUCR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

4fachPWM-Servo Platine mit Enable

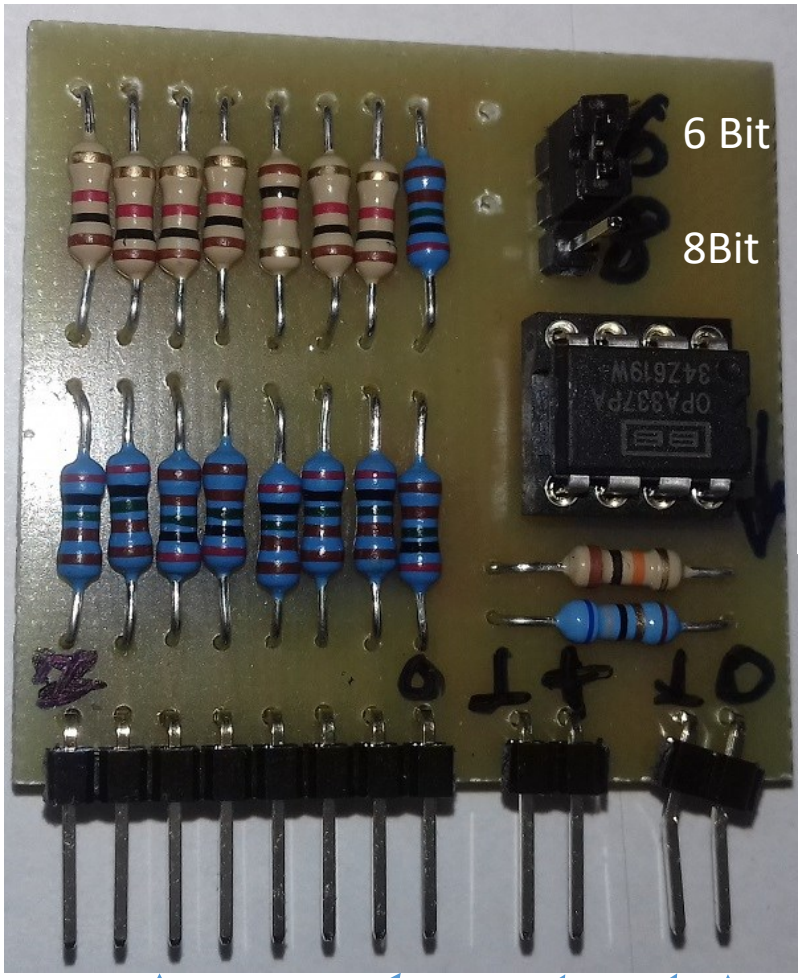


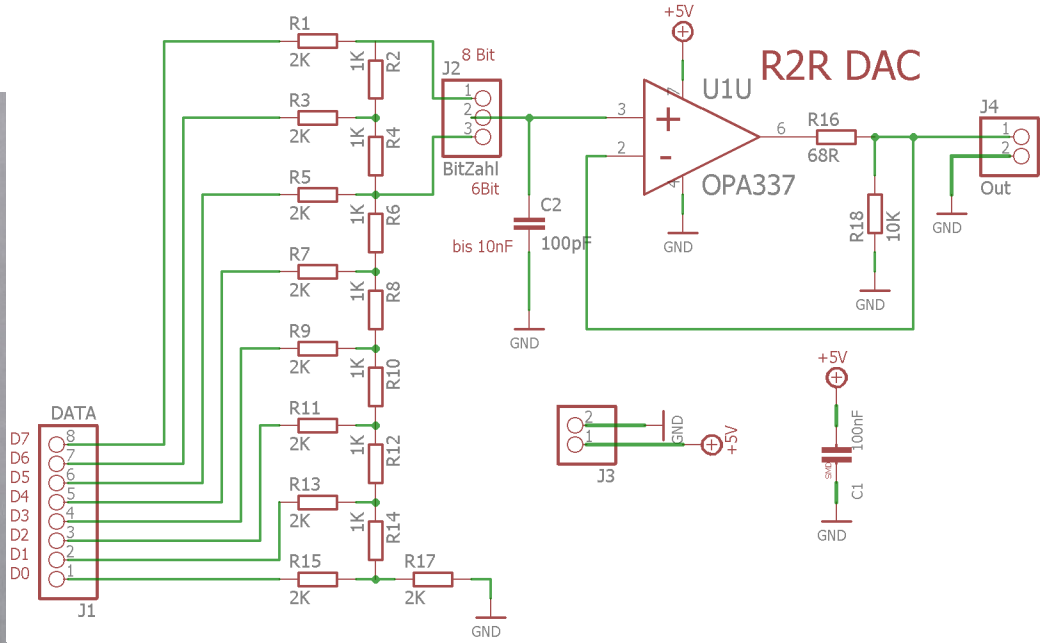

Supply ab 200mA
X2-1
D1
1N5408-15
1N4007
Entweder
X2-2
GND
Netzteil V5+ab 1A
Wenn ein Netzteil benutzt wird, dann bitte "exakt" auf +5V einstellen.
Servo PWM zwischen 1mS und 2mS
50Hz = 18mS Pause
+5V
X1
...oder
USB
USB-
USB+
F1
PFRA
PFRA 030
C1
470µF
C3
470µF
C4
470µF
C5
470µF
100nF
C2
J8
J10
J1
J3
Braun = GND
Rot = <6V
Orange = PWM
PWM
PWM
PWM
PWM
(E)
GND
JP1
Q1
BUZ11GND
R4
1K
R3
47..100K
Enable mit 5V+ = '1'
GND
Extern USB als +5V Versorgung oder auf Servo Spezifikation
nur für kleine Servos und Ströme < 200mA

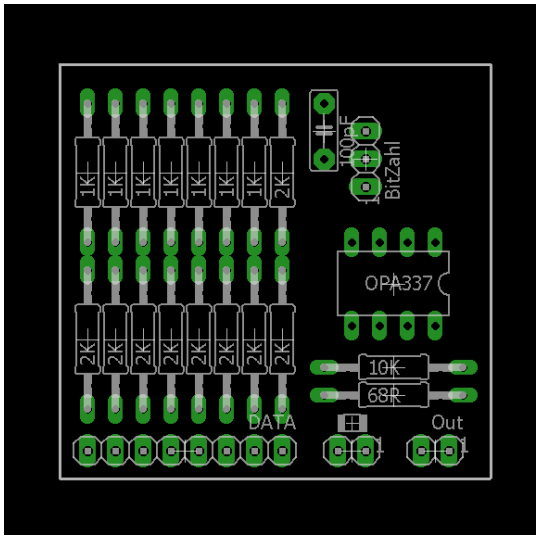


GND
V+
PWM

DAC mit R2R 2017

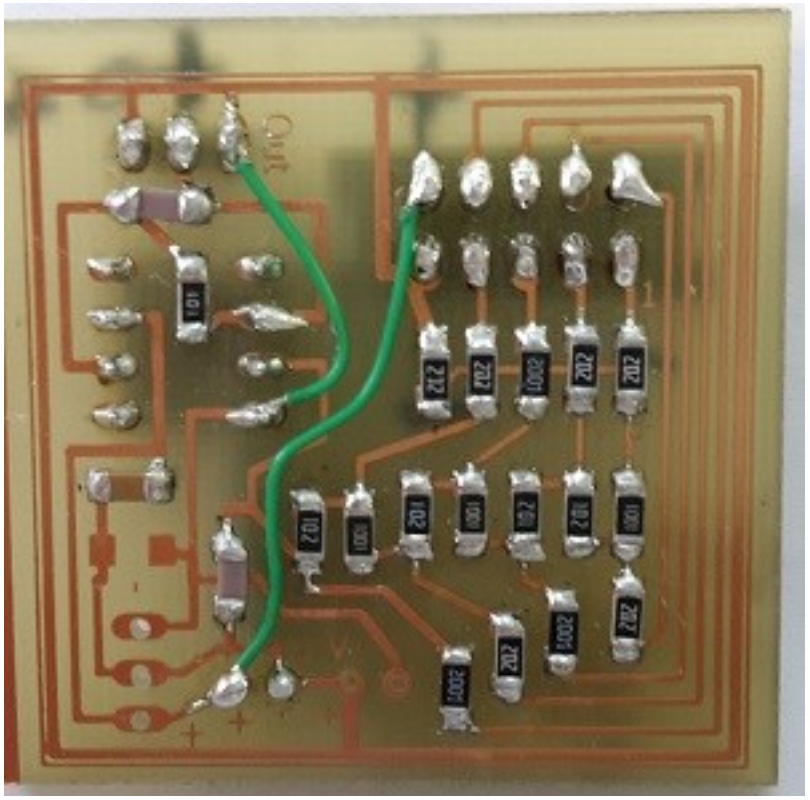


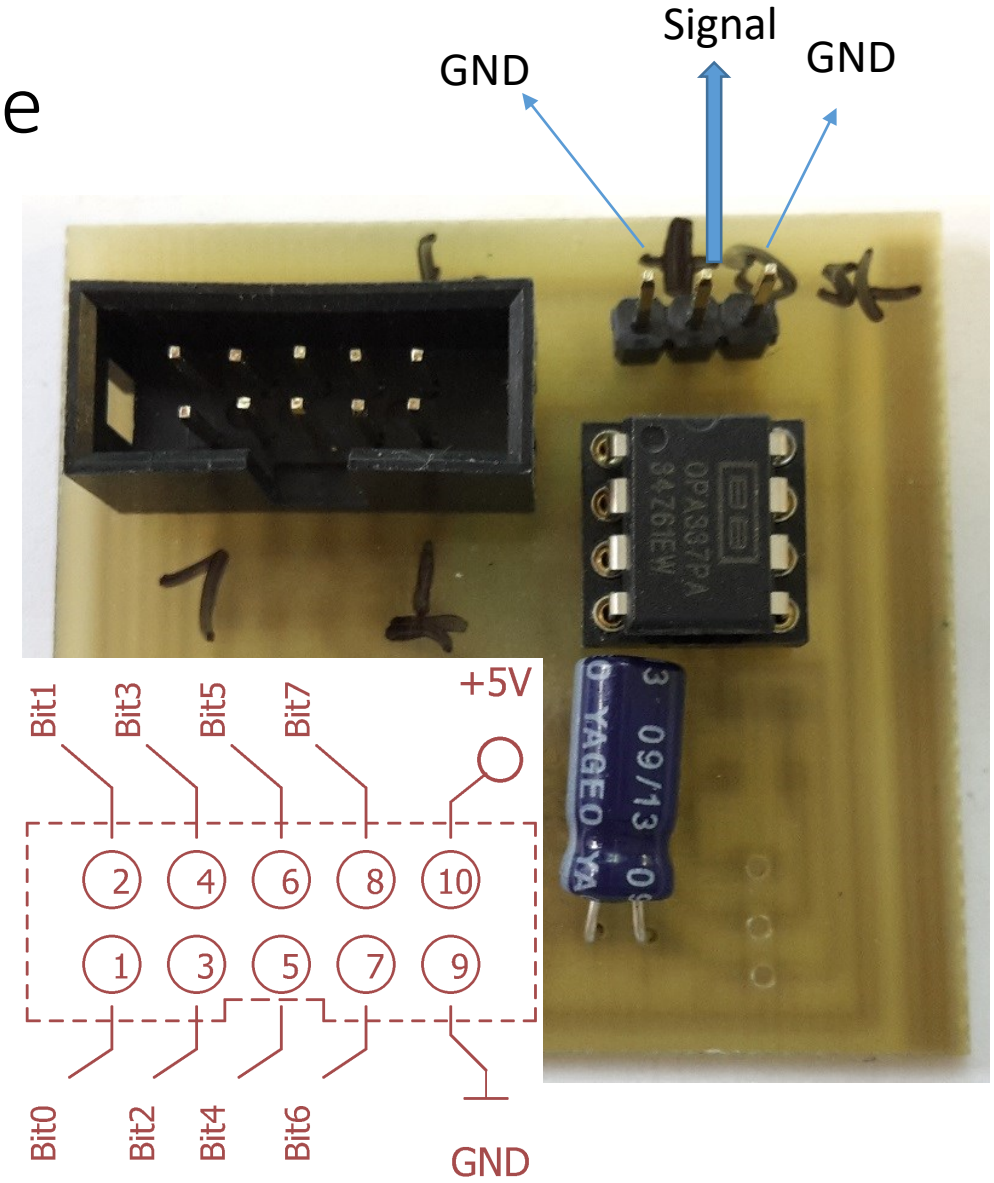




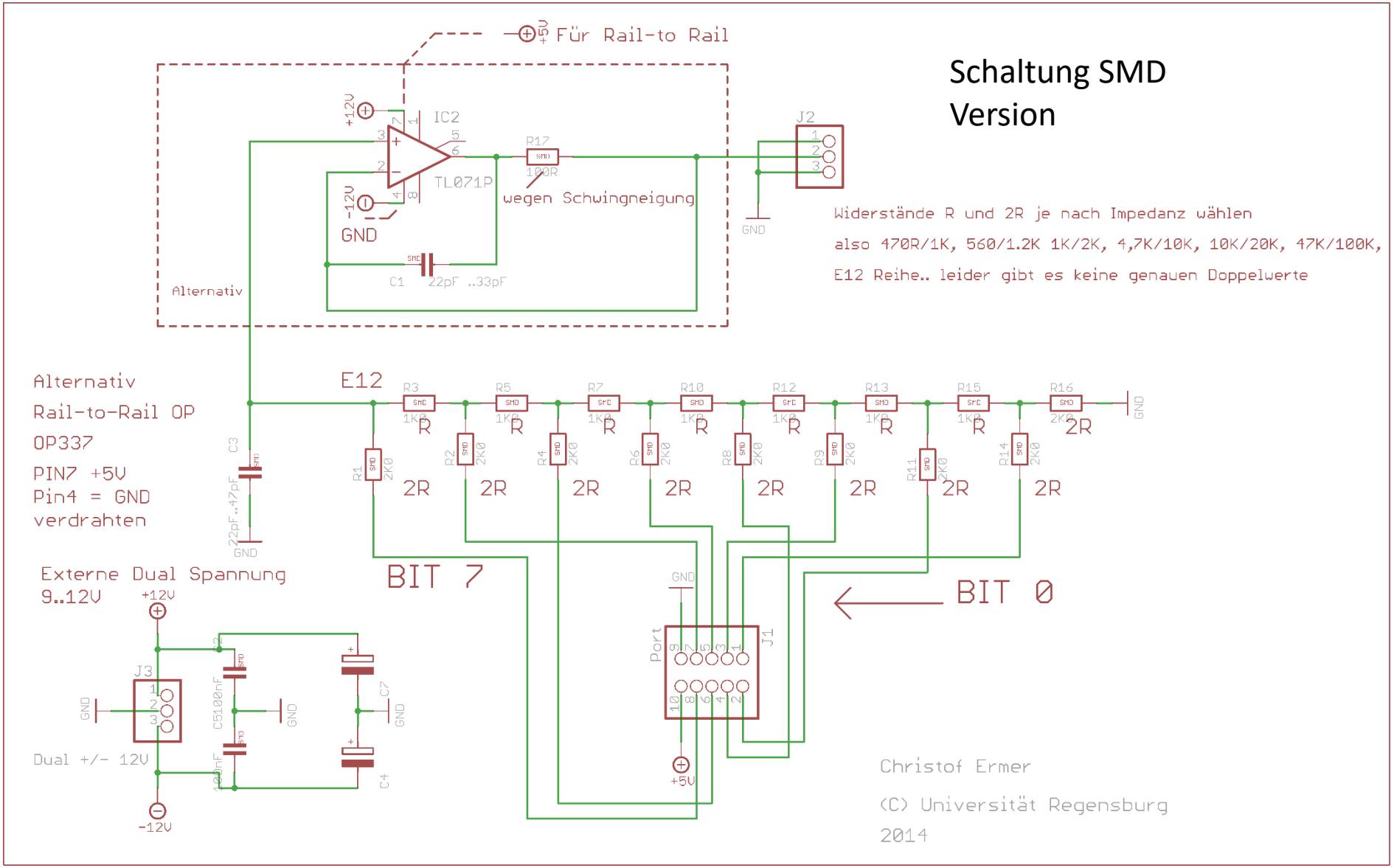
D7..D5.....D0, GND,+5V, GND, Analog

8Bit R2R- DAC Platine



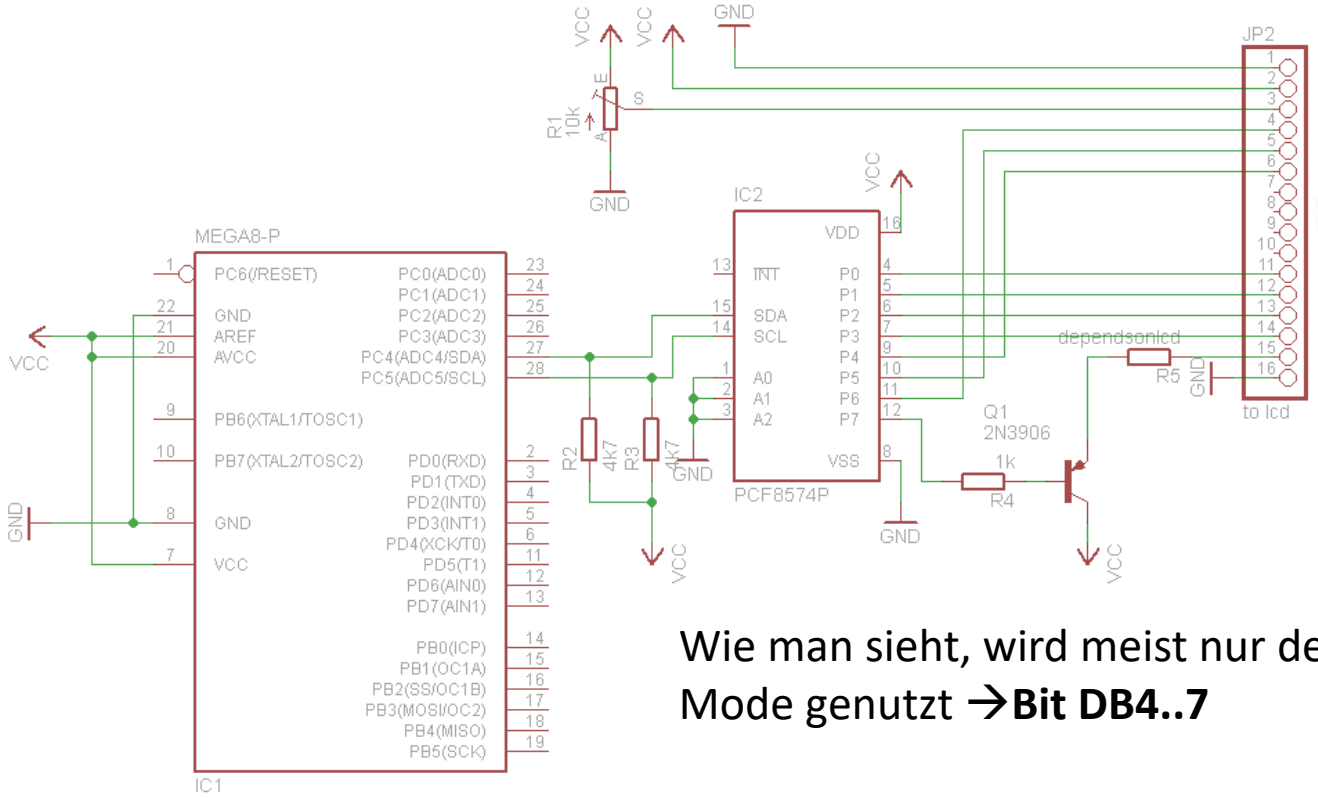


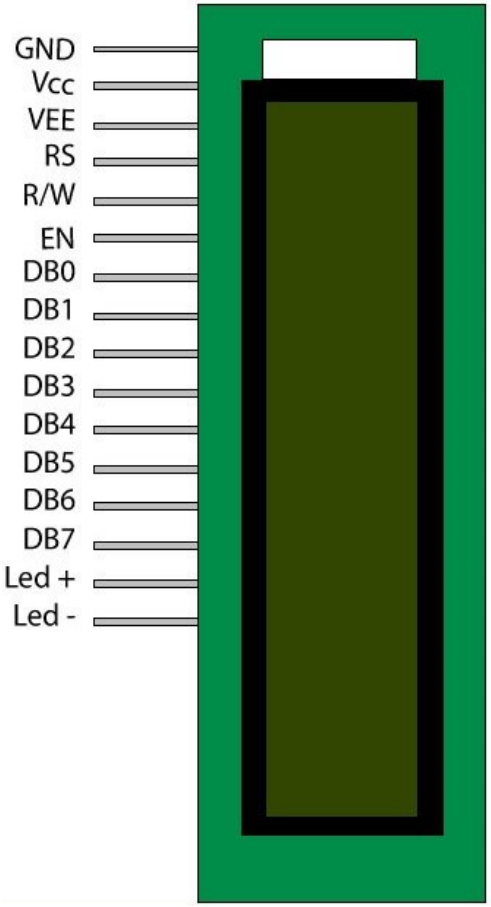
Schaltung SMD Version



I2C-LCD Display

Das I2C-IC **PCF8574** wird auch hier verwendet.
Es stellt die Daten I-O parallel zu Verfügung.
Eine freie Adresse (A0..2) auswählen und in der Software eintragen. Beachte, dass dies eine **7 Bit Adresse** ist →
(Adr << 1),.....weil 8 Bit Übergabe.



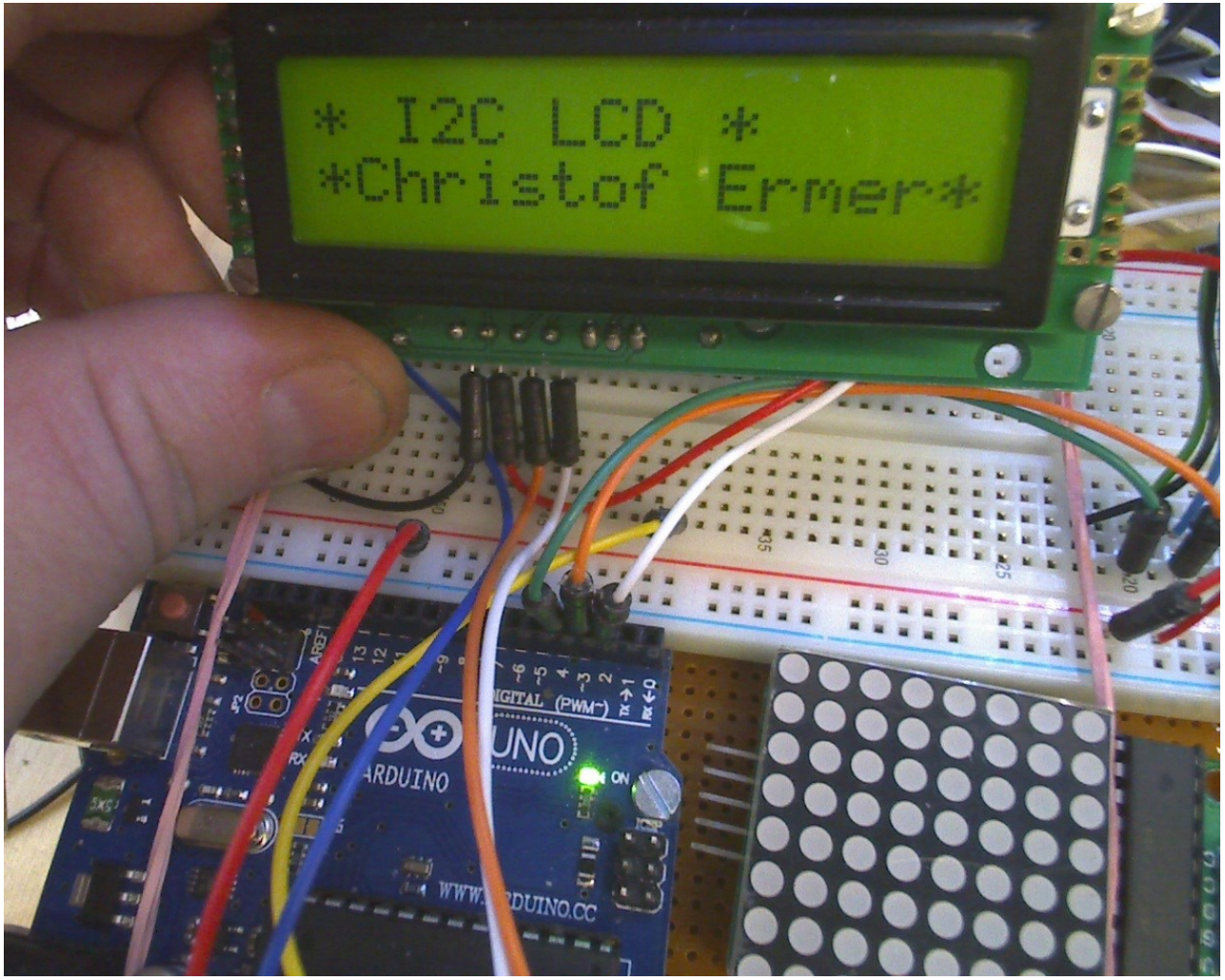


Wie man sieht, wird meist nur der 4bit Mode genutzt →**Bit DB4..7**

I2C LCD + 8x8 Matrix mit MAX7219

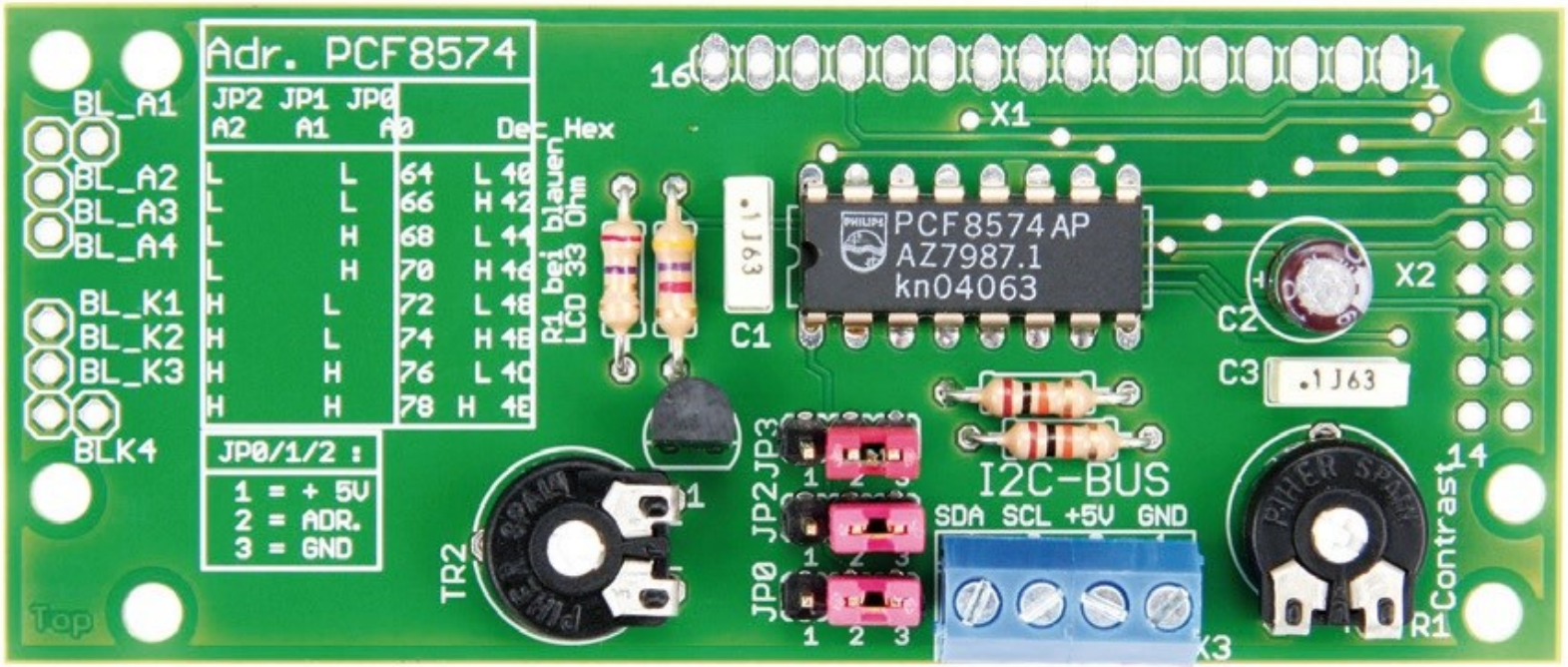
Software: 93-I2C-LCD-Modul_PCF8574 MAX7219

Youtube: http://youtu.be/KoqKPxGFp78



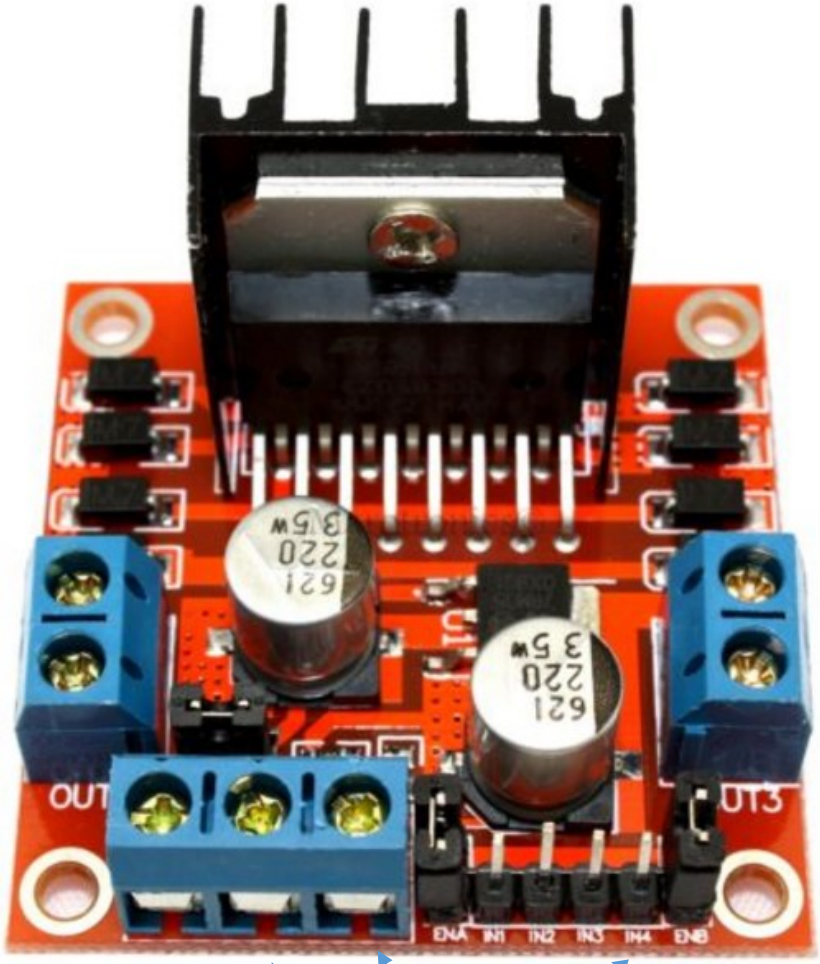
I2C Modul mit PCF8574 für Standard LCD Display

Es bietet eine Montagemöglichkeit für 1-Reihen und Doppelreihen LCD.
TIPP: Achte bei Doppelreihen LCD auf die Vertauschung: Pin 1=+5V, Pin2=GND



SDA, SCL, +5V, GND

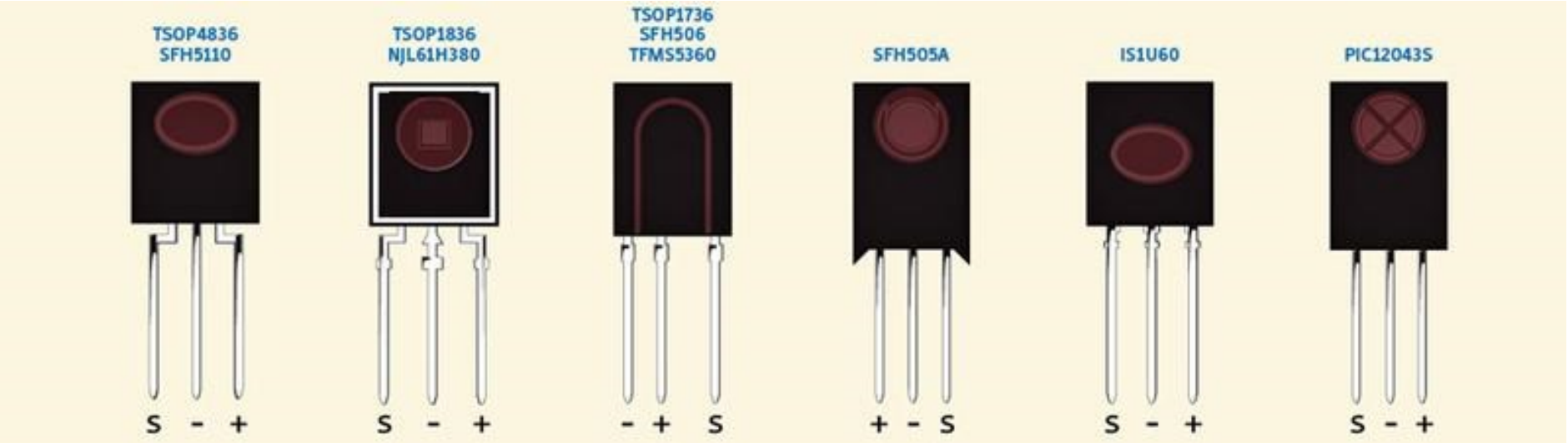
Neueres L298N H-Brückenmodel (China)



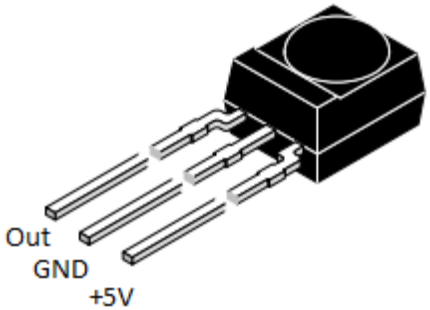
Versorgung: 8..~24V
VORSICHT: Verlustleistung beachten (Kühlkörper ist klein).
5V Klemme = Output !! KienEingang

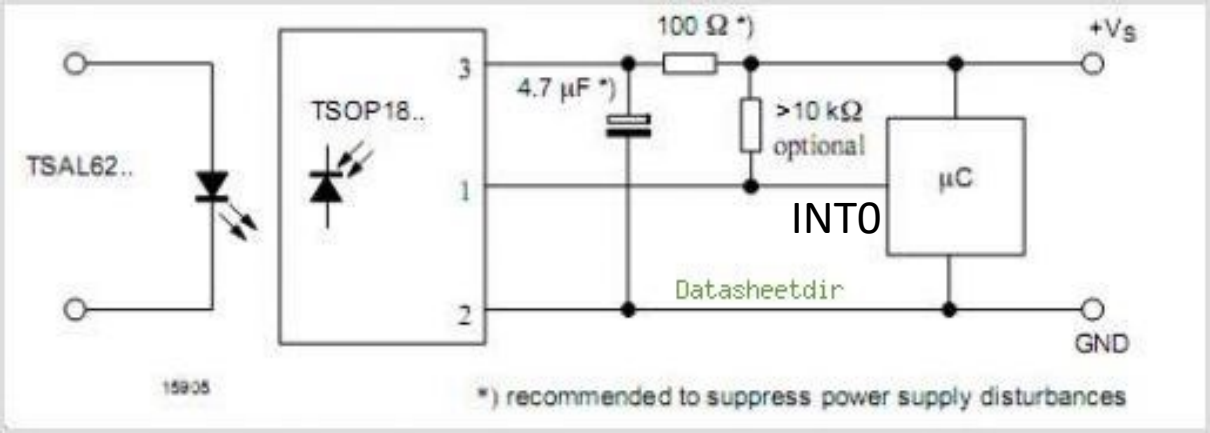
+12V-IN GND +5V-OUT

ENA In1, IN2, In3, IN4, ENB

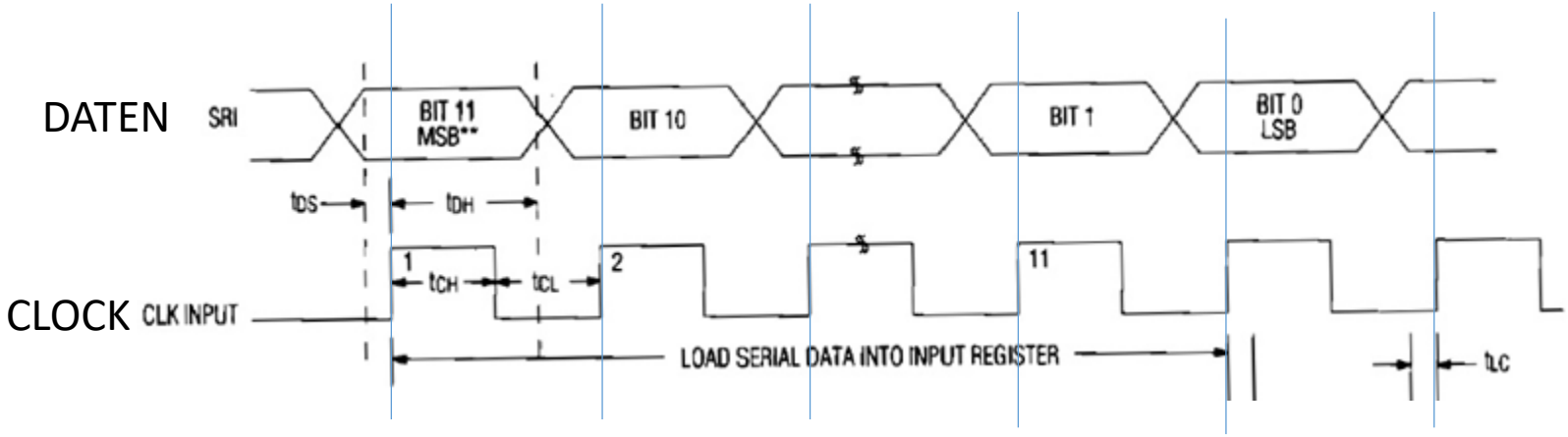


TSOP1838T liegt dem Kurs-Arduino Kit bei





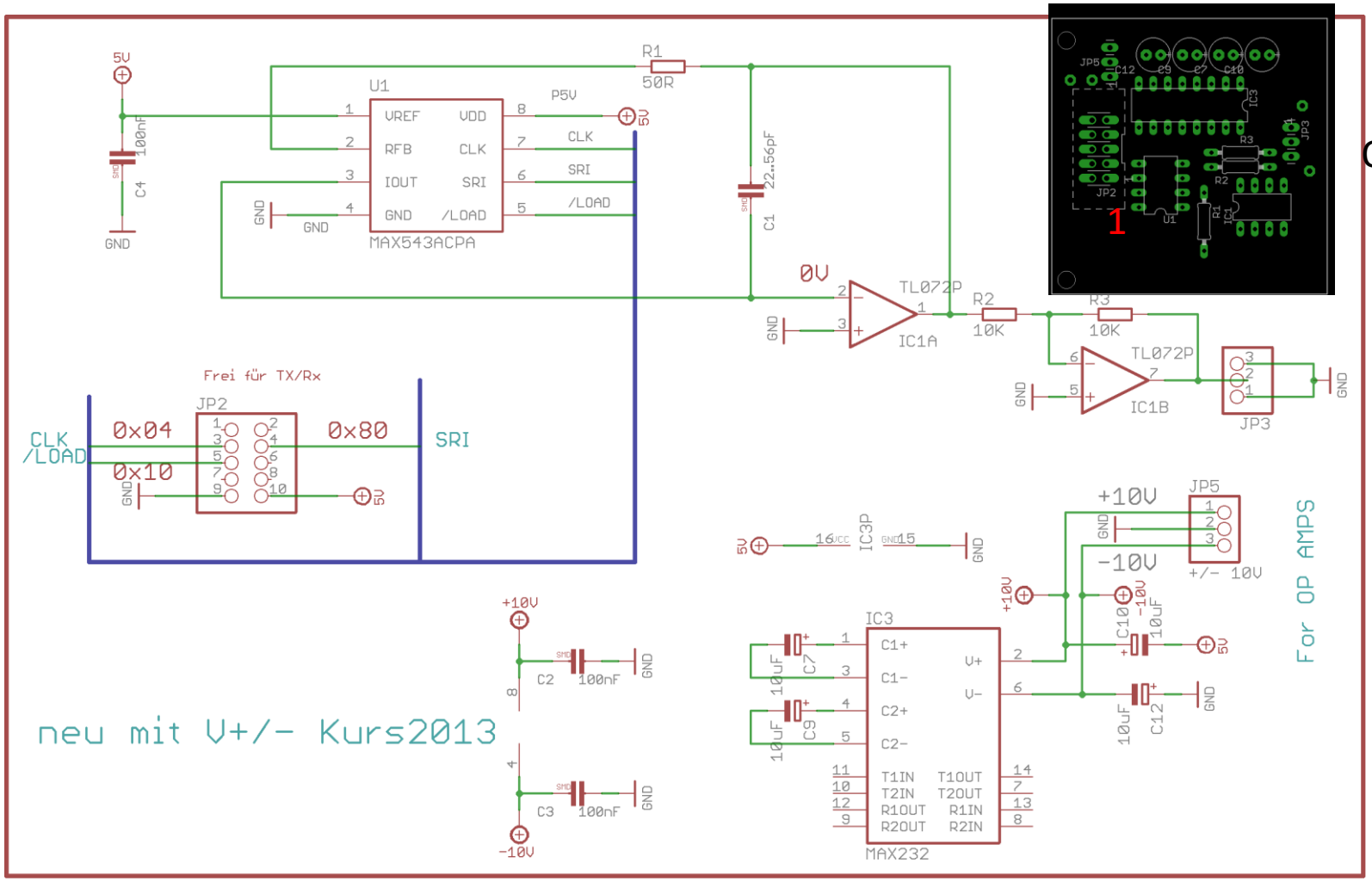
Bitweise Abtastung einer Variable (Prinzip)



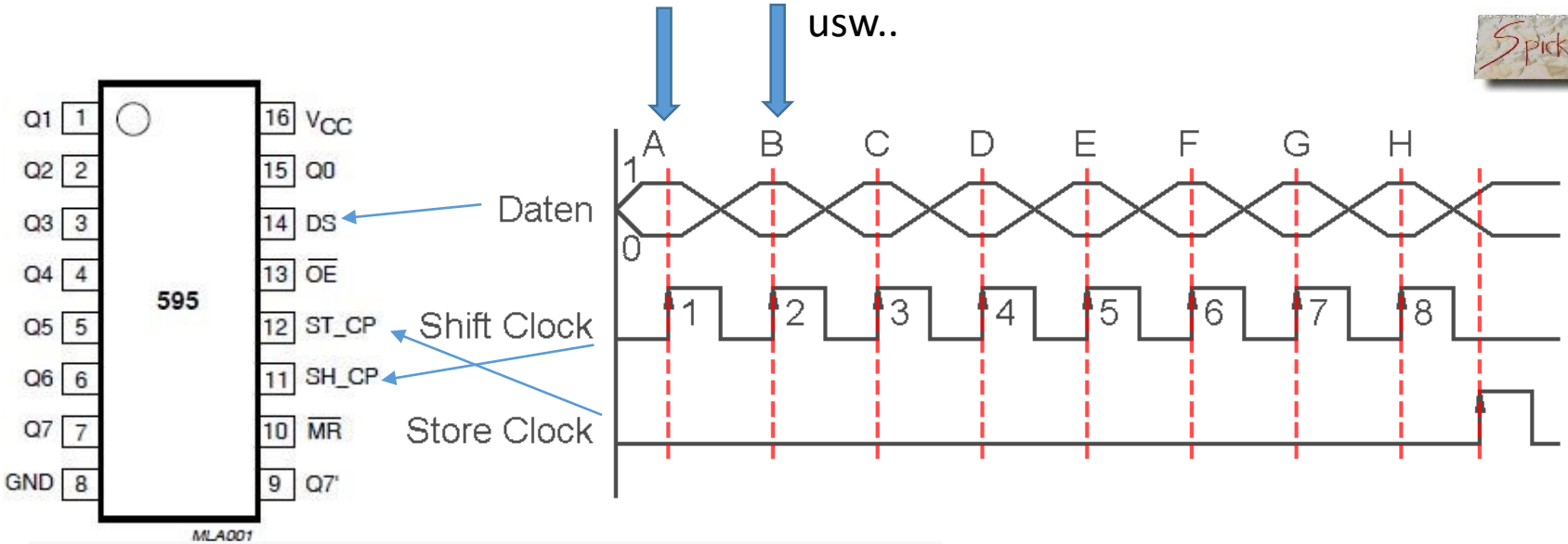
Analysiere eine Variable bitweise mit dem **UND**-Vergleicher einer geschobenen **„1“**

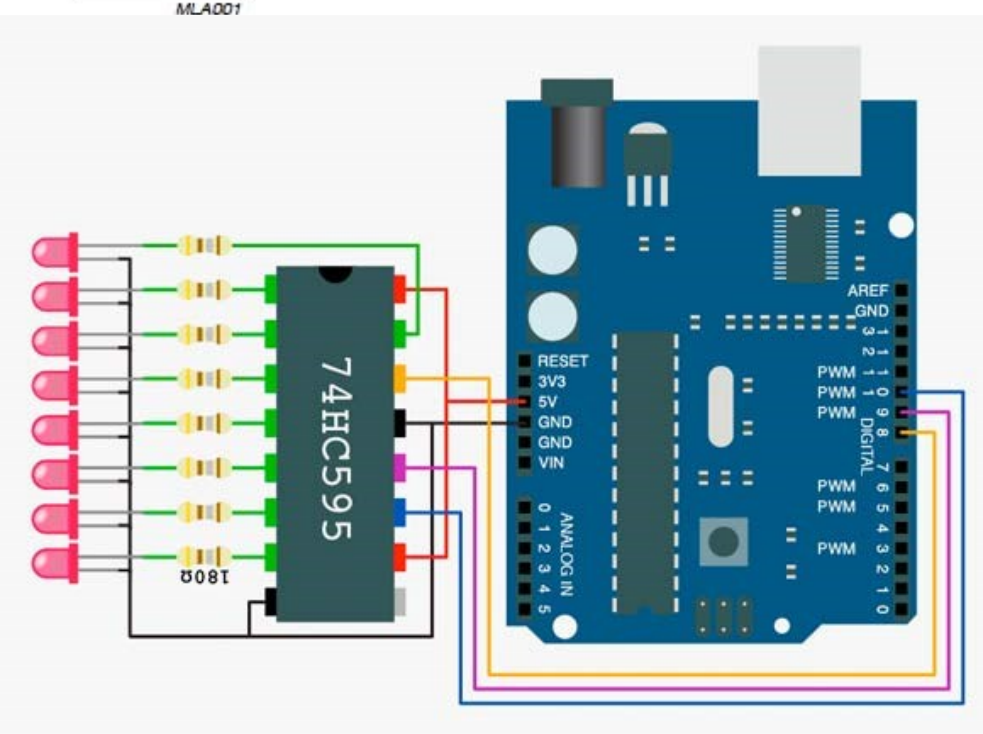
```
u8NN = ShiftBitsCount - 1:      //→z.B 8Bit -1
do  // Clock out bits
   {if(  u8OutVal  &  (1U << u8NN)  )     // Bitweiser & vergleich?
                        {  SET_PIN_TO_1();  }            //Wenn  „1“ Dann setze Data auch 1
            else
                        {  SET_PIN_TO_1();  };           //Wenn  „1“ Dann setze Data auch 0
           _delay_us(1);                    // Warte bist Bit stabil steht
                                            //Clock Job
           CLOCK_TO 1();   _delay_us(1);            //Shift Clock = Übertragung des Bits
           CLOCK_TO 0 ();  _delay_us(1);
}while( u8NN-- );
```

MAX543 DAC 12 BIT



Out

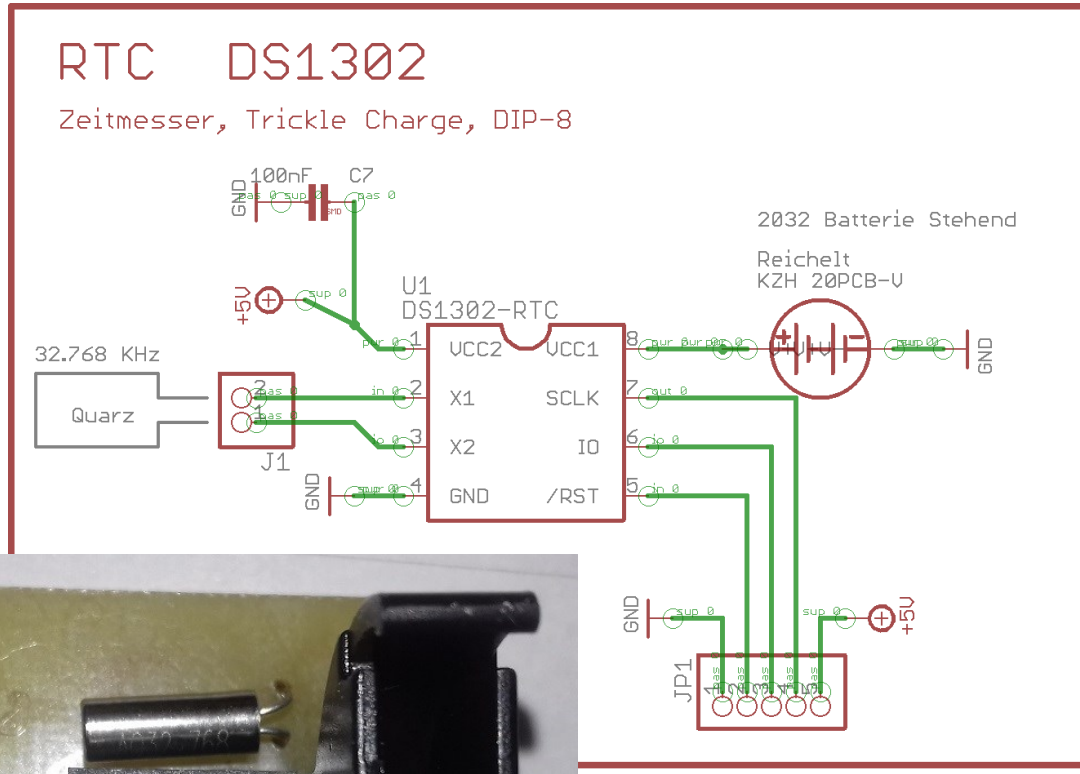


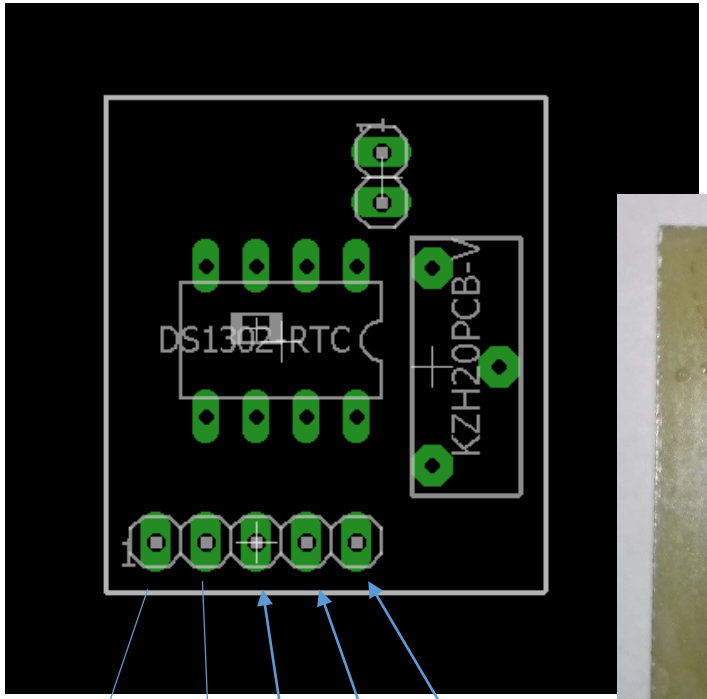



```
#define 74595_DDR      DDRB
#define 74595_PORT     PORTB

#define DS_74595       (1<<PB0)
#define ST_CP_74595    (1<<PB1)
#define SH_CP_74595    (1<<PB2)
```

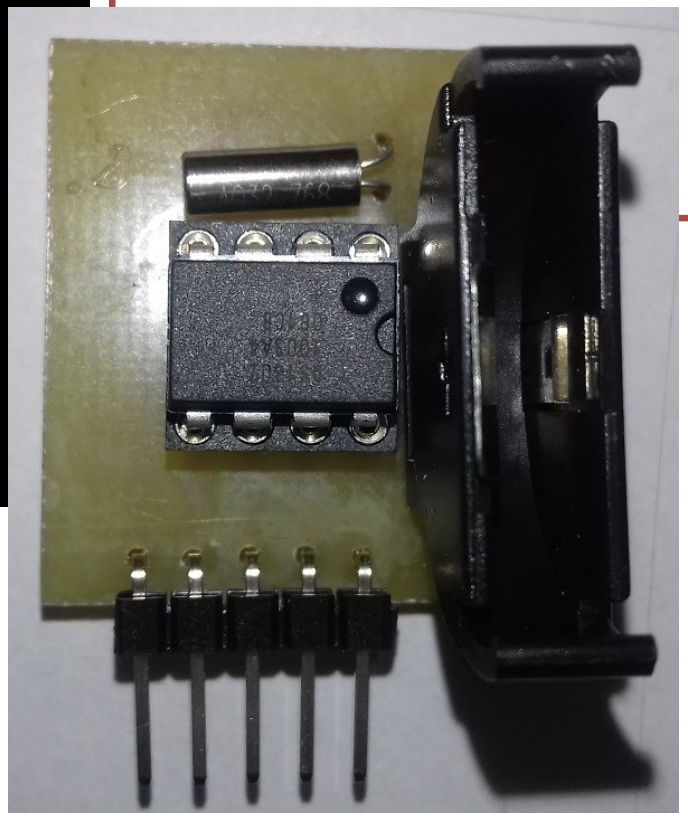
/OE = 0
/MR = 1
Q7 → DS nächster Baustein
wenn da

RTC Real Time Clock DS1302





GND, /RST, IO, SCLK, V+



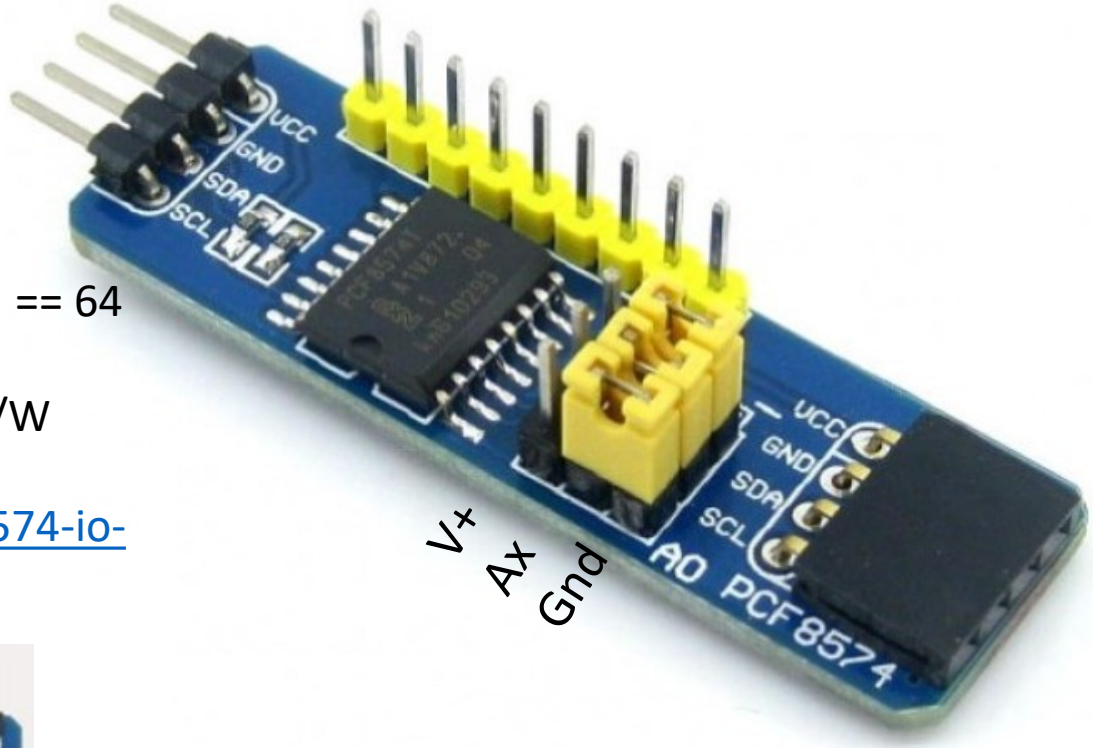
3V Batterie,
Minus = aussen.

ADRESSE codiert ->

| 0 | 1 | 0 | 0 | A2 | A1 | A0 | R/W | == 64

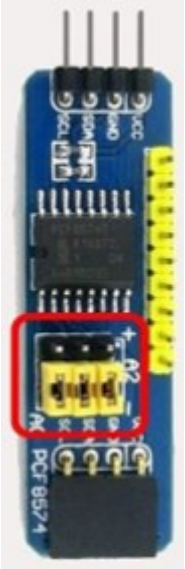
DEZ = 0x40

7-Bit codiert als linksbündig 8 Bit + r/W



https://www.waveshare.com/pcf8574-io-expansion-board.htm

A0	A1	A2
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1



=0x40

0 100 000 0 = 0x40

+ (3Bit)=7 Adressen

0x40..0x47

Alle Jumper Richtung

Stecker= 0x47

ACHTUNG :

out = open colletor

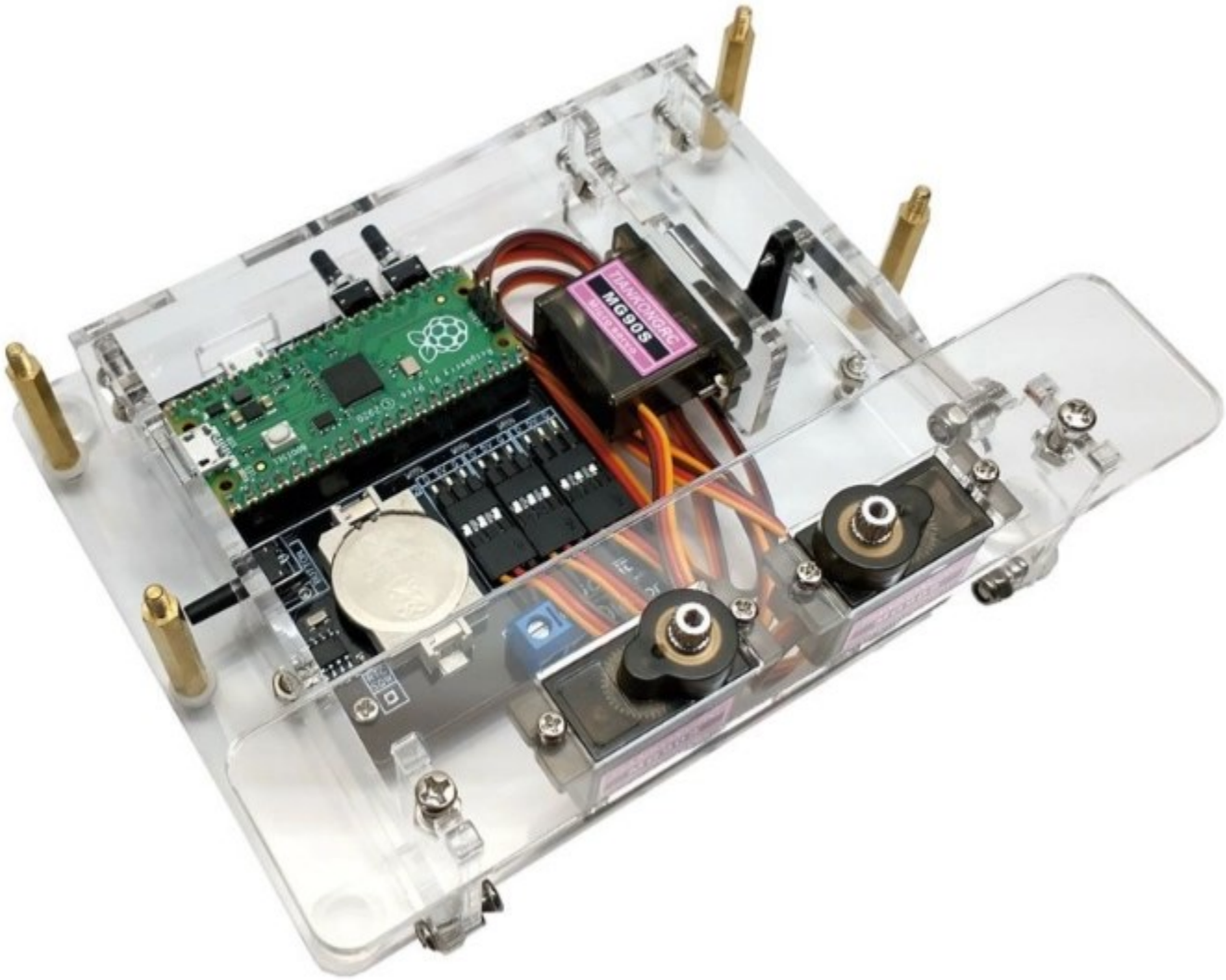
Wegen Kollision mit LCD auf 0x47 ändern

Zur Info und Weiterbildung:

Sehr gute Mathe Übungen. Bestens erklärt. Nachvollziehbar !!!
Macht Spaß
https://www.youtube.com/@MathemaTrick

Mathe auf Profi Ebene.
Sehr gute Vorlesungen: Ebenfalls gut verständlich
Ganz toll !!!
https://www.youtube.com/@WeitzHAWHamburg

Quelle :
https://www.elektormagazine.de/news/raspberry-pi-pico-basierter-sanduhr-bausatz


TOWER PRO
MG90S

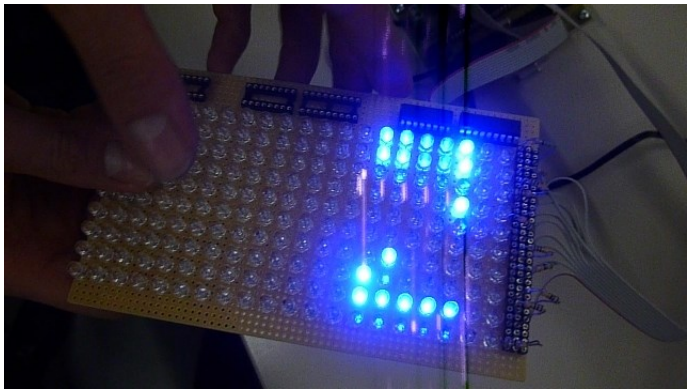
Frequenzen der Töne in der Musik

	Ton	Bezugspunkt	Frequenz HZ	vielfache
1	C	261,63	261,63	**1**
2	Cis	261,63	277	1,06
3	D	261,63	293,66	1,12
4	Dis	261,63	311	1,19
5	E	261,63	329,63	1,26
6	F	261,63	349,23	1,33
7	Fis	261,63	370	1,41
8	G	261,63	392	**1,5**
9	Gis	261,63	415	1,59
10	A	261,63	440	1,68
11	Ais	261,63	466	1,78
12	H	261,63	493,88	1,89
1	C	261,63	523,25	**2**
2	Cis	261,63	554	2,12
3	D	261,63	587,33	2,24
4	Dis	261,63	622	2,38
5	E	261,63	659,26	2,52
6	F	261,63	698,46	2,67
7	Fis	261,63	740	2,83
8	G	261,63	783,99	**3**
9	Gis	261,63	831	3,18
10	A	261,63	880	3,36
11	Ais	261,63	932	3,56
12	H	261,63	987,77	3,78
1	C	261,63	1046,5	**4**

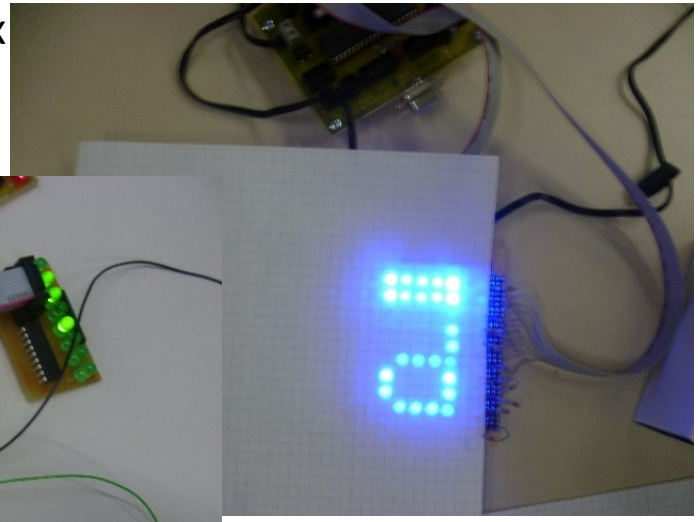
Aufgabe :
Synthetisiere diese Frequenzen mit den 16 Bit Timer mit automatischer Vorteiler Anpassung für einen möglichst hohen Compare Register Wert OC1A
Lasse diese Töne zufällig oder in Terzen zufällig ausgeben am zugehörigen OC1A Ausgang

Bilder aus Projekten

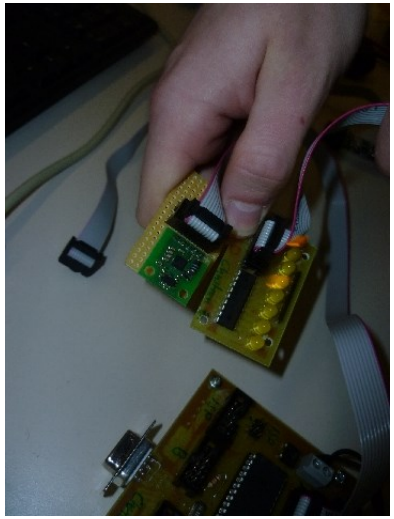
Vorführung



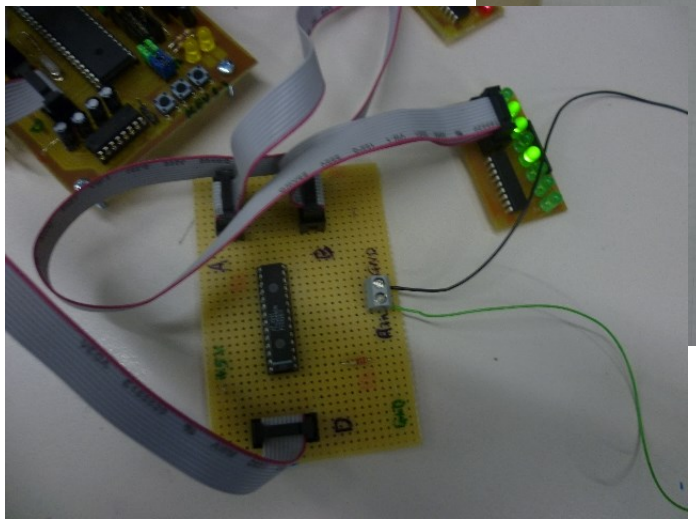
X-Y Matrix



X-Y Matrix

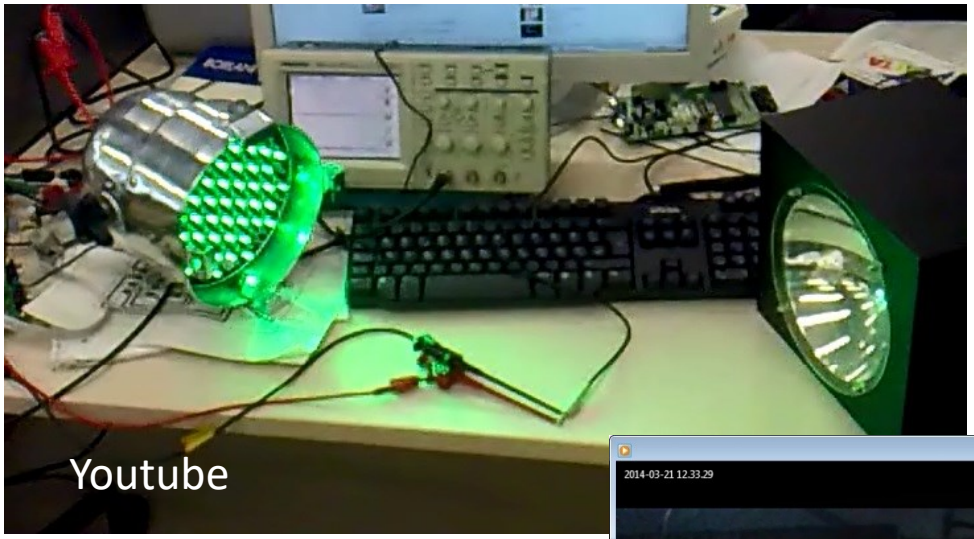


Beschleunigungssensor



24Bit-ADC

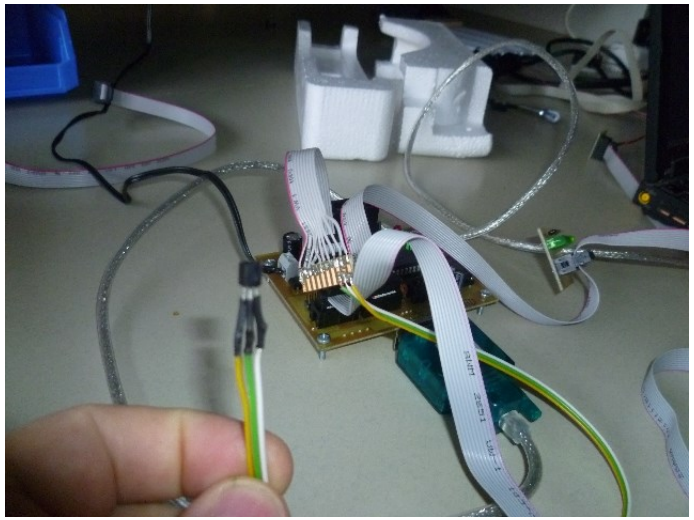
Youtube: http://youtu.be/v4kdm0panGs



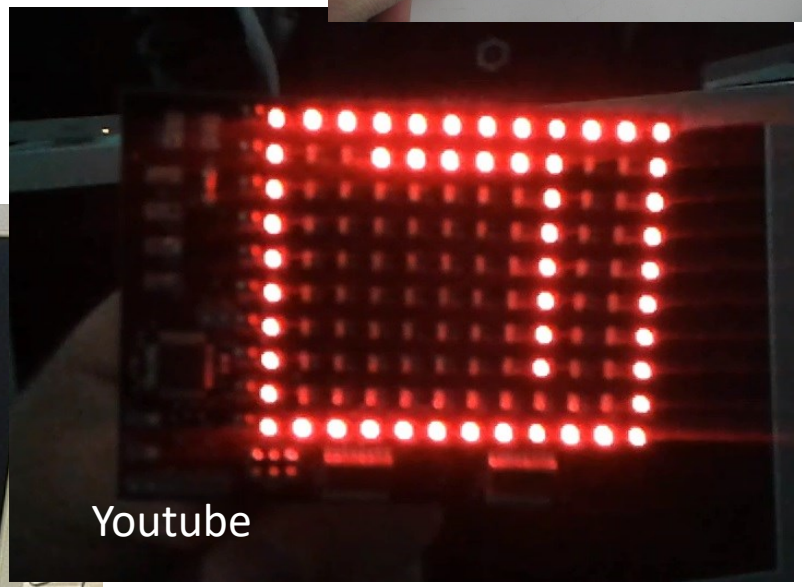
Kurs Projekte

LED Cube und Snake mit PingPong Board

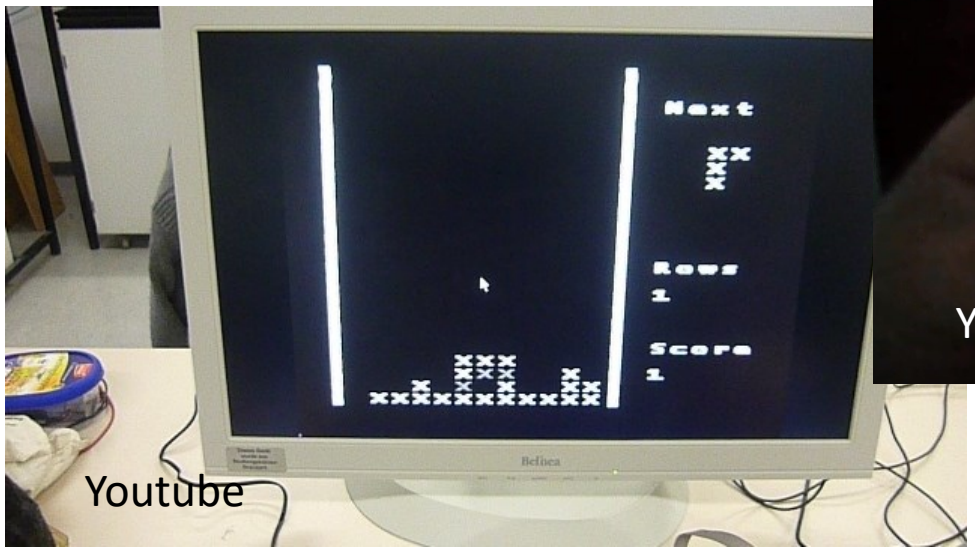
Ein-Draht Temperatur Sensor IC



PID-Regelung mit MOS-Relais



Snake mit Pingpong Board



Video-BAS Tetris

So fing es an.
„Alles baut **ELEKTROMANN**, was man mit Strom betreiben kann“

Mein **Kosmos Radiomann** 1967/68: Ich war ~8 „vom Gebirge bis zum Ozean, alles hört der **RADIOMANN**“ Mit einer Röhre und einem Germanium Transistor. Pfeil Schwarzer Glaskolben: OC83 (…..an RADIOFRAUEN hat man damals nicht gedacht.)


UNITED FEDERATION OF PLANETS
SAMSUNG
CELESTRON
iPhone 4
DELL
SAMSUNG
SAMSUNG

FORTSETZUNG
FOLGT...


Bitte Türen
schließen

Für Kursteilnehmer:
Praktikumsraum 1.0.01 . Tel: 0941-943-2161

@Mobile Privat: 0179-2431170
@work Uni : Tel:0941-943-2140
E-Mail: priv. E-Mail: ChristofErmer@gmail.com und Christof.Ermer@ur.de

Anfragen, Wünsche oder Visiten in meinem Büro sind ausdrücklich erwünscht.
Meine *Merkzettel* Homepage: https://homepages.uni-regensburg.de/~erc24492/
(Dies ist keine Designerseite, sondern wie gesagt, ein Merkzettel)