Hwe45^{xh+ku} v0.251 XbR

BAUSTELLE

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstrasse
Abteilung
Elektronik & Technische Informatik

Ausgeführt im Sommer 2028 von:

Max Power 6aHEL
Suzie Bode 6bHEL
Cate Mos 6cHEL

Projektpartner:
Fa. Icon O.L.S.,
Umbriggler Alm

Betreuer/Betreuerin:

Pull G. Lear Bsc.
Dipl.-Ing.^{se} Kristel Schönfrau
Dr. Gyn Tar

pgfcircversion 0.8.3

Innsbruck, am 01.Apr.0815

Abgabevermerk:

Datum:
99.Dez.3001

Betreuer/in:
Schnapsi Murgs

Bereiche:
1- **Schaltungsentwicklung**
2- **Schaltungsanalyse und -simulation**
3- **Fertigen von elektronischen Schaltungen?**
4- **Projektmanagement und Qualitätssicherung**

'Handlungsdimensionen'/eMail30Aug18:
(AB) Wiedergeben und *Verstehen* (S.1) ↓
(AB) Wiedergeben und *Beschreiben* (S.5)
(C) Anwenden
(D) Analysieren
(E) Entwickeln

'Themenbereiche':
nicht definiert

AV-Vorgaben/270716:
"Im 5. Jahrgang werden gezielte Kleinprojekte in EINZELARBEIT durchgeführt, damit werden die Schülerinnen und Schüler auf die Struktur der HWE-Matura vorbereitet. Die nicht standardisierte Aufgabenstellung für die Reife- und Diplomprüfung muss der Kompetenzorientierung entsprechen (Handlungsdimensionen und Notenschlüssel)." "[...] auch möchte er (AV Stecher) auf keinen Fall die 5. Klasse gegenüber den Besprechungen von 2017 abändern."

'Anforderungsbereiche'/260416:
I Reproduktion
II Reorganisation & Transfer
III Reflexion & Problemlösung

'Erwartungshorizonte'/040218:
?

Inhaltsverzeichnis

[1] AF (audio frequency) ≡ NF (Niederfrequenz)

für oe7ESE:
wieFunktioniertEineAntenne → `https://youtu.be/Uj8oN3WETKo`

i Abgabemodus

- per eMail (c.schoenherr at tsn.at)
- PDF oder .TXT
- FileName mit Autor + Datum + Thema,
  nur ASCII-Buchstaben/Ziffern, keine Blanks, keine Sonderzeichen, keine Üümläüter
- die Bewertung verspäteter Einreichungen verzögert sich entsprechend und verschlechtert sich um 1 Notengrad pro Kalendertag
- Noten/Notenlisten versenden/vermailen ist gem. DSGVO u. Verordnung verboten
- .DOCX + .XLSX gelten als nichtexistent

ii Altium-Modell erstellen/ändern

04.Nov'21 by n17bSanOl

"Also zerst habi

- unter File > Create New > Integrated library a Lib package erstellt und da dann
- mit rechtsklick a schematic library hinzugfügt. Dann habi
- Add component gmacht und
- mei bauteil gezeichnet und
- rechts im properties register
- beim unterpunkt "Models"
- genau deine SPICE parameter aus da pdf eingfügt. Nachan musch
- mit rechtsklich auf die integrated library
- wieder ganz oben auf compile library gehn und
- scho kannsch de in deinem hauptprojekt
- unter libraries hinzufügen.
- Dann wie alle bauteile ins schematic ziehn und
- die schaltungen für die kennlinien aufbaun und simulieren im dc sweep mode."

10.Apr'22 by n17aEllSi

Simulationsparameter eines Bauteils in Altium bearbeiten

1. Altium starten und Twix von Christoph verzehren, bis es geladen wurde
2. Gewünschtes Bauteil aus HTLinn Library (z.B. BC547A) in Projekt ziehen
3. In den "Properties" unter "Models" auf das "Simulation" Modell doppelt klicken
4. Im Pop-Up im unteren Reiter auf "Model File" klicken und gesamten Text kopieren
5. Neue Datei mit UTF-8 Codierung erstellen
   (z.B. einfach im Windows Explorer oder mithilfe des Editors)
   → Achtung: Dateiendung muss .mdl sein
   (Tipp: Dateiendungen im Explorer anzeigen lassen)
6. Erstellte Datei im Editor öffnen und kopierten Text einfügen
7. Gewünschte Parameter bearbeiten (z.B. 'BF' für den Stromverstärkungsfaktor im AP)
8. Zurück zu Altium navigieren und im immer noch geöffneten Pop-Up unter "Model Location" → "Full Path" die gerade erstellte Datei auswählen
   → Achtung: Angegebener Name in der erstellen Datei neben "MODEL" muss mit dem "Model Name" im Pop-Up übereinstimmen
9. Auf "Ok" klicken
   → fertig

iii Lehrstoffverteilung Hwe5 u. Leitfragen

Leitfragen-Koordinatoren: keine

Kompetenzbereiche:
1-**Schaltungsentwicklung**
2-**Schaltungsanalyse und -simulation**
3-**Projektmanagement und Qualitätssicherung**

Themenbereiche:
sind für HWE nicht definiert, da es kein mündliches Maturafach darstellt.

'Handlungsdimensionen'/270716:
(AB) Wiedergeben und *Verstehen* (S.5) ↓
(AB) Wiedergeben und *Beschreiben* (S.7)
'[...]Handlungsdimension AB:
"R e p r o d u k t i o n"
umfassen das Wiedergeben und Beschreiben[...]'
06.Sep17-S.5: *'[...]zusätzlich ergänzt von Helmut Stecher*
Handlungsdimension AB:
"R e p r o d u k t i o n"
umfassen das Wiedergeben und Beschreiben[...]'
(C) Anwenden
(D) Analysieren
(E) Entwickeln

'Anforderungsbereiche'/260416:
I Reproduktion
II Reorganisation & Transfer
III Reflexion & Problemlösung

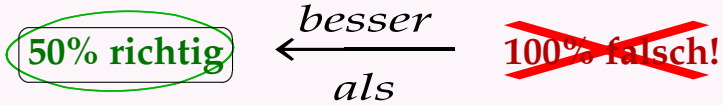
Teil I
Einstieg/WHG

iv YH-Stoffliste HWE

- ☐ Digitaltechnik: Truthtable, KV Diagramm, Boole'sche Glg., Schaltwerk(Automaten)-Entwurf
- ☐ Logarithmus (dB) und Pegel (dBm, dbW, dBuV)
- ☐ Komplexes Rechnen
- ☐ Grundschaltungen von Transistoren und MosFET,
- ☐ Oszillatoren, (An-)Schwingbedingung
- ☐ OP-Schaltungen
- ☐ Instrumentationsverstärker, single supply OPV/INA
- ☐ Filter allgemein, aktive und RLC Filter, Filtercharakteristiken, Übertragungsfkt. H(s) von Filtern u.Amp. (transfer function T(s))
- ☐ Datenblattlesen, Umgang mit Schulbuch 'Böhmer'
- ☐ Kenntnisse der Tools aus 4. und 5. JHG
- ☐ Timerbaustein 555, Funktionsprinzip, Beschaltungen, Dimensionierung, Signalformen
- ☐ Zählschaltungen, Decoderschaltungen u. -Anwendungen
- ☐ Blockschaltbild, Schaltplan, Print-Layout, Funktionserklärung, Diagramme, Bauteildimensionierung
- ☐ 30..40 Teilaufgaben in 300 Minuten
- ☐ Auswahl und Ausführungsreihenfolge der Aufgaben-Erledigung

✓ leichte Aufgaben zerscht

✓ nur mit 85% Tempo arbeiten:



✓ **Punktegewichtung** beachten — manche Zusatzaufgaben *'bringens nicht*

Teil II

Hwe4

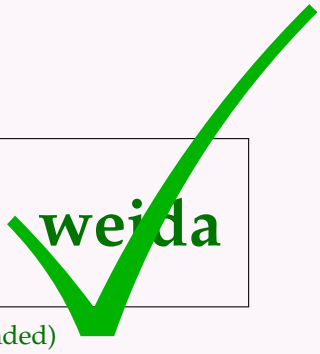
here begin die

HWE4 - Inhalte:

abort

blattl wei da

(recommended)



v Fertigkeiten

v .1 Widerstandsfarbcode, resistor color code

bk bn rd or ye gn bu vi gy wt

v .2 E12 Reihe, E12 series

10		bn bk
12		bn rd
15		bn gn
18		bn gy
22		rd rd
27		rd vi
33		or or
39		or wt
47		ye vi
56		gn bu
68		bu gy
82		gy rd

Die häufigen sind:

10		bn bk
15		bn gn
22		rd rd
33		or or
47		ye vi

v .3 Widerstände einsortieren

Übung: Das eigene Bauteilset bzw. die Schulbauteilboxen ausleeren und neu einsortieren

v .4 Ebers-Moll Model

$$\boxed{I_C = I_S * e^{\frac{U_{BE}}{U_T}}}$$

mit

$$U_T = \frac{kT}{q}$$

(= **25.7mV** @ 298.15K ≙ 25°C,
30mV @75°C,
34mV @120°C)

k . . . 1.38E-23 Boltzmannkonstante

T . . . abs. Temp. in $[K]$

q . . . 1.6E-19 Elektronenladung, electron charge

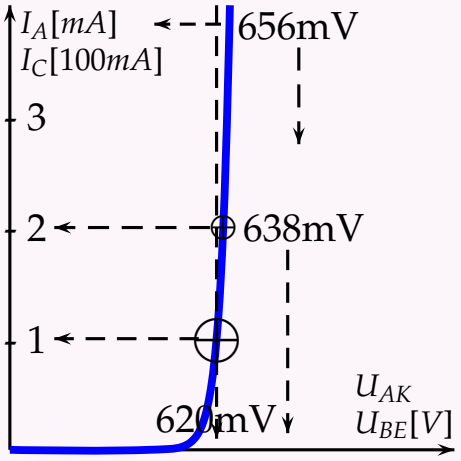
I_S . . . Sperrsättigungsstrom, sehr temperaturabhängig!

v .5 Konstruktion der BJT-Steuer- u. Diodenkennlinien

1. x-Achse: U_{BE} oder U_{AK}
2. y-Achse: I_C oder I_A
3. Messpunkt zB 650mV → 100mA
   (Diode 620mV → 1mA)
4. weitere Kurvenpunkte: ΔI_C:
   I_C mal 2 je +18mV U_{BE}
   I_C mal 0.5 je -18mV U_{BE}
   I_C mal 10 je +62mV U_{BE}
   I_C mal 0.1 je -62mV U_{BE}
5. Steigung (der Tangente)

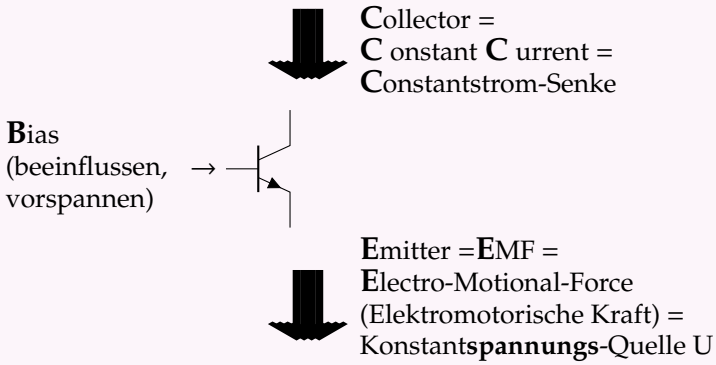
   $\boxed{s = I_C/U_T}$

   zB. I_C=1mA DC
   $\Longrightarrow$ s = 1mA/26mV
   ≈ 0.04
   ≈ 40mA/V
   $(\Delta I_C[A]/\Delta U_{BE}[V])$

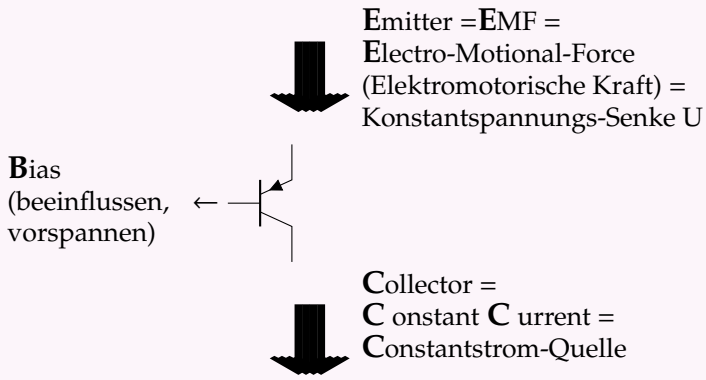


v .6 'C' wie 'Current'

v .6.1 NPN-Transistor



v .6.2 PNP-Transistor



Teil III
WHG Grundlagen Hwe-1

vi .1 Strom und Spannung

U

Spannung U ist
elektrischer Druck
(wie Pressluft)

Ohm'sches Gesetz:
je Spannung desto rinnt es!

"URI-Dreieck"

I

Ladungsmenge Q sind
Liter Strom

$1\ e^- \equiv 1.6021766208 * 10^{-19}[Cb]$

$1\ Cb \equiv 6.241509125 * 10^{18}$ Stk. e^-

Stromstärke I (intensity) ist
Liter Strom
pro Sekunde

richtig ist *'Coulomb'* [$Cb = A{\cdot}s$], nicht *'Liter'* :-))

C

Kondensator aufladen ist wie Reifen **aufpumpen**
(auch Luft will wieder aussa)
Kondensator entladen ist wie Luft **auslassen**

L

Spule einschalten ist wie Schwungrad **anschubsen**
U ≡ F
I ≡ v
Spule abschalten ist wie Schwungrad **bremsen**

vi .2 Widerstand u. Leitfähigkeit

Feststoffe, Flüssigkeiten, Gase und Dämpfe sind verschieden elektrisch leitfähig. Den Messwert dieser Leitfähigkeit nennt man **Leitwert G**. Die Masseinheit is das *Siemens*, Kurzzeichen 'S'.
die Amis haben nach dem Krieg hektisch deutsche Erfinder und Masseinheiten aus ihrer Literatur gestrichen und durch Ami ersetzt, so schreiben sie oft heute noch '[mho]' oder '[℧]' statt '[S]' — 'Ohm' rückwärts :∧) (dass Georg Simon OHM oder Walter Hermann Schottky auch Deutsche waren, hams nid checkt ☺ ☺ ☺

Jetzt hat aber dieser *Leitwert G* oft unhandliche Werte, so Millisiemens [mS] oder Mikrosiemens [μS]. Der Kehrwert $R = 1/G$ hingegen liegt in der Elektrotechnik – aus der die Elektronik ja stammt – meist bei einigen Zehn oder Hundert Ohm [Ω] – das ist angenehmer zu sprechen, denken, schreiben – und so denkt und rechnet man alles mit Widerständen R (statt der begreiflicheren Leitwerte G).
stell Dir vor, im E-Magazin sagt wer "bitte um Leitwerte"; man hätte einen Leitwertefarbcode, eine E12-Leitwertreihe, einen Wellenleitwert, Innenleitwert, C/G statt RC usw.

→ Fertigkeiten U, I, R, RC, $e^{\frac{U_{PN}}{U_T}}$ Kap.'v ' S.13

vi .3 RC-Auf/Entladekurve

Die Kondensator-**Ent**ladung eines RC-Gliedes folgt

$\boxed{U_C(t) = U_{C,max} * e^{-t/\tau}}$ und die **Auf**ladung $\boxed{U_C(t) = U_{C,max} * (1 - e^{-t/\tau})}$

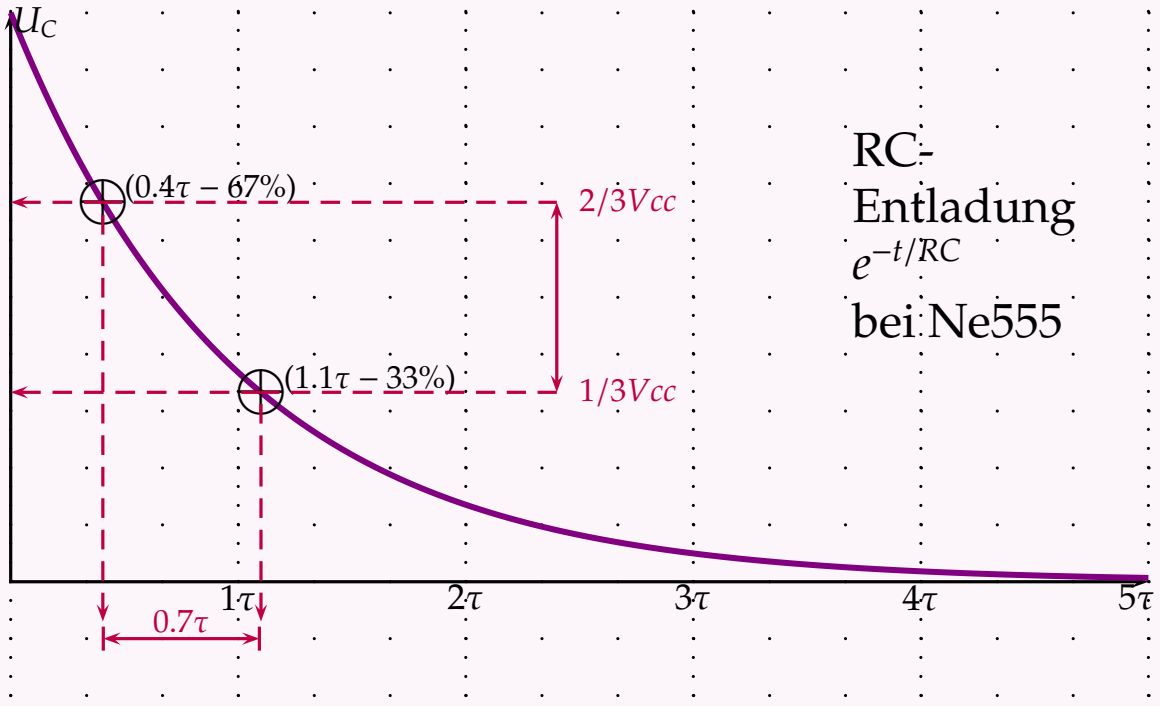
mit $\tau = RC$.

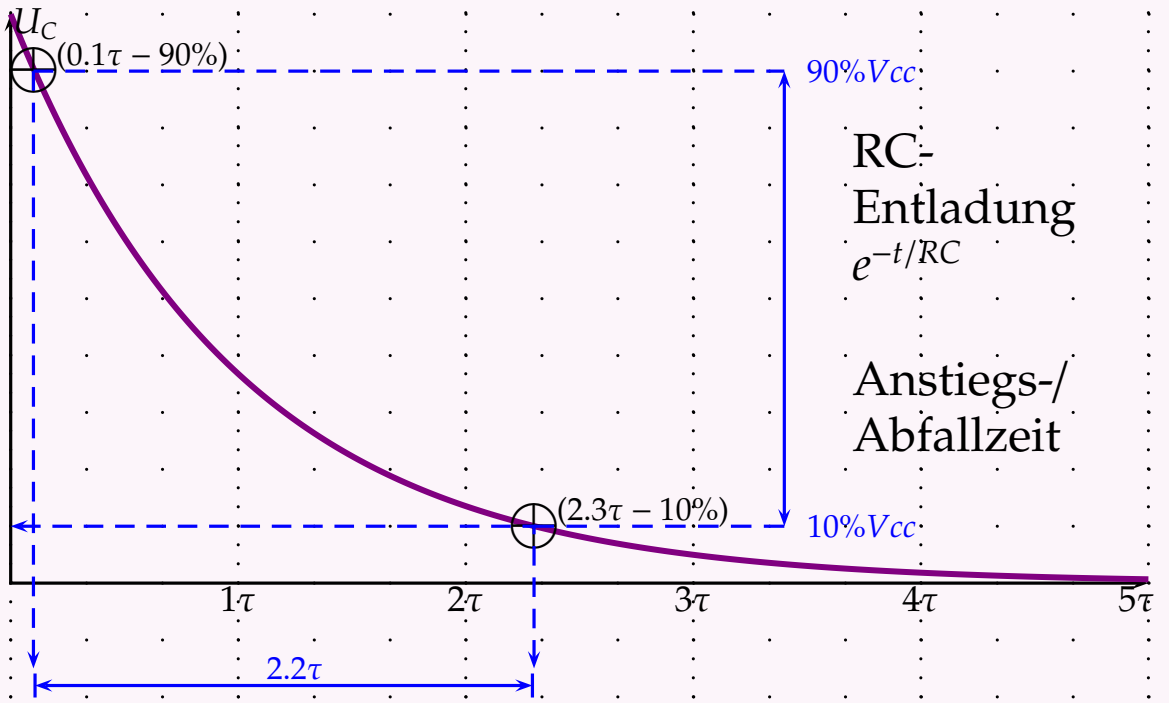
Die oft gesuchte Umkehrfunktion ist

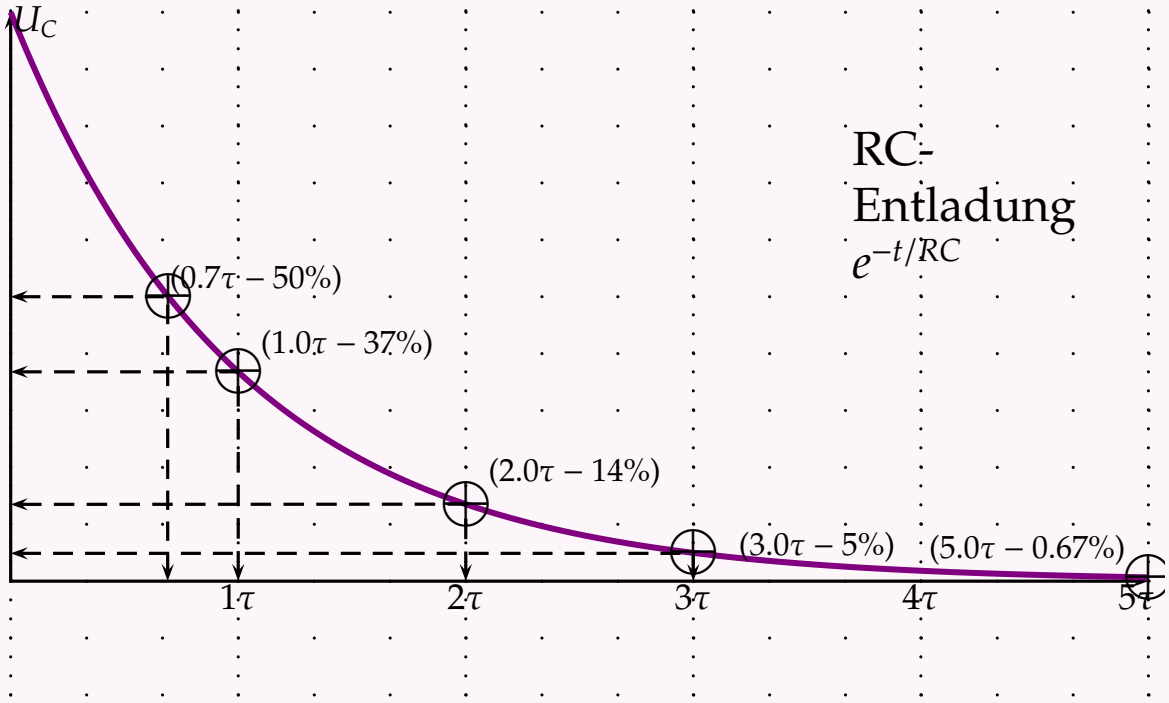
$\boxed{t/\tau = -ln(\,U_C(t)/U_{C,max}\,)}$ bzw. in Sekunden: $\frac{t}{[s]} = -RC * ln(\,\frac{U_C(t)}{U_{C,max}}\,)$

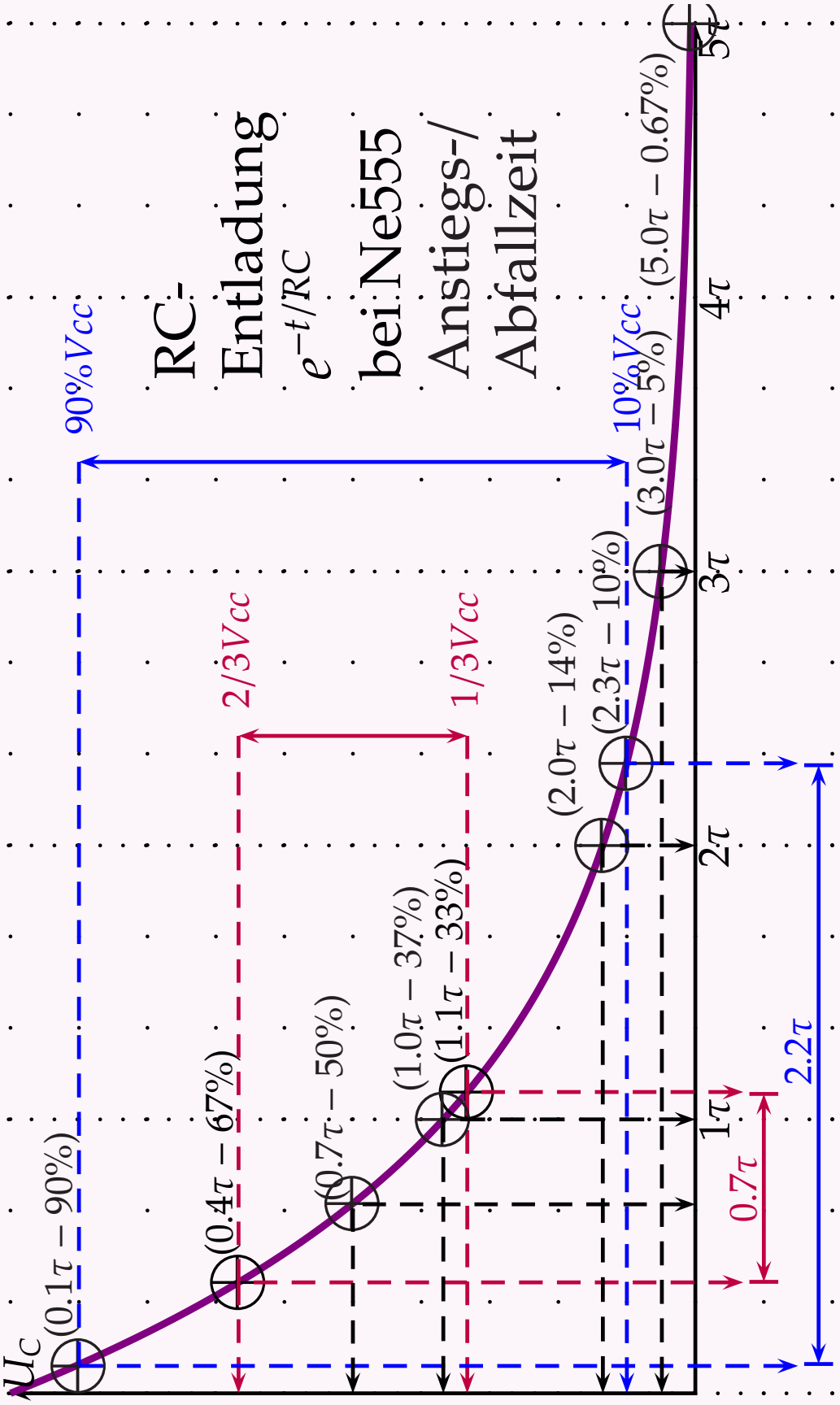
und falls Du keine "ln()" Funktion hast: $\frac{t}{[s]} = -\frac{RC}{log(e)} log(\,\frac{U_C(t)}{U_{C,max}}\,)$

Wir normieren $U_{C,max} = 1$ (zB. $U_{C,max} = 1.0V$ Kondensator-Entlade-Anfangsspannung)
Für andere Werte ist die Kurve in "y-Richtung" entsprechend zu skalieren (="zoomen")
Wegen der einfacheren Formel ($e^{-t/\tau}$) merken wir uns die ENTladung:




U_C
(0.1τ – 90%)
90%Vcc
RC-
Entladung
$e^{-t/RC}$
Anstiegs-/
Abfallzeit
(2.3τ – 10%)
10%Vcc
1τ
2τ
3τ
4τ
5τ
2.2τ


U_C
RC-
Entladung
$e^{-t/RC}$
(0.7τ – 50%)
(1.0τ – 37%)
(2.0τ – 14%)
(3.0τ – 5%)
(5.0τ – 0.67%)
1τ
2τ
3τ
4τ
5τ


RC-
Entladung
$e^{-t/RC}$
bei Ne555
Anstiegs-/
Abfallzeit
u_C
$90\%V_{CC}$
$10\%V_{CC}$
$2/3V_{CC}$
$1/3V_{CC}$
$(0.1\tau - 90\%)$
$(0.4\tau - 67\%)$
$(0.7\tau - 50\%)$
$(1.0\tau - 37\%)$
$(1.1\tau - 33\%)$
$(2.0\tau - 14\%)$
$(2.3\tau - 10\%)$
$(3.0\tau - 5\%)$
$(5.0\tau - 0.67\%)$
1τ
2τ
3τ
4τ
5τ
0.7τ
2.2τ

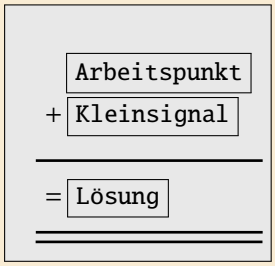
vi .4 Kleinsignal-Ersatzschaltung mit M.Miljak

wird auch
"Wechselstrom-Ersatzschaltung"
genannt.

Zur Erleichterung der Berechenbarkeit mittels **Näherung** wird die Schaltungsberechnung aufgesplittet:

a) Großsignal-Arbeitspunkt
(engl. 'Q-point' oder 'bias point')

b) kleine Veränderungen "Kleinsignal": lineare Näherung



Als lineare Bauelemente sind bekannt:

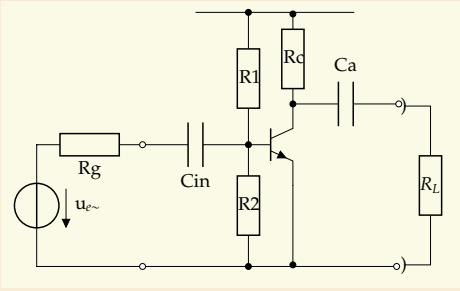
- ideale Strom- und Spannungsquellen
- die grundlegenden passiven Bauelemente R, L und C

Nicht Bestandteil des Ersatzschaltbildes sind Bauelemente, die der Arbeitspunkteinstellung dienen, sowie die nicht linearen Bauelemente, die ersetzt werden sollen.

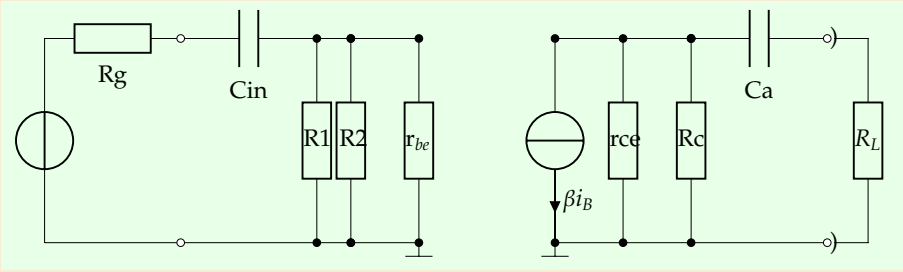
Schrittweise Entwicklung eines Kleinsignal-Ersatzschaltbildes:

- Bestimmung der Arbeitspunkte der zu ersetzenden Bauteile
- Bauteile 'an Vcc' werden als 'an GND' betrachtet (weil 'Vcc' als U-Quelle mit $R_i = 0$ zählt)
- Alle nicht gesteuerten oder variierten Spannungsquellen durch Kurzschlüsse ersetzen, die nicht gesteuerten oder variierten Stromquellen durch Leerläufe
- Variierte Quellen, werden in Kleinsignaldarstellung nur durch den Variationsteil dargestellt.
- Nichtlineare Bauelemente (z.B. Transistoren) werden durch ihre entsprechenden Kleinsignal-Ersatzschaltbilder ersetzt.

Kleinsignalrechnungen werden vor allem verwendet, um Verstärkungen, Ein- und Ausgangsimpedanzen usw. zu berechnen.



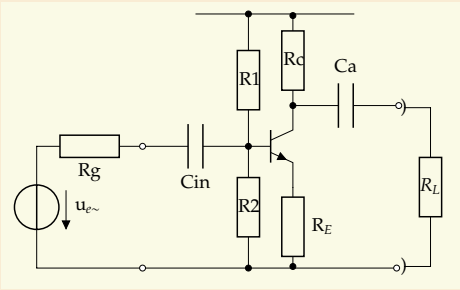
Schalt
plan
ohne RE
(geht nit!)



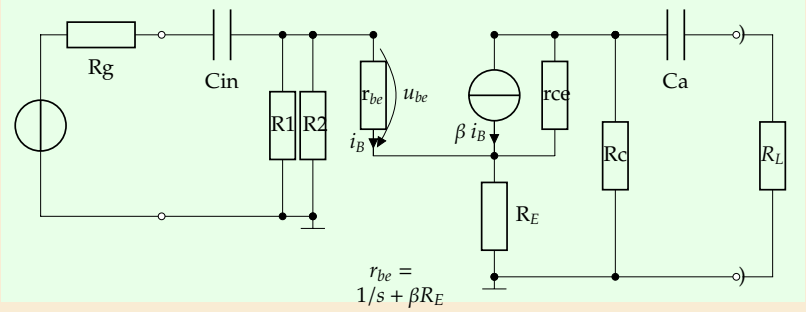
Klein
signal
ersatz
schaltung

$\underbrace{Rg + X_{Cin}}$ $\underbrace{R1 \,\|R2\, \| \, r_{be}}$ $\underbrace{r_{CE} \parallel Rc \parallel (X_{Ca} + R_L)}$

$\underbrace{(Rg + X_{Cin}) \quad \| \quad R1 \parallel R2 \parallel r_{be}}$



Schaltplan
(incl. RE)



Klein
signal
ersatz
schaltung

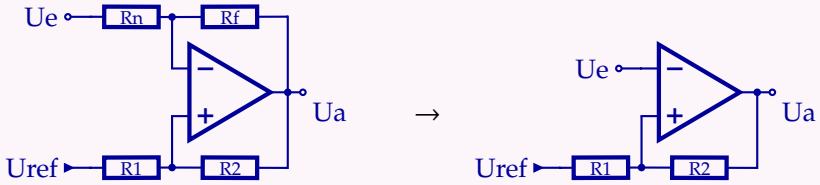
wie
oben

$\underbrace{r_{be} = 1/s + \beta R_E \approx \beta R_E}$

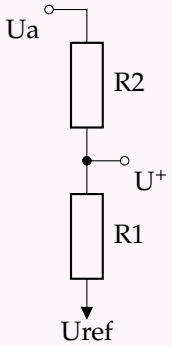
$\underbrace{u_{be} = u_b - \beta i_B * R_E}$

vii hyst.Komparator Berechnung

s. Böhmer:158, 220



invertierend (inverting)



$$U^+ = U_{ref} + (Ua - U_{ref})\frac{R1}{R1+R2}$$
$$= \frac{U_{ref}(R1+R2) + (Ua - U_{ref})R1}{R1+R2}$$
$$= \frac{U_{ref}R2 + UaR1}{R1+R2}$$
$$= \text{(mit Leitwerten gewichtetes Mittel:)}$$
$$= \frac{U_{ref}/R1 + Ua/R2}{1/R1 + 1/R2}$$

Stell Dir vor, wir mischen U_{ref}=Essig und U_a=Öl, indem R2 der Öl-Schlauch und R1 2 Stk parallele Schläuche darstellten. Dann ergäbe sich doch das Mischungs-Verhältnis 2:1 - Je besser der Ölschlauch 'leitet', desto mehr Öl ist drin - ergo U1 *MAL Leitwert:* $\rightarrow U_{Knoten} = \frac{U_{ref}/R1 + Ua/R2}{1/R1 + 1/R2}$

Die Umschaltschwellen sind natürlich bei $U^- = U^+$

mit der Fallunterscheidung $U_{a,max}, U_{a,min}$ wird daraus

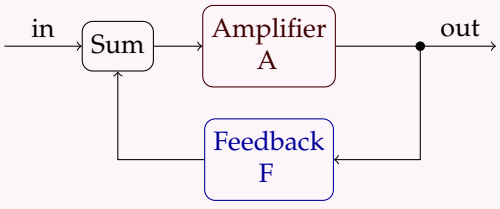
$$U_{e,th,HI} = \frac{U_{ref}/R1 + U_{a,max}/R2}{1/R1 + 1/R2}$$
$$U_{e,th,LO} = \frac{U_{ref}/R1 + U_{a,min}/R2}{1/R1 + 1/R2}$$

nichtinvertierend (non inverting)

für den nichtinvertierenden Komparator mit Hysterese
vertauscht man einfach U_{ref} und U_e.

Signalgraph

(Signalgraphen aus der Systemtheorie/Regelungstechnik sind ganz was anderes als Signaldiagramme!)



out = A* (in + out*F)
out - out*A*F = A*in
out = in $\frac{A}{1-AF}$
(vgl. charakteristische Glg.)
→ wenn AF > 1 → invertierend stabil
→ wenn AF < 1 → nichtinvertierend instabil (bistabil)

Teil IV

Analog Teil

1 Audioverstärker

1 .1 EGS die Emitter- Grundschaltung

Vorgaben:

- v_u
- f_g
- V_{CC}
- R_{in}, R_{out}

Aufgabe: Dimensionierung von

- $R_C, R_{E1}, R_{E2}, R_{B1}, R_{B2}$
- C_{in}, C_{out}, C_E

  $f_{g,in} = f_{g,E} = f_{g,out} = f_g * \sqrt{3}$

  die -3dB Grenzfrequenz
  N identischer Tiefpässe 1-ter Ord.
  ist bekanntlich

  $f_{-3dB,N} = \frac{f_{-3dB,1}}{\sqrt{N}}$

Emittergrundschaltung EGS dimensionieren (f.Laborzwecke)

Die *eingentliche Kunst* so einer Dimensionierung besteht im Verständnis tolerierbarer Vernachlässigungen.

> Bitte **nicht** geschlossene Monster-Formeln hernehmen (die alles berücksichtigen), verständnislos stur einsetzen und am Taschenrechner durchtappen, sondern **vor** dem Rechnen vereinfachen. Besser wissen, ob ein geschätztes Ergebnis *eher zu groß* oder *eher zu klein* ist.

Bauteile mit "krummen" Werten sind eh nicht verfügbar, und man muss entscheiden, ob der *nächst größere* oder der *nächst kleinere* Bauteilwert besser passe (natürlich kann man gewünschte Bauteilwerte durch Serien- und Parallelschaltungen nähern, aber *bitte* nur, wenns wirklich sein muss).

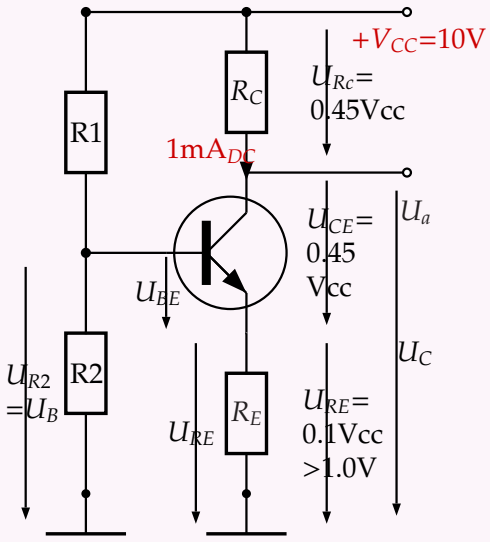
Ohne Emitter-Gegenkopplungswiderstand wird die EGS nur im Schaltbetrieb (dh. entweder $U_C \approx V_{CC}$ oder $U_C = U_{CE,sat}$) oder in Regelschleifen eingebettet sinnvoll verwendbar. Zur Berechnung im Folgenden einfach $R_E = 0$ setzen.

Praktisch wird diese Schaltung nur mehr selten (eher im HF-Bereich), in ICs gar nicht (wegen der Widerstände), verwendet, fördert aber das Grundverständnis und lehrt den Dimensionierungsvorgang.

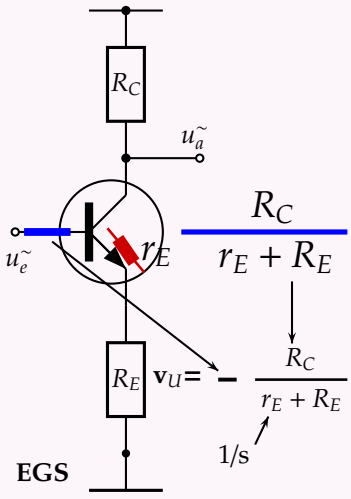
Moderne, IC-tauglichere Schaltungskonzepte verwenden zur Arbeitspunkteinstellung Differenzverstärker, **Stromspiegel**, Darlingtonschaltung, **Konstant**strom- und spannungsquellenkombinationen sowie Rückkopplung (feedback) anstelle von Widerständen, weil sich das besser *von selber* stabilisiert (zB. LM324 soll mit Vcc = 3.0 . . . 32.0 Volt gleich funktionieren!); das erfordert aber schon gute Dimensionierungskenntnisse.

Leistungsverstärker brauchen Zusatzschaltkreise zur Stromverstärkung, Schwingdämpfung, Kurzschlußschutz, Temperaturkompensation udgl. (die Prinzipschaltungen hier funktionieren so noch nicht wirklich!)

DC-Arbeitspunkt:



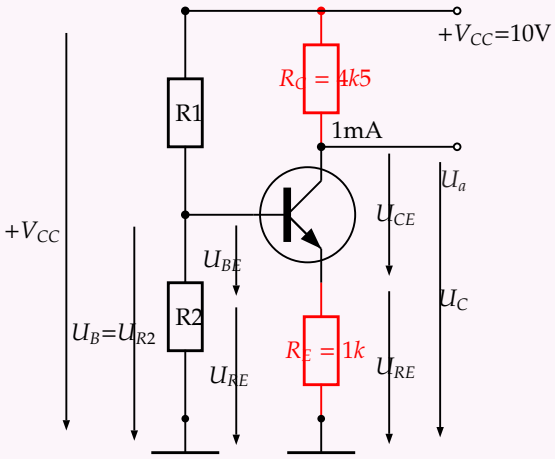
Kleinsignal - Verstärkung:



Dimensionierung, nur für Kleinsignalbetrieb: Zuerst legen wir den **DC-(Gleichstrom)-Arbeitspunkt** fest. Achtung: Datenblätter geben Werte immer unter günstigsten Bedingungen an, das schaut *besser* aus - niemals gelten sie alle gleichzeitig! (β ist 550, aber **niemals** bei Vcc_{min} **und** Ic_{max} **und** f_{max} **und** 175°c **und** nach 85 Jahren und. . .):

1. Die Betriebsspannung V_{CC} wird festgelegt; zB. Vcc=10[V]
2. I_C wird entschieden (nach gewünschtem Rout, Rin, s,. . .). Wenn nicht anders gefordert, nehmen *wir* (einfach) Ic=1mA
3. Der Stromverstärkungsfaktor β wird ermittelt (Datenblatt oder Annahme β=100 oder 250).
4. R_E wählt man so, dass sich U_{RE} zu
   - 10% von Vcc,
   - mindestens jedoch 1.0 Volt,

   einstellt.
   hier:R_E = 0.1 Vcc/Ic = 0.1*10V/0.001A = 1k
5. Die Kollektorgleichspannung U_C soll maximalen *Spannungshub* $\widehat{u_C}$ zwischen Vcc und U_E erlauben, also *genau zwischen Vcc und* U_E liegen.
   $\rightarrow\ U_C = U_E + \frac{(Vcc - U_E)}{2} = \frac{Vcc + U_E}{2}$
   hier: Uc = (10+1)/2 = 5.5[V]
   das ist *kompliziert*; wir teilen einfach:
   $U_{R_C} : U_{CE} : U_{R_E} = 45\% : 45\% : 10\%$
6. Daraus bestimt man
   R_C=(Vcc-Uc)/Ic=$45\%V_{CC}$/Ic,
   hier 4.5/0.001=4k5
7. Nun haben wir $I_C \approx I_E$=1mA, Vcc=10V, Uc=5.5V, U_E=1.0V, s=$\frac{I_C}{U_T} = \frac{1mA}{26mV}$ = ~40mS (manche schreiben 40m℧ oder mMho)
   r_E = 1/s =26Ω und $r_{BE} = \beta/s = \beta * r_E$ = 2k6
8. Die effektiv wirksame Vorwärts-Steilheit s' ist wegen der Gegenkopplung nach
   $s' = \frac{s}{1 + sR_E}$ zu korrigieren
   (Rechnung mit $r_E + R_E$ statt nur r_E)



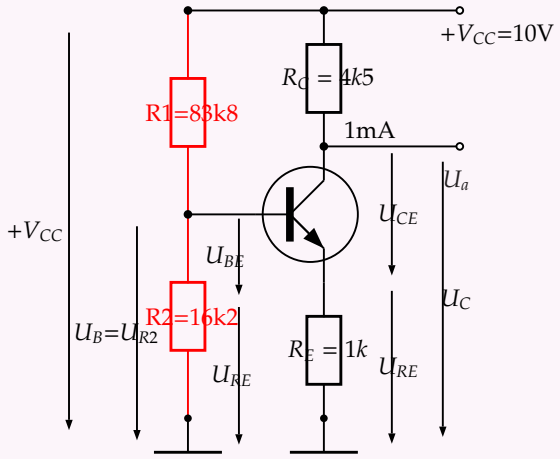
EGS Dimensionierung

weitere Dimensionierung: Der Basisspannungsteiler R1/R2:

1. Es ist zu bedenken, dass der Strom durch R1=I_{R1} auch den Transistor-Basisstrom I_B trägt, also $I_{R1} = I_{R2} + I_B$
2. Zur vereinfachten Errechnung ersetzen wir (auf Kosten unnötig hoher I_{R1} und I_{R2} Ströme) die Basisstrom- I_B Beachtung durch die Daumenregel:
   Nimm I_{R2} = 10 Mal I_B als *Basisspannungsteiler-Querstrom* I_{quer} und ignoriere dann I_B, sodass $I_{R1} = I_{R2} = I_{quer}$
3. Unser $I_B = I_E/\beta$ also 1mA/100 = 10uA
4. $I_{quer} = 10 * I_B$ = 10*10uA = 100uA
5. An der Basis brauchen wir die Gleichspannung U_B um die Basis-Emitter-Flussspannung U_BE höher (als am Emitter). Basisspannung $U_B(1.62V) = U_{RE}(1.0V) + U_{BE}(620mV)$ (die U_{BE}=620mV sind BC546-*typisch* für Ic=1mA)
6. nun kriegen wir R2 = U_B/Iquer = 1.62V/100uA = 16k2 und
7. R1 = (Vcc-U_B)/Iquer = (10V-1.62V)/100uA = 83k8 [2]
   Vorschlag: Wie wärs mit 82k und 18k (die es wirklich gibt)

[2] zum Glück hast Du ja genau diese Werte in Deiner Bauteildose (*junkbox*)

Die Frage ist mehr *wie empfindlich reagiert der Arbeitspunkt auf Abweichungen bei* $R_E, R_C, R1, R2, V_{CC}$*?* Dazu kannst Du ja die Ableitungen $\frac{\partial U_C}{\partial R_E}$ usw. bilden



EGS Dimensionierung

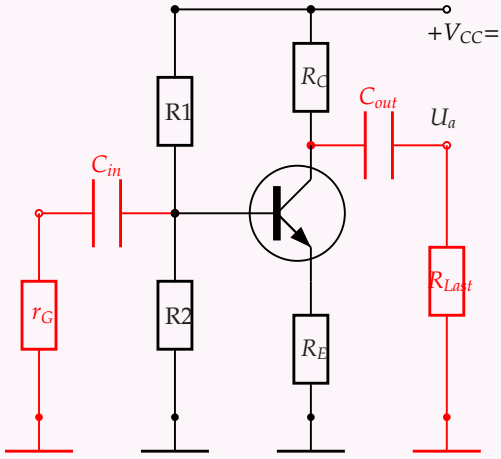
Nun wollen wir ein *kleines* Signal an der Basis *hinzuaddieren* und am Kollektor wieder heraus*trennen*. Dazu benutzen wir sog. *Trenn-* oder *Koppel*-Kondensatoren C_{in} und C_{out}

Ein *Signal* ist eine *zeitabhängige* Funktion f(t,. . .), d.h. da *ändert* sich was.

Für Größen, die **klein und veränderlich** sind, schreiben wir **Kleinbuchstaben** i_B, u_{BE}, r_{CE}, r_G usw.

(Bei einem ohmschen Widerstand R_X ist $r_X = R_X$)

Also Koppel-Kondensatoren:



EGS m.Stromgegenkopplung

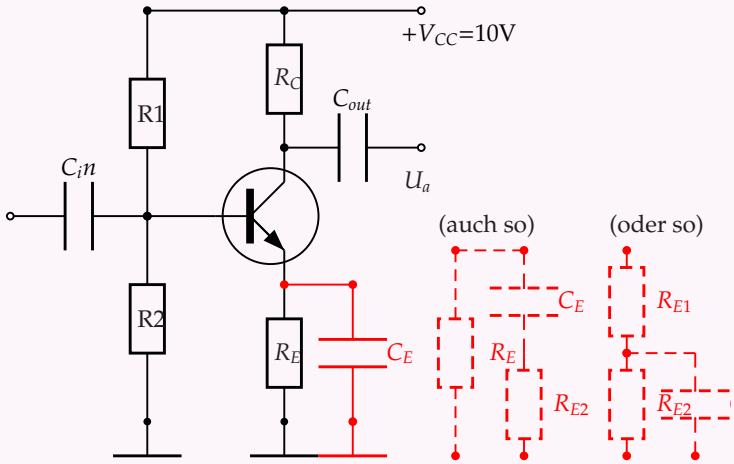
weitere Dimensionierung (Koppelkondensatoren Cin, Cout):

1. Kondensatoren und Widerstände bilden frequenzabhängige *RC-Glieder*.
2. Cin bildet mit (dem per Ersatzschaltbild zu errechnenden) r_{in} einen Hochpass (high pass, low cut) mit der -3dB Grenzfrequenz $\omega_{G,in}=1/r_{in}C_{in}$
3. r_{in} ist die Parallelschaltung
   R1 || R2 || r_B
   in Serie mit r_G (Generator-Innenwiderstand)
   $r_B = \beta^*(r_E + R_E)$
4. $\omega_{G,in}$ machst Du so niedrig, dass all Deine Signal-Frequenzanteile *vernachlässigbare* Abschwächung (attenuation) erfahren.
5. Cout bildet mit (dem per Ersatzschaltbild zu errechnenden) r_{out} einen Hochpass mit der -3dB Grenzfrequenz $\omega_{G,out}=1/r_{out}C_{out}$
6. r_{out} ist die Parallelschaltung
   Rc || r_{CE} (und ggf. || $r_{feedback}$)
   in Serie mit r_L (Last-Innenwiderstand)
   $r_{CE} = (Uy + U_{CE})/I_C$ (Early-Spannung)
7. $\omega_{G,out}$ machst Du so niedrig, dass all Deine Signal-Frequenzanteile *vernachlässigbare* Abschwächung (attenuation) erfahren.

Jetzt haben wir nur noch den Konflikt, den Emitterwiderstand R_E
a) zwecks thermischer und Arbeitspunkt- Stabilisierung **groß** und
b) wegen seiner gegenkopplungsbedingten *Verstärkungsbremswirkung* ($V_S = \frac{Rc}{r_E + R_E}$) **klein** machen zu wollen.

Die Lösung besteht (wieder) in der Auftrennung von Arbeitspunkteinstellung und *Kleinsignalverhalten*:
Ein Kondensator C_E **überbrückt** R_E.
Das ergibt (wieder) ein Hochpass-Verhalten mit der -3dB Grenzfrequenz $\omega_{CE} = 1/R_EC_E$ (welche man wieder niedriger als die unterste Signalfrequenz macht)
(V_S wird dann mit steigender Frequenz größer
$V_S = \frac{Rc}{r_E + (r_{E,eff}||1/j\omega C_E)}$) mit $r_{E,eff} = r'_E||R_E$,
$r'_E = r_E + \frac{r_{B,in}}{\beta}$,
$r_E = 1/s$,
$r_{B,in} = r_G||R1_{B,oben}||R2_{B,unten}$ (f. HF auch noch C_{BE} und CCE einrechnen. . .)

Um dabei die Spannungsverstärkung V_S nach oben (beliebig) begrenzen zu können, schaltet man gelegentlich einen Widerstand R_{E2} mit C_E in Serie ($R_{E2} + C_E$ überbrücken R_E) (strichliert abgebildet)



EGS m.Stromgegenkopplung

Es gelten näherungsweise (bei I_C=1mA, T=298K):

$$V_s = -s * R_C = -\frac{R_C}{r_E} \qquad Spannungsverstärkung\ u_a/u_e \tag{1}$$

$$s = \frac{I_C}{U_T} = \frac{1}{r_E} = \frac{\beta}{r_{BE}} \qquad (40mS) \tag{2}$$

$$r_E = \frac{1}{s} = \frac{U_T}{I_C} \qquad (26\Omega) \tag{3}$$

$$\beta = \frac{I_C}{I_B} \qquad (100) \tag{4}$$

$$r_{BE} = \frac{\beta}{s} = \frac{U_T}{I_B} = \beta * r_E \qquad (2k5) \tag{5}$$

$$R_C \ldots Collector - Arbeitswiderstand\ (load\ resistor) \tag{6}$$

$$U_T \ldots \frac{kT}{q} \ldots 25,7mV \ldots Temperaturspannung \tag{7}$$

$$(k = 1.38E-23 \ldots Boltzmannkonstante) \tag{8}$$

$$(q = 1.60E-19 \ldots Elektronenladung) \tag{9}$$

$$(T \ldots absoluteTemperatur 293.15K, 298.15K oder 300.0K) \tag{10}$$

real mit Strom-Gegenkopplungs-Widerstand R_E

$$V_s' = -\frac{R_C}{r_E + R_E} = -s' * R_C \tag{11}$$

$$(s' = \frac{s}{1 + s * R_E}) \tag{12}$$

$$r_{B,in} = \beta * (r_E + R_E) = \frac{\beta}{s} * (1 + sR_E) = r_{BE} * (1 + sR_E) \tag{13}$$

$$R_E \ldots Emitter - Gegenkopplungswiderstand \tag{14}$$

(15)

real mit Spannungs-Gegenkopplungs-Widerstand R_F und Eingangs-Widerstand R_G

$$V'_s = -\frac{\beta R_C R_F}{R_G[(1+\beta)R_C + R_F]} =\sim R_F/R_G \tag{16}$$

$R_C \ldots Collector - Arbeitswiderstand$ (17)

$R_F \ldots Feedback - Gegenkopplungswiderstand$ (18)

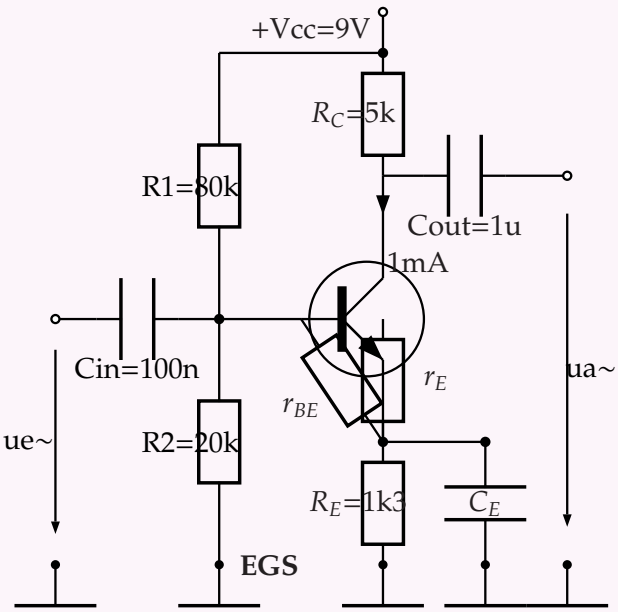
$R_G \ldots Eingangs - plusGeneratorwiderstand$ (19)

(20)

Die Arbeitspunkt - Einstellung erfolgt nach den Daumenregeln

- I_C = 1mA
- U_C = Vcc/2
- U_{BE} = 0,62V (0,7V)
- U_{R_E} = 10% von Vcc, jedoch mindestens 1V
- Basisspannungsteiler - Querstrom = 10* I_B (also ca. 100μA)

Etwas schlampig dimensioniert (~1k2, ~20k, ~80k, 9Vcc) ergibt sich:

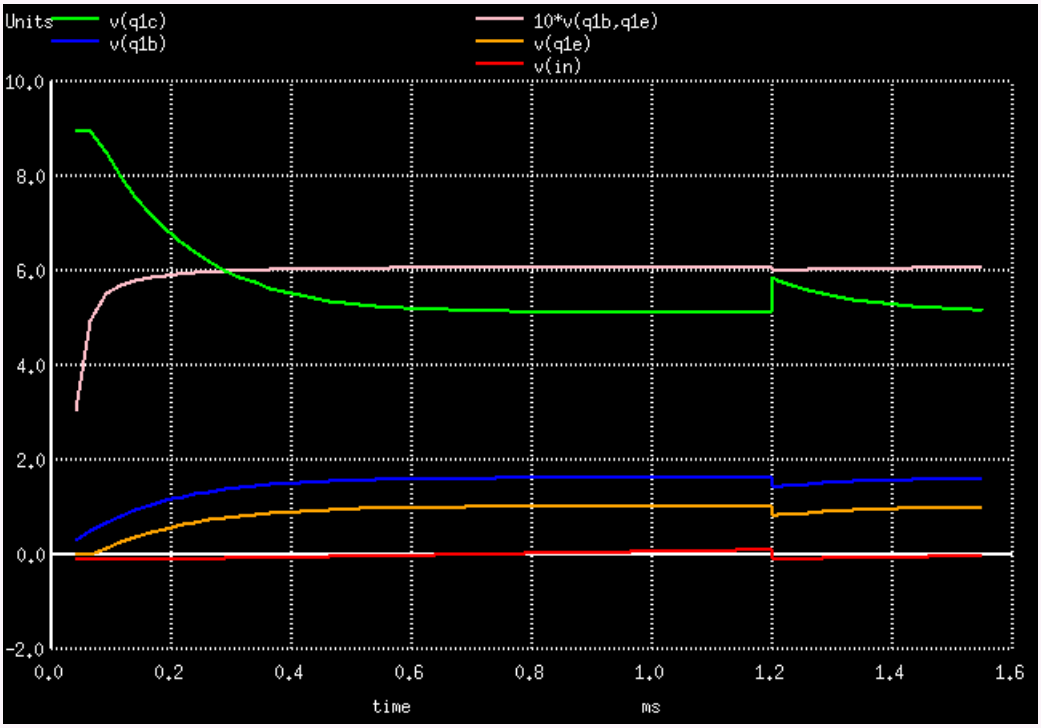


Listing 1: 11egs01Rc5k.cir SPICE Netzliste

```
* BJT in EGS EmitterGrundschaltung mit RE-Gegenkopplung Oc01
*** Strom-Versorgung ***
Vcc     plus    0       DC 9.0V

***Input-Signal:***
Vin     in      0       DC 0 PULSE( -100m +100m 200us 998us 1us  1us  1ms  )
*                                   Vmin  Vmax  td    tr    tf   ton  tperiode

***EGS: Q1, Rc, RE
Rc      plus    Q1C     5k
Q1      Q1C Q1B Q1E     BC550B
RE      Q1E     0       1.3k
R1      plus    Q1B     82k
R2      Q1B     0       18k
Cin     in      Q1B     10nF

.control
***TRANsient analysis: ************************************************
TRAN    25us    1.55ms  25us    UIC
*       steppW  t_stop  t_start
PLOT    V(in),  V(Q1B), v(Q1E), v(Q1C), 10*V(Q1B,Q1E)
**************************************************************************
.endc
.MODEL BC550B NPN (IS=50f VAF=100 BF=400)
.end
```

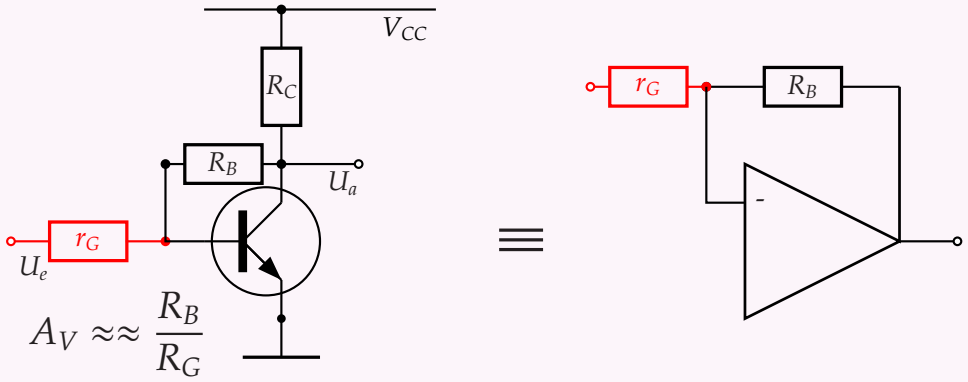
Listing 2: 11egs01Rc5k.cir SPICE Protokoll

```
sh-4.2\$ ngspice /SMUE/shr/c/spice3/x/Kleinsig/egs/11egs01_Rc_5k.cir
******
** ngspice-25 : Circuit level simulation program
** The U. C. Berkeley CAD Group
** Copyright 1985-1994, Regents of the University of California.
** Please get your ngspice manual from http://ngspice.sourceforge.net/docs.html
** Please file your bug-reports at http://ngspice.sourceforge.net/bugrep.html
** Creation Date: Tue Aug 27 22:45:45 UTC 2013
******

Circuit: * bjt in egs emittergrundschaltung mit re-gegenkopplung oco1

Doing analysis at TEMP = 27.000000 and TNOM = 27.000000

Initial Transient Solution
--------------------------

Node                                   Voltage
----                                   -------
plus                                         0
in                                           0
q1c                                          0
q1b                                          0
q1e                                          0
vin#branch                                   0
vcc#branch                                   0

No. of Data Rows : 81
ngspice 2 ->
```



EGS 9V,1mA,80k-20k-5k-1k3, green=Uc, white=10*Ube

Den EGS kannst auch mit *Spannungsgegenkopplung* bauen:



$$I_C \approx \frac{U_{R_C}}{R_C}$$
$$I_B \approx \frac{U_{R_B}}{R_B}$$
$$B = \frac{I_C}{I_B} = \frac{U_{R_C}/R_C}{U_{R_B}/R_B} = \frac{R_B * U_{R_C}}{R_C * U_{R_B}}$$
$$\frac{U_{R_C}}{U_{R_B}} = \frac{B * R_C}{R_B}$$

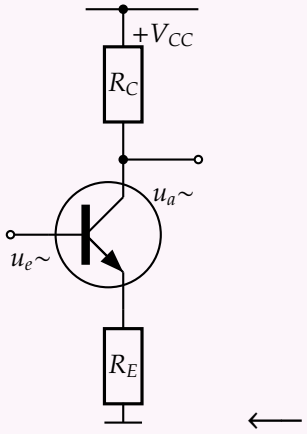
$$Vcc7 ::= Vcc - 0.7 \rightarrow U_{R_C} = Vcc7 - U_{R_B}$$
$$\frac{Vcc7 - U_{R_B}}{U_{R_B}} = \frac{B * R_C}{R_B}$$
$$Vcc7 - U_{R_B} = U_{R_B} * \frac{B * R_C}{R_B}$$
$$Vcc7 = U_{R_B} * [\frac{B * R_C}{R_B} + 1]$$
$$U_{R_B} = \frac{Vcc7}{B * R_C/R_B + 1}$$

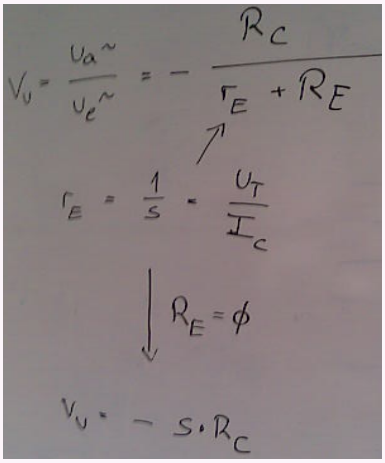
Das Verhältnis R_B/R_C verringert sich nahe der $U_{CE,sat}$-Sättigung (< ca. 1.0V).
Bei kleinem R_B/R_C geht also die U_{CE} soweit pässlich katholisch *in die Knie*, bis das 'B' zum R_B/R_C stimmt.

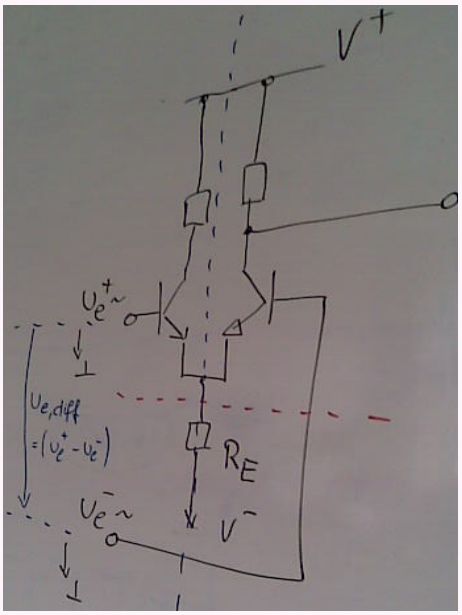
Schaut aus, als könnte man so wesentlich höhere Spannungsverstärkungen erreichen — nur — die Linearität wird auch entsprechend *miserabel*, zudem rauscht der grosse R_B (therm. Rauschspannung: $\sqrt{(4)k_{Bolzm.} * T_{Kelvin} * BW_{Hz} * R_\Omega}$). Daher findet man solche Stufen am ehesten in AM-Funkgeräteverstärkern, die sowieso krachen und fauchen, oder fernöstlichem MP3-Schrott für den zeitgenössischen Nervtöterkrawall, der durchweg eh nur nach Getriebeschaden klingt.

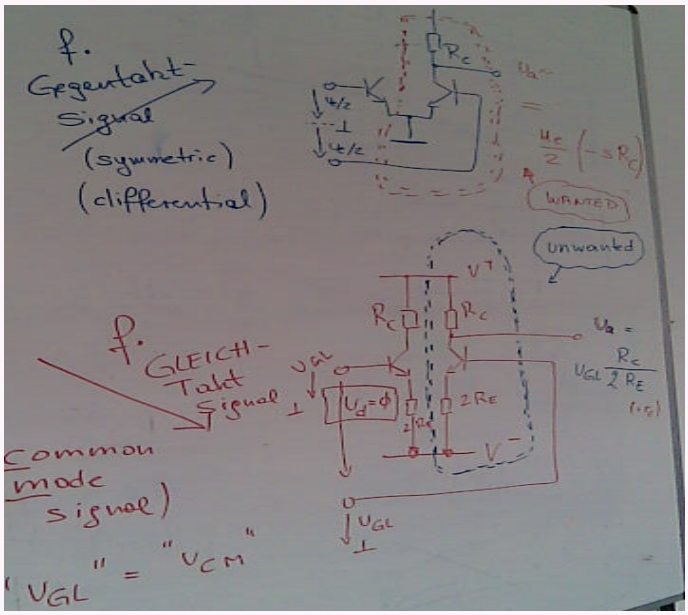
1 .2 BJT Differenz-Stufe, differential stage, long-tailed pair

1 .2.1 Tafelbilder 29Sep15



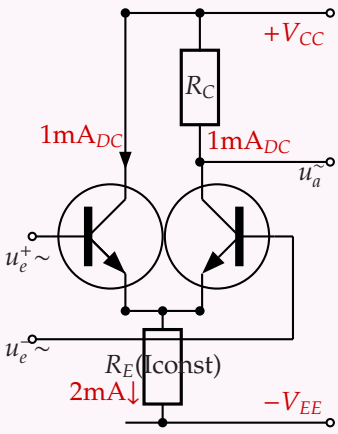






BJT Differenzverstärkerstufe differential stage (DiffStg)

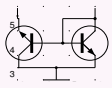
DC-Arbeitspunkt: Was tun die zwei?



- die *'raufen'* sich um den Emitterstrom:

  sobald ein BJT **mehr** Strom leitet (wodurch U(R_E) steigt) → kriegt der andere um dasselbe **weniger**

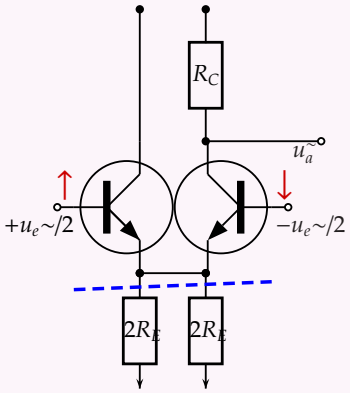
  und umgekehrt.
- gute DiffStg haben eine Konstantstromquelle *Iconst* als R_E
  → wir zeichnen hier **nur zum Rechnen** deren hohen Innenwiderstand R_E(Iconst).
- als Iconst nimmt man *Stromspiegel* (


) und *FET Stromquellen* (

)

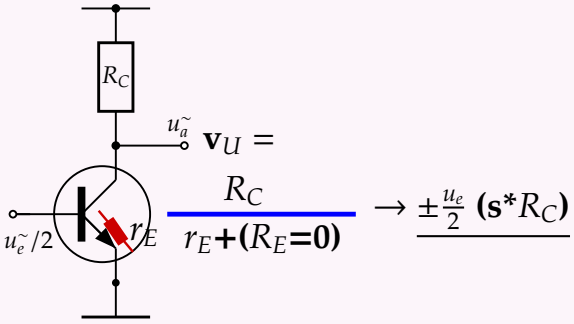
- im IC fällt R_S weg, der FET-Kanalwiderstand wird per Dotierung/ Weite höher fabriziert.

Gegentakt- Kleinsignal- Verstärkung, **differential)** small signal amplification:

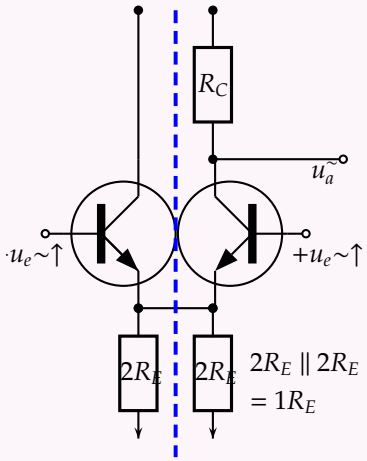


(u_e links um $+u_e/2$ steigern und rechts um $-u_e/2$ senken ergibt in Summe ein ganzes u_e!)

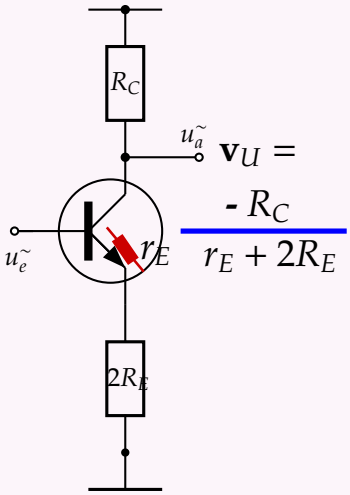
Beidseits **gegen**sinnige Ansteuerung - man nimmt dem einen und gibt es dem anderen:
→ Strom durch R_E bleibt **unverändert!**
↓
Emitterspannung **unverändert!** → wirkt wie ohne R_E, direkt an *Masse*
→ wir rechnen ohne R_E, dh. mit $R_E = 0$:



Gleichtakt- Kleinsignal- Verstärkung, **common mode** small signal amplification:
Beidseits gleiche Ansteuerung $+u_e{\sim}\uparrow$

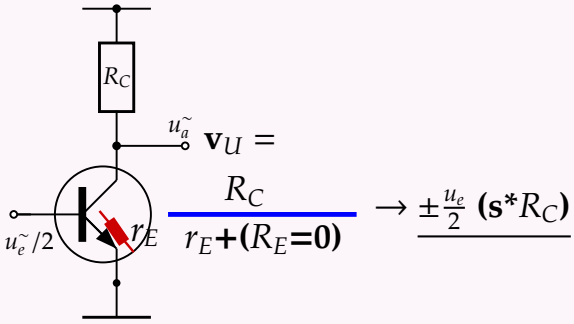


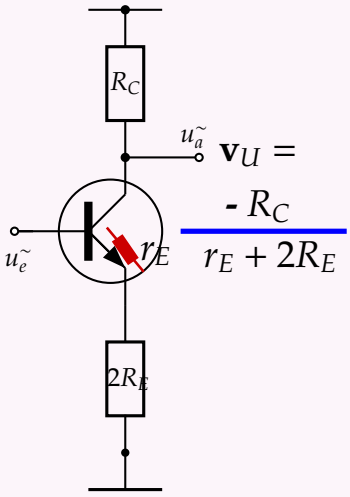
Beidseits gleiche Ansteuerung → identische Reaktion → brauche nur eine Seite zu rechnen (→ halbe Schaltung):



R_E ist ein Stromquellen--Innenwiderstand und folglich sehr hoch (Mega-Ω); da kann man $r_E = U_T/I_C = 25\text{mV}/1\text{mA} \approx 25\Omega$ ruhig **ignorieren**.

Gegentakt $\overset{contra}{\longleftrightarrow}$ Gleichtakt,
differential $\overset{versus}{\longleftrightarrow}$ common mode:





Wir haben also:

$$v_U = \frac{R_C}{25} \longleftrightarrow \frac{R_C}{1Mio} \text{ (grob)}$$

das sind $\frac{1Mio}{25} \approx 40000 \equiv 86\text{dB}$

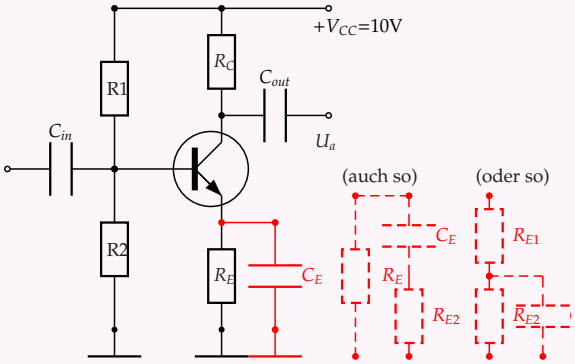
'Gleicktaktunterdrückung',
'common mode rejection ratio' (CMRR)

1 .3 BJT+FET Grundschaltungen

1 .3.1 EGS

die EGS ist ja bereits gut bekannt.
Dimensionierung Arbeitspunkt:

- I_C festlegen
- R_E= 0.1 Vcc, aber mindestens 1.0V
- $R_C = \frac{Vcc - V(R_E)}{2 \cdot I_C}$
- $R1 = \frac{Vcc - 0.7V - V(R_E)}{10 \cdot I_C/\beta}$
- $R2 = \frac{0.7V + V(R_E)}{10 \cdot I_C/\beta}$



EGS m.Stromgegenkopplung

Dimensionierung Kleinsignalverhalten:

- $v_U = -\frac{R_C}{Z_E + U_T/I_C} \quad (\overline{Z_E=0}) \quad -s \cdot R_C$
  mit $s = \frac{I_C}{U_T}$
  $U_T = \frac{kT}{q}$ (ca. 26mV)
  k ... 1.38E-23 Boltzmannkonstante
  T ... 298.15K Kelvin-Temperatur bei 25°C
  1 ... 1.6E-19 Elektronenladung
- $r_{Eingang} = Z_{in} + R1||R2||r_{be}$
- $r_{Ausgang} = r_{a,BJT}||R_C + Z_{out}$

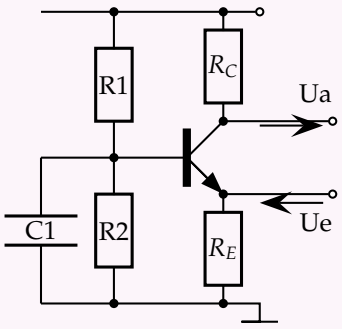
Leistungsverstärkung (power gain):

$$v_P = \frac{Pout}{Pin} \approx \frac{u_a^2/R'_C}{u_e^2/(\beta R'_E)} = \frac{(u_e \cdot R'_C/R'_E)^2/R'_C}{u_e^2/(\beta R'_E)} = \frac{u_e^2 \cdot {R'_C/R'_E}^2}{u_e^2/(\beta R'_E)} = \beta \frac{R'_C}{R'_E}$$

Spannungsverstärkungen um +20dB sind gut erreichbar. Kleine Betriebsspannungen von 5V oder 3.3V (heute verbreitet) erfordern aber R_E- Parallelkondensatoren.

1 .3.2 BGS

Basisgrundschaltung. (Basis signalmäßig (via C1) auf GND)



Dimensionierung Arbeitspunkt:

- I_C festlegen
- U_B festlegen
- $R_E = \frac{U_B - 0.7}{I_C}$
- $R_C = \frac{Vcc - U_B}{2 \cdot I_C}$
- $R_{B1} = \frac{Vcc - 0.7V - V(R_E)}{10 \cdot I_C/\beta}$
- $R_{B2} = \frac{0.7V + V(R_E)}{10 \cdot I_C/\beta}$

Dimensionierung Kleinsignalverhalten:

- $v_U = +s \cdot R_C = \frac{R_C}{U_T/I_C} = \frac{I_C \cdot R_C}{U_T}$
- $r_{Eingang} = R_E||(r_b/\beta) \quad (+Z_{KoppelCap})$
- $r_b = R_{B1}||R_{B2}||Z_{C1}$ (klein wegen C1!)
- $r_{Ausgang} = r_{a,BJT}||R_C \quad (+Z_{KoppelCap})$

Leistungsverstärkung (power gain):

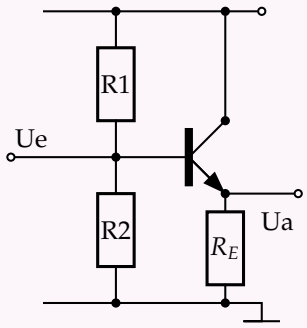
$$v_P = \frac{Pout}{Pin} \approx \frac{u_a^2/R'_C}{u_e^2/r_E} = \frac{(u_e \cdot s \cdot R'_C)^2/R'_C}{u_e^2 \cdot s} = \frac{(u_e^2 \cdot s^2 \cdot {R'_C}^2)/R'_C}{u_e^2 \cdot s} = s \cdot R'_C = \frac{R'_C}{r_E} = R'_C \frac{I_C}{U_T}$$

Wird verwendet für:
- Vermeidung des Miller-Effekts bei HF
- Differenzverstärker mit 2 BJT (wie OP und AudioAmp)

- Spannungsverstärkung in OPs
- Logik-Eingang (s.TTL-NAND)
- 'cascode' Schaltung (auch Schalt-FETs)

1 .3.3 CGS

Collectorgrundschaltung. (Collector an Vcc = via U-Quelle auf GND)



Dimensionierung Arbeitspunkt:

- I_C festlegen
- grob: $R_E = \frac{Vcc}{2 \cdot I_C}$
  $R1 = R2 = 10 \cdot R_E$
- $U_E = \frac{Vcc - 0.7}{2}$
- $R_E = \frac{U_E}{I_C}$
- $R_{B1} = \frac{Vcc/2 - 0.35V}{10 \cdot I_C/\beta}$
- $R_{B2} = \frac{Vcc/2 + 0.35V}{10 \cdot I_C/\beta}$

Dimensionierung Kleinsignalverhalten:

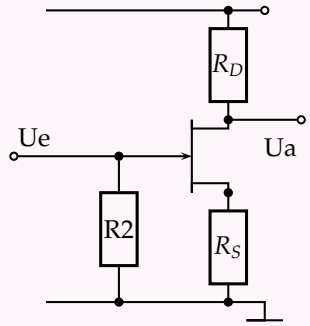
- $v_U = 0.97 \,..\, 0.99$ "Emitterfolger"

  (genauer:
  $$\frac{u_a}{u_e} = \frac{(\frac{\beta+1}{\beta})sR'_E}{(\frac{\beta+1}{\beta})sR'_E + 1} \approx \frac{sR'_E}{sR'_E + 1}$$
  (Tietze, Schenk 12, S.137))
- $r_{Eingang} = R_{B1} || R_{B2} || r_{be}$
- $r_{Ausgang} = r_{be}/\beta || R_E$

Wird verwendet für:
- Impedanzwandlung (Spannungsfolger)
- Stromverstärkung
- Entkopplung
- oft wird R_L direkt anstelle R_E angeschaltet

1 .3.4 jFET

Sourcegrundschaltung. (Drain an Vcc = via U-Quelle auf GND)
mit Gatevorspannung durch Sourcewiderstand
vgl. Radioröhren: *("Gittervorspannung durch Kathodenwiderstand")*
ohne Koppel-Caps:



Steuerkennlinie: $I_D = I_{DSS}(1 - \frac{V_{GS}}{V_P})^2$
$\rightarrow$
$V_{GS} = V_P \cdot (1 - \sqrt{I_D/I_{DSS}})$

Dimensionierung Arbeitspunkt:

- I_D festlegen
- $R1 = \infty$
- $R2 = 100..1000k$
- $V_{GS} = V_P \cdot (1 - \sqrt{I_D/I_{DSS}})$
- $R_S = \frac{V_{GS}}{I_D} = \frac{U_P}{I_D}(1 - \sqrt{I_D/I_{DSS}})$
- $R_D = \frac{Vcc - V_{SG}}{2 \cdot I_C}$

Dimensionierung Kleinsignalverhalten:

- $s = \frac{1}{r_S} = \frac{2}{|V_P|}\sqrt{I_D \cdot I_{DSS}}$

Herleitung: $s ::= \frac{\partial I_D}{\partial V_{GS}}$

$$\left[I_D = I_{DSS}\frac{(V_P - V_{GS})^2}{V_P^2}\right]$$

$$s = \frac{I_{DSS}}{V_P^2} \cdot 2 \cdot \underbrace{(V_P - V_{GS})}(-1)$$

$$\downarrow$$

$$\left[\underbrace{\frac{V_P - V_{GS}}{V_P} = \sqrt{I_D/I_{DSS}}}\right]$$

$$\downarrow$$

$$s = \frac{2 \cdot I_{DSS}}{V_P} \cdot \sqrt{I_D/I_{DSS}})$$
$$= \frac{2}{V_P} \cdot \sqrt{I_D \cdot I_{DSS}})$$

- $r_D = \frac{U_y + V_{DS}}{I_D}$ (U_y...*Earlyspannung*)
- $v_U = -\frac{R_{D,ges}}{r_S + R_S}$
- $r_{Eingang} = R2 \parallel Z_{C_{GS}+v_U \cdot C_{DS}}$
- $r_{Ausgang} = r_D \parallel R_D + Z_{out}$

Leistungsverstärkung (power gain):

$v_{P(DC)} = \infty$ (wegen $I_G = 0$)

1 .4 simple bipolar audio amp

Listing 3: ASCII-Art AudioAmp

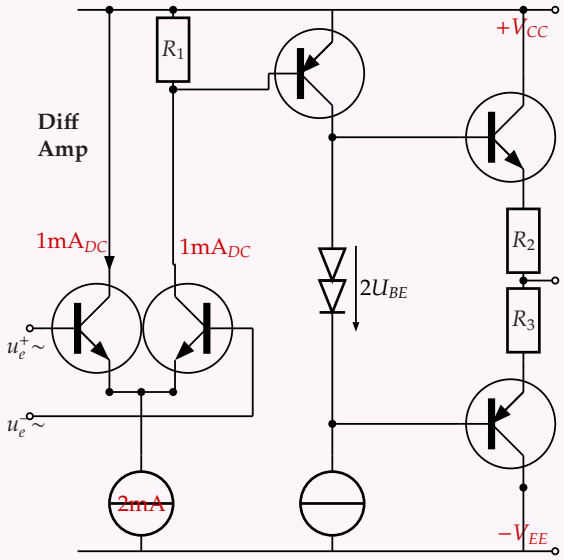
```
    ---+---+--------+--------+----o +Vcc
       !   R1       v        !
       !   +-------K         !                Prinzipschaltung:
       !   !        \        /                funktioniert so nicht
  E-   /   \         +-------K                einwandfrei!
   o--K    T2--+     !       v
       v   v   !     D1       R2
       +-+-+   !     !        +-----o out
         !     !     D2       R3
   o-----!-----+     !        v
  E+     !           +-------K
         !           !       \
        (-)2mA      (-)2mA   !
         !           !       !
      ---+----------+--------+----o -Vcc
     differen-     level     final
     tial Amp.     shifter   stage,
     long-tailed   Pegel-    push-pull,
     pair          Schieber  Gegentaktendstufe

 =>warum hat T1   =>wozu die    =>wozu R2
   keinen Rc ??     Dioden ??     und R3 ??
```

T1, T2, T4:	TUN ($\triangleq$ transistor universal NPN zB BC550)
T3:	TUP ($\triangleq$ transistor universal PNP zB BC560)
T5:	Power PNP, BD140, BD249, BD245, BD243, BD680, TIP147 usw.
D1, D2	DUS ($\triangleq$ diode universal silicium zB 1N4148)
R1:	~ 650Ω (zB 470+180 od. 680 evt. ‖15k)
R2, R3	'wenig' Ohm (rechnen!), so 0.22 . . . 1.0 . . .

(In ICs macht man alles aus Transistoren!
"bipolar baut man schon 25 Jahre nicht mehr - 'heute' ist alles MosFET")
(DI Helmut Schönherr, Infineon Villach, 21Sep15)

1 .4.1 bipolar NF Verstärker, BJT AF[*3] Amp

Prinzipschaltung

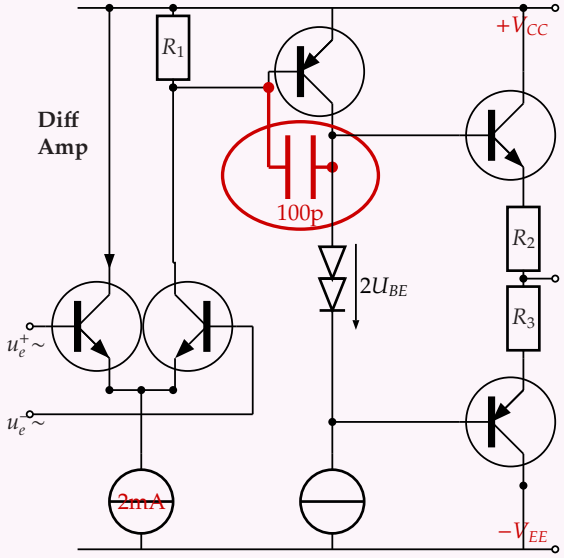


funktioniert so nicht:

- ≠ Temperaturstabilisierung
- ≠ Frequenzgangkompensation
- ≠ Kurzschlussschutz
- ≠ U_{BE} Vervielfacher

[3] AF (audio frequency) ≡ NF (Niederfrequenz)

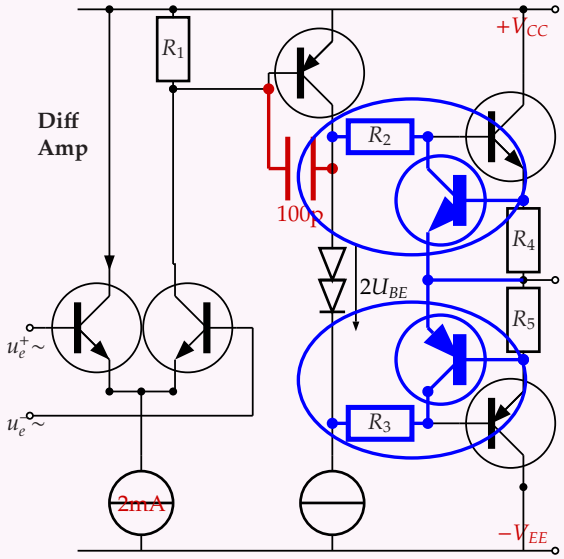
Frequenzgangkompensation:



- ⧾ Temperaturstabilisierung
- ⨁ **Frequenzgangkompensation**
- ⧾ Kurzschlussschutz
- ⧾ U_{BE} Vervielfacher

mit Kurzschlussschutz (short sircuit protection)

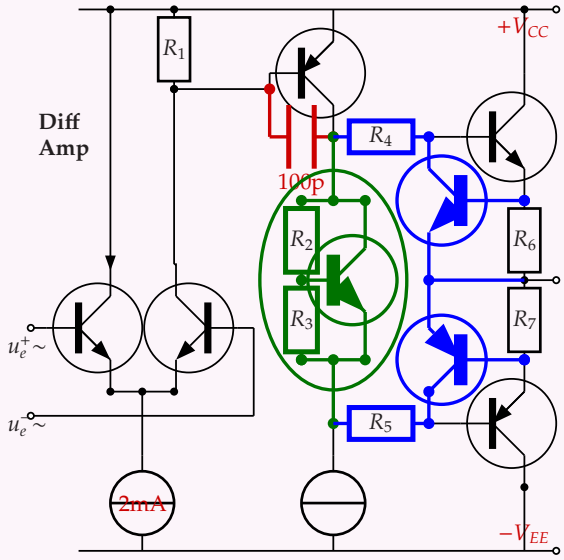
short circuit protection:



- ⧾ Temperaturstabilisierung
- ⨁ **Frequenzgangkompensation**
- ⨁]**Kurzschlussschutz**

  $I_{max} = \frac{620mV}{R4}$ *bzw.* $\frac{620mV}{R5}$
- ⧾ U_{BE} Vervielfacher:

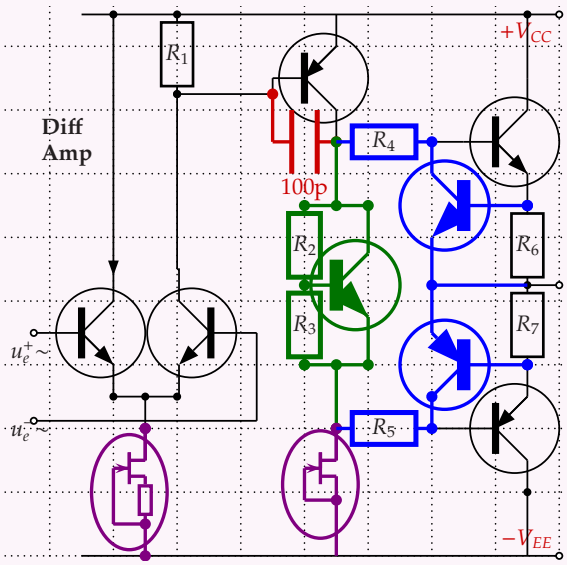
mit U_{BE} Vervielfacher:



- Temperaturstabilisierung
- ⊕ **Frequenzgangkompensation**
- ⊕ **Kurzschlussschutz**
- ⊕ U_{BE} **Vervielfacher**

$$\frac{R3}{R2+R3} = \frac{620mV}{U_{CE}}$$

jFET constant current sources:



- Temperaturstabilisierung
- ⊕ **Frequenzgangkompensation**
- ⊕ **Kurzschlussschutz**
- ⊕ U_{BE} **Vervielfacher**
- ⊕ **jFET Konstantstromquellen**

$$R_S = \frac{U_p}{I_D} * (1 - \sqrt{\frac{I_D}{I_{DSS}}}) \quad \text{... (Böhmer)}$$

$$= \frac{2V}{2mA} * (1 - \sqrt{\frac{2mA}{5mA}}) \ [\Omega]$$

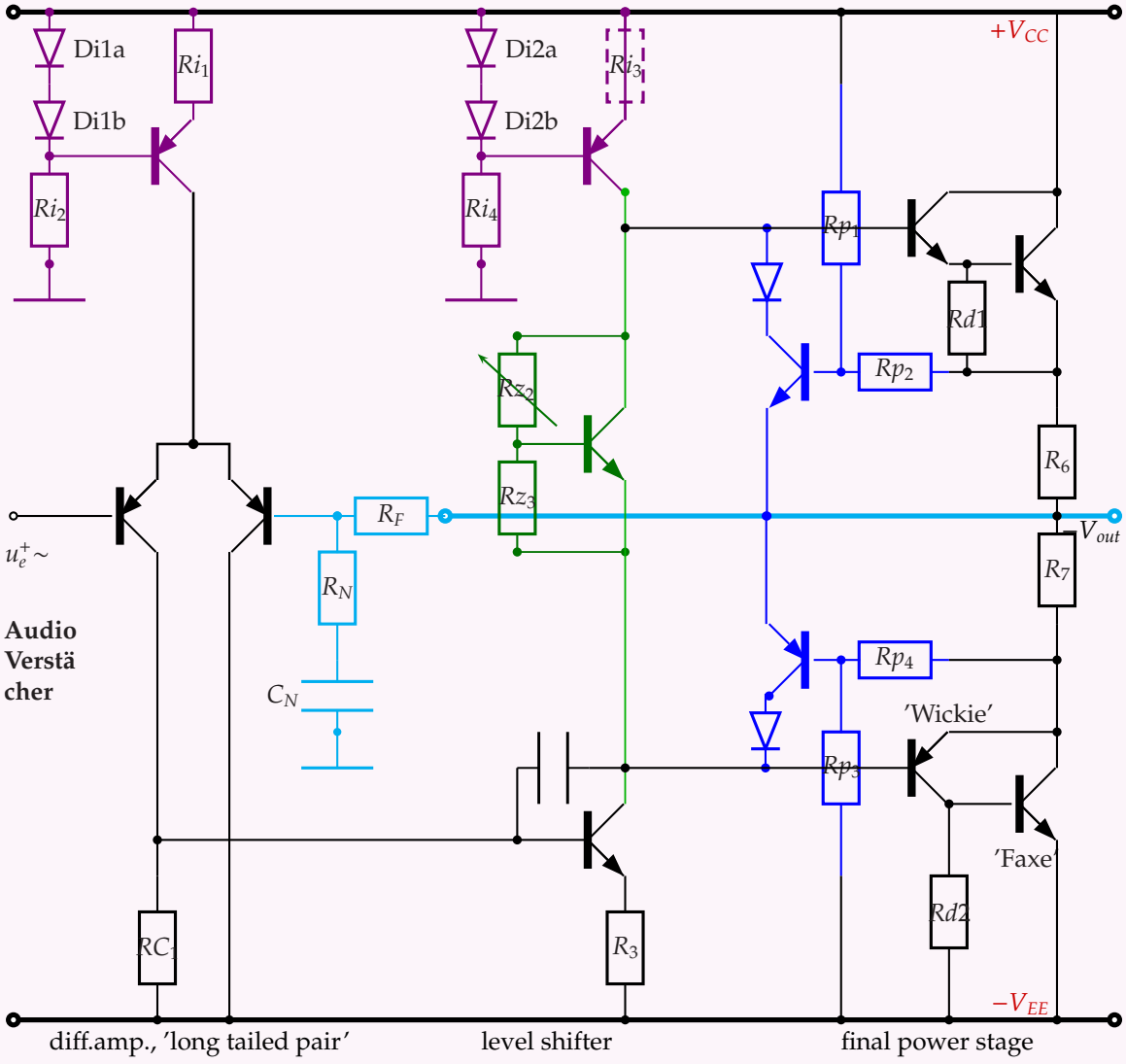
$$= 1000 * (1 - \sqrt{0.4}) \ [\Omega]$$

$$= 1000 * (1 - 0.63) \ [\Omega]$$

$$= 1000 * (0.37) \ [\Omega]$$

$$= 370\Omega$$

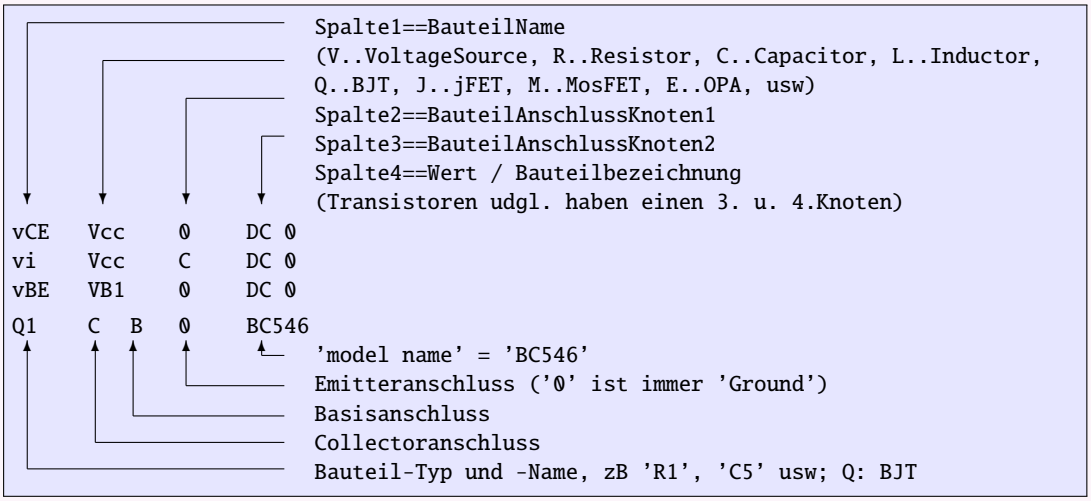
1 .4.2 der VerStecher V0.02 (Fehlersuch)



1 .5 SPICE net list

1 .5.1 SPICE net list
ngSPICE Simulator

- Die SPICE net list ist eine ASCII-Text-Datei, also mit Programm-Text-Editoren bearbeitbar
- '*' beginnt eine Kommentarzeile
- SPICE net lists beschreiben Elektronikschaltungen für den SPICE Simulator ('Simulation Program with IC Emphasis')
  zuerst sind den Schaltungsknoten Namen oder Nummern zu geben
  (iXH bevorzuge Namen wie 'Vcc', 'GND', 'in', 'out', 'T1C' . . .)
  der Knoten '0' muss existieren und bedeutet GND
  jede Zeile beschreibt ein Bauteil.
  1te Spalte: Bauteil
    R... Resistor
    C... Capacitor
    L... Inductor
    V... Voltage Source
    I... Current Source
    Q... BJT
    J... jFET
    M... MosFET
  2te Spalte: Anschlussknoten 1
  3te Spalte: Anschlussknoten 2
  (n-te Spalte: Anschlussknoten n, zB. bei Transistoren)
  letzte Spalte: Wert



- Beispiel:

```
Vplus      Vcc       0         5.0
Vsignal    in        0         DC 0 PULSE( 200m +2.40 200us 998us 1us 1us 1ms )
*                                        Vmin Vmax td tr tf ton tperiode
Rb         in        T1B       100k
Rc         Vcc       T1C       5k
Q1         T1C       T1B       0     BC547B
Cout       T1C       out       100n
RLast      out       0         8
```

```
.control
TRAN       25us      15.5ms    14ms       UIC
*          steppW    t_stop    t_start
PLOT       V(in),    V(T1C),   v(out)
.endc
```

```
.MODEL     BC547B    NPN       (IS=50f VAF=100 BF=400)
.end
```

- **.MODEL cards** beschreiben die Halbleiter-Eigenschaften.
  → Du brauchst *keine Bauteil-Bibliothek!*

  zB:

```
.Model   DUS   D
.model  1N4148  D  (is=2.495E-09 rs=4.755E-01 n=1.679E+00
+    tt=3.030E-09 cjo=1.700E-12 vj=1 m=1.959E-01 bv=1.000E+02 +    ibv=1.000E-04)
```

  oder:

```
.model  Z5V6   D   BV=5.6
.model  Zener9V1  D  (Is=0.6u Rs=.5 Cjo=110p nbv=6 bv=9.1 Vpk=9.1 type=zener )
.model  Zener60V  D  (Is=0.6u Rs=.5 Cjo=110p nbv=6 bv=60 Vpk=60 type=zener )
.model  1N4002  D  (IS=1n RS=5m N=1.65 TT=6u CJO=10p M=0.333 BV=100 IBV=1E-5
+    Iave=1 Vpk=100 type=silicon)
```

  oder:

```
.Model  BC550  NPN  ( is=10f vaf=100 bf=250)
.model  2N7000  NMOS  (Level=3 Gamma=0 Delta=0 Eta=0 Theta=0 Kappa=0.2 Vmax=0 Xj=0
+    Tox=2u Uo=600 Phi=.6 Kp=1.073u W=.12 L=2u Rs=20m Vto=1.73
+    Rd=.5489 Rds=48MEG Cgso=73.61p Cgdo=6.487p Cbd=74.46p Mj=.5
+    Pb=.8 Fc=.5 Rg=546.2 Is=10f N=1 Rb=1m)
```

- die **.control section** steuert den Simulationsprozess (s. 'TRANsient' und 'PLOT')

  zB:

```
*Parallelschwingkreis
v1   1   0   dc 0 ac 1
vi   1   2   dc 0
c1   2   0   5u
l1   2   3   5m
r1   3   0   10
```

```
.control
  ac dec 100 10 100k
  let z=v(1)/i(vi)
  plot mag(z) loglog
  plot mag(z) linear xlimit 0 2500 ylimit 0 150
  plot z polar xlimit 0 150
  set units=degrees
  plot ph(z)
.endc
```

```
.end
```

1 .5.2 LTSpice

- *download site for the software (info by Lor HiSpeed Noname):*
  https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html#

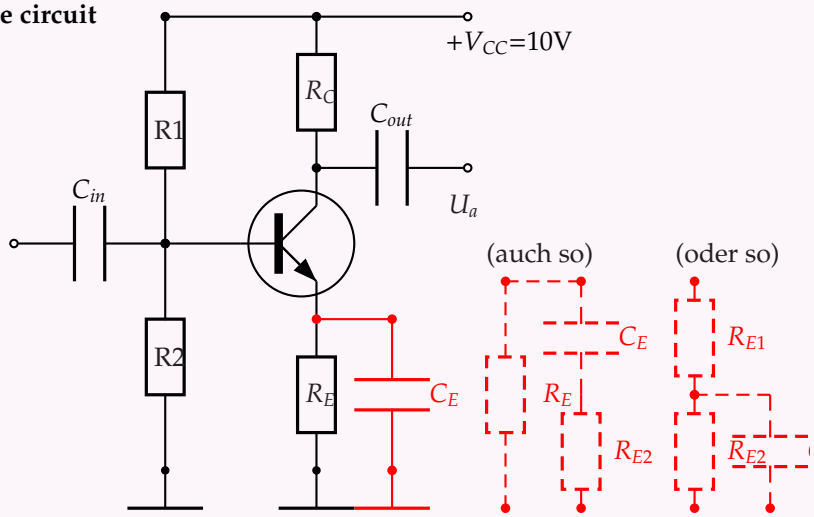
  used tutorials:
- https://www.youtube.com/watch?v=MsilYqaGPQw
- https://home.zhaw.ch/hrt/LTspice/LTSpice%20Tutorial%201%20V03.pdf
- http://www.home.hs-karlsruhe.de/~klal0001/download_frei/LTspice%20IV%20Tutorial_Tritschler_Kloenne.pdf

1 .5.3 EGS Amp SPICE net list

Let's begin with the *common (grounded) emitter* amplifier circuit including coupling Caps (EGS Emittergrundschaltung mit Koppel-C), the first SPICE program beeing a 'ngspice' simulated AC sweep with bode plots, secondly the same circuit but with 1kHz ramp input to show the impact of hi-cut and low-cut on more realistic wave forms (small distortion can hardly be seen on a sine wave, but very clear in the straightness of a ramp slope; the sudden downstep in the *sawtooth* wave discovers 'overshooting', oscillation tendency or slowness of low pass behaviour)

So, feel welcome to have a look at 'em:

the circuit



EGS m.Stromgegenkopplung

AC sweeping the EGS

Listing 4: SPICE: EGS AC sweep

```
* BJT in EGS EmitterGrundschaltung m.KoppelC
* Mess-Signal: Rampe
*       warum: is, wie wenn ma von Hand gleichmaessig groesser dreht und wieder zrugg

Vplus   Vcc     0       DC 10

***Input-Signal:***
Vin     in      0       DC 0 AC 1
*Vin    in      0       DC 0 PULSE( -300m +300m 200us 998us 1us  1us  1ms  )
*                                  Vmin  Vmax  td    tr    tf   ton  tperiode

CKin    in      Q1B     47nF
RQ1B1   Vcc     Q1B     82k
RQ1B2   Q1B     GND     18k
RQ1C    Vcc     Q1Cout  5k
viQ1C   Q1Cout  Q1C     DC 0
Q1      Q1C Q1B Q1E     BC550B
RE1     Q1E     RE12    330
RE2     RE12    0       910
CE      RE12    0       2uF

***Ausgang***
CKout   Q1Cout  out     22nF
Rlast   out     GND     10k


.control
***TRANsient analysis: ****************************************************
```

```
AC      DEC 10  100 20kHz
PLOT    mag(v(in))      mag(v(out)) loglog
*TRAN   25us    15.5ms  14ms    UIC
*       steppW  t_stop  t_start
*PLOT   V(in),  V(Q1B), v(Q1E)
*PLOT   V(Q1B), v(Q1E), v(Q1C)
*PLOT   v(Q1B,Q1E),     v(out)
*********************************************************************
.endc
.MODEL 1N4148 DMOD
.MODEL BC550B NPN (IS=18f VAF=100 BF=400)
.MODEL BC560B PNP (IS=50f VAF=100 BF=250)
.end
```

EGS at ramp input

Listing 5: SPICE: EGS(ramp)

```
* BJT in EGS EmitterGrundschaltung m.KoppelC
* Mess-Signal: Rampe
*       warum: is, wie wenn ma von Hand gleichmaessig groesser dreht und wieder zrugg

Vplus   Vcc     0       DC 10

***Input-Signal:***
*Vin    in      0       DC 0 AC 1
Vin     in      0       DC 0 PULSE( -300m +300m 200us 998us 1us  1us  1ms  )
*                               Vmin  Vmax  td    tr    tf   ton  tperiode

CKin    in      Q1B     47nF
RQ1B1   Vcc     Q1B     82k
RQ1B2   Q1B     GND     18k
RQ1C    Vcc     Q1Cout  5k
viQ1C   Q1Cout  Q1C     DC 0
Q1      Q1C Q1B Q1E     BC550B
RE1     Q1E     RE12    330
RE2     RE12    0       910
CE      RE12    0       2uF

***Ausgang***
CKout   Q1Cout  out     22nF
Rlast   out     GND     10k


.control
***TRANsient analysis: ***************************************************
**AC    DEC 10  100 20kHz
**PLOT  mag(v(in))      mag(v(out)) loglog
*
TRAN    25us    15.5ms  14ms    UIC
*       steppW  t_stop  t_start
*
PLOT    V(in),  V(Q1B), v(Q1E)
*PLOT   V(Q1B), v(Q1E), v(Q1C)
PLOT    v(Q1B,Q1E),     v(out)
*********************************************************************
.endc
.MODEL 1N4148 DMOD
.MODEL BC550B NPN (IS=18f VAF=100 BF=400)
.MODEL BC560B PNP (IS=50f VAF=100 BF=250)
.end
```

Listing 6: NPN Audio Amp

```
*bipolar NPN Audio diff.Amp 23Okt15 bnamp05.cir

*** Strom-Versorgung ***
```

```
Vcc     plus    0       DC 20
Vee     0       minus   DC 20


***Input-Signal:***
Vin     in      0       DC 0 SIN( 0 600m 1kHz )
*                       0dbm/600Ohm == 775mVeff (P*R=U^2 -> U=sqrt(0.6)=0.775)
Cin     in      Bdf1    10uF
RBdf1   Bdf1    0       10k
*               RBdf1 verbindet DC-maessig f.Arbeitspunkt (operating point)
*               f.Signale bilden Cin/RBdf1 einen Spannungsteiler
*               so erreicht man f.d. 'unwanted DC-Offset' KLEINERE Verstaerkung


***Differential Amplifier diffAmp "df1"***
*---T1:
vX      plus    plusX   DC 0
RCdf1   plusX   Cdf1    620
Qdf1    Cdf1    Bdf1    Edf1    BC550B
IEdf1   Edf1    minus   2mA     *constant current source CCS
*---T2:
vY      plus    plusY   DC 0
RCdf2   plusY   Cdf2    1m
Qdf2    Cdf2    BaseDf2 Edf1    BC550B
RBdf2   BaseDf2 Bdf2b   3.3k
CBdf2   Bdf2b   0       220n


***Pegel-Schieber LevelShifter "ls"***
*---T3:
vZ      plus    plusZ   DC 0
REls1   plusZ   Els1    1.2m
Qls1    A_D1    Bls1    Els1    BC560B
D1      A_D1    A_D2    1N4148
D2      A_D2    K_D2    1N4148
ICls1   K_D2    minus   2mA     *constant current source CCS
RBls1   Cdf1    Bls1    1


***EndStufe FinalStage "fs"***
*---T4+5:
Qfs1    plus    A_D1    Efs1    BC550B
Rfs1    Efs1    out     1.0
Rfs2    out     Efs2    1.0
Qfs2    minus   K_D2    Efs2    BC560B


***Feed-Back "fb"***
Rfb2    out     BaseDf2 110k

***Lautsprecher***
RL      out     0       1k

.MODEL 1N4148 DMOD
.MODEL BC550B NPN (IS=50f VAF=100 BF=400)
.MODEL BC560B PNP (IS=150f  VAF=100 BF=400)

.control
***TRANsient analysis: TRAN Schrittweite Stopzeit Startzeit ***
TRAN    20us 6ms 3ms
PLOT    v(Cdf1), v(Cdf2), v(A_D1), v(out)
PLOT    v(Bdf1,Edf1), v(BaseDf2,Edf1)
*******************************
.endc
.end
```

Listing 7: Si Dioden (DUS) Kennlinie

```
dio01.cir
```

```
****************************************
* Simulations-Programm fuer "SPICE"-
* Nachfolger "ngspice"
* run: "ngspice dio01.cir"
*
* (es gip etliche el.Simulatoren, die
*  de Programme fressen, zB.
*  pSPICE, winSPICE, LTspice ... )
****************************************

****************************************
* Diode "D1" mit Widerstand "R1"
* der Knoten "0" ist immer GND (Ground)
****************************************

* es eigentliche Programm,
* die "Netzliste:
**************************
Vcc     vb 0            DC 0.0Volt
Vx      vb Anode        DC 0.0Volt
D1      Anode Kathode   DUS
R1      Kathode 0       1.0milliOhm

****************************************
* "Batch"-Script: Simulations-Kommandos:
****************************************
.control
 DC Vcc 0.55 0.75 10m
*       ------------- .... X-Werte 0.55 bis 0.75V in 10milli-Schritten
 plot i(vx)
*       +------------ .... X/Y-Diagramm malen
.endc

****************************************************************
* Beschreibung des Bauteils "DUS"=Universal_Diode_aus_Silizium:
****************************************************************
.model DUS D BV=150Volt

.end
```

Listing 8: NPN-Steuerkennlinie (”620mV/1mA”)

```
*npn2.cir NPN-Steuerkennlinie ("620mV/1mA")
*****************************************
vCE     Vcc     0       dc 0
vi      Vcc     C       dc 0
vBE     VB1     0       dc 0
vbm     VB1     B       DC 0
q1      C  B  0         BC546

.model BC546    NPN  ( is=50f  bf=250  vaf=100  rb=50  rc=5 )
.control
DC      vBE     400m 700m 5m    vCE 4 6 1000m
PLOT    i(vi)
DC      vBE     500m 580m 1m    vCE 4 6 1000m
PLOT    i(vi)
DC      vBE     560m 640m 1m    vCE 4 6 1000m
PLOT    i(vi)
DC      vBE     610m 630m 0.1m  vCE 4 6 1000m
PLOT    i(vi)
.endc
.end
```

Listing 9: BC546 Ausgangskennlinienfeld

```
npn1.cir
ib      0 2     dc 0
vce     3 0     dc 0
vi      3 1     dc 0
```

```
q1      1 2 0   bc546
.model  bc546   npn ( is=50f bf=250 vaf=100 rb=50 rc=5 )
.control
dc      ib 0 10uA 0.1u  vce 1 9 2
plot    i(vi)
dc      vce 0 3 50m     ib 1u 4u 1u
plot    i(vi)
.endc
.end
```

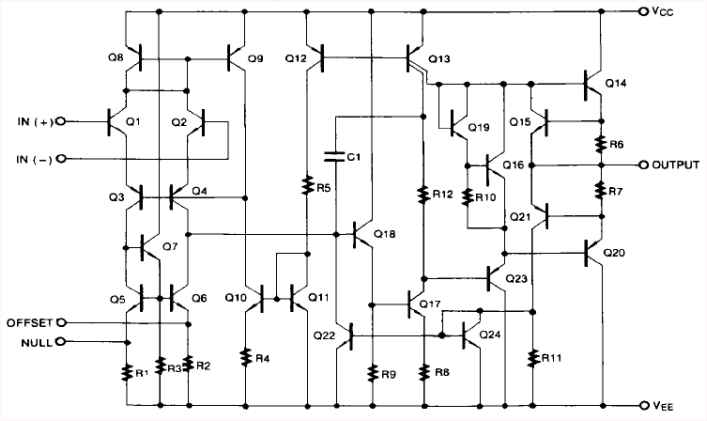
Listing 10: 2N7000 Kennlinie

```
*M2N7000.MdR.cir
Vin     G1      0       DC 0
Vplus   Vcc     0       DC 9V
RD1     Vcc     D1      1m
M1      D1 G1   S1 S1   M2N7000 L=2u W=0.12
RS1     S1      VEE1    1m
VEE1    VEE1    0       DC 0
.control
DC      Vplus 0 6 0.05  Vin 1.6 2.5 0.1
PLOT    i(VEE1)
.endc
.model M2N7000 NMOS ( Vto=1.73 Rd=0.55 Kp=1.1u Cgso=73p )
.model Mod2N7000  NMOS(Level=3 Gamma=0 Delta=0 Eta=0 Theta=0 Kappa=0.2 Vmax=0 Xj=0
+               Tox=2u Uo=600 Phi=.6 Kp=1.073u W=.12 L=2u Rs=20m Vto=1.73
+               Rd=.5489 Rds=48MEG Cgso=73.61p Cgdo=6.487p Cbd=74.46p Mj=.5
+               Pb=.8 Fc=.5 Rg=546.2 Is=10f N=1 Rb=1m)
.end
```

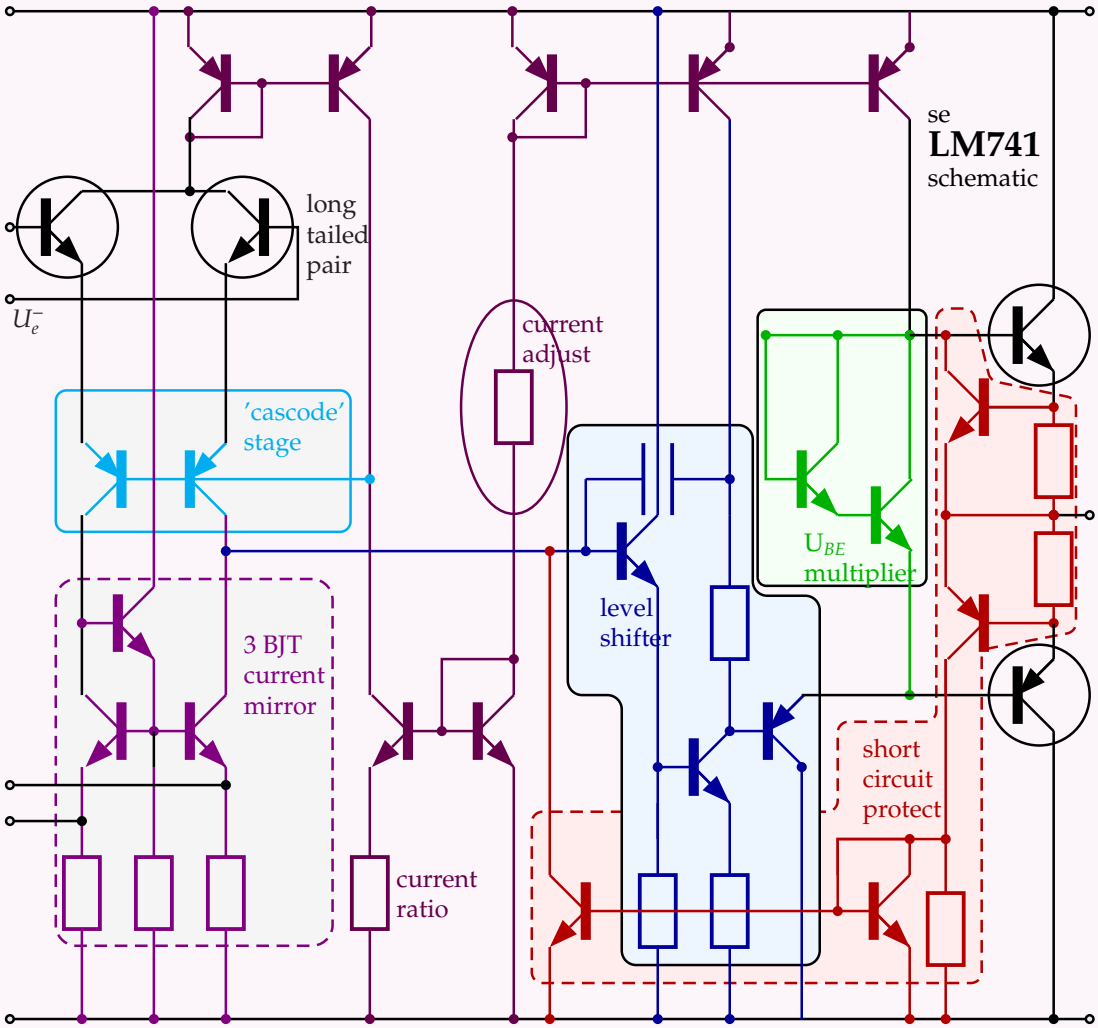
Listing 11: BF245a contra 2N5484

```
* BF245a Idss, Vp
*
Vcc     plus    0       DC 0

ViD1    plus    D1      DC 0
J1      D1 G1 S1        BF245A
Rs1     G1      S1      1m
RL1     G1      0       1m

ViD2    plus    D2      DC 0
J2      D2 G2 S2        j2N5484
Rs2     G2      S2      1m
RL2     G2      0       1m

.control
foreach x 1m 100 330 1k
 alter Rs1=$x
 alter Rs2=$x
 DC     vcc 0 10 0.05
end
plot    dc1.i(ViD1),    dc2.i(ViD1),    dc3.i(ViD1),    dc4.i(ViD1)
plot    dc1.i(ViD2),    dc2.i(ViD2),    dc3.i(ViD2),    dc4.i(ViD2)
.endc
.Model  BC550   NPN( is=10f vaf=100 bf=250)
.Model  DUS     D
.model  BF245x  njf(vto=-3 beta=0.11m lambda=0.01)
.model  j2N5484 njf(vto=-2 beta=0.99m lambda=0.01)
.model  j2N5485 njf(vto=-3 beta=0.45m lambda=0.01)
.MODEL BF245A NJF (VTO=-1.7372E+000 LAMBDA=1.77211E-002 BETA=1.16621E-003 IS=2.91797E
    -016
+               RD=9.01678E+000 RS=9.01678E+000 CGS=2.20000E-012 CGD=2.20000E-012 PB
    =7.80988E-001 FC=5.00000E-001
.model j2N5460  NJF(Beta=500u Betatce=-.5 Rd=1 Rs=1 Lambda=10m Vto=-3
+               Vtotc=-2.5m Cgd=2.5p M=.3333 Pb=1 Fc=.5 Cgs=4p Is=90p
+               Isr=800p N=1 Nr=2 Xti=3)
.end
```

LM741 Innenschaltung



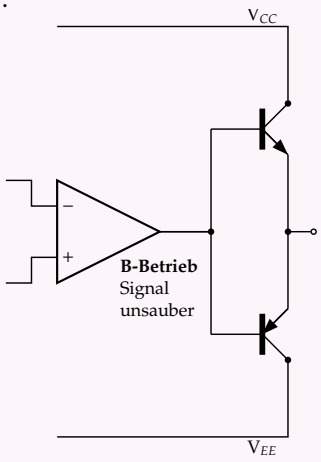
bipolar LM741 OP circuit schematic



1 .5.4 Audioverstärker mit OP

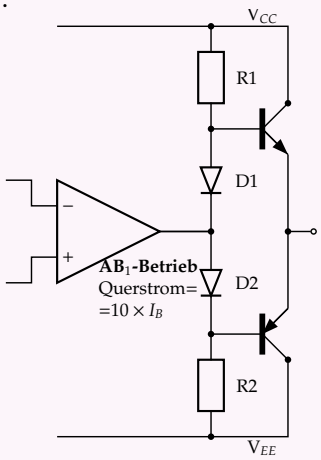
B-Betrieb (class'B' mode)

:



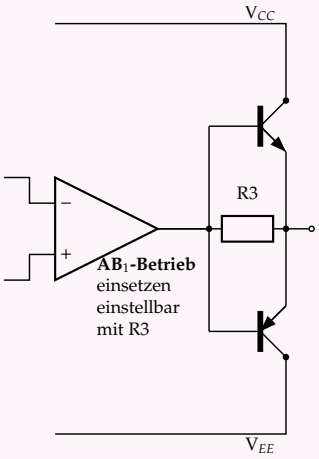
mit 2 Dioden Vorspannung, AB_1 Betrieb

:

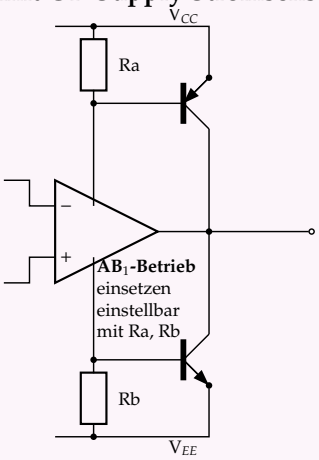


mit OP-Ausgangswiderstand

:

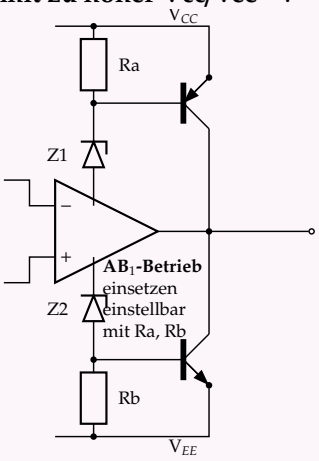


mit OP-Supply Stromsensor-Widerstand :



nur mit single OPs (keine dual- oder quad- Packages wie LM358, LM324 udgl. !)

mit zu hoher Vcc/Vee :



nur mit single OPs (keine dual- oder quad- Packages wie LM358, LM324 udgl. !)

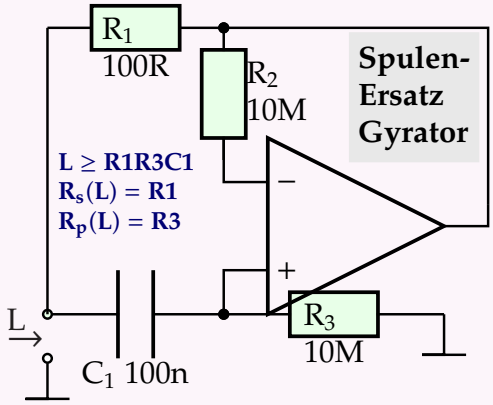
1 .6 OP

1 .6.1 NIC, INIC

s. zB 'Tietze-Schenk'

1 .6.2 Gyrator

s. zB 'Tietze-Schenk'

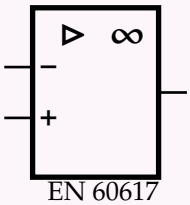


(bei 10M ist der
Ruhestrom-Spannungsabfall an R3
unbedingt mit R2 auszugleichen)

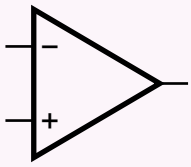
1 .7 OPamp mit LaurenzPREINDL

1 .7.1 Grundlagen

Schaltzeichen

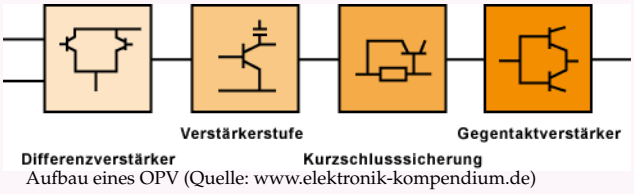


EN 60617



DIN 40900-10

Aufbau (Stufen)



Aufbau eines OPV (Quelle: www.elektronik-kompendium.de)

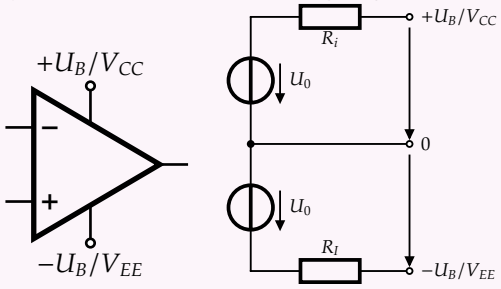
Eigenschaften

- "P+" Eingang (U_e^+)
- "N-" Eingang (U_e^-) (invertierend)
- U(t) (bzw. I(t))
  - addieren+subtrahieren,
  - multiplizieren (=verstärken),
  - differenzieren+integrieren,
  - Analogrechnen
  - (de-)symmetrieren
  - umschalten
- Signale erzeugen + formen

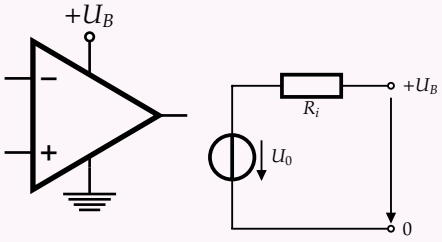
Vergleich der Eigenschaften von idealen und realen OPamps

	ideal	**real**
Eingangswiderstand	∞	$\epsilon[10^6 \ldots 10^{12}\Omega]$
Ausgangswiderstand	0	einige 100 Ω
Leerlauf-Verstärkung 'A$_0$' $U_A = A_0 * (U_{P+} - U_{N-})$ $\ldots A_0\ \epsilon\ [1000 \ldots 10^7]$ (typ. 25000)	∞	60..140dB (10^4..10^7)
Gleichtaktunterdrückung (CMRR common mode rejection ratio) (Gleichtaktverstärkung V_{CM}, A_{CM})	∞ 0	~ -60..=120dB sehr klein ~ 10^{-4}
Stromversorgungsunterdrückung (PSRR power supply rejection ratio)	∞	~ -100dB
Frequenzbandbreite -3dB	∞	0 typ. 10 Hz
gain-bandwidth-product GBP, Transitfrequenz f_T	∞	1MHz(standart) .. 1GHz (special)
Eingangsruhestrom I_q	0	pA..100μA

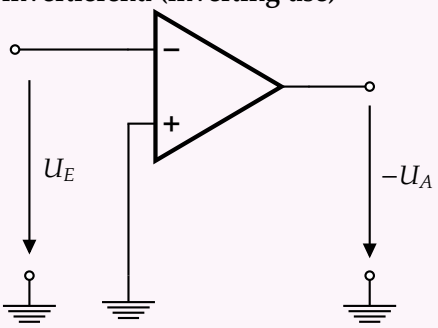
Symmetrische Stromversorgung



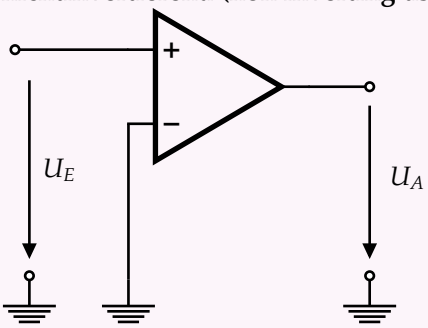
Asymmetrische Stromversorgung



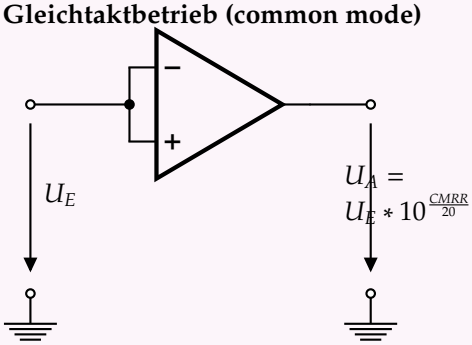
invertierend (inverting use)



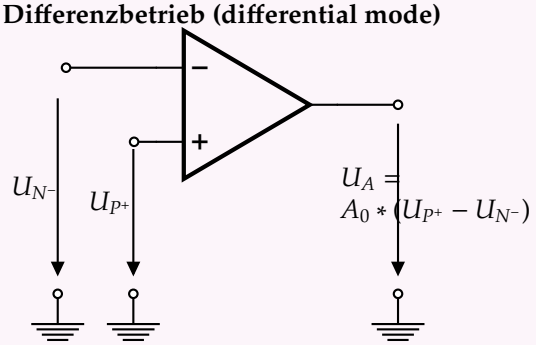
nichtinvertierend (non inverting use)



Gleichtaktbetrieb (common mode)



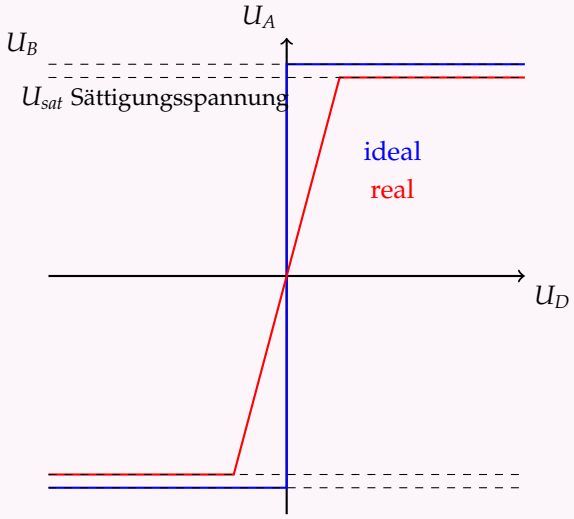
Differenzbetrieb (differential mode)



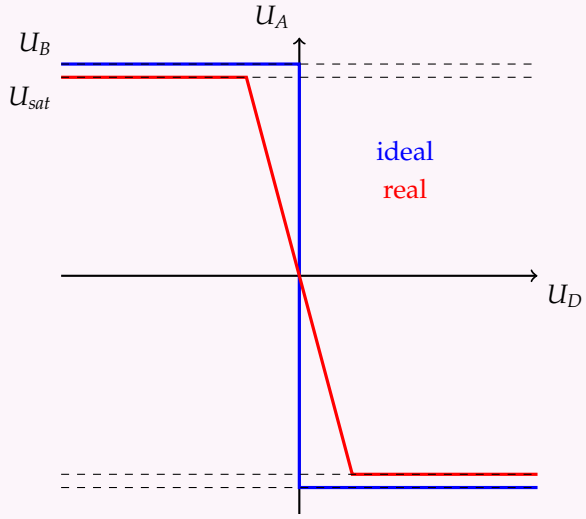
idealer OP: U_D=0V, realer OP: μV-Bereich.

Kennlinien

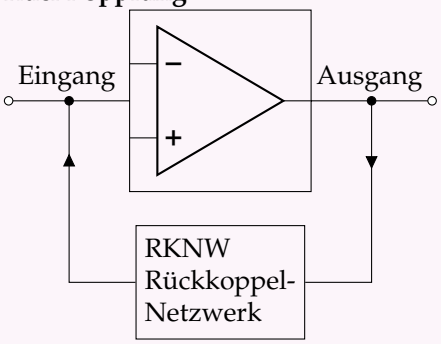
nichtinvertierend



invertierend

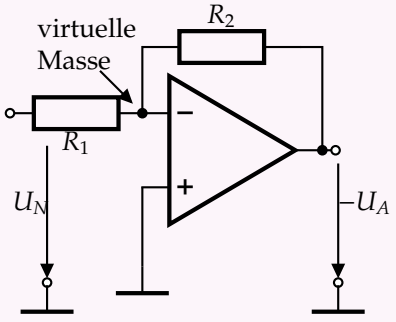


Rückkopplung

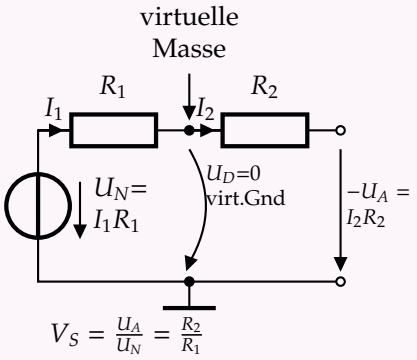


Anwendungen

invertierender Betrieb

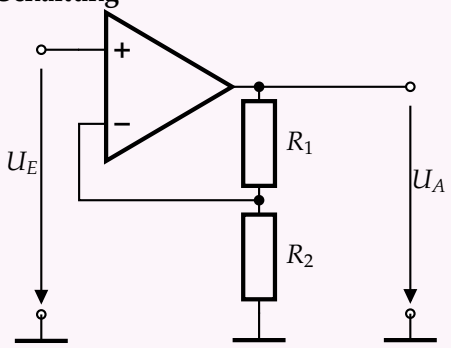


Ersatzschaltung



nichtinvertierender Betrieb

Schaltung



$$M_1 = U_1 + U_2 - U_A = 0$$
$$M_2 = -U_E + U_D + U_1 = 0$$

$$U_A = U_1 + U_2$$
$$U_D = 0$$
$$U_E = U_1$$

$$V_S = \frac{U_A}{U_E}$$
$$V_S = \frac{U_1 + U_2}{U_E}$$
$$V_S = \frac{R_1 + R_2}{R_1}$$
$$V_S = 1 + \frac{R_2}{R_1}$$

Beispiele

Beispiel 1

geg.: $R_2 = 10\text{k}\Omega$
$U_A = 3 \cdot U_E$

ges.: R_1, V
a.) invertierender OPV
b.) nichtinvertierender OPV

a.) invertierender OPV

$$V = \frac{U_A}{U_E}$$
$$V = \frac{3 \cdot U_E}{U_E} = 3$$
$$V = \frac{R_2}{R_1} = \frac{10k\Omega}{R_1}$$
$$R_1 = \frac{10k\Omega}{3} = 3,\dot{3}k\Omega$$

b.) nichtinvertierender OPV

$$V = \frac{U_A}{U_E}$$
$$V = \frac{3 \cdot U_E}{U_E} = 3$$

$$V = \frac{R_2}{R_1} + 1$$
$$3 = \frac{10k\Omega}{R_1} + 1$$
$$2 = \frac{10k\Omega}{R_1}$$
$$R_1 = \frac{10k\Omega}{2}$$
$$R_1 = 5k\Omega$$

Beispiel 2

geg.: $I_2 = 10\text{mA}$
$R_1 = 15\text{k}\Omega$
$U_E = \frac{1}{4}U_A$

ges.: V, R_2, U_A, I_1, U_E
a.) invertierender OPV
b.) nichtinvertierender OPV

a.) invertierender OPV

$$V = \frac{U_A}{U_E}$$
$$V = \frac{U_A}{\frac{1}{4}U_A} = 4$$

$$V = \frac{R_2}{R_1}$$
$$4 = \frac{R_2}{R_1}$$
$$R_2 = 4 \cdot 15k\Omega = 60k\Omega$$

$$U_A = U_2$$
$$U_A = I_2 \cdot R_2$$
$$U_A = 10mA \cdot 60k\Omega$$
$$U_A = 600V$$

$$I_1 = I_2 = 10mA$$

$$U_E = \frac{1}{4}U_A$$
$$U_E = \frac{600V}{4} = 150V$$

b.) nichtinvertierender OPV

$$V = \frac{U_A}{U_E}$$

$$V = \frac{U_A}{\frac{1}{4}U_A} = 4$$

$$V = \frac{R_2}{R_1} + 1$$

$$3 = \frac{R_2}{R_1}$$

$$R_2 = 3 \cdot 15k\Omega = 45k\Omega$$

$$U_A = U_1 + U_2$$

$$U_A = I_2 \cdot (R_1 + R_2)$$

$$U_A = 10mA \cdot 60k\Omega$$

$$U_A = 600V$$

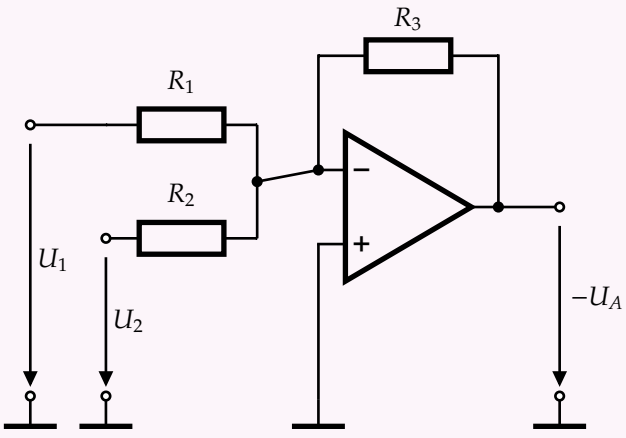
$$I_1 = I_2 = 10mA$$

$$U_E = \frac{1}{4}U_A$$

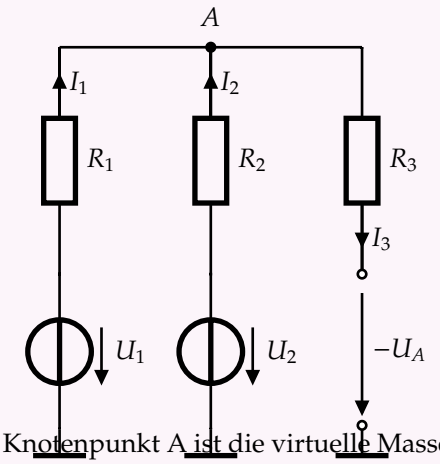
$$U_E = \frac{600V}{4} = 150V$$

Addierer

Schaltung



Ersatzschaltung



Knotenpunkt A ist die virtuelle Masse.

$$A : I_1 + I_2 = I_3$$
$$-U_A = U_{R_3}$$
$$I_3 = \frac{U_1}{R_1} + \frac{U_2}{R_2}$$

$$U_A = U_1 \frac{R_3}{R_1} + U_2 \frac{R_3}{R_2}$$

Beispiel 3

geg.: $R_1 = 1\text{k}\Omega$, $R_2 = 5\text{k}\Omega$, $R_3 = 15\text{k}\Omega$, $U_1 = 2\text{V}$, $U_2 = 8\text{V}$

ges.: U_A für Addierer

$$U_A = U_1 \frac{R_3}{R_1} + U_2 \frac{R_3}{R_2}$$
$$U_A = 2V \cdot \frac{15k\Omega}{1k\Omega} + 8V \cdot \frac{15k\Omega}{5k\Omega}$$
$$U_A = 69V$$

Beispiel 4 Es soll ein Addierer konstruiert werden, bei dem U_1 $8x$, U_2 $4x$ und $-U_A$ $5x$ vergrößert werden soll.

$$U_A = 5 \cdot (8 \cdot U_1 + 4 \cdot U_2)$$
$$U_A = 40 \cdot U_1 + 20 \cdot U_2$$
$$V_1 = 40 = \frac{R_3}{R_1}$$
$$V_2 = 20 = \frac{R_3}{R_2}$$

Ein Widerstand kann ausgewählt werden, da nur das Verhältnis zwischen den Widerständen ausschlaggebend ist.

$$R_1 = 1k\Omega$$

$$40 = \frac{R_3}{1k\Omega}$$
$$R_3 = 40k\Omega$$

$$\frac{1}{V_2} = \frac{R_2}{R_3}$$
$$R_2 = \frac{40k\Omega}{20}$$
$$R_2 = 2k\Omega$$

Beispiel 5

geg.: $R_1 = 3\text{k}\Omega$, $R_2 = 5\text{k}\Omega$, $I_3 = 3\text{mA}$, $U_1 = 2\text{V}$, $U_A = 20\text{V}$

ges.: R_3, U_2 für Addierer

$$R_3 = \frac{U_A}{I_3}$$
$$R_3 = \frac{20V}{3mA}$$
$$R_3 = 6,\dot{6}k\Omega$$

$$U_A = U_1 \frac{R_3}{R_1} + U_2 \frac{R_3}{R_2}$$

$$U_2 = \frac{U_A - U_1 \frac{R_3}{R_1}}{\frac{R_3}{R_2}}$$

$$U_2 = \frac{R_2 \cdot (U_A - U_1 \frac{R_3}{R_1})}{R_3}$$

$$U_2 = \frac{5k\Omega \cdot (-20V - 2V \cdot \frac{6,\dot{6}k\Omega}{3k\Omega})}{6,\dot{6}k\Omega}$$

$$U_2 = -18,\dot{3}V$$

Beispiel 6

geg.: $R_3/R_1 = 3$
$R_3/R_2 = 6$
$U_1 = 5V$
$U_2 = 8V$

ges.: U_A, R_1, R_2, R_3 für Addierer

$$U_A = 3 \cdot U_1 + 6 \cdot U_2$$
$$U_A = 63V$$

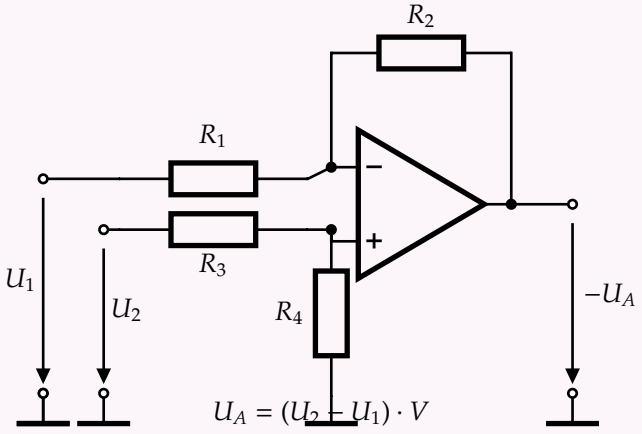
Es wird ein Wert für R_3 angenommen.

$$R_3 = 5k\Omega$$

$$R_1 = \frac{15k\Omega}{3}$$
$$R_1 = 5k\Omega$$

$$R_2 = \frac{15k\Omega}{6}$$
$$R_2 = 2,5k\Omega$$

Subtrahierer
Schaltung



Ruhestromkompensation

$$\frac{R_1}{R_2} = \frac{R_3}{R_4}$$

Beispiel 7

geg.: $V = 5$
$R_3 = 3,6k\Omega$
$U_1 = 12V$
$U_2 = 22V$
$R_1 \neq R_3$

ges.: U_A, R_1, R_2, R_4
für Subtrahierer

$$V = \frac{R_4}{R_3}$$
$$R_4 = V \cdot R_3$$
$$R_4 = 18k\Omega$$

Es wird ein Wert für R_1 angenommen.

$$R_1 = 2k\Omega$$
$$R_2 = V \cdot R_1$$
$$R_2 = 10k\Omega$$

$$U_A = (U_2 - U_1) \cdot V$$
$$U_A = (22V - 12V) \cdot 5$$
$$U_A = 50V$$

Beispiel 8

geg.:

R_1/R_2	= 0,5
R_4	$= 3 \cdot R_2$
U_{R4}	= 8V
U_1	= 6V
I_1	= 15mA

ges.: V, U_2, U_A, R_1, R_2, R_3, R_4
für Subtrahierer

$$R_1 = \frac{U_1}{I_1}$$
$$R_1 = \frac{6V}{15mA}$$
$$R_1 = 400\Omega$$

$$R_2 = 2 \cdot R_1$$
$$R_2 = 2 \cdot 400\Omega$$
$$R_2 = 800\Omega$$

$$R_4 = 3 \cdot R_2$$
$$R_4 = 3 \cdot 800\Omega$$
$$R_4 = 2,4k\Omega$$

$$R_3 = \frac{R_4}{2}$$
$$R_3 = \frac{2,4k\Omega}{2}$$
$$R_3 = 1,2k\Omega$$

$$U_2 = U_{R_3} + U_{R_4}$$
$$U_2 = U_4 \cdot \frac{R_3 + R_4}{R_3}$$
$$U_2 = 8V \cdot \frac{3,6k\Omega}{1,2k\Omega}$$
$$U_2 = 12V$$

$$V = \frac{R_2}{R_1}$$
$$V = 2$$

$$U_A = V \cdot (U_2 - U_1)$$
$$U_A = 2 \cdot (12V - 6V)$$
$$U_A = 12V$$

Beispiel 9

geg.: $R_4 = 2 \cdot R_3$, $I_3 = 36\text{mA}$, $U_2 = 3\text{V}$, $U_A = 5 \cdot U_1$

ges.: V, U_1, U_A, R_1, R_2, R_3 für Subtrahierer

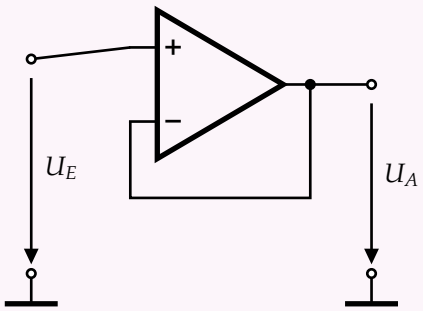
$$V = 2$$
$$R_3 = \frac{R_4}{2}$$
$$R_3 + R_4 = \frac{U_2}{I_3}$$
$$3 \cdot R_4 = 2 \cdot \frac{U_2}{I_3}$$
$$R_4 = 55,\dot{5}k\Omega$$
$$R_3 = 27,\dot{7}k\Omega$$

Der Wert R_1 wird gleich R_3 angenommen.

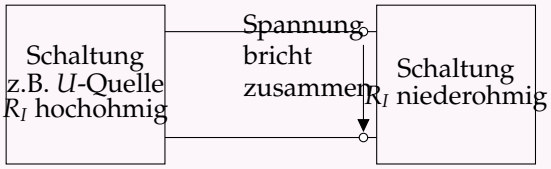
$$R_1 = 27,\dot{7}k\Omega$$
$$R_3 = 55,\dot{5}k\Omega$$

$$U_A = 5 \cdot U_1$$
$$5 \cdot U_1 = 2 \cdot (3V - U_1)$$
$$U_1 = \frac{6}{7}V$$
$$U_A = \frac{30}{7}V$$

Impedanzwandler, Spannungsfolger Schaltung



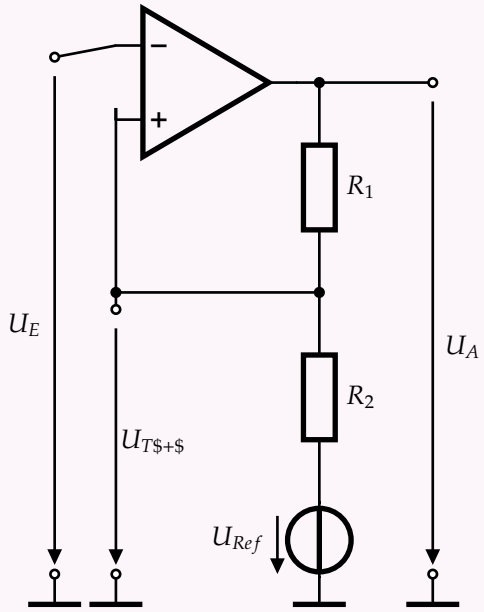
Anwendung



Wenn die Spannung zusammenbricht, kann man einen Impedanzwandler als Hilfe verwenden.
Eine Verstärkung von 1 ist dann sinnvoll, wenn einseitig eine Spannungsquelle mit hohem Innenwiderstand verwendet wird.
Hier wird die Eigenschaft genutzt, dass ein idealer OPV einen unendlich großen Eingangswiderstand hat und einen Ausgangswiderstand von 0Ω, daher spricht man bei dieser Schaltung von einem Impedanzwandler. Eine solche Schaltung kann durch eine relativ hochohmige Spannungsquelle eine niederohmige und durch Folgeschaltungen belastbare Spannungsquelle machen.

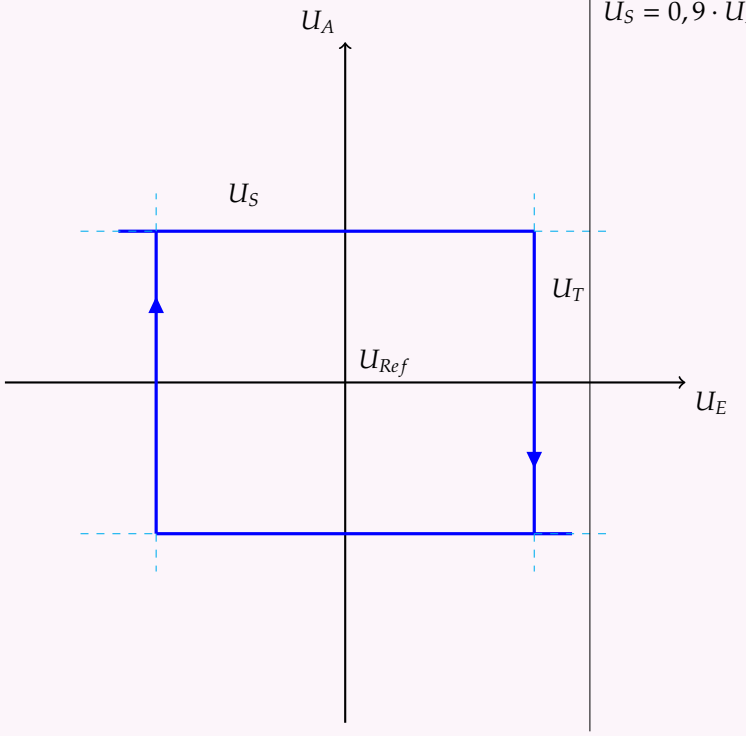
Komparatorschaltungen Ein OPV kann zwei Eingangsgrößen vergleichen. Komparatorschaltungen werden allgemein ohne Gegenkopplung betrieben. Bei direkter Ansteuerung mit geringfügig unterschiedlichen Eingangssignalen wechselt das Ausgangssignal nur zwischen den beiden Ansteuerungsgrenzen ($+U_S$ und $-U_S$ bzw. $+U_B$ und $-U_B$).
Wegen der hohen Verstärkung und der schnellen Arbeitsweise der Komparatoren entsteht bei annähernd gleichen Eingangspannungen ein undefiniertes Hin- und Herschalten der Ausgangsspannung (durch Rauschanteil). Am Ausgang zeigen sich sogenannte "wilde Schwingungen" im Hochfrequenzbereich. Abhilfe wird durch **Mitkopplung** erreicht, wo mittles Spannungsteiler ein Teil der Ausgangsspannung an dem (+)-Eingang des OPV zurückgeführt wird. Die Referenzspannung, mit der die Eingangsspannung verglichen werden soll, liegt entweder am nichtinvertierenden oder am invertierenden Eingang. Typische Kennzeichen der Komparatorschaltung ist die **Mitkopplung**.

Invertierender Komparator (invertierender Schmitt-Trigger)



Hysterese

Hysteresespannung $U_H = U_{T\$+\$} - U_{T\$-\$}$
Wenn $U_{Ref} = 0$V, ist die Hysterese Nullpunktsymetrisch.
$U_S = 0,9 \cdot U_B$



Beispiel 10 Invertierte Ausgangsspannung eines Schmitt-Triggers.
$U_{T\$+\$}$, $U_{T\$-\$}$

$$U_{T+} = U_{R_2} + U_{Ref}$$
$$U_{R_2} = \frac{R_2}{R_1 + R_2} \cdot (U_S - U_{Ref})$$

$$U_{T+} = \frac{R_2}{R_1 + R_2} \cdot (U_{S+} - U_{Ref}) + U_{Ref}$$
$$U_{T-} = \frac{R_2}{R_1 + R_2} \cdot (U_{S-} - U_{Ref}) - U_{Ref}$$

$$U_H = U_{T+} - U_{T-}$$
$$U_H = \frac{R_2}{R_1 + R_2} \cdot (U_{S+} - U_{S-})$$

Beispiel 11

geg.: $U_{Ref} = 2V$
$R_1 = 5k\Omega$
$R_2 = 45k\Omega$
$U_B = \pm 20V$

ges.: U_{T+}, U_{T-}, U_H
für invertierenden Schmitt-Trigger

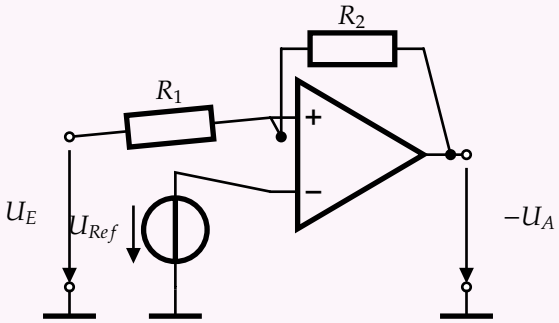
$$U_S = 0,9 \cdot U_B = 18V$$

$$U_{T+} = \frac{R_2}{R_1 + R_2} \cdot (U_{S+} - U_{Ref}) + U_{Ref}$$
$$U_{T+} = \frac{45k\Omega}{5k\Omega + 45k\Omega} \cdot (18V - 2V) + 2V$$
$$U_{T+} = 16,4V$$

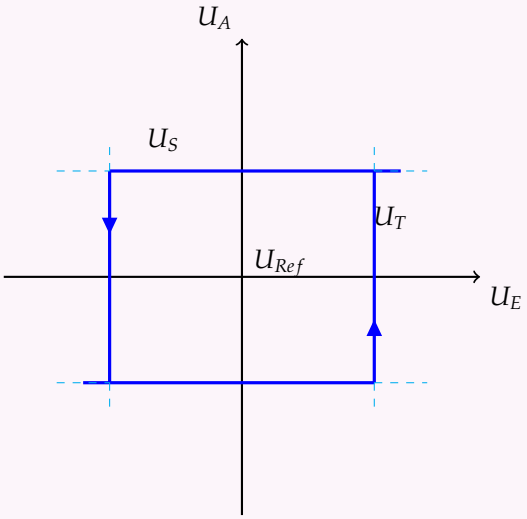
$$U_{T-} = \frac{R_2}{R_1 + R_2} \cdot (U_{S-} - U_{Ref}) + U_{Ref}$$
$$U_{T-} = \frac{45k\Omega}{5k\Omega + 45k\Omega} \cdot (-18V - 2V) + 2V$$
$$U_{T-} = -16V$$

$$U_H = U_{T+} - U_{T-}$$
$$U_H = 16,4V + 16V$$
$$U_H = 32,4V$$

Nichtinvertierender Komparator (Schmitt-Trigger)



Hysterese



$$U_H = U_{T+} - U_{T-}$$
$$U_H = \frac{R_1}{R_2}(U_{S+} - U_{S-})$$

$$U_{T+} = U_{Ref} - U_{R_1} = U_{Ref} - \frac{R_1}{R_2}(U_{S-} - U_{Ref})$$
$$U_{T-} = U_{Ref} - U_{R_1} = U_{Ref} - \frac{R_1}{R_2}(U_{S+} - U_{Ref})$$

Beispiel 12

geg.: $U_B = 12V$
$R_1 = 10k\Omega$
$R_2 = 4,7k\Omega$

ges.: U_{T+}, U_{T-}, U_H
für nichtinvertierenden Schmitt-Trigger
a.) $U_{Ref} = 0V$
b.) $U_{Ref} = ?$

a.) $U_{Ref} = 0V$

$$U_H = 0,9 \cdot U_B$$
$$U_H = 0,9 \cdot 12V$$
$$U_H = 10,8V$$

$$U_H = \frac{R_1}{R_2}(U_{S+} - U_{S-})$$
$$U_H = \frac{10k\Omega}{4,7k\Omega}(-10,8V - 10,8V)$$
$$U_H = 45,95V$$
$$U_H = 45,95V$$

Beispiel 13

geg.: $U_{Ref} = 2,5V$
$U_A = 12V$
$R_1 = 2 \cdot R_2$

ges.: U_{T+}, U_{T-}, U_H
für nichtinvertierenden Schmitt-Trigger

$$U_{S+} = U_A$$

$$U_{T-} = U_{Ref} - \frac{R_1}{R_2}(U_{S+} - U_{Ref})$$
$$U_{T-} = 2,5V - 2(12V - 2,5V)$$
$$U_{T-} = -16,5V$$

$$U_{T+} = U_{Ref} - \frac{R_1}{R_2}(U_{S-} - U_{Ref})$$
$$U_{T+} = 0V - \frac{10k\Omega}{4,7k\Omega}(-10,8V - 0V)$$
$$U_{T+} = 22,98V$$

$$U_{T-} = U_{Ref} - \frac{R_1}{R_2}(U_{S+} - U_{Ref})$$
$$U_{T-} = 0V - \frac{10k\Omega}{4,7k\Omega}(10,8V - 0V)$$
$$U_{T-} = -22,98V$$

b.) $U_{Ref} = 0$?

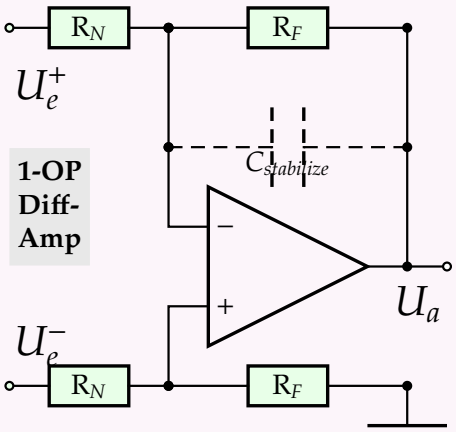
$$U_{Ref} = U_{T-} + \frac{U_H}{2}$$
$$U_{Ref} = U_{T+} - \frac{U_H}{2}$$

$$U_{Ref} = U_{T-} + \frac{U_H}{2}$$
$$U_H = 2 \cdot (U_{Ref} - U_{T-})$$
$$U_H = 2 \cdot (2,5V + 16,5V)$$
$$U_H = 38V$$

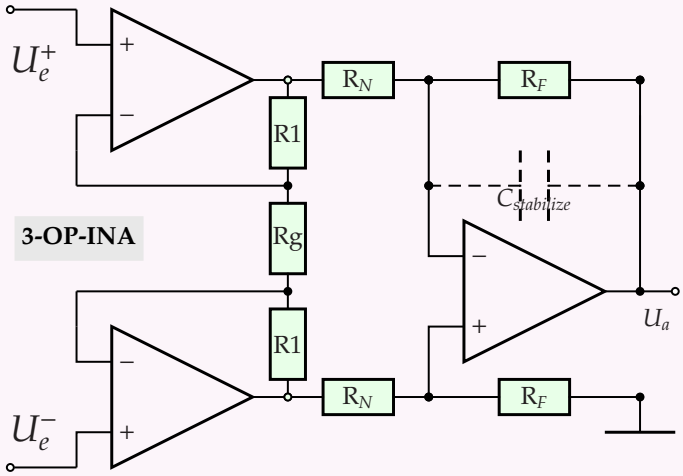
$$U_{T+} = U_H + U_{T-}$$
$$U_{T+} = 38V - 16,5V$$
$$U_{T+} = 21,5V$$

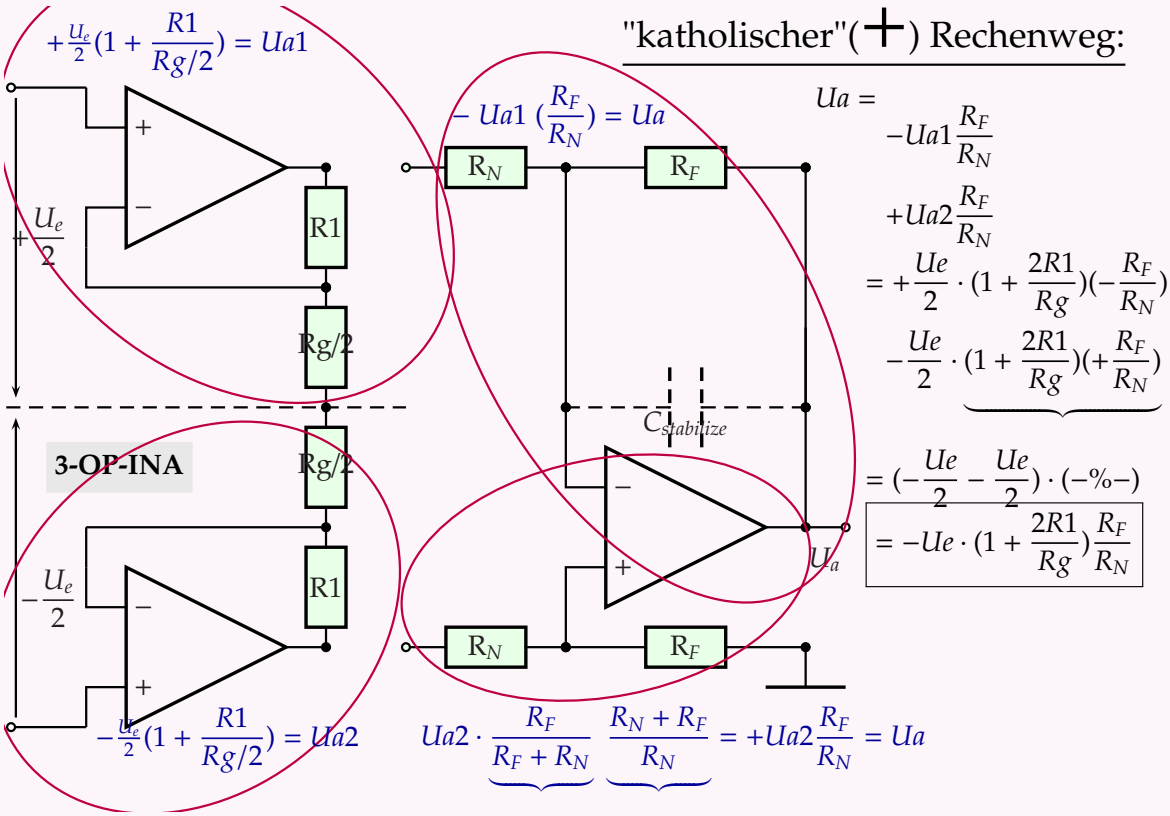
1 .8 INA Instrumentation Amplifier

1 .8.1 Difference Amp



1 .8.2 3-OP-INA

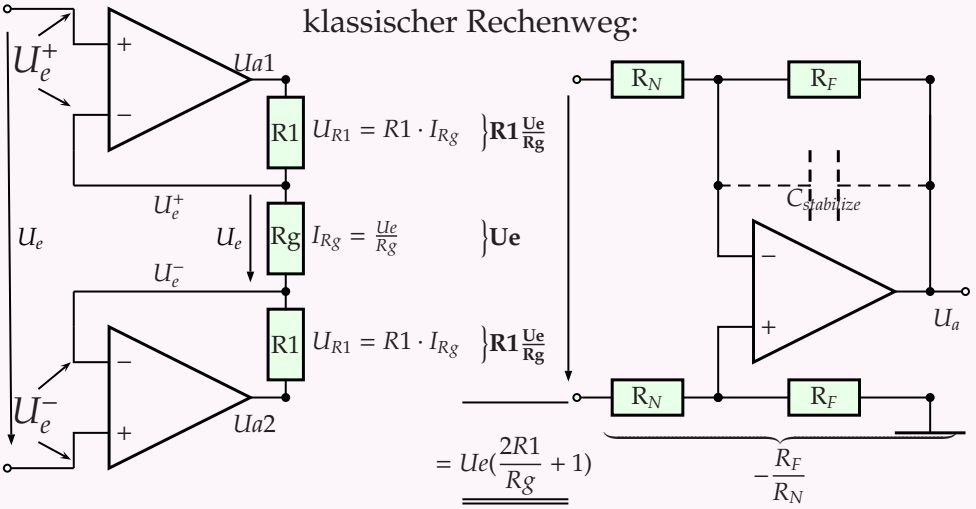




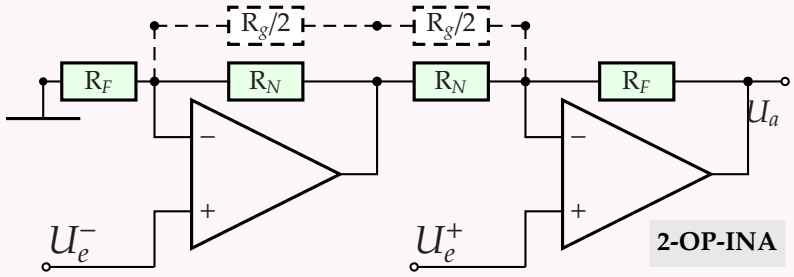
der Rechenweg beruht auf dem *'Superpositionsprinzip'* linearer Funktionen:

$$f(a+b) = f(a) + f(b)$$
$$f(a*x) = a*f(x)$$

klassischer Rechenweg:



1 .8.3 2-OP-INA



$$Ue^- \longrightarrow (1+\frac{R_N}{R_F}) \longrightarrow (-\frac{R_F}{R_N})$$
$$Ue^+ \longrightarrow (1+\frac{R_F}{R_N})$$

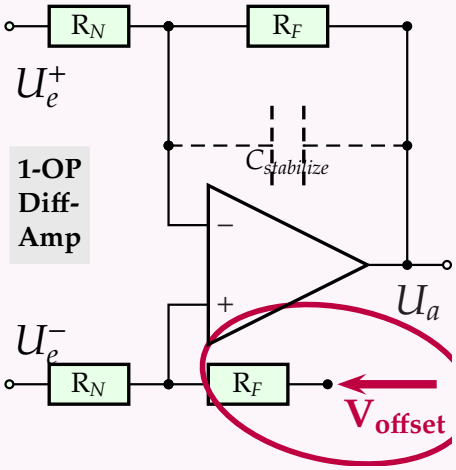
$$Ue^- \longrightarrow (\frac{R_F+R_N}{R_F}) \longrightarrow (-\frac{R_F}{R_N}) \searrow$$
$$Ue^+ \longrightarrow (\frac{R_F+R_N}{R_N}) \nearrow \quad \text{Ua}$$

$$\begin{aligned} Ua &= Ue^- \cdot \frac{R_F+R_N}{R_F} \cdot (-\frac{R_F}{R_N}) \\ &+ Ue^+ \cdot \frac{R_F+R_N}{R_N} \\ &= \frac{R_F+R_N}{R_N}(Ue^+ - Ue^-) \end{aligned}$$

$$Ua = (Ue^+ - Ue^-) \cdot (1+\frac{R_F}{R_N})$$

1 .9 single supply/single sided OP /RcT

1 .9.1 Difference Amp



Der Schwenk zur 'PC'-Computerisierung bringt eine dramatische Verarmung in der OP-Stromversorgungsszene mit sich. War da noch vor Kurzem ±15V OP- Standard, ist aktuell die Verwendbarkeit an

[+5V/0V] (≡ *'single Supply'*)

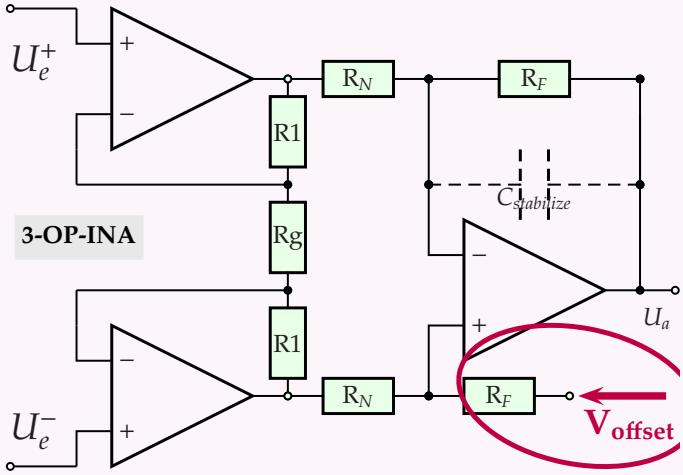
oft wichtiger, als dessen übrige Qualitätsmerkmale. Zu beachten sind bitte die genauen Spezifikationen ('Specs'), da die Titel

'single supply' und
'rail-to-rail'

oft nur vieldeutige Werbesprüche sind. So kommen die meisten 'rail-to-rail' OPs bei Ua nur mit $R_L > 10k$ in **die Nähe** der 'Rails' oder bei Ue nur nach GND, aber zu Vcc gar nicht udergl.

Massebezogene (*'ground referenced'*) Analogsignale schwingen gewöhnlich auch nach *'unten' ins Negative*, sodass ein OP sie ohne *'Minus-Versorgung'* nicht verarbeiten kann.

1 .9.2 3-OP-INA



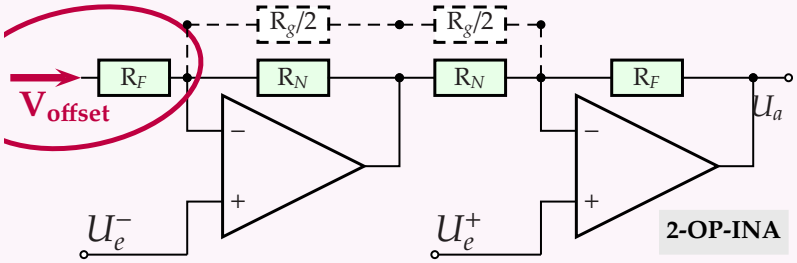
Man studiere besonders die Angaben zu
- *'input common mode range'*
- *'output voltage swing'*
- *'power supply voltage range'*
- *'output current'* (I_{source}, I_{sink})

(vgl. zB. LM6132(NS), LM6142(TI) data sheets)

Als Lösung *'shiftet'* man den Massebezug (*die 'ground reference'*) der OP-Schaltung mittels einer *'Offset-Spannung'* mehr in die Mitte des verwendbaren Eingangsspannungsbereichs. Auch die Inputsignale sind zu *shiften*, falls sie nicht in den Eingangs-Aussteuerungsbereich des OP fallen (da helfen die Ausgangs-Offsets hier nix).

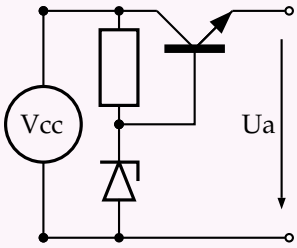
Die dazu üblichen Referenzpunkte bei Differenzverstärkern sind anbei gekennzeichnet.

1 .9.3 2-OP-INA



1 .10 Linearregler

Einfache Linearstabilisierung:

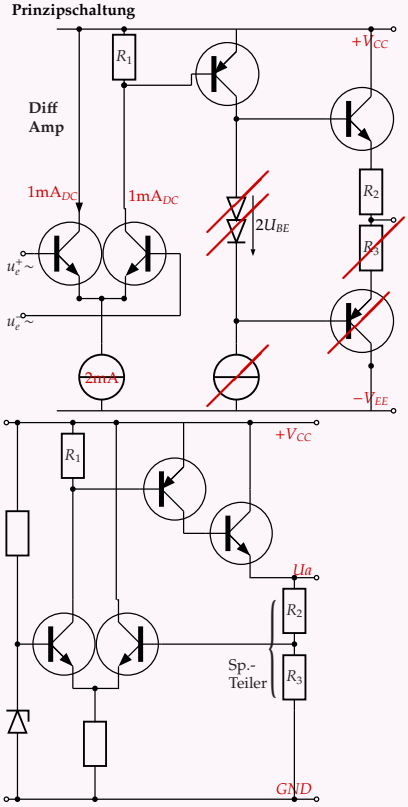


Übungsaufgabe: Dimensioniere Bauteile und Kühlkörper für Ue=10V, Ua=5V, Ia=1A

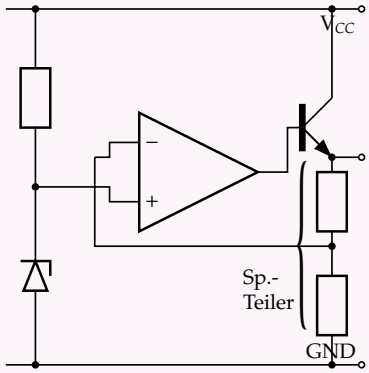
- ☐ Die Zenerspannung muss ca. 0.7V (U_{BE}) über der Emitter-Ausgangsspannung liegen, bei Darlington-Transistoren natürlich 2x U_{BE}.
- ☐ Nicht verfügbare Zenerspannungen bildet man aus Serienschaltung anderer Dioden.
- ☐ "Schutzwiderstände" am Ausgang verschlechtern die Stabilisierung entsprechend.

Für höhere Ansprüche geht es nicht ohne Regelkreis — das ähnelt dann der Audio-Endstufe unter Weglassung der 'negativen Hälfte':

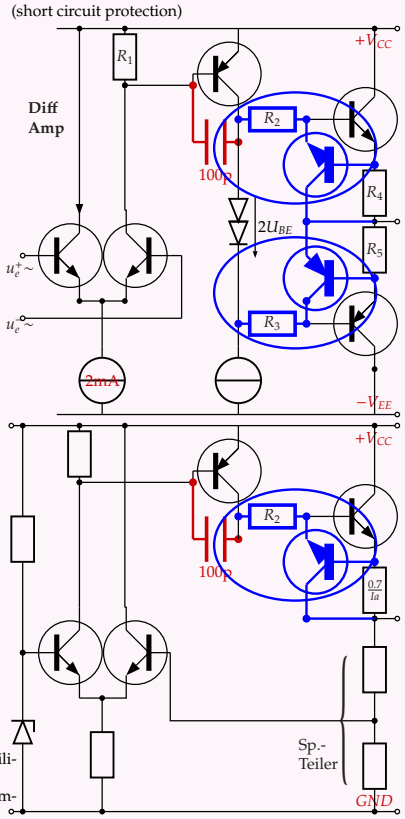
Prinzipschaltung



- o keine Temperaturstabilisierung
- o keine Frequenzgangkompensation
- o kein Kurzschlussschutz
- o kein U_{BE} Vervielfacher



Kurzschlussschutz
(short circuit protection)



- keine Temperaturstabilisierung
- mit **Frequenzgangkompensation**
- mit **Kurzschlussschutz/ Strombegrenzer**

$$I_{max} = \frac{620mV}{R4}$$

- kein U_{BE} Vervielfacher:

1 .11 Wärme-Ableitung (heat transfer)

Wärmemenge ist eine **Energie** Q [J]

↓

(bitte **nicht** mit **Temperatur** (Θ[K]) verwechseln!)

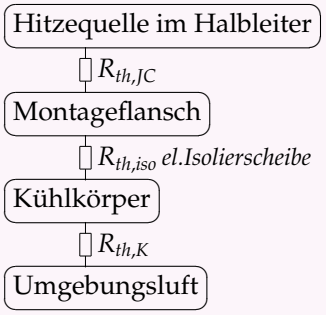
Q . . . . . Wärmeenergiemenge

Die Änderung der (Wärme-) Energiemenge pro Zeiteinheit $^{\Delta Q}/_{\Delta t}$ ist eine Leistung P [W].

Um die gut bekannten Widerstands-Rechenregeln verwenden zu können, definiert man für den Übergang von einem thermischen System (zB. p/n-Zone) zum anderen (zB. Gehäuseflansch) einen sog. **thermischen Widerstand:**

$$R_{th} = \frac{\Theta}{P}[K/W]$$

Die Vorstellung dazu ist, dass die Temperatur Θ wie eine treibende Spannung U den Wärmeenergie-Transport (vgl. Strom I) verursache.



Beispiel:
Im IRF540 habe die Wärmeableitungsstrecke vom Kanal (aktive Zone, 'junction') bis zur Oberfläche des Montageflansches ('case') einen Wärmewiderstand von

$$R_{th,JC} = 1.15\frac{K}{W},$$

eine zwischenzulegende Glimmer-Isolierscheibe

$$R_{th,I} = 0.6\frac{K}{W}$$

sowie ein Kühlkörper (heat sink)

$$R_{th,K} = 2.5\frac{K}{W}$$

Frage:
Wieviel Verlustleistung darf obiger MosFET bei 50°C Raumtemperatur 'verheizen', wenn sein Kanal maximal 150°C heiss werden darf?
→

$$R_{th,gesamt} = R_{th,JC} + R_{th,I} + R_{th,K} = 4.25\frac{K}{W} = \frac{\Delta\Theta}{P}$$

$$P = \frac{\Delta\Theta}{R_{th,gesamt}} = \frac{100K}{4.25K/W} = 23.5[W]$$

Beispiel:
Ohne Kühlkörper habe obiger IRF540 einen Wärmewiderstand zur Umgebungsluft von $R_{th,JA}$ ('junction to ambient')

$$R_{th,JA} = 62\frac{K}{W}.$$

Was kann er bei $\Theta_A = 50^oC$ 'verheizen':

$$P = \frac{\Delta\Theta}{R_{th,JA}} = \frac{100K}{62K/W} = 1.6[W]$$

→ Man solle pro Leistungstransistor

nie über ca. 20W

Wärmeverlust vorsehen!

→ Die Umgebungslufttemperatur solle man mit

mindestens 70°C

ansetzen (Kühlkörper in Sommersonne hinter einem Fenster).

→ die Halbleiter altern je 10K doppelt so schnell - die Lebensdauer (mit spezifizierter Verstärkung) ist also etwa

$$\frac{Lebensdauer_{25^oC}}{2^{\frac{\Theta-25^oC}{10}}}$$

25C	100 Y
110C	100 d
150C	6.3 d (Dauerbetrieb!)

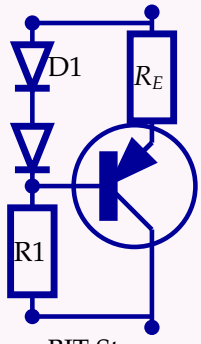
Ursache für die Alterung sei das Wandern von Aluminium und Dotieratomen in den Grenzzonen, sodass die Verstärkung (β, g_{fs}) aufgrund verwaschener Dotierprofile ständig abnimmt, bis sie schliesslich die Spezifikation unterschreitet.

Materielle Körper haben auch eine (materialabhängige) **Wärmekapazität** C_{th}, das ist die pro Kelvin K zu-/abgeführte Wärmeenergiemenge Q[J].

$$C_{th} = \frac{\Delta Q}{\Delta\Theta}$$

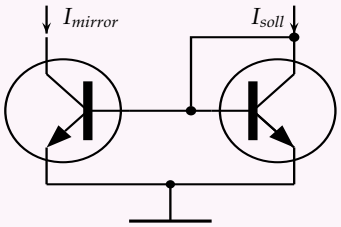
Dadurch kann ein kalter Kühlkörper kurzzeitig mehr Wärme aufnehmen als bei erreichter Betriebstemperatur ableiten und bewirkt auch, dass Halbleiter im Pulsbetrieb mehr Verlustleistung verkraften.

1 .12 Stromquellen und -Spiegel



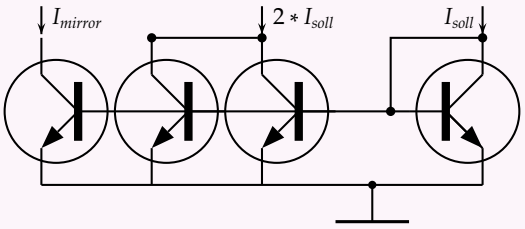
$I_{R_E} \quad \frac{1}{\alpha} \quad U_E$
$\Rightarrow$
$I_{R_E} \quad \frac{1}{\alpha} \quad U_{BE}$
$\Rightarrow$
$I_{R_E} \quad \frac{1}{\alpha} \quad I_B$
$\Rightarrow$ je $I_E \rightarrow$ weniger $U_{BE} \rightarrow$ weniger $I_B \rightarrow$ weniger I_E
$\equiv$ Regelkreis-Stabilisierung
Dimensionierung:
$U_{R_E} = U_{D1} \approx 0,7V$

BJT-Stromquelle/senke



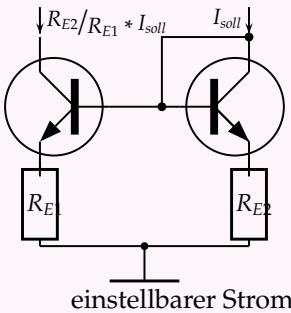
$U_{BE1} = U_{BE2} \quad \Rightarrow \quad I_{C1} = I_{C2}$
T2 kriegt so viel I_B,
dass $B * I_B = I_C$
BJT nicht in Sättigung!

einfacher Stromspiegel



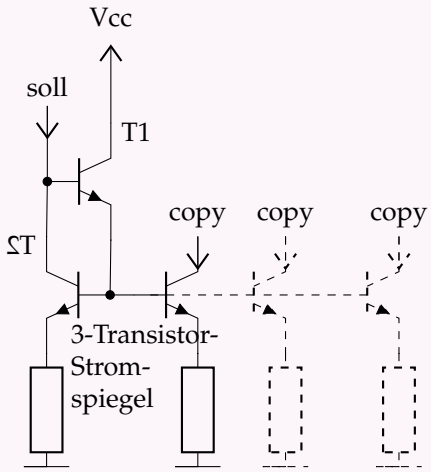
$U_{BE1} = U_{BE2} = U_{BE3} = U_{BE4} \quad \Rightarrow \quad I_{C1} = I_{C2} = I_{C3} = I_{C4}$
T4 kriegt so viel I_B,
dass $B * I_B = I_C$
BJT nicht in Sättigung!

mehrfach-Stromspiegel

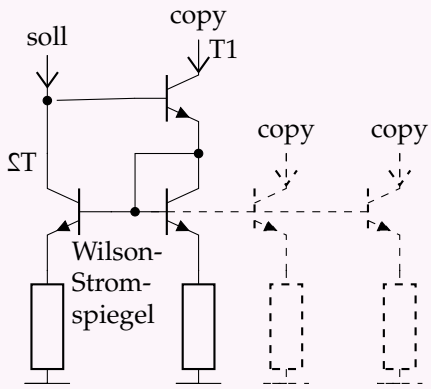


$U_{BE1} = U_{BE2}$ wenn $n * I_{R_{E1}} == I_{R_{E2}}$
$\Rightarrow I_{C1} = I_{C2}$
T2 kriegt so viel I_B,
dass $B * I_B = I_C$
(s. Tietze-Schenk, Kap."Stromspiegel")
BJT nicht in Sättigung!

einstellbarer Stromspiegel



weniger Fälschung durch Basisstrom-Bedarf

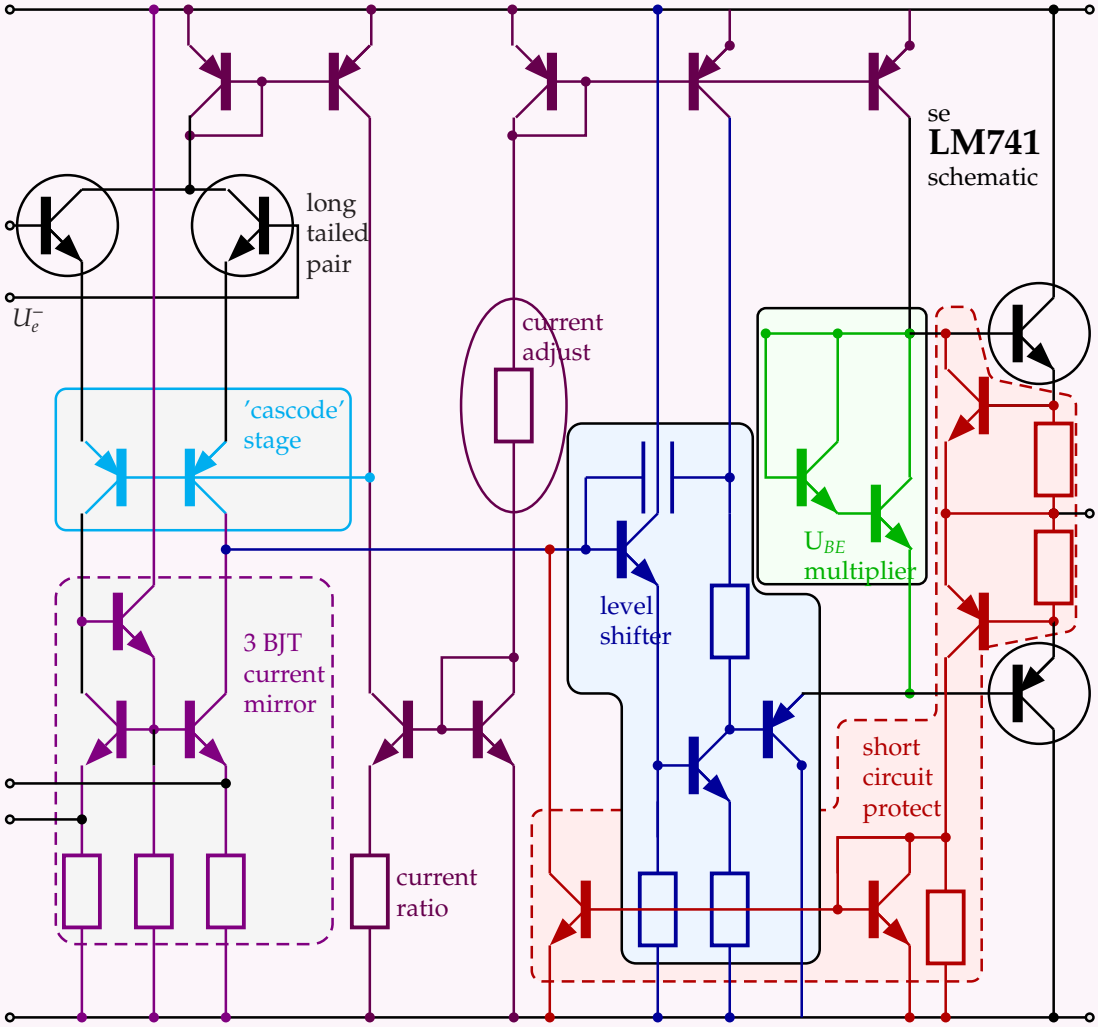


weniger Fälschung durch Basisstrom-Bedarf

1 .12.1 LM741-OPV Innenschaltung

schau mal, wieviele Stromspiegel die drin haben:

bipolar LM741 OP circuit schematic



2 Oszillator

2 .1 Schwing-Bedingung

1. es gibt eine Rückkoppel-Schleife (feedback loop)
2. bei der Schwingfrequenz (oscillation frequency) ω ist
   a) die Schleifenverstärkung (loop gain)

      $\boxed{A_s(\omega) \geq 1}$

      (">" zum "Anschwingen")

      (bei "=" stabile, unverzerrte Schwingung)

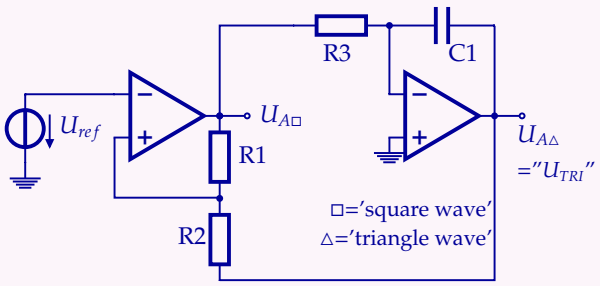
      (bei dauerhaftem ">" entstehen Übersteuerungen)

   b) die Phasenverschiebung (phase lag) in der Schleife

      $\boxed{\Delta\phi(\omega) \equiv 0°}$

      ("≡" = "kongruent",
      d.h. auch 360, 720, 1080°, ...)

2 .2 Rechteck-Dreieck-Oszillator



TBD. (bedeutet 'to be done' ≡ in Arbeit)

Rechteck Dreieck

2 .3 Sinusformer-Netzwerk

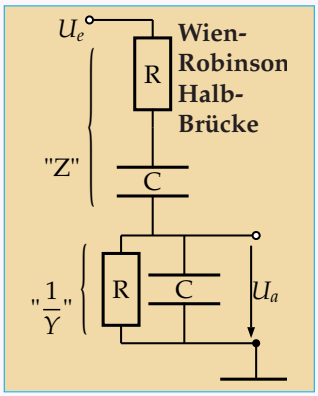
s. *Böhmer* S.228/229

TBD. (bedeutet 'to be done' ≡ in Arbeit)

Sinus Formung

2 .4 Wien-Robinson Oszillator

2 .5 H(jw) zur WienBridge



$$T(s) = \frac{u_a(s)}{u_e(s)} = \Big|_{Z::=R+\frac{1}{jwC}} \Big|_{Y::=\frac{1}{R}+jwC} \quad \frac{\frac{1}{Y}}{Z+\frac{1}{Y}} = \frac{1}{1+Z*Y}$$

$$= \frac{1}{1+\left(R+\frac{1}{jwC}\right)*\left(\frac{1}{R}+jwC\right)}$$

$$= \frac{1}{1+1+jwRC+\frac{1}{jwRC}+1} = \frac{1}{3+j\left(wRC-\frac{1}{wRC}\right)}$$

$$T(\omega{=}\frac{1}{RC}) = \frac{1}{3+j\left(\frac{RC}{RC}-\frac{1}{RC/RC}\right)} = \frac{1}{3+j\left(1-\frac{1}{1}\right)} = \frac{1}{3+j*0} = \frac{1}{3}$$

Knoten Ua:

Kirchhoff: $\underbrace{\frac{Ue-Ua}{Z}}_{I_{R+Xc}} - \underbrace{(I_R+I_{Xc})}_{Ua*Y} = 0 \quad \Longrightarrow \quad \underbrace{\frac{Ua}{Z}+Ua*Y}_{Ua*(\frac{1+ZY}{Z})} = \frac{Ue}{Z}$

$$Ue = Ua*(1+ZY) \quad \Longrightarrow \quad \frac{Ua}{Ue} = \frac{1}{1+\underbrace{ZY}_{\underbrace{(R+\frac{1}{jwC})(\frac{1}{R}+jwC)}_{1+jwRC+\underbrace{\frac{1}{jwRC}}_{\frac{-j}{wRC}}+1}}}$$

$$\Big|_{Z::=R+\frac{1}{j\omega C}}^{Y::=\frac{1}{R}+j\omega C}$$

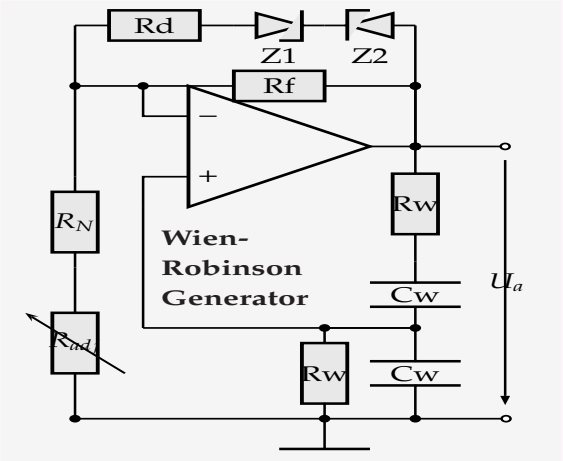
$$\boxed{\frac{Ua}{Ue} = \frac{1}{3+j(wRC-\frac{1}{wRC})}} \quad \Longleftarrow \quad \text{Bei } \circled{w=\frac{1}{RC}} \quad \Longrightarrow \quad wRC = \frac{RC}{RC} = 1$$

$$j(wRC-\tfrac{1}{wRC}) = j(1-1) = 0$$

Bei $\Im m(Ua/Ue) = 0$, $\Re e(Ua/Ue) > 0$ $\Longrightarrow$ $\circled{\Delta\phi(Ua/Ue) = 0}$ $\Longleftarrow$ erfüllt die Oszillator- Schwingbedingung

$$\boxed{w=\frac{1}{RC}} \quad \Longrightarrow \quad \circled{\boxed{\frac{Ua}{Ue}=\frac{1}{3}}}$$

2 .6 Wien-Robinson Osc mit YH



2 .6.1 Blockschaltbild

Erstellen Sie für diese Schaltung ein Blockschaltbild. Es müssen alle Komponenten, die einen Oszillator ausmachen, erkennbar sein. (15 Pkt)

2 .6.2 Brücke

Zeichnen Sie aus der Schaltung die Wien-Robinson-Brücke und berechnen Sie diese für eine Frequenz von 20kHz. Wählen Sie realistische Werte für die Berechnung.

Welches Ausgangssignal erhalten wir, wenn wir eine Eingangsspannung von 3Vss annehmen?

(25 Pkt)

2 .6.3 Zener-Dioden

Erklären Sie die Funktion der beiden Zenerdioden! (10 Pkt)

2 .6.4 Schwingbedingung

Welche Bedingungen müssen erfüllt sein, damit ein Oszillator schwingt? (10 Pkt)

a) Nenne die Anschwingbedingung (10 Pkt)

b) Wie groß muss die Verstärkung der vorgegebenen Schaltung sein, damit die Schwingbedingungen erfüllt ist? Wie wird diese Verstärkung eingestellt? (10 Pkt)

c) Berechnen Sie die Widerstände, damit die gewünschte Verstärkung eingestellt ist. Begründung!(10 Pkt)

2 .6.5 Signal-Diagramm

Zeichnen Sie möglichst genau das erwartete Ausgangssignal auf (X- und Y- Achse möglichst genau bezeichnen) (10 Pkt)

2 .6.6 Wien-Robinson Oszillator mit jFET Amplitudenstabilisierung

Ein oberwellenärmeres Oszillatorsignal ist erreichbar, wenn die eher *'hart'* schaltenden (Zener-) Dioden durch einen Regelkreis mit jFET *als steuerbarem Widerstand* ersetzt werden; vgl. JN-Schaltungen 7 bis 11 Kap.2 S. 100

→ Teile der negativen Halbwelle laden ein C negativ auf

→ "je Amplitude, desto negativ"

→ die geglättete U_{Cap} dient als negative jFET Gatevorspannung U_{GS}

→ die jFET Drain-Source Strecke ist parallel oder in Serie zum Verstärkungs-Einstellwiderstand R_N des Operationsverstärkers OP

→ dieses verringert die v_S (A_V) Verstärkung der OP-Schaltung

→ sodass

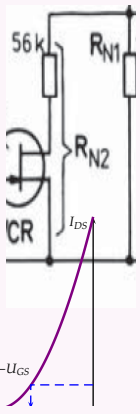
$$\text{Amplitude} \propto \frac{1}{\text{Verstärkung}}$$

"je Amplitude, desto schwächt es"

→ Wahl des jFET:
die Unterschiede liegen vor allem im A- ($U_p = -2V, I_{DSS} = 5mA$), B- ($U_p = -4V, I_{DSS} = 10mA$) oder C-Typ ($U_p = -8V, I_{DSS} = 20mA$), d.h. der C-Typ ist der leitfähigere, aber der A-Typ braucht viel weniger $-U_{GS}$. (Wenn zB. die Oszillatorsignalamplitude $3V_S$ betragen solle und dieses mit einer Si-Diode gleichgerichtet wird, erreiche ich damit $-V_{max} = -2.3V$, das ist für die Sperrung der B- und C-Typen zu wenig)

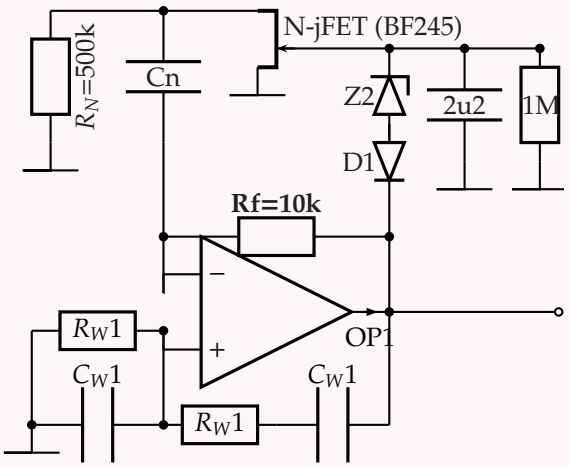
→ Dimensionierung:

1. der FET ist in seinem "ohmschen Bereich" (etwa $-0.5 < U_{DS} < +0.5$) zu betreiben
2. man bestimme die Stromstärke $I_{DS,max}$ durch den FET (ein eventueller R_D Drain-Serienwiderstand ist mitzubedenken), zB. über die gewünschte Oszillatorsignalamplitude ("wie groß ist die Amplitude im FET-Zweig").
3. mit dieser $I_{DS,max}$ sucht man in der Steuerkennlinie ($I_{DS}/-U_{GS}$) die notwendige $-U_{GS}$.
4. damit die Oszillatorsignalamplitude sowohl hinauf wie auch herunter geregelt werden



kann, muss die $-U_{GS}$ (und damit I_{DS}) noch gesteigert wie auch gesenkt werden können.

2 .6.7 Wien-Robinson-Oszillator, regelkreisstabilisiert



Die negative Amplitude der erzeugten Schwingung wird im 2u2-Kondensator als negative FET-Gate-Spannung gespeichert und reduziert seine Kanal-Leitfähigkeit, wodurch sich die OP-Verstärkung verringert. Die Dimensionierung dieses Amplitudenstabilisierungs-Regelkreises ist etwas *fiddly*. Man will die Verstärker-Rückkopplung für die Oszillator-Schwingung steuern und dabei die Gleichspannungs-Rückkopplung (für die Arbeitspunkteinstellung) unberührt lassen. Es kann Dich ganz schön *zwicken* :-))

- Diese Regelung reagiert nicht auf einzelne Sinusschwingungen, sondern *wesentlich* langsamer (einige -hundert Schwingungen; die Hüllkurve entspricht einer RC-Exponentialfunktion $U0 * (1 - e^{-t/RC})$),
  sodass nicht jede einzelne Sinus-Kuppe einzeln verzerrt wird.
  Zeitkonstante τ: R=1M, C=2u2
- Das macht das erzeugte Signal wesentlich *sauberer*.
- Dafür besteht die Gefahr, dass die Amplitude schwankt (wenn der Regelkreis langsam schwingt): Mal ist die Kreisverstärkung ein wenig zu groß, mal ein wenig zu klein. Das führt zu einer → `Amplitudenmodulation`. Das Signal trägt dann eine Amplitudenschwingung bzw. mehr *Amplitudenrauschen*. (Amplitudenrauschen führt zu messtechnisch thematisiertem *Phasenrauschen*)

Achtung mit der Wahl von jFET und Z-Diode:

- es muss sich ausgehen (genug negative Gate-Spannung),
  dass der jFET sperrt (> U_p),
  also kein 'C'-Typ (BF245C), sondern 'A'.
- die Z-Spannung vergrößert die Ua-Amplitude,
  aber bis zur Ua-Ausgangs-Sättigungs-Spannung sollte noch *etwas* Reserve bleiben.
- Nicht *große Ausgangsspannung* ist das primäre Ziel,
  sondern *sauberes und stabiles* Ausgangssignal.
  (*"[. . .] ist mit ±5mV sehr sauber [. . .]"*. No! :-))

Mit nur 5Vcc single-supply wirds *eng*, denn der OP-Aussteuerbereich reicht dann nur von V_{EE}+0.7 - V_{CC}-1.3 ≅ [0,7 .. 3,7V].
(die Amplitude soll ja reichen, um den jFET sperren zu können)
→ Unter 10Vcc:
→ Zener aussa
→ BF245**A** !

2 .7 Wien- Robinson- NF Oszillator

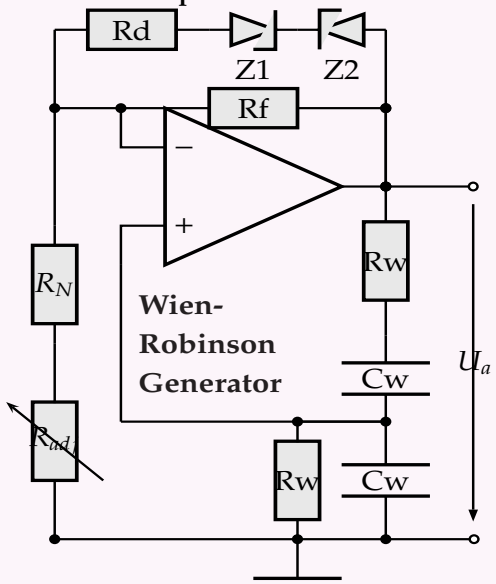
Abseits digitaler Signalgeneratoren (DDS-Prinzip, wie PC- Soundkarten) ist der Wien-Robinson Oszillator eine beliebte, verbreite und einfache NF Sinus- Oszillatorschaltung mit OPs. Vorteile sind:

- der einfache Aufbau mit OP Verstärkern
- geringer Bauteilbedarf
- einfache Berechnung

und nachteilig:

- zusätzlich erforderliche Amplitudenstabilierung
- wenig Frequenzkonstanz
- fehlende HF- Eignung aufgrund OP- Bandbreiten

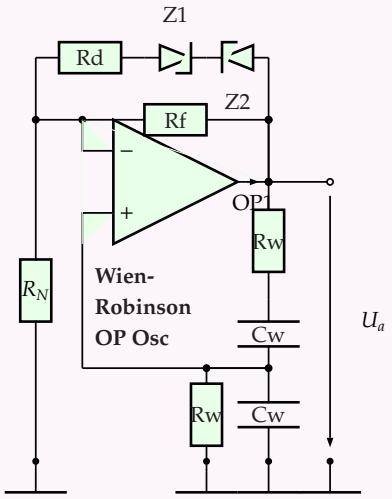
2 .7.1 YHs Schaltplan



Aufgabenstellung (YH):

- Berechnung und Simulation einer Wien-Brücke (Frequenz wird vorgegeben)
- Berechnung und Simulation der kompletten Oszillatorschaltung incl. Amplitudenbegrenzung
- Technische Erklärung der kompletten Schaltung
- Recherche: Welche Möglichkeiten zur Amplitudenbegrenzung haben wir noch?
- Aufbau und Inbetriebnahme am Steckbrett

2 .7.2 Wien-Robinson-Oszillator, diodenstabilisiert



- Immer, wenn Ua die Zenerspannung (Z1 oder -Z2) erreicht, wird die Gegenkopplung (durch Parallelschaltung von Rd) erhöht.
- Das verzerrt jede Kuppe der Sinuskurve. Rf/Rn sollte folglich nahe 3:1 sein (knapp drüber)
- und Rd *relativ* groß (sodass (Rf||Rd)/Rn knapp unter 3:1 ist)

Diese Stabilisierung verzerrt jede einzelne Schwingung etwas, indem sie ab einer gewissen Ausgangsspannung $U_{a,Z}$ die Rückkopplung (feedback) vergrößert (also auch die Schleifenverstärkung loop gain). Das Signal wird folglich aus zwei Ausschnitten verschiedener Grundsignale zusammengesetzt:

a) der *Fuß* von einer zu großen Verstärkung - das wäre ein Trapez-Signal (Rechteck mit schrägen Flanken)
und
b) dem *Hut* eines Sinus von einer zu kleinen Verstärkung (deren Signal langsam *sterben* würde).

Das mit dem langsamen FET-Abschwächer im Feedback (s.u.) arbeitet hingegen viel *sauberer*.

phase lag Osc

2 .8 Phasenschieber (phase lag, phase shift) Oszillator

bei Oszillatorfrequenz gilt:

Drei identische R/C-Glieder schieben die Ua/Ue-Phasenlage bei

$$w = \frac{1}{RC\sqrt{6}}$$

jeweils um

$$\Delta\varphi = +60°$$

(deshalb der Name 'Phasenschieberoszillator', 'phase lag oscillator') und dämpfen das Sinussignal dabei um den Faktor

$$a = \frac{1}{29.0}.$$

Das Verstärkerelement muss also eine Spannungsverstärkung von

$$v_S = A_v = \frac{u_a}{u_e} = 29.0$$

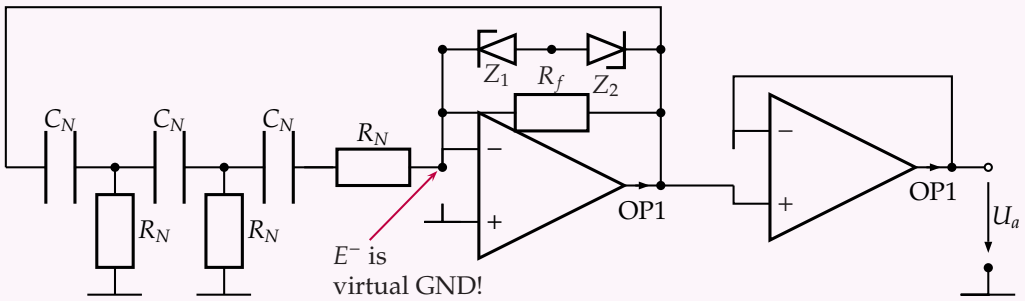
aufweisen, um die Schleifenverstärkung ((closed) loop gain)

$$v_S = 1$$

zu erreichen.

Die Schaltung ist sowohl mit OP als auch diskreten Transistoren (BJT, jFET, MosFET) realisierbar. Wie beim Wien-Robinson Oszillator ist auch hier eine passive oder aktive Amplitudenregelung/-begrenzung sinnvoll (beim diskreten Transistorverstärker aber nicht so nötig, weil deren Nichtlinearitäten schon für sanftere Begrenzung sorgen)

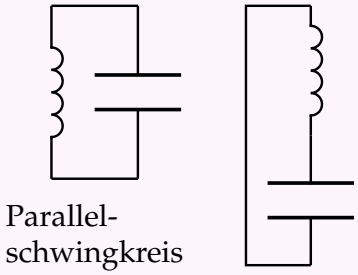
2 .8.1 Phasenschieber-Oszillator



- $\omega = \frac{1}{\sqrt{6} \cdot RC}$
- system gain ≥ 29
- diese Amplitudenstabilisierung ist wie beim Wien-Osc:
  die da verzerrt jede einzelne Sinus-Schwingung `ganz schauderhaft`
  (die Zenerdioden brauchen an Serienwiderstand - Du bist die Ingenieur-Fachkraft!)

s.auch S.100 Kap.2

2 .9 Schwingkreis

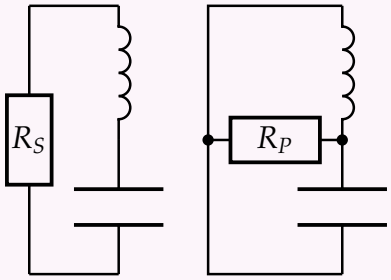


Parallel-schwingkreis

Serien-schwingkreis

Du erkennst keinen Unterschied?

- ich auch nicht.



Serien-schwingkreis

Parallel-schwingkreis

Ist R parallel zu C?

oder parallel zu L?

oder in Serie?

Resonanzfrequenz $w_0 = \frac{1}{\sqrt{LC}}$ $(f_0 = \frac{1}{2\pi\sqrt{L \cdot C}})$

gespeicherte Energie $W = \frac{L \cdot I_{max}^2}{2}$

$= \frac{C \cdot U_{max}^2}{2}$

Kennwiderstand $Z_0 = \sqrt{L / C}$

Resonanztransformation $R_S \cdot R_P = Z_0^2 \approx \frac{L}{C}$

$\rightarrow R_S = Z_0^2 / R_P$

$\rightarrow R_P = Z_0^2 / R_S$

Kreis-Güte (Quality Q_K)[1] $Q_K = \frac{Z_0}{R_S} = \frac{R_P}{Z_0}$

Grunddefinition der Güte::$= \frac{gespeicherteEnergie}{proSchwingungverloreneEnergie}$

relative -3dB-Bandbreite ($Z(f) = \frac{1}{\sqrt{2}} Z_{max}$) $B_r = \frac{\Delta f}{f_{res}} = \frac{f_{res}}{Q}$

Resonanzüberhöhung $U_{max\,an\,C} = U_0 \cdot Q$

$I_{max\,in\,L} = I_0 \cdot Q$

2 .10 Oszillator mit XH

Machen wir nun etwas HF (Hochfrequenz, engl. RF - radio frequency). Die folgenden Oszillatoren sind für Frequenzen etwa im 1..10MHz Bereich gedacht. Da ist das Steckbrett an sich nur mehr eingeschränkt verwendbar - benachbarte Kontaktzeilen haben einige pF Kapazität untereinander. Wenn diese jedoch Teil eines Schwingkreis-C sind, stören sie nicht: Diskrete C-Bauteile wirken um die paar pF mehr und Spulen verringern sich im $\frac{1}{j\omega C}$ Ausmaß.

2 .11 LC- und Xtal- Osc

Dear Readers and ReadInners![6]

Bitte besorge und studiere möglichst viele Oszillatorschaltungen (gutgemeinter 'tip')
So wirst viele finden, an denen man tagelang fummelt und feilt, aber sie wollen nicht wie Du; iXH nenne sie die Fuxkreise (das ist nicht zu verwechseln mit *'Fuchs-Kreis'*, den es als Anpassglied wirklich gibt und 1927 nach Prof. Josef Fuchs OE1JF benannt wurde).
Und Du wirst einige finden, mit denen man nicht blöd genug tun kann (sie haben keinen XH-Spitznamen, da ich mit denen keine Frust-Zeiten mit Sprücheklopfen zu überbrücken habe - da geht's immer sofort weiter), meine Lieblinge habi hier angführt (mein wirklicher, heimlicher Liebling ist der *'lambda oscillator'*, der eignet sich zudem hervorragend als GSG (Gunther- sekkier-Gerät), fehlt aber hier, is zu exotisch)

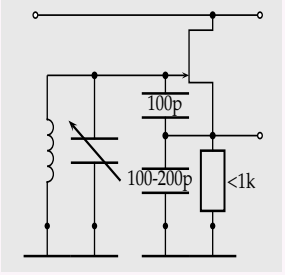
iXH mag Oszillatoren mit wenig Bauteilen und trotzdem guten Eigenschaften:

- ⊗ frequenz- stabil
  Vor allem langsam frequenzwacklige Signale (zB. Temperaturdrift, Spannungsdrift) lassen sich kaum bereinigen, sodass dieser Aspekt das Hauptaugenmerk verdient.
- ⊗ wenig Temperaturkoeffizient
  Temperaturänderungen mögen sich minimal auf die Resonanzfrequenz auswirken.
  'Kimble' VFO ist ein Spitzname für Frequenzwanderung — *'immer auf der Flucht'*, der TV-Serie 'Dr.Kimble' entlehnt.
- ⊗ viel PSRR (power supply rejection ratio)
  Schwankende Betriebsspannung soll die Resonanzfrequenz minimal verstimmen
- ⊗ Harmonische Oberwellen sind eher wurscht, die kann man ausfiltern.
  Oberwellen sind iA kein Problem — oft speist ein Oszillator (VFO) einen Frequenzmischer im Schaltbetrieb (zB Diodenringmischer, DBM, MosFET-Analogschalter), sog. *Schaltmischer*
- ⊗ wenig Phasenrauschen
  Stochastische Signal- Verzerrungen, die einer FM-Modulation mit Rauschen gleichkommen, wird besonders von Halbleiter- Bauteilen erzeugt (C-Dioden, Cbe-, Cgs-, Cak Kapazitäten), jedoch auch Magnetkerne und dielektrische Werkstoffe mit Nichtlinearitäten und Sättigungseffekten. Die saubersten Signale sind direkt am *heissen Ende* von Schwingquarzen entnehmbar.
- ⊗ wenig Mikrofonie
  mechanische Schwingungen von Gehäuse und Unterlage (zB. Tisch) beaufschlagt besonders Luftspulen mit Kapazität, was Oszillatoren verstimmt, sodass FM-Modulation entsteht.
- ⊗ wenig Handkapazitäts-Empfang
  Eine Annäherung von Personal ('Hand') beaufschlagt besonders Luftspulen mit Kapazität, was Oszillatoren natürlich verstimmt.
- ⊗ VFO = variable frequency oscillator, ein durchstimmbarer Osc mit Drehkondensator (polyvaricon), Kapazitätsdiode (varicap diode), Variometer (variable inductor)
- ⊗ Varicap = 'variable capacitor': Drehkondensator, Kapazitätsdiode
- ⊗ als Kapazitätsdiode (varicap diode, BB112, BB512 usw.) kannst auch a simple sonstige Diode (DUS = Diode Universal Silicium) verwenden — eher mit großer Sperrkapazität wie 1N4001, 1N4004 oder LED (rauschen mehr). Richtige Varicap Dioden sind nur besonder linear in der C/U- Kennlinie, sonst nix.
  Diodenabgestimmte LC-Kreise bringen aufgrund der ja mit der Signal-Spannung variierenden Kapazität (die Frequenz ändert sich zyklisch während einer einzigen Signalperiode) ein größeres Problem mit dem sog. 'Phasenrauschen'. An der Cap.Diode sollte das HF-Signal daher klein sein, was man zB. mit kapazitiven Spannungsteilern erreicht.
- ⊗ XO = 'crystal oscillator' Quarzoszillator
  Xtal = 'crystal' = Quarz (quartz)
- ⊗ die 'saubersten' Oszillatoren erzielt man mit XOs und Signalauskopplung direkt über dem Xtal (Schwingquarz) (weil er das Signal dort selbst filtert)

Vom HF- Kurzwellenfunkamateur werden sie für 'eine Handvoll Megahertz' (3.5, 7, 10 MHz udgl.) mit LC oder Xtal auf **Steckbrett, Streifenrasterplatten** oder der ***'dead bug'*** / ***'dirty construction*** method' aufgebaut (das Steckbrett hat zwischen benachbarten Lochreihen ganz deftige, sog. 'parasitäre' (weil unerwünschte) Kapazitäten, die ab ca. 1MHz mitunter desaströs wirken). Am besten geht es (mit Amateurmitteln) auf durchgehend kupferbeschichteten Grundplatten als Masefläche und aufgeklebten Lötinsel-Platinchen ('pads') als Netzknoten (zB. AM-Sender Referat von OE7JHI in KSN5 bei DS)

2 .11.1 LC Colpitts Drainschaltung

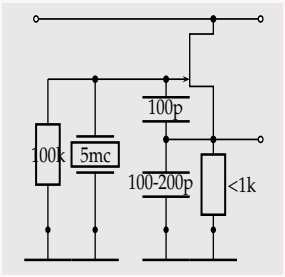
Der Colpitts-OSC in Drain-Schaltung wird in HF- VFOs und XOs unterhalb 20MHz bevorzugt verwendet — er ist einfach und frequenzstabil.



Im *'Böhmer'* findi nur Emitter- und Basisschaltung. Verbreitet aber ist die jFET Drainschaltung. Sie erspart Strombegrenzung:
$I_D = I_{DSS}$ und Arbeitspunkteinstellung:
$V_{GS} = -U_{R_S}$ (via L).

[6] iXH hoffe, die Lächerlichkeit dieser Nichtsprachkonstrukte ist erkennbar

2 .11.2 Xtal Colpitts Drainschaltung



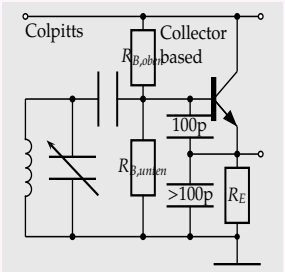
Da der Xtal Gleichstrom sperrt, brauchts R_{GS} für den Arbeitspunkt:

$V_{GS} = -U_{R_S}$

(wie Konstantstromschaltung:

$R_S = \frac{U_P}{I_D}(1 - \sqrt{\frac{I_D}{I_{DSS}}})$

2 .11.3 Der stinknormale Colpitts



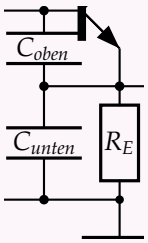
BJT brauchen Basisvorspannung für die Arbeitspunkteinstellung, das rechnet sich gem. Emittergrundschaltung

Dieser DC- Offset ist per Koppel- Cap abzutrennen, was wiederum Resonanz und Schleifenverstärkung beeinflusst.

Eben mehr Bauteile und Probleme als mit jFET.

$U_E = I_C * R_E,$
$U_B = U_E + 0.62V,$
$\frac{R_{B,unten}}{R_{B,oben} + R_{B,unten}} = \frac{U_B}{V_{CC}},$
$Querstrom = 10 \times I_B$
. . . usw.

Dimensionierung:



Der Oszillator lebt von der **Spannungsüberhöhung** des Parallelschwingkreises. C_{oben} bildet mit C_{unten} einen kapazitiven Spannungsteiler, sodass

$\frac{\hat{U}_{oben}}{\hat{U}_{unten}} = \frac{Z_{oben} + Z_{unten}}{Z_{unten}}$

mit

$\hat{U}_{oben} + \hat{U}_{unten} = \hat{U}_B$ und $\hat{U}_{unten} = \hat{U}_E$.

$\hat{U}_{oben}$ wird von der B-E Diode gekappt (begrenzt), womit wir abschätzen können:

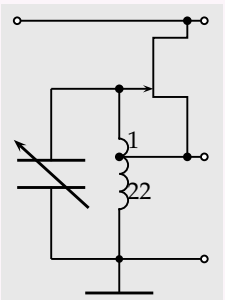
$\hat{U}_{oben} \approx 600mV$

$Z_{Ausgang}$ besteht aus:

- $\frac{Z_B(ImpedanzanderBasis)}{s(= I_C/U_T)}$
- $1/j\omega C_{oben}$ (=parallel zu r_{BE} bzw. r_E)
- $1/j\omega C_{unten}$
- R_E
- Z_{Load} Lastimpedanz

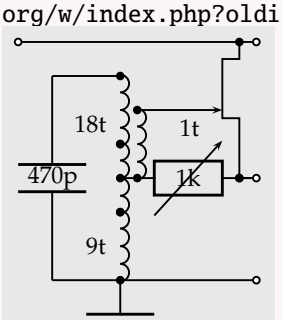
Man solle $\hat{U}_{BE}$ nicht mit der B-E (bzw. G-S) Diode begrenzen, weil das starke Temperaturdrift einbringe.

2 .11.4 ECO Electron Coupled Oscillator



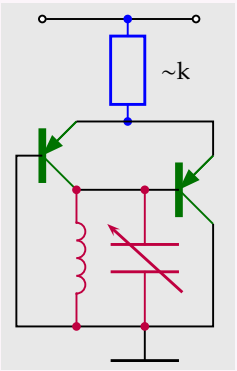
Die ECO Schaltung ist [...] *eine Variante der Hartley-Schaltung. Der Schwingkreis mit L1, L2 und C1 ist zwischen Gate und Source angeschlossen. Der FET Q1 in Drain-Schaltung hat Phasendrehung* $\equiv 0°$ *und Spannungsverstärkung* $v_S < 1$.

Die beiden Induktivitäten L1 und L2 arbeiten als Aufwärts-Spartransformator. Die Verstärkung wird durch das Verhältnis $(L1 + L2)/L2$ *der Induktivitäten bestimmt. Für die Aufrechterhaltung der Schwingung ist eine Verstärkung von mindestens 1 nötig. Für ein oberwellenarmes Ausgangssignal sollte die Verstärkung nur so groß sein, dass der Oszillator sicher anschwingt. [...]* (aus wikipedia Elektronengekoppelter Oszillator Quelle: `http://de.wikipedia.org/w/index.php?oldid=70019391`, 19Jul.2010)



L1, L2 und L3 bitte auf dem gleichen Kern; dieses Experiment zeigt lediglich, dass auch einer Fertigspule eine Koppelwicklung aufgebracht werden kann (Meißner-Oszillator).

2 .11.5 Cathode Follower Oscillator CFO, Differenzverstärker-Oszillator, Peltz Oscillator



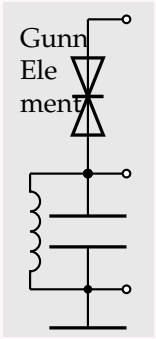
Der CFO ist eine weniger bekannte, vorteilhafte Schaltung mit wenig Bauteilen (2 BJT/FET + 1L + 1C + 1R) und geringer Betriebsspannung (ab ≈0.7Vdc); bei Verwendung von PNP bzw. pFET liegt angenehmerweise zudem der LC-Kreis einseitig an Masse, was die meisten Drehkondensatorkonstruktionen erzwingen.

Dimensionierungshinweis: C-B Dioden kappen $\hat{u}_{LC}$. Je $U_{CE}(T1) \rightarrow$ mehr $U_{BE}(T2) \rightarrow$ mehr $I_E(T2)$ weniger $U_{BE}(T1) \rightarrow$ weniger $I_C(T1)$, kurz:

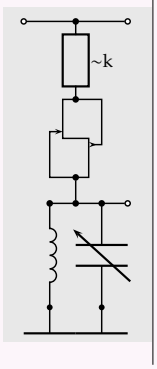
Je $U_{CE}(T1)$ desto weniger $I_C(T1)$.

T1: $U_{CE} \propto \frac{1}{I_C} \rightarrow$ **negative differential Resistance** r_i !

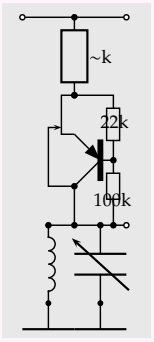
2 .11.6 Lambda Oscillator



Die *Lambda-Oszillatoren* sind ungewohnte, der GHz-Technik entnommene Schaltungen, die ab ca. 1960 mit Gunn-, Esaki- und IMPATT- Dioden *negativ differentielle Kennlinien* (= Strom I_A fällt trotz steigender Spannung U_{AK}, was *negativen Widerstand* wie bei Strom- und Spannungsquellen bedeutet) zur Schwingkreis-Entdämpfung durch simple Reihenschaltung ermöglichten.

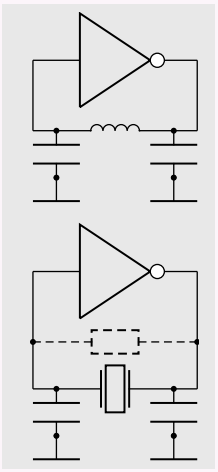


Später entwickelte der vielzitierte *Nomen Nescimus* Ersatzschaltungen mit ähnlichen Kennlinienverläufen aus der Kombination eines N-Kanal jFET und eines p-Kanal jFET für dieselben Oszillatorkonzepte bei niedrigeren Frequenzen Idealerweise braucht man auch hier nur wenig Bauteile.



Aufgrund schlechterer Verfügbarkeit und Bandbreite der seltenen p-Kanal jFET (zB 1N3820) ersetzte man diesen durch PNP-BJT. Das verkleinert den Spannungsbereich mit fallender Kennlinie, aber sie wird dort auch steiler. Insgesamt arbeitet das dem CFO sehr ähnlich

2 .11.7 Pierce Oscillator



Mit den zwei Cap's ('Bürde-Kapazität' (DC9XP), ~30pF) bildet die Schaltung ein die Phase 180° schiebendes π-Filter.

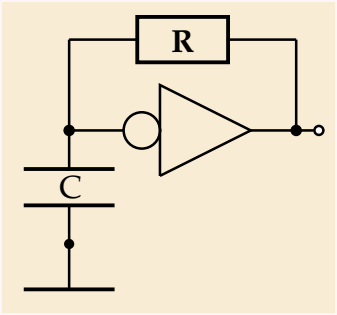
Ein Xtal fungiert als Induktivität hoher Güte und schwingt unterhalb seiner Resonanz.

Mit Serien-C kann die Frequenz (wieder) erhöht werden.

Mit einem Gegenkoppel- R erlauben ungepufferte Logikgatter (CMOS:'A') Linearbetrieb bei Sinusschwingung; gepufferte erzwingen Rechtecksignale.

2 .12 Digital-Oszillator (Taktgeber) mit Gatterschaltungen

2 .12.1 mit 2 Stk Inverter



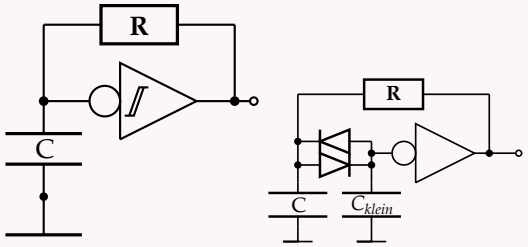
Oszillatorfrequenz:
R2 im Labor egal
@400Hz..400kHz

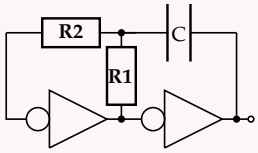
$$f \approx \frac{1}{3R_1C_1}$$

RC-Oszillatoren mit Gatterschaltungen wie
- NOT Gatter (Inverter)
- NAND / NOR Gatter
- Schmitt-Trigger
- Flipflops

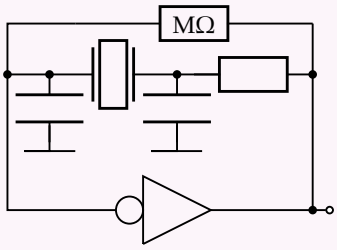
werden in digitalen Logikschaltungen als *Clock*generatoren und *Timer* verwendet, da sie besser kompatibel und *integrierbar* sind als lineare, analoge HF-LC-Oszillatoren, deren Signal man zudem noch in Logikpegel umsetzen müsste.

2 .12.2 mit 1 Stk Inverter





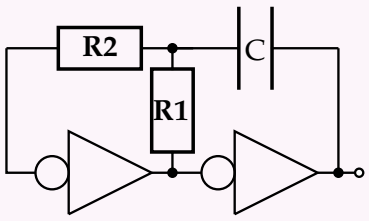
2 .12.3 mit 2 Stk Inverter



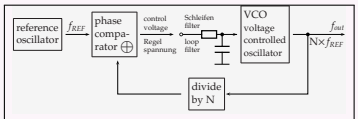
Oszillatorfrequenz:
$74HC : 10R_1 > R_2 > 2R_1$
@400Hz..400kHz

$$f \approx \frac{1}{3R_1C_1}$$

2 .12.4 mit Xtal



2 .12.5 zum Clock-Osc mit 2 Invertern (wie 4060)



Oszillatorfrequenz:

$74HC : 10R_1 > R_2 > 2R_1$

$@400Hz..400kHz$

$$f \approx \frac{1}{3R_1C_1}$$

bitte besichtige den Inverter-Oscillator s. `https://www.falstad.com/circuit/e-inv-osc.html`

simuliere selbst in 'C':

```
/* digOscOP1.c  XH@26Jan'21
 * Simulation des Inverter-Digitaloszillators
 *   mittels FE-numerischer Integration
 * compile:  'gcc digOscOP1.c -o digOsc.exe'
 * run:      './digOsc.exe'
 * stop:     Strg+C
 * OS:       Linux (Knoppix v8.6)
 */


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

struct timespec t={0L, 300000000L};

#define Vcc     5.0
double  Uc   = 0.0;
double  Uin;
double  R1   = 1000.;
double  C    = 1E-7;
double  dt   = 1E-5;
double  Ir,dUc,Uin;
int     inv1=0;
int     inv2=1;


int main(int argc, char *argv[]){

  for(;;){
    Uin = (inv2*Vcc) + Uc;
    if( Uin > Vcc/2 ){  inv1 = 0;
    }else{              inv1 = 1;
    }
    inv2 = inv1 ? 0 : 1;

    Ir   = (inv1*Vcc-Uin)/R1;
    dUc  = dt*Ir/C;
    Uc   += dUc;

    printf("Ir=%9g, dUc=%9g, Uc=%9g, Uin=%9g\n", Ir,dUc,Uc,Uin);
    nanosleep( &t, NULL );
  }
}
```

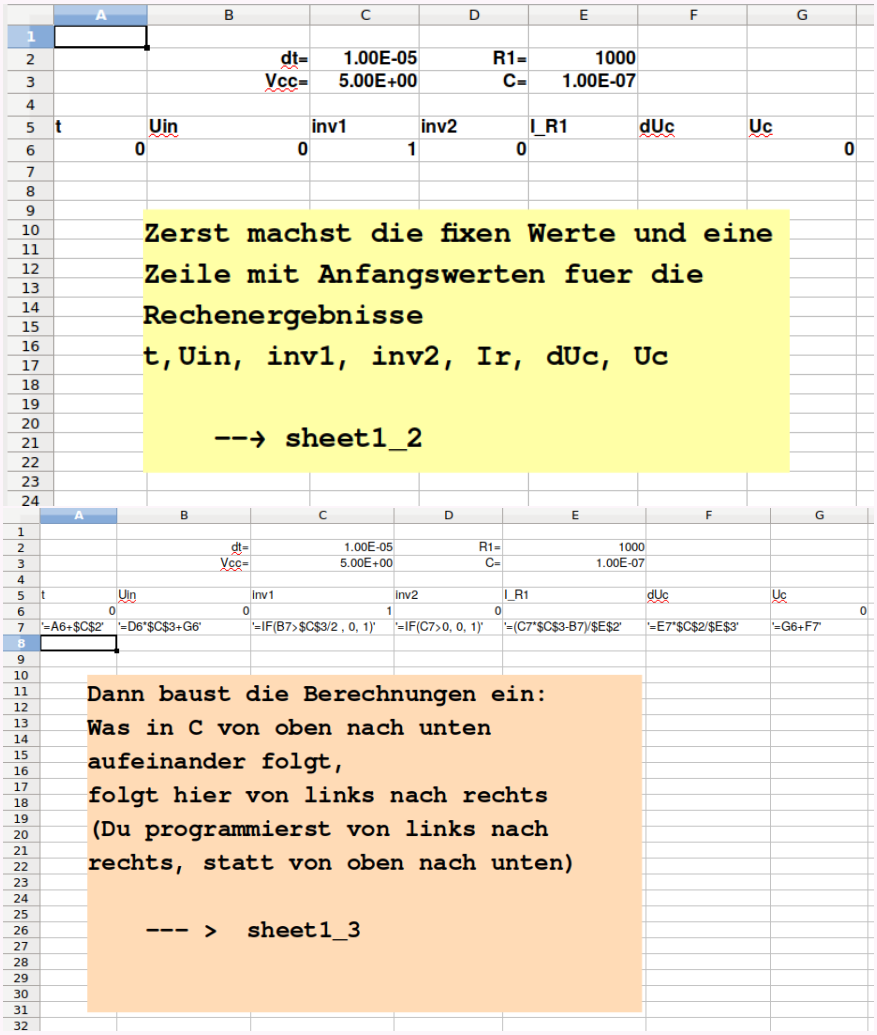
'digOscOP2.c' hat nur an oszilloskopigen Output (ASCII-Art 90° gedreht) dabei:

```
    printf("Ir=% 6.3f, dUc=% 6.3f, Uc=% 6.3f %-.*s*Uin=% 6.3f\n", Ir,dUc,Uc,
        (int)(5.*(4.+Uin)),
        "                                   "
        "                                   ",
        Uin );
```

mit Excel
`digOscOP1_sim_mit_Excel.ods` und mit
`https://www.falstad.com/circuit/e-index.html`

→ Auch hier kannst Du Werte und Schaltung ändern, Messwerte und Oszillogramme einblenden (s.'Draw'-Menu)

f = f' in Excel: df1dx-Vb2.ods



	A	B	C	D	E	F	G
2		dt=	1.00E-05	R1=	1000		
3		Vcc=	5.00E+00	C=	1.00E-07		
5	t	Uin	inv1	inv2	I_R1	dUc	Uc
6	0	0	1	0			0

Zerst machst die fixen Werte und eine Zeile mit Anfangswerten fuer die Rechenergebnisse
t,Uin, inv1, inv2, Ir, dUc, Uc

--→ sheet1_2

	A	B	C	D	E	F	G
2		dt=	1.00E-05	R1=	1000		
3		Vcc=	5.00E+00	C=	1.00E-07		
5	t	Uin	Inv1	inv2	I_R1	dUc	Uc
6	0	0	1	0			0
7	'=A6+$C$2'	'=D6*$C$3+G6'	'=IF(B7>$C$3/2 , 0, 1)'	'=IF(C7>0, 0, 1)'	'=(C7*$C$3-B7)/$E$2'	'=E7*$C$2/$E$3'	'=G6+F7'

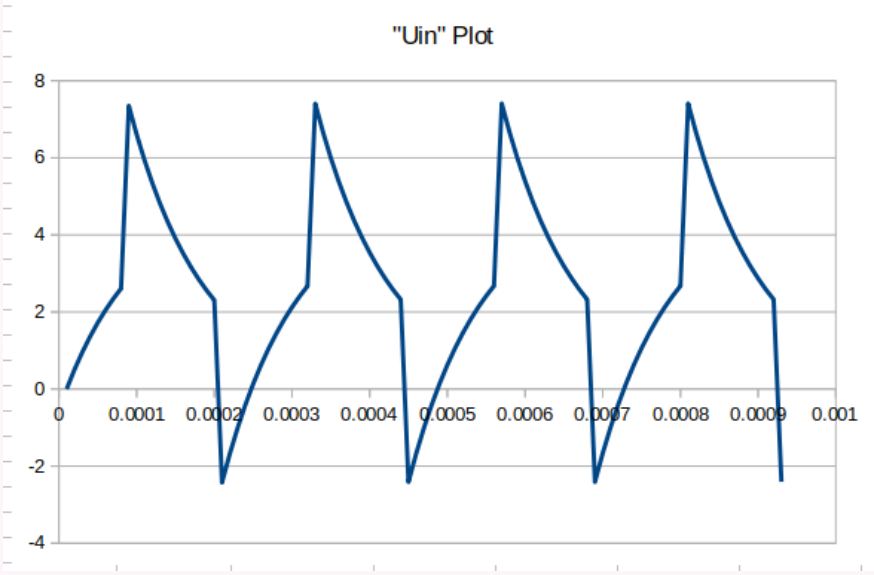
Dann baust die Berechnungen ein:
Was in C von oben nach unten aufeinander folgt,
folgt hier von links nach rechts
(Du programmierst von links nach rechts, statt von oben nach unten)

--- > sheet1_3

	A	B	C	D	E	F	G
1							
2			dt=	1.00E-05	R1=	1000	
3			Vcc=	5.00E+00	C=	1.00E-07	
4							
5	t	Uin	inv1	inv2	I_R1	dUc	Uc
6	0	0	1	0			0
7	1E-05	0	1	0	0.005	0.5	0.5
8	2E-05	0.5	1	0	0.0045	0.45	0.95
9	3E-05	0.95	1	0	0.00405	0.405	1.355
10	4E-05	1.355	1	0	0.003645	0.3645	1.7195
40	0.00034	6.662714779481 55	0	1	-0.006662714 78	-0.666271479 5	0.99644330153
41	0.00035	5.99644330153339	0	1	-0.0059964433	-0.59964433015	0.39679897138
42	0.00036	5.39679897138005	0	1	-0.00539679897	-0.53967989714	-0.14288092576

Nun
(1) markierst Du diese programmierte Zeile (hier #7) und vervielfachst sie (durch "ziehen") nach unten (zB. weitere 100 Zeilen).
Dann
(2) markierst Du den x/y-Wertebereich (hier Zellen A7 bis B99)
Und
(3) wählst *'insert'-'chart'-'XY scatter'-'Lines only'*

Fertig --→ (next sheet)



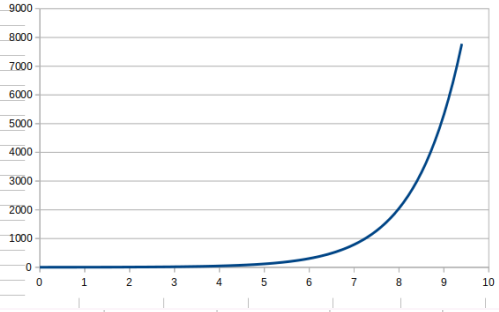
```
* ***********************************************
* dieser Code erzeugt a "ASCII-Art-Oszillogramm"
* Uin/t wennma en Kopf 90 Grad rechts draht
* ***********************************************
Ir=-0.005, dUc=-0.500, Uc=-0.500                                          *Uin= 5.000
Ir=-0.004, dUc=-0.450, Uc=-0.950                                        *Uin= 4.500
Ir=-0.004, dUc=-0.405, Uc=-1.355                                      *Uin= 4.050
Ir=-0.004, dUc=-0.364, Uc=-1.720                                    *Uin= 3.645
Ir=-0.003, dUc=-0.328, Uc=-2.048                                  *Uin= 3.280
Ir=-0.003, dUc=-0.295, Uc=-2.343                                *Uin= 2.952
Ir=-0.003, dUc=-0.266, Uc=-2.609                               *Uin= 2.657
Ir= 0.003, dUc= 0.261, Uc=-2.348                             *Uin= 2.391
Ir= 0.007, dUc= 0.735, Uc=-1.613         *Uin=-2.348
Ir= 0.007, dUc= 0.661, Uc=-0.952            *Uin=-1.613
Ir= 0.006, dUc= 0.595, Uc=-0.356               *Uin=-0.952
Ir= 0.005, dUc= 0.536, Uc= 0.179                  *Uin=-0.356
Ir= 0.005, dUc= 0.482, Uc= 0.661                    *Uin= 0.179
Ir= 0.004, dUc= 0.434, Uc= 1.095                       *Uin= 0.661
Ir= 0.004, dUc= 0.390, Uc= 1.486                         *Uin= 1.095
Ir= 0.004, dUc= 0.351, Uc= 1.837                           *Uin= 1.486
Ir= 0.003, dUc= 0.316, Uc= 2.153                             *Uin= 1.837
Ir= 0.003, dUc= 0.285, Uc= 2.438                              *Uin= 2.153
Ir= 0.003, dUc= 0.256, Uc= 2.694                                *Uin= 2.438
Ir=-0.003, dUc=-0.269, Uc= 2.425                                 *Uin= 2.694
Ir=-0.007, dUc=-0.742, Uc= 1.682                                                *Uin= 7.425
Ir=-0.007, dUc=-0.668, Uc= 1.014                                              *Uin= 6.682
Ir=-0.006, dUc=-0.601, Uc= 0.413                                            *Uin= 6.014
Ir=-0.005, dUc=-0.541, Uc=-0.129                                          *Uin= 5.413
Ir=-0.005, dUc=-0.487, Uc=-0.616                                        *Uin= 4.871
Ir=-0.004, dUc=-0.438, Uc=-1.054                                      *Uin= 4.384
Ir=-0.004, dUc=-0.395, Uc=-1.449                                    *Uin= 3.946
Ir=-0.004, dUc=-0.355, Uc=-1.804                                   *Uin= 3.551
Ir=-0.003, dUc=-0.320, Uc=-2.123                                 *Uin= 3.196
Ir=-0.003, dUc=-0.288, Uc=-2.411                                *Uin= 2.877
Ir=-0.003, dUc=-0.259, Uc=-2.670                               *Uin= 2.589
Ir= 0.003, dUc= 0.267, Uc=-2.403                             *Uin= 2.330
Ir= 0.007, dUc= 0.740, Uc=-1.663         *Uin=-2.403
Ir= 0.007, dUc= 0.666, Uc=-0.996            *Uin=-1.663
Ir= 0.006, dUc= 0.600, Uc=-0.397               *Uin=-0.996
Ir= 0.005, dUc= 0.540, Uc= 0.143                  *Uin=-0.397
Ir= 0.005, dUc= 0.486, Uc= 0.629                    *Uin= 0.143
Ir= 0.004, dUc= 0.437, Uc= 1.066                       *Uin= 0.629
Ir= 0.004, dUc= 0.393, Uc= 1.459                         *Uin= 1.066
Ir= 0.004, dUc= 0.354, Uc= 1.813                           *Uin= 1.459
```

```
/* digOscOP1.c  XH@26Jan'21
 * Berechnung des Digitaloszillator-Arbeitspunktes
 *   mittels FE-numerischer Integration
 * compile:  'gcc digOscOP1.c -o digOsc.exe'
 * run:      './digOsc.exe'
 * stop:     Strg+C
 * OS:       Linux (Knoppix v8.6)
 */

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

struct timespec t={0L, 300000000L};

#define Vcc     5.0
double  Uc  = 0.0;
double  Uin;
double  R1  = 1000.;
double  C   = 1E-7;
double  dt  = 1E-5;
double  Ir,dUc,Uin;
int     inv1=0;
int     inv2=1;

int main(int argc, char *argv[]){

  for(;;){
    Uin = (inv2*Vcc) + Uc;
    if( Uin > Vcc/2 ){  inv1 = 0;
    }else{              inv1 = 1;
    }
    inv2 = inv1 ? 0 : 1;

    Ir   = (inv1*Vcc-Uin)/R1;
    dUc  = dt*Ir/C;
    Uc   += dUc;

    printf("Ir=%9g, dUc=%9g, Uc=%9g\n", Ir,dUc,Uc);
    nanosleep( &t, NULL );
  }
}
```

	A	B	C	D	E	F	G	H
1								
2			dt=	1.00E-05	R1=	1000		
3			Vcc=	5.00E+00	C=	1.00E-07		
4								
5	t	Uin	inv1	inv2	I_R1	dUc	Uc	
6	0	0	1	0			0	
7	1E-05	0	1	0	0.005	0.5	0.5	
8	2E-05	0.5	1	0	0.0045	0.45	0.95	
9	3E-05	0.95	1	0	0.00405	0.405	1.355	
10	4E-05	1.355	1	0	0.003645	0.3645	1.7195	
11	5E-05	1.7195	1	0	0.0032805	0.32805	2.04755	
12	6E-05	2.04755	1	0	0.00295245	0.295245	2.342795	
13	7E-05	2.342795	1	0	0.002657205	0.2657205	2.6085155	
14	8E-05	2.6085155	0	1	-0.0026085155	-0.26085155	2.34766395	
15	9E-05	7.34766395	0	1	-0.00734766395	-0.734766395	1.612897555	
16	0.0001	6.612897555	0	1	-0.00661289756	-0.6612897555	0.9516077995	
17	0.00011	5.9516077995	0	1	-0.0059516078	-0.59516077995	0.35644701955	
18	0.00012	5.35644701955	0	1	-0.00535644702	-0.53564470196	-0.17919768241	
19	0.00013	4.820802317595	0	1	-0.00482080232	-0.48208023176	-0.66127791416	
20	0.00014	4.3387220858355	0	1	-0.00433872209	-0.43387220858	-1.09515012275	
21	0.00015	3.90484987725195	0	1	-0.00390484988	-0.39048498773	-1.48563511047	
22	0.00016	3.51436488952675	0	1	-0.00351436489	-0.35143648895	-1.83707159943	
23	0.00017	3.16292840057408	0	1	-0.0031629284	-0.31629284006	-2.15336443948	
24	0.00018	2.84663556051667	0	1	-0.00284663556	-0.28466355605	-2.43802799554	
25	0.00019	2.561972004465	0	1	-0.002561972	-0.25619720045	-2.69422519598	
26	0.0002	2.3057748040185	1	0	0.0026942252	0.2694225196	-2.42480267638	
27	0.00021	-2.42480267638335	1	0	0.00742480268	0.74248026764	-1.68232240875	
28	0.00022	-1.68232240874501	1	0	0.00668232241	0.66823224087	-1.01409016787	
29	0.00023	-1.01409016787051	1	0	0.00601409017	0.60140901679	-0.41268115108	
30	0.00024	-0.412681151083459	1	0	0.00541268115	0.54126811511	0.12858696402	
31	0.00025	0.128586964024887	1	0	0.00487141304	0.4871413036	0.61572826762	
32	0.00026	0.615728267622398	1	0	0.00438427173	0.43842717324	1.05415544086	
33	0.00027	1.054155440860 16	1	0	0.00394584456	0.39458445591	1.44873989677	

		dx=	1.00E-01

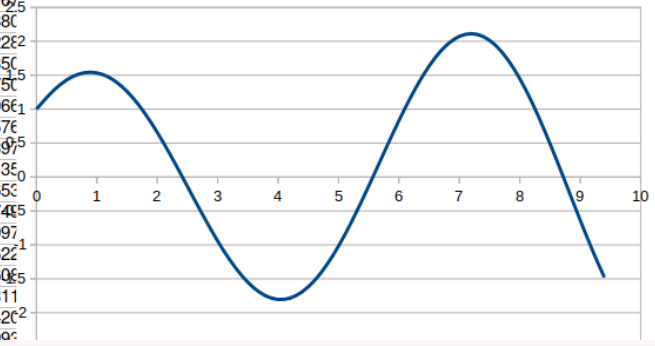
x	f=f+df	f1=f	df=f1*dx
0	1	1	0.1
0.1	1.1	1.1	0.11
0.2	1.21	1.21	0.121
0.3	1.331	1.331	0.1331
0.4	1.4641	1.4641	0.14641
0.5	1.61051	1.61051	0.161051
0.6	1.771561	1.771561	0.1771561
0.7	1.9487171	1.9487171	0.19487171
0.8	2.14358881	2.14358881	0.214358881
0.9	2.357947691	2.357947691	0.2357947691
1	2.5937424601	2.5937424601	0.25937424601
1.1	2.85311670611	2.85311670611	0.28531167061
1.2	3.138428376721	3.13842837672	0.31384283767
1.3	3.4522712143931	3.45227121439	0.34522712144
1.4	3.79749833583241	3.79749833583	0.37974983358
1.5	4.17724816941565	4.17724816942	0.41772481694
1.6	4.59497298635722	4.59497298636	0.45949729864
1.7	5.05447028499294	5.05447028499	0.5054470285
1.8	5.55991731349223	5.55991731349	0.55599173135
1.9	6.11590904484146	6.11590904484	0.61159090448
2	6.72749994932256	6.72749994933	0.67274999493
2.1	7.40024994425816	7.40024994426	0.74002499443
2.2	8.14027493868398	8.14027493868	0.81402749387
2.3	8.95430243255237	8.95430243255	0.89543024326
2.4	9.84973267580761	9.84973267581	0.98497326758
2.5	10.8347059433884	10.8347059434	1.08347059434
2.6	11.9181765377272	11.9181765377	1.19181765377



		dx=	1.00E-01

x	f=f+df	f2=(-f)	df1=f2*dx	f1=f1+df1	df=f1*dx	
0	1	0	0	1	0.1	
0.1	1.1	-1.1	-0.11	1	0.1	
0.2	1.2	-1.2	-0.12	0.89	0.089	
0.3	1.289	-1.289	-0.1289	0.77	0.077	f=-f''
0.4	1.366	-1.366	-0.1366	0.6411	0.06411	f'=df/dx
0.5	1.43011	-1.43011	-0.143011	0.5045	0.05045	f''=df'/dx
0.6	1.48056	-1.48056	-0.148056	0.361489	0.0361489	
0.7	1.516					
0.8	1.538					
0.9	1.54422					
1	1.535					
1.1	1.51037					
1.2	1.47038					
1.3	1.41528					
1.4	1.34547					
1.5	1.26151					
1.6	1.16410					
1.7	1.05407					
1.8	0.93241					
1.9	0.80020					
2	0.65866					
2.1	0.50913					
2.2	0.35301					
2.3	0.19179					

df=f'*dx
df'=f''*dx



2 .13 PLL phase locked loop, phasenstarre Regelschleife

Die 'phasenstarre Regelschleife' (phase locked loop) erzeugt aus dem Vergleich der Phasenlagen zweier Signalfrequenzen ein Fehlersignal wie die klassische Regelung (control loop).

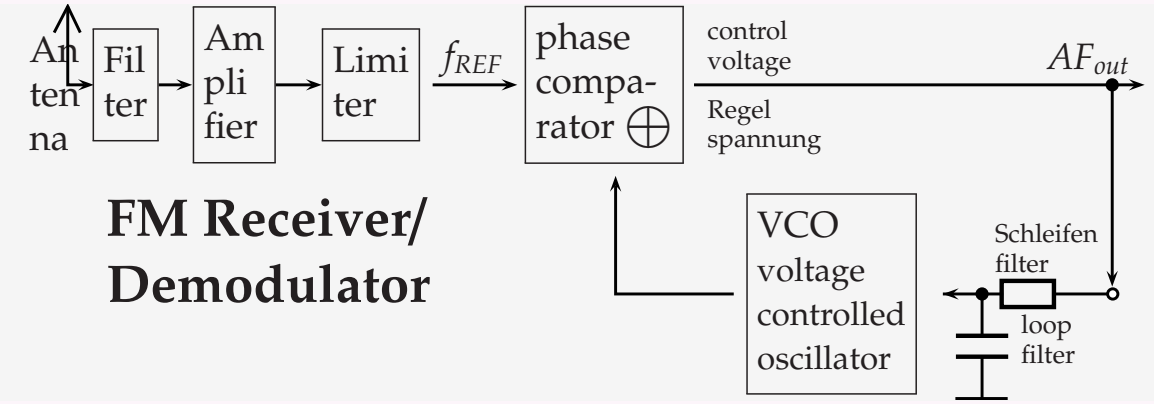
Man teilt nun die Frequenz eines *freilaufenden (free running)*, spannungsgesteuerten Oszillators (VCO, voltage controlled oscillator) um den Faktor $\frac{1}{N}$ 'herunter' und regelt dieses *phasenstarr* zu einem *stabilen Referenzsignal* nach.

- Man erhält die
  N-fache Ausgangsfrequenz mit der **Stabilität** des Referenz-Signals.
- Faktor N **variierbar**,
  → Ausgangsfrequenz **in Stufen** schaltbar
- Referenzfrequenz verstimmen,
  → ganze PLL *zieht* mit.
- Fehlerspannung mit NF-Signal (engl. AF) addieren,
  → Frequenzmodulation (FM) der PLL.
- umgekehrt: Frequenzmodulierte Referenz
  → Fehlersignal enthält demodulierte FM!

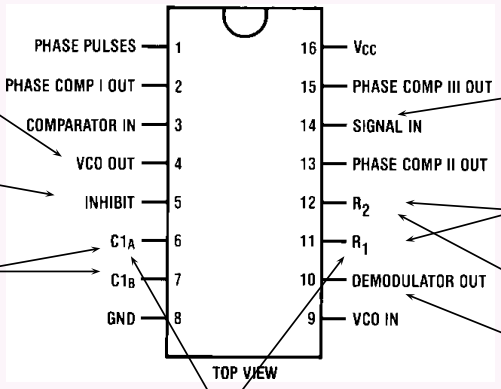
Die PLL wird deswegen gerne
a) zur Frequenzsynthese und
b) FM (de-)modulation
verwendet.

Begriffe:
die PLL ist *eingerastet*, *locked*, *fängt sich*

2 .13.1 PLL schema



2 .13.2 FM demod:



output of the VCO allows direct coupling to COS/MOS frequency dividers

A logic 0 on the INHIBIT input 'enables' the VCO and the source follower

The VCO requires one external capacitor C1 . . .

phase comparator signal input (terminal 14) can be directly coupled

. . . and one or two external resistors (R1 or R1 and R2)

R2 enables the VCO to have a frequency offset

a source follower output of the VCO input voltage is provided at terminal 10

R1 and C1 determine the frequency range

FAIRCHILD SEMICONDUCTOR™

April 1994
Revised April 1999

74VHC4046
CMOS Phase Lock Loop

General Description

The VHC4046 is a low power phase lock loop utilizing advanced silicon-gate CMOS technology to obtain high frequency operation both in the phase comparator and VCO sections. This device contains a low power linear voltage controlled oscillator (VCO), a source follower, and three phase comparators. The three phase comparators have a common signal input and a common comparator input. The signal input has a self biasing amplifier allowing signals to be either capacitively coupled to the phase comparators with a small signal or directly coupled with standard input logic levels. This device is similar to the CD4046 except that the Zener diode of the metal gate CMOS device has been replaced with a third phase comparator.

Phase Comparator I is an exclusive OR (XOR) gate. It provides a digital error signal that maintains a 90 phase shift between the VCO's center frequency and the input signal (50% duty cycle input waveforms). This phase detector is more susceptible to locking onto harmonics of the input frequency than phase comparator I, but provides better noise rejection.

Phase comparator III is an SR flip-flop gate. It can be used to provide the phase comparator functions and is similar to the first comparator in performance.

Phase comparator II is an edge sensitive digital sequential network. Two signal outputs are provided, a comparator output and a phase pulse output. The comparator output is a 3-STATE output that provides a signal that locks the VCO output signal to the input signal with 0 phase shift between them. This comparator is more susceptible to noise throwing the loop out of lock, but is less likely to lock onto harmonics than the other two comparators.

In a typical application any one of the three comparators feed an external filter network which in turn feeds the VCO input. This input is a very high impedance CMOS input which also drives the source follower. The VCO's operating frequency is set by three external components connected to the $C1_A$, $C1_B$, R_1 and R_2 pins. An inhibit pin is provided to disable the VCO and the source follower, providing a method of putting the IC in a low power state.

The source follower is a MOS transistor whose gate is connected to the VCO input and whose drain connects the Demodulator output. This output normally is used by tying a resistor from pin 10 to ground, and provides a means of looking at the VCO input without loading down modifying the characteristics of the PLL filter.

Features

- Low dynamic power consumption: (V_{CC} = 4.5V)
- Maximum VCO operating frequency: 12 MHz (V_{CC} = 4.5V)
- Fast comparator response time (V_{CC} = 4.5V)
  - Comparator I: 25 ns
  - Comparator II: 30 ns
  - Comparator III: 25 ns
- VCO has high linearity and high temperature stability
- Pin and function compatible with the 74HC4046

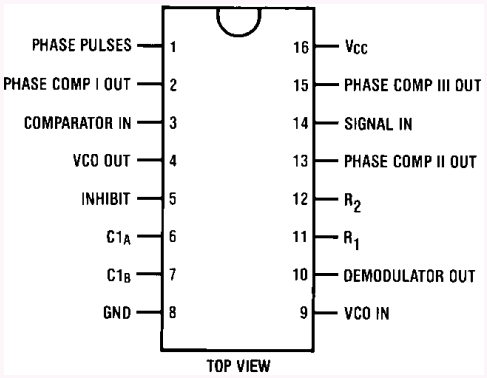
14(8)	13(7)	12(6)	11(5)	10(4)	9(3)	8(2)
Vcc	Dis	Thres	Ctl	-Rst	Out	Trig

NE-556

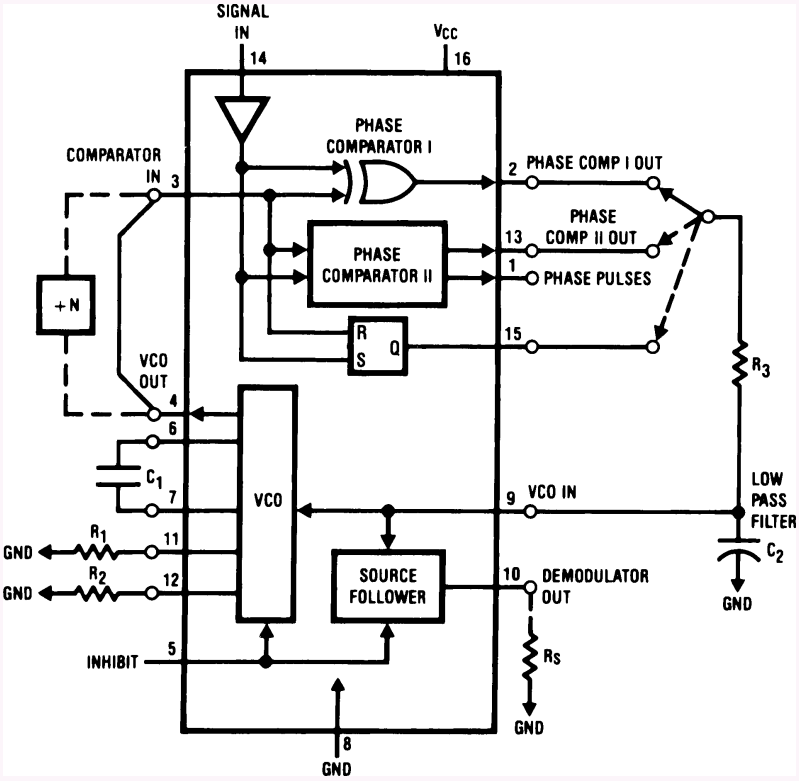
Dis	Thr	Ctl	-Rst	Out	Trig	Vss
1(7)	2(6)	3(5)	4(4)	5(3)	6(2)	7(1)

74VHC4046

Connection Diagram



Block Diagram



(from data sheets 'SGS-Thomson HCC/HCF4046B', 'FAIRCHILD SEMICONDUCTOR 74VHC4046':)

2 .13.3 VCO Section

The VCO requires one external capacitor C1 and one or two external resistors (R1 or R1 and R2). Resistor R1 and capacitor C1 determine the frequency range of the VCO and resistor R2 enables the VCO to have a frequency offset if required.

The high input impedance ($10^{12}\Omega$) of the VCO simplifies the design of low-pass filters by permitting the designer a wide choice of resistor-to-capacitor ratios. In order not to load the low-pass filter, a source-follower output of the VCO input voltage is provided at terminal 10 (DEMODULATED OUTPUT). If this terminal is used, a load resistor (R_S) of $10k\Omega$ or more should be connected from this terminal to VSS . If unused this terminal should be left open. The VCO can be connected either directly or through frequency dividers to the comparator input of the phase comparators.

A full COS/MOSlogic swing is available at the output of the VCO and allows direct coupling to COS/MOS frequency dividers such as the HCC/HCF4024B, HCC/HCF4018B, HCC/HCF4020B, HCC/HCF4022B, HCC/HCF4029B,and HBC/HBF4059A. One or more HCC/HCF4018B (Presettable Divide-by-N Counter) or HCC/HCF4029B (Presettable Up/Down Counter), or HBC/HBF4059A (Programmable Divide-by-N Counter), together with the HCC/HCF4046B (Phase-Locked Loop) can be used to build a micropower low-frequency synthesizer.

A logic 0 on the INHIBIT input 'enables' the VCO and the source follower, while a logic 1 turns off both to minimize stand-by power consumption.

2 .13.4 Phase Comparators

The phase-comparator signal input (terminal 14) can be direct- coupled provided the signal swing is within COS/MOS logic levels [logic '0' ≤ 30% (VDD–VSS), logic 1 ≥ 70% (VDD-VSS)]. For smaller swings the signal must be capacitively coupled to the self-biasing amplifier at the signal input. Phase comparator I is an exclusive-OR network; it operates analagously to an overdriven balanced mixer. To maximize the lock range, the signal- and comparator- input frequencies must have a 50% duty cycle. With no signal or noise on the signal input, this phase comparator has an average output voltage equal to VDD/2. The low pass filter connected to the output of phase comparator I supplies the averaged voltage to the VCO input, and causes the VCO to oscillate at the center frequency (f_0). The frequency range of input signals on which the PLL will lock if it was initially out of lock is defined as the frequency capture range ($2\ f_c$). The frequency range of input signals on which the loop will stay locked if it was initially in lock is defined as the frequency lock range ($2\ f_L$). The capture range is ≤ the lock range. With phase comparator I the range of frequencies over which the PLL can acquire lock (capture range) is dependent on the low pass filter characteristics, and can be made as large as the lock range. Phase-comparator I enables a PLL system to remain in lock in spite of high amounts of noise in the input signal. One characteristic of this type of phase comparator is that it may lock onto input frequencies that are close to harmonics of the VCO center frequency. A second characteristic is that the phase angle between the signal and the comparator input varies between 0° and 180°, and is 90° at the center frequency. Fig. (a) shows the typical, triangular, phase-to-output response characteristic of phase-comparator I. Typical waveforms for a COS/MOS phase-locked-loop employing phase comparator I in locked condition of f_0 is shown in fig. (b).

Phase-comparator II is an edge-controlled digital memory network. It consists of four flip-flop stages, control gating, and a three-stage output-circuit comprising p- and n- type drivers having a common output node. When the p-MOS or n-MOS drivers are ON they pull the output up to VDD or down to VSS, respectively. This type of phase comparator acts only on the positive edges of the signal and comparator inputs. The duty cycles of the signal and comparator inputs are not important since positive transitions control the PLL system utilizing this type of comparator. If the signal input frequency is higher than the comparatorinput frequency, the p-type output driver is maintained ON most of the time, and both the n- and p- drivers OFF (3 state) the remainder of the time. If the signal input frequency is lower than the comparator input frequency, the n-type output driver is maintained ON most of the time, and both the n- and p- drivers OFF (3 state) the remainder of the time. If the signal and comparator input frequencies are the same, but the signal input lags the comparator input in phase, the n-type output driver is maintained ON for a time corresponding to the phase difference. If the signal and comparator input frequencies are the same, but the comparator input lags the signal in phase, the p-type output driver is maintained ON for a time corresponding to the phase difference. Subsequently, the capacitor voltage of the low pass filter connected to this phase comparator is adjusted until the signal and comparator inputs are equal in both phase and frequency. At this stable point both p- and n- type output drivers remain OFF and thus the phase comparator output becomes an open circuit and holds the voltage on the capacitor of the low pass filter constant. Moreover the signal at the 'phase pulses' output is a high level which can be used for indicating a locked condition. Thus, for phase comparator II, no phase difference exists between signal and comparator input over the full VCO frequency range. Moreover, the power dissipation due to the low pass filter is reduced when this type of phase comparator is used because both the p- and n- type output drivers are OFF for most of the signal input cycle. It should be noted that the PLL lock range for this type of phase comparator is equal to the capture range, independent of the low pass filter. With no signal present at the signal input,

the VCO is adjusted to its lowest frequency for phase comparator II. Fig. (c) shows typical waveforms for a COS/MOS PLL employing phase comparator II in a locked condition.

Figure a : Phase-Comparator I Characteristics at Low-Pass Filter Output.

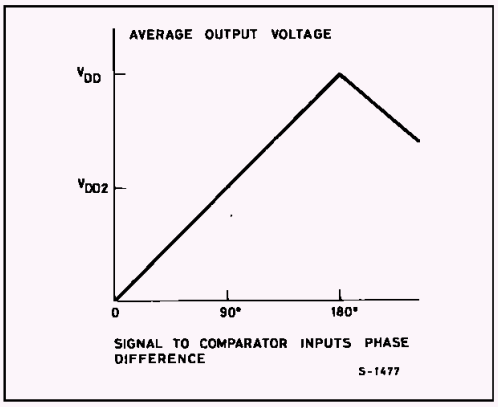


Figure b : Typical Waveforms for COS/MOS Phase Locked-Loop Employing Phase Comparator I in Locked Condition of f_0.

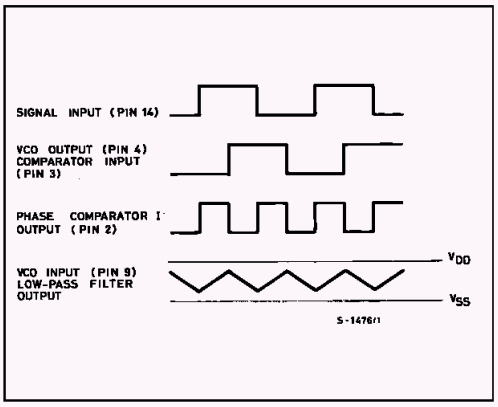
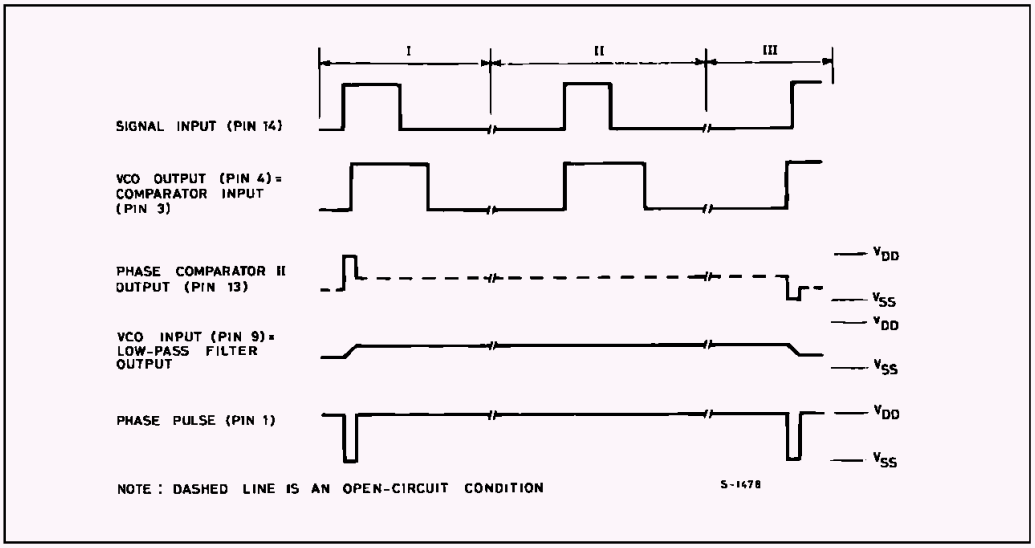


Figure C : Typical Waveforms For COS/MOS Phase-locked Loop Employing Phase Comparator II In Locked Condition.



Man sehe sich auch die 'DSPLL' ICs Si570/Si571 und Verwandte an.

DDS

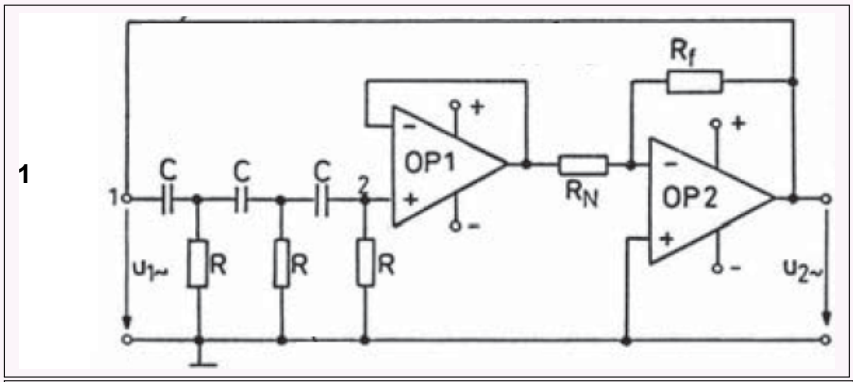
2 .14 DDS Direct Digital Synthesis, digitale Signalsynthese

DDS erzeugt beliebige (arbitrary) Signalformen durch Analogwandlung gespeicherter Digitalwerte in einem DAC (digital-/analog- converter). Es handelt sich somit nicht um Oszillatoren, sondern um (entsprechend schnelle) Funktionsgeneratoren, die mehreren MHz bis GHz getaktet werden.

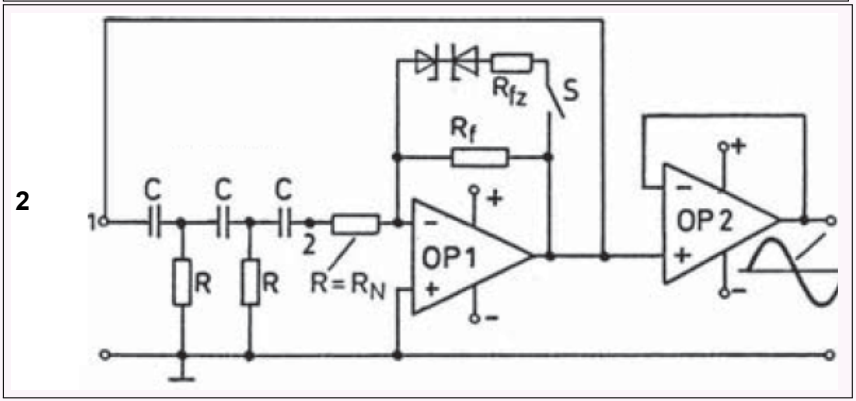
Verbreitete DDS-ICs sind etwa AD9850 (125MHz clock, 10bit DAC), AD9854 (300Mhz clock, 12bit DAC), AD9858 (1Ghz clock, 10bit DAC), AD9859 (400Mhz clock, 10bit DAC), AD9952 (400MHz clock, 14bit DAC)

2 .15 Oszillatorschaltungen JN

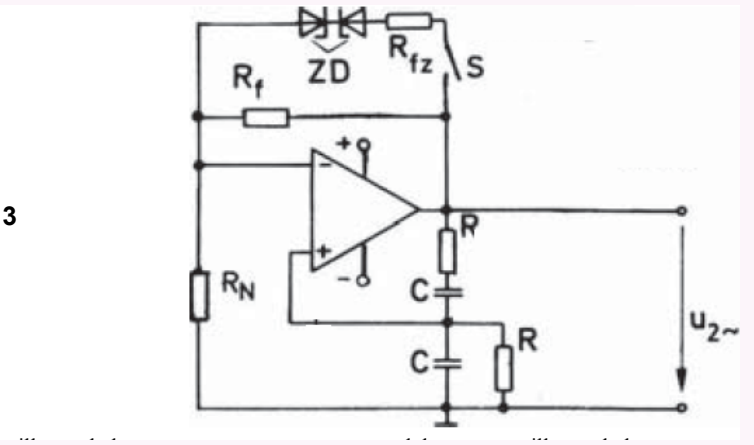
1



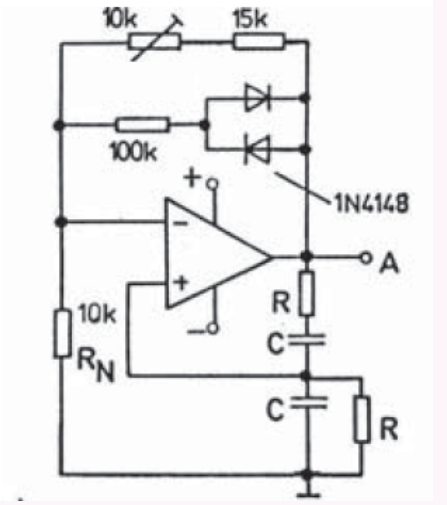
2



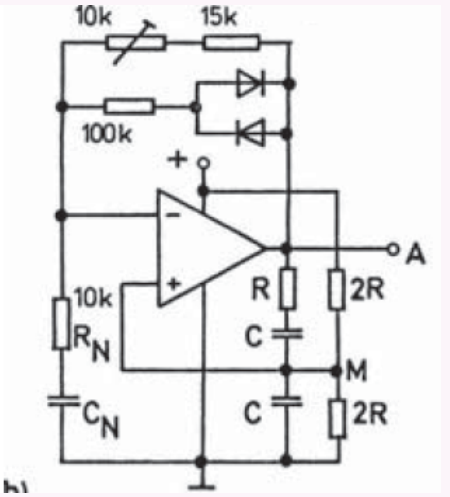
3



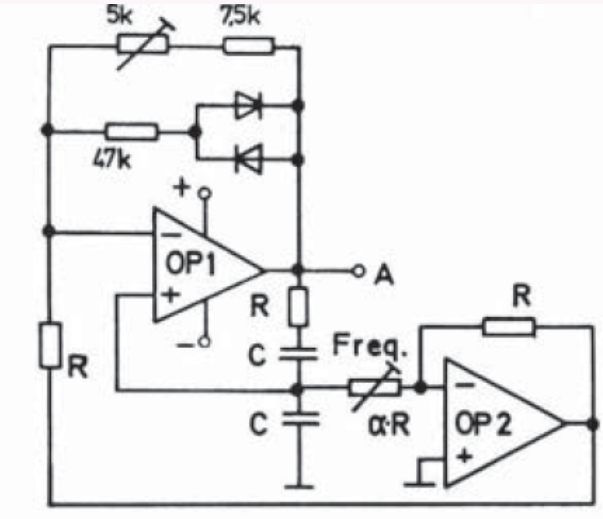
4


10k
15k
100k
1N4148
A
R
C
10k
R_N
C
R

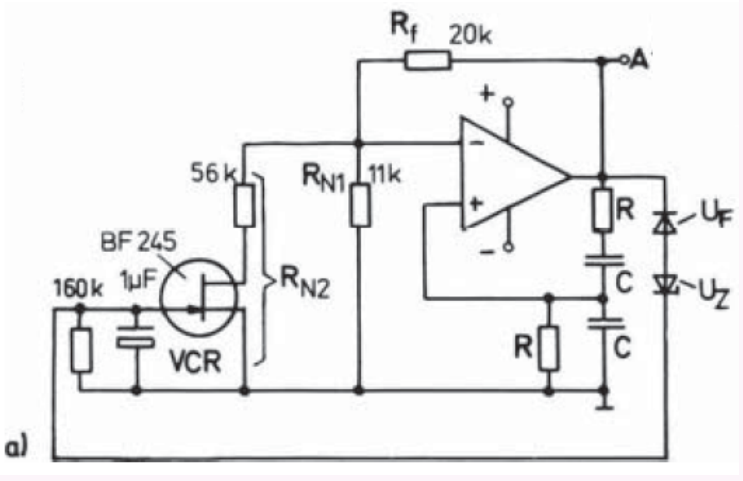
5


10k
15k
100k
A
R
2R
10k
R_N
C
M
C_N
C
2R

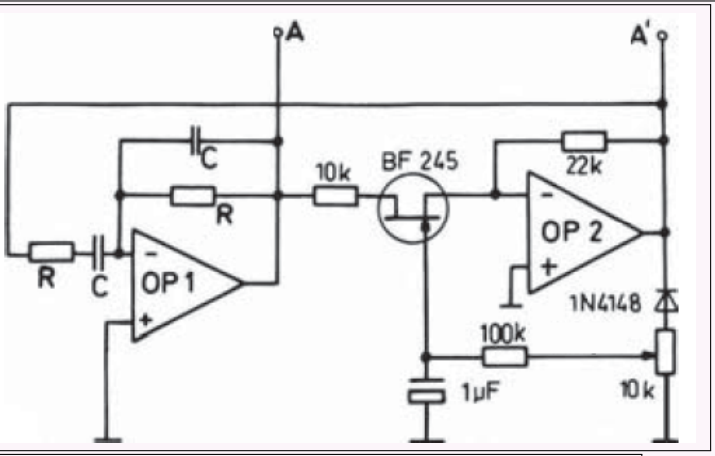
6


5k
7,5k
47k
OP1
A
R
R
C
Freq.
R
C
αR
OP2

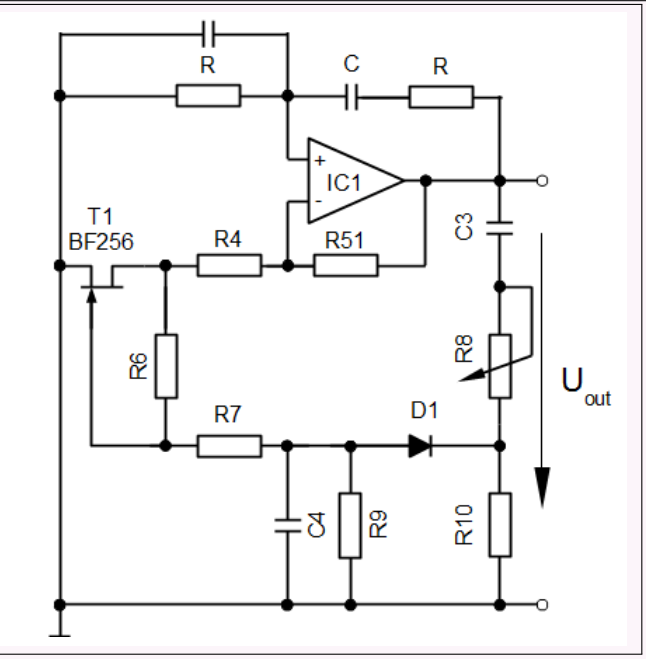
7


Rf 20k
A
56k
RN1
11k
R
UF
BF 245
160k
1µF
RN2
C
UZ
VCR
R
C
a)

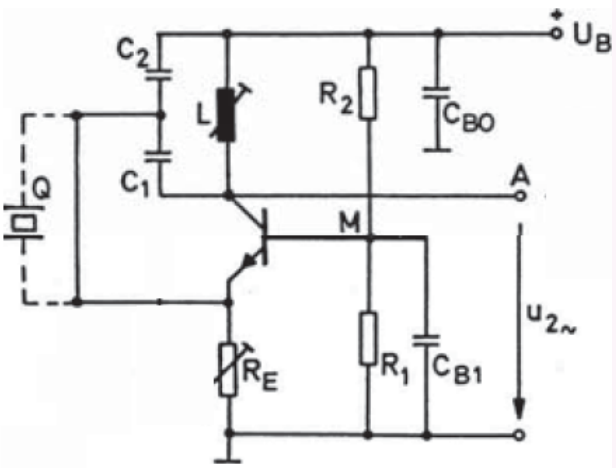
8


A
A'
C
10k
BF 245
22k
R
OP 2
R
C
OP 1
1N4148
100k
1µF
10k

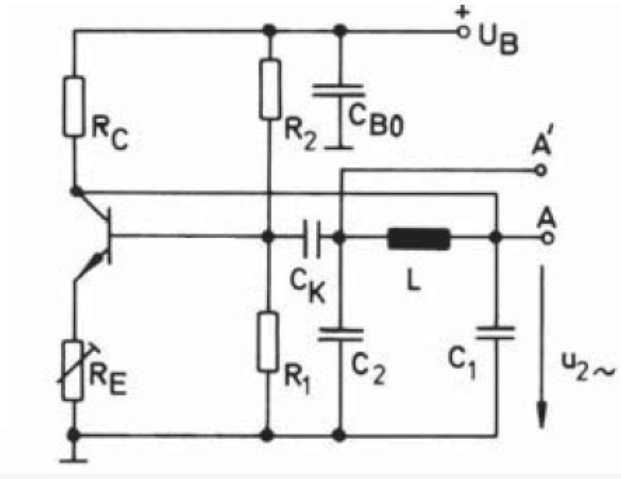
9


R
C
R
IC1
T1
BF256
R4
R51
C3
R6
R8
U out
R7
D1
C4
R9
R10

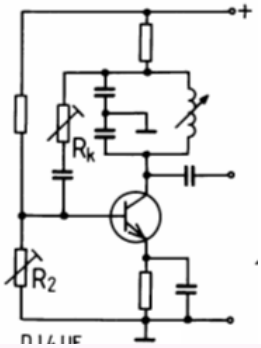
10



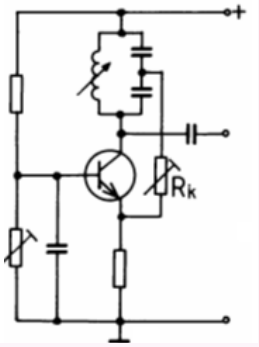
11



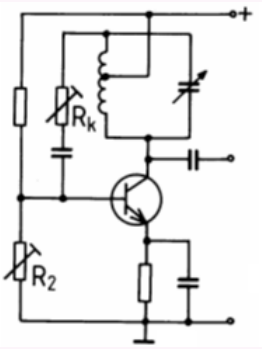
12



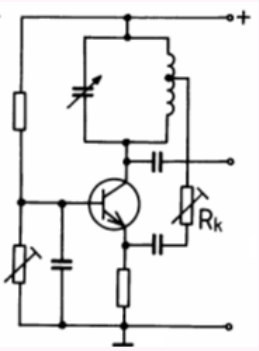
13



14



15



Quellenangabe:

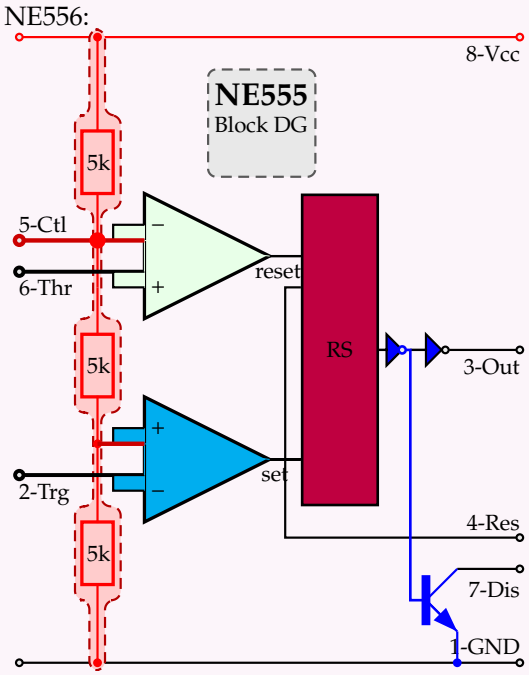
Schaltungen 1 bis 8	Böhmer, Elemente der angewandten Elektronik
Schaltung 9	Jahn, HTL Anichstraße
Schaltungen 10, 11	Böhmer, Elemente der angewandten Elektronik
Schaltungen 12 – 15	Deutscher Amateur Radio Club, www.darc.de

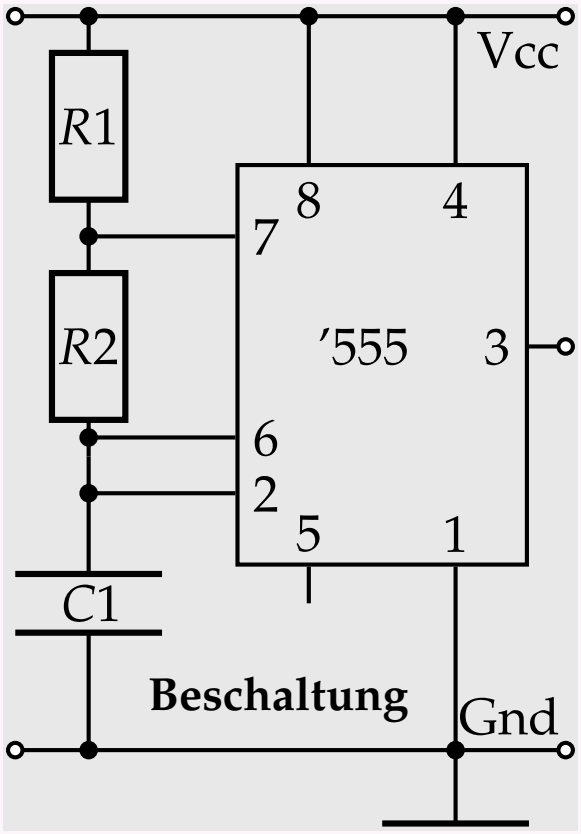
3 NE555

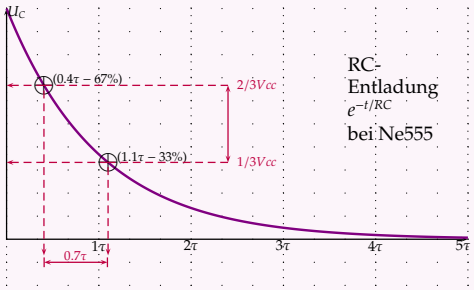
- ☐ besorge Dir ein NE555- Datenblatt mit
  - Pinbelegung,
  - Blockdiagramm (BD) und
  - Innenschaltplan
- ☐ Lerne die Pinbelegung auswendig!
- ☐ Lerne das Innen- Blockdiagramm (BD) auswendig
- ☐ identifiziere die Blöcke im Innenschaltplan
- ☐ V_{CC} Grenzen?
- ☐ Strombedarf?
- ☐ Wie erzeugt man Pulse?
- ☐ Wie erzeugt man 50/50 Squarewave?
- ☐ wie erzeugt man Rampen-Signale (ramp)?
- ☐ wie kann man FM modulieren?
- ☐ wie kann man PWM modulieren?

3 .0.1 Pinbelegungen

NE555:
1: GND
2: 'trigger' compare in = lower limit
3: square out
4: -Reset in
5: $V_{threshold}$ f. Modulation (2/3 Vcc Node)
6: 'threshold' compare in = upper limit
7: 'discharge' BJT Collector out
8: Vcc: ϵ 4,5 .. 16 V_{DC}







→RC-Entladekurve

Die Kondensator-**Ent**ladung eines RC-Gliedes folgt

$\boxed{U_C(t) = U_{C,max} * e^{-t/\tau}}$ und die **Auf**ladung $\boxed{U_C(t) = U_{C,max} * (1 - e^{-t/\tau})}$

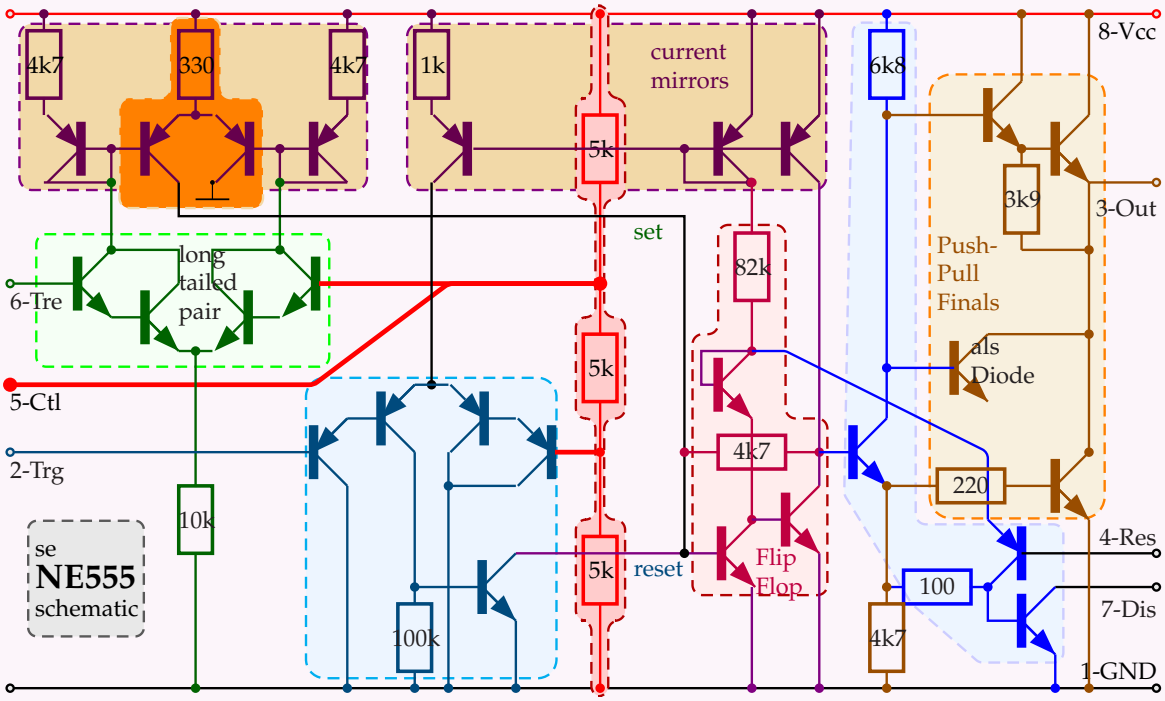
mit $\tau = RC$.

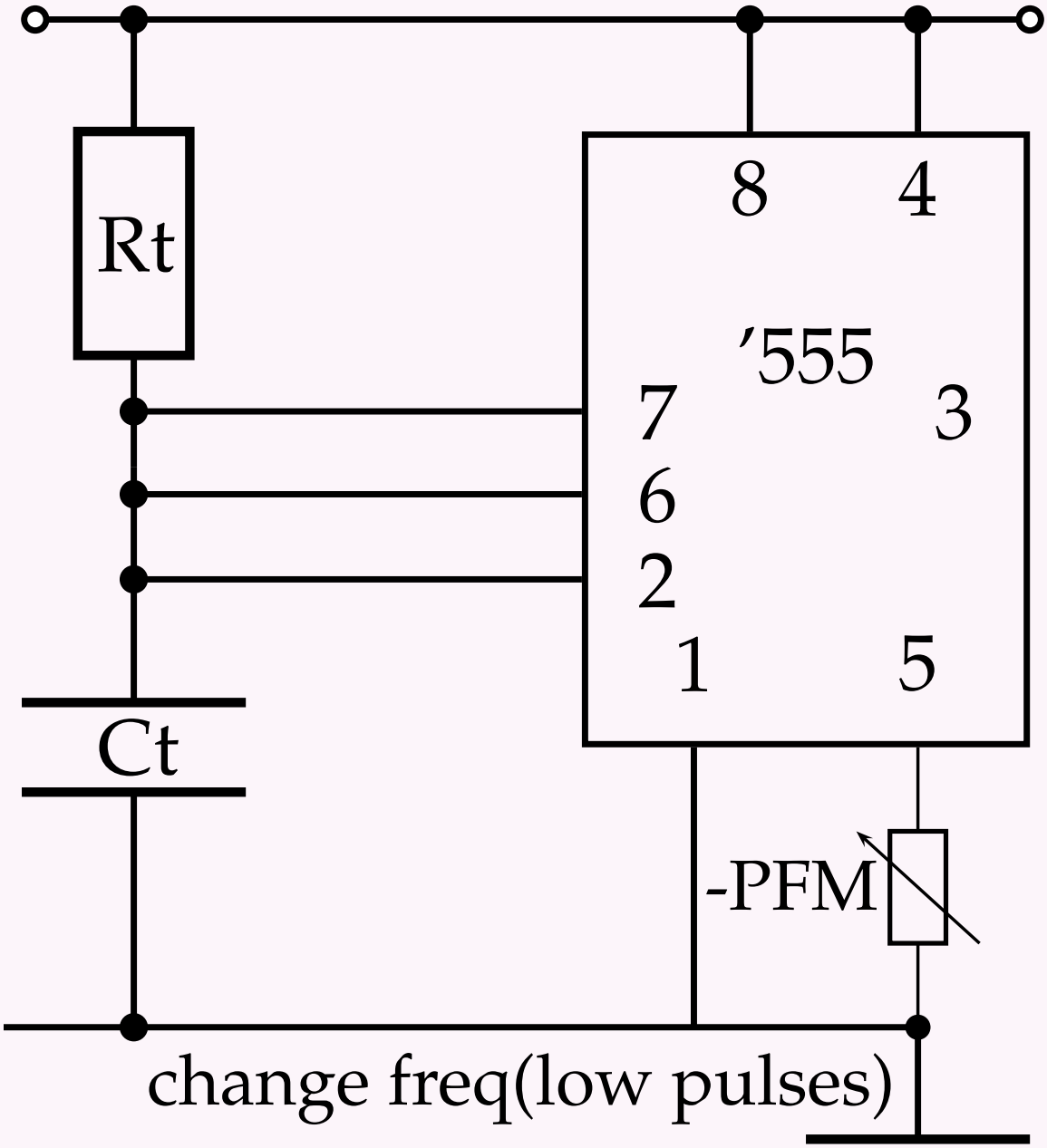
Die oft gesuchte Umkehrfunktion ist

$\boxed{t/\tau = -ln(\, U_C(t)/U_{C,max}\,)}$ bzw. in Sekunden: $\frac{t}{[s]} = -RC * ln(\,\frac{U_C(t)}{U_{C,max}}\,)$

und falls Du keine "ln()" Funktion hast: $\frac{t}{[s]} = -\frac{RC}{log(e)} log(\,\frac{U_C(t)}{U_{C,max}}\,)$

Wir normieren $U_{C,max} = 1$ (zB. $U_{C,max} = 1.0V$ Kondensator-Entlade-Anfangsspannung)
Für andere Werte ist die Kurve in "y-Richtung" entsprechend zu skalieren (="zoomen")
Wegen der einfacheren Formel ($e^{-t/\tau}$) merken wir uns die ENTladung:

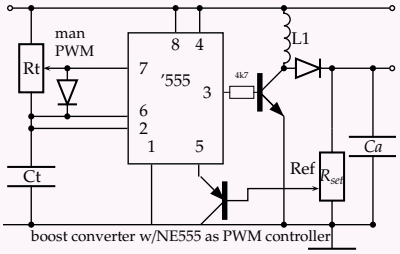

4k7
330
4k7
1k
current
mirrors
5k
6k8
8-Vcc
3k9
3-Out
Push-
Pull
Finals
als
Diode
set
82k
long
tailed
pair
6-Tre
5-Ctl
2-Trg
5k
4k7
220
4-Res
10k
5k
reset
Flip
Flop
100
7-Dis
4k7
100k
1-GND
se
NE555
schematic


Rt
8
4
'555
7
3
6
2
1
5
Ct
-PFM
change freq(low pulses)

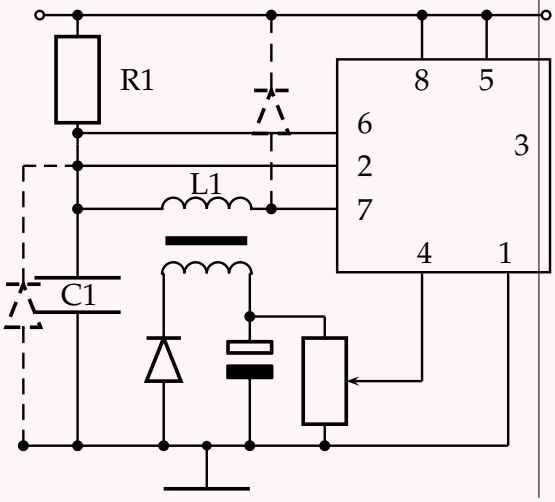
3 .0.2 PWM mit NE555

Wie wir alle auswendig wissen, sind am NE555

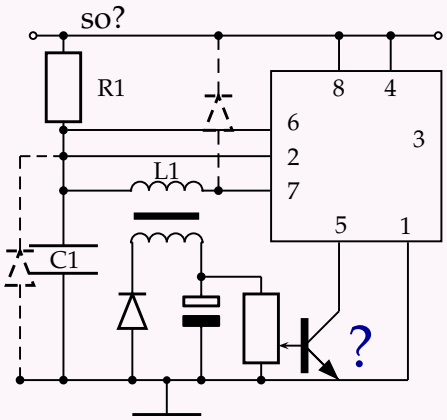
1: GND
2: 'trigger' compare in = lower limit
3: square out
4: -Reset in
5: $V_{threshold}$ f. Modulation (2/3 Vcc Node)
6: 'threshold' compare in = upper limit
7: 'discharge' BJT Collector out
8: Vcc 4,5..16 V_{DC}

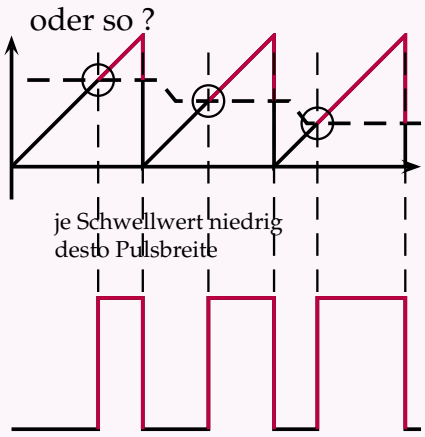


3 .0.3 SMPS mit NE555



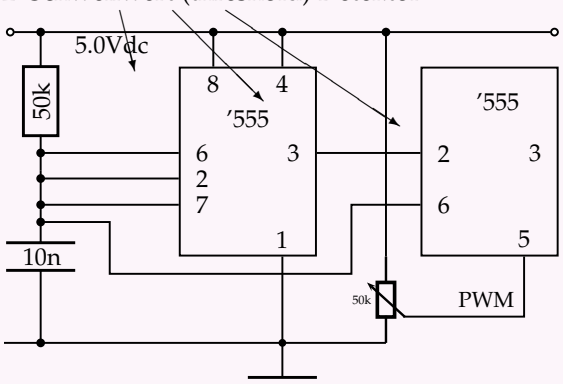
3 .0.4 Sonderbares mit NE555



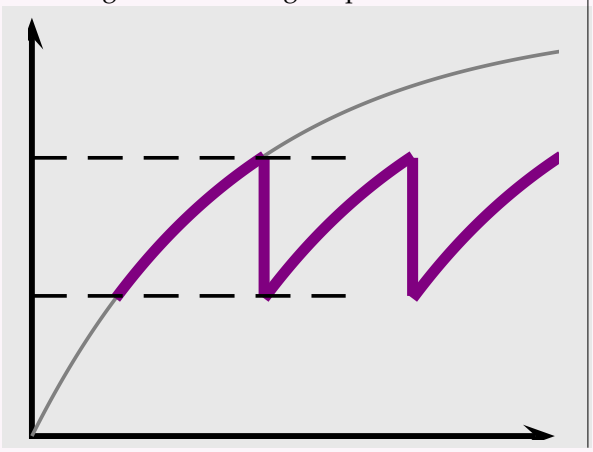


3 .0.5 Fixfrequenz-PWM mit 2xNE555 = NE556

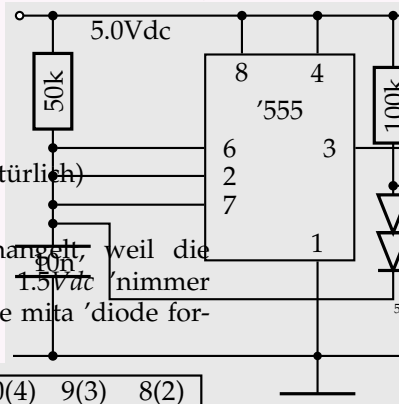
PWM fixer Frequenz erzeugt man aus
1 Ramp Signal (oder Triangle)
plus
2 Schwellwert (threshold) Detektor



Habe folgende Schaltung ausprobiert:



Die Pulsbreite am Ausgang (3) lasstsi vu ca. 40 bis 100% einstellen. Die 'ramp' isch freilich ka schiane /|, sondern der Ausschnitt der $1 - e^{-t/\tau}$ Funktion von 33% bis 67%, aber des is wurscht (weil a Regelkreis die Regelspannung eh einfach soweit aufi draht, bis passt; die Signalform muass nur immer genau gleich sein — und monoton natürlich)

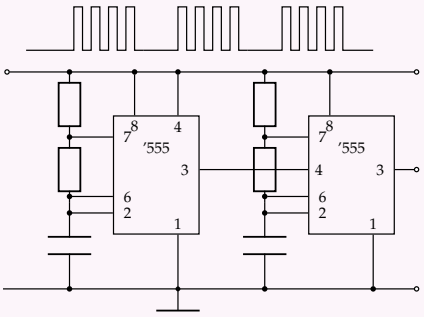


Varbessarig:
DutyCycle (Tastverhältnis) mangelt, weil die 555'er ba *ControlVoltage*(5) < ~~1.5Vdc~~ 'nimmer kennen'. Ez schifti des Triangle mita 'diode forward voltage drop' aui:

14(8)	13(7)	12(6)	11(5)	10(4)	9(3)	8(2)
Vcc	Dis	Thres	Ctl	-Rst	Out	Trig
			NE-556			
Dis	Thr	Ctl	-Rst	Out	Trig	Vss
1(7)	2(6)	3(5)	4(4)	5(3)	6(2)	7(1)

Freilich fälschen de 100k es Timing und verhunzen die Rampenform weiter — isma wurscht.

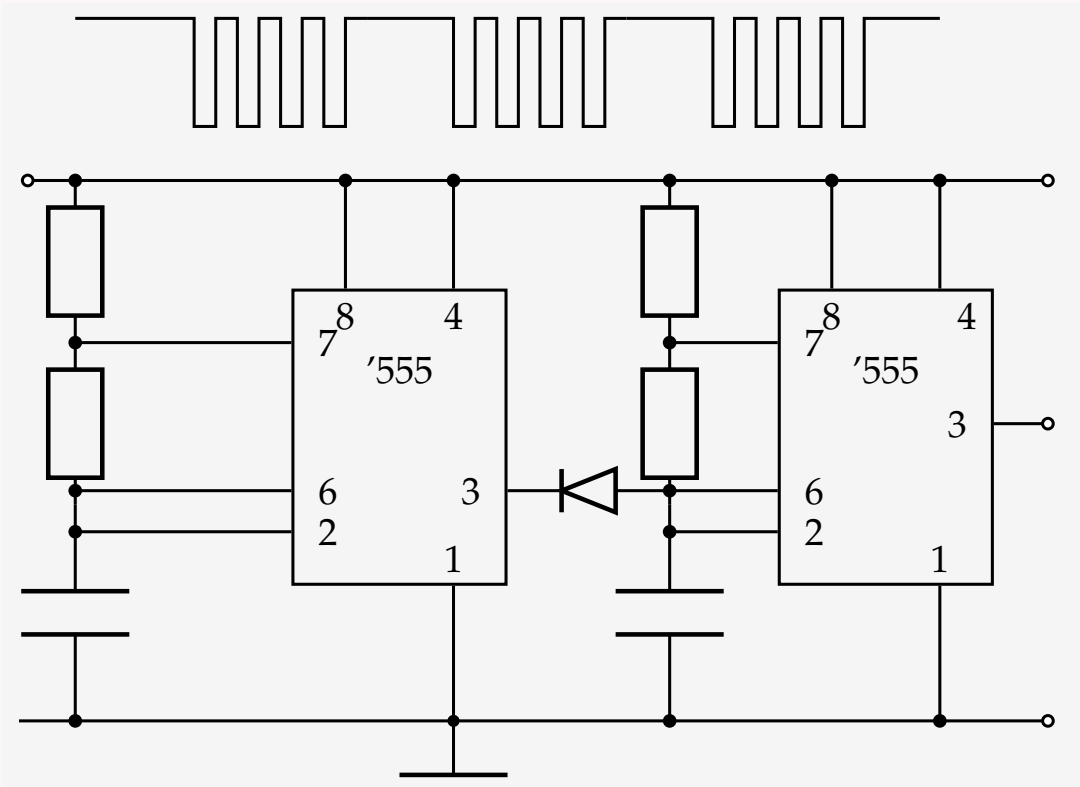
In Wahrheit hunin '556 nummen,
≡ 2 x '555



3 .1 NE555 + 'burst' Signale

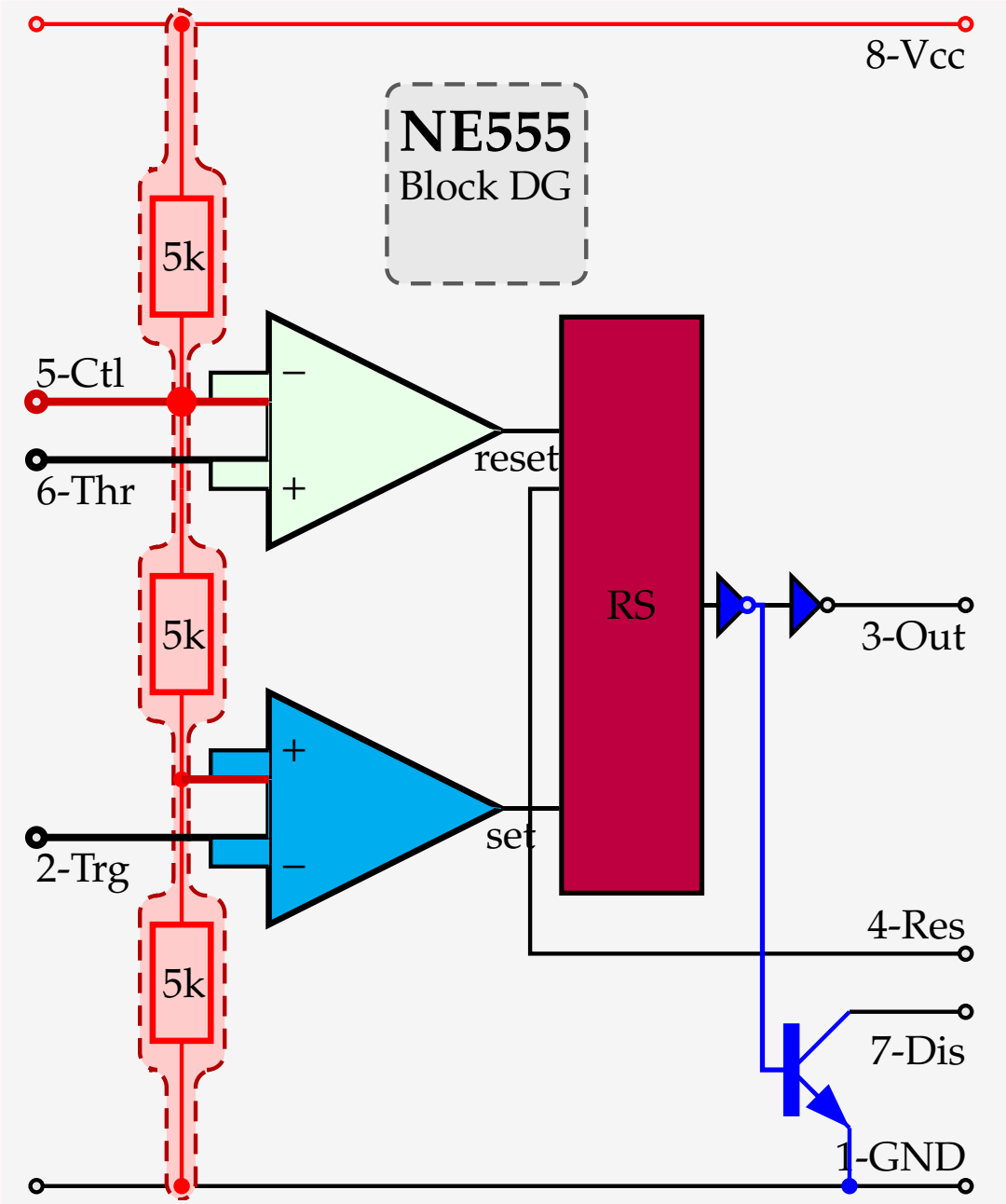
3 .1.1 Burstgenerator mit NE555/NE556

Variante-A: Gap Low



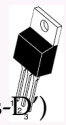
Der 555-links liefert ein langsames Rechteck, mit dem er den 555-rechts am Reset-Eingang (4) und damit die schnellen Pulse blockiert.
Das Tastverhältnis am Ausgang (3) des 555-links steuert Pulsanzahl und Pausenlänge.

Variante-B: Gap High



Der 555-links liefert ein langsames Rechteck, mit dem er die Aufladephase des 555-rechts über die Diode verlängert und damit die schnellen Pulse blockiert (Eingang (2) überschreibt (6)).
Das Tastverhältnis am Ausgang (3) des 555-links steuert wiederum Pulsanzahl und Pausenlänge.

4 SMPS + PWM + PwrMOS + NE555

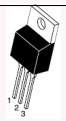



Wir bauen Schaltwandler für
1) DC-Motor Ansteuerung
2) LED Betrieb
3) Stromversorgung (Lastwiderstand)
4) digital-Audio-Leistungs-Verstärker ('class-D')

nach den
'buck-'
'boost-'
'buck/boost inverting'
'flyback-' (Sperrwandler)
-'converter' Konzepten

Ferrit-Kerne sind **nur** für *forward converter* geeignet

Energiespeicherung erfordert Eisenpulver-Kern oder Luftspalt

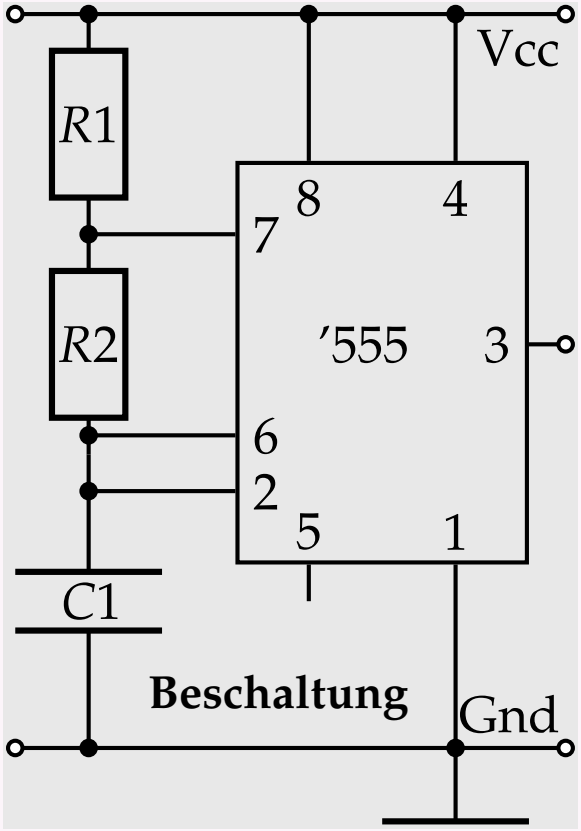



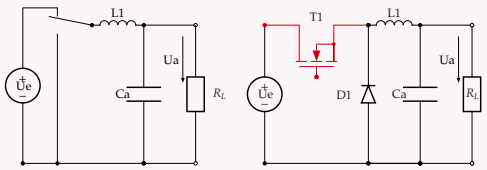
und testen sie mit dem Labornetzteil
als Primärstromversorgung variabler Spannung.

`Spannungen/Spitzen über 50.0 V dürfen keinesfalls auftreten!`

4 .0.1 Aufgabe-1:NE555

→ s. Kap.3 , S.105





4 .0.2 Aufgabe-2: Ferrit-Ringkern

Besorg Dir einen **Entstör-Ferrit-Ringkern**.
Die sind leicht als EMV- (EMI-) Entstörkerne zu *recyceln* zB. aus defekten (PC-) Schaltnetzteilen, Videosignalkabeln.

Ferrithülse ferrite bead/core, Ringkern toroid, Doppellochkern (*'Schweinenase'*)

iXH bringa 150uH Drossel auf Eisenpulverkern (zum Vergleichen):

→ Mit **Ferrit**kernen (ohne Luftspalt) macht man **Trafos** (Übertrager) und Entstördrosseln (Störenergieverheizer).

→ Mit **Eisenpulver** (oder mit Luftspalt) macht man **Speicher**drosseln und Schwingkreise. (Eisenpulverkerne sind Keramik mit Eisenpulver drin. Man nennt das auch *fein verteilter Luftspalt*)

4 .0.3 Aufgabe-3: PWM/PFM

4 .0.4 Aufgabe-3a: Erzeuge Rampensignale

4 .0.5 Aufgabe-3b: Erzeuge PWM

4 .0.6 Aufgabe-3c: Ansteuerung PwrMosFET

- Studiere die Möglichkeiten und Varianten zum Erzeugen von PWM- und PFM- Signalen mit einem (nötigenfalls zwei) NE555.
  idealerweise lässt sich das PWM- Tastverhältnis (duty cycle) $p = \frac{t_{ein}}{t_{ein} + t_{aus}}$ mittels einer Steuerspannung von 0..1 (0..100%) variieren. Oft ist es an den Grenzen nahe 0 und nahe 1 problematisch zu realisieren, weswegen die *'wichtigere'* bevorzugt realisiert wird.
- Betrachte die Modulation der PWM/PFM
  - sowohl mechanisch mittels Trimmpotentiometer
  - als auch mittels elektrischer Spannung
    (zB. via Input an Pin'5' ?).
- Zu den Prinzipien siehe auch Kap.4 , S.116

Legende:

PFM Puls-Frequenz Modulation (pulse frequency modulation). Die Frequenz (Anzahl/Sek.) der Pulse variiert analog zum Modulationssignal.

PWM Puls-Breiten Modulation (pulse width modulation). Die Breite (Dauer, Weite) der Pulse variiert analog zum Modulationssignal.

PPM Puls-Positions Modulation. Die Position (Ort, Phasenlage) der Pulse variiert analog zum Modulationssignal.

schrittweise Anleitung

1. erzeuge mit NE555 eine per Trimmpoti verstellbare PWM
2. mach dabei die Frequenz unbeeinflusst.
3. stell den PWM- Variationsbereich fest.
4. mach die PWM spannungssteuerbar, statt per Trimmpoti
5. stell wieder den PWM- Variationsbereich fest.
6. mach das Tastverhältnis bis 0 'herunter' steuerbar
7. sichere das PwrMosFET-Gate gegen $V_{GS} > 18V$
8. montiere den PwrMosFET auf ein Kühlprofil
9. speise das PWM- Signal (über $R_G \approx 5k$) an das PwrMosFET- Gate:

```
RG       V_PWM          Gate1    5k
RD       Vcc            Drain1   5k
MosFET1  Drain1  Gate1  Source1  einNchannelMosFET
VSrc1    Source1        0        DC 0.0
```

   (SPICE netlist)
10. bestimme die vorgesehenen Betriebsbedingungen für den PwrMosFET.
11. bestimme $t_{d,on}, t_{rise}, t_{fall}$ und $t_{d,off}$ am PwrMosFET unter obigen Betriebsbedingungen.
12. bestimme die maximal sinnvolle Betriebsfrequenz der Anordnung

Jeder Denk- und Arbeitsfortgang ist akribisch zu protokollieren.
Alles andere als Denk- und Arbeitsfortgänge hat zu unterbleiben.

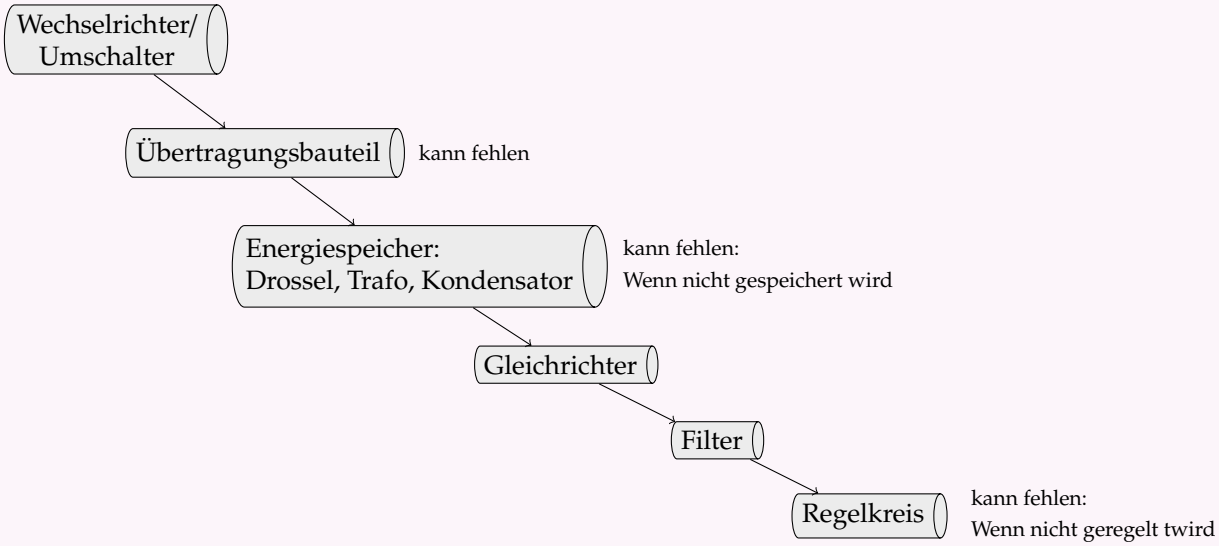
Das Projekt ist **matura**vorbereitend — wer sich nicht ausreichend einarbeitet, könnte bei einer **qualitätssichernden Überprüfung** des Unterrichtserfolges mit Mängeln konfrontiert sein.

4 .1 SMPS switched mode power supply, Schaltwandler, Schaltnetzteil

Schaltende Spannungs- und Stromkonverter übertragen und/oder speichern+wiedergeben die Energie, statt sie zu *verheizen*. Sie schalten die Energiezufuhr periodisch ein (t_{on}, t_{ein}) und aus (t_{off}, t_{aus})

$$t_{on} + t_{off} =:: \tau \quad \rightarrow \quad f ::= \frac{1}{t_{on} + t_{off}}$$

Sie bestehen aus



Man sagt auch *DC-DC-Schaltwandler (switching mode converter / power supply, SMPS)* und baut sie mit

Kondensatoren	→	Ladungspumpe (charge pump)
Drosselspulen	→	Aufwärts-, Abwärts- und Invers-Wandler (buck-, boost- and inverting converter; Drossel = engl. choke, choking coil, Spule = coil, inductor, solenoid).
Transformatoren	→	Sperr-, Durchfluss-, Gegentakt-Wandler
Sonderfall: Multi-Phase-Converter	→	sind mehrfache, zeitversetzt arbeitende Wandler obiger Bauarten.

dabei kann die Energie

- zeitgleich übertragen werden (Transformator)
  Durchfluss-, Gegentakt- u. Abwärtswandler
  (forward mode converter, push-pull converter, buck switcher)
  → Kern wird weniger magnetisiert → große Leistungen
- zwischen**gespeichert** werden (Spule, Drossel)
  Sperr-, Aufwärts-, Inverswandler
  (flyback mode converter, buck switcher, inverting converter)
  → Kern wird magnetisiert; Achtung: Mag.Sättigung!

Wir nennen

deutsch	Formelzeichen	symbol	english
Einschaltzeit der Energiezufuhr	t_{ein}	t_{on}	on time
Ausschaltzeit der Energiezufuhr	t_{aus}	t_{off}	off time
Periodendauer	$\tau = t_{ein} + t_{aus}$	T=$t_{on} + t_{off}$	period
Tastverhältnis	$p = \frac{t_{ein}}{\tau}$	$\delta = \frac{t_{on}}{T}$	duty cycle
Anstiegszeit der Energiezufuhr	t_{an}	t_r	rise time
Abfallzeit der Energiezufuhr	t_{ab}	t_f	fall time
Restwelligkeit	ΔU_a	ΔV_{out}	voltage ripple
Strom-Welligkeit	ΔI_L	ΔI_L	inductor ripple current

Bei den zwischenspeichernden Betriebsarten (energy storing mode) rechnet man vorteilhaft mit sog. ***Spannungs-Zeit-Flächen***

$$U_{hinein} \bullet t_{ein} = U_{heraus} \bullet t_{aus}$$

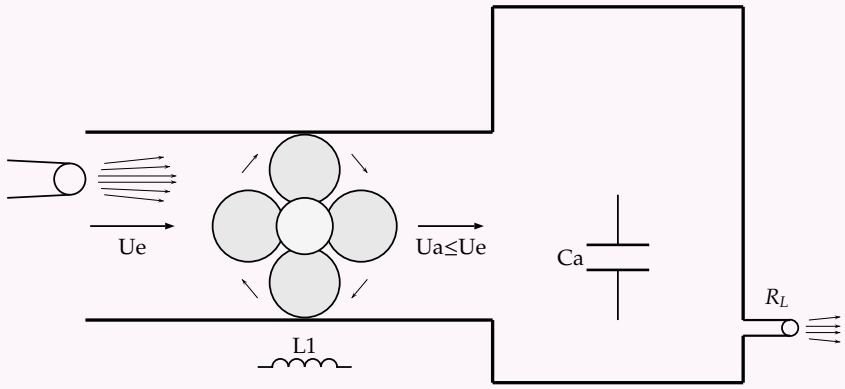
U_{hinein}, U_{heraus} : direkt an der Spule!

(das ergibt sich aus

$$\rightarrow \begin{matrix} U_{ind} = L \bullet \frac{\partial I}{\partial t} \\ U_{ind} \bullet \Delta t = L \bullet \Delta I \end{matrix}$$

unter stationären (=konstantbleibenden) Verhältnissen **und** gegenüber den Gleichstromgrößen *kleinen* Änderungen ΔI, ΔU)

4 .1.1 Prinzip Abwärtswandler (buck converter):



t_{ein}**-Stromfluss:**
Ue → T1 → L1 → Ca(I_C) + R_L(I_R)
I_L *steigt* um ΔI_L an.

t_{aus}**-Stromfluss:**
GND → D1 → L1 → Ca(I_C) + R_L(I_R)
I_L *sinkt* um $-\Delta I_L$ wieder ab.

Dimensionierung

$$\frac{U_a}{U_e} = \frac{t_{ein}}{t_{gesamt}} = p$$

$\frac{1}{2}\Delta I_L < I_a$ (mindest-Strom!)

Vereinfachung: $I_a \rightarrow R_L$, $\Delta I_L \rightarrow C$

$$\text{mittl.Ladestrom } \overline{I_C} = \frac{\Delta I_L}{2} = C\frac{+\Delta U_C}{(t_{ein} + t_{aus})/2}$$

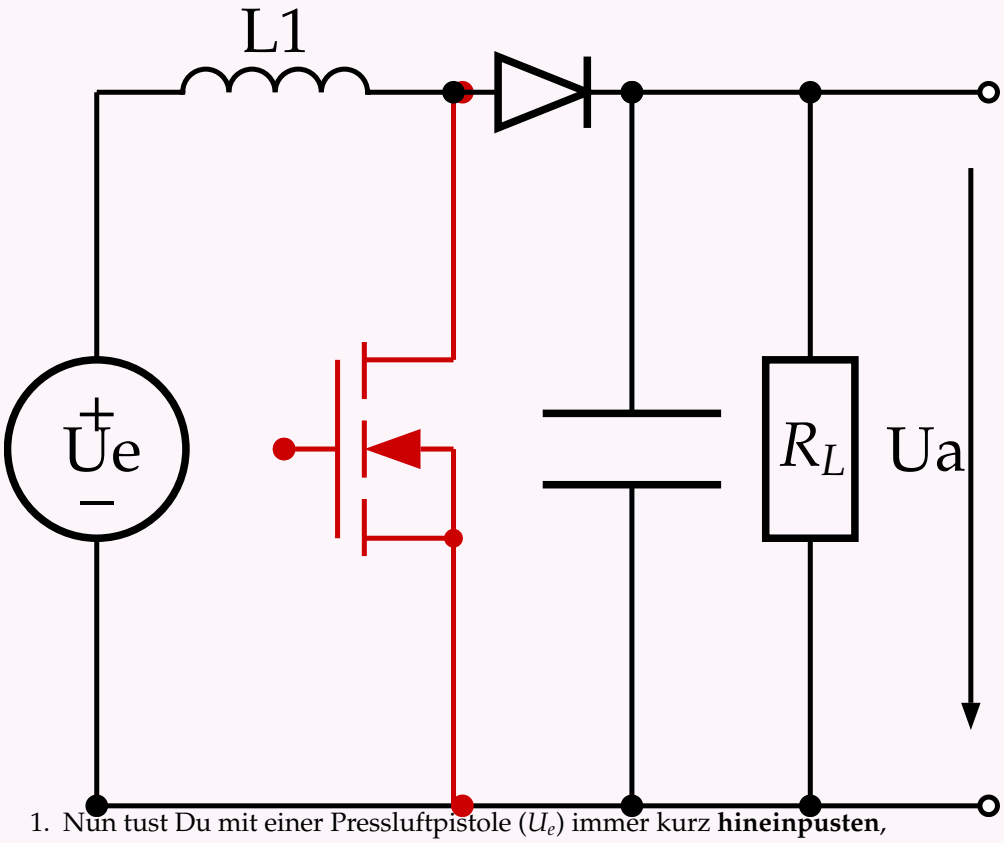
$$\Delta I_L = \frac{4C\Delta U_a}{t_{gesamt}} = 4C\,\Delta U_a\, fs$$ (fs: Schaltfreq.)

Die **Spule** wird von (oft großem) **Gleichstrom** durchflossen,

- dies führt zu kräftiger **Vormagnetisierung** (→ Verluste, Sättigung)
- Der Ferritkern wird deshalb mit einem →**Luftspalt** (typ. 0.5 .. 1.5mm) versehen *"Scherung"* oder ein **Eisenpulver-Kern** (***"verteilter Luftspalt"***) verwendet.

4 .1.2 Gleichnis:

Stell Dir vor, L sei ein Rohr mit einem L=**Lüfter** drin.

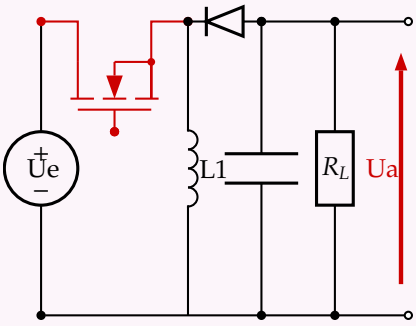


1. Nun tust Du mit einer Pressluftpistole (U_e) immer kurz **hineinpusten**,
   damit der Lüfter L Schwung kriegt
   (dieser *Schwung* ist Dein Energiespeicher).
2. In den Puste-Pausen lauft der Lüfter vom Schwung weiter
   und **pumpt** Umgebungsluft (GND) durch.
   Die Schwungmasse wirkt wie L, I wie der Luftstrom, U wie der Luftdruck, B wie die Lüfterdrehzahl.
3. beim Pusten ist der Druck vorn größer,
   in der Pause dafür kleiner.
4. Zum L anschubsen hast Du $Ue - Ua$, pumpen tut L dann alleine gegen Ua

$$(Ue - Ua) * t_{ein} = Ua * t_{aus}$$

$$Ua = Ue * \frac{t_{ein}}{t_{ein} + t_{aus}} = Ue * p$$

4 .1.3 boost converter, Aufwärtswandler:



ignorieren

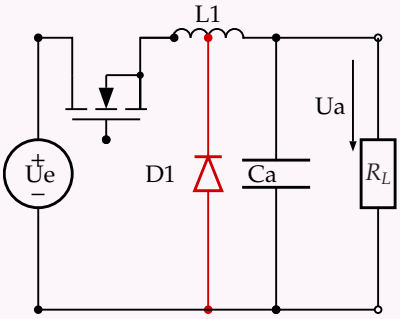
$$t_{ein} * U_e = t_{aus} * (U_a - U_e - U_{D1})$$

$$U_e * (t_{ein} + t_{aus}) = U_a * t_{aus}$$

$$\frac{U_a}{U_e} = \frac{t_{ein} + t_{aus}}{t_{aus}} = \frac{1}{1-p}$$

L und C s. buck converter

4 .1.4 buck/boost inverter, Inverswandler:



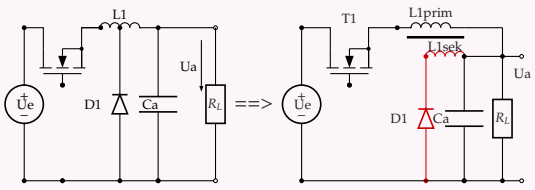
ignorieren

$$t_{ein} * U_e = t_{aus} * (-U_a + U_{D1})$$

$$\frac{U_a}{U_e} = -\frac{t_{ein}}{t_{aus}} = -\frac{\frac{t_{ein}}{t_{ein}+t_{aus}}}{\frac{t_{aus}}{t_{ein}+t_{aus}}} = -\frac{p}{1-p} = \frac{p}{p-1}$$

4 .1.5 Vom buck zum flyback, Sperrwandler:

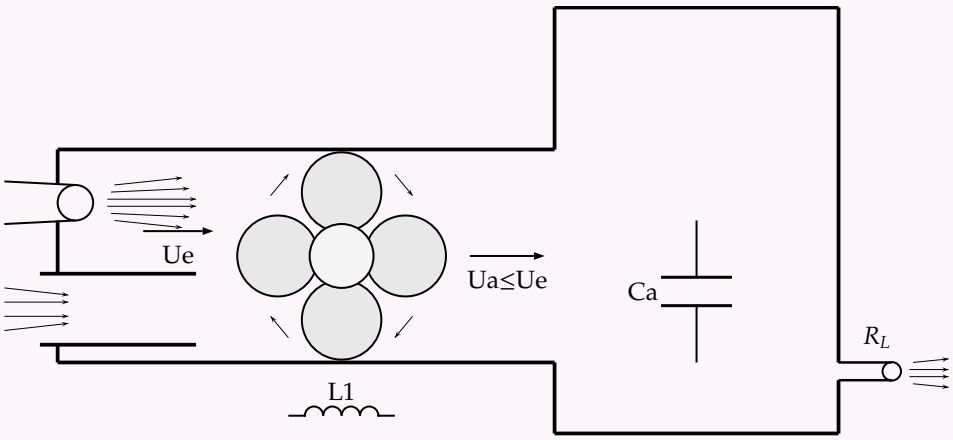
Was, wenn Du zum *Pumpen* nur einen Teil der L-Wicklung, also einen Spar-Transformator verwendest?



→ i sag's Dir:
Die Spannung reduziert sich, im Verhältnis der Windungszahlen (N1, N2).

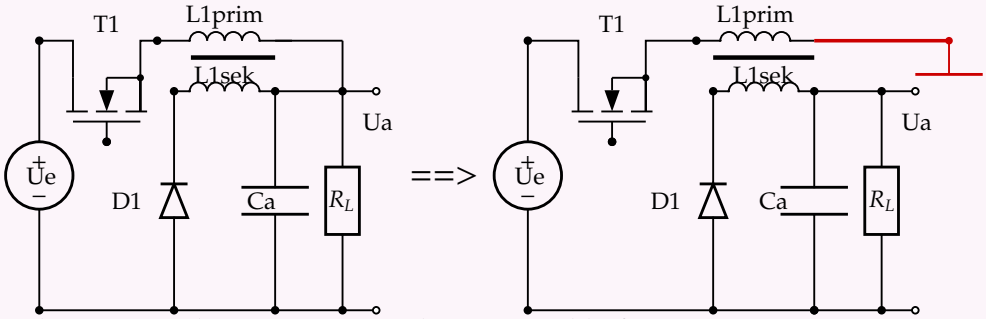
stutzig?
→ bedenke: B steigt mit der *Windungs*spannung $U_{ind}/N = A * dB/dt$. (die Windungszahl steigert B nicht)

. . . und wenn wir eine getrennte Wicklung nehmen?

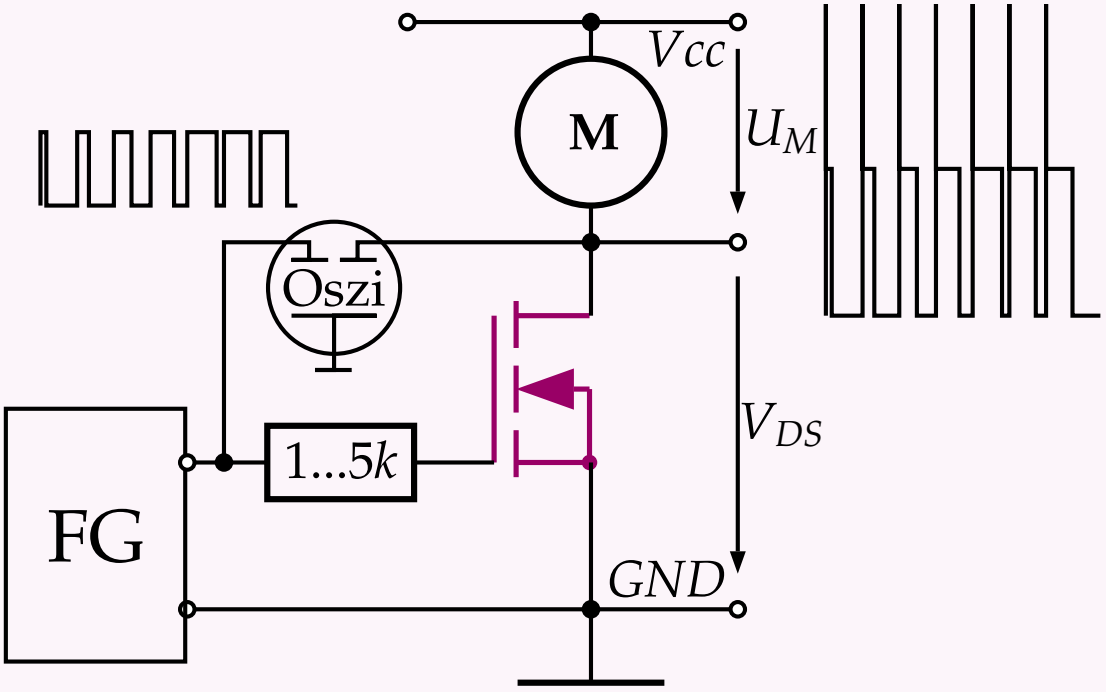


. . . das muss wohl noch *wurscht* sein! (hamma halt den *Ansaugstutzen* extra)

Merke:
es kommt darauf an, was sich **im Magnetkern tut**



und wenn man die Primärseite nach GND anschließt?



1. nicht mehr wurscht!
   → Die primärseitige *anschubs*-Spannung ist jetzt größer (Ue statt Ue-Ua)
   → Der primär-Strom rinnt getrennt ab (nicht mehr zum Speicher- Ca)

4 .1.6 Transformator-Wandler

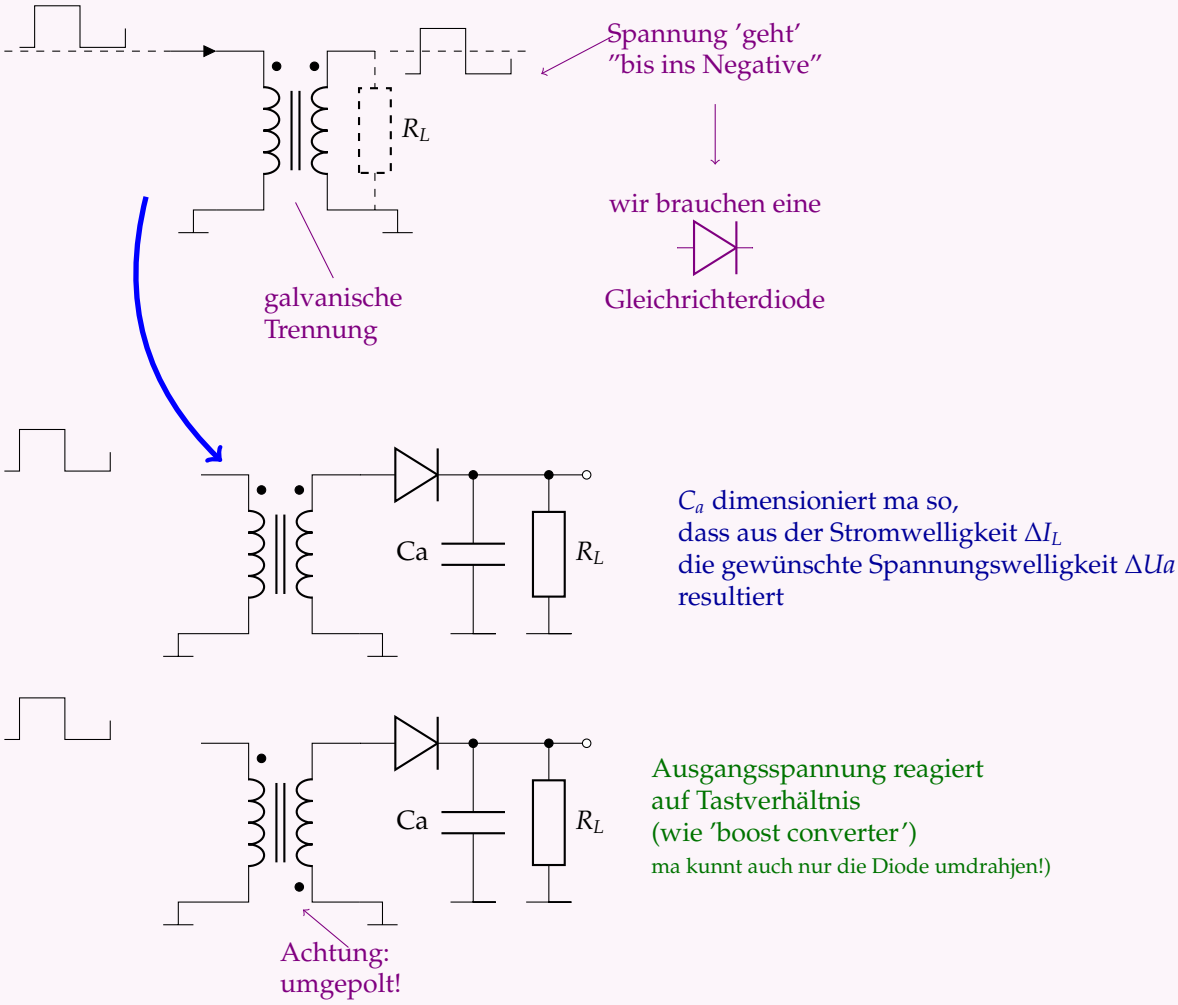
Windung::= 1 einzige Leiter-Windung (one turn of a winding)

Wicklung::= mehrere (durchgehende oder reihengeschaltete) Windungen *(multiple continuos windings form a coil)*

Durchfluss-Wandler (forward converter) übertragen die Energie
OHNE MAGNETISCHE ZWISCHENSPEICHERUNG
direkt gleichzeitig auf die Sekundärseite. Die Ausgangsspannung ist nur mit dem Windungsverhältnis, jedoch **nicht mit dem Tastverhältnis** (duty cycle) steuerbar

Sperr-Wandler (flyback converter) SPEICHERN die primärseitig eingespeiste Energie
IM MAGNETFELD
und liefern diese danach bei Magnetfeldabbau als Induktion zur Sekundärseite (die Induktion wirkt auf alle Windungen aller Wicklungen zugleich).
Die Energieübertragung erfolgt in 2 aufeinander folgenden Schritten (einspeichern: Magnetfeldaufbau, entnehmen: Magnetfeldabbau)

Strom-Welligkeit (current ripple) Die Induktor-Ströme werden (normalerweise) nur geringfügig gesteigert und wieder abgesenkt, und nicht "von 0 auf 100" aufgeschaltet und wieder "auf Null" reduziert (das würde man "trocken gefallene" (run dry) Spulen nennen, respektive "lückenden Betrieb" (discontinuos mode DCM))



Servus Christoph, fürn Sperrwandler brauchn ma a Spule mit 3-Pins soweit i des verstanden hab. Gibts da eine in LTSpice bzw. welche Bezeichnung hat de? Liebe Grüße und schenen Tag nu! (Anna Wundrig)

→ Sperrwandler sind **Trafo**-Wandler und haben pro Wicklung 2 Anschlüsse.

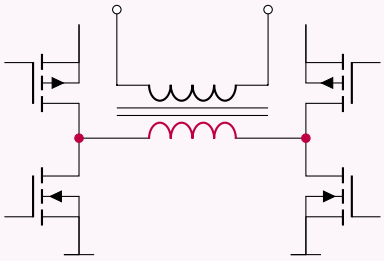
- 1 **Primär**wicklung
- 1 **Sekundär**wicklung
- 1 weitere Wicklung, falls damit eine Gegenkopplungs-**Regelschleife** zu realisieren ist
- evt. 1 weitere Wicklung (meist nur 1 Windung) für **Mess-Zwecke**: $U_{ind} = d(B * A)/dt$
- ggf. Wicklungen für **weitere Ausgangsspannungen**

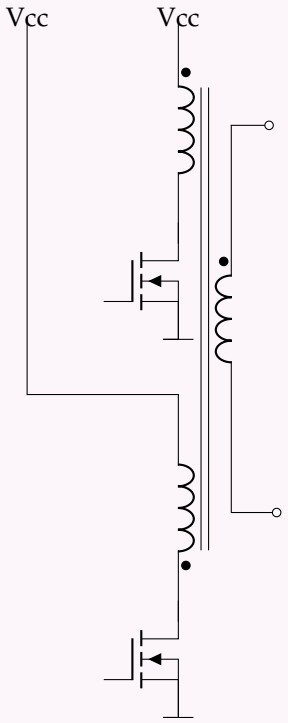
Auch die *Kombination* von Sperr- und Durchfluss-Prinzip ist denkbar: Eine Wicklung liefert in der AUS-Phase und eine in der EIN-Phase (wurde zB. im Bildröhren-Fernseher mit dem *Zeilen-Trafo* so gemacht: Die Durchfluss-Wicklung liefert die Horizontal-Ablenkspannung, die Sperr-wicklung mit dem Rückholimpuls die Hochspannungsimpulse für die Kaskade der Bildröhren-Beschleunigungsspannung)

(Berti Besserwiss)

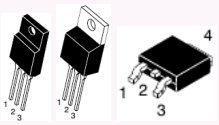
4 .1.7 Gegentakt-Wandler

Gegentakt-Wandler (push-pull converter) sind Durchfluss-Wandler, jedoch werden statt Aufmagnetisierung und Entmagnetisierung aufeinanderfolgende Um-Magnetisierungen vorgenommen - der Betrieb entspricht dem 'gewöhnlichen' Netztransformator, aber mit wesentlich höherer Betriebsfrequenz und folglich kleineren Kernvolumina. Die Ansteuerung erfolgt mit 'H'-Brückenschaltungen bzw. über getrennte Primärwicklungen.





4 .2 Handhabung von Power-HexMosFET



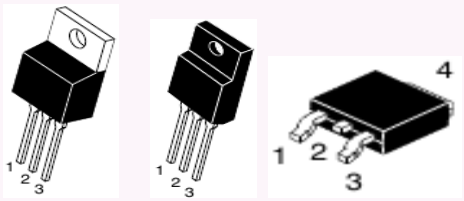
Unsere Impuls-Leistungs-MosFET (*'PwrMosFET'*) sind sog. selbstsperrende Anreicherungs- (enhancemend-) DMOS-HexFET (Weiterentwicklung aus dem VMOS); das *'Hex'* ist von der hexagonalen Sechseck-Wabenstruktur.

Die PwrMosFETs halten erstaunliches aus; zwei Misshandlungen **töten** sie jedoch rasch und dauerhaft:

a) Überschreiten der $U_{GS,max}$ (meist $\pm 20V$)
(auch **nur 1 [ns]** !)
Elektronen oder Metall-Ionen durchschlagen das Gate-Oxid und bleiben z.T. dort 'hängen', sodass diese Isolationsschicht leitend wird — das war's dann.
(besonders gefährlich bei
(selbst-)Induktionsspannungsspitzen)

b) Überhitzung == Betrieb ohne Kühlkörper (heatsink)
Die Atombewegungen werden so heftig, dass die Dotierungs-Atome ihren Platz verlassen (dem E-Feld entsprechend). Dies baut die Dotierungs-Grenzschichten ab/um zu einem falsch dotierten Bauteil, das dann immer schlechtere Verstärkungen und Schaltzeiten aufweist und schließlich dauerleitend wird.
Weiters wandert (*'vergiftet'*) Aluminium aus den Kontaktierungen ins Halbleitermaterial und überbrückt dort die Isolierungen; dies führt zu dauerleitenden Leistungshalbleitern.
Extrem überlasteten Bauteilen können aber auch interne Zuleitungs-Bond-Drähte ab-/durchschmelzen, sodass sie einfach nicht mehr kontaktiert sind (dann sperren sie natürlich).

Die TO-220/TO0247/DPAK Gehäuse (case) führen ohne Kühlkörper kaum Wärme ab und werden schon bei nur 1W Leistung unzulässig warm; bei 3W leben sie nur **wenige Sekunden** (experimentell ermittelt). Schon kleine U- oder Fingerkühlkörper verbessern das erheblich.
Wir messen deshalb, wie im Leistungsbereich üblich, *'gepulst'*, d.h. mit Rechtecksignalen *kurzer* Einschaltzeit. Mit *'kurz'* meinen wir solche Einschaltzeiten, die die spezifizierten, maximalen *'single pulse avalanche energy'* (zB. $E_{AS} = 91[mJ]$) und *'repetitive avalanche energy'* (zB. $E_{AR} = 4.8[mJ]$) -Angaben des Bauteils ausreichend unterschreiten.
(*Energie* $W[Joule] = P[Watt] \times t[Sekunden]$) Die verbreiteten Impuls-Schalt-Leistungs-MOSFET in den Gehäusen TO-220, TO-220FP/ISO, TO-247, DPAK, D2PAK ua:



Das heisst auch für DICH:

a) ausschließlich nur **mit** Kühlkörper! ← nur! ← **niemals!!!** ohne

b) **DU** sorgst dafür, dass die V_{GS} die max.±20V **niemals** überschreiten **kann!**

Die Anschlüsse sind (i.d.R.)

1-Gate

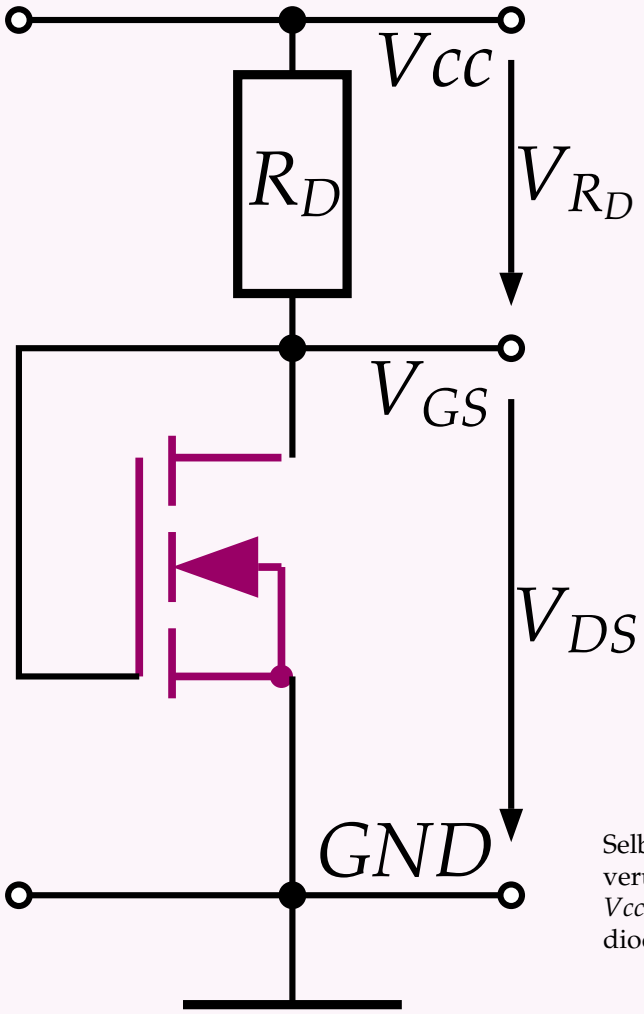
2-Drain (mit Kühllasche (4) verbunden)

3-Source

Die neueren SMD Varianten werden oft mit der Kühllasche (4) flach auf die Platine (PCB) gelötet, sodass die Kupferfläche ('pad') zugleich Drain-Anschluss und Kühlkörper (heat sink) darstellt und das Anschlussbein (2) fehlt.
Typisch sind:

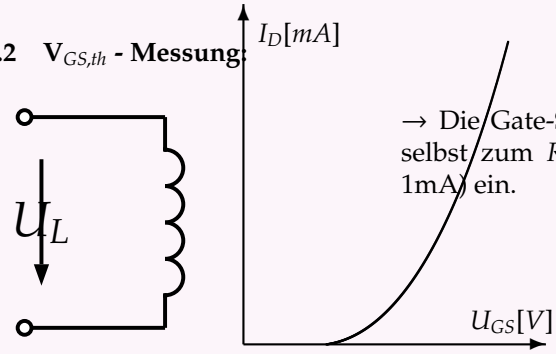
- $P_{V,max} \approx 50W$
- $V_{BR,DSS} \approx 50..800V$
- $I_{D,max} \approx 2..30A$

4 .2.1 Primitiv-Schaltung:



Selbst**induktion** ('inductive kickback') verursacht die bremsenden ($U_{spitz} > Vcc$) Spannungsspitzen → mit Freilaufdiode läuft der Motor besser.

4 .2.2 $V_{GS,th}$ - Messung:

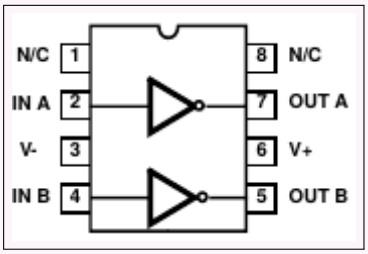


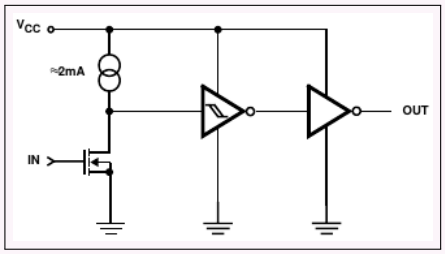
→ Die Gate-Spannung V_{GS} stellt sich selbst zum R_D-Strom passend (meist 1mA) ein.

4 .2.3 Gate-Treiber für Power MosFET

- Push/Pull BJT od. MosFET
- Gatterparallelschaltung (zB. alle 6 Inverter eines 4049 od. 4050 parallel)
- spez. ICs zB. TC4427 (< 1.5A), ICL7667 (< 300mA) ua.

  ICL7667:



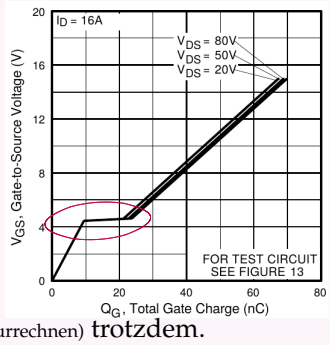


→ Wieviel Strom brauchi (um ein MosFET-Gate *schnell* aufzuladen) ?

Regel-1: **Kuh = Kuh:** **Q = C*U**: [As] = [F] * [V] ⟶ I = C*U/t [s] = [F]*[V]/[s]

Regel-2: **Miller-Effekt:** $C_{in} = C_{GS} + A_V * C_{GD}$ (A_V...Amplification of Voltage)

V(C_{GD}) muss sich nicht um ΔV_{in} erhöhen sondern
- weil gleichzeitig V_D sinkt -
um *viel* mehr (im kontinuierlichen (=analogen) Fall um die Spannungsverstärkung Av.



Im Schaltbetrieb ($A_V = \infty$) rechnet man **einfacher** mit den Ladungen:
Ein Datenblatt (zB. IRF540) zeigt das ausschlaggebende 'Plateau'
bei $V_{GS,th}$ mit dessen Miller-Effekt.
Die Q_G-Beiträge unterhalb und oberhalb
sind relativ bedeutungslos.
Der Datenblattwert 'Q_G'=Q_{GS} + Q_{GD} gilt für
ΔV_G: $0V \rightarrow 10V$
Obwohl eher der Bereich $3V \rightarrow 6V$ anzupeilen wäre,
verwenden wir den Datenblattwert einfachheitshalber (kannma ja korrekturrechnen) trotzdem.
Dann wird es einfach:

Mit Q = I*t ergibt sich | I = 72nCb / t |

Den IRF540 in 100ns einzuschalten erforderte I = 72nCb/100ns =

| 0.72A !!! |

Umgekehrt dauert das IRF540-Einschalten mit I_G=300mA (ICL7667) t = 72nCb/0.3A =

| 240 ns |

Ein BC560 ($I_{C,max} = 100mA$ braucht t = 72nCb/0.1A =

| 720 ns |

4 .2.4 Verlustleistungs-Berechnungs-Reparaturanleitung

XH am Sa-11Mar'22 an xyz

1) für die erwärmende Verlustleistung ist nicht die Schaltverzögerungszeit $t_{d,on}$ bzw. $t_{d,off}$ sondern die Strom-Anstiegszeit t_{rise} (t_r) bzw. Abfall-Zeit t_{fall} (t_f) maßgeblich. Beim IRF540N sind das die t_r=35ns und t_f=35ns (auch)
2) Phil hat inzwischen nachgerechnet: Die von mirXH vorgebrachte Schätzung
   Pv = Umax/2 * Imax/2 = Umax*Imax/4
   ist zu vorsichtig, der per Integral nachgerechnete Faktor ist
   Pv = Umax*Imax / 6
3) der IRF540 haltet ka $350V_{DS}$ aus, nur 100V
4) die 'leitend-Verluste $I_D^2 \cdot R_{DS}$' und die 'Schalt-Verluste $U_{DS} \cdot I_D/6$' derfsch nit einfach zamm zählen, weil sie zu unterschiedlichen Zeiten und vor allem unterschiedlich lang anfallen:
   - die Leitendverluste habi nur zur Einschalt-Dauer t_{ein}
   - die Schaltverluste habi nur beim Einschalten/Ausschalten fuer 35ns

   Ez mussma mitrechnen,
   a) wieviel % von einer Sekunde is des Ding eingeschaltet
   b) wieviel % von einer Sekunde wird grad von 'Lo' auf 'Hi' (Anstieg) oder von 'Hi' auf 'Lo' (Abfall) geschaltet.

Dazu brauchtma auch die Schaltfrequenz (von der hamma gar nie gredet)
→ Leitendverluste Pv,ein = $fs * 7,44W * t_{ein}/t_{periode}$
→ Schaltverluste Pv,S = $fs * 1137W * (t_r + t_f)/t_{periode}$
Ohne die Gewichtung mit der t_r bzw. t_f kimp ma auf vielzviel Pv

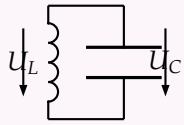
5) beim I_G hasch lediglich die 11ns t_d gnummen statt der 35ns t_r, sodass mirXH aussa kamat:
$I_G = (350V_{DS}+10V_{GS}) * 250pF_{Crss}/35ns(t_r) = 2.5Amp$

greez LGXh

4 .3 (Drossel-)Spule schalten, L*$\partial I/\partial t$

Spule	Kondensator
Induktionsspannung	Ladestrom
$U_L = L * \frac{\partial I}{\partial t}$	$I_C = C * \frac{\partial U}{\partial t}$
gespeicherte Energie	gespeicherte Energie
$W_L = \frac{L * I^2}{2}$	$W_C = \frac{C * U^2}{2}$

In einer Spule der Induktivität L=150μH
steigt bei $U_L = 5V$
die Stromstärke I um:

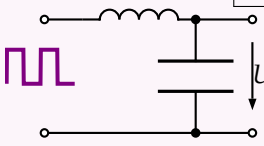


$$U_L = L * \frac{\Delta I}{\Delta t} \rightarrow \Delta I = \frac{U_L}{L} * \Delta t \quad (21)$$

$$= \frac{5V}{150\mu H} * 1\mu s = 33.3mA \text{ (je } \mu s) \quad (22)$$

$$\Delta I = 33.3\frac{mA}{\mu s} \quad (23)$$

Wenn wir damit einen Kondensator der Kapazität C=1μF
aufladen, dann ändert sich dessen Spannung U_C um:

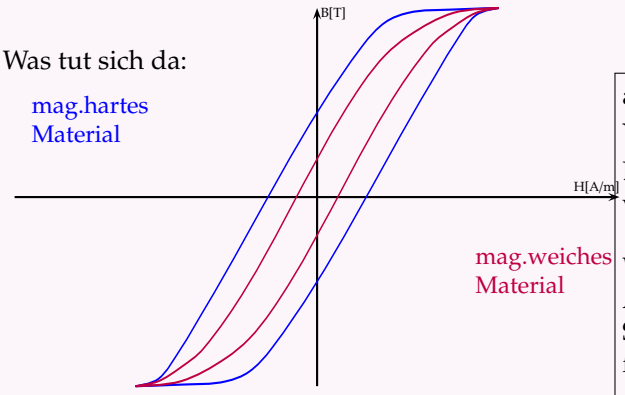


$$I_C = C * \frac{\Delta U}{\Delta t} \rightarrow \Delta U = \frac{I_C}{C} * \Delta t \quad (24)$$

$$= \frac{33mA}{1\mu F} * 1\mu s = 33.3mV \text{ (je } \mu s) \quad (25)$$

$$\Delta U = 33.3\frac{mV}{\mu s} \quad (26)$$

Was tut sich da:



allgemeine
wichtige Annahme:
Die Welligkeiten seien vernachlässigbar gering
→
Wir können während der Auf- und Entladevorgänge Spannungen und Ströme als fix so betrachten, dass
$\int_{\Delta t} U(t)dt \approx U(t) * \Delta t$
$\int_{\Delta t} I(t)dt \approx I(t) * \Delta t$

4 .4 Capacitor C und Induktor L mit PWM

Im Zuge von PWM sind Berechnungen und Dimensionierung mit Spulen und Kondensatoren vorzunehmen. Dazu brauchen wir

$$I_{Cap} = C\frac{\partial U_C}{\partial t}$$

$$U_{Ind} = L\frac{\partial I_L}{\partial t}$$

diskretisiert als:

$$I_{Cap} = C\frac{\Delta U_C}{\Delta t}$$

$$U_{Ind} = L\frac{\Delta I_L}{\Delta t}$$

(Δt so klein, dass U_{Ind} bzw. I_{cap} 'konstant genug')

Die Energie E=W in Cap und Ind ist:

$$W_{Cap} = C\frac{U^2}{2}$$

$$W_{Ind} = L\frac{I^2}{2}$$

Energie ≡ Arbeit
gespeicherte Energie [J] = aufgebrachte Arbeit [J] = entnehmbare Arbeit [J] (Energie-Erhaltung)

Im Stationärbetrieb müssen (logischerweise)

Spannungs-/Strom-Zunahme in der Einschaltphase	≡	Spannungs-/Strom-Abnahme in der Ausschaltphase
$+\Delta U_{c,ein}$	↔	$-\Delta U_{c,ein}$
$+\Delta I_{L,ein}$	↔	$-\Delta I_{L,ein}$

Daraus resultieren:

Spannungs×Zeit-Fläche (beim L):
$U_{ind} \bullet t_{ein} = -U_{ind} \bullet t_{aus} = L \bullet \Delta I_L$

und

Strom×Zeit-Fläche (beim C):
$I_{cap1} * t_{ein} = -I_{cap2} * t_{aus} = C * \Delta U_C$

Natürlich werden auch
- das OHM'sche Gesetz $U = R * I$
- die Kirchhoff'sche Maschenregel $\sum_m U_m = 0$ und Knotenregel $\sum_k I_k = 0$
- das lineare Überlagerungsprinzip (Helmholtz)

angewendet.

Puls =	Rechteck-Schwingung (square wave)	
f	Frequenz[1/s] (frequency)	$\frac{AnzahlPulse}{[s]} = \frac{1}{t_{ein}+t_{aus}} = \frac{1}{t_{gesamt}}$
T=1/f	Periode[s] (period)	$T = t_{gesamt} = t_{ein} + t_{aus} = t_{on} + t_{off} = t_{tot}$
t_{ein} (t_{on})	Pulsbreite[s] ('PulsWeite', pulse width)	$t_{ein} + t_{aus} = t_{gesamt}$ ($t_{on} + t_{off} = t_{tot}$)
t_{rise}	Anstiegszeit[s] (rise time)	von 10%*Umax bis 90%*Umax zu messen
t_{fall}	Abfallzeit[s] (fall time)	von 90%*Umax bis 10%*Umax zu messen
p	Tastverhältnis (duty cycle) ϵ[0..1] ≡ 0%..100%	$\frac{t_{ein}}{t_{gesamt}} = \frac{t_{ein}}{t_{ein}+t_{aus}} = f * t_{ein}$
$\hat{U}$	Pulsamplitude[V]	$U_{puls}\ \epsilon\ \{Umin, Umax\}$

4 .5 Design of the Drossel- /Trafospule

4 .5.1 se story of Magnetfeld

Es war einmal ein Magnetfeld H. Einsam, allein, lonely, fad. So sehnte sich H nach einem gleichartigen Partner, aber halt folgsam, abhängig, gut proportional, so, wie die elektrische Spannung U einen hatte - den elektrischen Strom I, der bekanntlich bei der geringsten Spannung dahinfließt. Und siehe da - gesucht - gefunden - der magnetische Fluss Φ (→magnetische Flussdichte $B = \frac{\Phi}{A_{quer}}$) bot sich an. Nach dem ***H**elvetischen **B**ekenntnis (**HB**)* wurden $\mathbb{H}$ und $\mathbb{B}$ auf dem magnetischen Standesamt vom ehrwürdigen Proportionalitätsfaktor *"Müh"* (ein Grieche, in den Buchstaben seiner Muttersprache 'μ') getraut (in doppelbrechenden Medien ist μ ein Tensor). Als Ehering diente ein damals hochmoderner Ferritring mit der Signatur 'A_L' ($\frac{\mu \cdot A_{quer}}{l}$) und B ist seither verdammt, ihrem H alles μ-fach nachzumachen

$$B = \mu * H$$

(gleich wie I, der muss bekanntlich immer das G-fache tun: $I = G * U \;\ldots\ldots\; G = \frac{1}{R}$

μ ist sowas wie $G = 1/R$, eine Art Leitwert - die "*magnetische Leitfähigkeit*"
(nennt man gelegentlich wirklich so)

Aus Phantasielosigkeit machen die nun alles dem Elektrischen nach:

$$H = \frac{I * N_{wdg}}{l}$$

↓

$$B = \mu * H$$

↓

$$U_{ind} = N_{wdg} * A_{quer,Kern} * \frac{\partial B}{\partial t}$$

Gib's zua, Du findeschdes kindisch, lesesches aber genauso wenig wia Fachbuch, sonsch würdeschdi ja scho auskennen. So gesehen ein *write only* Dokument, is dann eh wurscht, wasi schreib...

4 .5.2 (Selbst-) Induktion (inductive Kickback)

ist die Spannung U_{ind}, die in der Erreger(primär)wicklung selbst bei (wodurch immer ausgelöste) Magnetfeldänderung induziert wird.

$$\frac{U_{ind}}{N_{wdg}} = \frac{\partial(B * A_{\perp quer})}{\partial t}$$

entspricht

$$U_{ind} = L\frac{\partial I}{\partial t}$$

mit

$$\mathbf{L} = \underbrace{\frac{\mu \cdot A_{quer}}{l_e}}_{\text{"A}_L\text{ - Wert"}} * N_{wdg}^2$$

ACHTUNG: A_L- wird oft 'at 100 turns' (Windungen) angegeben

$$H = \frac{I}{l_e} \cdot N_{wdg}$$

$$B = \underbrace{\mu \cdot H}_{\mu \cdot I/l_e \cdot N}$$

$$U_{ind} = \frac{\partial(N_{wdg} * A_{\perp quer} * \mu \cdot I/l_e \cdot N_{wdg})}{\partial t}$$

$$U_{ind} \overset{\text{nur I variabel}}{=} L \cdot \partial I/\partial t$$

Gegeninduktion ist eine in einer anderen Wicklung induzierte Spannung $U_{ind,2}$ s.o.
Die *Windungsspannung* U_{ind}/N_{wdg} - die Induktionsspannung pro Wicklungswindung; sie ist in allen vom selben Magnetfeld durchflossenen Windungen gleich (Wicklungssinn = links-/rechtsdrehend beachten)

Streu-Induktivität L_s, Kopplungsfaktor M (mutual)
Der Grossteil des Magnetfeldes wird im *Magnetkern* geführt. Ein kleiner Rest der Magnetfeldlinien ausserhalb des Kerns wird entweder

a) als *'Streufeld'* in einer gesonderten *'Streufeldinduktivität'*, *'Streuinduktivität'* L_{streu} erfasst
($L_{streu} << L$ $\overset{erzeugt}{\Longrightarrow}$ Resonanzen mit $f_{streu} >> f_{schalt}$)

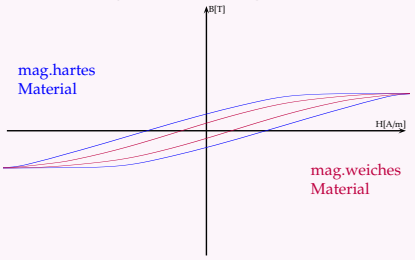
oder b)
als *Verlust* im Kopplungsfaktor $M < 1.0$ (zB 0,98) ignoriert.

Schaltungssimulatoren realisieren Trafos prinzipiell als zwei gekoppelte Spulen mit Kopplungsbefehl 'K' und Faktor M < 1
zB.

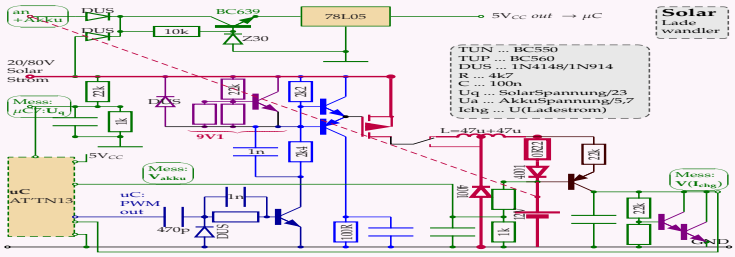
```
L1    node1    node2    160u
L2    node3    node4    1.6u
K12   L1       L2       0.98
```

4 .5.3 Kernmaterial

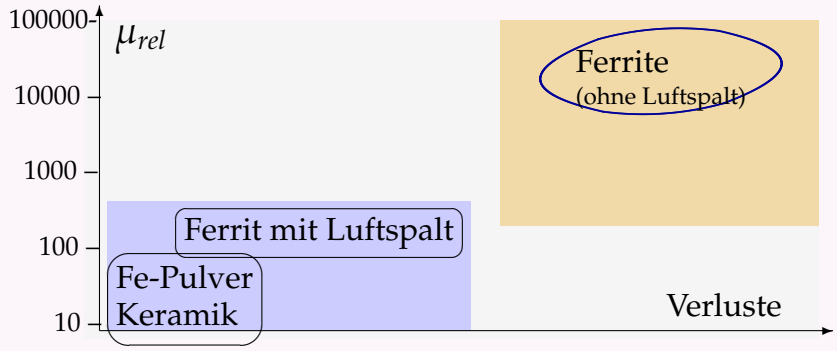
Ferrit-Magentisierungskurve



Magentisierungskurve mit Luftspalt bzw. Eisenpulver



- Luft
  hat den Vorteil, dass man sie magnetisieren kann, bis es die Moleküle zerreisst (und da magi nit dabei sein)
  dafür ist das "$\mu_{relativ}$" == 1.0, dh. sie hat nur wenig "L" bzw. braucht 'Millionen' Windungen, wodurch der Drahtwiderstand hoch wird.
- Ferrit
  wird wegen des hohen "$\mu_{relativ}$" (bis 10000 und drüber) meist verwendet.
  Für Speicherdrosseln ist ein "Luftspalt" erforderlich, (beim Eisenpulver ist der schon mit reingemischt)
- Eisenpulver
  wirkt wie ein fein verteilter Luftspalt
- Nägel, Wasserleitungsrohre, Zahnspangen, Dauermagnete . . .
  sind gänzlich ungeeignet, da es *magnetisch 'harte'* Werkstoffe sind, dh. sie lassen sich schwer magnetisieren und entmagnetisieren - jedes Mal wird Energie verheizt.



Ferrit:	**Luftspalt + Eisenpulver:**
+viel μ +Verluste	+wenig μ +hohe Güte 'Q'[7]
wenn ich Ferritmaterial magnetisiere, kriege ich hohe Verluste+Erwärmung → a) ich magnetisiere nicht (kaum) b) ich will die Verluste	Nachteil: wenig μ Vorteil: wenig Verlust Ein Luftspalt erzeugt starkes "Streufeld"

a) wenig magnetisiert → kaum Verluste:	Trafo	Schwingkreis:	hohe Güte
b) Verluste erwünscht:	EMI choke	Speicherkern:	maximal magnetisiert
viel Verlust		wenig Verluste	

4 .5.4 some Designrulez

- $B_{max} < 200mT$ immer sicher unterschreiten
- mach X_L bei der niedersten Betriebsfrequenz > 5 mal so gross wie die Quell-/Lastimpedanz (HF)
- "Windungsspannung": $\frac{U_{L,ind}}{proWdg.} = A_{querschnitt} * \frac{\partial B}{\partial t}$
  $\xrightarrow[Wicklung]{gesamte} U_{L,ind} = N * A_{querschnitt} * \frac{\partial B}{\partial t}$
- viel μ + **wenig** Windungen → Kernverluste
  wenig μ + **viele** Windungen → Drahtverluste, parasit.Kapazität, reduzierte Bandbreite
- HF- Übertrager haben wenig Windungen, nura handvoll, brauchen aber auch wenig L
- keine Ferrite für Schwingkreise!
  (zu hohe Temperaturkoeffizienten, Verluste + Rauschen)

4 .5.5 Aufgabe-3d: DC/DC Wandlung mit Drosselspule

Aufgabe:

Entwirf und dimensioniere
einen Abwärtswandler (*'buck converter'*)
f. folgende Angaben

geg.:

Wandlerdrossel $L_1 = 150uH$, $0.5A_{max}$
$U_{in} = 5V$ (nicht aus USB!)
$U_{out} < U_{in} = 3.3V$
Welligkeit (ripple) < 20mVss

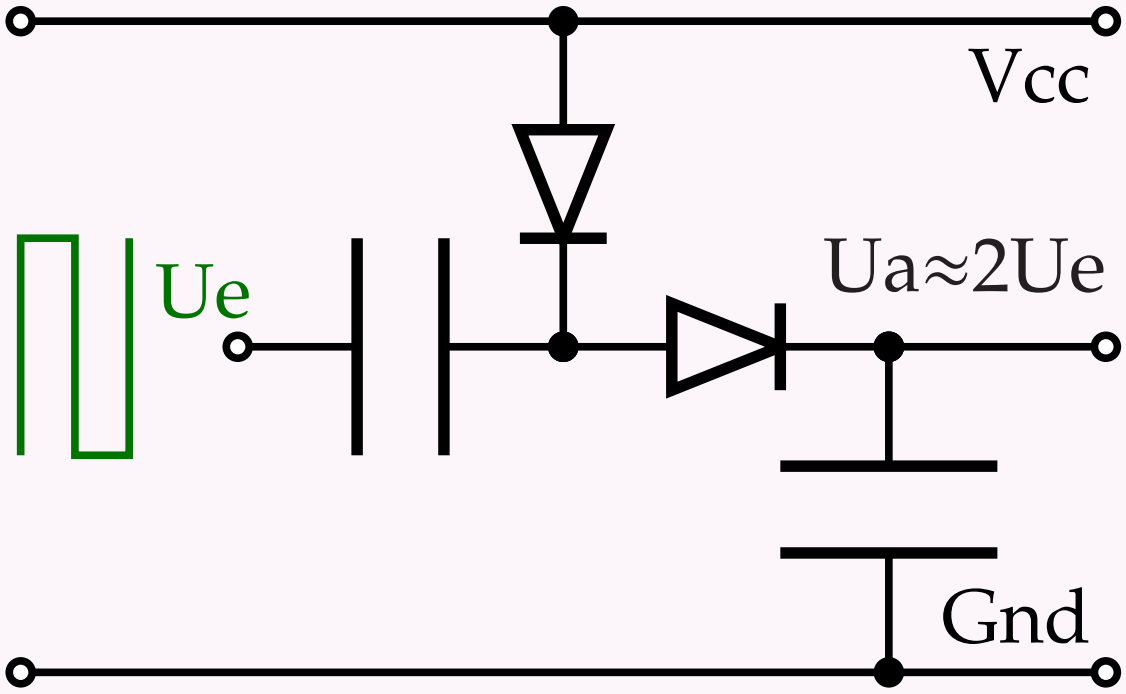
ges.:

Schaltfrequenz f_s,
Tastverhältnis p
Ladekapazität C_a

Prinzip und Dimensionierung:

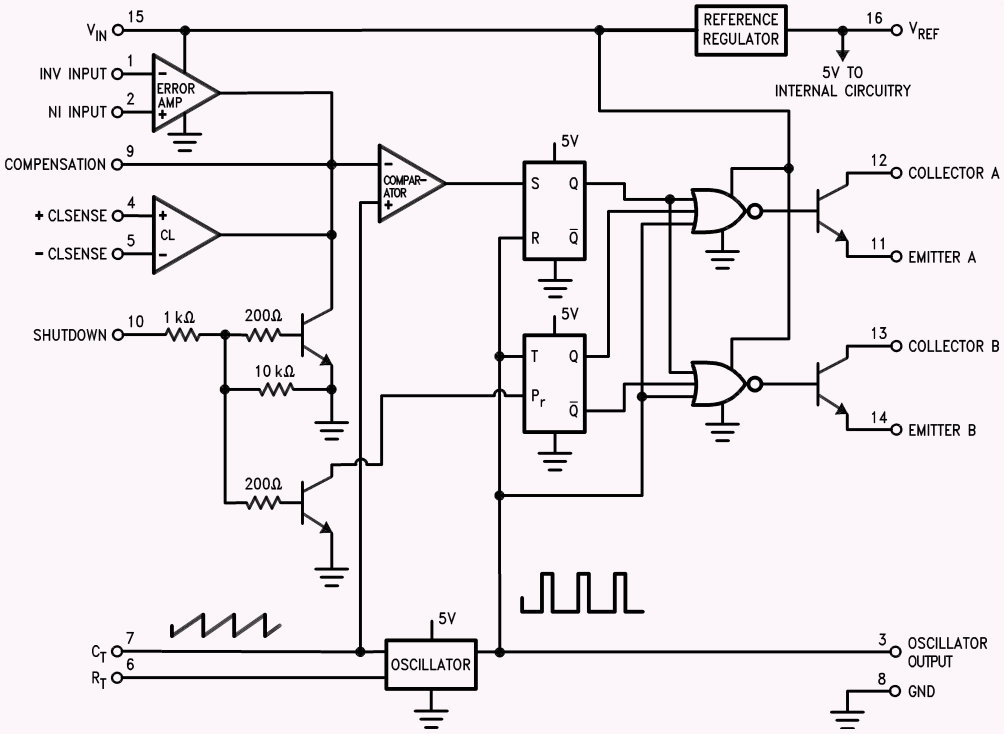
→
Abwärtswandler (buck converter, step-down converter)
s. Kap 4 , S.116,
'Prinzip: Abwärtswandler (buck converter)'

4 .6 Solarwandlerschaltung



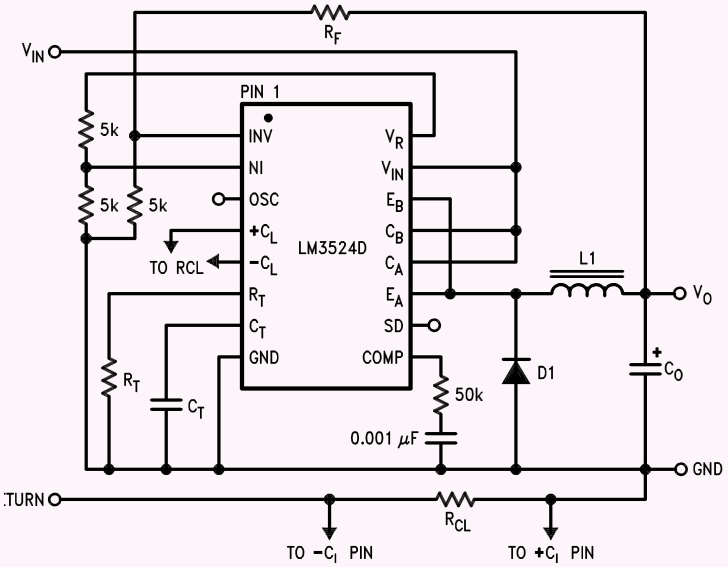
4 .7 PWM Regelung + ICs

4 .7.1 SMPS mit SG3524



BlockDG:

4 .7.2 SG3524 typical application



4 .7.3 SMPS mit Ne555

s. auch → "NE555 Application notes"

4 .7.4 SMPS mit uC (micro ontroller)

4 .7.5 Argumentation

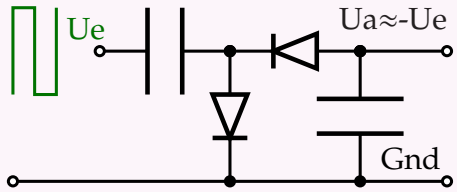
+ **'555** reicht bis 500kHz
+ **'555** ist nahe am Prinzip der PWM-Erzeugung mit Δ-Generator plus Komparator
− **uC** erfordert zusätzl. Programmierung (Einrichtung e. Entwicklungssystems)
+ **uC** kann volle 0..100% duty cycle (Tastverhältnis)

TBD. (bedeutet 'to be done' ≡ in Arbeit)

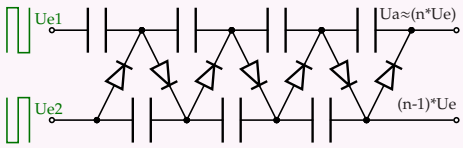
SG3524

4 .8 Ladungspumpe (*charge pump, voltage multiplier*), ICL7660-MAX1044-MAX680

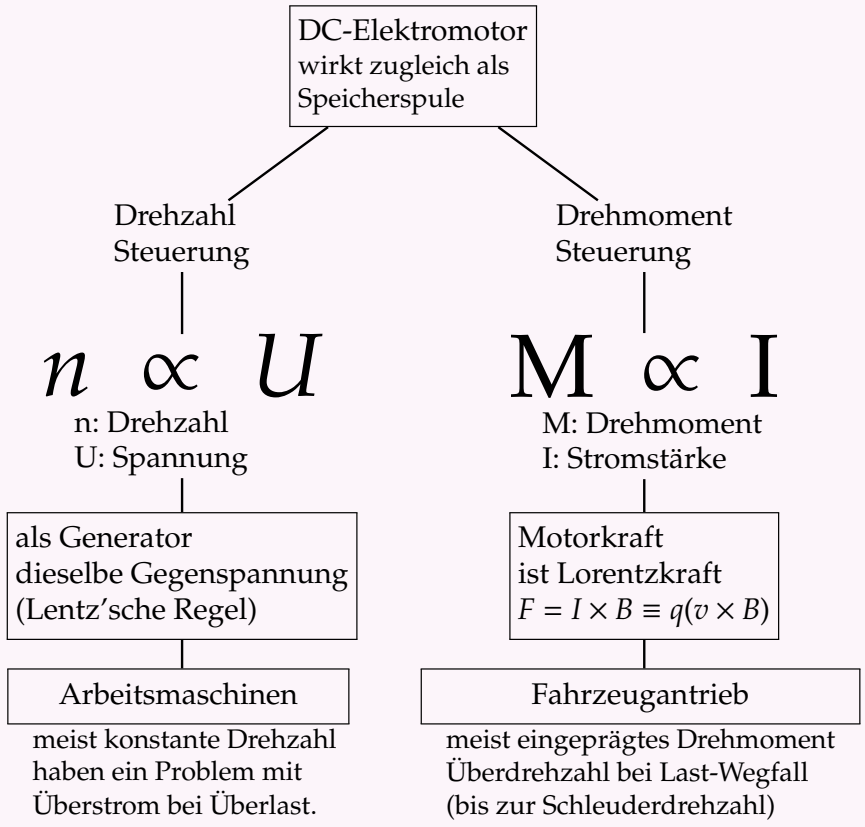
4 .8.1 Spannungsverdoppler (Villard)



4 .8.2 Inverter



4 .8.3 Spannungsvervielfacher (Greinacher)



entwirf und dimensioniere Ladungspumpen (charge pump) für

- Spannungsverdopplung aus USB
- inverse Spannung -5.0Vdc aus USB
- duale (symmetrische) Stromversorgung ±15V f. OpAmp
- 30Vdc Kapazitätsdioden- (varicap) Abstimmspannung
- ermittle den Innenwiderstand Deiner Ladungspumpe a) rechnerisch b) messtechnisch; kann

$Ri = \frac{N}{f \cdot C}$ stimmen?; als Rechteck-Quelle verwende den Funktionsgenerator (Ri=50Ω!) oder den NE555/556 (Bauteilset)

mit

- NE555
- ICL7660 ≡ MAX1044
- MAX680

TBD. (bedeutet 'to be done' ≡ in Arbeit)

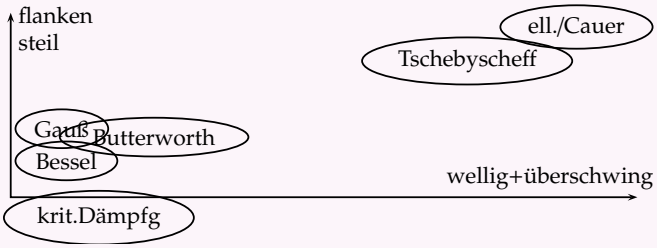
Ladungs pumpe

.

4 .9 Aufgabe-4: eMotor

Entwirf einen PWM Motorstromwandler für DC- Nebenschlussmotoren

- für 5V Ausgang (e-Motor)
- 12V_{DC} Eingang (Akku, Labornetzteil, Halogenvorschaltgerät)
- NE555 als PWM Generator
- als *'buck converter'*
- mit n-Kanal-MosFET (mir ham kan p-Channel MosFET)



4 .10 Aufgabe-5: LED Betrieb

Entwirf einen Konstant**strom**-Drosselwandler

- **20mA** Ausgang (LED Betrieb seriell,
  dh. 1, 2, 3... LED in Serie - immer 20mA)
- **12**[±5]V_{DC} Eingang (Akku, Labornetzteil, Halogenvorschaltgerät)
- NE555 als PWM Generator
  mit **n-Channel-PwrMosFET** als Leistungsschalter
- Design Job: Als *'buck? / boost? converter'*
- Design Job: Regelkreis für LED-Konstantstrom?

4 .11 Aufgabe-6: Stromversorgung

Zur Stromversorgung diverser Baugruppen wird (anstelle von 230V-Netzgeräten) in zunehmendem Ausmaß das *"PoL - Point of Load"* Konzept verfolgt: Direkt am Ort des Bedarfes (zB. am Prozessor, RAM, Cache, Sender,...) werden maßgeschneiderte DC/DC-Wandler eingebaut, die aus einer zentralen (weniger stabilen) Quelle (Akku, PoE, Energy Harvesting, Netzwandler) gespeist sind.

Solche Wandler sind also (neben LED/Motor/Akku-Vorschaltwandlern) von zentraler Bedeutung

Wir studieren deshalb die häufigsten Wandlerkonzepte aus dem Kleinleistungsbereich:

- **Durchfluss**wandler
- **Sperr**wandler
- **Drossel**wandler

TBD. (bedeutet 'to be done' ≡ in Arbeit)

4 .12 Digitalendstufe (class D Audio Amp)

4 .12.1 Class D Audio Amp

Im Gegensatz zur Gegentaktendstufe, die ein Signal analog zum Originalsignal soweit verstärken kann, bis es die maximal vom Netzteil zur Verfügung stehende Spannung erreicht, arbeitet die PWM-Endstufe nach einem anderen Prinzip: Es schwingt ein symmetrisch arbeitender Dreiecksgenerator mit einer typischen Frequenz von ca. 250..500 kHz. Das anliegende Tonsignal wird mit einem Komparator mit dem Dreiecksignal verglichen. Durch den Aufbau als Komparator verändert die Schaltung das sinusförmige Tonsignal in Rechteckpakete.

Ist das Dreiecksignal größer als das Tonsignal, springt der digitale Ausgang auf "high". Ist das Dreiecksignal kleiner, dann springt es auf "low". Das Tonsignal liegt nun im Tastverhältnis des modulierten Signals vor. Der Mittelwert liegt dadurch etwa proportional zum Mittelwert des Tonsignals. Diese Rechteckpakete werden der eigentlichen Endstufe zugeführt, welche nur ein- bzw. ausschalten aber keinen linearen Betrieb aufweist. Dadurch wird an der Endstufe, bestehend aus Leistungstransistoren im Schaltbetrieb, weniger Verlustleistung erzeugt und der Wirkungsgrad ist höher.

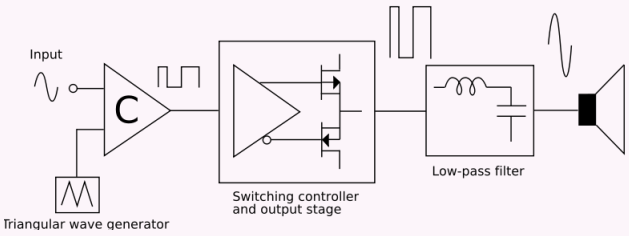
Vorteile sind neben dem

+ höheren Wirkungsgrad
+ ein einfacherer Netzteilaufbau, meist in Form eines Schaltnetzteils mit nur einer Versorgungsspannung;
+ geringere Wärmeverluste
+ und damit weniger Energieverbrauch.
+ Ein weiterer Vorteil ist das erheblich niedrigere Gewicht gegenüber analogen Geräten.

Es ergeben sich bei zu geringer Wahl der PWM-Frequenz Verzerrungen des Signals und Artefakte. Deswegen sollte die Schaltfrequenz möglichst hoch sein, was aber den Wirkungsgrad senkt. Ein weiterer Nachteil sind die systembedingten Oberschwingungen, die ein Störsignal im Frequenzbereichen von Langwellensendern abgeben können. Bei hochwertigeren Audioendstufen werden diese Störungen durch passive Filter zwischen Endstufe und Lautsprecher unterdrückt.

Anwendung:

- Handies, Smartphones, PC-Soundkarten
- Beschallungsanlagen ("PA-Anlagen")
- Kfz-Verstärker (bes.Subwoofer) (hohe Leistung bei geringer Versorgungsspannung)



5 (akt.) Filterberechnung mit Tabellen

(* Mittlerweile gibt es Entwurfs- und Simulationsprogramme für alle Filtertypen und -Ordnungen im 'www')

5 .1 Filtercharakteristiken

- Filter mit kritischer Dämpfung
  - rückwirkungsfreie(!) Reihenschaltung passiver RC-TP/HP
  - kein Überschwingen der Sprungantwort.
  - Char.Polynom besitzt nur reele Nullstellen
  - "schlechter Amplitudenfrequenzgang" ::= wenig Flankensteilheit
  - steile Filter brauchen hohe Filterordnung

- Bessel-Filter:
  - keine Welligkeit
  - bester Phasenfrequenzgang (konstante Gruppenlaufzeit) im Passband, → geringste Verzerrungen / bestes Impulsübertragungsverhalten.
  - Übergangsbereich nicht so 'scharf' wie Butterworth
  - $\tau_{gr} = const. \iff \varphi(\omega) \propto \omega$

    φ ... Phasenverschiebung, phase lag
    ω ... Frequenz (Kreisfrequenz)
    τ_{gr} ... Gruppenlaufzeit
    $\propto$ ... proportional
  - Flankensteilheit nur mit hoher Filterordnung → nur Spezialfälle.

- Gauß Filter:
  - [...] Sprungantwort keine Überschwingung
  - [...] maximale Flankensteilheit im Übergangsbereich aufweisen. Als Besonderheit besitzt bei diesem Filter sowohl
  - die Übertragungsfunktion als auch
  - die Impulsantwort den Verlauf einer gaußschen Glockenkurve [...] wovon sich auch der Name dieses Filtertyps ableitet. Anwendungsbereiche [...]
  - digitale Modulationsverfahren und [...] (Gauß-Filter besitzen eine konstante Gruppenlaufzeit im Sperr- und Durchlassbereich und kein Überschwingen in der Sprungantwort. Einsatzbereich dieses Filters liegt primär zur Impulsformung mit Anwendungsbereichen in der digitalen Signalverarbeitung. Die Impulsformung findet [Anwendung] bei digitalen Modulationsverfahren wie dem Gaussian Minimum Shift Keying (GMSK), da damit die einzelnen, meist rechteckförmigen Sendesymbole in Impulse der gaußschen Glockenkurve mit geringerem Bandbreitenbedarf als die ursprünglichen rechteckförmigen Sendesymbole umgewandelt werden können. Damit ist eine höhere spektrale Effizienz des Modulationsverfahrens verbunden.)
  - Bildverarbeitung (In der Bildverarbeitung werden Gauß-Filter zur Glättung oder Weichzeichnen des Bildinhaltes verwendet. Es kann damit das Bildrauschen vermindert werden: Kleinere Strukturen gehen verloren, gröbere Strukturen bleiben)

  ('*aus Wikipedia, der freien Enzyklopädie*' `de.wikipedia.org/wiki/Gauss-Filter` 21Mar16)

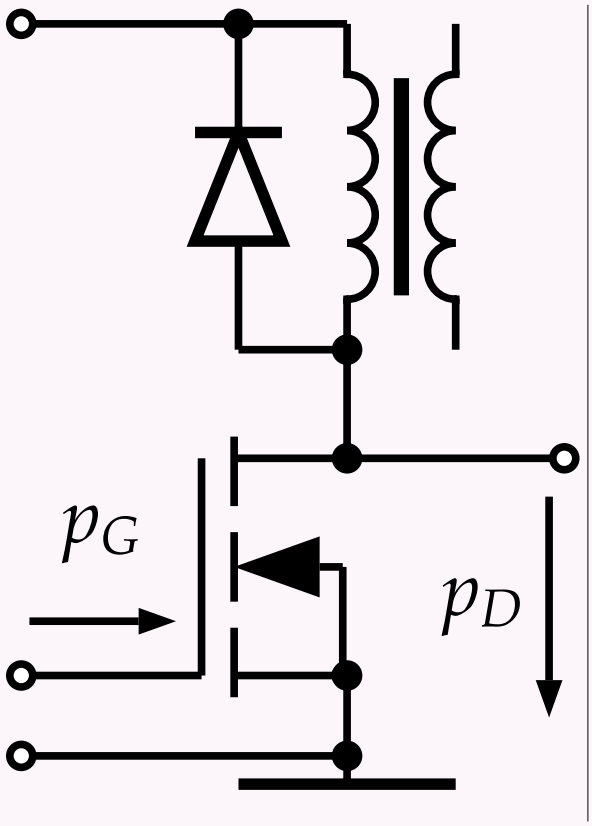
- Butterworth Charakteristik
  - best-horizontaler Amplitudenfrequenzgang
  - keine Welligkeit im Passband,
  - schärfster Knick vor der Grenzfrequenz + maximal steil
  - nicht konstante Gruppenlaufzeit
  - leichtes Überschwingen
  - gut dimensionier- u. realisierbar → verbreitet

- Tschebyscheff:
  - Amplituden-Welligkeit nur im Passband (passband ripple)
  - steilster Übergang (steepness)
  - Welligkeit $\overset{\propto}{\leftrightarrow}$ Flankensteilheit
  - mit Welligkeiten 0.1, 0.2, 0.5, 1, 2, 3dB tabelliert
  - gut dimensionier- u. realisierbar → verbreitet

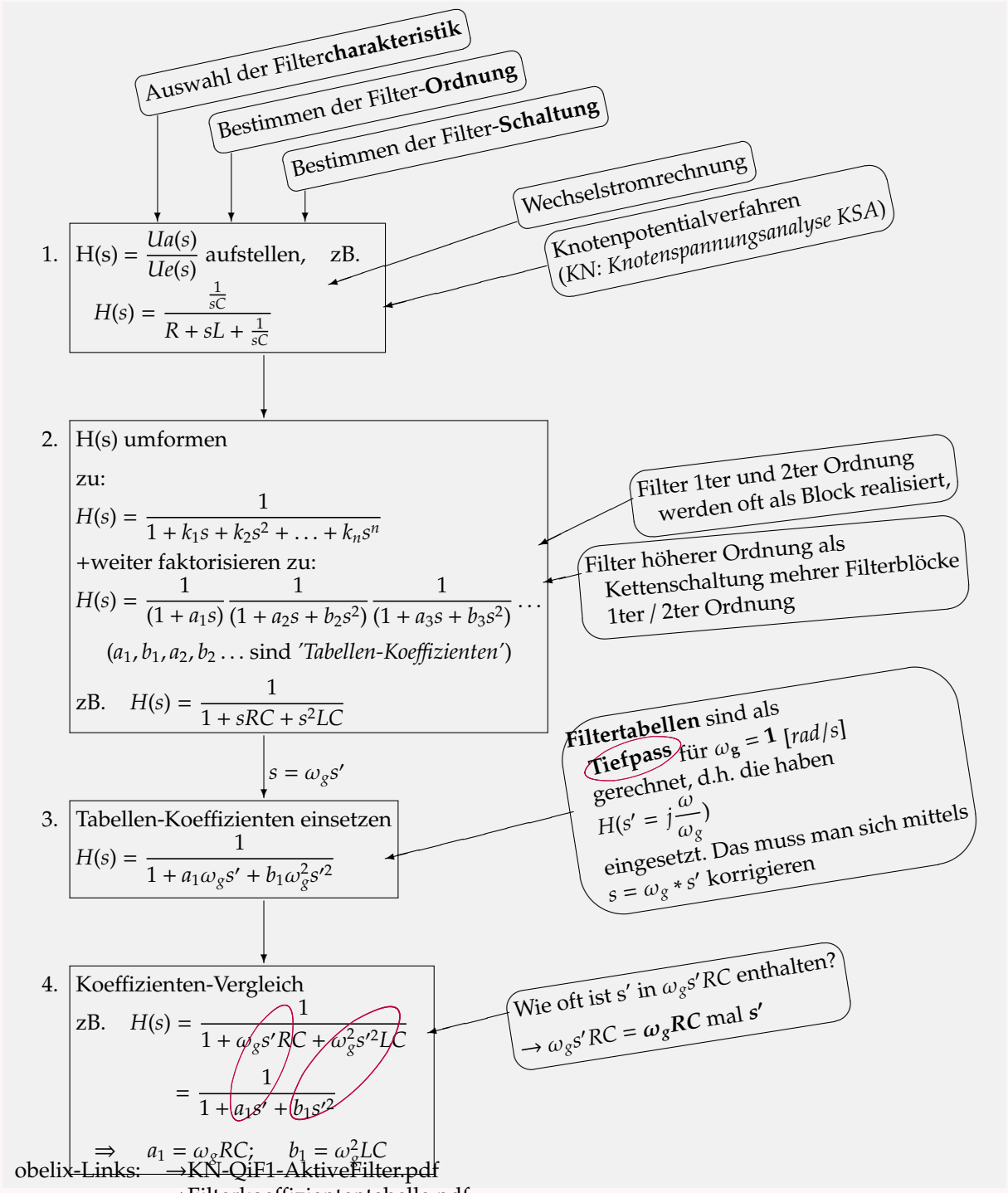
- Tschebyscheff invers:
  - Welligkeit nur im Sperrbereich
  - Gruppenlaufzeit/Phasenverschiebung besser als Tschebyscheff
  - gut dimensionierbar
  - weniger gut realisierbar wegen Nullstellen

- Elliptische Filter (Cauer):
  - größte Flankensteilheit
  - Welligkeit im Durchlass- und Sperrbereich (spezifizierbar)
  - aufwendige Dimensionierung u. Realisierung → Synthesesoftware

- nicht-impedanz-angepasste Eingangs- oder Ausgangsbeschaltungen erhöhen iA. die Welligkeit (Eingangs- u. Ausgangs-'R' sind Bestandteil der Dimensionierung)


p_G
p_D

5 .2 kontinuierliche, analoge Filter - DesignMethode



obelix-Links: →KN-QiF1-AktiveFilter.pdf
→Filterkoeffiziententabelle.pdf
→Aufgabenstellung.html
→AufgabenstellungEqualizer.pdf

5 .3 Knotenpotentialverfahren

Das Knotenpotentialverfahren (auch Knotenspannungsanalyse oder Knotenadmittanzverfahren) ist ein Verfahren zur Netzwerkanalyse in der Elektrotechnik. Mit dieser Methode lassen sich die Knotenpotentiale eines elektrischen Netzwerks aus linearen Bauelementen bestimmen.

Anwendung des Verfahrens
Das Verfahren wird gewöhnlich zur Bestimmung eines Stromes in einem Zweig verwendet. Gegenüber der Zweigstromanalyse werden bei diesem Verfahren so viele Gleichungen eingespart, wie das Netzwerk unabhängige Maschen besitzt. Im folgenden werden alle Schritte zum gesuchten Wert aufgezeigt. Dieses Verfahren gilt auch für komplexe und magnetische Netzwerke, sofern nur lineare Bauelemente vorkommen.

Knotenpotentiale und Bezugsknoten festlegen
Bei einem Netzwerk mit k Knoten gibt es k-1 unabhängige Knotengleichungen. Für einen Knoten muss keine Gleichung aufgestellt werden, da sich dessen Gleichung aus den Gleichungen der anderen Knoten aufstellen ließe und damit linear abhängig wäre. Dieser Knoten ist deshalb der Bezugsknoten mit Nullpotential (Masse) und kann beliebig gewählt werden. Zweckmäßigerweise sollte der Knoten an einem Zweig mit gesuchtem Spannungsabfall liegen, da so schon ein benötigtes Potential feststeht und das Gleichungssystem einmal weniger gelöst werden muss. Alle anderen Potentiale sind noch unbekannt und werden mit einem eindeutigen Variablennamen bezeichnet.

Umwandlung der Widerstände und Spannungsquellen
Die Zweigströme werden als Produkt aus Zweigleitwert und Knotenpotenzialdifferenz ausgedrückt. Deshalb werden die Zweigwiderstände durch deren Leitwerte ersetzt und die Spannungsquellen nach dem Norton-Theorem in Ersatzstromquellen umgeformt. Ideale Spannungsquellen ohne Widerstand im Zweig können nicht umgeformt werden. Weiteres dazu im Punkt Behandlung von idealen Spannungsquellen.

Matrix des linearen Gleichungssystems aufstellen
Die Leitwertmatrix wird wie folgt aufgestellt:

* Auf der Hauptdiagonalen $\forall i, j$ mit $i = j$ steht die Summe der Leitwerte aller Zweige, die mit Knoten i verbunden sind.
* An den anderen Stellen $\forall i, j$ mit $i \neq j$ steht die negative Summe der Leitwerte zwischen den benachbarten Knoten i und j (Koppelleitwerte). Besteht keine direkte Verbindung zwischen zwei Knoten, wird an dieser Stelle eine Null eingetragen.

Die Leitwertmatrix ist eine symmetrische Matrix. Folglich sind die gegenüberliegenden Koppelleitwertwerte (bezüglich der Hauptdiagonale) identisch ($G_{ij} = Gji \forall i, jmiti \neq j$). Das muss so sein, weil sich diese Koppelleitwerte in beiden Fällen zwischen denselben Knoten befinden. Im Gegensatz zu den positiven Summenleitwerten auf der Hauptdiagonalen sind alle Koppelleitwerte negativ.
Im Vektor der Knotenpotentiale muss die gleiche Reihenfolge wie auf der Hauptdiagonalen der Leitwertmatrix eingehalten werden.
Im Vektor der Knotenströme auf der anderen Seite des Gleichungssystems steht die Summe der Ersatzstromquellen mit denen der jeweilige Knoten verbunden ist. Hinfließende Ströme gehen positiv, wegfließende Ströme gehen negativ in die Summe ein (es geht auch andersherum, es muss nur einheitlich für alle Knoten erfolgen). Sind keine Quellen mit dem Knoten verbunden, wird eine Null eingetragen.

Behandlung idealer Spannungsquellen
In sehr seltenen Fällen kann sich eine ideale Spannungsquelle (ohne Innen-/Zweigwiderstand) in einem Zweig zwischen zwei Knoten befinden. Dadurch ist die Spannungsdifferenz zwischen den beiden Knoten bekannt und ein Potential kann mit Hilfe des konstanten Wertes der Quellspannung aus dem anderen direkt berechnet werden. Dabei ist zu beachten, in welche Richtung die Spannung der Quelle abfällt:

$$\phi_{high} - \phi_{low} = U_{q_{ideal}}$$

mit Spannungsabfall von "high"-Knoten zu "low"-Knoten
Die Gleichung wird nach dem zu ersetzenden Potential umgeformt und in das Gleichungssystem eingesetzt. Falls einer der Knoten der Bezugsknoten ist, muss selbstverständlich das Potential des anderen ersetzt werden. Im Gleichungssystem wird der eingesetzte Term in jeder Zeile mit den Leitwerten in der zugehörigen Spalte multipliziert. Die Terme mit $U_{q_{ideal}}$ werden auf die Seite der Stromquellen verschoben.

Das weitere Vorgehen ist nun abhängig von der Position des Bezugsknotens. Alle Zeilen der ersetzten Potentiale, deren ideale Spannungsquelle mit dem Bezugsknoten direkt verbunden ist, müssen gestrichen werden. Dadurch reduziert sich der Grad des Gleichungssystems mit jeder idealen Spannungsquelle am Bezugsknoten um Eins. Für alle anderen idealen Spannungsquellen wird in ihren Zweig ein unbekannter Zweigstrom eingeführt. Diese werden zunächst auf die Seite der Stromquellen nach dem gleichen Schema wie die Stromquellen eingetragen. Hinfließende

addiert, Wegfließende subtrahiert. Abschließend werden die unbekannten Zweigströme auf die linke Seite gebracht. Für ideale Spannungsquellen ohne direkte Verbindung zum Bezugsknoten reduziert sich der Grad des Gleichungssystems folglich nicht, da für jedes entfallene Potential ein unbekannter Strom hinzukommt.

Gesuchte Potentiale berechnen
Vor der Berechnung eines Zweigstromes müssen die Potenziale der beiden angrenzenden Knoten (ϕ_i und ϕ_j) bekannt sein. Dazu wird das Gleichungssystem für eines der i, j Potentiale gelöst. Dies geschieht entweder mit Hilfe der cramerschen Regel oder durch das Gaußsche Eliminationsverfahren. Sollte einer davon der Bezugsknoten sein, muss nur ein Potential berechnet werden. Die Zweigspannung wird durch die Differenz der Knotenpotentiale für gewöhnlich so berechnet, dass die resultierende Zweigspannung in die vermutete Richtung des gesuchten Stroms abfällt. Der Wert einer eventuell vorhandenen Spannungsquelle im Zweig muss nach dem Maschensatz von der Zweigspannung subtrahiert werden, wenn ihre Spannung in Richtung Zweigspannung abfällt, oder addiert werden, wenn sie in entgegengesetzte Richtung verläuft. Das Ergebnis wird anschließend durch den Zweigwiderstand geteilt bzw. mit dem Zweigleitwert multipliziert, um den gesuchten Strom zu erhalten. Ein positiver Zweigstrom fließt in Richtung des Spannungsabfalls der Knotenpotenzialdifferenz, ein negativer Zweigstrom in entgegengesetzte Richtung.

$$I_{ij} = \frac{\phi_i - \phi_j \pm U_{q_{ij}}}{R_{ij}} = \phi_i - \phi_j \pm U_{q_{ij}} \cdot G_{ij}$$

Anwendung
Das Knotenpotentialverfahren eignet sich hervorragend zur computerunterstützten Berechnung des Lösungsvektors, da sein Lineares Gleichungssystem durch einen einfacher zu programmierenden Algorithmus aufgestellt werden kann als beim Maschenstromverfahren, bei dem zunächst das Netzwerk graphentheoretisch nach einem vollständigen Baum abgesucht werden muss. Es bildet deshalb die Basis der meisten Rechnerprogramme zur Analyse Linearer Elektrischer Netzwerke. Allerdings ist die optimale Auswahl des zu verwendenden Netzwerkanalyseverfahrens abhängig von der Struktur des Netzwerks (Anzahl der Zweige verglichen mit der Anzahl der Knoten) und in der Praxis individuell für jedes Netzwerk.

Quelle: `http://de.wikipedia.org/w/index.php?oldid=94636288` Bearbeiter: Aka, Backsideficker, Balumir, Biezl, BuSchu, Ephraim33, Gereon K., Heinte, Ilion, Jodo, Mathemaduenn, Mik81, Pavel007, Regi51, Reseka, Saehrimnir, Segelmaus, Steevie, Stefan Birkner, Tobydox, Tram fan, 35 anonyme Bearbeitungen (21.Apr2012)

5 .4 Tiefpass/Hochpass Transformation

In der logarithmischen Darstellung kommt man vom Tiefpass zum analogen Hochpass, indem man die **Frequenzgangkurve der Verstärkung** (Bodediagramm)

an der Grenzfrequenz spiegelt,

d.h. Ω durch 1/Ω bzw. P durch 1/P ersetzt.

(mit $\Omega = \frac{\omega}{\omega_g}$, $P = j\Omega$)

Die Grenzfrequenz bleibt dabei erhalten und A_0 geht in A_∞ über. $H(P = j\omega/\omega_g)$ lautet dann

$$H(P) = \frac{A_\infty}{\prod_i (1 + \frac{a_1}{P} + \frac{b_i}{P^2})}$$

Die Überlegungen können allerdings nicht auf das Verhalten im Zeitbereich übernommen werden, da die Sprungantwort ein prinzipiell anderes Verhalten aufweist. (Es ergibt sich selbst bei Hochpassfiltern mit kritischer Dämpfung eine Schwingung um den stationären Wert.) Die Übertragungsfunktion eines Tiefpasses erster Ordnung lautet allgemein:

$$H(P) = \frac{A_0}{1 + a_1 P}$$

Um den analogen Hochpass zu erhalten, muss man P durch 1/P ersetzen.

$$H(P) = \frac{A_0}{1 + a_1/P}$$

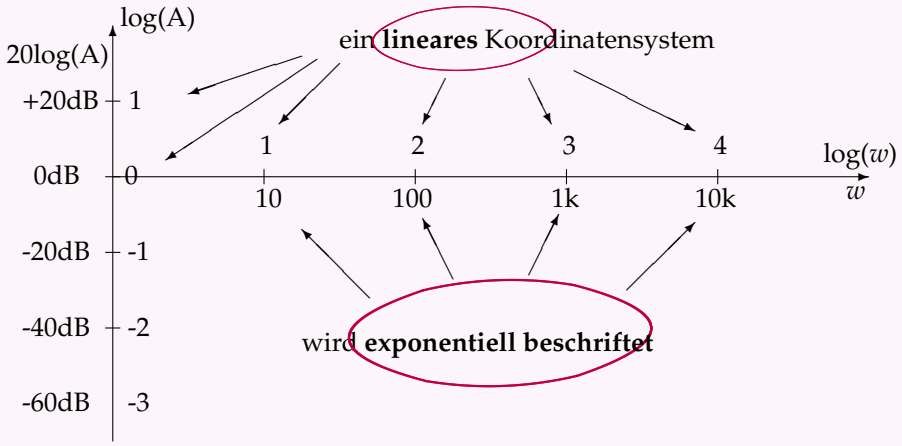
In der Schaltung lässt sich dies ganz einfach dadurch realisieren, dass man **R mit C vertauscht** (was natürlich eine andere Uebertragungsfunktion ergibt).

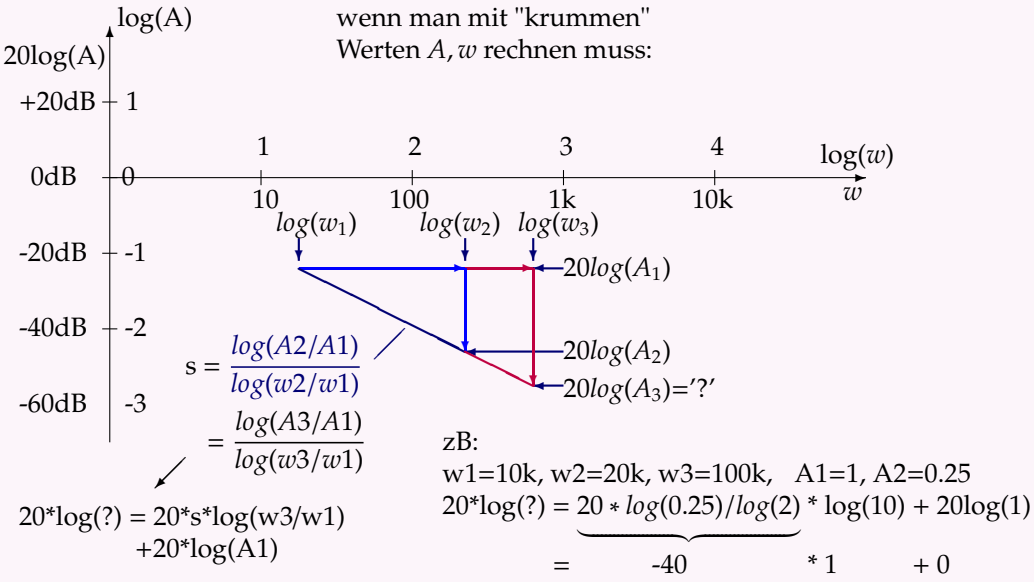
5 .5 Ordnung und Reihenfolge

Wenn die Filtercharakteristik nicht scharf genug ist, muss man Filter höherer Ordnung verwenden. Dazu schaltet man Filter erster und zweiter Ordnung in Reihe. Dabei multiplizieren sich die Frequenzgänge der einzelnen Filter. Die Koeffizienten, die für a1 und b1 zu wählen sind, um die Grenzfrequenz fg zu erzielen, sind für jede Stufe verschieden und in der Literatur tabelliert, z.B. in Tietze Schenk. Entsprechend können dann die Bauelemente der verschiedenen Stufen für die gemeinsame Grenzfrequenz fg bestimmt werden.

Im Prinzip ist es gleichgültig, in welcher Reihenfolge man die einzelnen Filterstufen angeordnet, da der resultierende Frequenzgang immer derselbe bleibt. In der Praxis gibt es jedoch verschiedene Gesichtspunkte für die Reihenfolge der Filterstufen, z.B. die Aussteuerbarkeit. Nach diesem Gesichtspunkt ist es günstig, die Teilfilter der [*steigenden*(*xh*)] Grenzfrequenz nach zu ordnen und das mit der niedrigsten Grenzfrequenz an den Eingang zu schalten. Sonst kann die erste Stufe bereits übersteuert werden, wenn am Ausgang der zweiten noch keine Vollaussteuerung auftritt. Das kommt daher, dass die Filterstufen mit der höheren Grenzfrequenz durchweg eine höhere Polgüte besitzen und damit auch einen Anstieg der Verstärkung in der Nähe ihrer Grenzfrequenz aufweisen.
(aus ZHAW School of Engineering, EK2, HS2009)

5 .6 rechnen im Bode-Diagramm





5 .7 Filter mit Mehrfachgegenkopplung 'MGK'

s. →"KN-QiF1-AktiveFilter.pdf"

5 .8 Projekt-1 KN'1617: akt. Filter mit OP

KN:

- 3-Kanal-NF-'Equalizer'
  mit Tiefen: TP 4ter Ordng.
  mit Mitten: TP 2ter + HP 2ter Ordg.
  mit Höhen: HP 4ter Ordg.
  mit jeweils Follower f. Entkopplung
  mit 3 × Klangregler
  mit Summierverstärker
- Filtercharakteristik lt. Tabelle v 'moodle2' [KN]
- Koeffiziententabellen lt. Tabelle v. 'moodle2' [KN]
- bei mind. einem Filter: Übertragungsfunktion manuell herleiten, Koeffizientenvergleich, Filtertabellenwerte einsetzen, Bauteilwerte ausrechnen.
- Abgabe erste Rechnung Stundenbeginn Mi-05Okt16
- Abgabe Dimensionierung, Gesamtschaltplan, Simulation (BodeDGM)

Teil V
Mixed analog/digital Teil

6 dB Rechnung

6 .1 dB-Rechng Bspe.XH

- welchem Faktor entsprechen _____ dB?
- wieviel dB entsprechen dem Faktor _____?
- was bedeutet "-6 dB *pro Oktave*" ?
- was bedeutet "-20 dB *pro Dekade*" ?

dB	Leistung P[W]	Spannung U, Stromstärke I
+20 dB	P= 100 fach	U= 10 fach
-20 dB	P= 1/100-stel	U= 1/10-tel
+10 dB	P= 10-fach	U= $\sqrt{10}$-fach (= 3.1623)
-10 dB	P= 1/10-tel (x 0.1)	U= $1/\sqrt{10}$-fach (= 0.31623)
+6 dB	P= 4-fach	U= 2-fach
-6 dB	P= 1/4-fach	U= 1/2-fach
+3 dB	P= 2-fach	U= $\sqrt{2}$-fach (=1.4142)
-3 dB	P= 1/2-fach	U= $1/\sqrt{2}$-fach (0.7071)
Beispiele:		
+ 1 = 10-9	10 /2 /2 /2 = 1.25 fach	
+ 2 = 12-10	16 /10 = 1.6 fach	
+ 4 = 10-6	10 /4 = 2.5 fach	
+ 5 = 10/2	$10^{1/2} \equiv \sqrt{10}$ = 3.16 fach	
+ 7 = 10-3	10 /2 = 5 fach	
+ 8 = 20-6-6	100 /4 /4 = 6.25 fach	
+ 9 = 6+3	4 mal 2 = 8 fach	
+11 = 20-6-3	100 /4 /2 = 12.5 fach	
+12 = 6+6	4 mal 4 = 16 fach	U= 4 fach
+13 = 10+3	10 mal 2 = 20 fach	
+14 = 20-6	100 /4 = 25 fach	U= 5 fach
+15 = 10+5	10 mal $\sqrt{10}$ = 31,6 fach	
+16 = 10+6	10 mal 4 = 40 fach	
+17 = 20-3	100 /2 = 50 fach	U= 7,...
+18 = 12+6	16 mal 4 = 64 fach	U= 8 fach
+19 = 10+9	10 mal 8 = 80 fach	U= *sim* 9 fach
+46 = 40+6	10e4 mal 4 = 40 000 fach	100*2= 200 fach

wenn 1mW ≡ 0 dBm, dann sind

- +3dBm ..... ? mW
- +5dBm ..... ? mW
- +6dBm ..... ? mW
- +10dBm ..... ? mW
- +20dBm ..... ? mW
- +13dBm ..... ? W
- +60dBm ..... ? W
- -3dBm ..... ? mW
- -5dBm ..... ? mW
- -6dBm ..... ? mW
- -10dBm ..... ? mW
- -100dBm ..... ? mW

- -73dBm ..... ? mW

6 .2 dB-Rechng Bspe.YW

a Ein Spannungsteiler soll eine Abschwächung a von -40 dB aufweisen und dabei für die speisende Quelle einen Innenwiderstand von 100 kΩ haben. Berechnen Sie die beiden Widerstände des Spannungsteilers.

b Wie groß ist die Abschwächung a eines Oszillografentastkopfes in dB, wenn dieser Tastkopf ein Teilungsverhältnis von 10:1 aufweist?

c Die Gleichtaktunterdrückung G eines Differenzverstärkers ist definiert als das Verhältnis der Differenzverstärkung v_D zur Gleichtaktverstärkung v_{GL}. Geben sie G als lineares Verhältnis der Verstärkungen an, wenn G (logarithmisch) 80dB, 100 dB und 120 dB beträgt.

d Wie groß ist die Leistungsverstärkung vP eines Verstärkers in dB, wenn der Verstärker bei 1mW Eingangsleistung eine Ausgangsleistung von 10W liefert?

e Ein Verstärker kann eine konstante Leistungsverstärkung nur in einem bestimmten Frequenzbereich liefern. Man gibt meist den Frequenzbereich an, in dem die Ausgangsleistung P_2 zu niedrigen und hohen Frequenzen hin auf 50 Prozent ihres Höchstwertes P_1 absinkt. Dies nennt man Leistungshalbwert. Welcher Pegel P in dB gehört zu diesem Leistungshalbwert?

f Fertigen Sie eine Tabelle zur Umrechnung des linearen Spannungsverhältnisses $v = {u_2}/{u_1}$ in das logarithmische Spannungsverhältnis in dB an für $v = 0.5, {1}/{\sqrt{2}}, 1, \sqrt{2}, 2, 10, 100 und 1000$.

linear	logarithmisch
0.5	
$1/\sqrt{2}$	
1	
$\sqrt{2}$	
2	
10	
100	
1000	

6 .3 XH: dB-Aufgaben

↪ (a) (s.YW) the input resistance of a voltage divider be 10k and its input-to-output attenuation be -40dB

solution: (1) R1+R2=10k
(2) $\frac{R1+R2}{R2}$ = 100-fach } ==> R1= 9k9, R2=0 k 1

tip: if $(\frac{R1+R2}{R2} = x)$ → $R1/R2 = (x-1)$

Bauernregel: Für Dämpfung um *Faktor n*
nimmsch R2 = ein 'n'-tel von R_{gesamt}
und den Rest mochsch R1

bei R_{gesamt} == 10k:
zB. -40dB ≙ 1/100 → R2= $1/100$ *10k = 100Ω
→ R1= 10k - 100Ω = 9k9
zB. -20dB ≙ 1/10 → R2= $1/10$ *10k = 1k
→ R1= 10k - 100Ω = 9k

bei R_{gesamt} == 100k:
zB. -40dB ≙ 1/100 → R2= $1/100$ *100k = 1k
→ R1= 10k - 100Ω = 99k

Spannungsteiler isch 1te Klass...

↪ (b) Dämpfung (attenuation) eines Tastkopfkabels 10:1 (von 10 auf 1) in dB?
solution: U von 10 auf 1: 0.1 ≙ **-20 dB**

↪ (c) the common mode rejection ration (CMRR) of an amplifier
is its amplification A of differential mode signals u_D in comparison to common mode signals u_{CM} (u_{GL}) $CMRR = A_D/A_{CM}$. Its measure is usually logarithmic dB (von David "**d**eci" **B**ÄNSCH, 2012a) The higher, the better. — give CMRR at 80dB, 100dB, 120dB as linear numbers A_D/A_{CM} =?.

↪ (d) Give the power ratio of an amplifier delivering 10W out from 1mW in in dB!

↪ (e) How much is the 50% reduced power "Leistungshalbwert" at the -3dB TP- or HP- cut-off-frequency (crossover frequency) given in dB?
Zusatzfrage (XH): Da HF Komponenten oft > 3 dB Welligkeiten aufweisen, wird dort die -6dB Grenzfrequenz (halbe Spannung) verwendet. Welcher Leistungsreduktion in % entspricht dieses?
Wieviel ist das bei 36dBm ?
Kommentiere:
Die Specs eines Speki enthalte *'$P_{in} < 20dBm$'*. Ein Komilitone messe den Leistungsfrequenzgang eines 2W Notsenders und sagt: "des passt scho..."

↪ (f) Give a list of linear voltage relations v and dB-measures for values v = 0.5, $1/\sqrt{2}$, 1, $\sqrt{2}$, 2, 10, 100 und 1000.

↪ Die *thermische Rauschleistung* P_Θ ohm'scher Widerstände ist $P_\Theta = k \cdot T$, $k_{Boltzmann} \ldots 1.38 * 10^{-}23$, T. . . abs.Temp. zB.300K pro Hz Bandbreite. Wie groß ist diese thermische Rauschleistung bei 25° Zimmertemperatur?
Lösung:
$25°C \equiv 298.15K \rightarrow P_R[dBm] = 10 \cdot \log kT + 30 = 10 \cdot \log(1.38 * 10^{-23} \cdot 298.15) + 30 \equiv -174dBm$

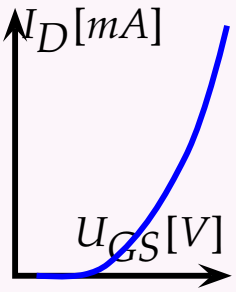
6 .4 PWM Übungen 21Dez'16

6 .4.1 PwrMosFET Datenblatt IRF820 (Vishay)

PARAMETER	SYMBOL	CONDITIONS	TYP.	UNIT
Gate-Source Threshold Voltage	$V_{GS(th)}$	$V_{DS} = V_{GS}, I_D = 250\mu A$	2.0-4.0	V
Drain-Source On-State Resistance	$R_{DS(on)}$	$V_{GS} = 10V, I_D = 1.5A$	3.0	Ω
Forward Transconductance	g_{fs}	$V_{DS} = 50V, I_D = 1.5A$	1.5	S
Input Capacitance	C_{iss}	$V_{GS} = 0V$, $V_{DS} = 25V$	360	pF
Output Capacitance	C_{oss}		92	pF
Reverse Transfer Cap.	C_{iss}		37	pF
Total Gate Charge	Q_g	$V_{GS} = 10V$, $I_D = 2.1A$, $V_{DS} = 400V$	24	nC
Gate-Source Charge	Q_{gs}		3.3	nC
Gate-Drain Charge	Q_{gd}		13	nC
Turn-On Delay Time	$t_{d(on)}$	$V_{DD} = 250V$, $I_D = 2.1A$, $R_G = 18\Omega$, $R_D = 100\Omega$	8.0	ns
Rise Time	t_r		8.6	ns
Turn-Off DelayTime	$t_{d(off)}$		33	ns
Fall Time	t_f		16	ns
Internal Drain Inductance	L_D	from lead to	4.5	nH
Internal Source Inductance	L_S	die contact	7.5	nH

6 .4.2 PwrMosFET Schaltzeiten

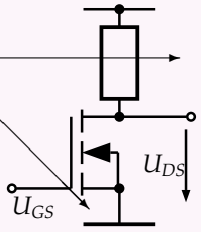
Obiger MosFET habe einen Transformator mit
Schaltfrequenz $f_s = 50kHz$ und
Tastverhältnis $p_D = 0.45$ (am Drain) an die
Betriebsspannung $U_q = 250V$ mit
Stromstärke $I_{Dmax} = 2.1A$
zu schalten.

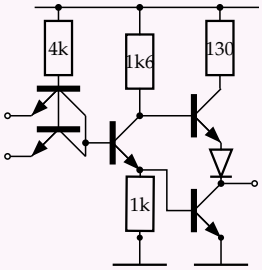


Mit welchem Tastverhältnis
p_G =?
ist er am Gate anzusteuern?

6 .4.3 PwrMosFET Schaltstrom

Das Kennlinienknie obigen MosFETs verlaufe von
($U_{GS} = 3.8V, U_{DS} = 250V, I_D = 0.001A$)
bis
($U_{GS} = 5.4V, U_{DS} = 2V, I_D = 2.1A$)
Dieser Umschaltvorgang sei in der
Schaltzeit $t_s = 50ns$
zu durchlaufen.





Wie groß sind die dazu erforderliche
Gate-Ladung $Q_{G,knie}$ =?
Gate-Stromstärke $I_{G,knie}$ =?

6 .4.4 NE555 PWM

Der Timerbaustein NE555 ist so zu beschalten, dass er am Anschluss Nr.3 ein PWM-Signal mit $f_s = 50kHz$, $p = 0.40$ ausgibt.

6 .4.5 NE555 PWM

Was bewirkt ein Widerstand $R = 10k$ von Anschluss Nr.3 nach Anschluss Nr.5 ?

6 .4.6 Übungsaufgaben PWM+SMPS

a) Ein PwrMosFET mit
$C_{GS} = 1500pF, C_{GD} = 300pF,$
$t_{d,on} = 40ns$ (*"Einschaltverzögerungszeit"*)
sei via Gatespannungsanstieg von
$U_{GS} = 3.0V$ auf $U_{GS} = 6.0V$
binnen $t_{ein} = 100ns$
von $U_{DS} = 350V/I_D = 1mA$
auf $U_{DS} = 3.0V/I_D = 5A$ zu schalten.
Berechne die mittlere Gatestromstärke I_G

b) Ein PwrMosFET mit
$C_{GS} = 1500pF, C_{GD} = 300pF,$
$t_{d,on} = 40ns$ (*"Einschaltverzögerungszeit"*)
sei via Gatespannungsanstieg von
$U_{GS} = 3.0V$ auf $U_{GS} = 6.0V$
binnen $t_{ein} = 100ns$
von $U_{DS} = 600V/I_D = 1mA$
auf $U_{DS} = 3.0V/I_D = 6A$ zu schalten.
Berechne die mittlere Gatestromstärke I_G

c) Ein PwrMosFET mit
$C_{GS} = 2000pF, C_{GD} = 280pF,$
$t_{d,on} = 40ns$ (*"Einschaltverzögerungszeit"*)
sei via Gatespannungsanstieg von
$U_{GS} = 3.0V$ auf $U_{GS} = 6.0V$
binnen $t_{ein} = 100ns$
von $U_{DS} = 600V/I_D = 1mA$
auf $U_{DS} = 3.0V/I_D = 7A$ zu schalten.
Berechne die mittlere Gatestromstärke I_G

d) Ein PwrMosFET mit
$C_{GS} = 2000pF, C_{GD} = 280pF,$
$t_{d,on} = 40ns$ (*"Einschaltverzögerungszeit"*)
sei via Gatespannungsanstieg von
$U_{GS} = 3.0V$ auf $U_{GS} = 7.0V$
binnen $t_{ein} = 100ns$
von $U_{DS} = 350V/I_D = 1mA$
auf $U_{DS} = 4.0V/I_D = 8A$ zu schalten.
Berechne die mittlere Gatestromstärke I_G

e) Ein PwrMosFET mit
$C_{GS} = 2000pF, C_{GD} = 300pF,$
$t_{d,on} = 30ns$ (*"Einschaltverzögerungszeit"*)
sei via Gatespannungsanstieg von
$U_{GS} = 3.0V$ auf $U_{GS} = 7.0V$
binnen $t_{ein} = 100ns$
von $U_{DS} = 600V/I_D = 1mA$
auf $U_{DS} = 4.0V/I_D = 8A$ zu schalten.
Berechne die mittlere Gatestromstärke I_G

f) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 30ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 6.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

g) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 40ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 7.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

h) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 50ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 8.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

i) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 55ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 5.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

j) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 60ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 6.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

k) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 70ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 6.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

l) Mit NE555 ist an Pin3 das PWM Signal
$t_{ein} = 60ns,$ $t_{aus} = 80ns,$
$V_{OL} = 0.2V,$ $V_{OH} = 6.0V$
zu erzeugen.
- Zeichne die relevante Beschaltung und
- dimensioniere die Bauteile.

7 Schaltungsanalyse

7 .1 Analyse-Übung

Im folgenden Übungsbeispiel (PDF-Datei) sind Funktion und Dimensionierung bis auf Bauteilebene zu ergründen:

→ "obelix: Hwe5-Sb51-Moodle-JN-Schaltungsanalyse.pdf"
(SMUE/shr/shr/pdf01)

→ "lokal: Hwe5-Sb51-Moodle-JN-Schaltungsanalyse.pdf"

Teil VI
Digital Teil

8 .1 Logik-Familien, Specs

s. "https://de.wikipedia.org/wiki/Kategorie:Digitale_Schaltungstechnik"
s. Böhmer Kap.15 "Digitale Verknüpfungs- und Speicherschaltungen" S.236 (Aufl.16), Interfacing 246/247

8 .2 Pegel: Logic Specs

Familie	V_{IL} max	V_{IH} min	V_{OL} max	V_{OH} min
TTL 74xx	0.8	2.0	0.4	2.4
LVTTL 3V3	0.8	2.0	0.4	2.4
TTL 74H	0.8	2.0	0.4	2.4
TTL 74S	0.8	2.0	0.5	2.5
TTL 74LS	0.8	2.0	0.5	2.7
TTL 74ALS	0.8	2.0	0.5	2.5
TTL 74F	0.8	2.0	0.5	2.5
74HC	1.35	3.15	0.26	3.98 @4mA
74HCT	0.8	2.0	0.26	3.98 @4mA
CMOS+5Vcc	1.5	3.5	0.5 @1mA	4.44 @1mA
→4001B@5Vcc	1.5	3.5	0.05 @1μA	4.95 @1μA
			0.4 @0.36mA	4.5 @0.36mA
LOCMOS HEF4000	1.5	3.5	0.05	4.95 @1μA
CMOS 2V5	0.7	1.7	0.2	2.3
CMOS 1V8	0.7	1.17	0.45	1.2
LVDS			1.0	1.4 @3.5mA/100Ω
ECL	-1.4	-1.2		

8 .3 Pegel: Standard TTL

	High	Low
Ausgang:	≥ 2.4V @-400uA	≤ 0.4V @16mA
Eingang:	≥ 2.0V @40uA	≤ 0.8V @-1.6mA

Lies: "Bei $I_a = -400\mu A$ muss der Ausgang noch mindestens $U_a \geq 2.4V$ liefern"
(Strom zählt immer ins IC hinein)

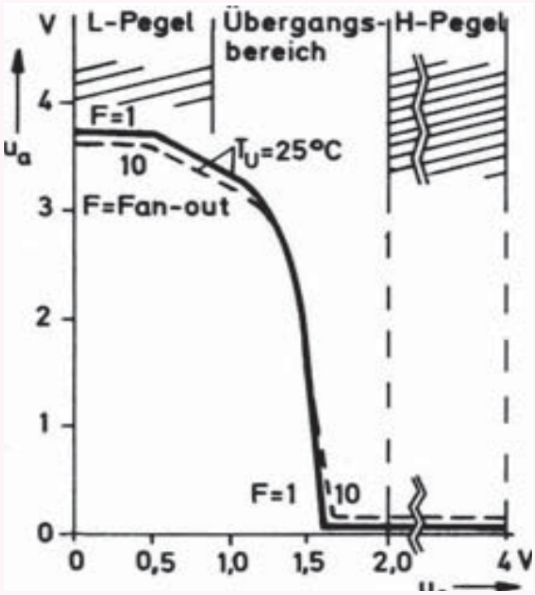
Fan-Out:
Die *Anzahl* von Eingängen (derselben Logikserie zB. 74HC, s.u.), die ein Ausgang maximal ansteuern kann (ohne Specs zu verletzen), nennt man ***fan-out*** $= {}^{I_{a,max}}/_{I_{e,max}}$.

Für *'standart TTL 7400'* ist *fan-out = 10* vorgegeben.
Das erklärt, warum in den Specs die Ausgangsströme 10-fach höher sind als die Eingangsströme.

8 .4 Pegel: Communication

	High	Low
IIC:	≥ 0.7 V_{DD}	≤ 0.3 V_{DD}
CAN(diff.)	≥ 3.5V @3mA	≤ 1.5V @3mA
USB1(diff.)	2.8. . . +3.3V	-2.8. . . -3.3V
USB2(diff.)	2.8. . . +3.3V	-2.8. . . -3.3V
USB3(diff.)	+17.78mA (0.4V)±10%	-17.78mA (-0.4V ±10%)
D+ or D- line	≤ 0.3V	≥ 2.8 – 3.0V
RS232-RX:	≤ -3.0V	≥ +3.0V
RS232-TX:	≤ -5.0V @3kΩ	≥ +5.0V @3kΩ

8 .5 TTL Inverter Übertragungskennlinie



8 .6 rise/fall times

t_{rise}, t_{fall} von 10% bis 90% der Endpegel

[...] Anstiegs- und Abfallzeiten beschreiben in der Digitaltechnik und bei Schalttransistoren die beim Umschaltvorgang charakteristischen Zeiten, in denen das Signal nicht mehr den alten und noch nicht den neuen definierten Logikpegel („0“ bzw. „1“) bzw. Schaltzustand innehat. Siehe auch Flankensteilheit.

In der Digitaltechnik werden hierbei meistens die im ungünstigsten Fall (worst case) garantierten Zeiten genannt. Sie beschreiben die Zeit, die ein Signal (beispielsweise in einem Computerprozessor) zum sicheren Umschalten zwischen den beiden binären Zuständen benötigt. Die für ein Bauteil spezifizierten Anstiegs- und Abfallzeiten sind oft keine Messwerte, sondern ein Merkmal des Bausteins bzw. der betrachteten Logikfamilie bei bestimmten Parametern (Betriebsspannung, Temperatur), das durch das Design bzw. den Herstellungsprozess gesichert ist.

Sämtliche Digitaltechnik basiert letztlich auf analog arbeitenden Schaltungselementen, die für die Bearbeitung digitaler Signale optimiert sind. Dabei ist zu beachten, dass man bei den digitalen Signalen „1“ bzw. „0“ zwar „Strom ein“ bzw. „Strom 0“ unterstellt, in der Praxis aber meistens mit Spannungs- und Strompegeln gearbeitet wird, die ungleich dieser Idealwerte sind. Weiterhin ist das Verhalten der Schaltung bei Zustandswechsel meistens unsymmetrisch, weshalb Anstiegs- und Abfallzeit dann unterschiedlich lang sind.

Für ein Logiksignal in einer Schaltung ist es notwendig, Schwellenwerte festzulegen. Für logisch „0“ wird bei TTL beispielsweise ein zulässiger Bereich von 0. . . 0,4 V festgelegt und als logisch „1“ ein Bereich von 2. . . 5 V. Die tatsächliche Schaltschwelle der Logikbausteine (ca. 1,4 V) liegt im „verbotenen Bereich“ zwischen diesen beiden Werten und daher im Bereich der Anstiegs- und Abfallzeit. Daraus ergibt sich ein besonders schnelles Durchlaufen des verbotenen Bereiches, was für viele Logikbausteine wichtig und oft auch mit einer minimal zulässigen Spannungsanstiegs-Geschwindigkeit spezifiziert ist.

Anstiegs- und Abfall-Zeit beschreiben die Zeitintervalle, in denen dieser undefinierte „verbotene“ Zustand während des Umschaltens auftritt. Wichtig ist das insbesondere bei flankengetriggerten Schaltungen, d. h. Schaltungen, die auf die Änderung des Signals reagieren (z. B. flankengetriggertes Flipflop). Andernfalls kann es zu Fehlfunktionen kommen; siehe Race condition.

Um eine Sicherheit gegen Störungen zu erhalten, gelten die erlaubten Pegel auch für die Ausgänge dieser Bausteine. Beispielsweise werden für einen Gatterausgang in der klassischen TTL-Technik max. 0,4 V für „0“ und min. 2,4 V für „1“ garantiert.

Sehr kurze Anstiegs- und Abfallzeiten im Signal bedeuten auch, dass im Spektrum des Signals sehr hohe Frequenzanteile vorhanden sind, die zur Aussendung (Abstrahlung) von elektromagnetischen Wellen führen. Durch diese Störsignale können andere Schaltungsteile in ihrer Funktion beeinflusst werden. Um die elektromagnetische Verträglichkeit sicherzustellen, werden deshalb die Ausgänge von Digital- und Treiberschaltungen so ausgelegt, dass die Anstiegs- und Abfallzeiten nur so kurz wie unbedingt nötig sind. Dazu wird die Flankensteilheit (slew rate) des Ausgangstreibers begrenzt.

In der Messtechnik sowie zur Charakterisierung analoger und digitaler Schaltungen werden zur Spezifikation der Zeiten meistens die Werte von 10 % bzw. 90 % des Schaltpegels bzw. Sollsignales definiert. Auch Schalttransistoren und andere leistungselektronische Bauteile werden damit charakterisiert. Analoge Verstärker, Schaltverstärker, Leuchtdioden, Laser, Photodioden und Fototransistoren werden ebenfalls durch Anstiegs- und Abfallzeiten charakterisiert, die sie als Antwort auf eine Sprungfunktion liefern. Wenn ein exponentieller Verlauf $U(t) = e^{\frac{t}{\tau}}$ *angenommen wird, beträgt die Anstiegs- und Abfallzeit für einen Tiefpass 1. Ordnung oder vergleichbare Systeme jeweils*

$t_{fall} = t(10\%) - t(90\%) = 2.3\tau - 0.1\tau = 2.2\tau$ *[...]*

(`https://de.wikipedia.org/wiki/Anstiegs-_und_Abfallzeit` 14Feb17)

8 .7 Propagation Delay

[...] Als Gatterlaufzeit (engl. propagation delay) bezeichnet man in der Digital- und Elektrotechnik die für die verwendeten Bauteile (Gatter) charakteristische Laufzeit von Signalen von einem Eingang zu einem Ausgang des Gatters. Diese Zeit wird mit t_P *oder* t_{PD} *abgekürzt, oder richtungsabhängig mit* t_{PLH} *für steigende Flanken bzw.* t_{PHL} *für fallende Flanken am Ausgang.*

Gatter als diskrete Bauelemente haben Laufzeiten im Bereich weniger Nanosekunden (ns) bis über 100 ns, siehe Logikfamilie. Die Zeiten werden von den Herstellern in den Datenblättern für bestimmte Bedingungen angegeben, meist für eine dem Fan-Out entsprechende kapazitive Last und eine hohe Temperatur, manchmal getrennt für verschiedene Versorgungsspannungen. Meist sind je drei Werte angegeben, Minimalwert, typischer Wert und Maximalwert. Bei den Grenzen handelt es sich um Zusicherungen des Herstellers. Sie sind großzügig bemessen, da die Hersteller höchstens Stichproben messen.

Präzise gemessen werden Laufzeiten über die Frequenz eines Ringoszillator. Die Zeiten streuen innerhalb einer Charge und unterscheiden sich manchmal deutlich von Charge zu Charge und zwischen Herstellern.

Innerhalb integrierter Schaltungen sind Gatter für kleinere Spannungen und Ströme ausgelegt, die Eingänge haben eine geringere Kapazität, weil ohne Schutzbeschaltung. Daher sind die Gatterlaufzeiten viel kleiner. Für die auf Gatterebene konfigurierbaren FPGAs liegen sie im Bereich einiger 10 bis 100 Picosekunden, innerhalb von Prozessorkernen teilweise weit unter einer Picosekunde.

Zu Gatterlaufzeiten kommen Laufzeiten auf Signalwegen hinzu. Konkrete Methoden zur Ermittlung von Gesamtlaufzeiten durch beliebige Netzwerke stellt die sog. Laufzeittoleranzrechnung bereit. Die Folge einer Nichtbeachtung der Gatterlaufzeiten können Timingverletzungen und Glitches in der Schaltung sein. [...]

(`https://de.wikipedia.org/wiki/Gatterlaufzeit` 14Feb17)

TBD. (bedeutet 'to be done' ≡ in Arbeit)

Impedanz. . .

8 .7.1 Glitch

kurzes, zwischenzeitliches Umschalten auf *'falsche'* Logik-Pegel (in Digitalsystemen; zB. aufgrund Laufzeitunterschieden, typisch etwa bei asynchronen Johnson-Zählern, Arithmetikbaugruppen (zB. Addierwerk), asynchronen Schaltwerken udgl.) Synchronisierung mit einem Taktsignal dient als Gegenmaßnahme.

[...] In der Elektronik bezeichnet man mit Glitch eine ***kurzzeitige Falschaussage in logischen Schaltungen*** *und temporäre Verfälschung einer booleschen Funktion.*

Diese tritt auf, weil die Signallaufzeiten in den einzelnen Gattern niemals vollkommen gleich sind. *Diese Verfälschung wird daher auch als Race*

Condition bezeichnet. *Die Anfälligkeit für Glitches steigt mit der Komplexität, der Geschwindigkeitserhöhung und der Verkleinerung der Schaltungen, kann aber auch bereits bei sehr einfachen Schaltungen vorhanden sein. Sie stellen ein wesentliches Problem bei der Entwicklung moderner elektronischer Schaltungen und schneller Mikroprozessoren dar, das war aber auch schon bei der älteren elektromechanischen Relaistechnik so. [...]*

(`https://de.wikipedia.org/wiki/Glitch_(Elektronik)` 14Feb17)

8 .7.2 Spike

kurzes **Störsignal**, oft ausserhalb der Logikpegel, aufgrund Nebensprechens oder Stromversorgungs-/bedarfs-schwankungen (Schaltvorgänge) im Zusammenwirken mit parasitären Reaktanzen des mechanischen Aufbaues
In electrical engineering, spikes are fast, short duration electrical transients in voltage (voltage spikes), current (current spikes), or transferred energy (energy spikes) in an electrical circuit. Fast, short duration electrical transients (overvoltages) in the electric potential of a circuit are typically caused by

- Lightning strikes
- Power outages
- Tripped circuit breakers
- Short circuits
- Power transitions in other large equipment on the same power line
- Malfunctions caused by the power company
- Electromagnetic pulses (EMP) with electromagnetic energy distributed typically up to the 100 kHz and 1 MHz frequency range.
- Inductive spikes ('inductive kickback')

In the design of critical infrastructure and military hardware, one concern is of pulses produced by nuclear explosions, whose nuclear electromagnetic pulses distribute large energies in frequencies from 1 kHz into the gigahertz range through the atmosphere.

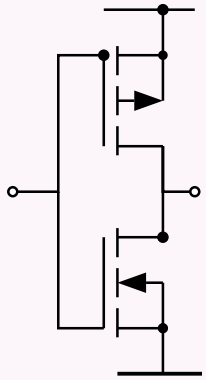
The effect of a voltage spike is to produce a corresponding increase in current (current spike). However some voltage spikes may be created by current sources. Voltage would increase as necessary so that a constant current will flow. Current from a discharging inductor is one example. For sensitive electronics, excessive current can flow if this voltage spike exceeds a material's breakdown voltage, or if it causes avalanche breakdown. In semiconductor junctions, excessive electric current may destroy or severely weaken that device. An avalanche diode, transient voltage suppression diode, transil, varistor, overvoltage crowbar, or a range of other overvoltage protective devices can divert (shunt) this transient current thereby minimizing voltage. While generally referred to as a voltage spike, the phenomenon in question is actually an energy spike, in that it is measured not in volts but in joules; a transient response defined by a mathematical product of voltage, current, and time. Voltage spikes may be created by a rapid buildup or decay of a magnetic field, which may induce energy into the associated circuit. However voltage spikes can also have more mundane causes such as a fault in a transformer or higher-voltage (primary circuit) power wires falling onto lower-voltage (secondary circuit) power wires as a result of accident or storm damage. Voltage spikes may be longitudinal (common) mode or metallic (normal or differential) mode. Some equipment damage from surges and spikes can be prevented by use of surge protection equipment. Each type of spike requires selective use of protective equipment. For example a common mode voltage spike may not even be detected by a protector installed for normal mode transients. An uninterrupted voltage increase that lasts more than a few seconds is usually called a "voltage surge" rather than a spike. These are usually caused

by malfunctions of the electric power distribution system.

(`https://en.wikipedia.org/wiki/Voltage_spike` 14Feb17)

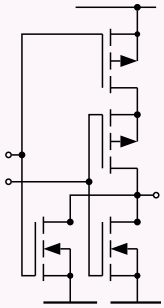
8 .8 TTL

8 .8.1 NAND

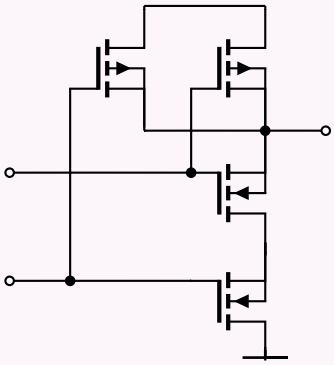


8 .9 CMOS

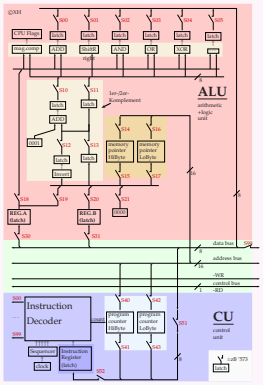
8 .9.1 CMOS Inverter



8 .9.2 CMOS NAND

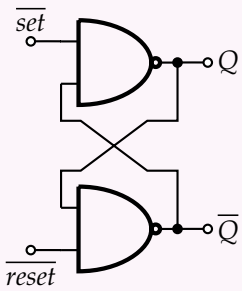


8 .9.3 CMOS NOR

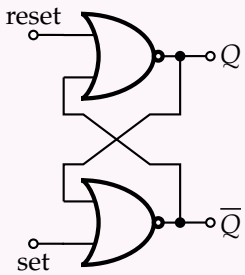


8 .10 Flipflop

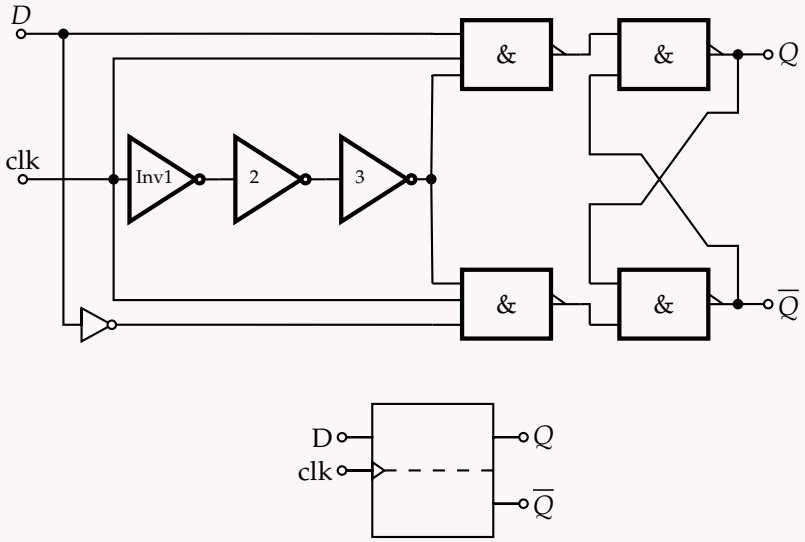
NAND-RS:



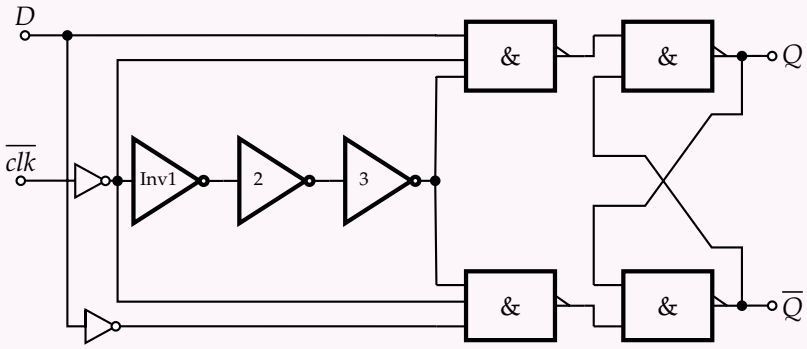
NOR-RS:

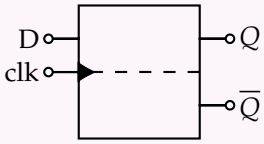


DFF positiv flankengetriggert (edge triggered):

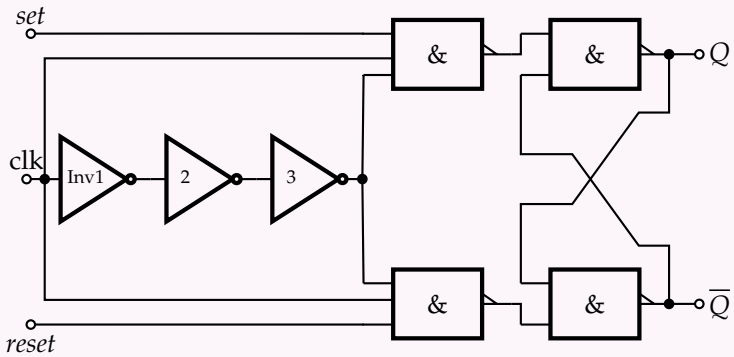


DFF negativ flankengetriggert (edge triggered):



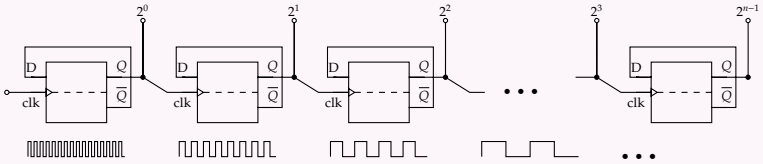


RSFF positiv flankengetriggert (edge triggered):

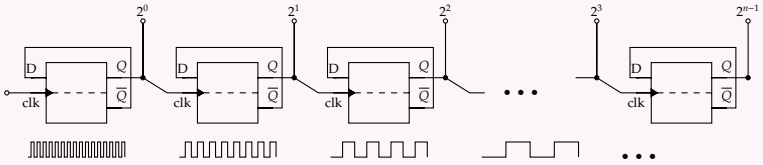


8 .11 Binärzähler/-Teiler

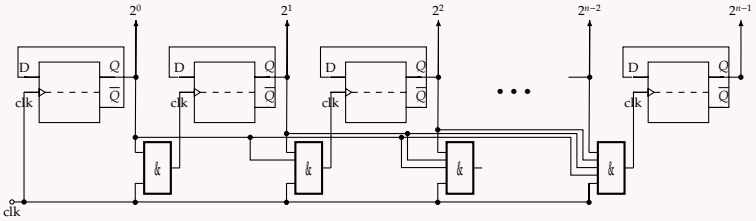
asynchronous binary down counter:



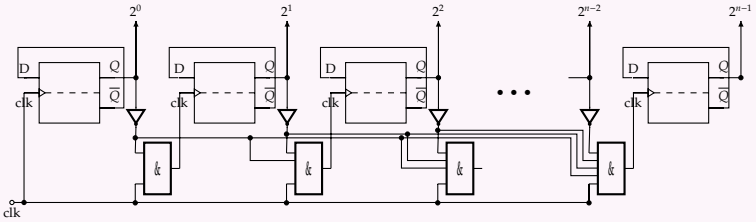
asynchronous binary up counter:



synchronous binary up counter:



synchronous binary down counter:



Ctr 7493-193-293-393 /Rc1

8 .12 74HC393

'74HC/HCT393 Dual 4-bit binary ripple counter':

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
1, 13	$1\overline{CP}$, $2\overline{CP}$	clock inputs (HIGH-to-LOW, edge-triggered)
2, 12	1MR, 2MR	asynchronous master reset inputs (active HIGH)
3, 4, 5, 6, 11, 10, 9, 8	1Q0 to 1Q3, 2Q0 to 2Q3	flip-flop outputs
7	GND	ground (0 V)
14	VCC	positive supply voltage
t_{PLH}, t_{PHL}	@Vcc=5V, C_L=15pF	5..20 ns
f_{max}		HC:99MHz, HCT:53MHz
C_i	input capacitance	3.5pF

weitere Details s.Datenblatt

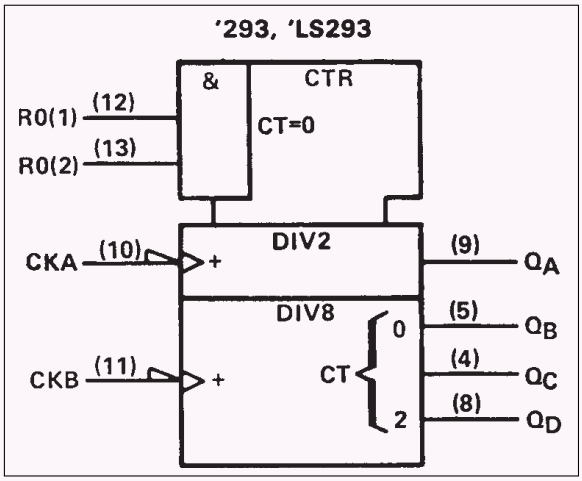
(Datenblatt 'Product specification File under Integrated Circuits, IC06 Philips Semiconductors December 1990' 28Feb'17)

8 .13 74HC293

'293, LS293 ... 4 Bit Binary Counters'

'[...] the '293 are electrically and functionally identical to the '93. Only the arrangement of the terminals has been changed [...]'

'[...] contains four master-slave flip-flops [...] to provide a divide-by-two counter and a three stage binay counter [...]' '[...] All of these counters have a gated zero reset [...]'



(Datenblatt 'Texas Instruments 1988' 'Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265 Copyright © 2009, Texas Instruments Incorporated', 28Feb'17)

weitere Details s.Datenblatt

8 .14 74HC193

'Presettable synchronous 4-bit binary up/down counter'

PIN DESCRIPTION

PIN NO.	SYMBOL	NAME AND FUNCTION
3, 2, 6, 7	Q0 to Q3	flip-flop outputs
15, 1, 10, 9	1D0 to D3	data inputs
4	$1CP_D$	count down clock input
5	$1CP_U$	count up clock input
8	GND	ground (0 V)
11	$\overline{PL}$	asynchronous parallel load input
12	$\overline{TC_U}$	carry output active low
13	$\overline{TC_D}$	borrow output active low
14	MR	asynchronous master reset input
16	VCC	positive supply voltage
t_{PLH}, t_{PHL}	@Vcc=5V, C_L=15pF	20 ns
f_{max}		HC:45MHz, HCT:47MHz
C_i	input capacitance	3.5pF

weitere Details s.Datenblatt

(Datenblatt 'Product specification File under Integrated Circuits, IC06 Philips Semiconductors December 1990' 28Feb'17)

8 .15 74HC93

'4-bit binary ripple counter [...] The 74HC/HCT93 are 4-bit binary ripple counters. The devices consist of four master-slave flip-flops internally connected to provide a divide-by-two section and a divide-by-eight section. Each section has a separate clock input (CP0 and CP1) to initiate state changes of the counter on the HIGH-to-LOW clock transition. State changes of the Qn outputs do not occur simultaneously because of internal ripple delays. Therefore, decoded output signals are subject to decoding spikes and should not be used for clocks or strobes. [...]'

PIN NO.	SYMBOL	NAME AND FUNCTION
1	$\overline{CP_1}$	clock input 2nd, 3rd and 4th section (HIGH-to-LOW, edge-triggered)
2,3	MR	asynchronous master reset (active HIGH)
4,6,7,13	n.c.	not connected
5	VCC	positive supply voltage
10	GND	ground (0 V)
12,9,8,11	Q0 to Q3	flip-flop outputs
14	$\overline{CP_0}$	clock input 1st section (HIGH-to-LOW, edge-triggered)
t_{PLH}, t_{PHL}	@Vcc=5V, C_L=15pF	HC:12ns HCT:15ns
f_{max}		HC:100MHz HCT:77MHz
C_i	input capacitance	3.5pF

(Datenblatt 'Product specification File under Integrated Circuits, IC06 Philips Semiconductors December 1990' 28Feb'17)

weitere Details s.Datenblatt

8 .16 4060 Ctr mit Oscillator

s. Unterricht 18Feb'19 ganzeKlasse u.Datenblatt:

'14-stage binary ripple counter'
- RC-Oszillator mit C, R1, R2
- Xtal-Oszillator

8 .17 4017 4-Bit-Ctr mit 1-aus-10 Decoder

s. Unterricht 18Feb'19 ganzeKlasse u.Datenblatt

Schrittfolge-Geber

8 .18 Zählerbeschaltungen

TBD. (bedeutet 'to be done' ≡ in Arbeit)

(wird von ZI beigebracht . . .)

8 .19 4015 2fach 4-Bit-ShiftReg.

s. Unterricht 18Feb'19 ganzeKlasse u.Datenblatt

- Kombination zu 8-Bit Schieberegister

- Kombination zu 12-Bit Schieberegister

8 .20 4046 PLL

s. Unterricht 18Feb'19 ganzeKlasse u.Datenblatt:

1: VCO+PhaseCompatator f. FM-Demodulation
2: VCO f. FM-Modulation

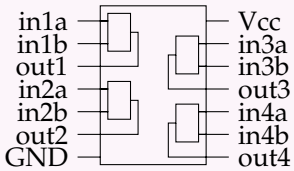
8 .21 Max170 serial ADC

s. Unterricht 04Feb'19, Unterricht 18Feb'19 ganzeKlasse u.Datenblatt

'auto' Modus, Startbit+Datenbits, Aufloesung in bit und dB, Wäge(SAR)- Verfahren

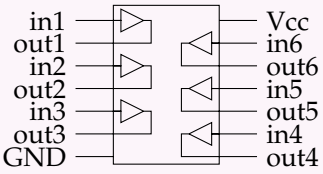
8 .22 Gatter

7400 quad NAND und
CD 4011 quad NAND 4 Stk 2-fach-NAND



7402 NOR 4 Stk 2-fach-NOR
7486 quad EXOR 4 Stk 2-fach-XOR
CD 4001 dual NOR 4 Stk 2-fach-NOR

7414 hex inverter 6 Stk Inverter mit Schmitt-Trigger-Eingang (Hysterese)
CD 40106 hex schmitt trigger inverter wie 7414
CD 4049 hex inverter 6 Stk CMOS-Inverter
CD 4069 hex inverter 6 Stk CMOS-Inverter



8 .23 Flipflop

7474 dual DFF 2 Stk D-Flipflop mit Set+Reset

CD 4013 dual DFF wie 7474

CD 4027 dual JKFF

8 .24 Latches + Register

74373, 74573 8-Bit-Latch

74374, 74574 8-Bit-Register

8 .25 Counter

CD 4020 - 14 stage binary counter

CD 4024 - 7 stage binary counter

CD 4040 - 12 stage binary counter

CD 4060 - 14-stage ripple-carry binary counter/divider and oscillator

CD 4029 - 4-Bit Presettable Binary/Decade Up/Down Counter w/parallel in+out

8 .26 analog switch

CD 4016 - quad analog switch SPST

CD 4066 - quad analog switch SPST

CD 4051 1x8 analog switch

CD 4052 2x4 analog switch

CD 4053 3x2 analog switch

CD 4067 1x16 analog switch

8 .27 div.

CD 4046 PLL
C744046 PLL
74HC9046 PLL
74HCT9046A PLL
CD 4008 adder 4-Bit binary adder
4007
4008
4017-HC4017
4020-24-40
4028-HEF4028B
4029
4063
4070
4076
4093
4095JKFF
4516
74107
74112
74123
74132
74138
7413
744060
74688
74HC109
74HC13
74HC191
74HC193
74hc245
74HC393

74HC93
74HCU04
74LS293
74LS380
74LS90
74LS93

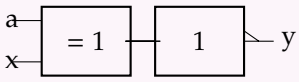
DM74S00-Tk91.pdf
./Comm: MCP2515-CAN-Rl71.pdf
./Eth: enc28j60.pdf
./FTDI: DS-FT232BL.pdf DS-FT245BL.pdf
./I2C: PCF8583-1.pdf
./LCD: HD44780-1.pdf REVERSE-Qk3.txt
./Memory: 2764.pdf 28C16.pdf 4164DRAM-1.pdf 6116-HM6116.pdf 6264.pdf 93LC46-56-66.pdf AT28C16.pdf
./RS232: 16450UART.pdf 16550D-UART.pdf 6402.pdf 82050.pdf 8250A-UART.pdf Max232.pdf
./ShiftReg: 4014.pdf 4015.pdf 4021-cd4021.pdf 74164.pdf 74165.pdf 74166.pdf 74198-sn74198.pdf 74299-2.pdf 7491-ETC.pdf
./Switch: 4053b.pdf 4066.PDF 4067.PDF DG211a.pdf DG306.pdf DG308a.pdf MAX392-a.pdf MAX4541CPA-3.pdf MAX4544CPA.pdf MAX4614anasw.pdf
msp430g2231.pdf
6502
./uC/6502: 6502-RdR4-rockwell
./uP: AR9331IntroToCortex-M3.pdf msp430.PDF
./uP/Mips24k: 24K - LinuxMIPS.html
./uP/NextChip:
./uP/Z80: z80.pdf

9 VHDL Einleitung

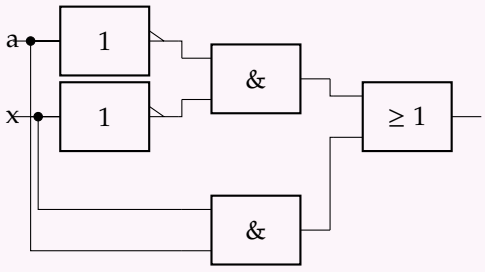
Die Probleme, die beim Einsatz von VHDL auftreten, dürfen jedoch nicht verschwiegen werden. Es handelt sich um eine sehr mächtige Sprache, die erst nach längerem praktischen Einsatz richtig beherrscht wird. Der Einstieg ist insbesondere für diejenigen Hardwareentwickler schwierig, die noch nicht intensiv mit einer Programmiersprache gearbeitet haben. Die psychologische Barriere darf dabei nicht unterschätzt werden. Hinzu kommt, daß es mit der Einführung der SSprache"VHDL allein nicht getan ist: Die darauf basierende Entwurfsmethodik erfordert eine neue Arbeitsweise, ein Überdenken gewohnter Schemata und nicht zuletzt die Verwendung neuer Werkzeuge.
[...] Eine [...] Herausforderung stellt allerdings die Tatsache dar, daß der durch verschiedene Syntheseprogramme unterstützte VHDL-Sprachumfang eingeschränkt und nicht identisch ist.
Ein gravierendes Problem ist auch die starke Abhängigkeit des Syn- theseergebnisses von der Qualität der VHDL-Beschreibung, mit dem Schlagwort "what you write is what you get"treffend beschrieben.
(aus *Karlsruhe und Erlangen, im März 1994 Gunther Lehmann, Bernhard Wunder, Manfred Selz*)

- VHDL ist eine **Sprache** *('very high speed hardware definition language')*
  (*Verilog* ist die bedeutendste Konkurrenz)
- VHDL beschreibt *digitale Schaltungen*
  (aus *Leitungen* und *Gattern*)
  Leitungen heissen *'Signals'* und
  Gatter heissen *Logical Operators*: and
  or
  nand
  nor
  xor
  not
  ('xnor' haben sie angeblich nicht; da muss man 'not (a xor b)' verwenden)
- $\boxed{x = a}$
  ist *KEINE Zuweisung!*, sondern der logische *Aquivalenz*-Operator
  !(a xor x)
  mit dem booleschen DNF-Ausdruck[8]
  $y = (a \wedge x) \vee (\overline{a} \wedge \overline{x})$
  aus `'y <= (x = a);'` baut es:

wenn mein PLD XOR besitzt:



wenn mein PLD keine XOR hat:



"kleine" Verknüpfungen können also schon stattliche Schaltungen erzeugen.

- VHDL ist case-insensitive: *'Xeins'* ist gleich *'xEins'*
  (man soll die Schreibweisen aber *nicht mischen*!)

[8] *DNF* . . . disjunktive Normalform (*'sum of products'*) der *'boolean expression'*

- ***x"3"*** ist eine hexadezimal-Zahl
- ***o"7"*** ist eine oktal-Zahl
- ***b"101"*** ist eine Binärzahl

Eingangs-	Leitungen	**in** *port*
Ausgangs-	heissen	**out** *port*
innere **Verbindungs-**		***Signal***

- Anweisungen

ausserhalb	eines *'Process'*-	→	**parallele** (kombinatorische) Schaltungen (*'concurrent statement'*)
innerhalb	Blocks erzeugen	→	**sequentielle** (mit Flipflops und Takt) Schaltungen

[9]

[9] wie wir wissen, entstehen Flipflops ja aus rückgekoppelten Gattern

- das VHDL-"Programm" besteht aus
  - "Head"

```
library IEEE;
use IEEE.std_logic_1164.all;
entity meinPRODUKT is port (
      e1, e2, e3 : in std_logic;
      y1 : out std_logic
);
end meinPRODUKT;
```

  - "Body"

```
architecture Innenleben of meinPRODUKT is
begin
      y1 <= NOT (e1 AND e2 AND e3);
end Innenleben;
```

    component
    function
    procedure
  - Simulationsteil mit *stimulus*-Test-Input
- die erzeugte Schaltung (die *'Synthese'*) und die Optimierung hängen sehr stark vom Ziel-PLD ab!

9 .1 Process

VHDL presents a particular structure named process that defines the limits of a code that will be simulated (or executed) if and only if any of the signals included in its sensitivity list has been modified in a previous simulation step.

A process features the following structure:

```
process ( sensitivity_list )
  -- Assignments to variables
  -- This is optional and, in general, not recommended
begin
  -- Conditional statements
  -- Assignments to variables or to signals
end process;
```

Process statements are VERY used in VHDL programming, since, for software programmers, it is very easy to code the behavior of a hardware circuit as if it was a software program. However, this is an important drawback for beginners, since the software-like description of the behavior of the circuit may not actually synthesize into hardware. For this reason, a number of good coding practices exist, which are directly related with the properties of the process statement. One should be VERY aware of them in order to code a hardware circuit that simulates and synthesizes correctly.

Property I

Statements existing inside a process only run in the instant 0 of simulation OR if any of the signals of the sensitivity list changes.

Problem:
The result of the simulation of the circuit may be unexpected due to the "malign effect" of the sensitivity list.

Solution:
The sensitivity list MUST include all the signals that are read inside the process.
`(signal_written <= signal_read)`.

Let us explain this point by means of an example. In this example, no value is assigned to C until the instant 10 ns, although B changes at 5 ns. This happens because the code inside the process is not executed unless A changes (this happens at 10 ns). However, at the hardware level, one would expect C to take the value of A as soon as B changes to 1 (at 5 ns).

```
process(A)
begin
 if B= '1' then
   C <= A;
 end if
end process;
```

Thus, following the solution proposed above, the correct code should be as follows:

```
process(A,B)
begin
 if B= '1' then
   C <= A;
 end if
end process;
```

Property II

Assignments to signals made inside a process have memory.

Problem:

If, in a given simulation step, a process is executed and as a consequence, a signal S is modified; and if in a subsequent simulation step, the process is again executed but S is not modified inside the code of the process, then C will conserve the value assigned in the first process execution. This may lead to an expected behavior because of the ”malign effect” of the process memory.

Solution:

Conditional statements inside processes MUST assign a value to the same set of signals in any of the branches of the statements. In addition, unless it is strictly forbidden at the design level, all the conditions MUST have their corresponding else branch. This is explained in the examples. The first code includes an else branch, which means that there is a by-default value for C (in this example, at 5 ns). However, the second code does not include this else branch. Hence, at 5 ns, both for Case_1 and Case_2, the value of C differs in both codes (”00” vs. ”10” and ”00” vs. ”11”, respectively). The reason is that processes have memory and in this case, the input combination A = ”11”; B = ”10” does not match any branch in the second code. Hence, the value for C at 5 ns (”10” for Case_1 and ”11” for Case_2) is the same one as in the previous simulation step (i.e., at 0 ns). In addition, note that the output value for C is ”00” in both Case_1 and Case_2 at 5 ns (in both cases, A = ”11” and B = ”10”). This is correct; however, this is not true for Case_2. The code in Figure 3.5 is another example of an incomplete conditional statement. in this case, two different values at 5 ns are obtained for C and D in Case_3 and Case_4, even though the input values are exactly the same in both cases (A = ”10”; B = ”10”). This shows us that it is EXTREMELY important to make sure that not only the if-then-else statement has else branch, but also that the very same set of signals are assigned in all the branches. In this case, the problem arises because the if branch assigns a value to C, but not to D; whereas the elsif assigns a value to D, but not to C.

```
-- CONDITIONAL COMPLETE:
process(a,b)
begin
  if a = b then
    c <= a or b;
  elsif a<b then
    c <= b;
  else
    c <= "00";
  end if;
end process;
```

```
--CONDITIONAL INCOMPLETE:
process(a,b)
begin
  if a = b then
    c <= a or b;
  elsif a < b then
    c <= b;
```

```
  end if
end process;
```

Property III

All the statements inside a process run in parallel, in a similar way as the statements outside a process do. However, if inside a process a signal is given a value at two different points, the final result will be the one of the last assignment, similarly as what happens in software programming languages. This may turn problematic and not synthesizable if not properly coded.

Solution:

It is convenient to double-check that a signal is not assigned twice in the same process (this can be done in two different branches of an if-then-else statement). In the following example, at 0 ns and at 10 ns, two values are assigned to C. When the process finishes, C takes the last assigned value, the one in the if and elsif branches, respectively. At 5 ns, C is assigned only one value ”00”, which is its final value.

```
-- Conditional incomplete ?
process (a,b)
begin
  if a = b then
    c <= a or b ;
  elsif a < b then
    d <= b ;
  else
    c <= "00" ;
    d <= "11" ;
  end if
end process;
```

Property IV

All process statements run in parallel.

Problem:

In two processes P1 and P2 modify the same signal, then it is impossible to know its actual value. (The one assigned by P1 or P2?).

Solution:

One should always double-check that a signal is not modified in two or more different processes. In that case, a possible solution is to merge the involved processes.

Property V

The values of all the signals that are modified inside a process are not updated until the whole process finishes.

Problem:

If the sensitivity list is not correctly coded, the update of a signal may be postponed one or several simulation events. This can be observed in the example of Figure 3.7, where only A is included in the sensitivity list. The example in Figure 3.8 does not present this problem any more.

Solution:

As in Property IV, always double-check that a signal is not modified in two or more different processes.

. . .

9 .2 Conditional

These statements are assignment statements to variables that may or may not be based on a condition. As previously pointed out, they MUST be placed inside a process. The following conditional statements exist in VHDL:

9 .2.1 IF-THEN-ELSE

```
process(sensitivitylist)
begin
  if condition_1 then
    -- assignments
  elsif condition_2 then
    -- assignments
  else
    -- assignments
  end if;
end process;
```

```
-- Example IF-THEN-ELSE
process(control,A,B)
begin
  if control = ''00'' then
    output <= A + B;
  elsif control = ''11'' then
    output <= A - B;
  else
    output <= A;
 end if;
end process;
```

It is possible to chain as many if-then-else statements as desired, as in software description languages, such as Pascal, C, Java . . .

TIP: if-then-else statements should always have an else. In addition, it is convenient to assign values to the same signals in each one of the branches of the statement, even if the value of some signals should be a don't care.

9 .2.2 CASE-WHEN

```
process (sensitivitylist)
begin
  case signal_condition is
    when value_condition_1 => -- assignments
    ...
    when value_condition_n => -- assignments
    when others => -- assignments
  end case;
end process;
```

In this case, assignments may also be if-then-else statements. The when others clause must appear in the statement, but it is not necessary to write any assignment associated to it.

```
-- Example CASE-WHEN
process(control,A,B)
begin
  case control is
    when ''00'' => result <= A+B;
    when ''11'' => result <= A-B;
    when others => result <= A;
  end case;
end process;
```

As in other software programming languages, several types of loops are possible:

9 .2.3 FOR-LOOP

```
process (sensitivitylist)
begin
  for var_loop in range loop
     -- assignments
  end loop ;
end process;
```

The range can be defined as 0 to N or as N downto 0.

```
-- Example FOR-LOOP:
process(A)
begin
  for i in 0 to 7 loop
    B(i+1) <= A(i);
  end loop ;
end process;
```

9 .2.4 WHILE-LOOP

```
process (sensitivitylist)
begin
  while condition loop
    -- assignments
  end loop;
end process;
```

```
-- Example WHILE-LOOP
process(A)
  variable i : natural := 0;
begin
  while i < 7 loop
    B(i+1) <= A(i);
    i := i+1;
  end loop;
end process;
```

For Xilinx^{TM} users: For loops are supported as long as the index range is static (0 to N or N downto 0, where N is a constant) and the loop body does not contain any wait statement. In general, while loops are not supported.

9 .2.5 Discussion about using signals vs. variables

Signals are used to connect different components in a circuit, whereas variables are used inside process **to compute certain values**.

One could think that both should return exactly the same result, since the operations that are carried out are exactly the same. However, this is not true.

(aus *Introduction to VHDL programming, Juan Antonio Clemente.*
Translation to the English of the material written by: Marcos Sánchez-Élez
(Introducción a la programación en VHDL) November 13, 2014)

9 .2.6 VHDL Grundgerüst

VHDL - "Very High Speed Integrated Circuit Hardware Description Language"
(die Deutung änderte sich mehrfach)

- HDL (hardware description language HDL) ist kompakter als Schaltpläne
- ist leichter kopierbar (StrgC + StrgV) (als Schaltpläne)
- lässt sich automatisch überprüfen
- lässt sich simulieren
- lässt sich ”synthetisieren” (in eine FPGA-Innen-Schaltung umwandeln)
- ist vorteilhaft bei Prozessordesign
- VHDL steht in Konkurrenz zu "Verilog"
- wir verwenden das Produkt ’Quartus II’ (Nachfolger ’Quartus Prime’) und das ’DE0’ Development Board

```
-- (this is a VHDL comment)
/*
    this is a block comment (VHDL-2008)
*/
-- VHDL is case-insensitive!

library IEEE;
use IEEE.std_logic_1164.all;

entity SchaltungsName is
  port (
        ...
  );
end entity SchaltungsName;

architecture Verarbeitung of SchaltungsName is
begin
        ...
end architecture Verarbeitung;
```

das Grundgerüst: Du hast
a) einen Header (so was wie diese ’#include’ in ’C’) → das schreiben wir einfach jedes Mal so ab

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
```

b) eine Modul-Beschreibung (’Entity’) mit Ein- und Ausgängen:

```
ENTITY <Modulname> IS PORT(
  ...
);
END <Modulname>;
```

und
c) die Schaltungsbeschreibung (’Architecture’):

```
ARCHITECTURE <Schaltungsname> OF <Modulname> IS
BEGIN
  ...
END <Schaltungsname>;
```

→
das Ganze schaut dann so aus:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY <Modulname> IS PORT(
  ...
);
END <Modulname>;

ARCHITECTURE <Schaltungsname> OF Modulname IS
BEGIN
  ...
END <Schaltungsname>;
```

9 .2.7 simples Beispiel

```
library IEEE;
use IEEE.std_logic_1164.all;

entity SchaltungsName is
  port (
    Eingang : in   std_logic;
    Ausgang : out  std_logic;
    BusOut  : out  std_logic_vector(7 downto 0); --'vector'==Leitungs-Buendel
    BusIn   : out  std_logic_vector(7 downto 0);
  );
end entity SchaltungsName;

architecture Verarbeitung of SchaltunngsName is
begin
  Ausgang <= not Eingang;
  BusOut  <= not BusIn when(Eingang) --invertieren
           else BusIn;               --kopieren
end architecture Verarbeitung;
```

9 .2.8 Fachsprache

wie immer ist händische Mitschrift zu führen

- **Synthese** ::= zusammensetzen, *"erzeugen"*
- *kann **synthetisiert** werden* ::= lässt sich in eine Schaltung umsetzen
- **Analyse** ::= zerlegen, *"schauen, was drin ist"*
- **Simulation** ::= gleichwertige Ergebnisse liefernde Software;
  Simulatoren *ignorieren das Zeitverhalten*, sind meist viel langsamer, aber vorteilhaft für Prognose und Untersuchungen
- "nur für Simulation" ::= lässt sich **nicht!** als Schaltung realisieren
- **Emulator** ::= nachahmende Ersatzhardware,
  Emulatoren zeigen auch das *originale Zeitverhalten*, sind daher **ersatzweise** nutzbar
- VHDL wird **nicht!** von einem Computer ausgeführt!
  → nicht alle Programmiersprachen erzeugen ausführbaren Computer-Code.
  - HTML erzeugt → Webseiten
  - LaTeX erzeugt → PDF-Dokumente (damit schreibi inser U-PDF)
  - SPICE erzeugt → Simulationsdiagramme
  - VHDL erzeugt **Hardware**
    daraus werden dann ICs gefertigt oder FPGAs programmiert (ge'flash't)
    'flashen' ist nicht englisch, es stammt von der Speicher-Technologie 'Flash-EEPROM'

die Elektronikabteilung hat eine handvoll FPGA-EntwicklungsModule, wo man mit dem Utility 'Quartus' VHDL-Programme hineinflashen und testen kann

→ es gibt zwar (was iXH Enk aus Verwirrungsgefahr no nit zeigen will) auch Variable und Schleifen in VHDL, die führen aber nicht dazu, dass etwas mehrfach ausgeführt, sondern dieselbe Hardware mehrfach erzeugt wird.

· "nur für Simulation" ::= lässt sich *nicht* als Schaltung realisieren

· "signal" ::= erzeugt schaltungsinterne Verbindungsleitungen

Aufgabe: Schreib in Deine Handmitschrift in VHDL

a) ein UND-Gatter
b) ein NOR-Gatter
c) einen Inverter
d) eine Prozessor-ALU, die '+', '-', +1, -1, AND und OR kann

und maile mirXH dann das Foto der Mitschrift

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY dingsda IS PORT(
  A1,A2,
  A3    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  B     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
  sel0,
  sel1  : IN  STD_LOGIC;
);
END dingsda;

ARCHITECTURE Funktion OF dingsda IS
  SIGNAL vonAnachB: STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  vonAnachB <= A2 when (sel0 = '1') else A1;
  B         <= A3 when (sel1 = '1') else vonAnachB;
END Funktion;
```

dieses "A2 when(...) else A1;"
erzeugt einen **"Selector"** = **"Multiplexer"**,
der - abhängig von der Bedingung im "when(...)" - entweder das Bündel A2 oder A1 auswählt ('selektiert')

* Aufgabe: Schreib einen 2:1 Multiplexer mit 8-Bit-Inputs A und B, 8-Bit-Output Y und einem Auswahl-Eingang "s", der bei s=0 das A und bei s=1 das B an Y ausgibt
* Abgabetermin: Stundenende

9 .2.9 VHDL signal

wie immer ist händische Mitschrift zu führen

"signal" ::= erzeugt schaltungsinterne Verbindungsleitungen

```
ARCHITECTURE Funktion OF dingsda IS
  SIGNAL   vonAnachB : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
  ...
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY dingsda IS PORT(
  A1,A2 : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  A3    : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  B     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
  sel0  : IN  STD_LOGIC;
  sel1  : IN  STD_LOGIC;
);
END dingsda;
```

Initialisierung ≡ Anfangswert:

```
ARCHITECTURE Funktion OF dingsda IS
  SIGNAL   vonAnachB : STD_LOGIC_VECTOR(7 DOWNTO 0) := (others => '0');
```

```
                                                    --Initialisierung alle '0'
BEGIN
  vonAnachB <= A2 when (sel0 = '1') else A1;
  B         <= A3 when (sel1 = '1') else vonAnachB;
END Funktion;
```

Wenn 'signal' als Bitspeicher (Flipflop) verwendet wird, ist die Initialisierung oft erforderlich

9 .2.10 Selector-Multiplexer-Decoder

wie immer ist händische Mitschrift zu führen

→ gib die Funktion der Schaltung als Wahrheitstabelle an
(da stehen dann beim 'Y' a, b und c statt 0 und 1 drin)
vgl. mit Baustein '74HC138'

```
begin
  -- r, s, t   ... Auwahl-Inputs (std_logic)
  -- a, b, c   ... Daten -Inputs (std_logic)
  rst <= (r s t);
  with rst select
    Y <=
        a when ''001'',
        b when ''101'',
        c when ''110'';
end;
```

9 .2.11 if case when with

wie immer ist händische Mitschrift zu führen

Schreib in Deine Handmitschrift:
- Ist ein Prozessor eine FSM (Schrittschaltwerk)? (Argumente auflisten)
- Ist ein Prozessor ein *synchrones* oder ein *asynchrones* 'Zeug' ?
- Wieviele Zustände hätte ein Prozessor ?
- Wieviele Zustände hätte ein Computer ?
- schlag (zB. im Internet) die VHDL-Operatoren "srl", "sra" und "ror" nach - was tun sie?
- schreib ein Schaltnetz, welches aus den
  8-Bit-Inputs A und B
  ↓
  die Outputs
  'LT' .... bei A < B
  'GT' .... bei A > B
  'EQ' .... bei A==B
  auf '1' schaltet, in VHDL

und maile mirXH dann das Foto der Mitschrift mit Lösungen

Fallunterscheidungen

with/select:

```
with <auswahl-vector> select
ergebnis <= <Verkn&uuml;pfung_1> when <auswahlwert_1> ,
            <Verkn&uuml;pfung_2> when <auswahlwert_2> ,
            ...
            <Verkn&uuml;pfung_n> when others;
```

when/else:

```
<ergebnis> <=  <Verkn"upfung_1> when <Bedingung_1>
           else <Verkn"upfung_2> when <Bedingung_2>
                ...
           else <Verkn"upfung_n>;
```

if/then: only in sequential environment! (lernen wir erst)

```
if    <Bedingung_1> then <Sequentielle Anweisungen 1>;
elsif <Bedingung_2> then <Sequentielle Anweisungen 2>;
elsif <Bedingung_3> then <Sequentielle Anweisungen 3>;
        ...
else  <Sequentielle Anweisungen n>;
endif;
```

case/is: only in sequential environment!

```
case <testsignal> is
     when <Wert_1> => <Sequentielle Anweisungen 1>;
     when <Wert_2> => <Sequentielle Anweisungen 2>;
           ...
     when others => <Sequentielle Anweisungen n>;
end case;
```

9 .2.12 'Elevator' Steuerg. (6)

```
architecture ...
begin
  schliessen <= '1'
        when  timer1   ='1'
          AND timer2   ='0'
          AND zustand  ="stop"
          AND LBarrier ='1'
          AND openButt ='0'
        else '0';
  LangsamfahrtAuf <= '1'
        when  TuerZu   ='1'
          AND timer2   ='1'
          AND Richtung ="steigen"
        else '0';
end;
```

9 .2.13 in, out

wie immer ist händische Mitschrift zu führen

Ein - und Ausgänge:
Es gibt allerlei 'Datentypen', wir verwenden vorerst aber nur 'STD_LOGIC'.
einzelne Ein- und Ausgangsleitungen erzeugt man mit zB.:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
  x   : IN   STD_LOGIC;
  y   : OUT  STD_LOGIC;
);
END reg8;
```

ganze Leitungs-Bündel erzeugt man mit 'vector':
(hier ein 8-faches Bündel mit den Leitungen Nr.0 bis Nr.7)
→ es ist *ratsam*, die Reihenfolge abwärts anzugeben, sonst verdreht es Dir später bei Registern und Addierern die Bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
  d   : IN  STD_LOGIC_VECTOR(7 DOWNTO 0);
  q   : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
);
END reg8;
```

Mit 'in', 'out' (und 'inout') gibt man die Daten-Richtung an:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
  x   :  IN   STD_LOGIC;
  y   :  OUT  STD_LOGIC;
);
END reg8;
```

9 .2.14 Ampel, RSFF, 1-Bit-Latch, DFF

wie immer ist händische Mitschrift zu führen

was ist das:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY dingsda IS PORT(
  set,reset : IN    STD_LOGIC;
  Q,notQ    : INOUT STD_LOGIC;
);
END dingsda;

ARCHITECTURE Funktion OF dingsda IS
BEGIN
  Q    <= reset NOR notQ;
  notQ <= set   NOR Q;
END Funktion;

        -- was ist neu in diesem VHDL-Programm?
        -- wir haben dieses 'Ding' auch mit NAND statt NOR kennengelernt -
              beschreibe es in VHDL
        -- beizeiten sendest Du mir bitte wieder Mitschriftfotografien
```

was ist neu in diesem VHDL-Programm?

wir haben dieses 'Ding' auch mit NAND statt NOR kennengelernt - beschreibe es in VHDL!

ganze Prozessoren oder Speicher so zu beschreiben, gäbe schauderhaft umständlichen VHDL-Code
dafür hat man etwas Besseres/Einfacheres erfunden:
den VHDL-**'Process'**

Ein Latch (1):

- leider gibt es keinen Befehl "bau da jetzt ein DFF hinein" (oder ein Latch)
- man kann das VHDL aber unsichtbar dazu **zwingen**, Latches einzubauen, indem es anstatt eines neuen Wertes den alten nehmen muss
- diesen "alten" hat es aber nur, wenn er **gespeichert** wurde.
- was sind Latches? und Register? Was ist der Unterschied?

```
Q <= D when (Enable) else Q;
```

- **'others'** ::= *alle nicht genannten*
- **'all'** ::= *alle Möglichkeiten*

Ein Latch (2)

```
process(all)
begin
    Q <= D when (Enable);
end process;
```

Ein DFF ('D'-Flipflop) (3)

```
DFF : process(all) is
begin
  if (RST) then
    Q <= '0';
  elsif rising_edge(CLK) then
    Q <= D;
  end if;
end process DFF;
```

auch ein DFF (4)

```
DFF : Q <= '0' when (RST)  else D when rising_edge(clk);
```

9 .2.15 8-Bit-Latch, Register

wie immer ist händische Mitschrift zu führen

```
-- 8-Bit-Latch:
ENTITY Modul IS
  PORT
  (
  Din : IN STD_LOGIC_VECTOR(7 downto 0);
  ena : IN STD_LOGIC;
  Q   : OUT STD_LOGIC_VECTOR(7 downto 0);
  );
END Modul;
ARCHITECTURE DLatch8 OF Modul IS
BEGIN
  latch : PROCESS (clk, Din)
  BEGIN

    IF (ena = '1') THEN -- wenn clk == '0'
      Q <= Din;         -- kriegt das Q keinen Wert,
    END IF;             -- also nimmts den alten

  END PROCESS latch;

END DLatch8;

--If clk equals '1', the value Din is assigned to Q.
--If clk equals '0', the circuit maintains its previous state, creating a latch.
```

* die DFF-Übung wurde wunderbar gemacht!
* ganze Prozessoren oder Speicher so zu beschreiben, gäbe schauderhaft umständlichen VHDL-Code
* dafür hat man etwas Besseres/Einfacheres erfunden: den *VHDL-'Process'*

```
-- 8-Bit-Register, steigende clk-Flanke:
ENTITY Modul IS
  PORT
  (
  Din : IN STD_LOGIC_VECTOR(7 downto 0);
  clk : IN STD_LOGIC;
  Q   : OUT STD_LOGIC_VECTOR(7 downto 0);
  );
END Modul;
ARCHITECTURE Dreg8 OF Modul IS
BEGIN
  register : PROCESS (clk)
  BEGIN

    IF rising_edge(clk) THEN    -- ausser bei steigender clk-Flanke
      Q <= Din;                 -- kriegt das Q keinen Wert,
```

```
    END IF;                     -- also nimmts den alten

  END PROCESS register;
END Dreg8;

        --leider gibt es keinen Befehl "bau da jetzt ein DFF hinein" (oder ein Latch)
        --man kann das VHDL aber unsichtbar dazu zwingen, Latches einzubauen,
          indem es anstatt eines neuen Wertes den alten nehmen muss
        --diesen "alten" hat es aber nur, wenn er gespeichert wurde.
        --was sind Latches? und Register? Was ist der Unterschied?

        Aufgabe: Schreib ein (flankengetriggertes) DFF als VHDL-Process
        Abgabe Deadline: Samstag Abend

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
  Din : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
  Q   : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
  clr : IN  STD_LOGIC;
  clk : IN  STD_LOGIC;
);
END reg8;

ARCHITECTURE working OF reg8 IS
BEGIN
  process(clk, clr)
  begin
   if clr = '1' then
        Q <= x''00000000'';
   elsif rising_edge(clk) then
        Q <= Din;
   end if;
 end process;
END working;
```

oder mit negativer Flanke:

```
-- 8-Bit-Register, fallende clk-Flanke:
ENTITY Modul IS
  PORT
  (
  Din : IN STD_LOGIC_VECTOR(7 downto 0);
  clk : IN STD_LOGIC;
  Q   : OUT STD_LOGIC_VECTOR(7 downto 0);
  );
END Module;
ARCHITECTURE Dreg8 OF Modul IS
BEGIN
  register : PROCESS (clk, Din)
  BEGIN
    IF falling_edge(clk) THEN
      Q <= Din;
    END IF;
  END PROCESS register;
END Dreg8;
```

9 .2.16 VHDL-'process'

wie immer ist händische Mitschrift zu führen

primitiv-Ampel:
schreib eine ganz primitive rot-gelb-grün-Ampel ohne irgendwelche Fussgänger oder Blinkphasen **in VHDL**.
Du wirst merken, dass Du SPEICHER-Elemente (DFF) für die Zustände brauchst.

- tu so, als könne man *Bits in Verbindungsleitungen speichern*
  (was natürlich in Wirklichkeit nicht geht)
- innere Verbindungsleitungen legt man mit
  signal (wie bei 'port')
- Syntax: **signal** Name **: typ**;

```
zB.:
 architecture Schaltung1 of Modul is
  signal vonAnachB: std_logic;
  signal vonCnachD: std_logic_vector( 7 downto 0);
 begin
  ...
 end;
```

- Web-Recherche:
  mache Dich zur Eignung der FPGA-Programme "Quartus IIünd "Quartus Prime"für unsere Zwecke schlau (wir sollten Digitalschaltungen erfassen und simulieren sowie das Ältera DE0Ëntwicklungskit programmieren können) zB. auf https://en.wikipedia.org/wiki/Intel_Quartus_Prime,
  https://www.google.com/search?q=Quartus+II
  und https://www.google.com/search?q=Quartus%20Prime.
  Fertige dabei ein "Recherche-Protokoll" (wo, was, wie nützlich...) an (Handmitschrift)
  (ganz professionell wäre gleich eine Nutzwertanalyse)
- beizeiten sendest Du mir bitte wieder Mitschriftfotografien

Der *'sequential'* **process**

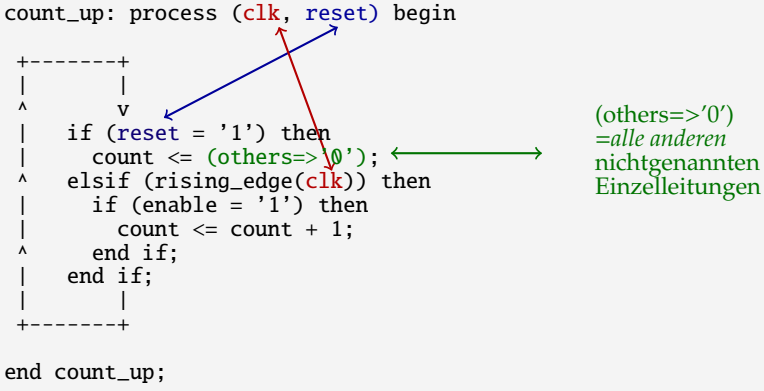
- der 'process' wurde **für Programmierer** erfunden
- es wird **Hardware** erzeugt, die *Ähnliches* leistet.
  (*) das Zeitverhalten ist nicht das einer von-Neumann-Programmsoftware

```
<proc-name> ':' 'process'  <sensitivity-list>  'begin'

    -- hier koennen lokale Signale
       oder Variablen deklariert werden

begin

    -- hier ist ein 'sequential environment'
    ...

end <proc-name>;
```

<proc-name>	ist frei wählbar (bis auf die 'reserved words' wie *'entity'*, *'port'*, *'process'* ...)
<sensitivity-list>	führt alles an, worauf ein 'process' *sensibel* reagiert, dh. einen *'process'-Durchlauf* auslöst.

ein VHDL-Compiler erzeugt die Hardware so,
als würde der 'process' immer wieder durchlaufen
(Funktionen hintereinander in Serie geschaltet):

zum Beispiel:



jetzt kannst Du in VHDL schon
→ Register
→ Latches
→ Counter
→ Multiplexer und Decoder
→ kominatorische Schaltungen wie Prozessor-ALU

bauen!
Es wird Zeit, dass Du einen ganzen Prozessor baust!

dazu brauchsch den → *'von-Neumann-Zyklus'* **p.??**

9 .2.17 nomal von vorn

Aufgabe: Schreib (in Dei Mitschrift) VHDL mit

- 8-Bit-Inputs A, B
- 8-Bit-Output Y
- einem 1-Bit-Input 'AddSub'
- wenn AddSub='0' gebe Y die Summe A+B aus,
  wenn AddSub='1' die Differenz A-B
- mache die Übung einmal mit 'with/select' und einmal mit 'when/else'
- primitiv-Ampel:
  schreib eine ganz primitive rot-gelb-grün-Ampel ohne irgendwelche Fussgänger- oder Blinkphasen **in VHDL**.
- Du wirst merken, dass Du SPEICHER-Elemente (DFF) für die Zustände brauchst.
- tu so, als könne man *Bits in Verbindungsleitungen speichern*
  (was natürlich in Wirklichkeit nicht geht)
- *innere Verbindungsleitungen* legt man mit
  signal (wie bei 'port')

Syntax: **signal** <Name> **: typ;**
zB.:

```
architecture Schaltung1 of Modul is
        ...
  signal   vonAnachB :  std_logic;
  signal   vonCnachD :  std_logic_vector( 7 downto 0);
begin
  ...
end;
```

9 .2.18 primitiv-ALU OPs + Phasen

wie immer ist händische Mitschrift zu führen

```
  with OPCODE select
    ERGEBNIS <=\\
        ERGEBNIS when ''1111 1111'' --NOP
        A+A      when ''0000 0100'', --ADD A,A u. SLA A
        A+B      when ''0000 0101'', --ADD A,B
        A-A      when ''0000 1100'', --SUB A,A
        A-B      when ''0000 1101'', --SUB A,B
        A+1      when ''0000 0000'', --INC A
        B+1      when ''0000 0001'', --INC B
        A-1      when ''0000 1000'', --DEC A
        B-1      when ''0000 1001'', --DEC B
        (-)A     when ''0001 1000'', --INV A
        (-)B     when ''0001 1001'', --INV B
        A AND B  when ''0011 0101'', --AND A,B
        A NAND B when ''0011 1101'',
        A OR B   when ''0011 0001'', --OR A,B
        A NOR B  when ''0011 1001'',
        A XOR B  when ''0011 0011'', --XOR A,B
        A XNOR B when ''0011 1011'',
        NOT A    when ''0010 1010'', --NOT A
        NOT B    when ''0010 1011'', --NOT B
        '1111 1111' when others;

        ALU - Arithmetics + Logics Unit ('Rechenwerk'):

          Ueberlege, welche 'operations' Dein Prozessor haben soll, wie zB.

                 operation:             'mnemonic'      binaercode:
                 'nichts'               NOP             1111 1111
                 'A+1'                  INC A           0000 0000
                 'B+1'                  INC B           0000 0001
                 'A-1'                  DEC A           0000 1000
                 'B-1'                  DEC B           0000 1001
                 'A+B'                  ADD A,B         0000 0100
                 'A-B'                  SUB A,B         0000 1100
                 (-)A ( == not(A-1))    INV A           0001 1000
                 (-)B ( == not(B-1))    INV B           0001 1001
                 'not A'                CPL A           0010 1000
                 'not B'                CPL B           0010 1001
                 'A and B'              AND A,B         0011 0000
                 'A or B'               OR  A,B         0011 1000
                usw.

                -- Gib den 'operations' binaere Nummern, die dann in VHDL mit
                   'if'/'case'/'when-else'/'with-select' unterschieden werden koennen.
                -- so eine Tabelle nennt man das 'instruction-set'
                -- die Zuordnung des Binaercode hat ein "System"
                -- 'Mnemonics' sind textuelle Instruktions-Namen

        CU - Control Unit ('Leitwerk')
                steuert die ALU und den gesamten Ablauf s. ==> 'von-Neumann-Zyklus'.
                Daraus ergibt sich der Bedarf an zusaetzlichen Registern
                zur Zwischenspeicherung binaerer Codenummern und Datenwerte:
          in der CU:
                -- von-Neumann-Phasen-Zaehler
                -- program counter ( = instruction ointer)
                -- instruction register
                -- Fallunterscheidung: von-Neumann-Phase-1, s. ==> 'von-Neumann-Zyklus'
```

```
                -- Fallunterscheidung: von-Neumann-Phase-2, s. ==> 'von-Neumann-Zyklus'
                -- Fallunterscheidung: von-Neumann-Phase-3, s. ==> 'von-Neumann-Zyklus'
                -- Fallunterscheidung: von-Neumann-Phase-4, s. ==> 'von-Neumann-Zyklus'
                -- Fallunterscheidung: von-Neumann-Phase-5, s. ==> 'von-Neumann-Zyklus'
          in der ALU:
                data pointer
                Rechen-Register zB.:
                        ALU-Register A
                        ALU-Register B
                        (oder eine ganze 'register-bank' R0, R1, R2 ...)
                ALU-Ergebnis-Register s. auch ==> 'von-Neumann-Zyklus'

        Abgabe: Foto der Handmitschrift mit Deinem bisherigen Prozessor-Entwurf in VHDL
```

9 .2.19 ALU, CU, FETCH

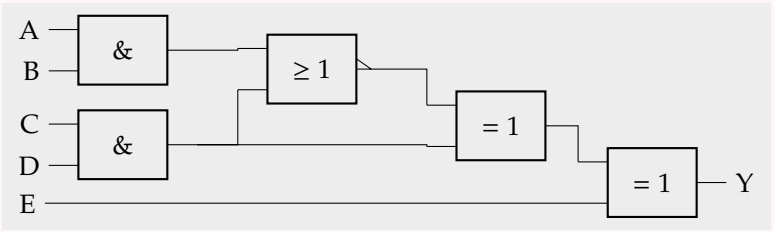
Fr-11Dez20:
wie immer ist haendische Mitschrift zu fuehren

```
ENTITY Prozessor IS
  PORT (
  Databus  : inout  STD_LOGIC_VECTOR(7 downto 0);
  Adressbus: out    STD_LOGIC_VECTOR(7 downto 0);
  clk      : in     STD_LOGIC;
  notWrite : out    STD_LOGIC;
  notRead  : out    STD_LOGIC;
  );
END Prozessor;
ARCHITECTURE Innenleben of Prozessor IS BEGIN
  ...
  ALU:process(clk) begin
  ...
  end ALU;
  CU: process(clk) begin
  ...
  end CU;
END Innenleben;
END Prozessor;


        Ergaenze zum bisherigen Prozessor-VHDL:
        Umsetzung der von-Neumann-Phase-1 nach VHDL s. 'von-Neumann-Zyklus':

        Instruction FETCH:
          --erhoehen des 16-Bit-Programmzaehlers (program counter, instruction pointer)
          --Programmzaehlerinhalt auf den 16-Bit-Adressbus legen,
                solange clock=="Hi"
          --READ-Signalleitung aktivieren (active low)
          --Wartezeit fuer RAM-Speicherbausteine
          --neuen Befehlscode (instruction code) vom 8-Bit-Datenbus
                ins 8-Bit-InstructionRegister einlesen
          --READ-Signalleitung de-aktivieren
          --Adressbus wieder freigeben (=Verbindung mit program counter trennen),
                sobald clock="Low"

        Abgabe: Handmitschrift-Foto Deines heute erweiterten Prozessor-Entwurfs in VHDL
        Abgabe-Termin: glei!
```

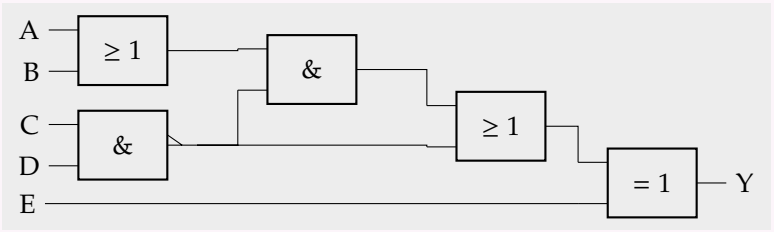
bitte fang damit schon einmal an!
Vielleicht beginnst Du bei der ALU, des hast schon einmal gemacht:
Überlege, welche 'operations' Dein Prozessor haben soll, wie zB. 'A+B', 'A-B', 'A+1', 'A-1', (-)A, 'not A', 'A and B', 'A or B' usw.
Gib den Operations binäre Nummern, die dann in VHDL mit 'if', 'when', 'select' udgl. unterschieden werden können.

9 .2.20 VHDL-Übungsaufgaben

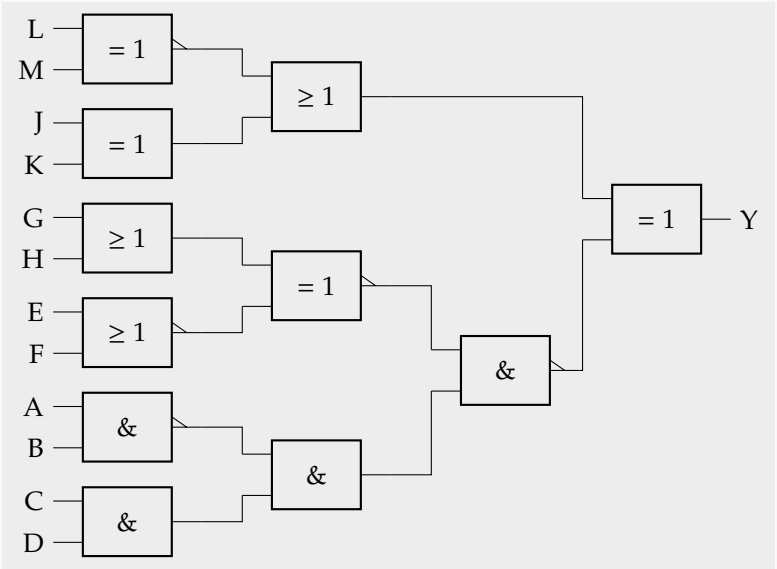
↪ schreibe folgende Gatterschaltungen auf VHDL um:
↪ vereinfache folgende Gatterschaltungen:



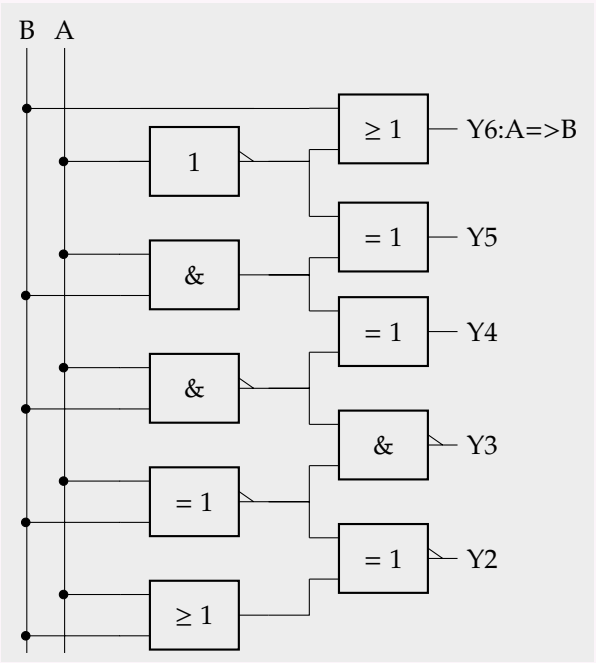
Bsp.Lösung:
Y <= (((A and B) nor (C and D))
xor (C nand D)) xor E



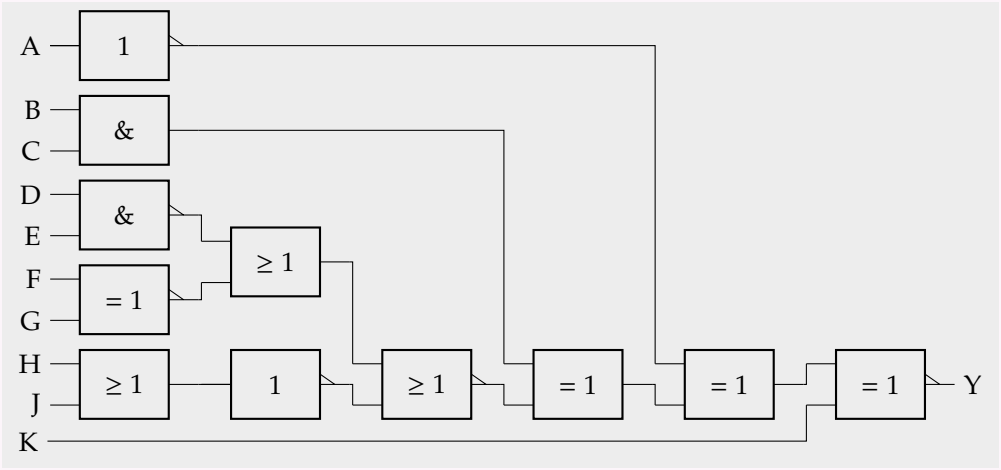
Bsp.Lösung:
Y <= (((A or B) and (C nand D))
or (C nand D)) xor E



Bsp.Lösung:
Y <= (((L xnor M) or (J xor K))
xor
(((G or H) xnor (E nor F))
nand
((A nand B) and (C and D))));



Bsp.Lösung:
Y6 <= B or not A;
Y5 <= not A xor (A and B);
Y4 <= (A and B) xor not(A and B);
Y3 <= not(not(A and B) and not(A xor B));
Y2 <= not(not(A xor B) xor (A or B));



Y = not xor((not A xor ((B and C) xor (((D nand E) or (F xnor G)) nor (not(H or J))))) xor K);

9 .2.21 Truth Table (5)

→ gib die Funktion der Schaltung als **Wahrheitstabelle** an

- 'realisiere folgende Wahrheitstabelle in VHDL'

a b c	Y
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	1
1 1 0	1
1 1 1	0

```
begin
  rst <= (r s t);
  with rst select
    Y <=
```

```
        '1' when ''001'',
        '1' when ''101'',
        '1' when ''110'',
        '0' when others;
end;
```

oder (sequential):

```
architecture ... begin
  process (a,b,c) begin
    abc <= (a b c);
    case abc is
      when ''001'' => Y <= '1';
      when ''101'' => Y <= '1';
      when ''110'' => Y <= '1';
      when others => Y <= '0';
    end case;
  end process;
end;
```

9 .2.22 Kombinatorik

↪ realisiere in VHDL:
Wenn timer1 und nicht timer2 und Zustand==stop aber Lichtschranken nicht blockiert und TorAuf-Taste nicht gedrückt, dann aktiviere Türschließmotoren

↪ realisiere in VHDL:
Wenn TuerZu und timer2 und Richtung='steigen', dann aktiviere LangsamfahrtAuf

```
Bsp.concurrent:

architecture ...
begin
schliessen <= '1'
   when timer1 = '1'
   AND timer2='0'
   AND zustand="stop"
   AND LBarrier='1'
   AND openButton='0'
   else '0';
LangsamfahrtAuf <= '1'
   when TuerZu = '1'
   AND timer2 = '1'
   AND Richtung="steigen"
   else '0';
end;
```

```
sequential:

architecture ...
begin
process(timer1,timer2,zustand,LBarrier,openButton)
begin
IF timer1 = '1'
   AND timer2='0'
   AND zustand="stop"
   AND LBarrier='1'
   AND openButton='0'
THEN schliessen <= '1'
ELSE schliessen <= '0'
END IF;
end process;
process(timer2,TuerZu,Richtung) begin
IF TuerZu = '1'
   AND timer2 = '1'
   AND Richtung="steigen"
THEN LangsamfahrtAuf <= '1'
ELSE LangsamfahrtAuf <= '0'
END IF;
end process;
end;
```

9 .2.23 Latch, Register

↪ beschreibe ein 8-Bit-Latch in VHDL
↪ beschreibe ein 8-Bit-Register in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
   d : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
   q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   clr : IN STD_LOGIC;
   clk : IN STD_LOGIC;
);
END reg8;

ARCHITECTURE working OF reg8 IS
BEGIN
   process(clk, clr)
   begin
     if clr = '1' then
       q <= x"00000000";
     elsif rising_edge(clk) then
       q <= d;
     end if;
   end process;
END working;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY reg8 IS PORT(
   d : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
   o : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   LE : IN STD_LOGIC;
   nOE : IN STD_LOGIC;
);
END reg8;

ARCHITECTURE working OF reg8 IS
   signal q : STD_LOGIC_VECTOR(7 DOWNTO 0);
BEGIN
   process(LE,nOE)
   begin
     if LE = '1' then
       q <= d;
     end if;
     if nOE = '0' then
       o <= q;
     else
       o <= "ZZZZZZZZ";
     end if;
   end process;
END working;
```

9 .2.24 counter

↪ realisiere up-counter und down-counter in VHDL

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ctr8 IS PORT(
   e : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
   q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   clr : IN STD_LOGIC;
   clk : IN STD_LOGIC;
);
END ctr8;

ARCHITECTURE working OF ctr8 IS
BEGIN
   process(clk, clr, e)
   begin
     if clr = '1' then
       q <= x"00000000";
     elsif falling_edge(clk) then
       q <= e + "1";
     end if;
   end process;
END working;
```

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY ctr8 IS PORT(
   e : IN   STD_LOGIC_VECTOR(7 DOWNTO 0);
   q : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
   clr : IN STD_LOGIC;
   clk : IN STD_LOGIC;
);
END ctr8;

ARCHITECTURE working OF ctr8 IS
BEGIN
   process(clk, clr, e)
   begin
     if clr = '1' then
       q <= x"00000000";
     elsif rising_edge(clk) then
       q <= e - "1";
     end if;
   end process;
END working;
```

9 .2.25 Bus+Register

18Dez20

Übung: Schreib folgende Prozessor-Bestandteile in VHDL:

↪ 16-Bit-Datenbus
↪ 24-Bit-Adressbus
↪ *'clk'*-Eingang
↪ *'notRead'*-Ausgang
↪ *'notWrite'*-Ausgang

↪ 16-Bit-ALU-Register 'A'
↪ 16-Bit-ALU-Register 'B'
↪ 24-Bit-ALU-Register 'DataPointer'

Musterlsg.:

```
ENTITY CPU16 IS PORT(
   DBUS : INOUT   STD_LOGIC_VECTOR(15 DOWNTO 0);
   ABUS : OUT STD_LOGIC_VECTOR(23 DOWNTO 0);
   clk  : IN STD_LOGIC;
   notRead : OUT STD_LOGIC;
   notWrite : OUT STD_LOGIC;
); END CPU16;
ARCHITECTURE working OF CPU16 IS
   SIGNAL A,
   B : STD_LOGIC_VECTOR (15 downto 0);
   SIGNAL DPTR : STD_LOGIC_VECTOR (23 downto 0);
```

9 .2.26 RAM-Zugriffe

22Dez20

es muss eine 'Instruction' geben (die uns noch fehlt), die ein Byte aus einer RAM-Speicher-Stelle holt ('RAM read') und ins CPU-Register 'A' **liest**/kopiert.

Die RAM-Speicher-Adresse dafür stehe im ALU-'DataPointer' Register

a) Bitte beschreibe die Prozessor-'Instruction' LOAD A,RAM gem. → ***'von-Neumann-Zyklus'*** **p.??**

b) bitte auch eine fürs ALU-Register 'B'

c) bitte auch eine 'Instruction', die aus ALU-Register 'A' ins RAM zurück **speichert**, wird man brauchen: 'STORE A,RAM'

d) und bitte ebenso für das ALU-Register 'B'

Musterlsg.:

```
RAMREAD: process(clk) BEGIN
  if (PhaseCTR=3)
  AND InstReg(7 downto 2)="0100 000" THEN
    if rising_edge(clk) THEN
      ABUS <= DPTR;     --DataPointer auf Abus
      notRead <= '0';   --READ signal
    END if;
      -- Wartezeit für Speicherbausteine:
      -- (== warten auf fallende clk-Flanke)
    if falling_edge(clk) then
       --Datenregister vom Datenbus lesen
      case InstReg is
        when "0100 0000" => A <= DBUS;
        when "0100 0001" => B <= DBUS;
      END case;
      notRead <= '1';    -- release READ
      ABUS <= others => 'Z'; -- Adressbus freigeben
    END if;
  END if;
END RAMREAD;
RAMWRITE: process(clk) BEGIN
  if (PhaseCTR=3)
  AND InstReg(7 downto 2)="0110 00" THEN
    if rising_edge(clk) THEN
      ABUS <= DPTR;     --DataPointer auf Abus
      case InstReg is
        when "0110 0000" => DBUS <= A;
        when "0110 0001" => DBUS <= B;
        when "0110 0010" => DBUS <= ErgReg;
      END case;
      notWrite <= '0';   --WRITE signal
    END if;
      -- Wartezeit für Speicherbausteine:
      -- (== warten auf fallende clk-Flanke)
    if falling_edge(clk) then
      notWrite <= '1';    -- release WRITE
      ABUS <= others => 'Z'; -- Adressbus freigeben
      DBUS <= others => 'Z'; -- Datenbus freigeben
    END if;
  END if;
END RAMWRITE;
```

9 .2.27 Phasen 2-5

wie immer ist händische Mitschrift zu führen

(mach vielleicht an Teil am Mo und einen am Di)

leider werdiXH wegen Maturaklassen-Unterrichts in der Anstalt am Dialog-Unterricht mit Dir gehindert, sodass Du das alleine im Arbeitsauftrag machen musst.

Es sind **ganz großartige Lösungen** der Aufgabe(Arbeitsauftrag) vom Fr-11Dez'20 eingesandt worden!

Ihr seids jo schun richtige CPU-Designer !!!

Teifl, Mander, Eys seids guat!
Nach 2 Monat schu Prozessor entwerfen ischa Hammer!

(leider von nur 6 ganz tollen Teilnehmern)

möchtegern-'Ingenieure', die nur wissen, wann sie *nicht* arbeiten wollen,
aber nicht einmal die Hälfte beherrschen,
braucht niemand :-((

Ergänze zum bisherigen Prozessor-VHDL (Bissel modifiziert und ergaenzt):

→ Umsetzung der von-Neumann-Phase-2 ins VHDL (s. ==> 'von-Neumann-Zyklus'):

2 Instruction DECODE
diese Wartezeit für die kombinatorische Decodier-Logik im 'Leitwerk' (control unit CU) besteht in der Wirklichkeit aus unumgänglichen Gatterlaufzeiten. Wenn diese zu knapp bemessen werden (zB. zu hohe 'clock' - Frequenz), dann wird schon weitergerechnet, bevor die Steuersignale/Ergebnisse stimmen, und die ganze 'Mühle' 'stürzt ab' (das erlebt man immer wieder beim 'Overclocking')

Eine der Abgaben sagt *'das RAM ist eh so schnell, dass man es nicht berücksichtigen muss'*
- da ist u.U. was dran, nur 'clocken' die Hersteller ihre PCs bis genau an diese Grenze

→ Umsetzung der von-Neumann-Phase-3 ins VHDL (s. ==> 'von-Neumann-Zyklus'):

3 Data LOAD
↓ DataPointer-Inhalt auf den Adressbus legen
↓ READ-Signalleitung aktivieren
↓ Wartezeit für Speicherbausteine
↓ Binärwert vom Datenbus in ein Datenregister einlesen
↓ READ-Signalleitung de-aktivieren
↓ Adressbus freigeben ('Z' state)

→ Umsetzung der von-Neumann-Phase-4 ins VHDL (s. ==> 'von-Neumann-Zyklus'):

4 Instruction EXECUTE
diese Wartezeit für die kombinatorische Rechen-Logik im 'Rechenwerk' (arithmetic and logic unit ALU) besteht wieder aus Gatterlaufzeiten in den Rechenschaltungen der ALU

→ Umsetzung der von-Neumann-Phase-5 ins VHDL (s. ==> 'von-Neumann-Zyklus'):

5 Data SAVE
↓ DataPointer-Inhalt auf den Adressbus legen
↓ Ergebnisregister-Inhalt auf den Datenbus legen
↓ WRITE-Signalleitung aktivieren
↓ Wartezeit für Speicherbausteine
↓ WRITE-Signalleitung de-aktivieren
↓ Adress- und Datenbus wieder freigeben ('Z' state)

→ Abgabe: Handmitschrift-Foto Deines heute erweiterten Prozessor-Entwurfs in VHDL
→ Abgabe-Termin: Di-15Dez'20 nach der Dic-Stunde

```
ENTITY Prozessor IS
  PORT (
  DBus     : inout  STD_LOGIC_VECTOR(7 downto 0);
  ABus     : out    STD_LOGIC_VECTOR(15 downto 0);
  clk      : in     STD_LOGIC;
  notWrite : out    STD_LOGIC;
  notRead  : out    STD_LOGIC;
  );
END Prozessor;
ARCHITECTURE Innenleben of Prozessor IS BEGIN
  /* 'ProgramCounter', auch 'InstructionPointer'
   *  == RAM-Adresse des naechsten CPU-Befehls ('instruction'):
   */
   PC  :signal STD_LOGIC_VECTOR (15 downto 0);


  /* von-Neumann-Phasen-Zaehler 'PhaseCTR' 1 bis 5
   * also 3 Bit:
   */
```

```
 PhaseCTR :signal STD_LOGIC_VECTOR (2 downto 0);  --sind 3 Bit!


/* 'Data Pointer', auch 'Memory Pointer'
 * == RAM-Adresse fuer Daten-Byte:
 */
 DPointer:signal STD_LOGIC_VECTOR (15 downto 0);


/* 'instruction register'
 * speichert den 'instruction code' waehrend der Datenbus
 * ALU-Daten zu uebertragen hat:
 */
 InstReg :signal STD_LOGIC_VECTOR ( 7 downto 0);


/* CPU-Rechen-Register A und B fuer arithmetische und
 * logische Operationen wie 'A+B' oder 'A and B':
 */
 CpuRegA,
 CpuRegB :signal STD_LOGIC_VECTOR ( 7 downto 0);--Rechen-Register


/* CPU-Rechen-Ergebnis-Register zur Zwischenspeicherung der Ergebnisse
 * bis zur von-Neumann-Phase-Nr5-'Data SAVE'
 */
 ErgReg  :signal STD_LOGIC_VECTOR ( 7 downto 0);



ALU:process(clk) begin

  --aus den Abgaben von '1_8', '1_6', '1_5', '2_7', '2_8':

  if (PhaseCTR=4) then
      case InstReg is
          when '00000000' => ErgReg <= A  +  1;
          when '00000001' => ErgReg <= B  +  1;
          when '00001000' => ErgReg <= A  -  1;
          when '00001001' => ErgReg <= B  -  1;
          when '00000100' => ErgReg <= A  +  B;
          when '00001100' => ErgReg <= A  -  B;
          when '00010000' => ErgReg <=    -  A;--'CPL'
          when '00010001' => ErgReg <=    -  B;--'CPL'
          when '00101000' => ErgReg <=   not A;
          when '00101001' => ErgReg <=   not B;
          when '00110000' => ErgReg <= A and B;
          when '00111000' => ErgReg <= A or  B;
          --when '11111111' => ErgReg <= ErgReg;--brauchma nid
          when  others    => ErgReg <= ErgReg;
      end case;
  end if;
end ALU;



CU: process(clk) begin
  PhaseCTR <= PhaseCTR+1;
  if rising_edge(clk) then
    if (PhaseCTR=1) then
      PC <= PC+1;
      ABus <=  PC;
      notRead <= '0';
          --'insert irrelevant Delay here'
      InstReg <= DBus;
      notRead <= '1';
      ABus <= (others => 'z');
    elseif (PhaseCTR=2) then
      --(tue gar nichts)
    elseif (PhaseCTR=3) then
      --'data LOAD'
      ABus <= DPointer;
      notRead <= '0';
        /* Huch!
         * da fehlen uns ja noch Befehle!
         * -> bitte selber erfinden
         */

      case InstReg is
        when ... => A <= DBus;
        when ... => B <= DBus;
```

```
        end case;
        notRead <= '1';
        ABus <= (others => 'z');
      elseif (PhaseCTR=4) then
        --'instruction EXECUTE'

        ...

      elseif (PhaseCTR=5) then
        --'data SAVE'
        ABus <= DPointer;
          /* Huch!
           * da fehlen uns ja  auch noch Befehle!
           * -> bitte selber erfinden
           */

        case InstReg is
          when ... => DBus <= A;
          when ... => DBus <= B;
        end case;
        notWrite <= '0';
          --'insert irrelevant Delay here'
        notWrite <= '1';
        DBus <= (others => 'z');
        ABus <= (others => 'z');
      end if;
    end if;  --rising_edge(clk)
  end CU;
END Innenleben;
```

9 .2.28 RESET input

↪ Implementiere einen low-aktiven '-RESET' Eingang
und alle seine Prozessor-Funktionen
in VHDL

↪ '-RESET' setzt den ProgramCounter (PC, InstructionPointer), das InstructionRegister (InstREG), den von-Neumann-Phasen-Zähler (PahseCTR) und die CPU-Flags (CY, N, V, I, ...) auf Null (=> others '0'), verändert aber keine Register- oder RAM-Inhalte

9 .2.29 von-Neumann Architektur

wie immer ist händische Mitschrift zu führen

* Schreib eine 8-Bit von-Neumann-Architektur in VHDL!
  (nicht den vonNeumann-Zyklus sondern die Architektur ie. die Bestandteile)
* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Fr-08Jan'21 nach der Dic-Stunde

9 .2.30 FETCH + SAVE

wie immer ist händische Mitschrift zu führen

* Schreib die vonNeumann-Phase-Nr1: 'Instruction Fetch' in VHDL,
  d.h. den Code im 'CU: process (clk)',
  und nur den Teil für die 'Instruction-Fetch'-Phase
  (die Architektur ie. die Bestandteile hast ja am 08.Jan'21 schon beschrieben)
* die "Wartezeit für Speicherbausteine" realisieren wir, indem wir auf die fallende Takt-Flanke 'falling_edge(clk)' warten:

```
if falling_edge(clk) then
  InstReg &lt;= DBus;
  notREAD &lt;= '1';
  ...
```

* Schreib auch die vonNeumann-Phase-Nr5: 'Data Save' in VHDL,
  d.h. nur den Teil fü die 'Data Save'-Phase im 'CU: process (clk)'
* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Di-12Jan'21 nach der Dic-Stunde

9 .2.31 Register File

wie immer ist händische Mitschrift zu führen

* Moderne Prozessoren haben statt Register A und B ein sogenanntes 'Register File' (die 'AVR-8-Bit'-Controller zB. 'R0'..'R31'), welche sie mittels eines **internernen Datenbus** mit den Recheneinheiten der ALU verbinden.
  → Schreib eine 8-Bit vonNeumann-Architektur mit zusätzlich 2 internen Register-Datenbussen RDbus1, RDbus2 und 2 zusätzlichen Register-Adressbussen RAbus1, RAbus2 zur Realisierung einer ALU mit 32er-Registerbank in VHDL!
* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Fr-15Jan'21 nach der Dic-Stunde

9 .2.32 falling_edge

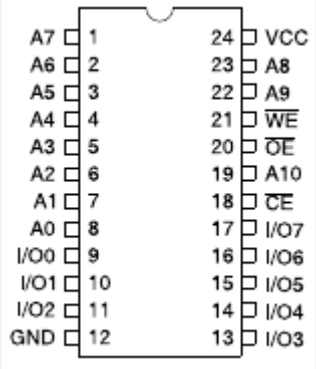
wie immer ist händische Mitschrift zu führen

* stell bitte alle 5 von-Neumann-Phasen in der CU so um, dass die 'Wartezeiten' durch fallende Flanken-triggerung ersetzt werden
* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Di-19Jan'21 nach der Dic-Stunde

9 .2.33 SRAM

wie immer ist händische Mitschrift zu führen

* In der vorletzten Übung (Register-Bank) ist Dir wohl die Idee zum VHDL-Statement für die Verwendung der Register-Datenbusse RDbus1, RDBus2 abgegangen. Klar, es fehlen ja auch die 32 Register.
* Das 'Pin-Out' eines statischen RAM-Bausteins 'SRAM 6116':



-WE ... 'Write Ennable' == notWrite
-OE ... 'Output Ennable' == notRead
-CE ... 'Chip Ennable' == notCS chip select
I/O0 - I/O7 ... Data Bus D0-D7 (8 Datenleitungen)
A0-A10 ... 11 Adress-Leitungen (Adressen 0..2047)

* wie lasse ich das VHDL einen Datenspeicher erzeugen?

fü unsere Prozessor-interne Registerbank schreibt man in VHDL:

```
  signal RgBank:  array(0 to 31) of STD_LOGIC_VECTOR(7 downto 0);

          wie in:

ARCHITECTURE Innenleben  of Processor IS BEGIN
    ...
  signal RgBank:  array(0 to 31) of STD_LOGIC_VECTOR(7 downto 0);
    ...
begin
  ALU: Process(clk) BEGIN
    ...
    elseif (PhaseCTR=3) then
      if rising_edge(clk) then
        RDBus1 <= RgBank(to_integer(unsigned( RABus1 )));
        ...
      end if;
    ...
    elseif (PhaseCTR=5) then
      if rising_edge(clk) then
        RgBank(to_integer(unsigned( RABus2 ))) <= RDBus2;
      end if;
    ...
  end process;
END Innenleben;


        * Einen reinen SRAM-Speicher-Baustein (ohne Prozessor) wie diesen '6116'
          wuerden wir so schreiben:

ENTITY RAM2k IS
  PORT (
  DBus    : inout  STD_LOGIC_VECTOR(7 downto 0);
  ABus    : out    STD_LOGIC_VECTOR(11 downto 0);
  nWR,
  nRD,
  nCS     : in     STD_LOGIC;
  );
END RAM2k;

ARCHITECTURE RamInnen of RAM2k IS BEGIN

  signal sRAM : ARRAY (0 to 2047) of STD_LOGIC_VECTOR(7 downto 0);

  Process(nRD, nWR) BEGIN
    if not( nCS ) AND NOT( nRD ) then
      DBUS  <=  sRAM( to_integer(unsigned( ABus )));
    elseif not( nCS ) AND NOT( nWR ) then
        sRRAM( to_integer(unsigned( ABus ))) <= DBus;
    end if;
  END;

END RamInnen;
```

* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Fr-22Jan'21 nach der Dic-Stunde

9 .2.34 JUMP

wie immer ist händische Mitschrift zu führen

* eine Klärung: (Dank an 'Boti'!) Die Trennung eines Prozessors in ALU und CU ist → von-Neumanns Idee. Müssen tut man nicht. Bis zur Erfindung des Mikroprozessor 1971 war diese Trennung zudem wegen der meterlangen Leitungen notwendig. Seither weicht diese Aufteilung auf den Prozessor-Chips zusehends einem gewissen Sauhaufen

  Wir halten die Trennung bitte weiterhin aufrecht.

  In der Programmcode-Struktur haben wir die 5 von-Neumann-Phasen in der CU, während Arithmetik

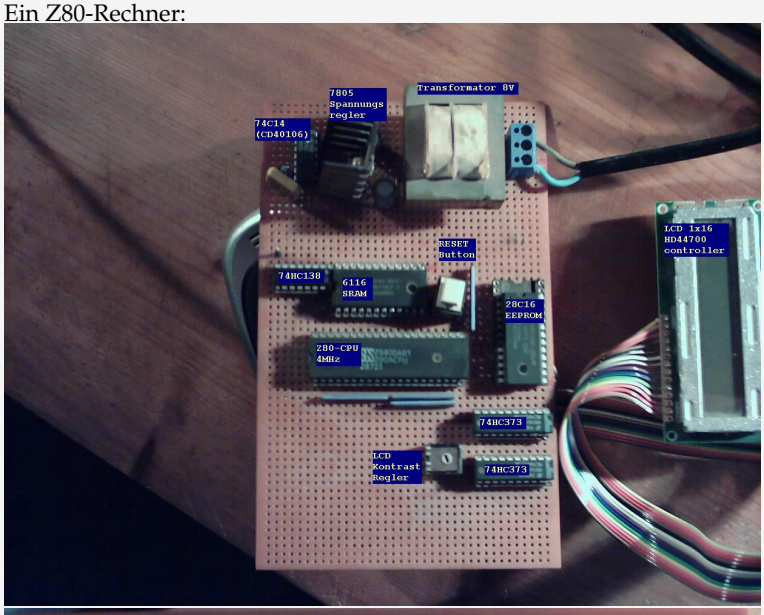
und Kombinatorik in der ALU landen.

(Das Prinzip, *alles was mit Daten zu tun hat in die ALU, Instructions in die CU*, is eher mutwillig)

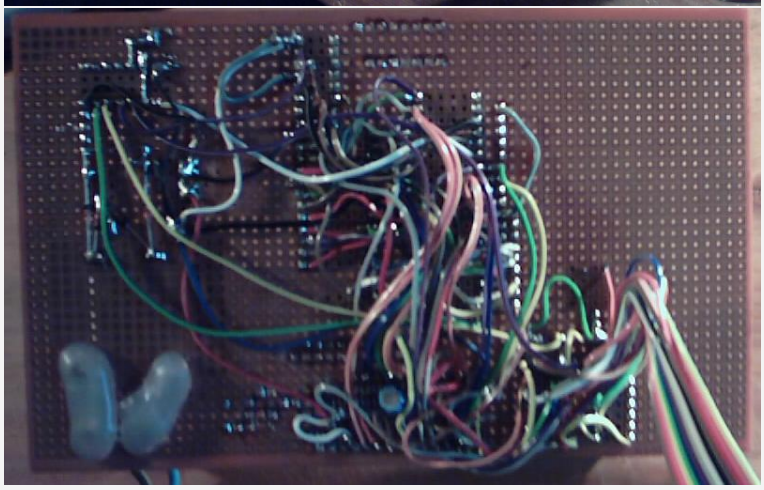
* Implementiere zusätzlich einen **GOTO**-Befehl (Prozessorfachleute sagen '**JUMP**-Instruction') in Deinem Prozessor in VHDL, bestehend aus
  - o 1 Byte Instruction Code (als Instruction FETCH)
  - o 1 Byte Sprung-Ziel-Adresse (als DATA LOAD)

  (für 16-Bit-Adressen isch des zwar 1 Byte zu wenig, des mochat ins die CU vorerst aber zu komplex)
* Bei diesem **'JUMP'** ist in Phase #3 der Datenbus-Inhalt in den Instruction-Counter/Pointer ('InstCTR') zu laden
* wir studieren im `Datenblatt des Microcontrollers AVR AT Tiny-13A`
  → das 'Pinout of ATtiny13A' im 8-pin-PDIP-Gehäuse (case)
  und im
  `avr-instruction-set-manual.pdf`
  → die **RJMP** Instruction im Vergleich zur **JMP** Instruction
* Abgabe: Handmitschrift-Foto(s) Deines VHDL-Entwurfs
  + Kurzfassung der heutigen Aufgabe
  per eMail an 'c.schoenherr' at 'tsn.at'
* Abgabe-Termin: Di-26Jan'21 nach der Dic-Stunde

Ein Z80-Rechner:



und die Unterseite:



9 .2.35 BRANCH

wie immer ist händische Mitschrift zu führen

- zum Programmieren sind 'if'-Fallunterscheidungen notwendig
- studiere im `Datenblatt des Microcontrollers AVR AT Tiny-13A`
  und im `avr-instruction-set-manual.pdf`

  ↪ welche Instructions zur Fallunterscheidung existieren
  ↪ die CPU-'Flags' (status register SREG)
  ↪ 'conditional BRANCH' instructions
- notiere jedes gefundene Faktum plus der Stelle im Handbuch (manual)

- Abgabe: Handmitschrift-Foto(s) Deines Manual-PDF-Research per eMail an 'c.schoenherr' at 'tsn.at'
- Abgabe-Termin: Fr-29Jan'21 nach der Dic-Stunde

9 .2.36 ASM = Assembler

wie immer ist händische Mitschrift zu führen

μC Programmierung (μC = MicroController):

Wir programmieren diese AVR-ATtiny13 (die wir noch gar nit haben) in der Programmiersprache **"Assembler"**, das ist die **Maschinensprache**, in der die Instruction-Codes durch **"Mnemonics"** genannte, sprechende Wortfetzen wie "ADD", "MOV", "JUMP" ersetzt sind.

→
Wir müssen die Assembler-Mnemonics im `avr-instruction-set-manual.pdf`
`http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`
`https://ww1.microchip.com/downloads/en/DeviceDoc/AVR-Instruction-Set-Manual-DS40002198A.pdf`
`https://en.wikipedia.org/wiki/Atmel_AVR_instruction_set`
lernen/verwenden.
Oft ist auch die Hardware-Ausstattung des μC lt.
local: `zuig/Datenblatt des Microcontrollers AVR AT Tiny-13A`
full: `https://ww1.microchip.com/downloads/en/DeviceDoc/doc8126.pdf`
full: `http://www.getchip.net/wp-content/uploads/ATTiny13.pdf`
short: `https://ww1.microchip.com/downloads/en/devicedoc/8126s.pdf`
anzusprechen.

Bitte noch keine Developement-Tools installieren!
(es würde due ersten Lernschritte nur stören).

(wir werden dann den `'gavrasm' - Gerd's AVR Assembler`
und später auch das
`Atmel Studio (vorm.AVR-Studio)` (mikrocontroller.net)
`www.microchip.com/avr-support/atmel-studio-7` (microchip.com)
verwenden)

Programm-1: Eine Zähl-Schleife:

```
        LDI     R30,10
LOOP:
        DEC     R30
        BRNE    LOOP
```

Fertige je ein

- Flussdiagramm
- Struktogramm
- Pseudocode-Programm

dieses Code-Stückes an.

Programm-2: "LebenszeichenËndlosschleife mit PortB.1 Ausgabe

```
MAIN:           ;das HAUPTPROGRAMM: Nur "Lebenszeichen" ausgeben:
        IN      R17,PORTB       ;PINB...InputRegister von Port-B
        CBI     PORTB,1         ;Bit Nr.1 auf "Low"
        SBRS    R17,1           ;Uebersprinqe, wenn das Bit==1 war
        SBI     PORTB,1         ;Bit Nr.1 auf "High"
        RJMP    MAIN            ;RJMP == "Relative JuMP" == "goto"
```

Fertige je ein

- Flussdiagramm
- Struktogramm
- Pseudocode-Programm

dieses Code-Stückes an.

Programm-3: das vollständige ”Lebenszeichen-Assembler-Programm”

```
;das (kleine) "LEERE" Atmel AVR AT Tiny13 Programm
;  (fuer Linux-Assembler "gavrasm")
;---------------------------------------------------------------------
.device ATtiny13        ;Gerd's AVR-Assembler taugt fyr viele Controller
.def    Rcnt    =R16    ;Schleifenzaehler Rcnt...alias/nickName fyr "Counter"
.def    Rtemp   =R17    ;"temporary" fuer kurzes Zwischenspeichern
 ;
        .CSEG           ;CodeSpeicher (ProgrammFlash) Segment
        .ORG    0x0000  ;Speicher-Adress-Festlegung "Origin"
        RJMP    INIT            ;wird beim Start und RESET aktiviert
         RETI   ;EXT_INT0       ;ISR ist "leer" (nur "RETI")
         RETI   ;PINCHG
         RETI   ;T0OVF
         RETI   ;EE_RDY
         RETI   ;ANACOMP
         RETI   ;TMR0_CMPA
         RETI   ;TMR0_CMPB
         RETI   ;WDR
         RETI   ;ADC
INIT:
        CLI                             ;disable all Interrupts
        LDI     Rtemp,Low(RAMEND)       ;"stack" einrichten:
        OUT     SPL,Rtemp
        LDI     Rtemp,0b00000011        ;PB0,PB1 ... output
        OUT     DDRB,Rtemp
        OUT     PORTB,Rtemp             ;Outputs auf "Hi" schalten
        SEI
;
MAIN:           ;das HAUPTPROGRAMM: Nur "Lebenszeichen" ausgeben:
        IN      Rtemp,PORTB     ;PINB ... InputRegister von Port-B
        CBI     PORTB,1         ;Bit Nr.1 auf "Low"
        SBRS    Rtemp,1         ;'SKIP if BIT in REGISTER is SET'
                                ;Uebersprinqe, wenn Bit==1 war
        SBI     PORTB,1         ;SET BIT in I/O
                                ;Bit Nr.1 auf "High"
        RJMP    MAIN            ;RJMP ... "Relative JuMP" == "goto"
.EXIT                           ;Ende der Assemblierung
```

Dies dient jetzt nur einmal zum Anschauen.
Aufgaben:

- was tut Programm-1 ?
- was tut Programm-2 ?
- was tut Programm-3 ?

Abgabe: Handmitschrift-Foto(s) Deines Manual-PDF-Research
per eMail an 'c.schoenherr' at 'tsn.at'

Abgabe-Termin: Di-02.Feb'21 nach der Dic-Stunde

9 .2.37 PulsGen

Fr 05Feb21 PulseGenerator

wie immer ist händische Mitschrift zu führen

Entwirf in einem selbsterdachten Maschinen-Pseudocode:
Ein Programm welches ewig kurze Pulse ausgibt,
indem es die Pausen zwischen den Pulsen sowie die Zeit während der Pulse mit nichtstuenden Warteschleifen verbringt ("busy waiting")

Schätze diese Zeiten bei 1MHz CPU-Takt (clock) anhand der Angaben
unter "Cycles"
im AVR8-instruction-set-manual.pdf

schreib es dann in "AVRischer" Assemblersprache.

9 .2.38 SP, CALL u. RETURN

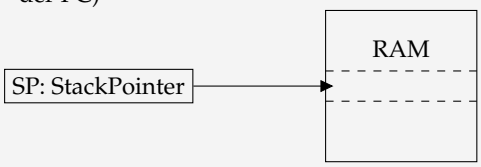
SP, CALL, RETURN Mo 15.Feb'21, Di 16.Feb'21

Übung: Schreibs in VHDL

wie immer ist händische Mitschrift zu führen

Der **Stack-Pointer SP**

Der **Stack Pointer "SP"** ist ein CPU-Register in der CU und speichert eine RAM-Adresse (ähnlich wie der PC)



Der SP-adressierte RAM-Bereich heißt "Stack", weil er als "Stapel" dient – man legt immer oben drauf und nimmt immer von oben wieder weg. Der AVR-uC-Stack ist kopfüber - er "wächst von oben nach unten".
bei gewohnter, "nach oben" wachsender RAM-Nutzung überschreiben sich Stack und Variablen auf diese Weise erst, wenn das gesamte RAM voll ist

Unterprogramm-Aufruf mit **CALL**

An der SP-Adresse speichert die **CALL**-Instruction die Angabe, wo es nach einer Unterbrechung '**wieder weiter** geht'. Der SP-adressierte RAM-Speicher ist quasi ein PC-'Backup'. Bei "CALL" (und bei "Interrupt") wird automatisch so ein 'Backup' im RAM angelegt,

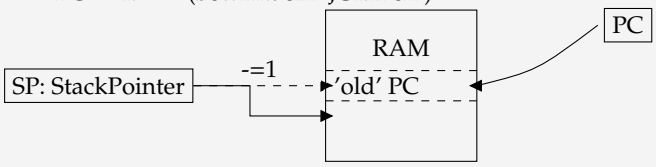
Ablauf von 'CALL k':

CALL (8 Bit OpCode)

k (2 x 8 Bit = 16 Bit Sprungziel-Adresse)

RAM[SP] ← PC+3 (Adresse der naechsten Instruction)

SP ← SP-2 (2 Byte der Ruecksprung-Adresse)
PC ← k (bewirkt ein 'JUMPen')



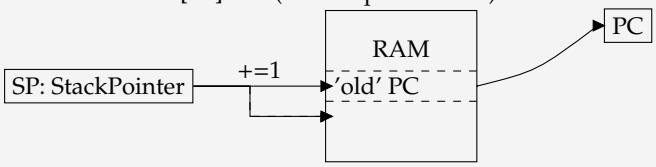
Die Rückkehr vom Unterprogramm mit **RETURN** bzw. **RETI**

Bei "RET" (bzw. "RETI") wird das 'PC-Backup' wieder aus dem RAM gelesen und in den PC kopiert. Das entspricht einem 'JUMP' zurück dorthin, wo vorher mit 'CALL' (bzw. 'Interrupt') unterbrochen wurde.

Ablauf von 'RET':

RET (8 Bit OpCode)

SP ← SP+2
PC ← RAM[SP] (StackTop in den PC)



Abgabe: Handmitschrift-Foto(s) per eMail an 'c.schoenherr' at 'tsn.at'

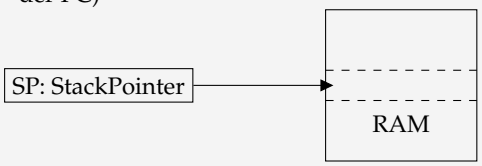
Abgabe-Termin: Di-16.Feb'21 nach der Dic-Stunde

9 .2.39 PUSH u.POP

SP, PUSH + POP
Übung: Schreibs in VHDL

Der **Stack-Pointer SP**

Der **Stack Pointer "SP"** ist ein CPU-Register in der CU und speichert eine RAM-Adresse (ähnlich wie der PC)

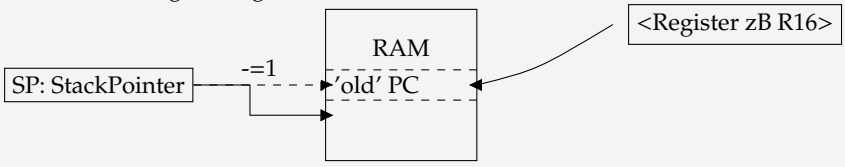


Der SP-adressierte RAM-Bereich heißt "Stack", weil er als "Stapel" dient – man legt immer oben drauf und nimmt immer von oben wieder weg. Der AVR-uC-Stack ist kopfüber - er "wächst von oben nach unten" s. **'CALL'**.
bei gewohnter, "nach oben" wachsender RAM-Nutzung überschreiben sich Stack und Variablen auf diese Weise erst, wenn das gesamte RAM voll ist

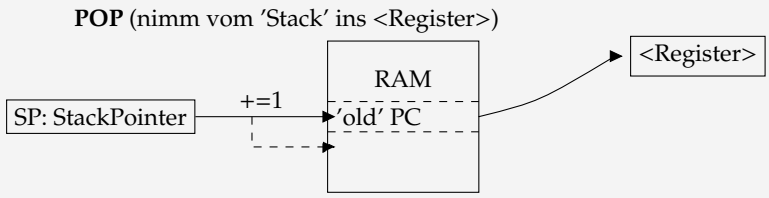
Um auch Anderes als die Unterprogramm-Rücksprungadresse 'stapeln' zu können, gibt es die Instructions

```
PUSH <Register>
```

PUSH (gib <Register>-Inhalt auf den 'Stack')




```
POP <Register>
```



Abgabe: Handmitschrift-Foto(s) per eMail an 'c.schoenherr' at 'tsn.at'

Abgabe-Termin: glei nach der Dic-Stunde

9 .2.40 Interrupt+RETI

Übung: Schreibs in VHDL

'Interrupt' (im Englischen ein Zeitwort (verb)!!) ist eine elektrisch ausgelöste 'CALL'-Instruction an eine hardwaremäßig fix 'verdrahtete' Unterprogramm-Adresse (zB. bei 'Timer0-Ablauf' nach Programmspeicher-Adresse '0003' (16-Bit-Instruction-Word), wo als Unterprogramm eine sog, 'Interrupt-Service-Routine' (ISR) gespeichert wird.

So ein 'Interrupt' kann an jeder Stelle des Hauptprogramms ausgelöst werden und dieses (unvorhersehbar) **unterbrechen** (daher der Name 'interrupt')

Dabei setzt die CPU das 'I'-Flag im Statusregister 'SREG', welches weitere Interrupts sperrt.

Statt 'RETURN' wird eine ISR mit 'RETI' (Return from Interrupt) beendet - das 'RETI' löscht das 'I'-Flag im SREG wieder, sodass nachfolgende Interrupts wieder aktiviert werden.

Sollte schon während der ISR ein weiteres Interrupt-Signal eingetroffen sein, so wird dieser 'Interrupt' sofort nach dem 'RETI' ausgelöst (nachdem zuvor jedoch genau 1 Hauptprogramm-Instruction ausgeführt wurde und diese nicht ausgerechnet ein 'CLI' ist)

Aufgabe-1: Formuliere eine Interrupt-Aktivierung in VHDL:

```
if(external_interrupt-signal) then
  --in der FETCH -Phase:
  PC <= ''0000 0000 0000 0001'';    --PGM-Adress 0x0001

  --in LOAD und SAVE-Phasen tut sich in diesem Fall gar nix
end if
```

Aufgabe-2: Formuliere ein RETI in VHDL:

```
if( InstReg = RETIcode ) then
  --Stack-Top in den PC
    ... (Du!)

  --SP adjustieren
    ... (Du!)

  --'I'-Flag loeschen
    ... (Du!)
end if
```

9 .2.41 ISR = InterruptServiceRoutine

ISR - InterruptServiceRoutine

```
Beispiel fuer eine primitive 'Anacomp-Analogkomparator'-ISR:

ANACOMP:PUSH    R16             ;R16 retten
        IN      R16,SREG        ;SREG nach R16 retten
        ADDIW   R30,1           ;R30 | R31 incrementieren
        OUT     SREG,R16        ;SREG restaurieren
```

```
        POP     R16             ;R16 restaurieren
        RETI                    ;ISR Ende
```

9 .2.42 Interrupt Vectors Vc2

AVR Interrupt Vectors

```
die 'AT tiny13A:
Pgm     Int.Vector
Address
------- -----------------
0000    RJMP    INIT            ;wird beim Start und RESET aktiviert
0001    RETI    ;EXT_INT0       ;ISR ist "leer" (nur "RETI")
0002    RETI    ;PINCHG
0003    RETI    ;T0OVF
0004    RETI    ;EE_RDY
0005    RJMP    ANACOMP
0006    RETI    ;TMR0_CMPA
0007    RETI    ;TMR0_CMPB
0008    RETI    ;WDR
0009    RETI    ;ADC
```

9 .3 VHDL Crashkurs vom Schlemmer

Crashkurs VHDL

FH München, FB 06

1 Einleitung

Die Hardwarebeschreibungssprache VHDL hat sich (Stand 2002) zum weltweiten Standard für den Entwurf digitaler Schaltungen entwickelt. Kenntnisse in VHDL entsprechen in ihrer Bedeutung in der Hardwareentwicklung in etwa Kenntnissen in der Programmiersprache „C“ in der Softwareentwicklung.

In diesem Kurs werden Ihnen die grundlegenden Konzepte eines Schaltungsentwurfs in VHDL erläutert. Dabei geht es ausschließlich um die Schaltungssynthese, der gesamte Bereich der Simulation wird ausgespart. Mit den vermittelten Kenntnissen können Sie zwar weder elegante Beschreibungen für kleine Schaltungen noch parametrisierbare komplexe Entwürfe formulieren– sie können aber kleinere digitale Schaltungen nach „Schema F“ in VHDL beschreiben.

Kursziele:

- Verständnis für grundlegende Konzepte in VHDL
- Sprachreferenz (Syntax) für wichtige Befehle
- Schemata für grundlegende Aufgabenstellungen (Kombinatoriken, Automaten)

Theoretische Voraussetzungen:

- Boolesche Algebra
- Automatenentwurf (Moore-Automat)

Bitte beachten Sie, dass die Programmiersprache, die Ihnen das Nachdenken abnimmt, noch nicht erfunden worden ist. Sie werden im Verlauf des Kurses feststellen, dass Ihnen VHDL teilweise sehr einfache und eingängige Beschreibungsmöglichkeiten bietet – sofern Sie wissen, was beschrieben werden soll! Verwenden Sie also bei einem neuen Entwurf zunächst einen guten Teil Ihrer Zeit darauf, Ihr Problem günstig zu strukturieren und die einzelnen Teilaufgaben sauber zu definieren. Scheuen Sie sich nicht, zunächst ganz ohne VHDL Papier zu benutzen um darauf Blockdiagramme für die Grobstrukturierung oder Zustandsdiagramme für Automaten zu zeichnen. Erst wenn Sie sicher sind, das Problem in sinnvolle Teile zerlegt zu haben und für jeden Teil wissen, was er genau leisten soll, denken Sie an die Umsetzung in VHDL. Bei diesem Vorgehen werden Sie feststellen, dass die reine Programmierung in VHDL zum Kinderspiel wird, da die einzelnen Teile einfach aufgebaut sind (und damit mit Methoden aus „Schema F“ abgedeckt werden können) und leicht unabhängig voneinander ausgetestet werden können.

Wenn Sie stattdessen ohne klares Ziel vor Augen sofort mit einem VHDL-Programm beginnen, so werden Sie nach aller Erfahrung sehr bald den Überblick verlieren und ständig neue Variablen und Verzweigungen einführen, um „merkwürdige“ (weil nicht bedachte) Effekte in der Schaltung zu eliminieren.

Das Resultat, sofern es eines gibt, ähnelt dann einer Bauruine, die nur mit Stützen und Streben notdürftig aufrechterhalten wird und vermutlich beim ersten Unwetter (sprich Praxiseinsatz) einstürzt.

2 Signale, Typen und Vektoren

Für die folgenden Erläuterungen soll als Beispiel die Schaltung in Abbildung 1 verwendet werden.

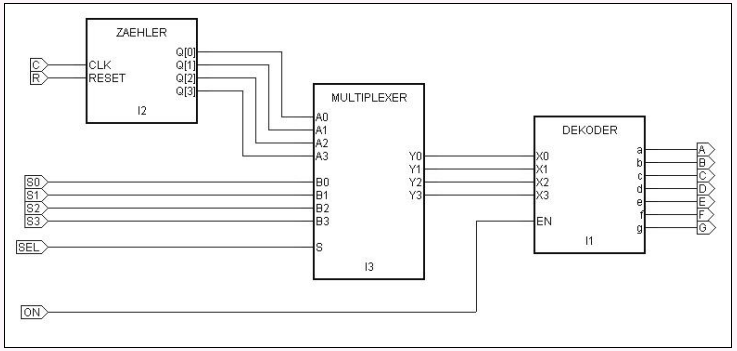


Abbildung 1: Blockschaltbild

Die Schaltung besteht aus drei **Blöcken** (Zaehler, Multiplexer, Dekoder), die untereinander durch **Signale** verbunden sind. Diese Darstellung entspricht dem Blockdiagramm auf Papier zur Aufteilung des Gesamtproblems in kleinere Teilprobleme. Die Gesamtschaltung hat die Aufgabe, entweder einen Zählerstand (0 bis 9) oder eine direkte Zahleneingabe (0 bis 9) an den Schaltern S0 bis S3 in Abhängigkeit von einem Wahlschalter SEL auf einer Siebensegmentanzeige auszugeben. Die Siebensegmentanzeige kann über einen Schalter ON ein- bzw. ausgeschaltet werden. Der Zähler kann mit zwei Tastern C und R entweder hochgezählt oder auf Null zurückgesetzt werden.
Diese Darstellung eignet sich für eine weiteren Entwurf in VHDL ideal, da in VHDL ebenfalls **Blöcke** beschrieben werden, die durch **Signale** untereinander verbunden sind.

Sofern in den Beispielen VDHL-Code verwendet wird, werden Schlüsselworte fett geschrieben oder farbig markiert.

2.1 Konventionen in VHDL

Vorab einige wichtige Regeln in VHDL, die Erläuterung der Beispiele folgt später.

2.1.1 Namensregeln

Für alle folgenden Ausführungen und natürlich den eignen Entwurf müssen Sie bei der Wahl von Bezeichnern (Namen von Signale etc.) beachten:

- Zwischen Groß- und Kleinschreibung wird in VHDL nicht unterschieden
- Namen müssen mit einem Buchstaben beginnen
- Schlüsselworte sind nicht als Namen erlaubt.

Da Sie nicht alle Schlüsselworte kennen können, sollten Sie entweder einen Editor verwenden, der VHDL-Schlüsselworte erkennt und von sich aus bereits hervorhebt oder in weiser Voraussicht verdächtige Namen wie „and“, „else“ oder „begin“ von sich aus vermeiden.

2.1.2 Kommentare

Kommentare werden durch einen doppelten Bindestrich eingeleitet und gelten bis zum Zeilenende (Abbildung 2).

```
-- Dieser Block enthält einen einfachen 2_zu_1 Multiplexer

entity multiplexer is
port
(
  S:        in bit;     -- Steuereingang zur Signalauswahl
  A, B:     in bit;     -- Dateneingänge
  Y:        out bit     -- Datenausgang
);
end;
```

Abbildung 2: Kommentare

2.1.3 Zuweisungen

Zuweisungen eines Signals an ein anders Signal bzw. eines konstanten Wertes an ein Signal werden in VHDL von rechts nach links vorgenommen und durch die Zeichenkombination <= dargestellt (Abbildung 2).

```
Y <= S;           -- S wird nach Y kopiert
Y <= A or B;      -- A und B werden mit “oder” verknüpft und das
                  -- Ergebnis wird Y zugewiesen
A => Y;           -- so rum geht es NICHT!
```

Abbildung 3: Zuweisungen

2.2 Typologie

VHDL ist eine streng typgebundene Sprache, d.h., jedem verwendeten Signal (bzw. Variablen) muss ein Typ zugewiesen werden. Eine implizite Typkonversion wie in „C“ oder einen Universaltyp wie „void“ gibt es nicht. Für den Entwurf genügt die Kenntnis der folgenden drei Typen: **boolean**, **bit** und **std_logic**.

Typ	**Wertevorrat**	**Verwendung**
boolean	true, false	logische Abfragen (if)
bit	0, 1	Entwurf
std_logic	0, 1, Z, -, U, L, H, X, W	Entwurf und Simulation

Tabelle 1: Standardtypen

Eine Deklaration eines Signals mit einem Typ sieht in VDHL wie in Abbildung 4 gezeigt aus.

Syntax:
signal <signalnamen>: typ;

Beispiele:

```
signal X0, X1, X2, X3: bit;   -- vier Signale X0 bis X3 vom Typ bit
signal EN: std_logic;         -- signal EN hat den Typ std_logic
signal ein_aus: boolean;      -- ein boolesches Signal ein_aus
```

Abbildung 4: Signaldeklarationen

Bei Verknüpfungen müssen die einzelnen Signale vom gleichen Typ sein. Das Ergebnis kann natürlich auch nur einem Signal vom gleichen Typ zugewiesen werden. Einzelne konstante Werte der Typen bit und std_logic müssen in Hochkommas eingeschlossen werden (Abbildung 5).

```
X0 <= ´0´;              -- X0 ist konstant 0
EN <= ´Z´;              -- EN wird Z zugewiesen (hochohmig)
ein_aus <= true;        -- ein_aus ist wahr
X1 <= ´Z´;              -- geht nicht, weil der Typ bit den Wert
                        -- Z nicht kennt
ein_aus <= ´1´;         -- geht auch nicht, der Typ boolean den
                        -- Wert 1 nicht kennt (1 ist nicht true)

X0 <= (´1´xor X1) and X2 or not X3;      -- geht problemlos
X0 <= X1 or EN;         -- geht nicht, da verschiedene Typen
ein_aus <=(X0=X1);      -- das geht, weil zuerst X0 mit X1 verglichen
                        -- wird und das Ergebnis eines Vergleichs wahr
                        -- oder falsch ist. Dieser boolesche Wert kann
                        -- dann auch nur einem Signal vom typ boolean
                        -- zugewiesen werden!
```

Abbildung 5: Typprüfung bei Zuweisungen

Die neun Werte des Typs std_logic sind in Tabelle 2 kurz erklärt.

Wert	Bedeutung	Verwendung
0	“starke” log. Null	Synthese, Simulation (wie 0 im Typ bit), technisch Push/Pull
1	„starke“ log. Eins	Synthese, Simulation (wie 1 im Typ bit), technisch Push/Pull
Z	hochohmig	Synthese, Simulation, technisch Tristate-Buffer, z.b. Bus
-	don’t care	Synthese, der Wert ist egal; speziell bei Wertetabellen
U	unbekannt	Simulation, der Wert ist nicht bekannt
X	Konflikt	Simulation, Erkennung von 0 gegen 1 - Treiberkonflikten
L	„schwache“ 0	Synthese, technisch open source (offener Emitter)
H	„schwache“ 1	Synthese, technisch open drain (offener Kollektor)
W	„schwaches“ X	Simulation, Erkennung von L gegen H - Treiberkonflikten

Tabelle 2: Wertevorrat in std_logic

Für den Schaltungsentwurf sind neben den Werten ´0´ und ´1´ noch speziell die Werte ´Z´ und ´-´ interessant.

2.2.1 Wert ´Z´

Der Wert ´Z´ kann einem Signal zugewiesen werden.
In der technischen Schaltung entsteht bei der **Zuweisung** ein Tristate-Buffer. Solange der Wert ´Z´ zugewiesen ist, wird der Buffer in den hochohmigen Zustand geschaltet. Wird dagegen (zu einem anderen Zeitpunkt) diesem Signal der Wert ´0´ oder ´1´ zugewiesen, dann wird der Buffer automatisch wieder auf Gegentaktbetrieb umgestellt und der zugewiesene Wert ausgegeben. Damit können also Bustreiber, die einen Tristatebetrieb erfordern, modelliert werden.
Aber Achtung: Wenn die Zielschaltung, beispielsweise ein CPLD oder FPGA, keine Tristate-Buffer bereitstellt, kann diese Methode natürlich nicht verwendet werden!
Die **Abfrage** eines Signals auf den Wert ´Z´ ist zwar in der Simulation sinnvoll, in der Synthese jedoch nicht.

2.2.2 Wert ´-´

Der Wert ´-´ kann ebenfalls einem Signal zugewiesen werden.
Bei der **Zuweisung** sucht sich das Synthesewerkzeug selbständig einen Wert (´0´ oder ´1´) aus, den das Signal dann in der technischen Schaltung führt. Welcher von beiden Werten das ist, wissen Sie nicht – sie haben ja angegeben, dass Ihnen das auch egal ist.
Bei der **Abfrage** würde man sich erhoffen, dass das Synthesewerkzeug selbständig eine Minimierung vornimmt. Das ist jedoch nicht der Fall; vielmehr müsste ein technisch nicht vorhandener Wert ´-´ verarbeitet werden können. Die Abfrage dieses Wertes ist also zwar möglich, wird aber in aller Regel weder in der Simulation noch in der Synthese zu den erwarteten Ergebnissen führen!

2.3 Vektoren

Eine wesentliche Erleichterung, die VHDL bietet, ist die Zusammenfassung von Signalen gleichen Typs zu Vektoren. Als Vektor wird dabei ein eindimensionales Array (Feld) verstanden. Der Laufindex des Vektors ist eine Integerzahl. Obwohl es mit dem Kommando **array** in VHDL prinzipiell möglich ist, selber beliebige Felder zu definieren, sind für die Standardtypen bit und std_logic entsprechende Felder als Typen bereits vordefiniert. Sie sollten diese vordefinierten Typen verwenden, weil diese Typen als Dualzahlen interpretiert werden können und Berechnungen und Vergleiche damit durchgeführt werden können.

2.3.1 Deklaration

Eine Deklaration eines Vektors sieht wie in Abbildung 6 gezeigt aus:

Syntax:
signal <signalnamen>: typ(<lower> to <upper>);
signal <signalnamen>: typ(<upper> downto <lower>);

Beispiele:

```
signal x: bit_vector(0 to 7);          -- acht Signale vom Typ bit: x(0), x(1), x(2), …,x(7)
signal a: std_logic_vector (2 to 4);   -- drei Signale vom Typ std_logic: a(2), a(3), a(4)
signal r: bit_vector(3 downto 0);      -- vier Signale vom Typ bit: r(0), r(1), r(2), r(3)
```

Abbildung 6: Syntax Vektordeklaration

Die Namen der Einzelsignale werden aus dem Signalnamen und dem Laufindex in runden Klammern gebildet. Der Signalvektor ´signal ena_2: bit_vector(3 to 5)´ besteht also aus den Einzelsignalen ena_2(3), ena2_(4) und ena_2(5). Die Einzelsignale ´e1, e2 e3' können dagegen nicht als Vektor 'e': geschrieben werden, da hier die runden Klammern fehlen!

2.3.2 Zuweisungen und Verknüpfungen

Mit Vektoren ist es besonders einfach, mit nur einer Anweisung parallel eine Operation auf alle Einzelsignale (oder eine Teilmenge davon) auszuführen Einige Beispiele zeigt Tabelle 3.

```
signal a,b,c: bit_vector(0 to 3);  -- Deklaration dreier Vektoren

c <= a or b;                       -- bitweises oder auf alle Elemente
c <= ('1','0','0','0');            -- Zuweisung als Aggregat
c <= "1000";                       -- Zuweisung als Bitstring
c <= x"A";                         -- Zuweisung als Hexadezimalzahl
c <= a(0) & "001";                 -- Zusammensetzen mit &
c(1 to 2) <= "11";                 -- Auswahl eines Teilvektors im Ziel
c <= a(0 to 1) & '1' & b(0);       -- Auswahl eines Teilvektors in der Quelle
(w,x,y,z) <= c;                    -- Zuweisung von c an Einzelsignale w,x,y,z
```

Ziel				VHDL-Kommando
c(0)	c(1)	c(2)	c(3)	
a(0) or b(0)	a(1) or b(1)	a(2) or b(2)	a(3) or b(3)	c <= a or b;
1	0	0	0	c <= ('1', '0', '0', '0');
1	0	0	0	c <= "1000";
1	0	1	0	c <= x"A";
a(0)	0	0	1	c <= a(0) & "001";
unverändert	1	1	unverändert	c(1 to 2) <= "11";
a(0)	a(1)	1	b(0)	c <= a(0 to 1) & '1' & b(0);

Tabelle 3: Arbeiten mit Vektoren

Beachten Sie dabei besonders, dass Sie sich sowohl aus dem Ziel als auch aus der Quelle jeweils Teile herausgreifen können und dass Sie Teile mit ‚&' zu breiteren Vektoren zusammensetzen können. Sie müssen lediglich darauf achten, dass die Gesamtbreite von Ziel und Quelle bzw. Quellen jeweils gleich ist. Speziell bei der Zuweisung konstanter Werte sehen Sie, dass Sie auch eine Darstellung als Hexadezimalzahl mit x"zahl" bzw. als Oktalzahl mit o"zahl" wählen können. Die angegebene Zahl wird dabei in einen Vektor mit binären Einzelwerten umgerechnet.
Sie können bei der Zuweisung konstanter Werte den Unterstrich zur besseren Kennzeichnung von zusammengehörenden Gruppen verwenden. Er hat aber sonst keine Bedeutung, d.h. er vertritt keine Position im Vektor. Falls Sie in einem Vektor alle bisher nicht belegten Positionen (d.h. Einzelsignale) mit demselben Wert belegen wollen, dann können Sie das Kommando (others => 'wert') verwenden (Abbildung 7).

```
signal c: bit_vector(0 to 7);

c <= "01001001";        -- diese und die beiden folgenden
c <= "0100_1001";       -- Zuweisungen haben gleiche Wirkung
c <= x"49";

c <= (others => '0');   -- allen Einzelsignalen von c wird '0'
-- zugewiesen
```

Abbildung 7: Zuweisung an Vektoren

2.3.3 Positionsbestimmung im Vektor

Die Anordnung der Einzelsignale im Vektor geschieht immer von links nach rechts. Bei der Zuweisung bzw. bitweisen Verknüpfung von Vektoren werden immer die Einzelsignale an gleicher Position miteinander verknüpft. Bei der Deklaration des Vektors kann mit Hilfe der Schlüsselworte to bzw. downto festgelegt werden, ob das am weitesten links stehende Einzelsignal die niedrigste bzw. höchste Nummer erhält.
Der Unterschied in beiden Deklarationsvarianten kommt vor allem dann zum Tragen, wenn der Vektor nicht mehr als ein Feld von Einzelsignalen sondern als Dualzahl interpretiert wird.

```
signal x: std_logic_vector(0 to 3);
signal y: std_logic_vector(3 downto 0);
x <= x"3";                  -- Zuweisung erfolgt nach Position
y <= x"3";                  -- Zuweisung erfolgt nach Position
r <= x+y;                   -- Arithmetik erfolgt nach Stellenwertigkeit
```

Vektor x				**Vektor y**				**Anweisung**
x(0)	x(1)	x(2)	x(3)	y(3)	y(2)	y(1)	y(0)	
0	0	1	1					x <= x ´´3´´ ;
				0	0	1	1	y <= x ´´3´´ ;
Stellenwertigkeit				**Stellenwertigkeit**				**Interpretation**
2^0	2^1	2^2	2^3	2^3	2^2	2^1	2^0	
0	0	1	1					x = 4+8 = 12
				0	0	1	1	y = 2+1 = 3

Tabelle 4: Position und Stellenwertigkeit

Das Beispiel nach Tabelle 4 zeigt, dass die Variante downto der gewohnten Stellenwertigkeit für Dualzahlen entspricht. Wenn Sie also Standardarithmetik auf Vektoren betreiben wollen, dann sollten Sie die Vektoren mit downto deklarieren.

3 Aufbau einer Schaltungsbeschreibung

Sie kennen jetzt die Bedeutung von Typen, können Signale in VHDL deklarieren und sie entweder einzeln oder als Vektoren bearbeiten. In diesem Kapitel lernen Sie den grundsätzlichen Aufbau einer Schaltungsbeschreibung, speziell die Beschreibung eines einzelnen Blocks, kennen. Verwenden Sie am besten für jeden Block eine eigene Datei und benennen Sie die Datei so, dass Sie die darin enthaltene Schaltung auch wiedererkennen. Die Datei besteht aus drei getrennten Abschnitten

- Header mit Bibliotheks- und Packageeinbindungen
- Entity für die Schnittstellendefinition
- Architecture für die Funktionsbeschreibung.

3.1 Header

In diesem Teil legen Sie die Definitionen fest, die für die folgende Schaltungsbeschreibung gelten sollen. Da diese Definitionen nur für die unmittelbar folgende Schaltung (definiert durch ihre Schnittstelle) gilt, müssten Sie den Header vor der nächsten Schaltung in derselben Datei wiederholen. Da Sie ja in jeder Datei nur eine Schaltung beschreiben, stellt sich das Problem für Sie nicht ...
Der Header sieht so gut wie immer wie in Abbildung 8 gezeigt aus.

```
library ieee;                          -- die verwendete Bibliothek
use ieee.std_logic_1164.all;           -- die Definition der Grundtypen und Werte
use ieee.std_logic_unsigned.all;       -- die Definition einer unsigned-Arithmetik
```

Abbildung 8: Typischer Header

Sofern Sie keine Arithmetik auf Dualzahlen betreiben, können Sie die dritte Zeile weglassen. Diese Art der Arithmetik funktioniert nur auf Vektoren des Typs std_logic und interpretiert die Zahlen als vorzeichenlos. Es gibt auch andere Arithmetiken, die teilweise mehr leisten, aber auch herstellerspezifisch sind (speziell Synopsys). Die angegebene Arithmetik ist standardisiert und funktioniert mit allen standardkonformen Syntheseprogrammen.
Der Header ist in „C“ vergleichbar mit #include-Anweisungen zu Beginn eines Programms.

3.2 Die Schnittstelle – Entity

Der nächste Abschnitt ist die Definition der Schnittstelle der Schaltung. An dieser Stelle wird der Name der Blockes, der die Schaltung repräsentiert, definiert sowie sämtliche Signale, die an diesen Block angeschlossen sind. Dieser Block ist in „C“ vergleichbar mit der Deklaration einer Funktion. Auch hier wird der Funktionsname sowie die Typen und Zahl der übergebenen Parameter definiert.
An einem Block gibt es Eingangssignale, Ausgangssignale sowie Signale, die ihre Richtung wechseln können.

Signalrichtung	VHDL-Schlüsselwort	Bemerkung
Eingang	in	kann nur gelesen werden (Quelle)
Ausgang	out	kann nur geschrieben werden (Ziel)
Bidirektional	inout	kann sowohl gelesen als auch beschrieben werden

Tabelle 5: Signalrichtungen

Die formale Syntax für die einfache Deklaration einer Schnittstelle (entity) zeigt Abbildung 9.

```
entity <blockname> is
port
(
  <signalnamen>: <richtung> <typ>;  -- bei weiteren Signalen: ´;´
  <signalnamen>: <richtung> <typ>   -- beim letzten Signal KEIN ´;´
);                                  -- dafür nach der Klammer der ´;´
end;
```

Abbildung 9: Rahmen einer Entity

Mit dieser Kenntnis können Sie jetzt die Schnittstelle des Multiplexers aus Abbildung 1 wie in Abbildung 10 gezeigt deklarieren.

```
entity MULTIPLEXER is
port
(
  A0, A1, A2, A3: in bit;      -- Datenwort A
  B0, B1, B2, B3: in bit;      -- Datenwort B
  S:              in bit;      -- Steuereingang zur Auswahl (0: A, 1: B)
  Y0, Y1, Y2, Y3: out bit      -- Datenausgang
);
end;
```

Abbildung 10: Deklaration einer Entity

Bitte beachten Sie zunächst, dass der Blockname exakt dem Namen im Schaltplan entsprechen muss. Das betrifft auch die Groß-/Kleinschreibung. Innerhalb einer reinen VHDL-Beschreibung wäre dies zwar nicht erforderlich; wenn Sie aber den Block in einer anderen Umgebung (wie dem Schaltplan) wiedererkennen wollen, dann kommt es ja auch darauf an, ob das andere Programm eine Unterscheidung zwischen Groß- und Kleinschreibung vornimmt. Um mögliche Fehler auszuschließen verwenden Sie daher gleiche Schreibweisen. Dies betrifft natürlich auch die Signalnamen!
Weiterhin sehen Sie, dass Sie mehrere Signale vom gleichen Typ auch zusammen deklarieren können. Es empfiehlt sich, die Signale bei der Deklaration sinnfällig zu gruppieren.
Des weiteren fördert es die Lesbarkeit, wenn Sie Einrückungen verwenden, um den Gültigkeitsbereich eines Konstrukts (hier ‚port') zu verdeutlichen.
In diesem Beispiel können Sie die Signale nicht als Vektor deklarieren, weil ja die runden Klammern nicht im Signalnamen auftauchen. Für den Block „Zaehler" ist das aufgrund der gewählten Signalnamen aber möglich (Abbildung 11).

```
entity ZAEHLER is
port
(
  CLK:   in bit;                        -- Taktsignal
  RESET: in bit;                        -- asynchroner Reset auf Null
  Q:     out bit_vector(3 downto 0)     -- Ausgabe
);
end;
```

Abbildung 11: Entitydeklaration mit Vektor

Die in der entity deklarierten Signale sind sowohl außerhalb des Blockes sichtbar (das ist für die Verbindung zu anderen Blöcken im Schaltplan wichtig) als auch innerhalb der nun folgenden Schaltungsbeschreibung. Auch in „C" sind ja die übergebenen Parameter unmittelbar, d.h. ohne weitere Deklaration, in der Funktion verwendbar. Im Beispiel (Abbildung 1) sehen Sie, dass das Ausgangsignal Q(0), das in der entity ZAEHLER deklariert ist, mit dem Eingangsignal A0 der entity MULTIPLEXER im Schaltplan verbunden ist. Die beiden entities in den beiden VHDL-Dateien brauchen dazu nichts voneinander zu wissen.

3.3 Die Funktion - Architecture

Der letzte Abschnitt ist die Beschreibung der Funktion des eben definierten Blocks. Es kann durchaus für eine entity mehrere verschiedene Funktionsbeschreibungen geben. Für einen Zähler gäbe es die Möglichkeit, ihn als Synchron- bzw. Asynchronzähler auszuführen. Je nach Bedarf könnte dann eine der beiden Möglichkeiten für einen bestimmten Entwurf verwendet werden. Zur Identifizierung muss daher auch die Funktionsbeschreibung einen eindeutigen Namen erhalten. In diesem Kurs wird immer nur eine Beschreibung pro entity verwendet und sie heißt immer 'verhalten'.
Die formale Syntax für die einfache Deklaration einer Funktionsbeschreibung (architecture) sieht wie folgt aus:

```
architecture <beschreibungsname> of <blockname> is
 -- hier können lokale Signale deklariert werden
begin

 -- hier steht die Funktionsbeschreibung (nebenläufig)

end;
```

Abbildung 12: Rahmen einer Architecture

Damit ist der größte Teil der Funktionsbeschreibung des Multiplexers bereits verständlich (Abbildung 13).

```
architecture verhalten of MULTIPLEXER is
  signal a,b,y: bit_vector (0 to 3);  -- Hilfssignal als Vektor

begin
  a <= (a0,a1,a2,a3);          -- Zuweisung der Eingangssignale an den Vektor A
  b <= (b0,b1,b2,b3);          -- Zuweisung der Eingangssignale an den Vektor B

  y <= a when (s='0') else b; -- Auswahl durch S

  y0 <= y(0); y1 <= y(1);      -- Zuweisung des Ergebnisses an die Ausgangssignale
  y2 <= y(2); y3 <= y(3);
end verhalten;
```

Abbildung 13: Funktion des Multiplexers

Zunächst sehen Sie, dass der Blockname genau wie in der entity deklariert lautet. In der zweiten Zeile werden lokale Hilfssignale definiert. Da der Multiplexer ja vier Signalpaare jeweils völlig identisch behandelt, ist eine Vektordarstellung dieser Signale angebracht.
In den ersten beiden Zeilen nach dem 'begin' werden zunächst die Vektoren a und b mit den entsprechenden Signalen aus der Schnittstellendeklaration belegt.
Anschließend erfolgt in Abhängigkeit vom Wert von s die bedingte Zuweisung an den internen Ergebnisvektor y. Dieses Kommando wird später erklärt.
Zuletzt werden die Signale y0 bis y3 aus der Schnittstellendeklaration wieder aus dem internen Vektor y gebildet.
Natürlich hätte der Multiplexer auch gleich mit Signalnamen a(0), .. a(3), die sich in VHDL als Vektor darstellen lassen, definiert werden können. In diesem Fall hätten die Deklaration der internen Vektoren a, b und y sowie die einleitenden und abschließenden Umkopieraktionen vollständig entfallen können. Dies ist hier nur aus Demonstrationsgründen nicht erfolgt.

4 Nebenläufige und sequentielle Umgebungen

Ein wesentlicher Unterschied zwischen VHDL und einer normaler rein sequentiellen Programmiersprache wie „C“ besteht darin, dass Anweisungen in VHDL in der Regel nebenläufig, d.h. parallel abgearbeitet werden. Das liegt daran, dass aus der Beschreibung eine Schaltung erzeugt wird, deren einzelne Bestandteile (Gatter) ebenfalls ständig parallel arbeiten. Sie müssen sich bei der Beschreibung in VHDL zu jedem Zeitpunkt darüber klar sein, ob Sie sich derzeit in einer nebenläufigen oder einer sequentiellen Umgebung befinden.
Eine Umgebung ist dabei zwischen einem ’begin’ und dem zugehörigen ’end’ definiert. Die erste, von der architecture bereits eröffnete, Umgebung ist nebenläufig.
VHDL unterstützt Sie bei der Unterscheidung zwischen nebenläufigen und parallelen Umgebungen dadurch, dass die meisten Kommandos nur in einer der beiden Umgebungen erlaubt und bei illegaler Verwendung solcher Kommandos Fehlermeldungen erzeugt werden.

4.1 Nebenläufige Umgebung

In dieser Umgebung werden alle Kommandos bzw. zu Blöcken zusammengefassten Kommandos parallel ausgeführt. Die Reihenfolge, in der Anweisungen in der Umgebung stehen, hat keinen Einfluss auf das Ergebnis! Sehen Sie sich dazu den Code in Abbildung 14 an, bei der die Reihenfolge der Zeilen gegenüber Abbildung 13 umgestellt worden ist.

```
architecture verhalten of MULTIPLEXER is
  signal a,b,y: bit_vector (0 to 3);  -- Hilfssignal als Vektor

begin
  a <= (a0,a1,a2,a3);          -- Zuweisung der Eingangssignale an den Vektor A
  b <= (b0,b1,b2,b3);          -- Zuweisung der Eingangssignale an den Vektor B
  y0 <= y(0); y1 <= y(1);      -- Zuweisung des Ergebnisses an die Ausgangssignale
  y2 <= y(2); y3 <= y(3);

  y <= a when (s='0') else b; -- Auswahl durch S
end verhalten;
```

Abbildung 14: Nebenläufigkeit I (erlaubt)

Da die erste Umgebung in einer Architecture nebenläufig ist, spielt die Reihenfolge, in der Anweisungen aufgeschrieben werden, keine Rolle für die Funktion. In einer sequentiellen Programmiersprache wäre die Berechnung von y (letzte Zeile) **nach** der Verwendung in der dritten und vierten Zeile doch etwas merkwürdig.
Die Nebenläufigkeit führt allerdings auch dazu, dass die Zuweisung verschiedener Werte an ein- und dasselbe interne Signal illegal ist.

	VHDL	C
1	`signal x0,x1,a,b,y1,y2: bit;`	`int a,b,c;`
2	`c  <= a and b;`	`c = a + b;`
3	`y1  <= x0 when (c = ‘0’) else x1;`	`if (c>1) then …`
4	`c  <= a or b;`	`c = a-b;`
5	`y2  <= x0 when (c = ‘0’) else x1;`	`if (c>0) then …`

Abbildung 15: Nebenläufigkeit II (verboten)

Die Idee ist in beiden Fällen, ein Hilfssignal ’c’ zur leichteren Lesbarkeit vor der Auswertung zu berechnen (Zeilen 2 und 4). In „C“ ist das auch kein Problem, da die Anweisungen ja nacheinander abgearbeitet werden. In einer nebenläufigen VHDL-Umgebung werden aber alle Zeilen, speziell also auch die beiden Zuweisungen an dasselbe Signal ’c’, gleichzeitig ausgeführt. Damit entsteht ein Konflikt, da ’c’ nicht zur gleichen Zeit das Ergebnis der beiden Verknüpfungen von ’a’ und ’b’ sein kann.

4.2 Sequentielle Umgebung

4.2.1 Der Prozess

Die einzige sequentielle Umgebung, die in VHDL existiert, wird durch einen Prozess definiert (Abbildung 16).

```
process <empfindlichkeitsliste>
 -- hier können lokale Signale oder Variablen deklariert werden
begin
 -- hier ist eine sequentielle Umgebung
end process;
```

Abbildung 16: Prozess, Syntax

Die Anweisungen werden in der sequentiellen Umgebung nacheinander abgearbeitet. Wenn das Prozessende erreicht ist, startet die Ausführung sofort wieder am Prozessbeginn.
Eine parallele Umgebung kann auch mehrere Prozesse beinhalten. Diese Prozesse werden dann alle parallel ausgeführt, wobei die Anweisungen innerhalb eines Prozesses nacheinander abgearbeitet werden. Prozesse werden sehr häufig für die Beschreibung sequentieller Schaltungen (Automaten) verwendet. Ein Moore-Automat besteht ja definitionsgemäß aus zwei Schaltnetzen (Zustandsübergangsfunktion und Ausgabefunktion) sowie dem Zustandsspeicher. Hier bietet es sich geradezu an, die beiden Schaltnetze nebenläufig zu beschreiben und den sequentiell arbeitenden Speicher innerhalb derselben architecture mit einem Prozess zu beschreiben.
Die Empfindlichkeitsliste soll alle Signale enthalten, die innerhalb des Prozesses zu einer Änderung führen können.
Die lokal definierten Signale und Variablen sind grundsätzlich nur innerhalb des Prozesses sichtbar. Natürlich muss der Prozess auch Ausgangssignale an seine Umgebung zur weiteren Auswertung weiterleiten können. Im Beispiel des Moore-Automaten wäre das der aktuelle Zustand. Dies kann nur über Signale erfolgen, die außerhalb des Prozesses deklariert sind. Dabei muss beachtet werden, dass dann Zuweisungen an diese Signale nur im Prozess erlaubt sind. Das wird offensichtlich, wenn man sich überlegt, dass die Signalzuweisungen zwar in einem Prozess nacheinander ablaufen, diese aber mit allen nebenläufigen Anweisungen außerhalb des Prozesses konkurrieren. Betrachten Sie das Beispiel in Abbildung 17.

```
architecture verhalten of ZAEHLER is
  signal qint: std_logic_vector(3 downto 0);

begin

  process (reset, clk)                -- Prozess hängt von clk und reset ab
  begin
    if (reset='0') then qint <= x"0"; -- Asynchrones Rücksetzen auf Null
    elsif (clk='1') and clk'event     -- sonst warten auf Taktflanke
    then
      qint <= qint+1;                 -- Zählerstand hochzählen
    end if;
  end process;

  q <= qint;                          -- Nebenläufige Ausgabe

end;
```

Abbildung 17: Zähler als Prozess

Es handelt sich um eine Funktionsbeschreibung des Zählers aus Abbildung 1. Innerhalb der nebenläufigen Umgebung der Architecture laufen ein Prozess und eine Zuweisung (letzte Zeile) parallel. Sie können den Zähler als einen Moore-Automaten betrachten, hier wäre die Ausgabefunktion der Zustandsvektor ’qint’ selbst. Der Prozess selbst reagiert sowohl auf den Takt ’clk’ als auch das asynchrone Rücksetzsignal ’reset’. Daher stehen diese beiden Signale

in der Empfindlichkeitsliste. Innerhalb des Prozesses wird eine bedingte Zuweisung (Syntax wird später erklärt) ausgeführt. Falls 'reset' aktiv ist wird der Zählerstand sofort zurückgesetzt. Ansonsten wird auf eine steigende Taktflanke gewartet. Trifft diese ein, dann wird der Zählerstand um eins erhöht. Da hier das Prozessende erreicht ist, beginnt die Ausführung wieder am Prozessbeginn. Damit wird wieder das Signal 'reset' abgefragt bzw. auf die nächste steigende Taktflanke gewartet.

4.2.2 Signalzuweisung innerhalb und außerhalb des Prozesses

Betrachten Sie jetzt den Code in Abbildung 18.

```
architecture verhalten of ZAEHLER is
  signal qint: std_logic_vector(3 downto 0);

begin

  qint <= x"0" when (reset='0');       -- Asynchrones Rücksetzen auf Null

  process (clk)                        -- Prozess hängt von clk ab
  begin
    if (clk='1') and clk'event         -- warten auf Taktflanke
    then
      qint <= qint+1;                  -- Zählerstand hochzählen
    end if;
  end process;

  q <= qint;                           -- Nebenläufige Ausgabe

end;
```

Abbildung 18: Zuweisung außerhalb des Prozesses

Die beschriebene Funktion ist prinzipiell dieselbe, nur wird hier das asynchrone Rücksetzsignal 'reset' außerhalb der Prozessumgebung abgefragt. Das erscheint eigentlich als sinnvoll, denn der Rücksetzvorgang ist ja nicht an den Takt gebunden. Leider kommt es dabei aber zu zwei nebenläufigen Zuweisungen zu dem Signal 'qint', einmal außerhalb des Prozesses und einmal innerhalb des Prozesses. Damit ist diese Beschreibung nicht möglich.

4.2.3 Zeitpunkt der Signalaktualisierung

Weiterhin müssen Sie in einer sequentiellen Umgebung beachten, dass Signalzuweisungen, die innerhalb eines Prozesses vorgenommen werden (in Abbildung 17 die Zeile 'qint <= qint +1') erst mit der nächsten Taktflanke wirksam werden. Wenn Sie also ein Signal in einem Prozess abfragen, dann wird der Wert **vor** der Taktflanke verwendet und nicht der gerade eben erst neu berechnete Wert. Sie können sich auch dies erklären, wenn Sie an den Speicher des Automaten denken. Auch hier wird der eben mit der Zustandsübergangsfunktion berechnete neue Zustand erst mit der nächsten Taktflanke in den Zustandsspeicher übernommen. Bis dahin steht im Speicher der alte (also bisherige) Wert zur Auswertung zur Verfügung.

4.2.4 Unerwünschte Latches

Die letzte Falle, die in einer sequentiellen Umgebung lauert, ist die Abfrage eines Signals vor einer Zuweisung. In einer nebenläufigen Umgebung hatte das ja keine weiteren Folgen, da die Reihenfolge der Anweisungen ohnehin keine Bedeutung hat. Wenn aber in einer sequentiellen Umgebung auf ein Signal zugegriffen wird, dem bisher kein Wert zugewiesen wurde, dann muss offensichtlich auf einen älteren, d.h. gespeicherten, Wert zurückgegriffen werden. Das Synthesewerkzeug fügt deshalb ein speicherndes Element, meist ein Latch, ein. Genau das wünschen Sie aber in der Regel nicht. Achten Sie also darauf, dass Sie in einer sequentiellen Umgebung einem Signal zuerst einen Wert zuweisen, ehe Sie es verwenden.

4.2.5 Variablen im Prozess

Wie in Kapitel 4.2.3 dargelegt, werden Signale grundsätzlich erst mit dem nächsten Prozessdurchlauf aktualisiert. Da es manchmal wünschenswert ist, auf berechnete Ergebnisse sofort zugreifen zu können, können innerhalb eines Prozesses Variablen deklariert werden. Für Variablen gelten die gleichen Konventionen (Typen, Vektoren) wie für Signale, nur wird statt des Schlüsselwortes 'signal' das Schlüsselwort 'variable' verwendet und Zuweisungen werden nicht mit '<=' gekennzeichnet sondern mit ':='.

```
14  architecture verhalten of ZAEHLER is
15    signal qint: std_logic_vector(3 downto 0);
16
17  begin
18
19    process (reset, clk)                -- Prozess hängt von reset und clk ab
20      variable V: std_logic_vector(3 downto 0);
21    begin
22      if reset='0'                      -- Asynchrones Rücksetzen auf Null
23      then
24        qint <= x"0";
25        V := x"0";
26      elsif (clk='1') and clk'event     -- warten auf Taktflanke
27      then
28        qint <= qint+1;                 -- Zählerstand hochzählen
29        V := V+1;                       -- Hilfsvariable hochzählen
30      end if;
31    end process;
32
33    q <= qint;                          -- Nebenläufige Ausgabe
34
35  end;
```

Abbildung 19: Variablen in Prozessen

In Abbildung 19 ist der bereits bekannte Zähler um die Deklaration und Verwendung einer Variablen 'V' erweitert. Beachten Sie zunächst die lokale Deklaration in Zeile 20. Die Variable ist damit nur innerhalb des Prozesses sichtbar.
In Zeile 25 wird die Variable bei einem Reset ebenfalls auf Null gesetzt; beachten Sie hier die geänderte Syntax für die Zuweisung.
In Zeile 29 wird die Variable synchron mit dem Signal 'qint' um eins erhöht. Der Unterschied ist aber jetzt, dass in weiteren Anweisungen im 'then'-Zweig (Zeilen 27 bis 30) bereits der neue Wert zur Verfügung steht. Wenn also vor der Taktflanke sowohl 'qint' als auch 'V' die Hexadezimalzahl 8 enthalten haben sollten, dann enthält nach Zeile 29 das Signal 'qint' **weiterhin** die Zahl 8 während die Variable 'V' **bereits** die Zahl 9 enthält.
Erst mit der nächsten Taktflanke wird dem Signal 'qint' die neue Zahl 9 zugewiesen, so dass Signal und Variable wieder synchron laufen.

5 Anweisungen

5.1 Einfache Verknüpfungen

Signale und Variablen können miteinander verknüpft werden und das Ergebnis kann einem Signal oder einer Variablen zugewiesen werden. Die Verknüpfung und Zuweisung ist sowohl in nebenläufigen als auch sequentiellen Umgebungen zulässig. Die verfügbaren Verknüpfungen sind in Abbildung 20 gezeigt.

Schlüsselwort	Operation	Bemerkung
not	Negation	hat Vorrang vor allen anderen Operatoren
and	und	Bei „Serienschaltung“ kann die Klammerung entfallen, ’a **and** b **and** c’ ist zulässig
or	oder	
nand	negiertes und	„Serienschaltung“ ohne Klammerung ist nicht zulässig, ’a **nor** b **nor** c’ ist also verboten!
nor	negiertes oder	
xor	Antivalenz (exklusiv oder)	
xnor	Äquivalenz (neg. xor)	

Abbildung 20: Boolesche Operatoren

Grundsätzlich ist es empfehlenswert, Klammern zu setzen, um die Reihenfolge der Verknüpfungen unmittelbar einsichtig zu machen.
Damit können Sie den Multiplexer aus Abbildung 1 in VHDL wie in Abbildung 21 gezeigt beschreiben.

```
library ieee;                          -- die verwendete Bibliothek
use ieee.std_logic_1164.all;           -- die Definition der Grundtypen und Werte


entity MULTIPLEXER is
port
(
  A0, A1, A2, A3: in bit;      -- Datenwort A
  B0, B1, B2, B3: in bit;      -- Datenwort B
  S:              in bit;      -- Steuereingang zur Auswahl (0: A, 1: B)
  Y0, Y1, Y2, Y3: out bit      -- Datenausgang
);
end;


architecture verhalten of MULTIPLEXER is
begin
  y0 <= (a0 and not s) or (b0 and s);  -- Multiplexergleichung
  y1 <= (a1 and not s) or (b1 and s);
  y2 <= (a2 and not s) or (b2 and s);
  y3 <= (a3 and not s) or (b3 and s);
end verhalten;
```

Abbildung 21: Beispiel für Verknüpfung

5.2 Arithmetische Operatoren

In VHDL sind auch komplexere Operationen möglich. Für den Schaltungsentwurf kommen dabei aber meist nur Addition und Subtraktion in Frage, da Operationen wie Multiplikation oder Division in der Regel zu unverhältnismäßig aufwendigen Schaltungen führen würden. Für die Anwendung arithmetischer Operationen muss bekannt sein, auf welchem Typ die Operation arbeitet und wie eine Dualzahl interpretiert wird. Dies wird durch das verwendete Package (das ist die 'use'-Anweisung im Header) festgelegt. Leider stehen nicht in allen Entwurfswerkzeugen alle Möglichkeiten zur Verfügung.
Am sichersten fahren Sie, wenn Sie die im Standard definierte unsigned-Arithmetik auf den Typ std_logic_vector verwenden. Dabei müssen die Signale oder Variablen vom Typ std_logic_vector sein und die Interpretation des Vektors erfolgt als vorzeichenlose Dualzahl.
Falls Sie einen anderen Typ bearbeiten wollen, müssen Sie eine explizite Typkonversion vorsehen. Ebenso können Sie eine vorzeichenbehaftete Interpretation der Zahl, z.B. im Zweierkomplement, selbst ableiten.

Schlüsselwort	Operation	Bemerkung
+	Addition	
-	Subtraktion	
=	gleich	unabhängig vom Typ, geht ohne Arithmetikpackage
/=	ungleich	erzeugt, wie der Vergleich auf Gleichheit, einen booleschen Wert (Typ **boolean**) Anwendung in Abfragen wie **if/elsif/when**
<	kleiner	
<=	kleiner gleich	
>	größer	
>=	größer gleich	

Abbildung 22: Vergleiche und Arithmetik

Sehen Sie sich den Code in Abbildung 23 an. Es handelt sich um eine vollständige Beschreibung des Zählers aus Abbildung 1.

```
 1  library ieee;
 2  use ieee.std_logic_1164.all;
 3  use ieee.std_logic_unsigned.all;
 4
 5  entity ZAEHLER is
 6  port
 7  (
 8    CLK:   in bit;                          -- Taktsignal
 9    RESET: in bit;                          -- asynchroner Reset auf Null
10    Q:     out bit_vector(3 downto 0)       -- Ausgabe
11  );
12  end;
13
14  architecture verhalten of ZAEHLER is
15    signal qint: std_logic_vector(3 downto 0); -- internes Hilfssignal
16
17  begin
18
19    process (reset, clk)                    -- Prozess hängt von reset und clk ab
20    begin
21      if (reset='0') then qint <= x"0";     -- Asynchrones Rücksetzen auf Null
22      elsif (clk='1') and clk'event         -- warten auf Taktflanke
23      then
24        if (qint > x"8")                    -- Abfrage auf >8, da Signal noch den
25        then                                -- vorherigen Zählerstand enthält
26          qint <= x"0";
27        else
28          qint <= qint+1;                   -- Zählerstand hochzählen
29        end if;
30      end if;
31    end process;
32
33    q <= to_bitvector(qint);                -- Nebenläufige Ausgabe mit Typkonversion
34  end;
35
```

Abbildung 23: Arithmetik und Vergleich im Zähler

Zunächst einmal wird in Zeile 3 das Package für vorzeichenlose Arithmetik auf den Typ ’std_logic_vector’ eingebunden. Sie benötigen dies sowohl für den Vergleich in Zeile 24 als auch die Addition in Zeile 28.
In Zeile 15 wird ein lokales Hilfssignal von diesem Typ deklariert. Selbst wenn das Ausgabesignal (hier ’Q’ in Zeile 10) bereits als ’std_logic_vector’ deklariert worden wäre, hätten Sie dennoch ein internes Hilfssignal deklarieren müssen. Der Grund ist, dass das Signal ’Q’ ja ein reines Ausgabesignal mit der Richtungsangabe ’out’ ist . Diese Signale können aber nicht gelesen werden! Also könnten Sie auch nicht (wie in Zeile 28) auf den letzten Zählerstand zugreifen und damit letztlich nicht zählen.
Da der Zähler nur bis 9 zählen soll, wird in Zeile 24 wird auf Zählerüberlauf abgefragt. Es wäre natürlich auch möglich, auf ’qint = 9’ abzufragen. In diesem Fall würde aber der Zähler bei den fehlerhaften bzw. beim Einschalten zufällig entstandenen Zuständen 10 bis 15 nicht sofort zurückgesetzt.
Die Abfrage erfolgt auf größer als 8, da das Signal ’qint’ ja wie in Abschnitt 4.2.3 beschrieben um einen Takt „nachhinkt“. Wenn der Zähler vor dem Takt auf 8 stand, dann wird zwar mit der Zuweisung in Zeile 28 der Zählerstand auf 9 erhöht. Für Abfragen wird dies aber erst **nach dem nächsten Takt** wirksam.
Zuletzt wird der aktuelle Zählerstand in Zeile 33 ausgegeben. Hier ist eine explizite Typkonversion erforderlich, da ja das interne Signal einen anderen Typ als das Ausgabesignal hat.

5.3 with/select

Mit dieser **nebenläufigen** Anweisung kann **ein** Auswahlsignal **mehrfach** abgefragt werden und in Abhängigkeit von dem gewählten Zweig können **einem** Signal beliebige Verknüpfungen zugewiesen werden. Die Anweisung führt in der Schaltung häufig zu einem Multiplexer, bei dem das Auswahlsignal an den Steuereingängen anliegt und die einzelnen Verknüpfungen an den Dateneingängen des Multiplexers anliegen. Das Ergebnis kann dann am Ausgang des Multiplexers abgeholt werden.

with <auswahlsignal> **select**
ergebnis <= <Verknüpfung_1> **when** <auswahlwert_1>,
<Verknüpfung_2> **when** <auswahlwert_2>,
<Verknüpfung_n> **when others**;

Abbildung 24: Syntax with/select

Die Syntax der Anweisung zeigt Abbildung 24. Zu beachten ist zunächst, dass das Auswahlsignal zwar ein Vektor sein, der aber bereits vorher aus Einzelsignalen zusammengesetzt worden sein muss. Deklarieren Sie, wenn notwendig, einfach ein lokales Hilfssignal. Natürlich muss der Typ des Ergebnisses mit dem Typ der einzelnen Verknüpfungen jeweils übereinstimmen. Ebenso muss der Typ des Auswahlsignals mit den Typen der Auswahlwerte übereinstimmen.
Wenn Sie nicht alle Möglichkeiten erschöpfend behandeln, dann sollten Sie noch eine Defaultzuweisung mit der Anweisung 'when others' vornehmen. Speziell beim Typ 'std_logic' haben Sie ja neun Werte. Bei einer erschöpfenden Aufzählung müssten Sie auch alle Werte außer '0' und '1' abfragen – was Sie natürlich nicht tun. Statt dessen verwenden Sie für die letzte Auswahl einfach 'others'. Der Code für den Dekoder aus Abbildung 1 in Abbildung 25 ist damit selbsterklärend.

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

entity DEKODER is
port
(
  X0,X1,X2,X3:   in  bit;
  EN:            in  bit;
  a,b,c,d,e,f,g: out bit
);
end;

architecture behaviour of DEKODER is
signal y: bit_vector(0 to 6);
signal sel: bit_vector(3 downto 0);

begin
  sel <= (x3,x2,x1,x0);                -- Zusammensetzung erforderlich
  with sel select                      -- Umsetzung Dualzahl in
  y <= "1111110" when x"0",            -- Siebensegmentcode
       "0110000" when x"1",
       "1101101" when x"2",
       "1111001" when x"3",
       "0110011" when x"4",
       "1011011" when x"5",
       "1011111" when x"6",
       "1110000" when x"7",
       "1111111" when x"8",
       "1111011" when x"9",
       "0000000" when others;          -- hier bleibt die Anzeige dunkel

  a <= y(0) and en; b <= y(1) and en;  -- Enable wirkt auf alle Ausgänge
  c <= y(2) and en; d <= y(3) and en;
  e <= y(4) and en; f <= y(5) and en;
  g <= y(6) and en;

end;
```

Abbildung 25: Dekoder mit with/select

5.4 when/else

Auch diese Anweisung ist wie with/select nur in **nebenläufigen** Umgebungen zulässig. Mit dieser Anweisung können jedoch wechselnde Signale auf zutreffende Bedingungen geprüft werden und in Abhängigkeit davon einem Ergebnissignal verschiedene Verknüpfungen zugewiesen werden. Außerdem ist es möglich, dass sich die Bedingungen überlappen – in diesem Fall wird die erste zutreffende Bedingung ausgewählt. Dies war bei with/select nicht möglich, da dort alle Verzweigungen disjunkt sein mussten.

```
<ergebnis>   <=      < Verknüpfung_1> when <Bedingung_1>
                 else  < Verknüpfung_2> when <Bedingung_2>
                 else  <Verknüpfung_n>;
```

Abbildung 26: Syntax when/else

Die in Abbildung 26 gezeigte Syntax zeigt, dass hier beliebige Bedingungen verwendet werden können. Der letzte Zweig ohne Bedingung entspricht der Defaultzuweisung ’when others’.

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

entity DEKODER is
port
(
  X0,X1,X2,X3:   in  bit;
  EN:            in  bit;
  a,b,c,d,e,f,g: out bit
);
end;

architecture behaviour of DEKODER is
signal y: bit_vector(0 to 6);
signal sel: bit_vector(3 downto 0);

begin

  sel <= "1111" when (EN='0')          -- Enable hier am Eingang
              else (x3,x2,x1,x0);

  with sel select                      -- Umsetzung Dualzahl in
  y <= "1111110" when x"0",            -- Siebensegmentcode
       "0110000" when x"1",
       "1101101" when x"2",
       "1111001" when x"3",
       "0110011" when x"4",
       "1011011" when x"5",
       "1011111" when x"6",
       "1110000" when x"7",
       "1111111" when x"8",
       "1111011" when x"9",
       "0000000" when others;          -- hier bleibt die Anzeige dunkel

  a <= y(0); b <= y(1); c <= y(2); d <= y(3);
  e <= y(4); f <= y(5); g <= y(6);

end;
```

Abbildung 27: Beispiel when/else

In diesem Beispiel wird die Enable-Funktion dadurch realisiert, dass in einer when/else-Anweisung das interne Auswahlsignal ’sel’ entweder mit dem nicht vorkommenden Wert ’1111’“ belegt wird oder eben mit dem anzuzeigenden Wert. Die nachfolgende Codeumsetzung sorgt dafür, dass bei der Auswahl ’1111’ die Anzeige dunkel bleibt.

5.5 if/then

Diese Anweisung entspricht der when/else-Anweisung, ist jedoch nur in **sequentiellen** Umgebung zulässig. Mit dieser Anweisungen können beliebige Blöcke von sequentiellen Anweisungen in Abhängigkeit von beliebigen Bedingungen ausgeführt werden.
Die Syntax zeigt Abbildung 28.

```
if      <Bedingung_1>     then  <Sequentielle Anweisungen 1>;
elsif   <Bedingung_2>     then  <Sequentielle Anweisungen 2>;
elsif   <Bedingung_2>     then  <Sequentielle Anweisungen 3>;
else                            <Sequentielle Anweisungen n>;
end if;
```

Abbildung 28: Syntax if/then

Beachten Sie die unterschiedliche Schreibweise von 'elsif' und 'end if'. Die Bedingungen müssen vom Typ boolean sein. Das können entweder Signale/Variablen von diesem Typ sein oder aber Vergleiche zweier Ausdrücke gleichen Typs.
Da in jedem Zweig ein ganzer Block von Anweisungen stehen kann, können if/then-Anweisungen beliebig tief geschachtelt werden.

```
process (reset, clk)                    -- Prozess hängt von reset und clk ab
begin
  if (reset='0') then qint <= x"0";     -- Asynchrones Rücksetzen auf Null
  elsif (clk='1') and clk'event         -- warten auf Taktflanke
  then
    if (qint > x"8")                    -- Abfrage auf >8, da Signal noch den
    then                                -- vorherigen Zählerstand enthält
      qint <= x"0";
    else
      qint <= qint+1;                   -- Zählerstand hochzählen
    end if;
  end if;
end process;
```

Abbildung 29: Beispiel if/then

Der Code in Abbildung 29 ist ein Ausschnitt aus dem Code für den Zähler (Abbildung 23). Dabei wird eine geschachtelte if/then-Konstruktion verwendet, denn in dem then-Zweig der ersten elsif-Bedingung ist eine weitere if/then-Anweisung enthalten.
An diesem Beispiel wird auch deutlich, dass mit der if/then-Anweisung eine Priorisierung bei mehreren gleichzeitig wahren Bedingungen erreicht wird.
Angenommen, eine steigende Taktflanke trifft ein während das Signal 'reset' den Wert '0' hat. Dann wird dennoch nur der then-Zweig der ersten if/then-Anweisung ausgeführt, d.h. der Zähler bleibt zurückgesetzt. So soll es bei einem asynchronen Reset auch sein.

5.6 case/is

Diese Anweisung entspricht der with/select-Anweisung, ist jedoch nur in **sequentiellen** Umgebung zulässig. Mit dieser Anweisungen können ebenfalls beliebige Blöcke von sequentiellen Anweisungen in Abhängigkeit von **einem** Testsignal ausgewählt werden.

```
case    <testsignal> is
        when  <Wert_1>     => <Sequentielle Anweisungen 1>;
        when  <Wert_2>     => <Sequentielle Anweisungen 2>;
        when  <Wert_2>     => <Sequentielle Anweisungen 3>;
        when others        => <Sequentielle Anweisungen n>;
end case;
```

Abbildung 30: Syntax case/is

Das zu testende Signal und die Testwerte müssen natürlich vom gleichen Typ sein. Wie bei der with/select-Anweisung sollten Sie eine Defaultzuweisung in einem Zweig 'when others' vorsehen. Dies ist hier besonders wichtig, weil die case/is-Anweisung fast immer bei der Beschreibung einer Zustandsübergangsfunktion benutzt wird. Ist diese Funktion aber nicht vollständig definiert, dann besteht die Gefahr, dass der Automat in einen ungewollten Zustand übergeht und evtl. dort bleibt.
Da ganze Blöcke aufgrund einer Bedingung ausgeführt werden können, sind auch hier beliebig tiefe Schachtelungen möglich.
Ein ausführliches Beispiel zur case/is-Anweisung folgt in der Automatenbeschreibung.

```
architecture verhalten of MULTIPLEXER is
  signal a,b,y: bit_vector(0 to 3);   -- Hilfsvektoren

begin
  a <= (a0,a1,a2,a3);                 -- Kopieraktionen
  b <= (b0,b1,b2,b3);
  y0 <= y(0); y1 <= y(1); y2 <= y(2); y3 <= y(3);

  process (s,a,b)                     -- Empfindlichkeitsliste
  begin
    case s is
      when '0' => y <= a;
      when '1' => y <= b;
    end case;
  end process;

end;
```

Abbildung 31: Beispiel case/is

Im Code aus Abbildung 31 für den Multiplexer musste zunächst mit dem Prozess eine sequentielle Umgebung geschaffen werden, da case/is in nebenläufigen Umgebungen unzulässig ist. Als Besonderheit kann hier die Defaultzuweisung entfallen, weil das Testsignal vom Typ 'bit' ist und damit tatsächlich nur die Werte '0' und x'1' annehmen kann. Diese beiden Möglichkeiten werden auch explizit abgefragt, so dass keine weitere Möglichkeit mehr übrig bleibt.

6 Automatenbeschreibung

Automaten können in VHDL auf vielfältige Weise beschrieben werden. Die Art der Beschreibung hat teilweise Einfluss auf das Syntheseergebnis; manche Entwurfsprogramme können auch nicht alle möglichen Beschreibungsformen umsetzen. Aus diesem Grund wird hier nur eine einzige Möglichkeit zur Beschreibung eines Moore-Automaten vorgestellt. Die Beschreibung kann von allen bekannten Entwurfswerkzeugen umgesetzt werden und ist gleichzeitig übersichtlich. Die Erweiterung auf den Mealy-Automaten ist trivial.
Ein Moore-Automat enthält drei voneinander getrennte Blöcke:

- Zustandsspeicher
- Zustandsübergangsfunktion
- Ausgabefunktion

Im diesem „Schema F“-Entwurf werden diese drei Blöcke in einer nebenläufigen Umgebung als zwei parallel ausgeführte Prozesse modelliert. Der erste Prozess beschreibt lediglich den Zustandsspeicher, während der zweite Prozess sowohl die Zustandsübergangsfunktion als auch die Ausgabefunktion beschreibt.
Für die Zustandsübergangsfunktion und die Ausgabefunktion wären auch nebenläufige Anweisungen denkbar. Die Darstellung als Prozess hat aber den Vorteil, dass die case/is-Anweisung verwendet werden kann, die für jede Bedingung wieder einen ganzen Block mit Anweisungen ermöglicht.
Das Grundgerüst hat in diesem Schema die in Abbildung 32 gezeigte Struktur:

```
architecture verhalten of automat is
  signal z_alt, z_neu: zustaende;                  -- symbolische zustaende
begin

  process (clk, init)                              --  zustandsspeicher
  begin
    if (init) then zustand <= resetzustand;        -- hier die asynchrone initialisierung
    elsif (clk = '1') and clk'event                -- warten auf steigende taktflanke
    then
      z_alt <= z_neu;                              -- zustandsaktualisierung
    end if;
  end process;

  process (z_alt, eingaenge)                       -- ausgabe-/uebergangsfunktion
  begin
    case z_alt is
      when z0 => z_neu <= f_u(z0,eingaenge);       -- uebergaenge vom zustand z0
                 aus   <= f_a(z0, eingaenge);      -- ausgabe im zustand z_0
      when z1 => z_neu <= f_u(z1,eingaenge);       -- uebergaenge vom zustand z1
                 aus   <= f_a(z1, eingaenge);      -- ausgabe im zustand z_1
      ....
     end case;
  end process;

end architecture;
```

Abbildung 32: Schema für den Moore-Automat

6.1 Spezielle Konstrukte

Für eine gut lesbare und erfolgreich synthetisierbare Beschreibung nach diesem Schema benötigen Sie noch zwei bisher nicht vermittelte Kenntnisse.

6.1.1 Symbolische Zustandskodierung

Bei Ihrem Papierautomaten haben Sie sinnvollerweise symbolische Zustandsnamen wie ein, aus, vorwaerts und rueckwaerts gewählt. Dafür gibt es in VHDL keinen geeigneten Typ, obwohl es sicherlich günstig wäre, diese Namen zunächst beizubehalten. Mit der in Abbildung 33 gezeigten Anweisungsfolge können Sie sich aber in VHDL einen eigenen Aufzählungstyp definieren.

Syntax:
type <typname> **is** (wert1, wert2, ..., wert_n)**;**

Beispiel:
type gaenge **is** (leerlauf, vorwaerts, rueckwaerts);
signal z_alt, z_neu: gaenge;

Abbildung 33: Syntax Definition eines Aufzählungstyps

Sie haben damit zwei Signale ’z_alt’ und ’z_neu’ zur Verfügung, die jeweils die Werte ’leerlauf’, ’vorwaerts’ und ’rueckwaerts’ annehmen können. Sie können mit diesen Werten zwar keine Verknüpfungen bilden – Sie können aber die symbolischen Namen einem geeigneten Signal zuweisen und ein Signal mit einem Wert auf Gleichheit prüfen. Mehr benötigen Sie in der Automatenbeschreibung auch nicht!
Wenn Sie den fertig beschriebenen Automaten synthetisieren wollen, dann müssen Sie natürlich den symbolischen Namen Dualzahlen zuordnen. Wenn Sie nichts weiter tun, dann wird das Entwurfswerkzeug automatisch eine Zuordnung durchführen. Bei manchen Werkzeugen können Sie auch noch als Option auswählen, nach welchen Verfahren das geschehen soll. Übliche Möglichkeiten sind binär, one-hot und gray.
Sie können aber auch selbst eine Kodierung vornehmen, und zwar ohne dass Sie in der gesamten VHDL-Beschreibung die symbolischen Namen durch Dualzahlen ersetzen. Erweitern Sie die Definition aus Abbildung 33 um eine Zuordnung nach Abbildung 34.

Syntax:
type <typname> **is** (wert1, wert2, ..., wert_n)**;**
attribute enum_encoding: string**;**
attribute enum_encoding of <typname> **: type is “**zahl_fuer_wert1,zahl_fuer_wert2,…”;

Beispiel:
type gaenge **is** (leerlauf, vorwaerts, rueckwaerts);
attribute enum_encoding: string;
attribute enum_encoding of gaenge: type is “00 11 10”;
signal z_alt, z_neu: gaenge;

Abbildung 34: Beispiel Zustandskodierung

In diesem Beispiel haben Sie festgelegt, dass der Zustand ’leerlauf’ mit “00“ kodiert wird, der Zustand ’vorwaerts’ mit “11“ und der Zustand ’rueckwaerts’ mit ’10’.

6.1.2 Prozessstruktur

Die meisten Synthesewerkzeuge können eine sequentielle Schaltung nur dann fehlerfrei abbilden, wenn sie in einer „Normaldarstellung“ beschrieben sind. Alle bekannten Synthesewerkzeuge können Beschreibung nach Abbildung 35 erkennen und erfolgreich synthetisieren.

```
process (<empfindlichkeitsliste>)
-- hier lokale Signale und Variablen deklarieren
begin
  if (<asynchrone bedingung>)
    then
      <z_neu> <= initialzustand;
    elsif <takt> = ’wert’ and <takt>’event
      then
        -- hier beginnt die sequentielle umgebung
  end if;
end process;
```

Abbildung 35: Synthetisierbarer taktsynchroner Prozess

Beachten Sie zunächst, dass Sie alle notwendigen Signale auch wirklich in die Empfindlichkeitsliste des Prozesses eintragen.
Falls Ihr Automat mit einem asynchronen Signal in einen initialen Zustand gebracht werden kann (Beispiel: Reset), dann beginnen Sie den Prozess sofort mit einer entsprechenden if/then-Anweisung.
In den elsif-Bedingung dieser Anweisung (bzw. die if-Bedingung, wenn Sie keine asynchrone Initialisierung benötigen) schreiben Sie für einen Übergang bei

- **steigender** Taktflanke: (takt = ’1’) and takt’event
- **fallender** Taktflanke: (takt = ’0’) and takt’event

Versuchen Sie nicht, einen Automaten zu kreieren, der auf beide Taktflanken reagiert.
Versuchen Sie nicht, die Taktflankenabfrage an eine andere Stelle zu verschieben.
Versuchen Sie nicht, die asynchrone Bedingungsabfrage an eine andere Stelle zu verschieben.
Versuchen Sie nicht, mehrere Taktflankenabfragen einzubauen.

Sollten Sie in einer nicht von Ihnen verfassten Beschreibung eine ’wait’-Anweisung am Ende des Prozesses sehen, dann denken Sie sich statt dieser Anweisung die entsprechende Taktflankenabfrage an den Beginn des Prozesses (wie in Abbildung 35). Dies ist eine weitere übliche Methode, die jedoch (Stand 2002) an Bedeutung verliert.

6.2 Beispiel Frag-O-Mat

In diesem Kapitel wird mit den bisherigen Kenntnissen ein Moore-Automat in VHDL entworfen. Die bisherige Erfahrung hat gezeigt, dass sich professorales Frageverhalten (leider) einigermaßen zuverlässig mit einem Automaten namens „Frag-O-Mat“ nachbilden lässt. Es seien in der Vorlesung noch ein harter Kern von vier Hörern A,E,L und S übriggeblieben. Diese werden normalerweise der Reihe nach befragt. Kann die Frage beantwortet werden (Bedingung w=weiß es), dann kommt als nächstes der alphabetisch folgende Kandidat dran. Dies ist im Zustandsübergangsgraph (Abbildung 36) die rechte Schleife A -> E -> L -> S -> A.

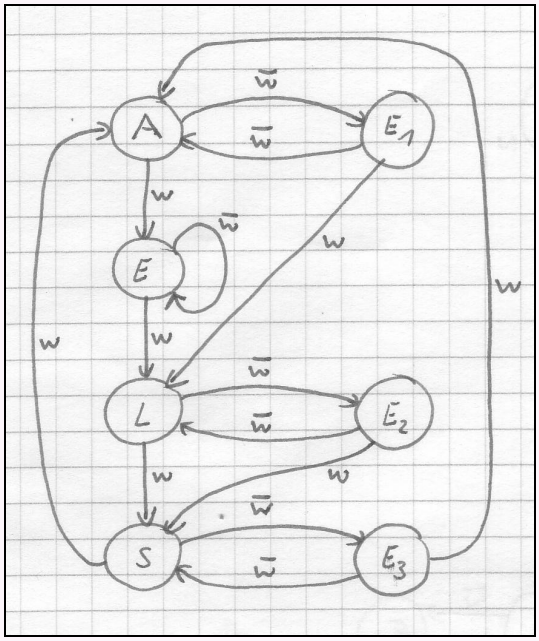


Abbildung 36: Zustandsübergänge „Frag-O-Mat“

Wenn die Frage nicht beantwortet werden kann (w_bar = weiß es nicht), dann kommt zunächst E an die Reihe, da hier die größte Wahrscheinlichkeit besteht, eine zufriedenstellende Antwort zu erhalten. Im Zustandsübergangsgraph sind dies die Übergänge nach rechts in die Zustände E1, E2 und E3 sowie der Sonderfall E -> E.
Weiß es E auch nicht, dann war die Frage zu schwer oder schlecht formuliert. In diesem Fall erhält der zuerst befragte Hörer eine weitere Chance mit einer besser angepassten Frage (Übergänge von E1, E2 und E3 nach links sowie der Sonderfall E -> E).
Konnte E dagegen die Frage beantworten, dann kommt als nächstes der ohnehin als „normaler“ Nachfolger vorgesehene Hörer an die Reihe (Übergänge von E1 und E2 nach links unten bzw. von E3 nach A).
Eine kurze Prüfung ergibt, dass der Graph sowohl vollständig als auch widerspruchsfrei ist.
Die Ausgabefunktion ist besonders einfach, es wird einfach der einem Zustand zugeordnete Hörer angezeigt (Abbildung 37)

Zustand	A	L	S	E	E1	E2	E3
Ausgabe	A	L	S	E			

Abbildung 37: Ausgabefunktion Frag-O-Mat

Der Automat ist, zumindest nach außen, sehr genügsam (Abbildung 38)

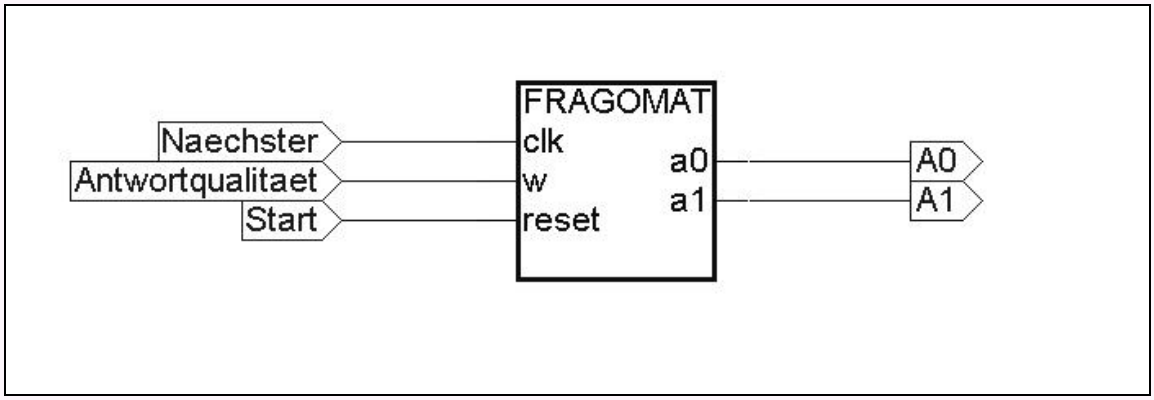


Abbildung 38: Blocksymbol des Frag-O-Mat

Die ersten beiden Abschnitte der dazupassenden VHDL-Beschreibung sind daher kaum mehr als eine Fingerübung (Abbildung 39).

```
 1
 2  library ieee;
 3  use ieee.std_logic_1164.all;
 4
 5
 6  entity fragomat is
 7  port
 8  (
 9    clk, w, reset: in std_logic;
10    a0, a1:        out std_logic
11  );
12  end;
13
14
```

Abbildung 39: Header und Entity des Frag-O-Mat

```
15  architecture verhalten of fragomat is
16   type hoerer is (Z_A,Z_E,Z_L,Z_S,Z_E1,Z_E2,Z_E3);
17   attribute enum_encoding: string;
18   attribute enum_encoding of hoerer: type is "000 100 001 010 101 110 111";
19   signal zalt, zneu: hoerer;
20   signal ausgabe: std_logic_vector(0 to 1);
21
22  begin
23
24   process (clk, reset)
25   begin
26     if (reset = '1')
27       then zalt <= Z_A;
28       elsif (clk = '1') and clk'event
29         then zalt <= zneu;
30    end if;
31    end process;
32
```

Abbildung 40: Deklarationen und Speicherprozess

In Zeile 15 beginnt die Funktionsbeschreibung. Zunächst wird ein eigener Aufzählungstyp ’hoerer’ definiert, damit die symbolischen Zustandsnamen weiterverwendet werden können. Außerdem wird - nur zu Demonstrationszwecken - eine Zustandskodierung vorgegeben.
Der erste Prozess für den Zustandsspeicher beginnt in Zeile 24. Bei einem Reset wird asynchron der Zustand Z_A eingestellt, ansonsten mit der steigenden Flanke am Takt der neue Zustand übernommen.
Die Beschreibung hält sich genau an das in Abbildung 35 angegebene Schema!

```
33
34  process (w, zalt)
35    variable x: boolean;
36  begin
37    if (w = '1')
38      then x := true;
39      else x := false;
40    end if;
41
42    case zalt is
43      when Z_A  => if x
44                     then zneu <= Z_E;
45                     else zneu <= Z_E1;
46                   end if;
47                   ausgabe <= "00";
48      when Z_E  => if x
49                     then zneu <= Z_L;
50                     else zneu <= Z_E;
51                   end if;
52                   ausgabe <= "11";
53      when Z_L  => if x
54                     then zneu <= Z_S;
55                     else zneu <= Z_E2;
56                   end if;
57                   ausgabe <= "01";
58      when Z_S  => if x
59                     then zneu <= Z_A;
60                     else zneu <= Z_E3;
61                   end if;
62                   ausgabe <= "10";
63      when Z_E1 => if x
64                     then zneu <= Z_L;
65                     else zneu <= Z_A;
66                   end if;
67                   ausgabe <= "11";
68      when Z_E2 => if x
69                     then zneu <= Z_S;
70                     else zneu <= Z_L;
71                   end if;
72                   ausgabe <= "11";
73      when Z_E3 => if x
74                     then zneu <= Z_A;
75                     else zneu <= Z_S;
76                   end if;
77                   ausgabe <= "11";
78    end case;
79  end process;
```

Abbildung 41: Zustandsübergangsfunktion und Ausgabefunktion

Die Beschreibung wird jetzt mit dem zweiten Prozess für die beiden Schaltnetze fortgesetzt. Beachten Sie zunächst, dass die Empfindlichkeitsliste dieses Prozesses ganz anders als die des ersten ist: Der Prozess hängt vom Zustand und den Eingangssignalen ab, nicht aber vom Takt oder dem Reset.
Nur zu Demonstrationszwecken wird ein lokales boolesches Signal ’x’ deklariert, das mit der if/then-Anweisung in den Zeilen 37 bis 40 mit einem später direkt abfragbaren Wert für die Übergangsbedingungen „w“ und „w_bar“ belegt wird.
In den Zeilen 42 bis 78 sind nun in einer einzigen case/is-Anweisung alle notwendigen Übergänge und Ausgaben beschrieben. Beachten Sie, dass zunächst mit der case/is-Anweisung die einzelnen Zustände unterschieden werden können. Die Zeilen 53 bis 57 gelten zum Beispiel, wenn sich der Automat gerade im Zustand Z_L befindet. Die if/then-Anweisung für diesen Fall (Zeilen 53 bis 56) berechnet die beiden hier möglichen Folgezustände. Dabei wird schlicht aus dem Papierentwurf abgeschrieben!
Ebenso einfach erfolgt in jedem Zustand die direkte Zuweisung an den Ausgabevektor in Zeile 57.
Die Beschreibung endet mit dem Umkopieren an die Ausgangssignale (Abbildung 42).

```
80
81
82   a0 <= ausgabe(0);
83   a1 <= ausgabe(1);
84  end;
85
```

Abbildung 42: Ausgabe des internen Ausgabesignals

9 .4 VHDL Übungslösung 'DecisDecoder'/Grado

Listing 12: 'Decis Prozessor VHDL'

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

entity DE0Board is
    port(
        sw            :    in  std_logic_vector(9 downto 0);
        led       :    out std_logic_vector(9 downto 0);
                  hex0                  :         out std_logic_vector(6 downto 0);
                  key                          :       in std_logic_vector(2 downto
                      0)
    );
end DE0Board;

architecture behavior of DE0Board is
    -- Eingˆ/nge --
    signal instruction     :    std_logic_vector(7 downto 0);
    signal sr_result        :    std_logic;
    -- Ausgˆ/nge --
    signal alu_en            :    std_logic;
    signal alu_sel            :    std_logic_vector(2 downto 0);
    signal rf_a            :    std_logic_vector(1 downto 0);
    signal rf_b            :    std_logic_vector(1 downto 0);
    signal rf_q            :    std_logic_vector(1 downto 0);
         signal rf_q2                          :         std_logic_vector(1 downto 0)
             ;
    signal rf_wr           :    std_logic;
    signal sr_cond            :    std_logic_vector(2 downto 0);
    signal pc_skip            :    std_logic;
    signal bus_imm            :    std_logic_vector(3 downto 0);
    signal bus_imm_hl        :    std_logic;
    signal pher_en            :    std_logic;
    signal pher_wr            :    std_logic;
         signal status                                  : std_logic_vector(3 downto 0)
             ;
         signal clear                                   : std_logic;
begin

    -- Instanz der entity InstructionDecoder
   InstrDec : entity work.InstructionDecoder port map(
                                    -- Eingˆ/nge --
                                    instruction => instruction,
                                    sr_result     => '0',
                                    -- Ausgˆ/nge --
                                    alu_en        => alu_en,
                                    alu_sel         => alu_sel,
                                    rf_a            => rf_a,
                                    rf_b            => rf_b,
                                    rf_q            => rf_q,
                                    rf_wr            => rf_wr,
                                    sr_cond        => sr_cond,
                                    pc_skip        => pc_skip,
                                    bus_imm        => bus_imm,
                                    bus_imm_hl    => bus_imm_hl,
                                    pher_en        => pher_en,
                                    pher_wr        => pher_wr
                                );

        Alu : entity work.alu port map(

                        -- Eingˆ/nge --
                        rf_a => rf_a,
                        rf_b => rf_b,
                        instruction => instruction(6 downto 4),
```

```
                        -- Ausgˆ/nge --
                        rf_q => rf_q2
        );

        StatusRegister: entity work.status_register port map(

                        -- Eingˆ/nge --
                        rf_a => rf_a,
                        rf_b => rf_b,
                        instruction => instruction(6 downto 4),
                        clear => clear,
                        -- Ausgˆ/nge --
                        status => status
        );

    -- Zuweisung der Schalter & LEDs --
    -- Eingˆ/nge
    instruction          <= sw(7 downto 0);
    sr_result            <= sw(8);
         clear                                  <= key(2);

process(sw) begin
    -- Ausgˆ/nge
         case status is
                        when "0000" => hex0 <= "1111111"; -- Clear
                        when "0001" => hex0 <= "1111001"; -- í Û 1 í Û  -- carry
                        when "0010" => hex0 <= "0100100"; -- í Û 2 í Û  -- equal
                        when "0100" => hex0 <= "0110000"; -- í Û 3 í Û  -- greater

                        when "1000" => hex0 <= "0011001"; -- í Û 4 í Û  -- less
                        when others => hex0 <= "1111111";
         end case ;

         if sw(9) = '0' then

                if(alu_en = '1') then
                        led(1 downto 0) <= rf_q2;
                else
                        led(1 downto 0) <= rf_q;
                end if;

                led(3 downto 2) <= rf_b;
                led(4) <= rf_wr;
                led(5) <= alu_en;
                led(8 downto 6) <= alu_sel;
                led(9) <= pc_skip;

         elsif(sw(9) = '1') then

                led(9 downto 6) <= bus_imm;
                led(5) <= bus_imm_hl;
                led(4) <= pher_en;
                led(3) <= pher_wr;
                led(2 downto 0) <= sr_cond;

         else
                led(9 downto 0) <= "0000000000";
         end if;
end process;
end behavior;
```

Listing 13: 'Decis Control Unit VHDL'

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

entity InstructionDecoder is
    port(
            instruction    :        in  std_logic_vector(7 downto 0);

            alu_en         :       out std_logic;
            alu_sel        :        out std_logic_vector(2 downto 0);

            rf_a           :       out std_logic_vector(1 downto 0);
            rf_b           :       out std_logic_vector(1 downto 0);
            rf_q           :       out std_logic_vector(1 downto 0);
            rf_wr          :       out std_logic;

            sr_cond        :        out    std_logic_vector(2 downto 0);
            sr_result      :       in  std_logic;
            pc_skip        :        out    std_logic;

            bus_imm        :        out    std_logic_vector(3 downto 0);
            bus_imm_hl     :       out    std_logic;

            pher_en        :        out    std_logic;
            pher_wr        :        out    std_logic
    );
end InstructionDecoder;

architecture behavior of InstructionDecoder is
    -- Buffer fÂ´...r die rf_q <= rf_b zuweisung.
    signal rf_q_t : std_logic_vector(1 downto 0);
begin
    decode : process  (instruction, sr_result)
    begin
        if(instruction(7) = '0') then    -- ALU --
            alu_sel     <= instruction(6 downto 4);
            alu_en      <= '1';
            rf_q_t      <= instruction(1 downto 0);
            rf_b        <= instruction(3 downto 2);
            rf_wr       <= '1';
            -- not used
            sr_cond        <= "000";
            pc_skip        <= '0';
            pher_en        <= '0';
            pher_wr     <= '0';
            bus_imm     <= "0000";
            bus_imm_hl    <= '0';

        elsif(instruction(7) = '1' and instruction(6) = '0') then
            alu_en      <= '0';
            rf_q_t      <= instruction(1 downto 0);
            rf_b        <= instruction(3 downto 2);
            sr_cond        <= "000";
            pc_skip        <= '0';
            pher_en        <= '1';
            bus_imm     <= "0000";
            bus_imm_hl    <= '0';
                                rf_wr                   <= instruction(4);
                                pher_wr                         <= instruction(4);

                        elsif(instruction(7) = '1' and instruction(6) = '1' and
                            instruction(5) = '0') then
            alu_en      <= '0';
            rf_q_t      <= "00";
            rf_b        <= "00";
            rf_wr       <= '1';
            sr_cond        <= "000";
            pc_skip        <= '0';
            pher_en        <= '0';
```

```
            pher_wr     <= '0';
            bus_imm     <= instruction(3 downto 0);
                               bus_imm_hl      <= instruction(4);

                           elsif(instruction(7) = '1' and instruction(6) = '1' and
                               instruction(5) = '1' and instruction(4) = '0' and
                               instruction(3) = '0') then
                                   alu_en      <= '0';
            rf_q_t      <= "00";
            rf_b        <= "00";
            rf_wr       <= '1';
            sr_cond        <= instruction(2 downto 0);
            pher_en        <= '0';
            pher_wr     <= '0';
            bus_imm     <= "0000";
                               bus_imm_hl      <= '0';
                               pc_skip <= sr_result xor instruction(3);

                  end if;

        rf_q         <= rf_q_t;
        rf_a         <= rf_q_t;

    end process;

end behavior;
```

Listing 14: 'Decis Alu VHDL'

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

entity alu is
        port(
                rf_a, rf_b: in std_logic_vector(1 downto 0);
                instruction: in std_logic_vector(2 downto 0);
                rf_q: out std_logic_vector(1 downto 0)
        );
end alu;

architecture beh_alu of alu is
begin
        with instruction select
                rf_q <=         rf_a                                    when "000",
                                not rf_a                                when "001",
                                rf_a and rf_b           when "010",
                                rf_a or rf_b            when "011",
                                rf_a xor rf_b           when "100",
                                rf_a + rf_b             when "101",
                                rf_a + "01"                             when "110",
                                "ZZ"                                            when
                                    others;

end beh_alu;
```

Listing 15: 'Decis Registerfile VHDL'

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

entity status_register is
port(
        instruction: in std_logic_vector(2 downto 0);
        rf_a, rf_b: in std_logic_vector(1 downto 0);
        clear:  in std_logic;
        status: out std_logic_vector(3 downto 0)
        );
end status_register;

architecture beh_status_register of status_register is
signal var1: std_logic;
begin

        var1 <= ((rf_a(1) xor rf_b(1)) and (rf_a(0) and rf_b(0))) or (rf_a(1) and rf_b
            (1));

        process(instruction, rf_a, rf_b, clear)
        begin
                if(clear = '0') then
                        status <= "0000";
                elsif(clear = '1') then

                        if(instruction = "111") then

                                if(rf_a = rf_b) then
                                        status <= "0010";
                                elsif(rf_a > rf_b) then
                                        status <= "0100";
                                elsif(rf_a < rf_b) then
                                        status <= "1000";
                                end if;

                        elsif(instruction = "101") then

                                if( var1 = '1') then
                                        status <= "0001";
                                end if;

                        end if;
                end if;
        end process;
end beh_status_register;
```

9 .6 VHDL Übungsabgabemurks

die folgenden Übungsabgaben sind 'Fehlersuchrätsel'
und deshalb mit einer **WARN-HINTERGRUNDFARBE** gekennzeichnet

9 .8 Programmierung width

9 .9 VHDL-Code

```
  ----------------------------------------------------------------------
  -- Kommentarschreim tui grundsaetzli nit, soa Bledsinn.
  -- Undscho gornit, wennma so insistierend drauf hingwiesen wird,
  -- undscho dreimal nid fyr den ollen XH, den Gschaftlhuaber.
  -- geahtn Shas un, va wem des is und vu wann;
  -- wers nit sieht, was des is und wiamas braucht
  -- is eh die volle Pfeifn
  -- und tuat gscheiter was unders, zB TV schaun oder notenausrechna
  ----------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

signal SWn     : BIT_VECTOR (3 downto 0);   -- 4 Input Lines
signal DISPLAY : BIT_VECTOR (6 downto 0);   -- die 7 LED Segmente

architecture main of VHDL_7Segment is
begin
        SWn <= not SW;          -- 'SW' is ... aehm ... brauxt ned wissn
        LED <= SWn;             -- 'LED' is, aaah, ''Low Education Decade''

        with (SWn) select DISPLAY <=
                "0000001" when "0000",
                "1001111" when "0001",
                "0010010" when "0010",
                "0000110" when "0011",
                "1001100" when "0100",
                "0100100" when "0101",
                "0100000" when "0110",
                "0001111" when "0111",
                "0000000" when "1000",
                "0000100" when "1001",
                "0001000" when "1010",
                "1100000" when "1011",
                "1110010" when "1100",
                "1000010" when "1101",
                "0110000" when "1110",
                "0111000" when "1111";

        Segment_a <= DISPLAY(6);
        Segment_b <= DISPLAY(5);
        Segment_c <= DISPLAY(4);
        Segment_d <= DISPLAY(3);
        Segment_e <= DISPLAY(2);
        Segment_f <= DISPLAY(1);
        Segment_g <= DISPLAY(0);
end architecture
```

9 .10 Pinbelegung

```
FPGA Pin & Beschreibung:
```

```
-------------------------

  PIN_E11:  Digit 'a'         PIN_J6:  Switch 0
  PIN_F11:  Digit 'b'         PIN_H5:  Switch 1
  PIN_H12:  Digit 'c'         PIN_H6:  Switch 2
  PIN_H13:  Digit 'd'         PIN_G4:  Switch 3
  PIN_G12:  Digit 'e'
  PIN_F12:  Digit 'f'
  PIN_F13:  Digit 'g'
```

9 .11 VHDL CODE woldi

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY SevenSEG IS
port (
  bcd      : in std_logic_vector (3 downto 0);  --BCD in
  segment7 : out std_logic_vector (6 downto 0)  --Decoded out
);
END SevenSEG;


ARCHITECTURE Behavioral OF SevenSEG IS BEGIN
  process (bcd) BEGIN
    case bcd is
      when "0000" => segment7 <= "0000001"; -- '0'
      when "0001" => segment7 <= "1001111"; -- '1'
      when "0010" => segment7 <= "0010010"; -- '2'
      when "0011" => segment7 <= "0000110"; -- '3'
      when "0100" => segment7 <= "1001100"; -- '4'
      when "0101" => segment7 <= "0100100"; -- '5'
      when "0110" => segment7 <= "0100000"; -- '6'
      when "0111" => segment7 <= "0001111"; -- '7'
      when "1000" => segment7 <= "0000000"; -- '8'
      when "1001" => segment7 <= "0000100"; -- '9'
      when "1010" => segment7 <= "0001000"; -- 'A'
      when "1011" => segment7 <= "1100000"; -- 'B'
      when "1100" => segment7 <= "0110001"; -- 'C'
      when "1101" => segment7 <= "1000010"; -- 'D'
      when "1110" => segment7 <= "0110000"; -- 'E'
      when "1111" => segment7 <= "0111000"; -- 'F'
    end case;
  end process;
END Behavioral;
```

9 .12 VHDL CODE (7 segment decoder) mikno

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SegmentAnzeige IS
PORT (
  x0,x1,x2,x3          : IN STD_LOGIC ;
  d0,d1,d2,d3,d4,d5,d6 : OUT STD_LOGIC );
END SegmentAnzeige ;

ARCHITECTURE LogicFunction OF SegmentAnzeige IS BEGIN
  d0 <= NOT (( NOT x2 AND NOT x0 )  OR (NOT x3 AND x1 )
         OR (NOT x3 AND x2 AND x0 ) OR ( x2 AND x1 )
         OR (x3 AND NOT x2 AND NOT x1 ) OR (x3 AND NOT x0 ));
  d1 <= NOT ((NOT x3 AND NOT x2) OR (NOT x3 AND NOT x1 AND NOT x0 )
         OR (NOT x2 AND NOT x0)  OR (NOT x3 AND x1 AND x0 )
         OR (x3 AND NOT x1 AND x0));
  d2 <= NOT ((NOT x3 AND NOT x1) OR (NOT x3 AND x0) OR (NOT x1 AND x0)
         OR (NOT x3 AND x2 ) OR ( x3 AND NOT x2 ));
  d3 <= NOT ((NOT x3 AND NOT x2 AND NOT x0) OR (NOT x2 AND x1 AND x0)
         OR (x2 AND NOT x1 AND x0) OR (x2 AND x1 AND NOT x0)
         OR (x3 AND NOT x1));
  d4 <= NOT ((NOT x2 AND NOT x0) OR (x1 AND NOT x0) OR (x3 AND x1 )
         OR (x3 AND x2 ));
  d5 <= NOT ((NOT x1 AND NOT x0 ) OR (NOT x3 AND x2 AND NOT x1)
         OR (x2 AND NOT x0)  OR (x3 AND NOT x2) OR (x3 AND x1));
  d6 <= NOT ((NOT x2 AND x1) OR (x1 AND NOT x0)
         OR (NOT x3 AND x2 AND NOT x1)
         OR (x3 AND NOT x2) OR (x3 AND x0 ));
END LogicFunction;
)
```

9 .13 Generated VHDL Code from Altium(to simulate)

```
-------------------------------------------------------------
-- VHDL Testbench for fpga_project
-- 2016 2 16 8 47 7
-- Created by "EditVHDL"
-- "Copyright(c) 2002 AltiumLimited"
-------------------------------------------------------------
Library IEEE ;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_textio.all;
Use STD.textio.all;
-------------------------------------------------------------

entity Testfpga_project is
end Testfpga_project ;
-------------------------------------------------------------

architecture stimulus of Testfpga_project is
file RESULTS : TEXT open WRITE_MODE is "results.txt";
procedure WRITE_RESULTS(
  INPUT1 : std_logic;
  INPUT2 : std_logic;
  OUTPUT : std_logic
) is
variable l_out : line;
begin
  write(l_out, now, right, 15) ;
  write(l_out, INPUT1, right, 2) ;
  write(l_out, INPUT2, right, 2) ;
```

```
  write(l_out, OUTPUT, right, 2) ;
  writeline(RESULTS, l_out);
end procedure ;
component fpga_project
port(
  INPUT1 : in std_logic;
  INPUT2 : in std_logic;
  OUTPUT : out std_logic
);
end component;

  signal INPUT1 :
  signal INPUT2 :
  signal OUTPUT :std_logic

begin
  DUT: fpga_project port map (
    INPUT1 => INPUT1 ,
    INPUT2 => INPUT2 ,
    OUTPUT => OUTPUT
  );

  STIMULUS0 : process
  begin
    -- insert stimulus here
    INPUT1 <= '0' ;
    INPUT2 <= '0' ;
    wait for 20ns ;

    INPUT1 <= '1' ;
    INPUT2 <= '0' ;
    wait for 20ns ;

    INPUT1 <= '0' ;
    INPUT2 <= '1' ;
    wait for 20ns ;

    INPUT1 <= '1' ;
    INPUT2 <= '1' ;
    wait for 20ns ;
    wait ;
  end process ;

  WRITE_RESULTS(INPUT1, INPUT2, OUTPUT);
end architecture;
------------------------------------------------------------
```

9 .14 Der VHDL-Code in Quartus wurde dadurch erstellt - nirra

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SSegmentAnzeige IS
  PORT (
    s0,s1,s2,s3          : IN STD_LOGIC;
    d0,d1,d2,d3,d4,d5,d6 : OUT STD_LOGIC
  );
END SSEgmentAnzeige;

ARCHITECTURE LogicFunction OF SSEgmentAnzeige IS BEGIN
  d0 <= NOT((NOT s2 AND NOT s0) OR (NOT s3 AND s1 )
         OR (NOT s3 AND s2 AND s0) OR (s2 AND s1)
         OR (s3 AND NOT s2 AND NOT s1 ) OR (s3 AND NOT s0));
  d1 <= NOT((NOT s3 AND NOT s2)
         OR (NOT s3 AND NOT s1 AND NOT s0) OR (NOT s2 AND NOT s0)
         OR (NOT s3 AND s1 AND s0) OR (s3 AND NOT s1 AND s0));
  d2 <= NOT((NOT s3 AND NOT s1) OR (NOT s3 AND s0) OR (NOT s1 AND s0)
         OR (NOT s3 AND s2) OR (s3 AND NOT s2));
  d3 <= NOT((NOT s3 AND NOT s2 AND NOT s0) OR (NOT s2 AND s1 AND s0)
         OR (s2 AND NOT s1 AND s0) OR (s2 AND s1 AND NOT s0)
         OR (s3 AND NOT s1 AND NOT s0));
  d4 <= NOT((NOT s2 AND NOT s0) OR (s1 AND NOT s0) OR (s3 AND s1)
         OR (s3 AND s2));
  d5 <= NOT((NOT s1 AND NOT s0) OR (NOT s3 AND s2 AND NOT s1)
         OR (s2 AND NOT s0) OR (s3 AND NOT s2) OR (s3 AND s1));
  d6 <= NOT((NOT s2 AND s1) OR (s1 AND NOT s0)
         OR (NOT s3 AND s2 AND NOT s1)
         OR (s3 AND NOT s2) OR (s3 AND s0));
END LogicFunction;
```

9 .15 Goliaths VHDL.Scripten

1 Synthese von Schaltnetzen

1.1 Entity – Architecture

Entity und Architecture sind die **grundlegenden Strukturen** einer VHDL – Beschreibung.

- Die ***Entity*** beschreibt die **Schnittstellen** eines VHDL – Funktionsblocks (Designs) nach außen, die Deklaration der Anschlüsse erfolgt mittels der ***port – Anweisung***. Die Kommunikation einer Entity nach **außen** läuft über ***port – Signale***. Jeder Entity muss zumindest einer Architecture zugeordnet sein.
- Die ***Architecture*** beschreibt die **Funktionalität** des Designs in Form eines **VHDL – Code**s.

1.1.1 Kommunikation nach Außen

Port-Signale verbinden **Design-Entitie**s („ICs auf einer Platine“) miteinander.

Sie werden innerhalb einer Entity mit dem Schlüsselwort **port** deklariert.

Jedes Port-Signale besitzt einen **Richtung**smodus und einen Daten**typ** (std_logic, bit..).

Port-Signale können zu **Gruppen** zusammengefasst werden (std_logic_vector).

1.1.1.1 Richtungsmodi

- in: Eingangssignal (nur **rechte Seite** einer Zuweisung/Abfrage)
- out: Ausgangssignal (nur **linke Seite** einer Zuweisung)
- inout: Bidirektionales Signal (erfordert Datentyp std_logic)
- buffer: Ausgangssignale, die in die Schaltung zurückgeführt werden (**linke und rechte Seite** einer Zuweisung/Abfrage). **!!!**Eher nicht verwenden und mittels interner Signale nachbilden**!!!**

1.1.2 Vorlage

```
------------------------------------------------------------------------------------------------------------------------
---                                    Bibliotheken                                                                  ---
------------------------------------------------------------------------------------------------------------------------
library IEEE;                                       --Standard-LIB-Package ieee
      use   IEEE.std_logic_1164.all;                --Einbindung der Bibliotheken
--Ein Package enthält Komponenten, Typen, Funktionen, Unterprogramme die durch den use –
--Deklaration eine Design-Entity zur Benutzung zur Verfügung gestellt werden.

------------------------------------------------------------------------------------------------------------------------
---                                    Entity                                                                        ---
------------------------------------------------------------------------------------------------------------------------
--Die Eintity ist das erste der beiden Grundelemente eines VHDL-Designs. Sie stellt eine
--"BLACK BOX" zur Definition der SCHNITTSTELLEN nach außen (EIN.- und AUSGÄNGE) dar.

entity template is

      --generic(); zur Deklaration von Übergabeparametern

      --die "port"-Anweisung erzeugt die "Black-Box"
      port(
            tI1,tI2     :     in    std_logic;    --die "Anschlüsse" weisen je ein Richtung
            tO1, tO2    :     out   std_logic     --(in,out,..) und einen Datentyp
 );                                               --(sdt_logic....)auf
```

2 Entwurf digitaler Funktionselemente mit Prozessen

Das, bisher behandelte Syntaxelement **„nebenläufige Anweisung“** (concurrent statement) bieten nur eingeschränkte Modellierungsmöglichkeiten.

Mit der Einführung des Elements „**process**“ werden diese wesentlich erweitert da aufgrund der darin enthaltenen **hintereinander** ablaufenden Anweisungen (sequential statement) komplexere Sprachelemente (**Verzweigung**en und **Schleife**n) und die Modellierung **zeit**lichen Verhaltens möglich werden.

2.1 Prozesse

Prozesse treten **innerhalb** einer **architecture** auf und laufen **nebenläufig** (untereinander und in Bezug zu nebenläufigen Anweisungen innerhalb er architecure) ab.

Prozesse verwenden **eigene Anweisungen** (sequentielle Anweisungen, sie dürfen nur innerhalb von Prozessen verwendet werden, Ausnahme „unbedingte Signalzuweisung“) die **nacheinander abgearbeitet** werden.

Des weiteren sind **unbedingte Signalzuweisungen** erlaubt wobei zu berücksichtigen ist, dass die tatsächliche **Aktualisierung** immer am **Ende** des Prozesses erfolgt (einem Signal können somit innerhalb eines Prozesses mehrere Werte zugewiesen werden, übernommen wird nur der letzte).

Prozesse eignen sich besonders zur Beschreibung **getakter Funktionselemente** (und damit zur Definition zeitlicher Signalabfolgen), es können mit Prozessen aber sowohl **kombinatorische** (der Prozess kann in diesem Fall durch „nebenläufige Anweisungen“ ersetzt werden) als auch **sequentielle** Schaltungen modelliert werden.

2.1.1 Variable

Muss innerhalb eines Prozesses auf einen gerade aktualisierten Wert zugegriffen werden, muss diese Zuweisung an eine **variable** (anstelle eines signal) ausgeführt werden.

Diese wird **innerhalb** des Prozesses deklariert und besitzt nur dort Gültigkeit.

Soll ihr Wert **in anderen Prozessen** verwendet werden, muss er auf ein **signal** kopiert werden.

2.1.1.1 Beispiel

fulladder3

2.1.2 Deklaration und Ausführung

Die Deklaration erfolgt **innerhalb** einer **architecture**, je nach Aktivierung und Ausführung werden unterschieden:

2.1.2.1 Prozess mit Empfindlichkeitsliste

Der Prozess wird **aktiviert**, wenn sich (zumindest) ein Signal in der **Empfindlchkeitsliste** ändert.

Die Prozessanweisungen werden **nacheinander** durchlaufen, **unbedingte Signalzuweisungen werden erst am Prozessende aktualisiert**.

Der Prozess stoppt danach und wartet auf eine neue Änderung von Signalen in der Sensitivity-list.

2.1.2.1.1. Vorlage

archictecture xy_a of xy is

2.3 Standardbausteine

Im Gegensatz zu Schaltnetzen hängt der **aktuelle Zustand** von Schaltwerken nicht nur vom aktuellen **Eingangszustand** sondern auch von, in der **Vergangenheit** durchlaufenen Zuständen ab.

Allgemeine Schaltwerke sind der maßgebliche Teil jeder digitalen Schaltung, ihre Grundstruktur beruht auf den Modellen nach **Mealy** bzw. **Moore**.

Daneben gibt es **grundlegende** Schaltwerk**strukturen** (Standardbausteine), aus denen sich komplexere Schaltwerke zusammensetzen lassen. Zu diesen gehören **FlipFlop**s, **Zähler** und **Schieberegister**.

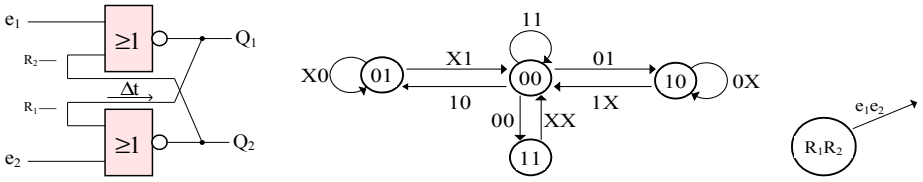
2.3.1 FlipFlops und Register

FlipFlops und Register sind die **Grundbauelemente** mit denen der Großteil der Schaltwerke aufgebaut wird

2.3.1.1 BasisFlipFlop

Ein Basis-Flipflop muss folgende Forderungen erfüllen:

- Es muss zwei verschiedene **innere Zustände** annehmen können, von denen der eine am Ausgang Q die logische 0 und der andere die logische 1 erzeugt.
- Es muss mindestens **zwei Eingänge** e_1 und e_2 aufweisen. Bei einer Eingangskombination $f_1(e_1,e_2)$ müssen beide inneren Zustände der Schaltung stabil sein (**bistabil**es Verhalten), so dass der Ausgang Q entweder 0 oder 1 sein kann, bei einer zweiten $f_2(e_1,e_2)$ muss der Ausgang der Schaltur auf binär **1 gesetzt** werden, bei einer dritten $f_3(e_1,e_2)$ auf binär **0**.

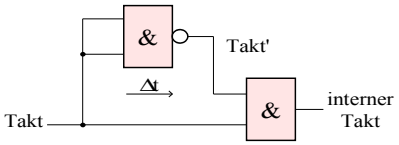


Die Basis-Flipflops sind für viele Aufgaben noch **ungeeignet** (z.B.: 0 oder 1 können nicht über einen Eingar eingschrieben werden (Speicherfunktion), eine Impulsfolge an einem Eingang kann keinen steten Wechsel des Ausgangszustands erzeugen (Togglefunktion)).

Um dies zu ermöglichen, werden Basis-Flipflops durch eine **Ansteuerschaltung** vor den Eingängen erweitert, die 0-setzende und die 1-setzende Wirkung der Eingänge wird durch ein zusätzliches **Taktsignal** ausgelöst (**taktgesteuert**e oder **synchron**e FlipFlops).

Taktgesteuerte Flipflops werden so aufgebaut, dass sich das Flipflop bei **fehlendem Takt** im **bistabil**en Zustand befindet.

2.3.1.2 Taktgesteuerte FlipFlops



5 Arithmetik.- und Vergleichsoperatoren

Die Realisierung **komplexer Komponente**n (Universalzähler, Universalschieberegister...) wird durch Verwendung von

- **Arithmetik**.- und **Vergleich**soperatoren und den zugehörigen
- **numerisch**en Datentypen: signed, unsigned, (integer)

wesentlich vereinfacht.

Für Synthese-Anwendungen bietet sich dazu vor allem die Anwendung

- **synthese**-fähiger **arithmetisch**er Operatoren: **+**, **-**, *, **abs**[Betrag] und **[2er-Potenz]
- **Vergleich**soperatoren: **=**, **/=** (ungleich), **<**, **<=**, **>**, **>=** (siehe auch „bedingte Signalzuweisungen“)

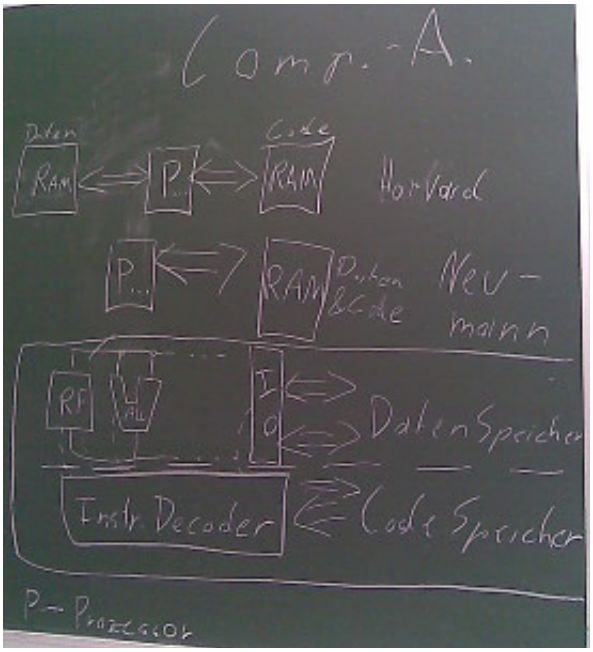
auf Signale und Variable des schon bekannten Datentyp **std_logic_vector** durch Einbindung zusätzlicher **Bibliothek**en an.

Die **Bits** des Typs std_logic können damit, nach einem **Typ-Cast** für **arithmetisch** Operatoren als signed/unsigned (**Zahlen**werte) interpretiert werden, umgekehrt gelten signed/unsigend für **logisch**e Operatoren als std_logic.

```
library IEEE;
        use    IEEE.std_logic_1164.all;
        use    IEEE.numeric_std.all;              --Einbindung der IEEE – Numeric – Bibliothek

entity template is
        …;
        port(
                …;
                x               :       in              std_logic_vector(3 downto 0);
                ….;
        );
end template;

architecture template_a of template is
        …;
        signal          xInt    :       unsigned(3 downto 0);          --Datentyp unsigned
        …;
        begin
                …;
                CNTPRO:process(..)
                        begin
                                …;
                                xInt <= unsigned(x);       --Typen-Cast
                                …;
                                --Operator + nur auf "numeric" anwendbar , !!! 1 anstellen von '1'
                                        xInt <= xInt + 1;
                                …;
                end process CNTPRO;
                …;
                x <= std_logic_vector(xInt);     --Typen-Cast
                …;
end template_a;
```

9 .16 FPGAprojekt ProzessorDesign

9 .17 Prozessor-Architekturen



Harvard Architektur:
* *instruction word'* unabhängig von Datenbusbreite,
* schneller — kein Datenbus-Engpass

von-Neumann Architektur:
* Weniger externe Busleitungen

(Referat dB 14Sep16)

9 .18 Theorie-Seminar vom dB

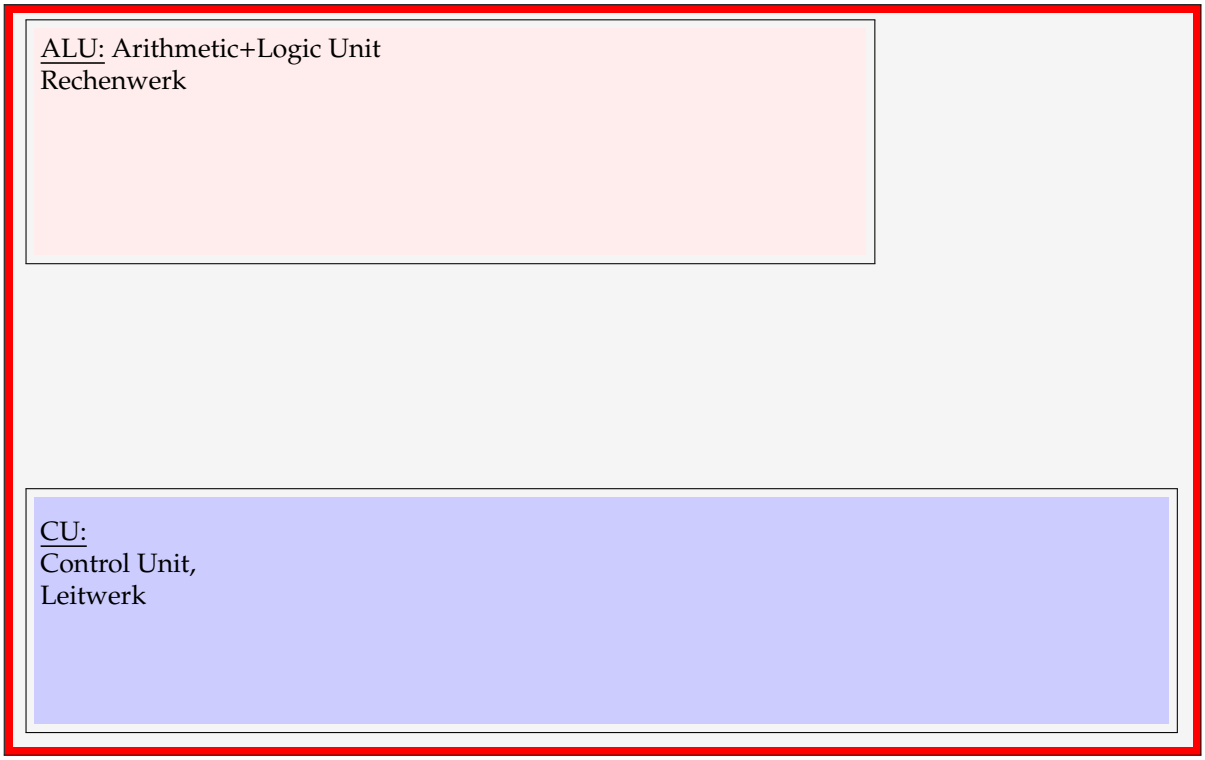
→ obelix: "ProzessorDesign-Deci.pdf"

→ lokal: "ProzessorDesign-Deci.pdf"

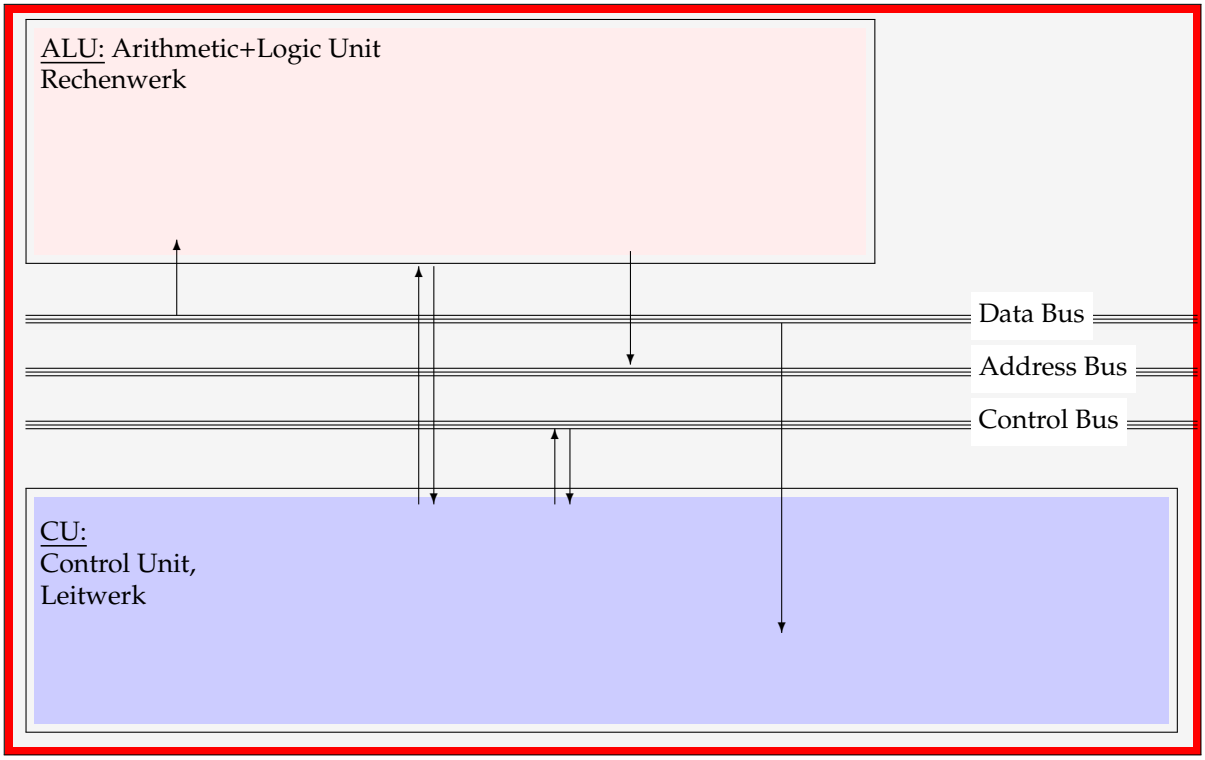
Vielen Dank für Ihre Aufmerksamkeit

9 .19 Prozessor Blockdiagramm

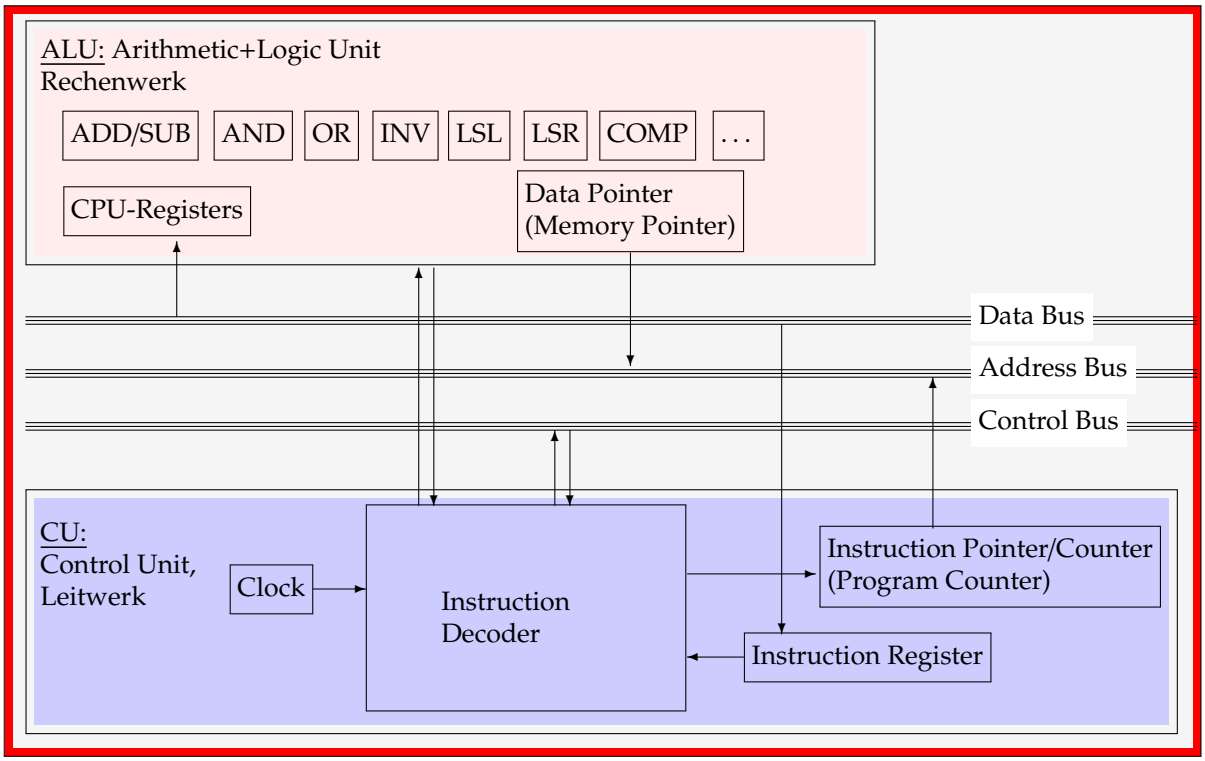
Hauptbestandteile: Rechenwerk (ALU), Leitwerk (CU)



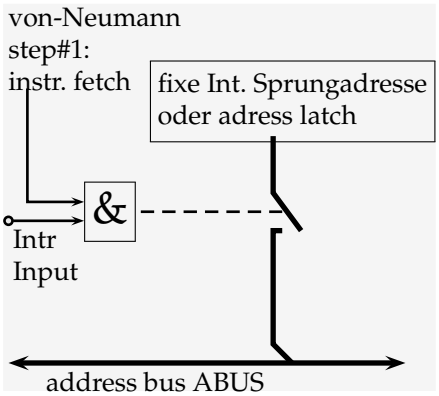
ALU und CU sind untereinander und nach "draussen" verbunden



CU steuert, ALU arbeitet

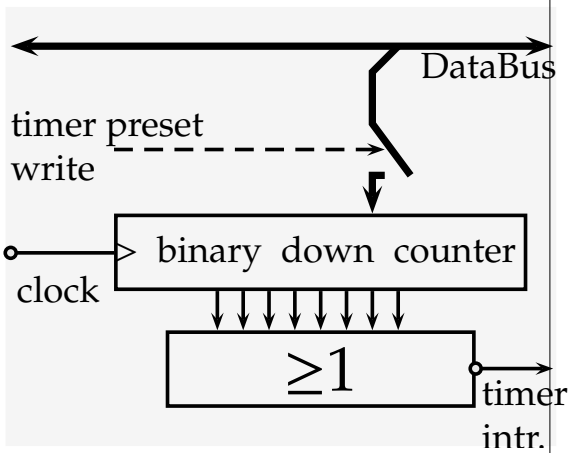


9 .20 Du kannsch Prozessoren entwerfen!

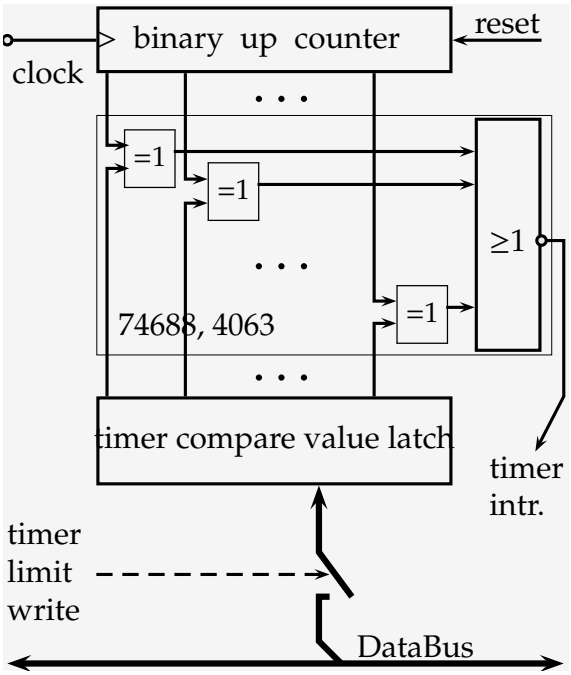


9 .21 Erweiterungen

9 .22 Interrupt am Prozessor



9 .23 Timer, down counting



9 .24 Timer, up counting

9 .24.1 Prozessor Instruction Decoder (VHDL, dB)

Listing 16: Decis InstructionDecoder (VHDL)

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

entity IDec_Lab is
  port(
        instruction:    in      std_logic_vector(7 downto 0);
        alu_sel:        out     std_logic_vector(2 downto 0);
        rf_a:           out     std_logic_vector(1 downto 0);
        rf_b:           out     std_logic_vector(1 downto 0);
        rf_cond:        out     std_logic_vector(1 downto 0);
        rf_result:      in      std_logic;
        rf_pc:          in      std_logic_vector(7 downto 0);
        rf_skip:        out     std_logic;
        rf_wr:          out     std_logic;
        rom_addr:       out     std_logic_vector(7 downto 0);
        bus_imm:        out     std_logic_vector(3 downto 0);
        bus_imm_hl:     out     std_logic;
        ram_en:         out     std_logic;
        ram_wr:         out     std_logic
  );
end IDec_Lab;

architecture behavior of IDec_Lab is
begin
    decode : process  (instruction, rf_result, rf_pc)
    begin
        if(instruction(7) = '1') then   -- ALU --
                alu_sel <= instruction(6 downto 4);
                rf_a    <= instruction(1 downto 0);
                rf_b    <= instruction(3 downto 2);
                rf_wr   <= '1';
                -- not used
                rf_cond <= "00";
                rf_skip <= '0';
                ram_en  <= '0';
                ram_wr  <= '0';
                bus_imm <= "0000";
                bus_imm_hl <= '0';
        else
                alu_sel <= "000";
                if(instruction(6) = '1') then -- IN & OUT --
                        ram_en  <= '1';
                        ram_wr  <= instruction(4);
                        rf_wr   <= not instruction(4);
                        -- not used
                        rf_a    <= instruction(1 downto 0);
                        rf_b    <= instruction(3 downto 2);
                        rf_cond <= "00";
                        rf_skip <= '0';
                        bus_imm <= "0000";
                        bus_imm_hl <= '0';
                else    if(instruction(5) = '1') then -- LDH & LDL --
                          bus_imm <= instruction(3 downto 0);
                          bus_imm_hl <= instruction(4);
                          rf_a    <= "00"; -- Dest.
                          rf_wr   <= '1';
                          -- not used
                          rf_b    <= "00";
                          rf_cond <= "00";
                          rf_skip <= '0';
                          ram_en  <= '0';
                          ram_wr  <= '0';
                        else
                          if(instruction(4 downto 3) = "00") then
                            rf_skip <= rf_result xor instruction(2);
                            rf_cond <= instruction(1 downto 0);
                            -- not used
                            rf_a    <= "00";
                            rf_b    <= "00";
                            rf_wr   <= '0';
                            ram_en  <= '0';
```

```
                            ram_wr  <= '0';
                            bus_imm <= "0000";
                            bus_imm_hl <= '0';
                          else
                            rf_a    <= "00";
                            rf_b    <= "00";
                            rf_wr   <= '0';
                            rf_cond <= "00";
                            rf_skip <= '0';
                            ram_en  <= '0';
                            ram_wr  <= '0';
                            bus_imm <= "0000";
                            bus_imm_hl <= '0';
                          end if;
                        end if;
                end if;
        end if;
        rom_addr <= rf_pc;
    end process;
end behavior;
```

9 .24.2 Prozessor-ALU (Verilog, dB)

Listing 17: Decis ALU (Verilog)

```
module  Core_ALU(
                // Data
                input           [15:0] BUS_A,
                input           [15:0] BUS_B,
                output          reg [15:0] BUS_Q,

                // Control
                input           EN,
                input           [4:0]   SEL,
                input           FR_CARRY,
                output reg      [6:0]   FR_ALU,
                output reg      [6:0]   FR_ALU_UPDATE
);

        // ------------------------------------------------------------------------------------------
        // --- internal variables
        // ------------------------------------------------------------------------------------------
        reg [16:0] alu_out;

        // All needed bundles
        wire    [1:0] instr_group;
        wire    [2:0] instr_type;

        reg     flag_carry,     flag_carry_up;
        reg     flag_zero,      flag_zero_up;
        reg     flag_parity,    flag_parity_up;
        reg     flag_equal,     flag_equal_up;
        reg     flag_greater,   flag_greater_up;
        reg     flag_smaller,   flag_smaller_up;
        reg     flag_overflow,  flag_overflow_up;

        // ------------------------------------------------------------------------------------------
        // --- internal assignments
        // ------------------------------------------------------------------------------------------
        assign  instr_group     =       SEL[4:3];
        assign  instr_type      =       SEL[2:0];

        // ------------------------------------------------------------------------------------------
        // --- Behaviour
        // ------------------------------------------------------------------------------------------

        // Calculations ------------------------------------------------------------------------------
        always_comb
  begin
  case(instr_group)
  'h0: begin
        case(instr_type[2:0])
        default :alu_out <= BUS_A;
        'h1:    alu_out <= BUS_A & BUS_B;
        'h2:    alu_out <= BUS_A | BUS_B;
        'h3:    alu_out <= BUS_A ^ BUS_B;
        'h4:    alu_out <= ~(BUS_A);
        'h5:    alu_out <= ~(BUS_A & BUS_B);
```

```
        'h6:    alu_out <= ~(BUS_A | BUS_B);
        'h7:    alu_out <= ~(BUS_A ^ BUS_B);
        endcase
end

'h1: begin
        case(instr_type[2:0])
        'h0:    alu_out <= BUS_A << BUS_B;      // LSL
        'h1:    alu_out <= BUS_A >> BUS_B;      // LSR
        'h2:    alu_out <=
                (BUS_A << BUS_B[3:0]) | (BUS_A >> ( 16 - BUS_B[3:0])); // Test ROL - TODO: TEST
        'h3:    alu_out <=
                (BUS_A >> BUS_B[3:0]) | (BUS_A << ( 16 - BUS_B[3:0])); // Test ROL - TODO: TEST
        'h4:    alu_out <= BUS_A |  (16'h1 << BUS_B[3:0]);      // SEB
        'h5:    alu_out <= BUS_A & ~(16'h1 << BUS_B[3:0]);      // CLB
        default:alu_out <= 0;
                        // TEB
        'h7:    alu_out <= { BUS_A[7:0], BUS_A[15:8]};                  // SWAP
        endcase
end

'h2: begin
        case(instr_type[2:0])
        'h0:    alu_out <= BUS_A + BUS_B;
        'h1:    alu_out <= BUS_A - BUS_B;
        'h2:    alu_out <= BUS_A + BUS_B + FR_CARRY;
        'h3:    alu_out <= BUS_A - BUS_B + FR_CARRY;
        'h4:    alu_out <= BUS_A + 16'h1;
        'h5:    alu_out <= BUS_A - 16'h1;
        'h6:    alu_out <= (~BUS_A) + 16'h1;
        default: alu_out <= 0;  //???  - RESERVED
        endcase
end

default: begin // RESERVED & CMPs
        alu_out <= 0;
end
endcase
end

        // Flags ------------------------------------------------------------------------------
always_comb     //      Comp
begin
        case(SEL)
        default:begin
                {flag_overflow, flag_carry, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_carry_up, flag_equal_up,
                        flag_greater_up, flag_smaller_up} <= 0;
        end
        'h10: begin     //ADD
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= alu_out[16];
        end

        'h11: begin     //SUB
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= ~alu_out[16];
        end

        'h12: begin     //ADC
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= alu_out[16];
        end

        'h13: begin     //SBC
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= ~alu_out[16];
        end

        'h14: begin     //INC
```

```
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= alu_out[16];
        end

        'h15: begin     //DEC
                {flag_overflow, flag_equal, flag_greater, flag_smaller} <= 0;
                {flag_parity_up, flag_overflow_up, flag_equal_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_carry_up   <= 1;
                flag_carry      <= ~alu_out[16];
        end

        'h0E: begin     //TEB
                { flag_overflow, flag_carry, flag_equal, flag_greater, flag_smaller} <= 0;
                { flag_parity_up, flag_overflow_up, flag_carry_up, flag_greater_up,
                        flag_smaller_up} <= 0;
                flag_equal_up   <= 1;
                flag_equal      <=      BUS_A[2^BUS_B[3:0]] ? 1 : 0;
        end

        'h1E: begin     //CMP
                { flag_overflow, flag_carry} <= 0;
                { flag_overflow_up, flag_carry_up} <= 0;
                { flag_equal_up, flag_greater_up, flag_smaller_up, flag_parity_up} <= 1;
                flag_equal      <=      BUS_A == BUS_B;
                flag_greater    <= \$signed(BUS_A) > \$signed(BUS_B);
                flag_smaller    <= \$signed(BUS_A) < \$signed(BUS_B);
        end

        'h1F: begin     //UCMP
                { flag_overflow, flag_carry} <= 0;
                { flag_overflow_up, flag_carry_up} <= 0;
                { flag_equal_up, flag_greater_up, flag_smaller_up, flag_parity_up} <= 1;
                flag_equal              <=      BUS_A == BUS_B;
                flag_greater    <= \$unsigned(BUS_A) > \$unsigned(BUS_B);
                flag_smaller    <= \$unsigned(BUS_A) < \$unsigned(BUS_B);
        end
        endcase
  end

        // ------------------------------------------------------------------------------------------
        // --- Assignments
        // ------------------------------------------------------------------------------------------
        assign  flag_zero       =       ~( |alu_out[15:0] );
        assign  flag_zero_up    =       EN;
        assign  flag_parity     =       ~( ^alu_out[15:0] );
        assign  FR_ALU          =       {flag_zero, flag_parity, flag_overflow,
                                        flag_carry, flag_equal,  flag_greater, flag_smaller};
        assign  FR_ALU_UPDATE   =       {flag_zero_up, flag_parity_up, flag_overflow_up,
                                        flag_carry_up, flag_equal_up, flag_greater_up,
                                        flag_smaller_up};
        assign  BUS_Q           =       (EN) ? alu_out[15:0] : 'z;

endmodule
```

9 .24.3 Prozessor-InstructionDecoder (Verilog, dB)

Listing 18: Decis Instruction Decoder (Verilog)

```
module  Core_InstructionDecoder (
        input           [15:0]  INSTRUCTION,
        output reg      [15:0]  ROM_ADDR,       //
        input                   WAIT,           //

        input           [15:0]  RF_PC,          //
        input                   RF_CONDITION_RESULT,
        output reg      [4:0]   RF_CONDITION,
        output reg              RF_PC_SKIP,
        output reg              RF_PC_WAIT,     //

        output reg      [2:0]   RF_ADDR_A,
        output reg      [2:0]   RF_ADDR_B,
        output reg      [2:0]   RF_ADDR_Q,
        output reg              RF_WR,
        output reg              RF_SPECIAL_WR,
```

```
        output reg              RF_SPECIAL_RD,

        output reg      [4:0]   ALU_SEL,

        output reg              ALU_EN,
        output reg              STACK_EN,
        output reg              LRAM_EN,
        output reg              IOC_EN,

        output reg              BUS_WR,
        output reg              BUS_LD_SEL,
        output reg              BUS_REPLACE_HL,
        output reg      [7:0]   BUS_REPLACE,
        output reg              BUS_IMMEDIATE_SEL,
        output reg      [15:0]  BUS_IMMEDIATE
);

        // ------------------------------------------------------------------------------
        // --- Enumerations
        // ------------------------------------------------------------------------------
        //typedef enum { IT_UNKNOWN, IT_ALU, IT_STLD, IT_CTRL } Instruction_Type;
        //Instruction_Type itype;

        // ------------------------------------------------------------------------------
        // --- Variables
        // ------------------------------------------------------------------------------

        // ------------------------------------------------------------------------------
        // --- Assignments (internal)
        // ------------------------------------------------------------------------------

        // ------------------------------------------------------------------------------
        // --- Behaviour
        // ------------------------------------------------------------------------------
  always_comb begin
        case(INSTRUCTION[15:13])
        3'h0, 3'h1: begin       // ALU
                RF_ADDR_A       <= INSTRUCTION[8:6];
                RF_ADDR_B       <= INSTRUCTION[5:3];
                RF_ADDR_Q       <= INSTRUCTION[2:0];
                {RF_WR, RF_SPECIAL_WR, RF_SPECIAL_RD} <=        3'b100;
                RF_PC_SKIP      <= 1'b0;
                RF_CONDITION    <= 4'h0;
                ALU_SEL         <= INSTRUCTION[13:9];
                {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b1000;
                BUS_WR          <= 1'b0;
                {BUS_LD_SEL, BUS_REPLACE_HL, BUS_IMMEDIATE_SEL} <= 3'b000;
                BUS_REPLACE     <= 8'h0;
                BUS_IMMEDIATE   <= 16'h0;
        end

        3'h2: begin             // Store & Load TODO
                RF_ADDR_A       <= INSTRUCTION[12] ?
                                        INSTRUCTION[8:6] : 3'h0; // if STORE -> 2. Src
                RF_ADDR_B       <= INSTRUCTION[5:3];
                RF_ADDR_Q       <= INSTRUCTION[12] ?
                                        3'h0 : INSTRUCTION[2:0]; // if LOAD  -> Dest
                RF_WR           <=      ~INSTRUCTION[12];       // @ LOAD
                BUS_WR          <= INSTRUCTION[12];     // @ STORE
                ALU_SEL         <= 4'h0;                // Also used for PASS!
                RF_PC_SKIP      <= 1'b0;                // nu
                RF_CONDITION    <= 4'h0;                // nu
                {BUS_LD_SEL, BUS_REPLACE_HL, BUS_IMMEDIATE_SEL} <= 3'b000;      // nu
                BUS_REPLACE     <= 8'h0;                // nu
                BUS_IMMEDIATE   <= 16'h0;       // nu
                case(INSTRUCTION[11:9]) // Src/Dest of Data
                3'h0:begin              // Special Register
                        RF_SPECIAL_WR   <= INSTRUCTION[12];
                        RF_SPECIAL_RD   <= ~INSTRUCTION[12];
                        {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b1000;
                end
                3'h1:begin              // Stack
                        {RF_SPECIAL_WR, RF_SPECIAL_RD}  <= 2'b00;
                        {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b0100;
                end
                3'h2:begin              // Ram
                        {RF_SPECIAL_WR, RF_SPECIAL_RD}  <= 2'b00;
                        {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b0010;
                end
                3'h3:begin              // IO
                        {RF_SPECIAL_WR, RF_SPECIAL_RD}  <= 2'b00;
                        {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b0001;
```

```
                end
                default :       begin
                        {RF_SPECIAL_WR, RF_SPECIAL_RD}  <= 2'b00;
                        {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b0000;
                end
                endcase
        end
        3'h3: begin             // Control TODO
                ALU_SEL         <= 4'h0;                // PASS
                RF_SPECIAL_RD   <= 1'b0;                // nu
                {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b1000; // nu ?
                BUS_WR          <= 1'b0;                // nu
                RF_ADDR_B       <= 3'h0;                // nu
                // Add InstrDec_Imm
                case(INSTRUCTION[12:11])        // Subcategory
                2'h0, 2'h1: begin       // LDL & LDH
                        RF_WR           <= 1'b1;
                        RF_ADDR_A       <= INSTRUCTION[2:0];
                        RF_ADDR_Q       <= INSTRUCTION[2:0];
                        {BUS_LD_SEL, , BUS_IMMEDIATE_SEL} <= 2'b10;                                     //
                            BUS_LD_SEL
                        BUS_REPLACE_HL  <= INSTRUCTION[11];     // High or low Byte
                        BUS_REPLACE     <= INSTRUCTION[10:3];
                        RF_SPECIAL_WR   <= 1'b0;                // nu
                        RF_PC_SKIP      <= 1'b0;                // nu
                        RF_CONDITION    <= 4'h0;                // nu
                        BUS_IMMEDIATE   <= 16'h0;       // nu
                end
                2'h2: begin                     // RJMP
                        RF_WR   <= 1'b1;                // Write to
                        RF_SPECIAL_WR   <= 1'b1;                // Specialregister :
                        RF_ADDR_Q       <= 3'h2;                // PC-R
                        {BUS_LD_SEL, , BUS_IMMEDIATE_SEL}               <= 2'b11;
                                // Sign-Extended 11bit immediate Value
                        BUS_IMMEDIATE <= { {5{INSTRUCTION[10]}}, INSTRUCTION[10:0] };
                        RF_ADDR_A <= 3'h0;              // nu
                        RF_PC_SKIP <= 1'b0;             // nu
                        RF_CONDITION <= 4'h0;           // nu
                        BUS_REPLACE_HL <= 1'b0;         // nu
                        BUS_REPLACE <= 8'h0;            // nu
                end
                default :begin                  // Subsubcategory
                        RF_WR           <= 1'b0;
                        RF_ADDR_A       <= 3'h0;
                        RF_ADDR_Q       <= 3'h0;
                        RF_SPECIAL_WR   <= 1'b0;
                        BUS_REPLACE_HL  <= 1'b0;
                        {BUS_LD_SEL, , BUS_IMMEDIATE_SEL} <= 2'b00;
                        BUS_REPLACE     <= 8'h0;
                        BUS_IMMEDIATE   <= 16'h0;
                        if(INSTRUCTION[10:9] == 2'b11) begin // Skip
                          RF_PC_SKIP    <= RF_CONDITION_RESULT;
                          RF_CONDITION  <= INSTRUCTION[4:0];
                        end else begin
                          RF_PC_SKIP    <= 1'b0;
                          RF_CONDITION  <= 4'h0;
                        end
                end
                endcase/**/
        end

        default :       begin           // Reserved
                RF_ADDR_A <= 3'h0;
                RF_ADDR_B <= 3'h0;
                RF_ADDR_Q <= 3'h0;
                {RF_WR, RF_SPECIAL_WR, RF_SPECIAL_RD} <=        3'b000;
                RF_PC_SKIP <= 1'b0;
                RF_CONDITION <= 4'h0;
                ALU_SEL <= 4'h0;
                {ALU_EN, STACK_EN, LRAM_EN, IOC_EN} <= 4'b0000;
                BUS_WR <= 1'b0;
                {BUS_LD_SEL, BUS_REPLACE_HL, BUS_IMMEDIATE_SEL} <= 3'b000;
                BUS_REPLACE <= 8'h0;
                BUS_IMMEDIATE <= 16'h0;
        end
        endcase
end

        // ------------------------------------------------------------------------
        // --- Assignments (external)
        // ------------------------------------------------------------------------
assign          ROM_ADDR        = RF_PC;
```

```
  assign        RF_PC_WAIT      = WAIT;

endmodule
```

9 .24.4 Prozessor-RegisterFile (Verilog, dB)

Listing 19: Decis Registerfile (Verilog)

```
module  Core_Registerfile (
        output  reg [15:0] outA,
        output  reg [15:0] outB,
        input   [15:0]  inC,

        input   [2:0]   addrA,
        input   [2:0]   addrB,
        input   [2:0]   addrC,

        input   clk,
        input   res,
        input   write,
        input   specialRead,
        input   specialWrite,

        input   pcSkip,
        input   pcFreeze,

        input   [15:0]  fr_alu_update,
        input   [15:0]  fr_alu_input,
        output  reg     fr_carry_out,

        input   [4:0]   inCondition,
        output  reg     outConditionResult,

        output  reg [15:0] rom_addr
);

        // ------------------------------------------------------------------
        // --- Variables
        // ------------------------------------------------------------------

        // GPR
        reg     [15:0] _inC_buf;
        reg     [15:0] _gpr_registerfile [7:0];

        // SR
        reg     [15:0] _fr;
        reg     [15:0] _pc;

        reg     [15:0] _outA_GPR, _outB_GPR;
        reg     [15:0] _outA_SR, _outB_SR;
        reg     _freeze;

        assign  _freeze = pcFreeze;

        // ------------------------------------------------------------------
        // --- Behaviour - GPRs
        // ------------------------------------------------------------------

        // GPR Buffer ------------------------------------------------------
        always @(~clk)
        begin
          _inC_buf <= inC;
        end

        // GPR Write -------------------------------------------------------
        always_ff @(posedge clk)
        begin
          if(write & ~_freeze & ~specialWrite) begin
                _gpr_registerfile[addrC] <= _inC_buf;
          end
        end

        // GPR Read A & B --------------------------------------------------
        assign  _outA_GPR       = _gpr_registerfile[addrA];
        assign  _outB_GPR       = _gpr_registerfile[addrB];

        // ------------------------------------------------------------------
        // --- Behaviour - Special Registers
```

```
        // ------------------------------------------------------------------

        // Program Counter ---------------------------------------------------
        always @(posedge clk) begin
          if(res) begin
                _pc <= 16'h0000;
          end else begin
                if(specialWrite & ~_freeze) begin
                        case(addrC)
                        4'h0:    _pc <= inC;
                        4'h2:    _pc <= _pc + inC;
                        default: _pc <= _pc + 16'h1;
                        endcase
                end else if( ~_freeze) begin
                        _pc <= _pc + 16'h1;
                end
          end
        end

        assign  rom_addr = _pc;

        // Flagregister ------------------------------------------------------
  integer i;
  always @(posedge clk) begin
        if(res) begin
                _fr <= 16'h0000;
        end else begin
          if(specialWrite) begin
                case(addrC)
                4'h4            : _fr <= inC>>1;
                default : _fr <= _fr;
                endcase
          end else begin
                for( i = 0; i < 9; i = i + 1) begin
                        if(i == 0)
                                _fr[i] = 1;
                        else if(fr_alu_update & 1 << i) begin
                                _fr[i] = fr_alu_input[i];
                        end
                end
                //if(fr_alu_update & 16'h1)
                //  fr[]
          end
        end
  end

  wire          condition          = _fr[inCondition[3:0]];
  assign        outConditionResult = inCondition[4] ? condition : ~condition;
  assign        fr_carry_out       = _fr[4];

  /* always @(posedge clk) begin
        if(res) begin
                _pc <= 16'h0;
        end else if(specialWrite & Write) begin
                case(addrC)
                4'h0            : _pc <= inC;
                default :
                endcase
        end
  end */

        // ------------------------------------------------------------------
        // --- Assignments (external)
        // ------------------------------------------------------------------

  assign        outA    = (!specialRead) ? _outA_GPR : _outA_SR;
  assign        outB    = (!specialRead) ? _outB_GPR : _outB_SR;

endmodule


module  HWTB_Core_Registerfile (
        input                           clk,
        input                           res,

        // Test Results
        output reg  [15:0] sevenseg,
        output reg   [9:0] leds,

        // Module Test IO
        input           [15:0] outA,
        input           [15:0] outB,
```

```
        output reg      [15:0] inC,

        output reg      [2:0] addrA,
        output reg      [2:0] addrB,
        output reg      [2:0] addrC,

        output reg      clk_rf,
        output reg      res_rf,
        output reg      write,
        output reg      specialRead,
        output reg      specialWrite,

        output reg      pcSkip,
        output reg      pcFreeze,

        output reg      [15:0] fr_alu_update,
        output reg      [15:0] fr_alu_input,
        input           fr_carry_out,

        output reg       [4:0] inCondition,
        input           outConditionResult,

        input           [15:0] rom_addr
);
endmodule
```

9 .24.5 Prozessor-IO controller (Verilog, dB)

Listing 20: Decis IO controller (Verilog)

```
module  Core_IOController (
        input   clk,
        input   res,
        input   enable,
        input   write,
        input   [15:0]  addrIn,
        input   [15:0]  dataIn,
        output reg [15:0]       dataOut,
        input   [15:0]          p0_in,
        output reg [15:0]       p0_out,
        input   [15:0]          p1_in,
        output reg [15:0]       p1_out,
        input   [15:0]          p2_in,
        output reg [15:0]       p2_out,
        input   [15:0]          p3_in,
        output reg [15:0]       p3_out
);
        // ------------------------------------------------------------------
        // --- Variables
        // ------------------------------------------------------------------
        reg     [15:0] bufferIn [3:0];
        // ------------------------------------------------------------------
        // --- Behaviour
        // ------------------------------------------------------------------
        // Read
  always_ff @(~clk) begin
                bufferIn[2'h0] <= p0_in;
                bufferIn[2'h1] <= p1_in;
                bufferIn[2'h2] <= p2_in;
                bufferIn[2'h3] <= p3_in;
  end
  assign        dataOut = (enable && ~write) ? bufferIn[addrIn[1:0]] : 16'h0000;
        // Write
  always_ff @(clk) begin
        if(enable && write) begin
                case(addrIn[1:0])
                default: p0_out <= dataIn;
                2'h1:    p1_out <= dataIn;
                2'h2:    p2_out <= dataIn;
                2'h3:    p3_out <= dataIn;
                endcase
        end
  end
endmodule
```

9 .24.6 Prozessor-Stack (Verilog, dB)

Listing 21: Decis Stack (Verilog)

```
module  Core_Stack #(
        // ------------------------------------------------------------------
        // --- Parameters
        // ------------------------------------------------------------------
        parameter WIDTH = 16, // -> (WIDTH % 8)*(2^Depth) B in size = 256B
        parameter  DEPTH =  8  // -> (2^Depth) pushes possible = 128
)       (       // ------------------------------------------------------------
                // --- Operands
                // ------------------------------------------------------------
                input   wire    [WIDTH-1:0]     dataIn,
                output  reg     [WIDTH-1:0]     dataOut,
                //output reg    [DEPTH-1:0]     ptrOut,         // DEBUG
                input   wire    clk,
                input   wire    res,
                input   wire    pop_push,
                input   wire    en,
                output  reg     full,
                output  reg     empty
        );

        // ------------------------------------------------------------------
        // --- Variables
        // ------------------------------------------------------------------
        reg     [DEPTH-1:0] ptr;
        //assign ptrOut = ptr;  // DEBUG
        reg     [WIDTH-1:0] memory [2^(DEPTH)-1:0];
        reg     [WIDTH-1:0] dataOut_t;

        // ------------------------------------------------------------------
        // --- Behaviour
        // ------------------------------------------------------------------
  assign        empty   = (ptr == {DEPTH{1'b0}}) ? 1'h1 : 1'h0;
  assign        full    = (ptr == {DEPTH{1'b1}}) ? 1'h1 : 1'h0;
  assign        dataOut_t = memory[ptr - {{DEPTH-1{1'b0}}, 1'b1 } ]; // ptr - 1
  assign        dataOut = (pop_push && en) ? dataOut_t : {WIDTH{1'bZ}};

        // Save input -------------------------------------------------------
  always_ff @(posedge clk) begin
        if(~pop_push && en && ~full) begin
                memory[ptr] <= dataIn;
        end
  end
        // Ptr control ------------------------------------------------------
  always_ff @(posedge clk) begin
        if (res) begin  // RESET
                ptr     <= 0;
                                        // ptr = 0
        end else
        if (pop_push && en && ~empty)  begin     // POP
                ptr     <= ptr - { {DEPTH-1{1'b0}}, 1'b1 };     // ptr - 1
        end else
        if (~pop_push && en && ~full)   begin   // PUSH
                ptr     <= ptr + { {DEPTH-1{1'b0}}, 1'b1 };     // ptr + 1
        end else begin
                        // IDLE
                ptr     <= ptr;
        end
  end
endmodule
```

10 Digitaltechnik mit FPGA u. VHDL (DE0 Boards)

FPGA ..... Field Programmable Gate Array,
ein IC mit vielen vorgefertigten Gatterschaltungen, aber die Verbindungen unter diesen Gattern kann man hinein-*"flashen"*

VHDL ..... Programmiersprache für Logikschaltungen
('VLSI Hardware Definition Language')

Verilog ..... amerikanische Alternative zu VHDL

DE0 Board . FPGA Entwicklungs-Kit für *'Altium Designer'* und *'Quartus-II'* von *'Altera'*

FPGA Varianten:

- mit EEPROM-Speicher für die Verbindungen
- mit RAM-Speicher für die Verbindungen,
  dessen Inhalt bei einem *'boot'*- Vorgang
  aus externem EEPROM geladen wird

simpler Instruction Decoder

(vormals 7-Seg. Decoder)

Decis Entwurfsanleitung:

→ obelix: "ProzessorDesign-Deci.pdf"

→ lokal: "ProzessorDesign-Deci.pdf"

Decis FPGA-Quartus-Files:

→ obelix: "ProzessorRd3Db.zip"

→ lokal: "ProzessorRd3Db.zip"

bitte selber downloaden: (hier nur leere Files)

→ obelix: "cyclone_web-13.0.1.232.qdz (582MB)"

→ obelix: "ModelSimSetup-13.0.1.232.exe (798MB)"

→ obelix: "QuartusSetupWeb-13.0.1.232.exe (1.6GB)"

siehe Kap. "VHDL" im "Dic5"-Script

→ obelix: "Dic5RXh01L.pdf"

→ lokal: "Dic5RXh01L.pdf"

→ obelix: "DS_David_B_Jakob_H.pdf"

→ lokal: "DS_David_B_Jakob_H.pdf"

10 .1 FPGAprojekt: DE0-Board Einstieg

QuartusII/Altium/DE0board: Funktionstest mit Minimalschaltung

→ obelix: "fpga_tutorial_HN5.pdf"

→ lokal: "fpga_tutorial_HN5.pdf"

11 PSoC — (freies Programm)

11 .1 SingleChip, uC, SoC . . . nix Neues

"PSoC" steht für *"Programmable System on Chip"*.

Das bedeutet so ziemlich dasselbe wie "SoC" (System on a Chip), "Single Chip Computer" (Intel) oder "MicroController" (Microchip) — die Hersteller erfinden sich vielfach eigene Bezeichnungen und Massenweise Abkürzungen, teils aus Urheberrechtsgründen, teils um Alleinstellung vorzugaukeln (zB: *"19 PAB, 30 PDB, 4xCMP, 4xUAB+CTB, NDA required"*). Während etwa Cypress behauptet, ein SoC müsse über *"vielfältige analoge und digitale Funktionseinheiten auf einem Chip"* (Cypress SCHILF PSoC5LP 31.05.2016 0900-1320 Uhr HTBLuVA Anichstraße, S.7) verfügen (sind DACs, ADCs, Komparatoren ... nicht analog?), sagt `https://de.wikipedia.org/wiki/System-on-a-Chip`: *[...] Unter System-on-a-Chip (SoC, dt. Ein-Chip-System), auch System-on-Chip, versteht man die Integration aller oder eines großen Teils der Funktionen eines programmierbaren elektronischen Systems auf einem Chip (Die), also einem integrierten Schaltkreis (IC) auf einem Halbleiter-Substrat, auch monolithische Integration genannt. Bei der Nutzung von Silizium als Substratmaterial spricht man auch von System-on-Silicon (SoS). Als System wird dabei eine Kombination unterschiedlicher Elemente (logischen Schaltungen, Taktgebung, selbständiges Anlaufen, mikrotechnische Sensoren usw.) aufgefasst, die zusammen eine bestimmte Funktionalität bereitstellen, beispielsweise ein Beschleunigungssensor samt Auswertungselektronik. Eingesetzt werden SoCs üblicherweise in eingebetteten Systemen. Während Systeme anfänglich aus einem Mikroprozessor- oder Mikrocontroller-IC und vielen anderen ICs für spezielle Funktionen bestanden, die auf einer Platine aufgelötet waren, lässt die heute mögliche Integrationsdichte zu, nahezu alle Funktionen auf einem einzigen IC zu vereinigen. Dabei werden digitale, analoge und Mixed-Signal-Funktionseinheiten integriert. Vorteile sind vor allem Kosteneinsparung, geringerer Energieverbrauch beziehungsweise Verlustleistung und umfassende Miniaturisierung. So ist heute beispielsweise bei Mobiltelefonen die digitale Funktion, gegebenenfalls mit Ausnahme des Speichers, auf einem IC realisiert. Auch die Schnittstellen beispielsweise zur Tastatur, zur SIM-Karte oder zum Display sind bereits auf diesem IC enthalten.*

Eine ähnliche Technik, um hohe Integrationsdichten auch von Bauelementen von stark unterschiedlicher Technik zu erreichen, ist das sogenannte System-in-Package (SiP). Dabei werden mehrere Chips in einem Gehäuse zusammengefasst. [...]

oder engisch:
`https://en.wikipedia.org/wiki/System_on_a_chip`:
[...] A system on a chip or system on chip (SoC or SOC) is an integrated circuit (also known as an "IC" or "chip") that integrates all components of a computer or other electronic systems. These components typically include a central processing unit (CPU), memory, input/output ports and secondary storage - all on a single substrate. It may contain digital, analog, mixed-signal, and often radio-frequency functions, depending on the application. SoCs are very common in the mobile computing market because of their low power consumption. SoCs are commonly applied in the area of embedded systems. Systems on Chip are in contrast to the common traditional motherboard-based PC architecture, which separates components based on function and connects them through a central interfacing circuit board, so-termed a "mother board". Whereas a motherboard houses and/or connects detachable or replaceable components such as a CPU, graphics and memory interfaces, hard-disk and USB connectivity, memories (both random access and read only) and secondary storage; SoCs integrate all of these components into a single integrated circuit, as if all these functions were built into the motherboard. More integrated hardware designs improve performance and reduce power consumption and semiconductor die area needed for an equivalent design at the cost of reduced modularity and replaceability of components, and SoC designs are by definition fully- or nearly fully integrated across different components. For these reasons, there has been a general trend towards tighter integration of components in the computer hardware industry, in part due to the influence of SoCs and lessons learned from the mobile and embedded computing markets. A SoC integrates a microcontroller (or microprocessor) with advanced peripherals like graphics processing unit (GPU), Wi-Fi module, or coprocessor. Similar to how a microcontroller integrates a microprocessor with peripheral circuits and memory, a SoC can be seen as integrating a microcontroller with such advanced peripherals. In general, there are three distinguishable types of SoCs:

SoCs built around a microcontroller;

SoCs built around a microprocessor (often found in mobile phones);

and specialized SoCs designed for specific applications that do not fit into the above two categories.

A separate category may be Programmable SoC (PSoC), where some of the internal elements can be programmable in a manner analogous to a field-programmable gate array (FPGA) or a complex programmable logic device (CPLD).

[...]

11 .2 Interview mit KU, 11.Jul'16, 02.Mai'18

kann iXH des als 'IoT'-Device verwenden?

- ja (110616)

ist also ein LAN Interface mit drauf?

Nein! Aber man kann sich die Hardware dazubauen und den Protokollstack dazuschreiben.
Man könnte auch ein Arduino-Shield verwenden.
Auf den neuen PSoc ist auch Bluetooth mit drauf für serielle Kommunikation im RS232-Mode.

(ich denke, "IoT" heißt *Internet* of Things?

ist WLAN drauf?

Ja, da gibt es den bekannten EN... (Nummer vergessen) Chip, mit dem kann man sich das bauen.

und sonst?

From: Michael Kupfner Monday - July 11, 2016 7:30 AM To: Christoph Schoenherr Subject: Antw: pSoC-in-FSST4u5 Hallo Christoph,
... schau' Dir einmal die Seite von Cypress an - speziell zu PSoC5:

http://www.cypress.com/products/32-bit-arm-cortex-m3-psoc-5lp

Sowohl beim Board von Felix als auch bei meinem wird der CY8CKIT-059 Stick verwendet:

http://www.cypress.com/documentation/development-kitsboards/cy8ckit-059-psoc-5lp-prototyping-kit-onboard-programmer-and
Zu Deinen Fragen:

hat das pSOC ein Linux und eine gcc-Toolchain drauf?

- die IDE läuft nur unter Windows
- der PSoC5 selbst hat kein Linux "drauf", dafür ist der Speicher zu klein (... müsste extern erweitert werden); wenn dann könnte es nur uCLinux sein
- gcc Toolchain läuft im Hintergrund (für die Code-Generierung des Cortex M3)

von Linux aus ansprechbar via LAN, Usb-Gadget undso?

- via LAN über externes Arduino-Shield (Ethernet-Shield, WLAN-Shield, ESP...)
- USB 2.0 Slave Interface ist nativ vorhanden
-> alles nur eine Frage der Programmierung... aber das passt ja genau zum FSST Unterricht!
;-)

12 RasPi - RaspberryPi 3B

12 .1 Allgemein

Der Raspberry Pi (RasPi, rPi) ist ein Einplatinencomputer [...]. Der Rechner enthält ein Ein-Chip-System von Broadcom mit einem ARM-Mikroprozessor, die Grundfläche der Platine entspricht etwa den Abmessungen einer Kreditkarte (85 x 56 x 17mm).
[...]
Als Betriebssystem kommen vor allem angepasste Linux-Distributionen mit grafischer Benutzeroberfläche zum Einsatz; für das neueste Modell existiert auch Windows 10 in einer speziellen Internet-of-Things-Version ohne grafische Benutzeroberfläche. Der Startvorgang erfolgt gewöhnlich von einer wechselbaren SD-Speicherkarte als internes Boot-Medium. Bei der neueren Generation mit dem BCM2837 ist der Start auch von einem USB-Massenspeicher oder Netzwerk möglich. Eine native Schnittstelle für Festplattenlaufwerke ist nicht vorhanden, zusätzlicher Massenspeicher kann per USB-Schnittstelle angeschlossen werden, beispielsweise externe Festplatten bzw. SSDs oder USB-Speichersticks.
[...]
Am 29. Februar 2016 wurde der Raspberry Pi 3 Model B vorgestellt. Er erweitert das Vorgängermodell um integriertes WLAN und Bluetooth Low Energy und hat eine schnellere CPU mit 64-bit-ARMv8-Architektur.

Ende 2016 wurde die Version 1.2 des Raspberry-Pi-2-Modell B mit der neuen CPU des Modell 3 vorgestellt. Abgesehen von dieser ist die Ausstattung identisch mit der des ursprünglichen Modell 2; auch die CPU ist weiterhin nur mit 900 MHz getaktet, anstatt der 1200 MHz des Modell 3.

Am 16. Januar 2017 wurde das Compute Module 3 (CM3) vorgestellt. Es hat den SoC des Raspberry Pi 3 und 1 GB RAM (vorher: 512 MB). Die CPU-Leistung soll sich im Vergleich zum CM1 etwa verzehnfacht haben. Das CM3 ist in zwei Varianten verfügbar: eine Standardvariante und eine Lite (CM3L), wobei letztere nicht über den aufgelöteten 4-GB-Flashspeicher verfügt. Das CM3 ist mit dem CM1 kompatibel, einziger sichtbarer Unterschied ist die um 1 mm gewachsene Breite.

Am 28. Februar 2017 wurde der Raspberry Pi Zero W vorgestellt. Die Ausstattung des Modells ist nahezu identisch zum Raspberry Pi Zero, wurde jedoch durch den schon beim Raspberry-Pi-3-Modell B eingesetzten zusätzlichen Chip um die Funktionalität von integriertem WLAN und Bluetooth Low Energy erweitert.

Seit dem 12. Januar 2018 ist der Raspberry Pi Zero WH erhältlich, der technisch dem Raspberry Pi Zero W entspricht aber dessen 40-poligen Anschluss bereits werkseitig mit entsprechenden Pfostensteckern versehen ist.

Am 14. März 2018 (Pi-Tag) wurde das Raspberry-Pi-3-Modell B+ vorgestellt. Der Prozessortakt wurde um 200 MHz auf 1400 MHz erhöht und ein neues Funkmodul kommt zum Einsatz. Dieses beherrscht nun auch 5-GHz-WLAN nach dem IEEE-802.11ac-Standard und Bluetooth 4.2. Außerdem verfügt es jetzt auch über Gigabit-Ethernet, welches jedoch weiterhin über den einzigen USB-Port angebunden ist und damit die max. Übertragungsrate auf ca. 300 MBit/s limitiert. Das neue Modell ist vorbereitet für Power over Ethernet.

Der Raspberry Pi stellt eine frei programmierbare Schnittstelle für Ein- und Ausgaben bereit. Diese Allzweckeingabe/-ausgabe wird auch als „GPIO“ („General Purpose Input/Output“) bezeichnet. Über diese Schnittstelle können LEDs, Sensoren, Displays und andere Geräte angesteuert werden. Es gibt fünf GPIO-Anschlüsse, wobei im Allgemeinen nur der Anschluss P1 gebraucht wird. Die GPIO-Schnittstelle P1 besteht bei Modell A und Modell B aus 26 Pins und bei Modell A+ und Modell B+ aus 40 Pins, jeweils ausgeführt als doppelreihige Stiftleiste, wovon

- 2 Pins eine Spannung von 5 Volt bereitstellen, aber auch genutzt werden können, um den Raspberry Pi mit Strom zu versorgen,
- 2 Pins eine Spannung von 3,3 Volt bereitstellen,
- 1 Pin als Masse dient,
- 4 Pins, die zukünftig eine andere Belegung bekommen könnten, derzeit ebenfalls mit Masse verbunden sind,
- 17 Pins (Modell A und B) bzw. 26 Pins (Modell A+ und B+, sowie Raspberry Pi 2 Modell B), welche frei programmierbar sind. Sie sind für eine Spannung von 3,3 Volt ausgelegt. Einige von ihnen können Sonderfunktionen übernehmen:
  - 5 Pins können als SPI-Schnittstelle verwendet werden,
  - 2 Pins haben einen 1,8-$k\Omega$-Pull-up-Widerstand (auf 3,3 V) und können als I^2C-Schnittstelle verwendet werden,
  - 2 Pins können als UART-Schnittstelle verwendet werden.

Mit dem Modell B+ wurde eine offizielle Spezifikation für Erweiterungsplatinen, sogenannte Hardware attached on top (HAT), vorgestellt.

Jeder HAT muss über einen EEPROM-Chip verfügen; Darin finden sich Herstellerinformationen, die Zuordnung der GPIO-Pins sowie eine Beschreibung der angeschlossenen Hardware in Form eines "device tree"-Abschnitts. Dadurch können die nötigen Treiber für den HAT automatisch geladen werden. Auch die genaue Größe und Geometrie des HAT sowie die Position der Steckverbinder werden dadurch festgelegt. Modell A+ und Raspberry Pi 2 Modell B sind mit diesen ebenfalls kompatibel.
Die in der Revision 2 hinzugekommene GPIO-Schnittstelle P6 erlaubt es, den Raspberry Pi zurückzusetzen bzw. zu starten, nachdem er heruntergefahren wurde.
Zur Steuerung der GPIOs existieren Bibliotheken für zahlreiche Programmiersprachen. Auch eine Steuerung durch ein Terminal oder Webinterfaces ist möglich.

(aus `https://de.wikipedia.org/wiki/Raspberry_Pi`, 03Jun18)

SSH-Server aktivieren
1. Nachdem das Raspian-Image erfolgreich auf die SD-Karte geschrieben wurde, wird die /boot Partition im Windows-Explorer geoffnet.
2. Nun im root-Bereich (in keinem Unterverzeichnis!) der Partition ein leeres File mit Namen SSH anlegen. Achtung, darf keine File-Endung wie z.B. .txt haben!
3. Im Windows-Explorer dazu auf View oder Ansicht klicken und ein Hacken aneben dem Feld File name extension setzen.
4. Nun mit z.B. Notepad ein leeres File erzeugen und dann im Explorer umbenennen und die File-Endung loschen.

IP-Adresse herausfinden bzw. einstellen
Sowohl die Ethernet- als auch die WLAN-Schnittstelle sind standardmäßig auf DHCP eingestellt. Wenn man den Raspi nun in ein Netzwerk mit DHCP-Server einsteckt, kann man entweder uber den Server oder uber Tools wie Angry IP-Scanner (Windows) oder einen arp-scan -localnet (Linux/Mac) herausfinden welche IP dem Raspi zugeordnet wurde. Wenn man aber eine statische IP möchte kann man dies folgendermaßen erreichen:

- 1. Wie vorher im Windows-Explorer die /boot Partition öffnen und das File cmdline.txt finden und öffnen.
- 2. Erste Moglichkeit: Am Ende des files (ACHTUNG: Keinen Zeilenumbruch einfugen!!!), nur durch ein Leerzeichen getrennt die gewunschte IP in der Form:
  ip=192.168.1.115
  eingeben und wieder durch ein Leerzeichen getrennt
  net.ifnames=0.
  Info: Die Optionen in diesem File werden beim Boot-Vorgang dem Kernel ubergeben - einmal die statische IP zu verwenden und die zweite Option bezieht sich auf das alte (einfache) Namesschema fur die Netzwerk Interface-Namen. Nun dem Laptop auch eine statische IP im gleichen Netz zuweisen und schon kann man via ssh auf den Raspi zugreifen.
  Zweite Moglichkeit: Eine Alternative wäre es dem Kernel keine statische Adresse zuzuweisen und darauf vertrauen, dass sich der Raspi und euer Windows-Laptop eine Adresse aus dem APIPA (automatic private ip addressing 169.254.x.x/16) Bereich vergeben. Dies ist sinnvoll, weil dann dieam User-Laptop auf DHCP verbleiben kann und nicht manuell auf eine statische IP umgestellt werden muss. Um dann aber die tatsachliche IP des Raspis herauszufinden braucht man Hilfswerkzeuge wie Wireshark, Angry IP-Scanner, etc.
  Dritte Moglichkeit: Ansprechen des Raspi uber den Namen raspberrypi.local. Dazu muss dein Rechner Bonjour fahig sein (Apple ITunes oder der appleBonjour Dienst installiert) letzterer ware ggf. mit BonjourPSSetup.exe zu installieren.

(aus "FSST Raspberry Pi 3"
"no ©by Markus Signitzer September 2017 HTL Anichstraße",
Raspi_FirstSteps_05.pdf)

12 .2 IP Socket Programming

12 .2.1 Richards kleiner Webserver

Listing 22: Richards kleiner Webserver

```
/* File Richards 'wpeep.c' , cfs@GcF,PbQ
 * shows message from Web-Browser
 * +sends very simple reply
 * compile: gcc wpeep.c -o wpeep
 * run    : ./wpeep <portNr>  (zB "./wpeep 1234")
 * stop   : Strg+C   oder "serverstop" im URLstring
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#ifndef SO_REUSEPORT
 #define SO_REUSEPORT 15
#endif
#define IpPort  (argv[1])
int inhaltAnzeigenSieheUnten(char*);

    /*MAIN:*/
int main(int argc,char **argv){
  int           fdPORT, fdCOMM, lenC, r1=0;
  char          buf[4096], buf2[4096], *s, hostName[256];
  struct sockaddr_in    s_address;
  struct sockaddr       c_address;

  if(argc<2){ printf("usage: %s <PortNumber>",*argv); exit(0); }
  gethostname(hostName, 255);
  fdPORT = socket(AF_INET,SOCK_STREAM,0);
  s_address.sin_family      = AF_INET;
  s_address.sin_addr.s_addr = htonl(INADDR_ANY);        //ex:inet_addr("127.0.0.1");
  s_address.sin_port        = htons(atoi(IpPort));
  int one=1; setsockopt(fdPORT, SOL_SOCKET, SO_REUSEPORT, &one, sizeof one);
  if(bind(fdPORT,(struct sockaddr *) &s_address, sizeof(s_address))){
     perror("bind failed"); exit(EXIT_FAILURE);
  }
  listen(fdPORT,5);
  for(;;){
    printf("server: %s@%s waiting at port:%d\n", argv[0], hostName, atoi(IpPort));
    lenC    = sizeof(c_address);
    fdCOMM  = accept(fdPORT, &c_address, &lenC);
    if(1 > (r1=read(fdCOMM, buf, sizeof(buf)-2 ))) break; /*nix zu lesen*/
    buf[r1]=0; inhaltAnzeigenSieheUnten(buf);
    snprintf(buf2, sizeof(buf2)-1,
        "HTTP/1.0 200 OK\r\nServer: RWS\r\nContent-Type: text/html\r\n\r\n"
        "<html><body bgcolor=tan><h1>RWS - kleiner Webserver</h1>\n"
        "<br>Nur zum Testen, ein kleiner text...\n"
        "<hr><small>(c)2005 richard weinberger</small></body></html>\r\n\r\n"
        );
    write(fdCOMM, buf2, strlen(buf2));
    close(fdCOMM);
    if(strstr(buf,"serverstop")) break; /*"geheim"-shutdown URLstring*/
  }
  shutdown(fdPORT,SHUT_RDWR);
  close(fdPORT);
  printf("\nserver \"%s\" terminated.\n",*argv);
```

```
}/*end of main*/


int inhaltAnzeigenSieheUnten(char *s){
  printf("\n*--------new-message:------------*\n");
  for(;*s;s++){
    if(isprint(*s)) putchar(*s);
    else printf("(0x%02X)",*s);
    if('\n'==*s) putchar('\n');
  }
  printf("\n*--------message-end-------------*\n");
}
```

12 .2.2 Richards kleiner Webserver mit fork()

Listing 23: Richards kleiner Webserver mit fork()

```
/* File "websrv1.c"
 *
 * minimal-webserver, reentrant (fork())
 * by JESUS.RW 15-Mar-2005
 * cfs 28-Mar-2005
 ********************************************************************************
 */

#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>

/* ****************************************function prototypes: ***/
void send_to_client(const char *, int );
void read_http_request(int );
void child_exit_handler();

/* ****************************************config****************/
#define SERVERNAME      "rws"
#define SERVERVERSION   "0.01"
#define MAXQUEUELEN     5
#define BUFFERLEN       4096
#define LISTENPORT      80

/* ********************************************************************
 * *** M A I N ***
 ********************************************************************/
int main(){
 int    server_socket;
 int    new_client;
 int    client_len;
 pid_t  pid;
 struct sockaddr_in     server, client;
 struct sigaction       mysignal;

 printf("%s v %s"
        "\n----------\n"
        "no features, no performance, no sense, no nothing...\n\n"
        ,SERVERNAME, SERVERVERSION);
 if((server_socket = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) == -1){
        printf("socket() failed\n");
        return(-1);
 }
```

```
/* ****************************************socket:***/
server.sin_family = AF_INET;
server.sin_addr.s_addr = htonl(0);
server.sin_port = htons(LISTENPORT);
client_len = sizeof client;

/* ****************************************signalling:***/
mysignal.sa_handler = child_exit_handler;
sigfillset(&mysignal.sa_mask);
mysignal.sa_flags = SA_RESTART;
sigaction(SIGCHLD, &mysignal, 0);

if(bind(server_socket, (struct sockaddr*)&server, sizeof server) != 0){
       printf("bind() failed\n");
       return(-1);
}
if(listen(server_socket, MAXQUEUELEN) != 0){
       printf("listen() failed\n");
       return(-1);
}
while( -1 != (new_client = accept(server_socket, (struct sockaddr*)&client, &client_len))){
 if((pid = fork()) < 0){
       printf("fork() failed!\n");
       return(-1);
 }else if(pid == 0){   /* im child */
       close(server_socket);
       read_http_request(new_client);
       send_to_client("HTTP/1.0 200 OK\r\n"
               "Server: RWS\r\n"
               "Content-Type: text/html\r\n\r\n"
               "<html><head><title>RWS - ein kleiner Webserver</title></head>\n"
               "<body><h1>RWS - ein kleiner Webserver</h1>\n"
               "<p>Nur zum Testen, ein kleiner text...</p><hr>\n"
               "<h9>(c)2005 richard weinberger <richard@nod.at></h9></body></html>\r\n"
               "\r\n"
               , new_client);
       printf("child: beende mich.\n");
       _exit(0);
       printf("child: ich lebe noch -> BUG BUG BUG!\n");
 }else{
       close(new_client);
       printf("close()\n");
 }
}
printf("cu...\n");
return(0);
}


/* ****************************************send_2_client()**
 */
void send_to_client(const char *message, int client){
 unsigned int   len;
 len = strlen(message);
 write(client, message, len);
}

/* ****************************************http_request()**
 */
void read_http_request(int client){
 char                   buffer[BUFFERLEN];
 int                    read_count;
 unsigned short int     ok = 0;
 while(ok < 2){
  read_count = read(client, buffer, BUFFERLEN);
  if(buffer[read_count - 1] == '\n' && buffer[read_count - 2] == '\r'){
       ok++;
       if(buffer[read_count - 3] == '\n') ok++;
```

```
  }
 }
}

/* ********************** signal handler: ***************
 * child_exit_handler()
 * ********************************************************/
void child_exit_handler(){
 while (waitpid(-1, NULL, WNOHANG) > 0);
}
```

12 .2.3 simpler Browser-Fake

Listing 24: Browser-Fake

```
/* File wclnt2.c =BrowserFake/* Socket-Programmierung, cfs@GcF,Sf9 */

#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <netdb.h>

int main(int argc,char **argv){
  int                   fd1, result, l;
  char                  s[16384], hname[256];
  struct sockaddr_in    address;

  address.sin_family=AF_INET;
  address.sin_addr.s_addr = inet_addr("10.10.10.30");
  address.sin_port = htons(80);  /*(NetworkByteOrder)*/
  printf("client \"%s\" @%s\n",*argv,(gethostname(hname,255),hname));
  fd1=socket( AF_INET, SOCK_STREAM, 0); /* == socket erzeugen und verbinden **/
  if((result=connect(fd1,(struct sockaddr *) &address,sizeof(address))) < 0){
      perror("cant connect\n");
  }else{
      sprintf(s,"GET /SMUE/srcvw.php?fn=weliza/weliza1.c&nr=123 HTTP1.0/\n"
                "Host: localhost:80\n"  "\r\n\r\n" );
      write(fd1,s,l=strlen(s));
      while( read(fd1, s, 16383) > 0 ){
        printf("Reply from server: /%s/\n",s);
        usleep(100000L);
      }
      close(fd1);
  }
}
```

12 .2.4 'IPv4 socket establishment'

kann man zB. mit folgendem 'C'-Code Macro implementieren:

```
#include <sys/socket.h>
#define SO_REUSEPORT 15

#define MAKE_LISTEN_PORT( fdLISTEN, IpPort ) {\
    struct sockaddr_in  s_address;              \
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    fdLISTEN = socket(AF_INET,SOCK_STREAM,0);           \
    s_address.sin_family      = AF_INET;                \
    s_address.sin_addr.s_addr = htonl(INADDR_ANY);      \
    s_address.sin_port        = htons(atoi(IpPort));    \
    int one=1;  \
    setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/ \
    if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){       \
       perror("bind failed"); exit(EXIT_FAILURE);       \
```

```
    }   \
    listen(fdLISTEN,5); \
  }

main(){int fd1;  MAKE_LISTEN_PORT( fd1, "55555" ); ... }
```

12 .2.5 serverseitiges Warten auf 'IPv4' Anfragen

ist mit folgendem 'C'-Code Macro zu erwirken:

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>

#define WAIT_FOR_REQUEST( fdCOMM, fdLISTEN, IpPort ) {\
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    printf("server: %s@%s waiting at port:%d\n", argv[0], hostName, atoi(IpPort));      \
    fdCOMM  = accept(fdLISTEN, NULL, NULL);     \
  }

main(){
  int  fdc,fd1;
  char rxByteArray[4096], outstr[1000];

  //  ... (Vorbereitungen) ...

  WAIT_FOR_REQUEST( fdc, fd1, "55555" );
  r1=read(fdCOMM, rxByteArray, sizeof(rxByteArray)-2 ));

  //  ... (Verarbeitung) ...

RueckAntwort:
  write(fdCOMM, outstr, strlen(outstr));
  close(fdCOMM);
}
```

12 .2.6 Bsp. f. einen HTTP Server

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#define IpPort  (argv[1])
#include "wsock.h"              //Macros MAKE_LISTEN_PORT, WAIT_FOR_REQUEST,

    /*MAIN:*/
int main(int argc,char **argv){
  int           fdPORT, fdCOMM, r1=0;
  char          rxdata[4096], outstr[4096];
  if(argc<2){ printf("usage: %s <PortNumber>",*argv); exit(0);}
  MAKE_LISTEN_PORT( fdPORT, IpPort );
  for(;;){
    WAIT_FOR_REQUEST( fdCOMM, fdPORT, IpPort );
    if(1 > (r1=read(fdCOMM, rxdata, sizeof(rxdata)-2 ))) break;
    MACH_ABSCHLUSS_NULL: rxdata[r1]=0;
    inhaltAnzeigeFunctionSieheHeaderfile(rxdata);
```

```
      /*antworten:*/
    snprintf( outstr, sizeof(outstr)-1,
        "HTTP/1.0 200 OK\r\nServer: RWS\r\nContent-Type: text/html\r\n\r\n"
        "<html><body bgcolor=tan><h1>RWS - kleiner Webserver</h1>\n"
        "<br>Nur zum Testen, ein kleiner text...\n"
        "<hr><small>(c)2005 richard weinberger</small></body></html>\r\n\r\n"
        );
    write(fdCOMM, outstr, strlen(outstr));
    close(fdCOMM);
    if(strstr(rxdata,"serverstop")) break; /*"geheim"-shutdown URLstring*/
  }
  shutdown(fdPORT,SHUT_RDWR);
  close(fdPORT);
  printf("\nserver \"%s\" terminated.\n",*argv);
}/*end of main*/
```

Listing 25: wsock.h

```
#ifndef WSOCK_H
#define WSOCK_H
/* File wsock.h  cfs@PdG,Pe2
 */
#include <sys/socket.h>
//#include <poll.h>
//#include <fcntl.h>
//#include <sys/types.h>
#define SO_REUSEPORT 15

#define MAKE_LISTEN_PORT( fdLISTEN, IpPort ) {\
    struct sockaddr_in  s_address;              \
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    fdLISTEN = socket(AF_INET,SOCK_STREAM,0);           \
    s_address.sin_family      = AF_INET;                \
    s_address.sin_addr.s_addr = htonl(INADDR_ANY);      \
    s_address.sin_port        = htons(atoi(IpPort));    \
    int one=1;  \
    setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/ \
    if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){       \
       perror("bind failed"); exit(EXIT_FAILURE);       \
    }   \
    listen(fdLISTEN,5); \
  }

#define WAIT_FOR_REQUEST( fdCOMM, fdLISTEN, IpPort ) {\
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    printf("server: %s@%s waiting at port:%d\n", argv[0], hostName, atoi(IpPort));      \
    fdCOMM  = accept(fdLISTEN, NULL, NULL);     \
  }

int inhaltAnzeigeFunctionSieheHeaderfile(char *s){
  printf("\n*--------new-message:------------*\n");
  for(;*s;s++){
    if(isprint(*s)) putchar(*s);
    else printf("(0x%02X)",*s);
    if('\n'==*s) putchar('\n');
  }
  printf("\n*--------message-end-------------*\n");
}
#endif
```

12 .2.7 Bsp. f. eine HTTP client Anfrage

```
/* File wclnt.c
 * Socket-Programmierung, cfs@GcF
```

```
 */
 #include <sys/types.h>
 #include <sys/socket.h>
 #include <stdio.h>
 #include <netinet/in.h>
 #include <arpa/inet.h>
 #include <unistd.h>
 #include <netdb.h>
 #define IpPort 80

int main(int argc,char **argv){
  int           fd1, result, l, n;
  char          ch='A', s1[4096], *s, hname[256];
  struct sockaddr_in    address;

/*** socket configurieren: ***/
  address.sin_family=AF_INET;
  address.sin_addr.s_addr = inet_addr("10.10.10.30");
  address.sin_port = htons(IpPort);     //(NetworkByteOrder)

  s=(char*)malloc(16384);
  *s=0;

  for(;;){
    printf("client \"%s\" @%s\n",*argv,(gethostname(hname,255),hname));
    fd1=socket(AF_INET,SOCK_STREAM,0);  /* == socket erzeugen und verbinden **/
    if((result=connect(fd1,(struct sockaddr *) &address,sizeof(address))) < 0){
      perror("cant connect\n");
      break;
    }else{
      sprintf(s,
        "GET /SMUE/srcvw.php?fn=weliza/weliza1.c&nr=123 HTTP1.0/\n"
        "Host: localhost:80\n"
        "\r\n\r\n"
      );
      write(fd1,s,l=strlen(s));
      if(*s == 'C') break;
      while( read(fd1,s,16383) > 0 ){
        printf("Reply from server: /%s/\n",s);
        usleep(100000L);
      }
      close(fd1);
    }
    break;
  }
  free(s);
  exit(0);
}
```

12 .2.8 Bsp. f. serverseitige openSSL encryption

```
/* File srv03ssl.c , cfs@Pe2
 * HTTPS-Webserver
 *  ::#define- macros in separate header file
 * shows message from Web-Browser
 * +sends very simple reply
 * compile: gcc srv03ssl.c -o srvs03
 * run    : ./srvs03 <portNr>  (zB "./srvs03 4433")
 * stop   : Strg+C   oder "serverstop" im URLstring
 */
/* Certificate Generation:
#gen ca private key
 mkdir ca
 openssl genrsa -out ca/privkey.pem

#create server cert request
 mkdir ca/private
```

```
 openssl req -x509 -days 7200 -newkey rsa:1024 -keyout ca/private/ca.key -out ca/ca.crt

#create server cert request
 mkdir -p server/private
 openssl genrsa -out server/private/server.key 1024
 openssl req -new -key server/private/server.key -out server/server.csr

#create client cert request
 mkdir -p client/private
 openssl genrsa -out client/private/client.key 1024
 openssl req -new -key client/private/client.key -out client/client.csr

#sign certs:
 openssl x509 -req -days 7200 -in server/server.csr -CA ca/ca.crt -CAkey ca/private/ca.key -
     CAcreateserial -out server/server.crt
 openssl x509 -req -days 7200 -in client/client.csr -CA ca/ca.crt -CAkey ca/private/ca.key -CAserial
     ca/ca.srl -out client/client.crt

#check, using openssl a Server-Fake:
 openssl s_server -CAfile ca/ca.crt -cert server/server.crt -key server/private/server.key -Verify 1
 */
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <err.h>
//#include <stdarg.h>

#include <openssl/bio.h>
#include <openssl/ssl.h>
#include <openssl/err.h>
#include <openssl/pem.h>
#include <openssl/x509.h>
#include <openssl/x509_vfy.h>

#define SSL_CA_CRT      "./cert/ca/ca.crt"
#define SSL_SERVER_CRT  "./cert/server/server.crt"
#define SSL_SERVER_KEY  "./cert/server/private/server.pem"
#define fatalx(a)       {perror(a);exit(-1);}

#define IpPort  (argv[1])
//statt #include "wsock.h": -> hierherkopiert:
  /* File wsock.h  cfs@PdG,Pe2
   */
//#include <sys/socket.h>
#define MAKE_LISTEN_PORT( fdLISTEN, IpPort ) {\
    struct sockaddr_in  s_address;              \
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    fdLISTEN = socket(AF_INET,SOCK_STREAM,0);           \
    s_address.sin_family      = AF_INET;                \
    s_address.sin_addr.s_addr = htonl(INADDR_ANY);      \
    s_address.sin_port        = htons(atoi(IpPort));    \
    int one=1;  \
    setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/ \
    if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){       \
       perror("bind failed"); exit(EXIT_FAILURE);       \
    }   \
    listen(fdLISTEN,5); \
  }

#define WAIT_FOR_REQUEST( fdCOMM, fdLISTEN, IpPort ) {\
    char        hostName[256];                  \
    gethostname(hostName, 255);                 \
    printf("server: %s@%s waiting at port:%d\n", argv[0], hostName, atoi(IpPort));      \
    fdCOMM  = accept(fdLISTEN, NULL, NULL);     \
```

```
  }

int inhaltAnzeigeFunctionSieheHeaderfile(char *s){
  printf("\n*--------new-message:------------* Len=%d\n", strlen(s));
  for(;*s;s++){
    if(isprint(*s)) putchar(*s);
    else printf("(0x%02X)",*s);
    if('\n'==*s) putchar('\n');
  }
  printf("\n*--------message-end-------------*\n");
}
/*---End-Of-wsock.h---*/




  /**********/
   /**MAIN**/
    /*    */
     /*  */
      /**/
  /**********/
   /**MAIN**/
    /*    */
     /*  */
      /**/
int main(int argc,char **argv){ if(argc<2){printf("usage: %s <port>",*argv); exit(0);}
  int           fdLISTEN, fdCOMM, r1=0;
  char          rxdata[4096], outstr[4096];
  SSL_CTX       *ctx1;
  SSL           *ssl1;
  BIO           *sbio;


    /*SSL preparation:*/
  SSL_load_error_strings();
  OpenSSL_add_ssl_algorithms();
  if(NULL==(ctx1 = SSL_CTX_new(SSLv23_server_method())))               fatalx("ctx");
  if(!SSL_CTX_load_verify_locations(ctx1, SSL_CA_CRT, NULL))           fatalx("verify");
  SSL_CTX_set_client_CA_list(ctx1, SSL_load_client_CA_file(SSL_CA_CRT));
  if(!SSL_CTX_use_certificate_file(ctx1, SSL_SERVER_CRT, SSL_FILETYPE_PEM))  fatalx("cert");
  if(!SSL_CTX_use_PrivateKey_file(ctx1, SSL_SERVER_KEY, SSL_FILETYPE_PEM))   fatalx("key");
  if(!SSL_CTX_check_private_key(ctx1))  fatalx("cert/key");
  SSL_CTX_set_mode(ctx1, SSL_MODE_AUTO_RETRY);
  //SSL_CTX_set_verify(ctx1, SSL_VERIFY_PEER | SSL_VERIFY_FAIL_IF_NO_PEER_CERT, NULL);//
      willClientCert
  SSL_CTX_set_verify(ctx1, SSL_VERIFY_NONE, NULL/*callback*/);
  SSL_CTX_set_verify_depth(ctx1, 1);

    /*setup socket: socket()/bind()/listen() */
  MAKE_LISTEN_PORT( fdLISTEN, IpPort );

  for(;;){
    WAIT_FOR_REQUEST( fdCOMM, fdLISTEN, IpPort );
    //if((fdCOMM = accept(fdLISTEN, 0, 0)) == -1)  err(EX_OSERR, "accept");
    if(fdCOMM < 0){perror("TCP accept");exit(-2);}//err(EX_OSERR, "accept");
    //if(fdCOMM < 0) err(EX_OSERR, "accept");

    sbio = BIO_new_socket(fdCOMM, BIO_NOCLOSE);
    ssl1 = SSL_new(ctx1);
    SSL_set_bio(ssl1, sbio, sbio);

      //if(1 > (r1=read(fdCOMM, rxdata, sizeof(rxdata)-2 ))) break;
    if((r1 = SSL_accept(ssl1)) == -1)  warn("SSL_accept");
    printf("SSL_accept:%d(%s) err=%d\n",
        r1, ((0<r1)?"ok":((0>r1)?"fatal":"shut")), SSL_get_error(ssl1,r1) );
        //err(SSL_get_error(ssl1,r1),NULL);
        //if(1>r1)break;
```

```
    //do{
      if(0 > (r1=SSL_read(ssl1, rxdata, sizeof(rxdata)-2 ))){
        perror("**ERROR: SSL_read()<0!");
        break;
      }
    //}while(0==r1);
    MACH_ABSCHLUSS_NULL: rxdata[r1]=0;
    inhaltAnzeigeFunctionSieheHeaderfile(rxdata);
      /*antworten:*/
    snprintf( outstr, sizeof(outstr)-1,
        "HTTP/1.0 200 OK\r\nServer: RWS\r\nContent-Type: text/html\r\n\r\n"
        "<html><body bgcolor=tan><h1>RwXh's - kleiner https://Webserver</h1>\n"
        "<br>SSL/TLS SSLv2+SSLv3+TLSv1 w/o client cert request\n"
        "<hr><small>(c)2005/rw 2015/xh</small></body></html>\r\n\r\n"
        );
    r1=SSL_write(ssl1, outstr, strlen(outstr));
printf("done: SSL_write(): rc=%d.\n", r1);
    r1=SSL_shutdown(ssl1);
printf("done: SSL_shutdown(): rc=%d(%s).\n", r1, ((0<r1)?"ok":((0>r1)?"fatal":"pending")) );
    r1=close(fdCOMM);
printf("done: close(fdCOMM): rc=%d(%s)\n", r1, (0==r1)?"ok":"error");
    BIO_free_all(sbio);
printf("done: BIO_free_all().\n");
    SSL_CTX_free(ctx1);
printf("done: SSL_CTX_free().\n");
    if(strstr(rxdata,"serverstop")) break; /*"geheim"-shutdown URLstring*/
  }
  shutdown(fdLISTEN,SHUT_RDWR);
  close(fdLISTEN);
  printf("\nserver \"%s\" terminated.\n",*argv);

}/*end of main*/
```

12 .2.9 Bsp. f. clientseitige openSSL encryption

```
/* ssl04.c      cfs@Pe1,Pe6
 * home made ssl client, for learning purpose; from 'openssl in C.c'
 *
 * compile : 'gcc ssl04.c -lssl -oss4'
 * run     : './ss4  10.10.63.61  443'
 *
 *
*
*from file:
* A little while ago, I was working on a client/server communication module,
* and I wanted it to be secure.
* Looking at the documentation I could find,
* I quickly figured out that it wouldn't necessarily be easy to do.
*
* Amongst the first issues are the validity of the server's certificate,
* which, as I didn't want to battle with this, I decided to skip
*
* The example below shows how to connect and send/receive data on an SSL-encrypted socket
*
* To build this example program using gcc:
*   gcc -Wall -lssl -lcrypto -o ssl-demo ssl-demo.c
* Have fun with SSL!
*
 *not needed:
 * //#include <sys/socket.h>
 * //#include <sys/types.h>
 * //#include <netinet/in.h>
 * //#include <unistd.h>
 * //#include <errno.h>
 * //#include <openssl/rand.h>
 * //#include <openssl/err.h>
 * //#include <string.h>
```

```
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>     //4 getopt(),optarg,
#include <netdb.h>
#include <openssl/ssl.h>

// Simple structure to keep track of the handle, and
// of what needs to be freed later.
typedef struct {
    int         sockfd;
    SSL         *SSLssl1;
    SSL_CTX     *SSLctx1;
} myConnType;

  //For this example, we'll be testing on openssl.org
//#define SERVER_URL  "www.openssl.org"
//#define SERVER_URL  "www.google.at"
#define SERVER_URL  "tr1"
#define SERVER_PORT "4433"
#define ERRprint(n) { ERR_print_errors_fp(stderr); return(n);}



    /* Macro Code for command line parameter processing */
#define COMMANDLINE_PARAMETER_PROCESSING()      { \
  int opt;                                          \
  while ((opt = getopt(argc, argv, "hu:p:")) != -1) {   \
    switch(opt){                                    \
    case 'u':   strncpy(urlstr, optarg, sizeof(urlstr)-1); break;       \
    case 'p':   strncpy(portstr, optarg, sizeof(portstr)-1); break;     \
    case 'h':   \
    default:    printf("** usage: %s [-h] [-u URL] [-p port]\n",*argv);\
                exit(0); break; \
    }   \
  }     \
}



int sslConnect( myConnType *cc, char *srvName, unsigned int portint){
  printf("ask '%s'\n",srvName);

  //1.Establish regular tcp connection
  int                   erv;
  struct hostent        *host;
  struct sockaddr_in    clntsock;
  host        = gethostbyname(srvName);
printf("connecting %s(=%8X):%d...\n", srvName, *host->h_addr, portint);
  cc->sockfd = socket(AF_INET, SOCK_STREAM, 0);
  if(0>cc->sockfd){ perror ("Socket"); cc->sockfd=-1; }else{
    clntsock.sin_family = AF_INET;
    clntsock.sin_port   = htons(portint);
    clntsock.sin_addr   = *((struct in_addr *) host->h_addr);
    bzero( &(clntsock.sin_zero), 8);
    if(-1== connect( cc->sockfd, (struct sockaddr*) &clntsock, sizeof(struct sockaddr))){
      perror ("**Error: TCP connect().");  return(cc->sockfd= -1);
    }
  }

  //2.Add SSL
  if(0<=cc->sockfd){
    SSL_load_error_strings ();  //Register errorstrings for libcrypto & libssl
    SSL_library_init ();        //Register available ciphers+digests
    if(NULL == (cc->SSLctx1 = SSL_CTX_new (SSLv23_client_method() )))  ERRprint(-10);
    if(NULL == (cc->SSLssl1 = SSL_new(cc->SSLctx1)))  ERRprint(-11);
    if(! SSL_set_fd(cc->SSLssl1, cc->sockfd))  ERRprint(-12);
    if(1 != SSL_connect (cc->SSLssl1))  ERRprint(-13); //Initiate SSL handshake
```

```
  }else{  printf("me got bad sockfd.\n"); return(-3);}

    /*Ueberpruefung des Server-Certifikats: fehlt! */
  CHECK_OF_SERVER_CERTIFICATE_VALIDITY_is_MISSING_here:
  return(0);
}


void sslDestroy(myConnType *cc){
  if(1>cc->sockfd)  close(cc->sockfd);
  if(cc->SSLssl1){
    SSL_shutdown(cc->SSLssl1);
    SSL_free(cc->SSLssl1);
  }
  if(cc->SSLctx1) SSL_CTX_free(cc->SSLctx1);
}




void sslWrite (myConnType *cc, char *text){  // Write text to the connection
  if(cc) SSL_write( cc->SSLssl1, text, strlen (text));
}

  //Read much available text from connection
char *sslRead(myConnType *cc){
  #define       BLKsz  1024
  #define       BLKnum 10
  static char   rxbuf[BLKnum*BLKsz];  //muss persistent!
  int           rxcnt, count;
  if(cc) for( count=0; count<BLKnum; count++ ){
    rxcnt = SSL_read(cc->SSLssl1, rxbuf+count*BLKsz, BLKsz);
    if(rxcnt < BLKsz){
      rxbuf[count*BLKsz + rxcnt] = '\0';
      break;
    }
  }
  return(rxbuf);
}



   /***********************/
    /*Very basic M.A.I.N.*/
     /*we send "GET /" */
      /*  and print  */
        /*response.*/
          /*     */
            /**/
int main (int argc, char **argv){ if(argc<1){printf("usage: %s <url>\n");exit(0);}
  char          *re1, urlstr[1023]=SERVER_URL, portstr[31]=SERVER_PORT;
  int           rv1;
  unsigned int  portint;
  myConnType    cc1={ 0, NULL, NULL};

  COMMANDLINE_PARAMETER_PROCESSING();
  portint=atoi(portstr);
  if(0>(rv1=sslConnect(&cc1, urlstr, portint)) ){
    printf("Connection is hin:%d/[%s:%d]\n",rv1, urlstr,portint );
  }else{
    sslWrite(  &cc1, "GET /\r\n\r\n");
    re1= sslRead(&cc1); printf("%s\n", re1);
  }
  sslDestroy(&cc1);
  return(0);
}
```

12 .2.10 Benchmark Timing Example

Listing 26: Benchmark Timing Example

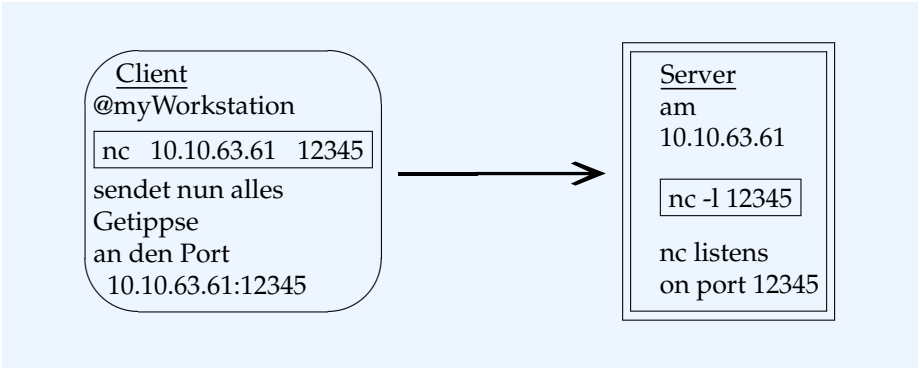
```
/* File httpRqTmg01.c, xh@QlF
 * Socket-Programmierung +GetTime => Benchmark
 * compile: 'gcc httpRqTmg01.c -o hRT1 -lrt'
 * run:     './hRT1'
 * stop:    (self halting)
 */

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

#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netdb.h>

#define IpPort  ((argc>1)?(argv[1]):"127.0.0.1")

int main(int argc,char *argv[]){
  int   fd1;
  char  anfrage[1024],
        antwort[4096],
        *errmsg;
  struct sockaddr_in    address;
  struct timespec       wtime = {0L, 50000000L}, //100ms
                        t1, t2;

  if((argc>1)?argv[1][1]=='h':0){
    printf("usage: '%s [IpAdresse] [p]'\n", argv[0] );
    exit(EXIT_SUCCESS);
  }

  address.sin_family=AF_INET;
  address.sin_addr.s_addr = inet_addr(IpPort);
  address.sin_port = htons(80); //(NetworkByteOrder)
  fd1 = socket(AF_INET, SOCK_STREAM, 0 );

  if(0 > connect(fd1,(struct sockaddr *) &address,sizeof(address))){
    asprintf(&errmsg,"cant connect '%s'", IpPort);
    perror(errmsg);
    free(errmsg);
    exit(EXIT_FAILURE);
  }else{
    sprintf(anfrage,
        "GET /wahuschi.html HTTP1.0/\nHost: localhost:80\n"
        "\r\n\r\n");
    clock_gettime(CLOCK_MONOTONIC, &t1 );
    write(fd1, anfrage, strlen(anfrage) );
    while( read(fd1, antwort, 4095) > 0 ){
      clock_gettime(CLOCK_MONOTONIC, &t2 );
      printf("reply time from t1=%ld.%09ld to t2=%ld.%09ld diff=%f[ms] \n",
        t1.tv_sec, t1.tv_nsec, t2.tv_sec, t2.tv_nsec,
        (t2.tv_sec-t1.tv_sec)*1000.0 + (t2.tv_nsec-t1.tv_nsec)/1000000.0 );
      if((argc>2)?(argv[2][0]=='p'):0){  //should print this?
        printf("Reply from server: '%s'\n", antwort);
      }
      nanosleep( &wtime, NULL);
    }
    close(fd1);
  }
}
```

12 .2.11 IoT Internet-of-Things Device

→ s. Kap.12 , Seite 295

12 .3 supersimpelServerMit'netcat'

Das Utility *'netcat'* bzw. *'nc'* kopiert zwischen LAN-Port und Konsole



12 .3.1 alittlemore

Mit einem Shellscript der Art

Listing 27: service.sh

```
#!/bin/sh
while((1)); do
  echo -n -e "give command [adhilnrt^D]:";
  read x;
  if [ $x == "d" ]; then
    echo "---> date=$(cat /sys/class/rtc/rtc0/date)";
  elif [ $x == "t" ]; then
    echo "---> time=$(cat /sys/class/rtc/rtc0/time)";
  elif [ $x == "a" ]; then
    echo "---> MACaddress=$(cat /sys/class/net/eth0/address)";
  elif [ $x == "h" ]; then
    echo "---> HOSTname: $(cat /etc/hostname)";
  elif [ $x == "i" ]; then
    echo "---> IP: $(ip a)";
  elif [ $x == "l" ]; then
    echo "---> IP: $(ip link)";
  elif [ $x == "n" ]; then
    echo "---> IP: $(ip neigh)";
  elif [ $x == "r" ]; then
    echo "---> IP: $(ip rou)";
  else
  echo "------------------------------>>> ur command was: /$x/";
  fi
done
```

und dem Haupt-Script

Listing 28: sv1.sh

```
#!/bin/sh
mkfifo /tmp/p11111
while((1)); do
cat /tmp/p11111 |    \
  ./service.sh  2>&1  |      \
```

```
  nc -k -l 11111 >    \
  /tmp/p11111
done
```

entsteht ein stinksimples IoT Device, das mehrere Abfragekommandos versteht und entsprechend antwortet:

Auf Hostrechner xy startest Du

`./sv1.sh`

Auf der Clientworkstation kannst nun mit

`nc xy 11111`

die Kommandos ausprobieren.

(den Client beendet man mit Strg+D und den Server mit Strg+C)

Mit zusätzlicher Absicherung durch einen ssh Tunnel ist das gar nicht mehr so schlecht. Sicherheitsbedenken gibt es aber gegenüber allen Shellscripten. Die Shells sind teilweise unglaublich uralt und enthalten noch schauderhafte security holes. Dies würde trotz ssh einem *'trusted client'* die unerwünschte Übernahme (hitch hiking) des Rechners ermöglichen.

12 .3.2 file contents reporting IoT Server (risky!)

Listing 29: one file reading server

```
/* File IoTreadfile1.c , cfs@RcJ
 * +sends very simple reply
 * compile: gcc IoTreadfile1.c -o IoTrf1
 * run    : ./IoTrf1
 * stop   : Strg+C
 */
#include <stdlib.h>     /* exit() */
#include <stdio.h>      /* printf() */
#include <string.h>     /* strlen() */
#include <fcntl.h>      /* O_RDONLY */
#include <arpa/inet.h>  /* SOCK_STREAM ...*/
#define IpPort  (argv[1])
#define INFILE  (argc>2 ? argv[2]:"antwort1")
//#define SO_REUSEPORT 15

    /*MAIN:*/
int main(int argc,char **argv){
  int           fdLISTEN, fdCOMM, r1=0, fdFILE;
  char          anfr[4096], antw[4096];
  struct sockaddr_in    s_address;

  if(argc<2){
    printf("usage: %s <PortNumber> <InputFileName>",*argv);
    exit(EXIT_FAILURE);
  }

    /*MAKE_LISTEN_PORT( fdPORT, IpPort );*/
  fdLISTEN = socket(AF_INET,SOCK_STREAM,0);
  s_address.sin_family      = AF_INET;
  s_address.sin_addr.s_addr = htonl(INADDR_ANY);
  s_address.sin_port        = htons(atoi(IpPort));
  int one=1;
  setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/

  if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){
     perror("bind failed"); exit(EXIT_FAILURE);
  }
  listen(fdLISTEN,5);

  for(;;){
      /*WAIT_FOR_REQUEST( fdCOMM, fdPORT, IpPort ): */
    fdCOMM  = accept(fdLISTEN, NULL, NULL);

      /*antworten:*/
    if(0> (fdFILE = open(INFILE, O_RDONLY))){
      perror(INFILE);
    }
    if(0> (r1=read(fdFILE, antw, sizeof(antw)-2))){
      perror(INFILE);
    }
    antw[r1]=0;
    close(fdFILE);
    write(fdCOMM, antw, strlen(antw));
    close(fdCOMM);
  }
  shutdown(fdLISTEN, SHUT_RDWR);
  close(fdLISTEN);
}/*end of main*/
```

12 .3.3 any file reporting IoT Server (risky!)

Listing 30: any file reading server

```
/* File IoTreadfile2.c , cfs@RcJ
 * +sends file contents as reply
 * compile: gcc IoTreadfile2.c -o IoTrf2
 * run    : ./IoTrf2
 * stop   : Strg+C
 */
#include <stdlib.h>     /* exit() */
#include <stdio.h>      /* printf() */
#include <string.h>     /* strlen() */
#include <fcntl.h>      /* O_RDONLY */
#include <arpa/inet.h>  /* SOCK_STREAM ...*/
#define IpPort  (argv[1])
//#define SO_REUSEPORT 15

    /*MAIN:*/
int main(int argc,char **argv){
  int           fdLISTEN, fdCOMM, r1=0, fdFILE;
  char          anfr[4096], antw[4096];
  struct sockaddr_in    s_address;

  if(argc<2){
    printf("usage: %s <PortNumber>",*argv);
    exit(EXIT_FAILURE);
  }

    /*MAKE_LISTEN_PORT( fdPORT, IpPort );*/
  fdLISTEN = socket(AF_INET,SOCK_STREAM,0);
  s_address.sin_family      = AF_INET;
  s_address.sin_addr.s_addr = htonl(INADDR_ANY);
  s_address.sin_port        = htons(atoi(IpPort));
  int one=1;
  setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/

  if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){
     perror("bind failed"); exit(EXIT_FAILURE);
  }
  listen(fdLISTEN,5);

  for(;;){
      /*WAIT_FOR_REQUEST( fdCOMM, fdPORT, IpPort ): */
    fdCOMM  = accept(fdLISTEN, NULL, NULL);
    if(-1< (r1=read(fdCOMM, anfr, sizeof(anfr)-2))){
      anfr[r1]=0;
      anfr[strcspn(anfr,"\n\r")]=0; /*eliminate CR LF...*/
        /*antworten:*/
      if(0> (fdFILE = open(anfr, O_RDONLY))){
        perror(anfr);
      }
      if(0> (r1=read(fdFILE, antw, sizeof(antw)-2))){
        perror(anfr);
      }
      antw[r1]=0;
      close(fdFILE);
      write(fdCOMM, antw, strlen(antw));
      close(fdCOMM);
    }
  }
  shutdown(fdLISTEN, SHUT_RDWR);
  close(fdLISTEN);
}/*end of main*/
```

12 .3.4 command executing IoT Server (extreme risky!)

Listing 31: command executing server

```
/* File IoTreadfile3.c , cfs@RcJ
 * +sends command as reply
 * compile: gcc IoTreadfile3.c -o IoTrf3
 * run    : ./IoTrf3
 * stop   : Strg+C
 */
#include <stdlib.h>     /* exit() */
#include <stdio.h>      /* printf() */
#include <string.h>     /* strlen() */
#include <fcntl.h>      /* O_RDONLY */
#include <arpa/inet.h>  /* SOCK_STREAM ...*/
#define IpPort  (argv[1])
//#define SO_REUSEPORT 15

    /*MAIN:*/
int main(int argc,char **argv){
  int           fdLISTEN, fdCOMM, r1=0, fdFILE;
  char          anfr[4096], antw[4096];
  struct sockaddr_in    s_address;

  if(argc<2){
    printf("usage: %s <PortNumber>",*argv);
    exit(EXIT_FAILURE);
  }

    /*MAKE_LISTEN_PORT( fdPORT, IpPort );*/
  fdLISTEN = socket(AF_INET,SOCK_STREAM,0);
  s_address.sin_family      = AF_INET;
  s_address.sin_addr.s_addr = htonl(INADDR_ANY);
  s_address.sin_port        = htons(atoi(IpPort));
  int one=1;
  setsockopt(fdLISTEN, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one); /*+SO_REUSEPORT*/

  if(bind(fdLISTEN,(struct sockaddr *) &s_address, sizeof(s_address))){
     perror("bind failed"); exit(EXIT_FAILURE);
  }
  listen(fdLISTEN,5);

  for(;;){
      /*WAIT_FOR_REQUEST( fdCOMM, fdPORT, IpPort ): */
    fdCOMM  = accept(fdLISTEN, NULL, NULL);
    if(-1< (r1=read(fdCOMM, anfr, sizeof(anfr)-2))){
      anfr[r1]=0;
      anfr[strcspn(anfr,"\n\r")]=0; /*eliminate CR LF...*/

        /*antworten:*/
      system(strncat(anfr,"> aw3", sizeof(anfr)));
      if(0> (fdFILE = open("aw3", O_RDONLY))){
        perror("aw3");
      }
      if(0> (r1=read(fdFILE, antw, sizeof(antw)-2))){
        perror(anfr);
      }
      antw[r1]=0;
      close(fdFILE);
      write(fdCOMM, antw, strlen(antw));
      close(fdCOMM);
    }
  }
  shutdown(fdLISTEN, SHUT_RDWR);
  close(fdLISTEN);
```

```
}/*end of main*/
```

12 .3.5 mit ssh - secure shell

12 .4 web based IoT sensor

12 .4.1 Messwert Server C Code

Ganz einfaches Beispiel für einen http Server,
der (hier fingierte zufzi) Messwerte antwortet:

```
/* File tmpSens1.c, XH, 18Mar15 Fsst3a
 * beantwortet Anfragen mit (simuliertem) Temperaturwert
 * compile: gcc tmpSens1.c -o ts1
 * run:     ./ts1  10101
 * stop:    Strg+C
 * Bedienung: im Firefox mit "http://10.10.63.61:10101/" abrufen */

 #include <stdio.h>
 #include <stdlib.h>
 #include <string.h>
 #include <sys/types.h>
 #include <sys/socket.h>
 #include <netinet/in.h>
 #include <arpa/inet.h>
 #include <unistd.h>
 #include <fcntl.h>

#define IpPort  (argv[1])
int peep(char*);

int main(int argc,char **argv){
  int           fds,fdc,lenc,r1=0;
  char          buf[4096], buf2[4096],
                hname[256],
                ADCval[10];
  double        TempKorrFaktor=10.72;
  struct
  sockaddr_in   s_address, c_address;

  if(argc<2){ printf("usage: %s <PortNumber>",*argv); exit(0); }

  gethostname(hname,255);
  fds=socket(AF_INET,SOCK_STREAM,0);
  s_address.sin_family=AF_INET;
  s_address.sin_addr.s_addr = htonl(INADDR_ANY);
  s_address.sin_port = htons(atoi(IpPort));
  int one=1; setsockopt(fds, SOL_SOCKET, SO_REUSEADDR, &one, sizeof one);
  if(bind(fds,(struct sockaddr *) &s_address, sizeof(s_address))){
   perror("bind failed"); abort();
  }
  listen(fds,5);
  for(;;){
    //printf("server \"%s\" @ \"%s\" waiting\n",*argv,hname);
    lenc=sizeof(c_address);
    fdc=accept(fds,(struct sockaddr *) &c_address, &lenc);
    ADCfd=open("adc0", O_RDONLY);
    read(ADCfd, ADCval, 5);
    close(ADCfd);
    sprintf(buf2,
      "HTTP/1.0 200 OK\r\nServer: RWS\r\nContent-Type: text/html\r\n\r\n"
      "<html><body>%dK</body></html>\r\n",
      atoi(ADCval)/TempKorrFaktor );
    write(fdc, buf2, strlen(buf2));
    close(fdc);
```

```
  }
  shutdown(fds,SHUT_RDWR);
  close(fds);
}
```

12 .4.2 Messwert 'Prozessvisualisierung' HTML Code

Ein sehr stark simplifiziertes Anfangsbeispiel für die Abfrage mehrerer 'IoT' IP Adressen/ Ports mit http- Rückmeldungen:

```
<html><head>
<script>
  window.setInterval( "reloader()", 3000 );
  function reloader(){
    window.location.reload();
  }
</script></head>
<frameset  cols="100,100,100,100,100,100,*" rows="100,100,100,*">
 <frame src=http://10.10.63.61:11999/></frame>
 <frame src=http://10.10.63.61:11998/></frame>
 <frame src=http://10.10.63.61:11997/></frame>
 <frame src=http://10.10.63.245:11999/></frame>
 <frame src=http://10.10.63.61:10201/></frame>
 <frame src=http://10.10.63.61:10202/></frame>
 <noframes>
   <body>
     reloading every 3000ms
   </body>
 </noframes>
</frameset>
</html>
```

12 .5 ssh Fernstart mehrerer Messwert Server

SSH kann auf der Zielmaschine (Host) einen Befehl (zB. Programmstart) ausführen. [10] Hier sind mehrere solcher Fernstarts in ein shellscript zusammengefasst:

```
#!/bin/bash
ssh -f -n dings@10.10.63.245 /home/dings/c/IoT/ts1 11999
ssh -f -n dings@10.10.63.61 "/home/dings/public_html/Fst3a14/IoT/ts1 11999"
ssh -f -n dings@10.10.63.61 "/home/dings/public_html/Fst3a14/IoT/ts2 11998"
ssh -f -n dings@10.10.63.61 "/home/dings/public_html/Fst3a14/IoT/ts3 11997"
```

(Passwort- Eingabe- Problem muasch ggf. lösen)

12 .6 ssh Fernstart plus encrypted port forwarding

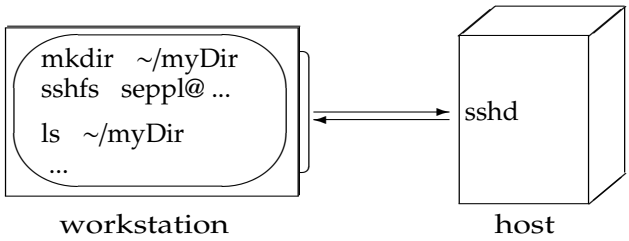
```
       /* ssh macht (auch) 'port forwarding'
          d.h. der lokale Port (hier 1234)
          wird zum Host-Port (hier 6666) durch'getunnelt',
          (ein User ist wegen Passwort anzugeben), mit: */

  ssh -f -L 1234:localhost:6666 ich@meinhost.mydomain

       /* zudem kannma am Host gleich ein Programm starten lassen
          (sinnvollerweise eines, das auf dem geforwardeten Port -hier 6666-
          kommuniziert): */

  ssh -f -L 1234:localhost:6666 ich@meinhost.mydomain "/meinPfad/meinProgramm"

       /* ausprobiertes Beispiel:
          (dann kann man 'http://localhost:1234' anbrausen) */
```

[10] wenn man den Befehl — wie üblich — weglasst, startez a Kommando- *'shell'*

```
ssh -f -L 1234:localhost:6666 cfs@tr3 "echo \"hallo\" | netcat -l 6666"


      /* encrypted 'Datums-Server';  local:1234 <---> tr3:6666
         -->
         Kommando  'netcat localhost 1234'
         liefert:  'Sun Feb 21 14:11:54 UTC 2016'
      */
ssh -f -L 1234:localhost:6666 cfs@tr3 "while((1)); do date | netcat -l 6666; done"
```

12 .7 'sshfs' secure shell (remote) file system

'sshfs' 'holt' (mounts) ein entferntes (remote) Verzeichnis (directory) in den lokalen Dateibaum (etwa wie eine 'Netzlaufwerk', aber hier muss nix vorher 'freigegeben' werden — die Existenz des 'user' reicht).



Die Datenübertragung erfolgt *chiffriert (encrypted)* mittels *'ssh'* Protokoll. Über die Angabe von *username* und *password* erfolgt eine Authorisierungsprüfung (sodass nicht jede 'Pfeife' Zugriff hat)
Man schreibt

```
sshfs  user@machine:DirectoryPath  localMountPoint
```

zB.

```
mkdir  ~/myDir
ssh  seppl@10.10.63.245:public_html/glump  ~/myDir
```

→ Password: ('seppl's Passwort am '10.10.63.245')

sodann ist unter *'~/myDir'* der Inhalt des Verzeichnisses '/home/seppl/public_html/glump' des users 'seppl' am Rechner 10.10.63.245 verfügbar:

```
ls -l  ~/myDir
```

Wo 'seppl' Schreibrechte hat, hat er sie auch unter ~/myDir.

Aufgehoben wird dieser 'mount' mit

```
fusermount  -u  ~/myDir
```

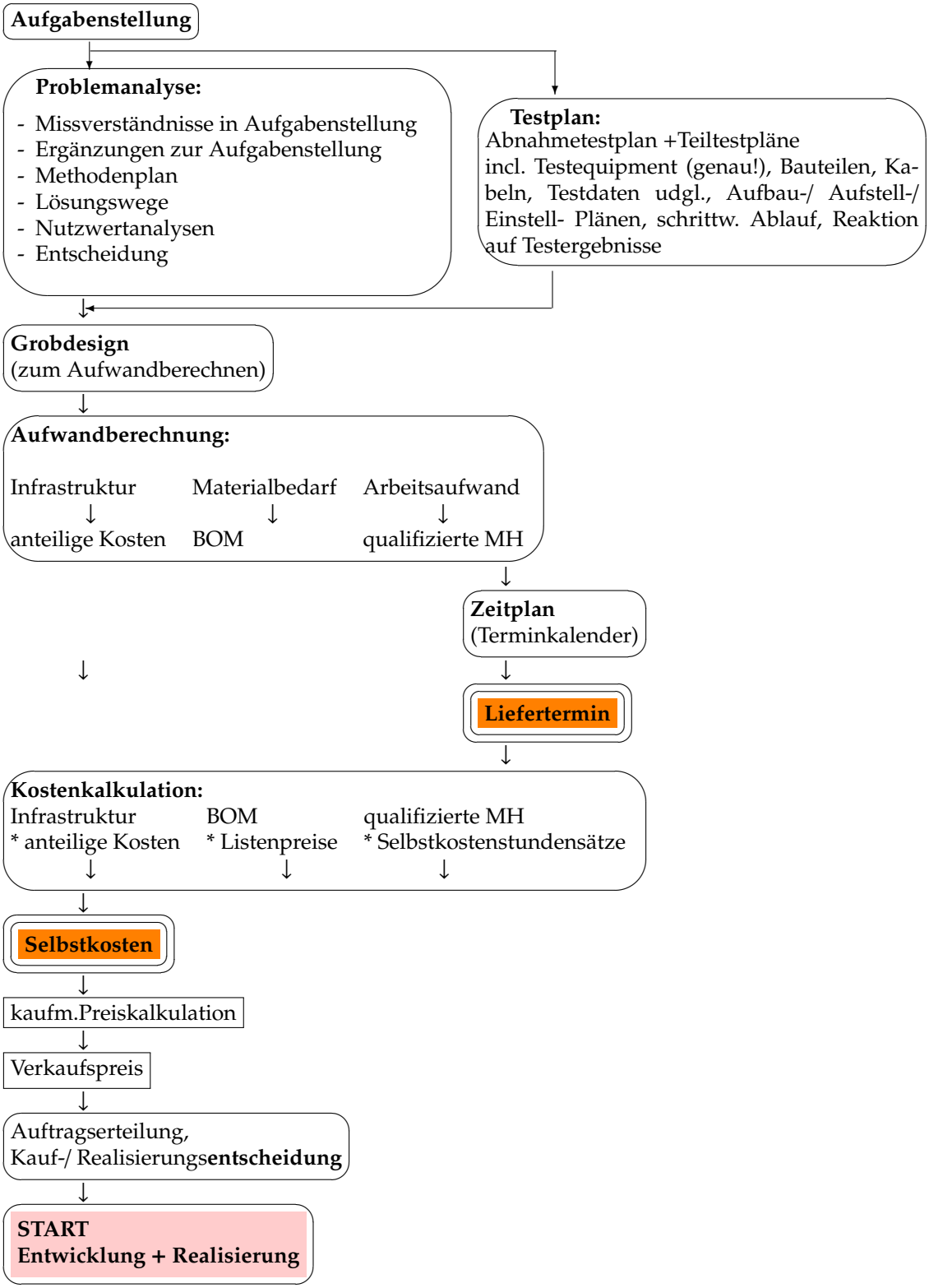
aus der manpage 'man 1 sshfs':

Listing 32: man 1 sshfs

```
SSHFS(1)                        User Commands                SSHFS(1)
SSHFS version 2.0 April 2008

NAME    SSHFS - filesystem client based on ssh
SYNOPSIS
   mounting:
       sshfs [user@]host:[dir] mountpoint [options]

   unmounting:
       fusermount -u mountpoint

DESCRIPTION
       SSHFS  (Secure  SHell  FileSystem)  is a file system for Linux (and other operating
```

```
systems with a FUSE implementation, such as Mac OS X or FreeBSD) capable of operat-
ing on files on a remote computer using just a secure shell login on the remote
computer. On the local computer where the SSHFS is mounted, the implementation
makes use of the FUSE (Filesystem in Userspace) kernel module. The practical effect
of this is that the end user can seamlessly interact with remote files being
securely served over SSH just as if they were local files on his/her computer. On
the remote computer the SFTP subsystem of SSH is used.
 [...]
```

13 Projektmanagement Übersicht



14 Projektmanagement U-Mitschrift

```
01Dez'17: today's topic:
se Projektmanagement:

Kunde:
  a) was kostet es?
  b) wann ist es fertig?

Was ist ein "Projekt"
  (von lat. 'pro'    = voraus,voran
            'jacere' = werfen)
  --klar definiertes Ziel (Produkt)
  --Ressourcenlimit (Geld)
  --Zeitlimit (Liefertermin)
  --Mindestkomplexitaet
  --Innovations-Anteil
  -->so, dass PLANUNG notwendig ist!
              -----------------
Kurzfassung: Ein Vorhaben ist dann
  ein Projekt, wenn man es PLANEN muss.


  Aufgabenstellung
        !
        v
  Analyse der Aufgabenstellung (Problemanalyse)
  (Fehlern, Missverstaendnisse, Ergaenzungen)
        !
        v
  Kreativ-Methoden (Brainstorm,Delphi,..128)
        v
  mehrere/viele Loesungswege
        v
  Nutzwertanalyse(n) (=Punktevergabe)
        v
  Entscheidung
        !
        v
  Grobdesign
  (so detailliert, dass man jeden
  Aufwand schaetzen kann)
        !
        v          Stundensaetze
  Aufwandschaetzung     !
             !          !
             !          v
             +--> Personalkosten
                  +Material(BOM)
                  +Infrastruktur
  Zeitplan                  !
     !   !                  !
     !   +----------+       !
     !              !       !
     v              v       v
  Liefertermin    Kostenvoranschlag
```

```
  Zeitplan
  zB. 'Netzplantechnik':
Zeitplan1:
  fruehester Beginn
    ==>fuehestes Ende
                        Zeitplan2:
                          spaetestes Ende
                            <== spaetester Beginn

Zeitplan1:              Zeitplan2:
   jetzt Einkaufen      1630h einkaufen
                          ^
   1230h kochen         1730 kochen
                          ^
   1330h essen          1830h essen
                          ^
   1430h abwaschen      1930 abwaschen
                          ^
   1500h fertig         2000h fertig



Fertigungsunterlagen:
  Schaltplan - schematic diagram, electrical circuit diagram,
               electrical wiring diagram
  ElektronikPlatine - PCB printed circuit board
  Platinenvorlage - PCB Layout
  Loetstopmaske - solder mask
  Beschriftung - designator
  Bohrplan - drill plan
  Bestueckungsplan - assembly
  Stueckliste - parts list, BOM
  Gehaeuse - case
  Frontplatte - front panel
  Explosionszeichnung -
  Zusammenbau - assembly

  Aufgabenstellung - task, problem statement,
      problem draft
  Problemanalyse - problem analysis
  Grobdesign - project draft, (rough design)
  Aufwandschaetzung - estimate, amount
  Zeitplan - schedule, time table
  Kostenplan - cost estimate
```

15 ing.mäßige Projektplanung

Am Anfang steht eine

Aufgabe (Vorhaben, Problemstellung, Aufgabenstellung)	→ "**Aufgabenstellung**"

mit den Fragen

• **was** wird das **kosten?** • **wann** wird das **fertig**? (Liefertermin)	→ "**Kostenvoranschlag**" → "**Zeitplan**"

Wer das *nicht / nicht ausreichend glaubwürdig* beantworten ≙ *belegen* kann, wird

→ **den Auftrag**
→ **die Finanzierung** **nicht bekommen!**

Falls die Beantwortung eine **Planung** erfordert, handelt es sich um ein → **Projekt**

- 1.klares Ziel (=exakte Aufgabenstellung)
- 2.Zeitbegrenzung (=Liefertermin)
- 3.Ressourcenbegrenzung (=Budget)
- 4.komplex (=schriftlich)
- 5.innovativ (=anders))

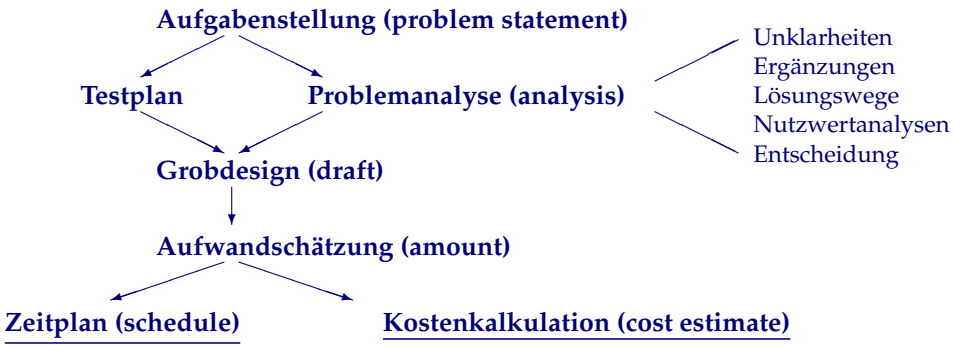
Aus dem Nichts lassen sich *Kostenvoranschlag* und *Liefertermin* nicht ermitteln; es erfordert vorgelagerte Kalkulationen, nämlich die gesamte Projektplanung bis zur *Aufwandsberechnung*.

Also was alles planen?

organisatorische Planung:
Teamorganisation, Aufgabenteilung, Kommunikation, Controlling, QS

kaufmännische Planung:
was wird es kosten?
wann ist es fertig?

technische Planung:
Machbarkeiten, Lösungswege, Realisierungsdetails



- Deckblatt (cover sheet, front/title page)
  (Vorwort = preface)
  (Widmung = dedication)
  (Einleitung = introduction)
- Aufgabenstellung (assignment, problem definition/statement)
- Problemanalyse (task analysis):
  - Methodenplan
  - mögl. Missverständnisse i.d. Aufgabenstellung
  - Ergänzungen zur Aufgabenstellung
  - Lösungswege
  - Nutzwertanalysen
  - Entscheidung
- Testplan/pläne (checklist)
- Grobdesign (draft, basic design)
- Aufwandsberechnung (estimate, expense prediction)
- Zeitplan (schedule)
- Kostenvoranschlag (cost accounting, Kostenkalkulation: cost estimate)
- techn.Detailplanung (detailed design)
- Realisierung (implementation)
- EndTest (final testing)

} Pflichtenheft (specification booklet)

16 Projekt-Mappe/Dokument gesamt

Die Planung umfaßt die Planung und nicht die Ausführung/Realisierung.
(*logisch* würde man meinen).
Alles wird geplant, auch die Planung (der Planungs-Plan heißt *Methodenplan*.

16 .1 □ Deckblatt

= Projekt, Firma+Auftraggeber/(Schule), Name, Datum;

nur in Schule: Klasse/Fach/Gruppe, Foto (vom Gsicht), Betreuer

16 .2 □ Inhaltsangabe

= Aufzählung, Kapitelgliederung, Seitenreferenzen

16 .2.1 *

□ In Pflichtenheften ... störend, in Diplomschriften angebracht: Einleitung, Vorwort, Abstracts, schmalzige Widmungen, scheinheilige Danksagungen...

16 .3 □ Aufgabenstellung

→ möglichst grafisch

- □ UseCaseDiagram (UCD)
- □ BlockDiagramm (BD)
- □ Tabelle
- □ Aufzählung
- □ textuelle Beschreibung

16 .4 □ Problem-Analyse

16 .4.1 □ Methodenplan

im Projekt einzusetzende, standardisierte, etablierte, angebrachte, knappe, präzise, automatisierbare Dokumentations-, Validierungs-, Verifikations-, Planungs-, Entwurfs-, Kalkulations-, Qualitätssicherungs-, Realisierungs-, Test- und Prüfungs- usw. Methoden/Systeme/Werkzeuge
(damit nicht jedes Team-Mitglied anders plant und dokumentiert);

am besten als Tabelle:

Projekt-Phase	Methode(n)
Aufgabenstellung	UCD UseCaseDiagram (aus UML) Block-Diagramm (BD)
Problemanalyse:	
Methodenplan	Tabelle
Mißverständnisse der Aufgabenstellung klären/aufzeigen	Tabelle
Ergänzungen der Aufgabenstellung finden	Tabelle

Projekt-Phase	Methode(n)
Lösungswege	Kreativ-Methoden wie . Brainstorming . MindMapping . 6-3-5 Methode . https://de.wikipedia.org/wiki/Kreativitätstechnik https://en.wikipedia.org/wiki/Creativity_techniques
Nutzwertanalysen	https://de.wikipedia.org/wiki/Nutzwertanalyse https://www.controlling-wiki.com/de/index.php/Nutzwertanalyse
Entscheidung	Punktemaximum
Testplan	. ISO9000x-/QS-, Dokumentations-Planung . Testverfahren+Messmittel . Werkzeuge,Daten/Files (ggf) . schrittw.Anleitung . Test-Personal . Test-Protokolle . Termine/Meilensteine?
Grob-Design	. Top-Down Design . Block-Diagramm . UCD, UML, DesignPatterns . Datenfluss-Graph . State-Charts . CRC-Karten-Methode, Klassendiagramm . Schreibtisch-Simulation . Prototyping Aufgabenteilung Organigramm Komminukationsplan
Aufwands-Kalkulation	Function-Point (cost prediction) Delphi-Methode Experten-Schaetzung
Zeitplanung	. Meilenstein-Plan . Gantt-Diagramm . Netzplan-Technik
Kostenkalkulation (s. BET/WIR)	. Stundensatz je nach Qualifikation . Material, Infrastruktur . Abgaben . Gewinn-Aufschlag . Earned-Value

Projekt-Phase	Methode(n)
Detail-Planungen	. TopDown-Design (+Bottom-up) . UML-Diagramme (DGM):= UseCaseDGM: Verwendungsfall-DGM class DGM: KlassenDGM state DGM: ZustandsDGM/state chart activity DGM: AktivitätsDGM sequence DGM: SequenzDGM component DGM: KomponentenDGM collaboration DGM:KollaborationsDGM deployment DGM: Einsatz/VerteilungsDGM . OO-design patterns . PseudoCode . Flussdiagramm, Struktogramm . ER, relationale DB-Normalisierung . Schreibtischsimulation . Prototyping
Arbeitsprotokolle	US-ASCII text file
Gebrauchsanweisung	. LaTeX-Modul . openoffice Dokument . cartoon, Demo-Film
Abnahmetest	. ISO9000 Testplan . Tabelle

16 .4.2 □ Mißverständnisse der Aufgabenstellung

klären, zB.

"unsere Schaltung besteht nur aus einem Draht?"

16 .4.3 □ Ergänzungen der Aufgabenstellung

zB. *"alle Varianten als Schaltung realisieren oder nur eine?"*

Bei Softwaresystemen sind hier Dinge wie Backup, Import/Export, Konfigurationsmenu, Mehrpsprachigkeit, Datenbank, Benutzerverwaltung, Protokollierung, Installations/DeinstallationsTools, Onlinehilfe, WebInterface, MultiUser-Fähigkeit, Chiffrierung, Passworte, Fernwartung, Anbindungen typisch

16 .4.4 □ Lösungswege

: verschiedene und viele(!),

- ggf. Prototypenbau
- Block-DG,
- Component-DG,
- Deployment-DG,
- Kreativ-Techniken:
  - Mindmaps,
  - Brainstorming,
  - 6-3-5-Methode,
  - Marktforschung,
  - Bionik,
  - Information-Retrieval
  - ...

16 .4.5 □ Nutzwertanalysen

Kriterien	Gewicht	Punkte		
		Lösung1: kaufen	Lösung2: klauen	Lösung3: selbermachen
Arbeitsaufwand	-5	2	1	20
Kosten	-4	5	0	10
Lerneffekt	+3.3	0	0	10
Risiko	-9.5	1	10	3
GesamtPunkte	$\sum$	-5*2 -4*5 -9.5*1 = -39.5	-5*1 -9.5*10 = -100	-5*20 -4*10 +3.3*10 -9.5*3 = -135.5

s.

→`https://www.controlling-wiki.com/de/index.php/Nutzwertanalyse`,

→`http://www.manager-wiki.com/methodik/57-nutzwertanalyse`,

→`https://de.wikipedia.org/wiki/Nutzwertanalyse`

16 .4.6 □ Entscheidung

Die Entscheidung hat allein anhand der Nutzwertanalysen zu fallen! Falls andere Argumente Einfluß nehmen, hat man diese Kriterien offensichtlich in den Nutzwertanalysen mißachtet / vergessen!

16 .5 □ Testplan (QS/QA)

> Achtung!
> QS prüft, ob Produkt und Plan übereinstimmen;
> (und *nicht*, ob das Produkt gut ist!)

- Vereinbarungen zu Produktion, Materialien, Kontrolle, Lieferung, Vorführung, Installation, Testabwicklung, Abnahme, Einschulung, Nachbesserung, Wartung usw.
- Milestones
- Aufzählungen, Tabellen, Screenshots, Prototypen, Flussdiagramme, . . . ,
- s. QA-Vorschriften ISO-9001:2000, ISO-9000-x, vorm. ISO9001, 9002 und 9003, eigene QA-Verfahren

Am besten wieder als Tabellen:

Situation: exakt	Bedienungs-Aktion: exakt	soll-Reaktion: exakt	"ok"	"FAIL"
...	...	...	X	
...	...	...		X
...	...	...	X	
...	...	...	X	
usw.				

16 .6 □ Grobdesign

Wie *detailliert* muß das Grobdesign sein?

→

so detailliert, daß man den **Aufwand klar berechnen** kann!

(... deswegen macht man ja überhaupt ein Grobdesign)

mach:

- □ = TopDownDesign, Blockdiagramme, UseCase-Diagramme(UCD), Klassendiagramme, Komponentendiagramme, Deployment-Diagramme, StateChart-Diagrams, SADT, ER-Design, ...
- □ QA- und Test-Automatisierung, Versionsverwaltung,
- □ Installations-Anleitung, Installations-Programme (Install-CD, Auto-SetUp, rpm, ...),
- □ Aufgabenteilung
- □ Organigramm + Kommunikationsplan(Stellen, Adr, Tel, Mail, Meetings)

16 .7 □ Aufwands-Berechnung (estimate)

- □ Arbeitsaufwand
  (Nicht schätzbarer Arbeits-Aufwand ist unzureichend detailliertem Grobdesign geschuldet!)
  Der Arbeits-Aufwand ist nach Qualifikations-Profilen aufzuschlüsseln. Projektplaner, Administratoren, Datenbankexperten, SecurityHoles, Programmierer, Tester, Handlanger, Sekretäre usw. unterliegen ja verschiedenen Stundensätzen.
  Jeder Kandidat sollte die Aufwandschätzung seiner Projekt-Teils selber machen. Die Stundensumme muß mit den Angaben im offiziellen Antrag übereinstimmen. Im gemeinsamen Teil ist dann nur noch die Summenbildung.
- □ Material
- □ Infrastruktur (Raum, IT, Werkzeuge, Messmittel)
- □ Abgaben

16 .8 □ Zeitplan

da die Zeitplanung kostenrelevante Folgen f. zB. Mieten haben kann, ist sie sinnigerweise vor der Kostenschätzung anzusetzen.

- □ Netzplantechnik (CPM/MPM)
- □ Gantt-Charts
- □ Milestones

Der beste Weg der Zeitplanung ist *Netzplantechnik*, da sie keine "Pufferzeiten" voraussetzt, sondern diese u.a. als Ergebnisse liefert (jeweils spätest - frühest).
Der Netzplan besteht aus zwei Zeitplänen:

(a) vom frühesten Projektbeginn an ohne Verzug vorwärts gerechnet.

(b) vom Liefertermin aus ohne Verzug rückwärts gerechnet.

16 .9 □ Kostenkalkulation + Kostenvoranschlag

- □ Arbeitsaufwand: qualifizierte Stundensätze x MannStunden
- □ Material
- □ (anteilige) Infrastrukturkosten
- □ Abgaben
- □ Einfluss des Zeitplans (längere Mietzeiten, Betriebskosten,. . .)
- □ Reise- und Konferenzkosten nicht vergessen

In der Regel sind (der Größe nach):

- □ Arbeitsstunden
- □ Gebäude (Miete/Rückzahlung u. BK)
- □ EDV-Anlage (incl. Administration)

die Hauptfaktoren.

Angaben wie *"als Schüler arbeiten wir unbezahlt und brauchen daher das Kostenkalkulieren nicht zu lernen"* sind inakzeptabel.

16 .10 Realisierungs-begleitende Dokumentation:

16 .10.1 □ Detailplanungen

Hier sind sämtliche techn. fachmännischen, standardisierten Methoden Planungsmethoden angesagt, natürlich auch die Produkte entsprechender Planungs- und Simulationswerkzeuge, je nach Disziplin;

- Top-Down und Bottom-Up
- in Elektronik und Softwaretechnik bes. UML-Methoden,
- Design Patterns (observer, container, ...),
- Blockdiagramm(BD),
- Blockschaltbild,
- Signalflußplan,
- Schaltplan,
- Datenfluß,
- Pegelplan,
- Pseudocode, Flußdiagramm, Struktogramm,
- ER, DB-Normalformen,
- UserInterface, Screenshots
- Ergonomie, Bedienungskonzepte (Kommando, Knopfdruck, Menu, Icons),
- QuellCode: Nur besonders erklärendswerte Sequenzen
- auch die älteren SADT, Projektstrukturplan, HIPO, Warner, Jackson udergl.

Selbsterfundenene Darstellungen sind dem Leser gewöhnlich unbekannt, stiften mehr Verwirrung als Klärung und sind daher ungeeignet.

16 .10.2 □ Controlling

ist die laufende Überwachung von

- Kosten,
- Zeit
- und Qualität (QS/QA ist also Teil des Controlling)

16 .10.3 □ Implementierung

→ Arbeitsprotokolle!

die sind gewöhnlich für Leser nicht interessant, lediglich für Prüfer. Daher kann man diese recht platzsparend abdrucken, zB. im Querformat zweispaltig mit kleiner Schrift und engem Zeilenabstand. (es ist ja schade um die Kopierkosten)

16 .10.4 □ Test

→ Test/Prüfprotokolle! s.Testplan

16 .10.5 □ Installation

→ Installationsanleitung

16 .10.6 □ Abnahme

→ Abnahmetestprotokoll

16 .10.7 □ Wartung

→ Wartungshandbuch

16 .10.8 □ Verbesserungen, Bug-List, Erfahrungen

Für Anwender und Weiterentwickler sind warnende Hinweise, Erfahrungen oder besondere Beobachtungen wertvoll.
Sonstige Berichte — da kann man Erzählungen unterbringen, etwa über besondere Ereignisse oder bemerkenswerten Projektverlauf.

16 .10.9 □ Erweiterungen

Vorschläge, für Weiterentwickler oder Nachahmer gedachte Vorstellungen zur Verwend- und Verwertbarkeit.

16 .10.10 □ Stichwortverzeichnis, Glossar

16 .11 Fertigungsunterlagen

Speziell für eine Elektronik-Platinen-Fertigung sind (zusätzlich zum allgemeinen PM)

- Schaltplan
- Printvorlage
  - alle Seiten/Layer
  - Positionsmarken
  - Vergrößerungsbemaßung
  - Löt-Lackierung
  - Lötstop-Lackierung
  - Beschriftungs-Lackierungem
- Bohrplan
- Bestückungsplan
- BOM (bill of material)
  mit Lieferanten u. Bestellnummern
- Testvorrichtungen u. -abläufe

erforderlich.

17 NPN als Attenuator - 5cHEL 23.Dez'21

Nachdem am 16.Dez'21 die Fragen

- bei welchen Kleinsignal-Amplituden ist die Signalverzerrung wie groß? (0.5mA Basisstrom)
- wie hängen Signalverzerrungen vom Basisstrom ab?

überwiegend NICHT beantwortet wurden, bitte ich, dies nachzuholen.

Weiters bitte ich, die Unteruchung nicht mit 100k odgl. Collectorwiderstand, sondern niederohmig durchzuführen, sodass zumindest in den Spitzen |Ic| > > |Ib| wird (es gilt nachzuweisen, dass nicht nur Basisstrom zum Collector fließt, sondern auch ('verkehrt' gerichteter) Emitterstrom).

18 NPN als Attenuator - 5cHEL 16.Dez'21

Bitte stelle experimentell fest, wie gut sich ein NPN-Transistor als Kleinsignal-Abschwächer (attenuator) eignet.

Materialbedarf:

```
Steckbrett              Eigentum
Steckdr"ahte            Eigentum
Bauteilset              Eigentum
Labornetzgeraete        Schule
Funktionsgenerator      Schule
Oszilloskop             Schule
Kabel                   Schule
```

Schaltung:

```
*Stromversorgung:
V1      Vbasis  0       ???

*Input-Signal:
Vsin    Uin     0       DC 0 AC ???

*Schaltung:
```

```
Rvor    Vbasis  B1      ???
R1      Uin     Ua      ???
Rc      Ua      C1      ???
Q1      C1  B1  E1      NPN
Rgnd    E1      0       0.000001        ;Direktverbindung
```

spezielle Detailfragen:

- bei welchen Kleinsignal-Amplituden ist die Signalverzerrung wie groß? (0.5mA Basisstrom)
- wie hängen Signalverzerrungen vom Basisstrom ab?

18 .0.1 Abgabemodus

- per eMail
- FileName mit Autor+Datum+Thema, nur ASCII-Buchstaben/Ziffern, kein Blank/Sonderzeichen
- .PDF oder .TXT
- .DOCX + .XLSX gelten als nichtexistent
- die Bewertung verspäteter Einreichungen verzögert sich entsprechend und verschlechtert sich um 1 Notengrad pro Kalendertag

19 JahresLeitfragen Hwe5/1617 XH

19 .1 Übersicht

wiederholen Sie - machen Sie verständlich - wenden Sie an - analysieren Sie - entwickeln Sie -	×	E12 Reihe resistor color code Ebers-Moll-Formel BJT/jFET Steuer-, Eingangs-, Ausgangs-kennlinien EGS Emittergrundschaltung TTL-NAND, CMOS-NAND, -NOR, -NOT Audio- und OP-Amp Innenschaltung SPICE-Netlist 2-u.3-OP-INA single supply-OPV INA Application, INA-118, Amp04 Linearregler ChgPumps Drossel/Trafowandler BuckConverter Ne555 Prinzip Ne555 an MosFET Taktgeber ProjMgmnt Filterberechnung n. Filtertabellen-Methode Gleichungen per KSA akt.Filt.: Koeffizientenvergleich Tiefpass/Hochpass Transformation Filter mit Mehrfachgegenkopplung Wien-Oscillator JN Phasenschieber Oscillator JN BJT LC-Oscillator JN digital: Zähler, KV & Co.

20 Sem9 Lehrstoff moodle2 [KN15]

Lehrstoff

() Bereich **Schaltungsentwicklung**:
Dimensionierung und Interfacing elektronischer Systeme.
() Bereich **Schaltungsanalyse und -simulation:**
Schaltungsbeschreibungen, Analyseverfahren und Simulationswerkzeuge für elektronische Systeme.
() Bereich **Projektmanagement und Qualitätssicherung:**
Verfahren und Standards der Qualitätssicherung

Umsetzung: HWE-GET bcvb HWE-EDT

21 Sem10 Lehrstoff moodle2 [KN15]

Lehrstoff

() Bereich **Schaltungsentwicklung**:
Technische Dokumentation elektronischer Systeme.
() Bereich **Schaltungsanalyse und -simulation**:
Interpretation der Systemsimulationen.
() Bereich **Projektmanagement und Qualitätssicherung**: Qualitätssicherung anhand ausgewählter Beispiele.

Umsetzung: HWE-GET bcvb HWE-EDT

22 se foreword

Die Inhalte dieses Dokuments sind der Phantasie und Utopie entstammende, rein künstlerische Darstellungen bzw. *word art* ohne inhaltlichen Sinn, nicht frei von Rechten Dritter, ohne Eignung für einen bestimmten Zweck, die weder mit Wahrheit, Rechtmässigkeit, Richtigkeit, Brauchbarkeit noch mit deutscher, neuer, diskriminierender[11] oder Wiener Rechtschreibung zu tun haben, und dürfen weder weitergegeben, noch veröffentlicht, noch mechanisch oder elektronisch verarbeitet oder gespeichert werden. Für Inhalte quellverwiesener, zitierter oder verlinkter Werke/Inhalte kann keine Verantwortung/Haftung übernommen werden.

XHs allerwichtigste 3 Grundregeln:
- Alls, was der XH sag, ischa Blödsinn!
- Arbeit muß[a] Spaß[b] machen.
- XH mag keine Texte.[c]
- "gealt nit" gibts nit. (Ingenieur-Prinzip)
- nur aus Fehlern wird man klug[d]

[a] stammt noch aus alter Rechtschreibzeit
[b] Arbeit macht 'Spaß', wenn mans gern tut und gut kann; erreichtma mit Verbesserungsbestrebungen = gut-sein-wollen
[c] also Diagramme, Skizzen, Screenshots, Bilder, Tabellen udgl. statt *benutzerfreundlich, übersichtlich, selbsterklärend, irgendwie*
[d] Misserfolge geben zu denken

Als Vorlage verwendi die
Diplomschrift- Vorlage
von AV YH
(s. moodle2 und obelix) mit geringen Anpassungen in Schriftstil (Arial) u. Layout (A4) . . .

Dem unerreichbaren, großen
Vorbild und LaTeX-Mentor
Prof. Dr. Robert SALVADOR
gewidmet und gedankt!

22 .1 der *obelix* Account

Der *obelix* ist mein jahrzehntelanger, schülergewidmeter Linuxrechner im Schulnetz. Er hat die IP `http://10.10.63.61/` und den *hostname obelix*, ist mit dieser URL aber nicht im Schul-DNS eingetragen (das wird zwar vielfach gewünscht, verweigere iXH aber bewusst, damit die Teilnehmer das Arbeiten mit IP statt URL sehen/lernen *müssen*). Das Ding ist im Unterricht und Alltag vielfach nützlich, als Webserver mit U-Material, Datenblattspeicher, Programmierwerkzeug, LAN-Experiment-Partner, Onlinedoku-Bibliothek (man-pages), WebDatenbankserver(MySQL), Schaltungssimulator(SPICE), DTP-Tool(LaTeX), Abgaben-Ordner und mein pers. zunehmend erforderliches *extended Memory*.

22 .1.1 Aufbau of se user name

am *obelix* `http://10.10.63.61/` hat man einen eigenen Account;
der Benutzername (zB. "n15afffvv") besteht aus

```
   n = EL-Abt. (w=WI Abt.)
  15 = Jahr 2015 des HTL-Einstiegs
   a = a-Klasse
 fff = Familienname, Buchst.1--3
  vv = Vorname, Buchst.1 u. 2
  zB. "n15amaifr", "n16cxmilu" usw.
```

(Umlaute und ß werden zweibuchstabig ersetzt ö=oe, ß=ss,..., auch *sch* durch "x"; das Standartpasswort lautet "htl" (klein!) — ändere es raschest!

[11] *[...] Das Wort Diskriminierung stammt von dem aus dem lateinischen Verb discriminare 'trennen', 'absondern', 'abgrenzen', 'unterscheiden' im Spätlateinischen abgeleiteten Verbalsubstantiv discriminatio 'Scheidung, Absonderung.' Das Verb diskriminieren wurde im 16. Jahrhundert in der wertneutralen Bedeutung 'unterscheiden', 'sondern', 'trennen' ins Deutsche entlehnt und ist dort seit dem 19. Jahrhundert kontinuierlich belegt [...]* (de.wikipedia.org 01Nov15)
'to discriminate': unterscheiden, einen Unterschied machen, unterschiedlich behandeln [...] (www.dict.cc 01Nov15)
Als Diskriminator (lat. discriminare = trennen, scheiden) werden verschiedene Geräte oder Baugruppen innerhalb von Geräten der Nachrichtentechnik, Elektronik und Messtechnik bezeichnet. Diskriminatoren dienen zur Auswertung analoger oder digitaler Signale [...] (de.wikipedia.org/wiki/Diskriminator 01Nov15)
Die Formulierung »Schüler und SchülerInnen« macht einen Unterschied, indem sie extra genannt werden, »diskriminiert« also; zudem sind Großbuchstaben mitten im Wort *nicht* deutsch! Es verbindet also »SchülerInnen« mit groben Rechtschreibfehlern. Darin ist mE. insgesamt eine ganz ganz große Beleidigung und Geringschätzung der gesamten Weiblichkeit zu sehen — mE. diskriminierende Nichtrechtschreibung.
Eine Titelschreibweise wie Mag[a] ist mW. normenwidrig, der Titel ist *nichtexistent* und vor allem ist er eine diskriminierende Frauenkennzeichnung (*'Frauen sind kennzeichnungspflichtig'*) wie »Intel inside«, vergleichbar dem »Judenstern« im dt.Reich; ist das »verbotene Wiederbetätigung« ?

22 .2 remote Login zum Linux Server (zB obelix)

Wie stelle ich eine *ssh*- Remote-Login Verbindung zum *obelix* her:

vom Linux:

`ssh n18bxxxyy@10.10.63.61`
password: 'htl'
'cd public_html'

Achtung:
Linux ist **"case-sensitive"**
Gross/Kleinschreibung
ist **relevant!**

vom MacOS:

Systemkonsole starten
dann wie auf Linux: `"ssh n18bxxxyy@10.10.63.61"`
password: 'htl'
'cd public_html'

am Winzigweich:

`"putty 10.10.63.61"` (oder zB TeraTerm)
username: 'n18bxxxyy'
password: 'htl'
'cd public_html'

*seit Version 10.?? habe 'powershell' auch ein (reduziertes) "ssh"

(mit xxx ... erste 3 Nachnamensbuchstaben,
yy ... erste 2 Vornamensbuchstaben)

22 .3 C-Programmcodeeingabe am 'obelix'

nach erfolgreichem 'remote login' mit 'ssh' oder 'putty':

im Webseitenbereich:

1. *'cd ~/public_html'*
2. *'mkdir -pv gibNichtXH/meinprojekt'*
3. *'cd gibNichtXH/meinprojekt'*
4. *'nano meinPgm.c'*
5. (C-Programmcode eintippen)
6. (abspeichern mit Strg+O)
7. 'nano' verlassen mit Strg+X

Achtung:
Linux ist
"case-sensitive"
Gross/Kleinschreibung
ist **relevant!**

ausserhalb
Webseitenbereichs:

1. *'cd ~'*
2. *'mkdir -pv gibNichtXH/meinprojekt'*
3. *'cd gibNichtXH/meinprojekt'*
4. *'nano meinPgm.c'*
5. `(C-Programmcode eintippen)`
6. `(abspeichern mit Strg+O)`
7. 'nano' verlassen mit Strg+X

Es gibt **kein**
Datei-
"WIEDERHERSTELLEN"!

22 .4 C-Code XH- Abgabe ins 'gibXH' am 'obelix'

Alles, was DU ins `public_html/gibXH` ablegst, wird bewertet!

Nach erfolgreichem 'remote login' mit 'ssh' oder 'putty',
erfolgter Programmcodeeingabe und Test:

gibXH-Abgabe:

1. *'mkdir -pv ~/public_html/gibXH/meinprojekt'*
2. *'cp -v meineAbgabe ~/public_html/gibXH/meinprojekt'*

22 .5 C-Compilieren am 'obelix'

nach erfolgreichem 'remote login' mit 'ssh' oder 'putty'
plus C-Programmcode eintippen:

ohne Zusatzlibrary: *'gcc -o myBin meinPgm.c'*

incl. 'math'-library: *'gcc -o myBin meinPgm.c -lm'* f. sin(), exp(), pow() udgl.

incl. 'pthread'-library: *'gcc -o myBin meinPgm.c -lptread'* (pthread ... Multithreading)

incl. 'rt'-library: *'gcc -o myBin meinPgm.c -lrt'* ('rt' ... RT, real time)

incl. math u. RT: *'gcc -o myBin meinPgm.c -lm -lrt'*

22 .6 compiliertes C-Programm ausführen am 'obelix'

nach erfolgreichem 'remote login' mit 'ssh' oder 'putty'
plus C-Programmcode Eintippen plus Compilieren:

im current directory: *'./myBin'*

in '/pfad/zum/dir': *'/pfad/zum/dir/myBin'*

22 .7 C++ Compilieren am 'obelix'

nach erfolgreichem 'remote login' mit 'ssh' oder 'putty'
plus Programmcodeeingabe (in diesem Bsp. ins File 'meinCCpgm.C'):

ohne Zusatzlibrary: *'g++ -o myCCbin meinCCpgm.C'*

'gcc' und 'g++' sind derselbe Compiler;
das 'g++' sagt ihm lediglich,
dass er C++ zu übersetzen hat
und C++ Libraries einbindet.

22 .8 Windows Netzlaufwerk mappen

(1) den Windows-Explorer starten
(2) Netzlaufwerk verbinden
(3) share: \\10.10.63.61\n18bxxxyy
(4) unter anderem Benutzernamen (sonst nimmt es den Win- User)
(5) Benutzername: n18bxxxyy
(6) Kennwort: htl
(7) fertigstellen

22 .9 Password

Das Linux-User-Passwort ändert man mit *passwd <username>*
und das Windows-Netzlaufwerk-Password (samba) mit *smbpasswd <username>*

23 Vorschlag f. HWE'1718

Erlaubte Hilfsmittel:

- Taschenrechner nicht programmierbar
- Böhmer Bauelemente der Elektronik
- Datenblätter zur Aufgabenstellung

pro Teilfrage ca. 9 Minuten Zeit.

· akt. u. pass. Bauteildimensionierung und -Auswahl
· E6, E12, E24 Reihen
· Datenblattlesen, Bauteilkennlinien
· Blockschaltbild, Prinzipschaltung
· Kleinsignal-Ersatzschaltung
· (kombiniertes) Signaldiagramm
· Schaltung erklären
· BJT, jFET, Verstärker, OPV, INA,
· RC- u. XTAL Oszillator, Amplitudenregelung
· single-/dual supply Operation (uni-/bi- polare Stromversorgung)
· akt.Filter, Filterordnung, Filtercharakteristik
· normierte Filtertabellen, H(s), H(jw)
· Sampling, Aliasing, S+H,
· H(s), H(jw), BodeDG dB-Rechnung, Frequenzgang, dB-Rechnung
· Gatter, Kippschaltungen, 555, Pulse, Tastverhältnis,
· 0.7τ - 1.4τ - 2.2τ, $U(t) = U_0 \cdot e^{-t/\tau}$, RC Auf-/Ent-ladekurven
· 74LS-, 74HC-, CD4000-, TTL- Logik
· Zähler/Teiler, Register/Latch, Decoder
· LED, 7seg.LED, Vorwiderstand
· analoger/digitaler Aufgabenteil

24 HWE-Matura/08Mai17

25 ©copyrights, Haftungsausschluss

©Copyrights by ©F.Klingler KN, ©M.Signitzer SM, ©R.Salvador SV, ©G.Schlemmer XM, ©C.Schönherr XH, ©A.Stumpfel YU, ©G.Steinwender YW