Analog- und Hybridrechnen im 21sten Jahrhundert

Prof. Dr. Bernd Ulmann, anabrid GmbH

22. November 2025

Nach vielen Jahrzehnten stößt das klassische Digitalrechnen langsam an prinzipielle Grenzen, was andersartige Ansätze in den Fokus rückt:

- Analog- und Hybridrechnen und
- Quantencomputer.

Was ist ein Analogrechner?

Ein Analogrechner löst Probleme, indem ein mathematisch äquivalentes (elektronisches) Modell aufgebaut wird, an dem Lösungen gemessen werden können. Dieser Ansatz ist grundlegend nicht (!) algorithmisch.

Warum interessiert man sich für solche Ansätze?

Probleme klassischer Digitalrechner

Probleme

Obwohl (speicherprogrammierte) Digitalrechner allgegenwärtig sind, weisen sie eine ganze Reihe von Nachteilen auf, die immer mehr in den Vordergrund rücken:

- Hoher Energiebedarf
- Beschränkte Taktfrequenzen (diese haben direkten Einfluss auf den Energiebedarf)
- Strukturen auf integrierten Bausteinen können nicht beliebig klein gemacht werden – am Ende sind auch Atome vergleichsweise große Objekte...
- Parallelismus ist hart! (AMDAHLsches Gesetz)
- Nur ein kleiner Teil der Chipfläche in einer modernen CPU rechnet wirklich – der Großteil ist mit „housekeeping“ Aufgaben beschäftigt...

Größe

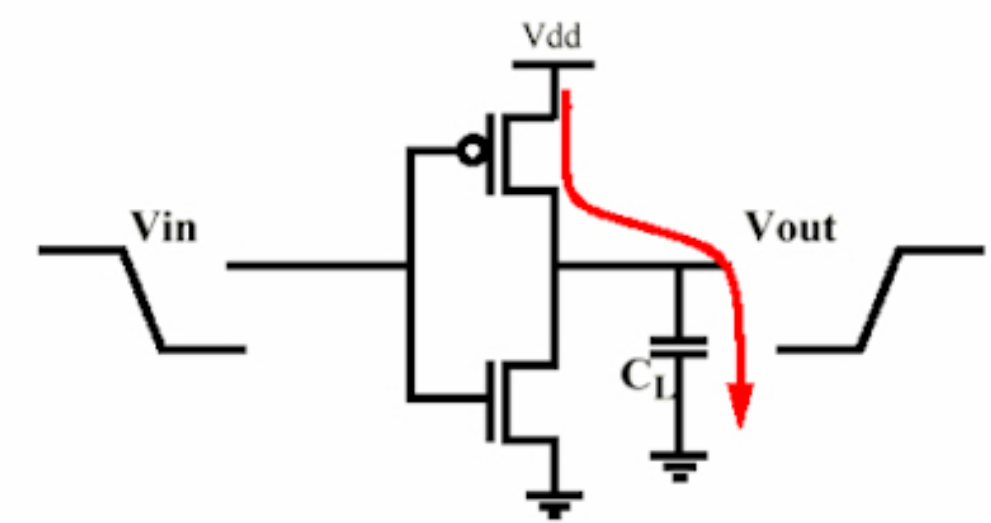
https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Frontier_Supercomputer_%283%29.jpg/2560px-Frontier_Supercomputer_%283%29.jpg, abgerufen am 05.06.2024

Kühlung

https://assets.bwbx.io/images/users/iqjWHBFdfxIU/iSYEWEeVCg_8/v1/-1x-1.jpg, abgerufen am 05.06.2024

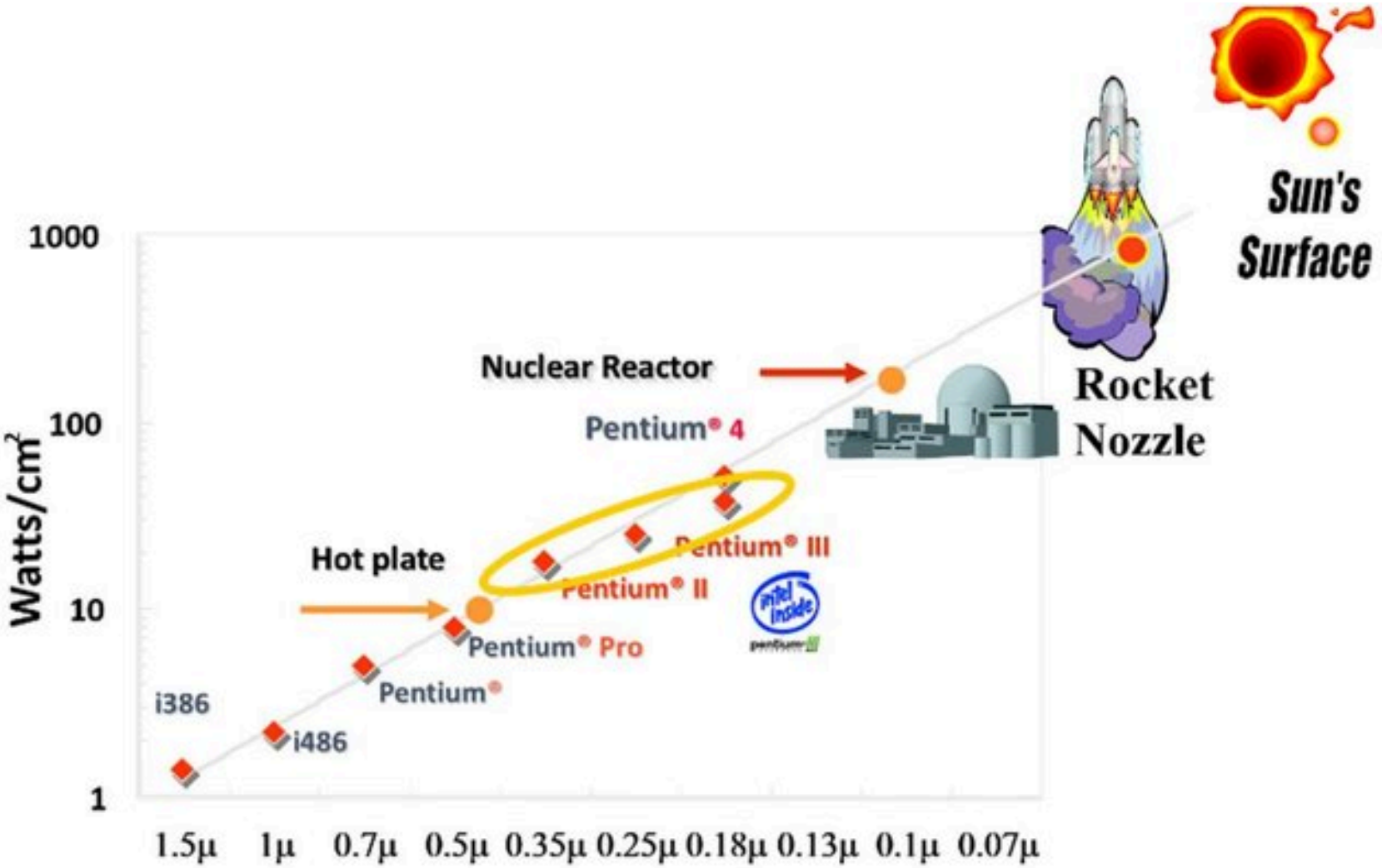
Energiebedarf

Woher rührt der hohe Energiebedarf im Wesentlichen? Eine typische digitale Ausgangsstufe sieht etwa so aus:



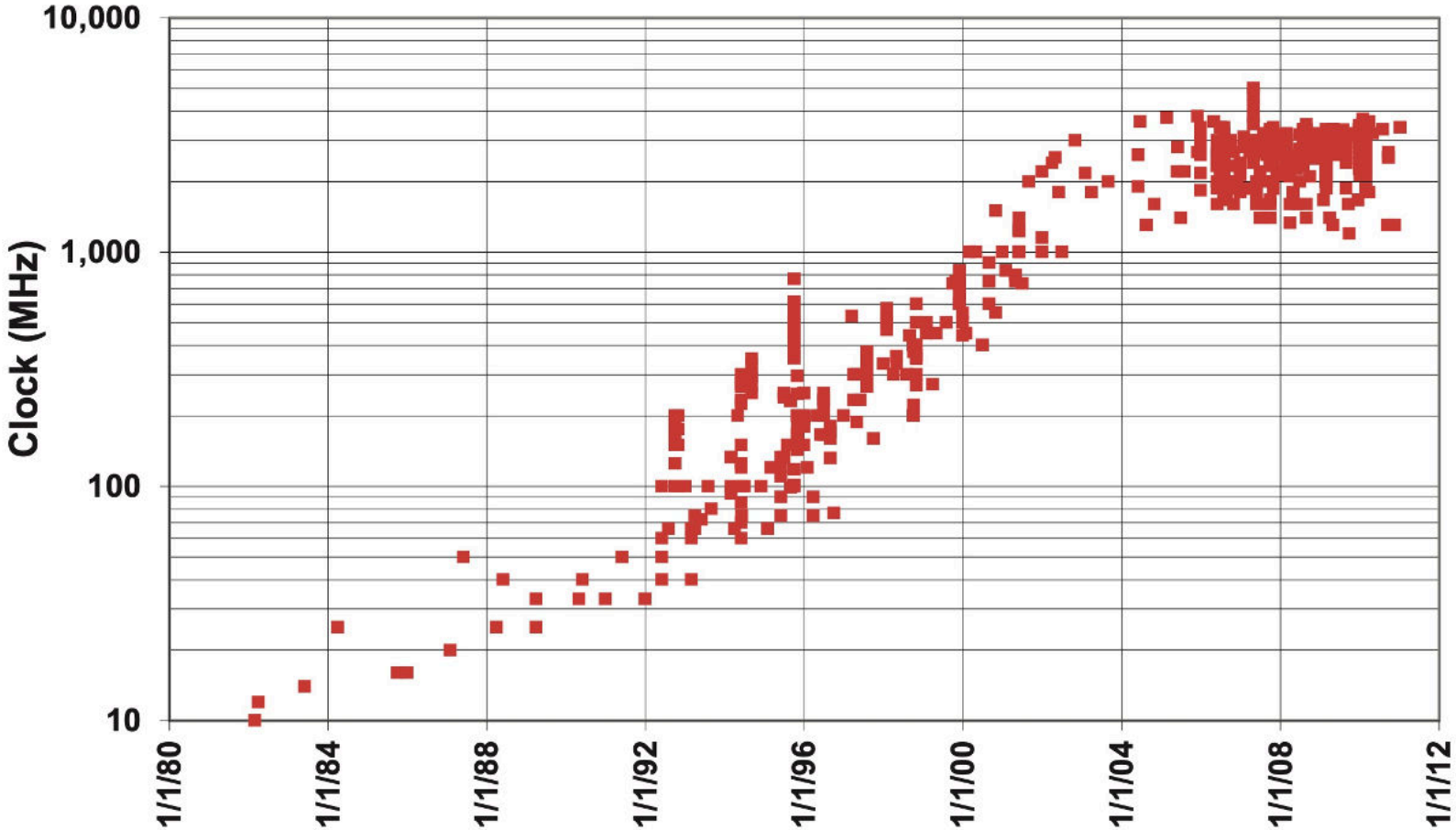
- C_L muss ständig umgeladen werden,
- für eine sehr kurze Zeit leiten beide Transistoren im Umschaltmoment,
- die Transistoren sind nicht perfekt und weisen Leckströme auf,
- $P_{cpu} = P_{dyn} + P_{sc} + P_{leak}$ ist super-linear bezüglich der Taktfrequenz!

Energiebedarf



Siehe [Etiemble(2018), Fig. 8].

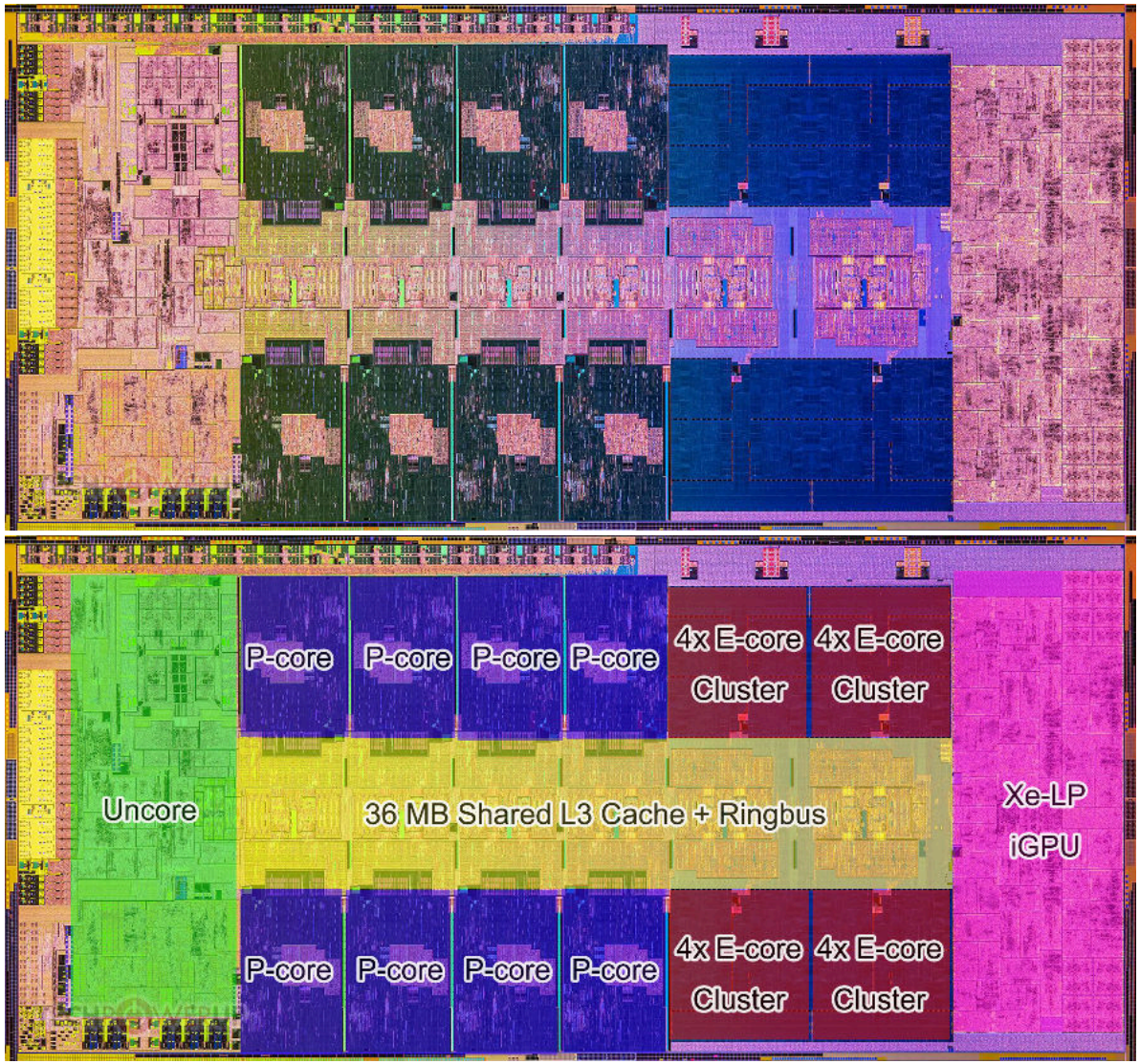
Taktfrequenzen



https://wgropp.cs.illinois.edu/courses/cs598-s15/lectures/lecture15.pdf, S. 11, 11.03.2024

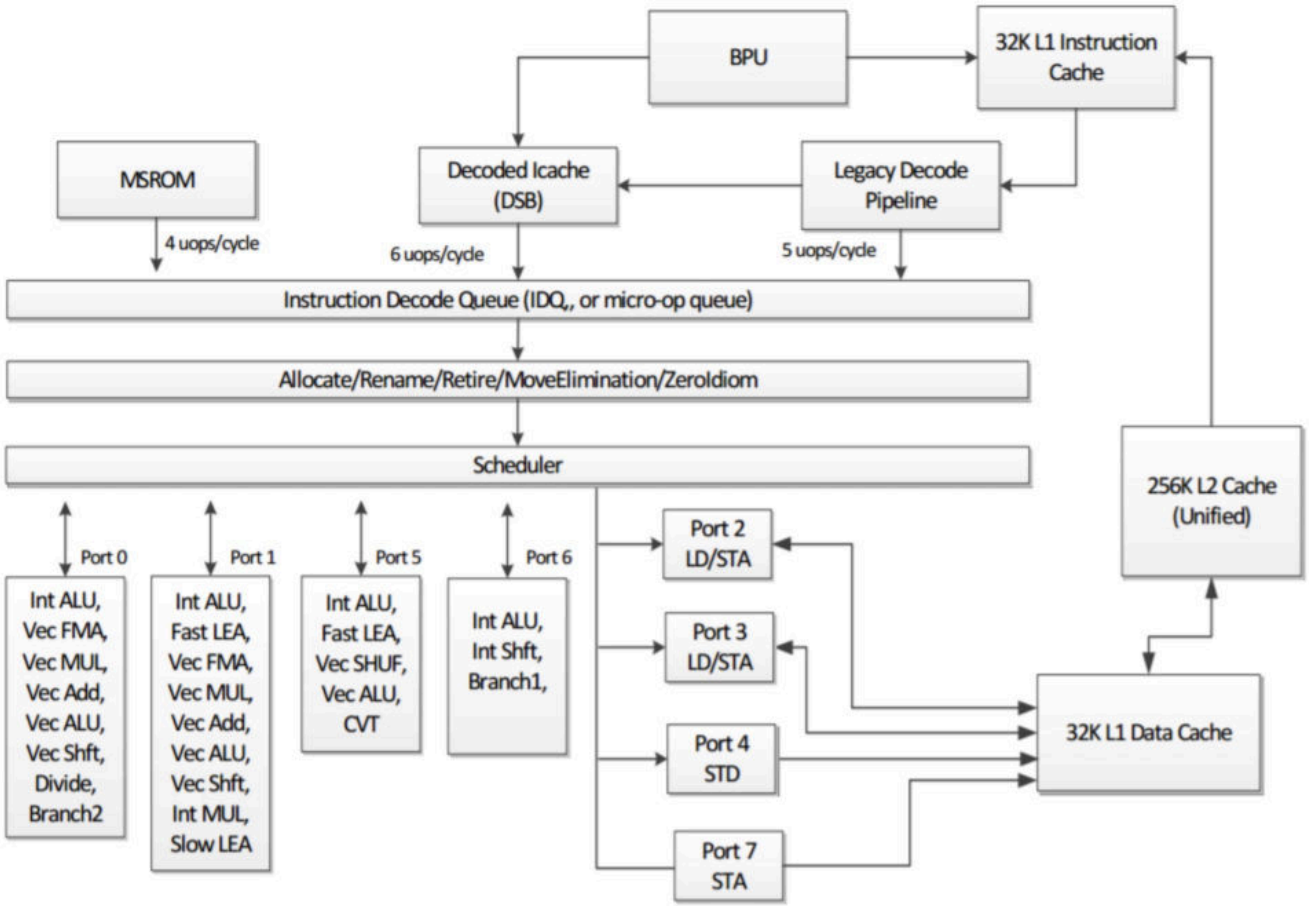
Typische Chip-Struktur

Intel Core i9-13900K:



https://www.techpowerup.com/review/intel-core-i9-13900k/2.html, abgerufen am 08.06.2023

Typische Chip-Struktur



https://www.anandtech.com/show/11550/

the-intel-skylakex-review-core-i9-7900x-i7-7820x-and-i7-7800x-tested/3, abgerufen am 08.06.2023

Vergleich

Schnellster digitaler Supercomputer (`top500.org`, November 2024):

El Capitan, DOE/LLNL USA

- Rmax: 1742 PFLOPS
- 11 039 616 Kerne
- Ca. 29.581 MW
- About 58.89 GFLOPS/W

Menschliches Gehirn:

- Ca. 38 PFLOPS
- 25 W
- Ca. 1.52 PFLOPS/W

Der „Trick“ biologischer Gehirne ist, dass sie nicht algorithmisch arbeiten – es sind Analogrechner!

Nachteile von Algorithmen

- Algorithmen bestehen aus Folgen einfacher Anweisungen, die im Wesentlichen sequenziell abgearbeitet werden.
- Im Unterschied hierzu sind reale Systeme hochgradig parallel. . .
- Ein biologisches Gehirn ist ein perfektes Beispiel hierfür: Es besteht aus einer Vielzahl kleiner Rechenelemente, Neuronen, deren Verschaltung, zusammen mit den Kopplungsgewichten, die „Programmierung“ darstellen.
- Ein Gehirn führt keinen Algorithmus aus, es besitzt keine Trennung zwischen Rechenelementen und Speicher, es ist ein Analogrechner.

Klassische Digitalrechner sind *speicherprogrammiert*, während Analog- und Quantencomputer nicht algorithmisch programmiert werden, sondern durch geeignete Verschaltung von Rechenelementen Probleme lösen.

Analogrechnen

Analogrechner

- Analogrechner werden programmiert, indem eine Vielzahl von *Rechenelementen* miteinander verschaltet werden, um ein Modell, ein *Analogon* des zu lösenden Problems zu bilden.
- Sie sind extrem energieeffizient.
- Werte werden i.d.R. als (kontinuierliche) Spannungen oder Ströme, nicht jedoch als diskrete Bits dargestellt.[1]
- Sie arbeiten vollkommen parallel – für sie gilt das AMDAHLsche Gesetz nicht.
- Sie passen perfekt zu unserer analogen Welt – keine Analog-Digital-Wandlungen etc. erforderlich.
- Sie sind perfekt zur Lösung von Problemen geeignet, die durch Differentialgleichungen beschrieben werden können.
- Sie können sogar einige Dinge, die sonst Quantencomputern zugeschrieben werden! :-)

[1] Auf *DDA*s wird im Folgenden nicht weiter eingegangen.

Was ist ein Analogrechner?

Ein Analogrechner

- besteht aus einer Vielzahl von *Rechenelementen* (Summierer, Multiplizierer, Integrierer etc.),
- die miteinander *verschaltet* werden, um ein bestimmtes Problem zu lösen. Diese Verschaltung muss natürlich änderbar sein.
- Ein Programm ist also ein *gerichteter Graph*.
- Sie besitzen keinen Speicher und führen keine Instruktionen aus.
- Alle Rechenelemente arbeiten vollständig parallel zueinander.

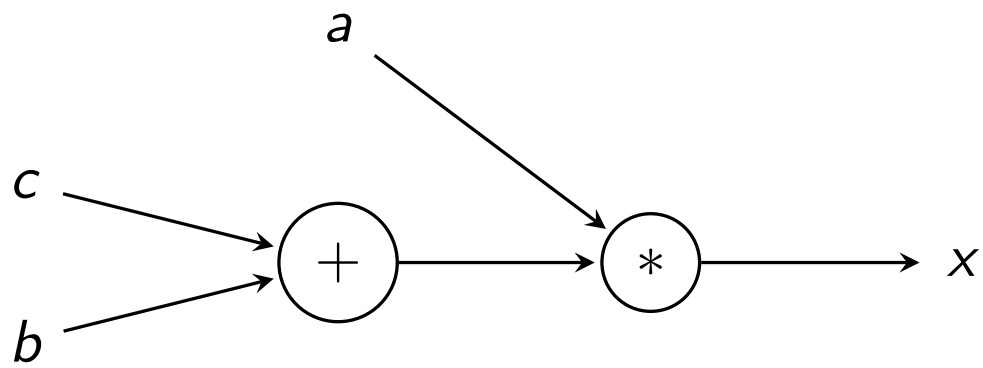
Das folgende Beispiel zeigt die zentralen Unterschiede zwischen Digital- und Analogrechnern auf:

Zu berechnen ist $a(b+c)$ auf einem speicherprogrammierten Digitalrechner:

```
LOAD  A, R0
LOAD  B, R1
LOAD  C, R2
ADD   R1, R2, R1
MULT  R0, R1, R0
STORE R0, ...
```

Das sind sechs Instruktionen, die bestenfalls teilweise überlappend ausgeführt werden können.

Berechne $a(b + c)$ mit einem Analogrechner:



Nachteile

Natürlich weisen auch Analogrechner Nachteile auf:

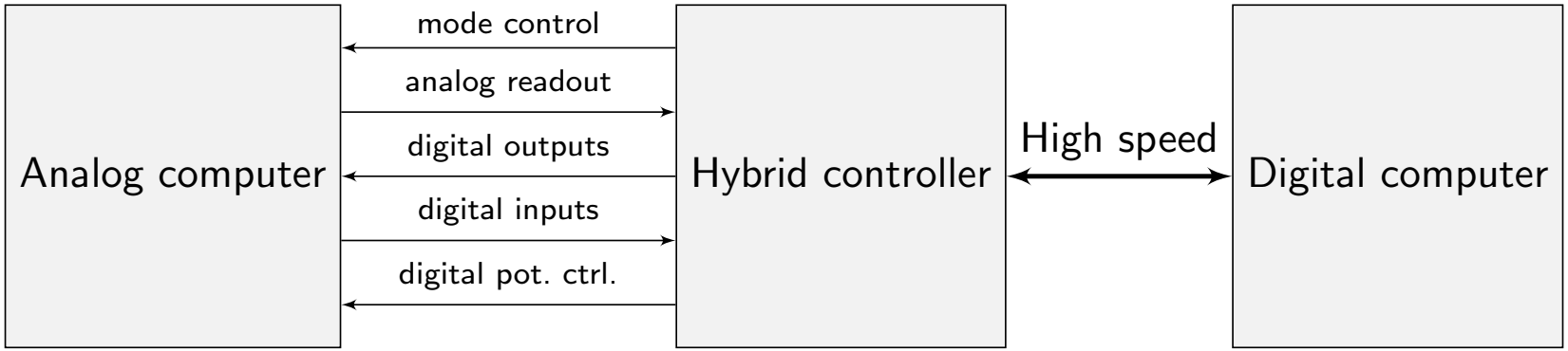
- Die Rechengenauigkeit ist auf drei bis vier Dezimalstellen beschränkt, was aber in der Mehrzahl praktischer Anwendungen kein wirkliches Problem darstellt.
- Integrationen können direkt nur bezüglich der Zeit ausgeführt werden, so dass partielle Differentialgleichungen besondere Lösungstechniken erfordern.
- Die Erzeugung arbiträrer Funktionen (z. B. auf Basis von Messdaten) ist aufwändig.
- Werte sind auf das Intervall $[-1, 1]$ beschränkt, was eine *Skalierung* des Problems erfordert.

Diese Probleme lassen sich aber mit *Hybridrechnern* (z. T.) lösen:

Was ist ein Hybridrechner?

- Ein Hybridrechner besteht aus einem Digital- und einem Analogrechner, die miteinander gekoppelt sind.
- Der Analogrechner dient hier als *Co-Prozessor* und entlastet den Digitalrechner bei bestimmten Problemklassen.
- Der Digitalrechner übernimmt die Konfiguration (Programmierung) und Parametrisierung des Analogrechners, versorgt ihn mit Daten und holt Resultate ab.
- Der Analogrechner muss nahtlos in das digitale Umfeld eingebunden werden, was entsprechende Bibliotheken, aber auch Programmiersprachen erforderlich macht.

Was ist ein Hybridrechner?



Anwendungsbereiche

- Hochleistungsrechnen
- Künstliche Intelligenz (Training/Inferenz etc.)
- Medizinische Anwendungen (Herz-/Hirnschrittmacher etc., die ihren Energiebedarf z. B. über *Energy Harvesting* decken können)
- Industrielle Mess-, Steuer- und Regelungssysteme (hier stehen vor allem die hohe Stabilität, fast ausgeschlossene Angreifbarkeit etc. im Vordergrund)
- Signalvor- und -nachverarbeitung für tragbare Geräte oder *always on devices*
- Triggerworterkennung für *smart devices*
- Monte-Carlo-Simulationen, Finanzmathematik
- Optimierungsprobleme

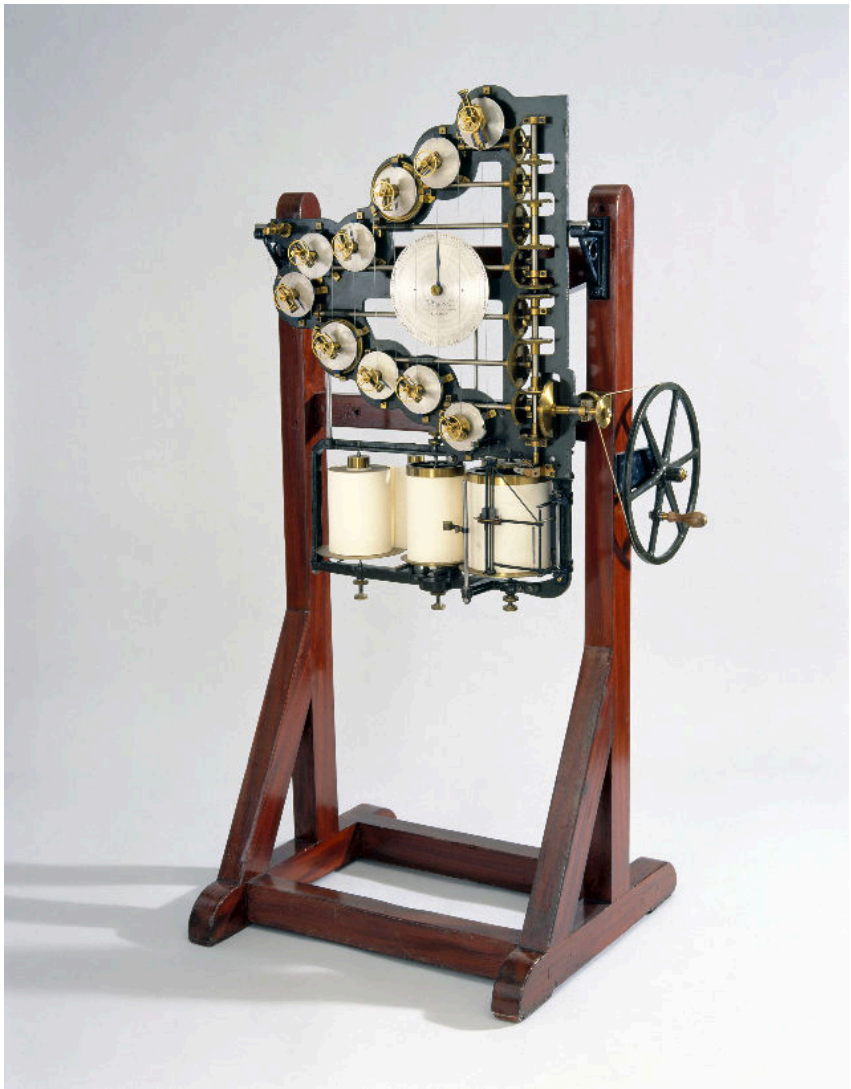
Eine kurze Geschichte...

Der *Mechanismus von Antikythera*:

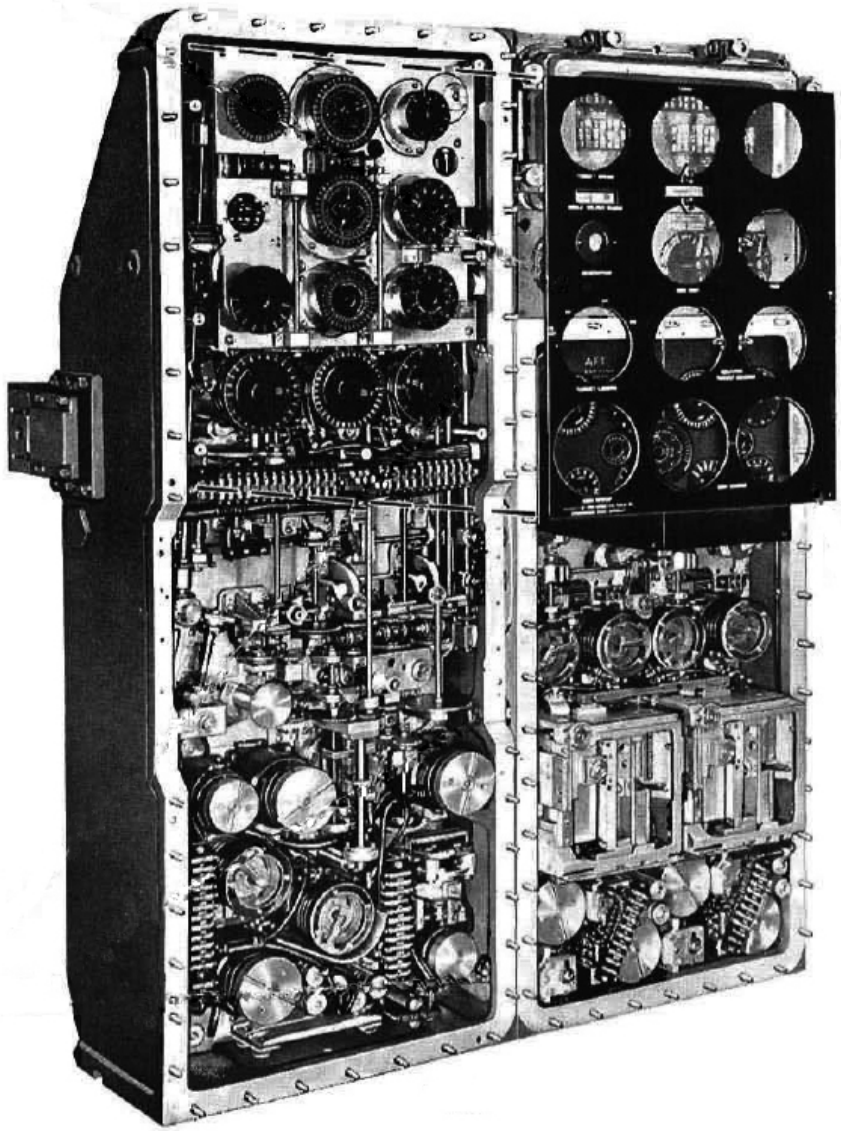
(http://upload.wikimedia.org/wikipedia/commons/6/66/NAMA_Machine_d%27Anticythre_1.jpg)

Kelvins Gezeitenrechner

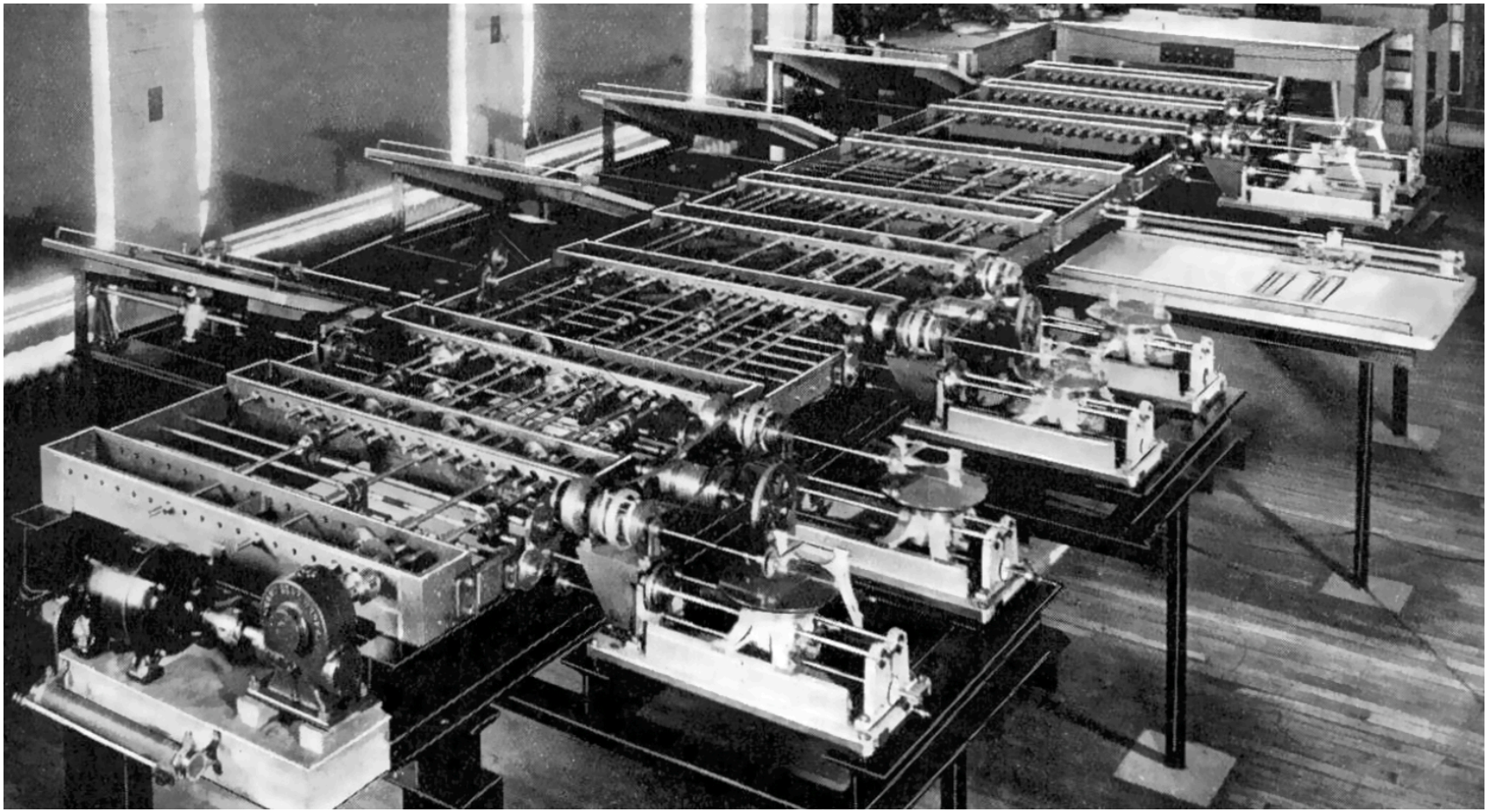
Ein harmonischer Synthesizer (Fourier-Synthese) mit 10 *partial tides*, 1876:



Torpedo Data Computer Mark 3 ([BORD44, S. 150]):

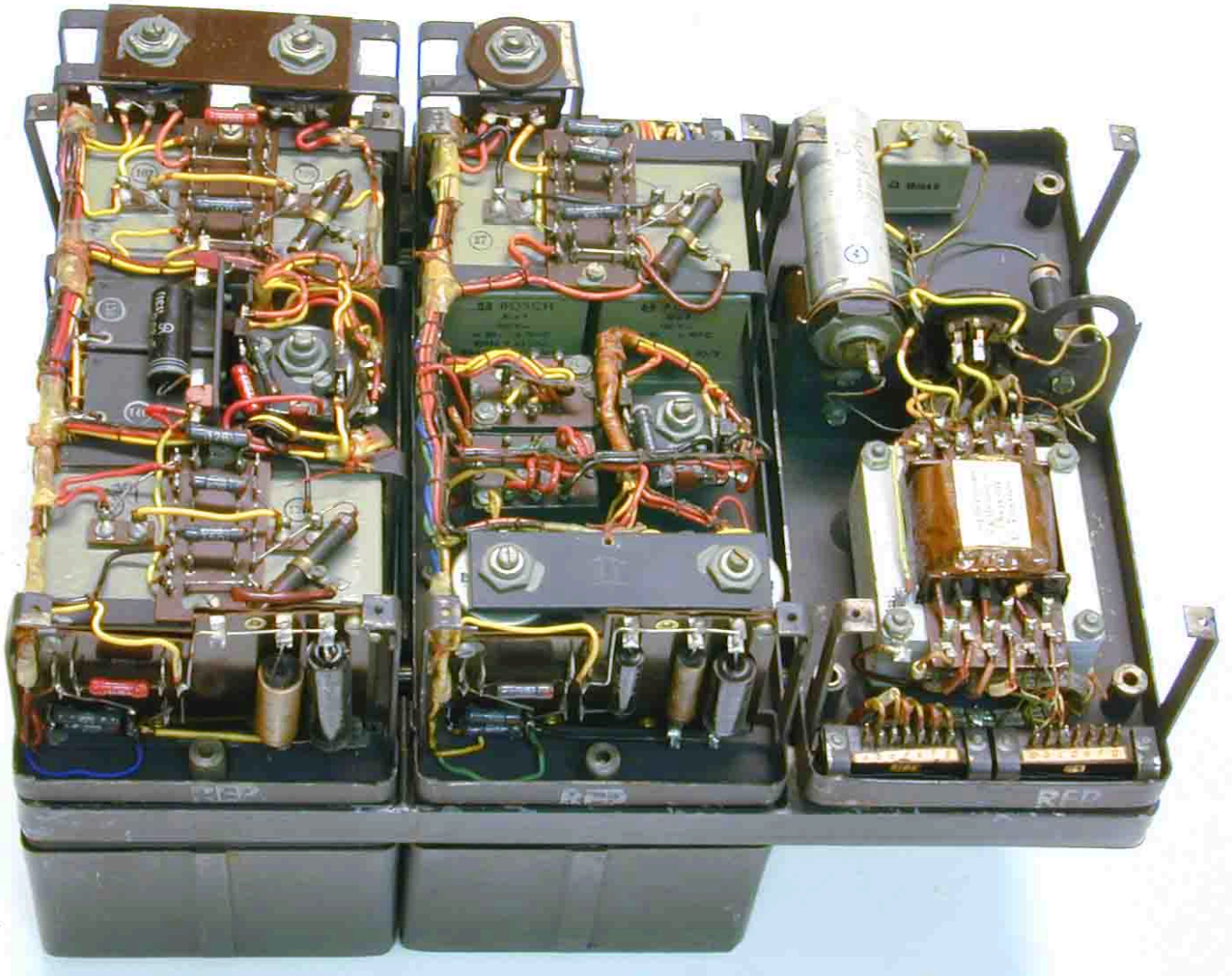


Vannevar Bushs DA, 1920er Jahre:



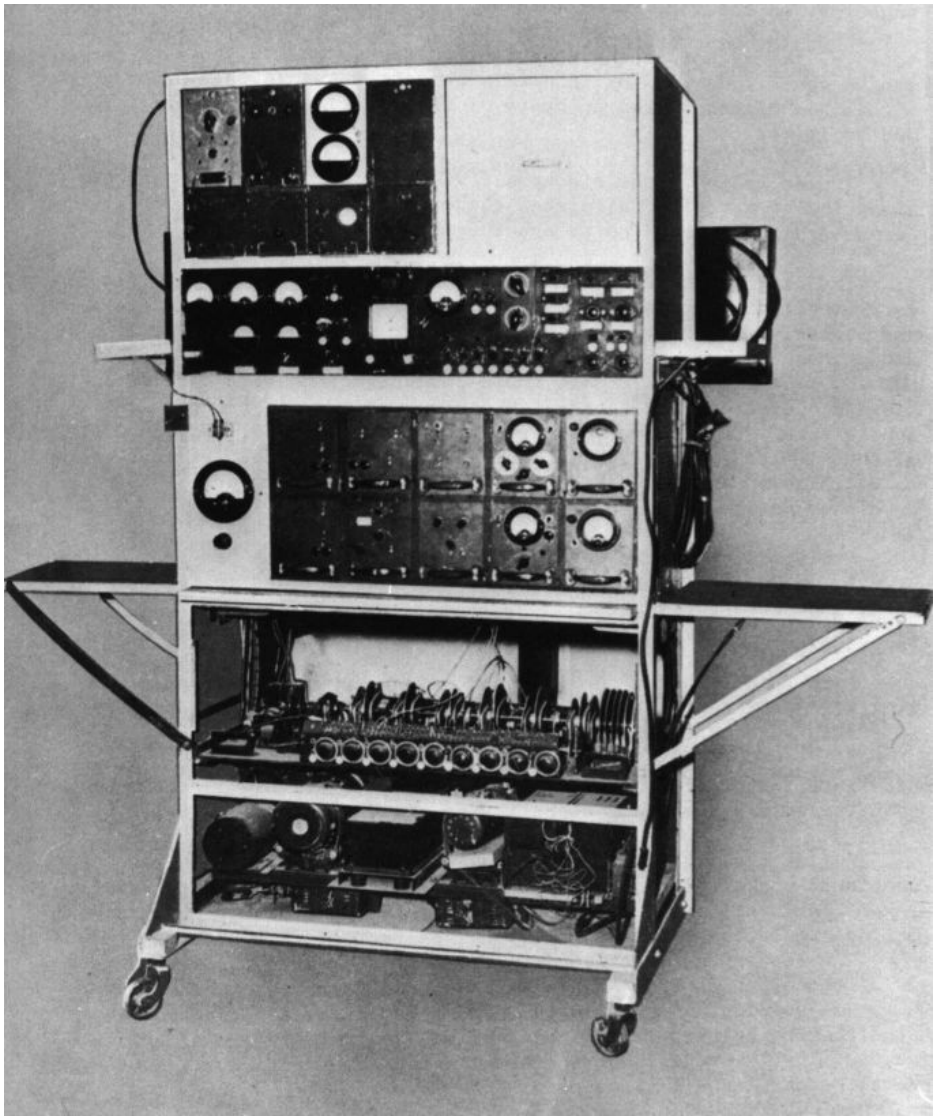
(Siehe [Meccano 1934, S. 443].)

Das *Mischgerät* (1940er Jahre):

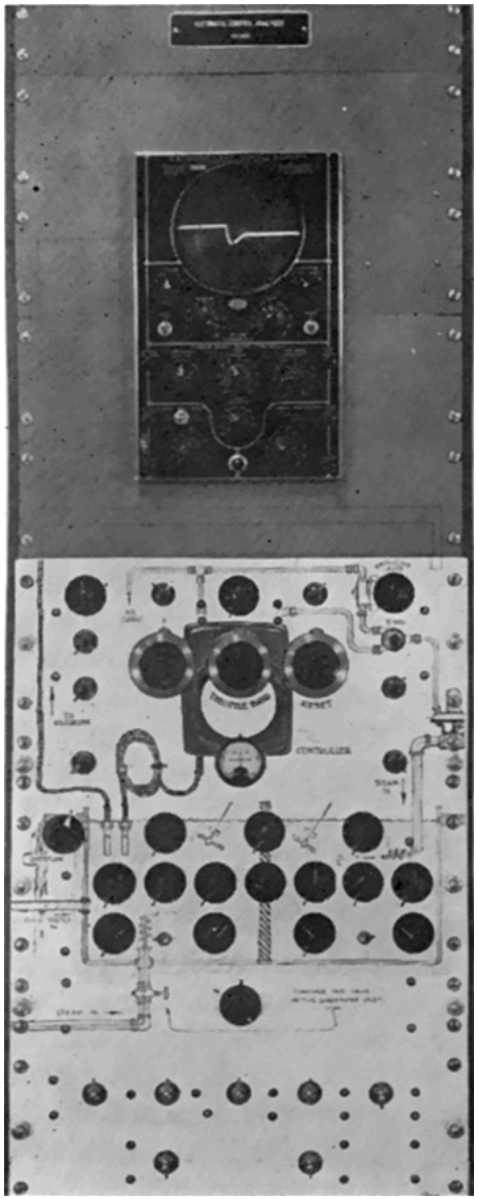


(Photo von Adri de Keijzer.)

HOELZERs Analogrechner:

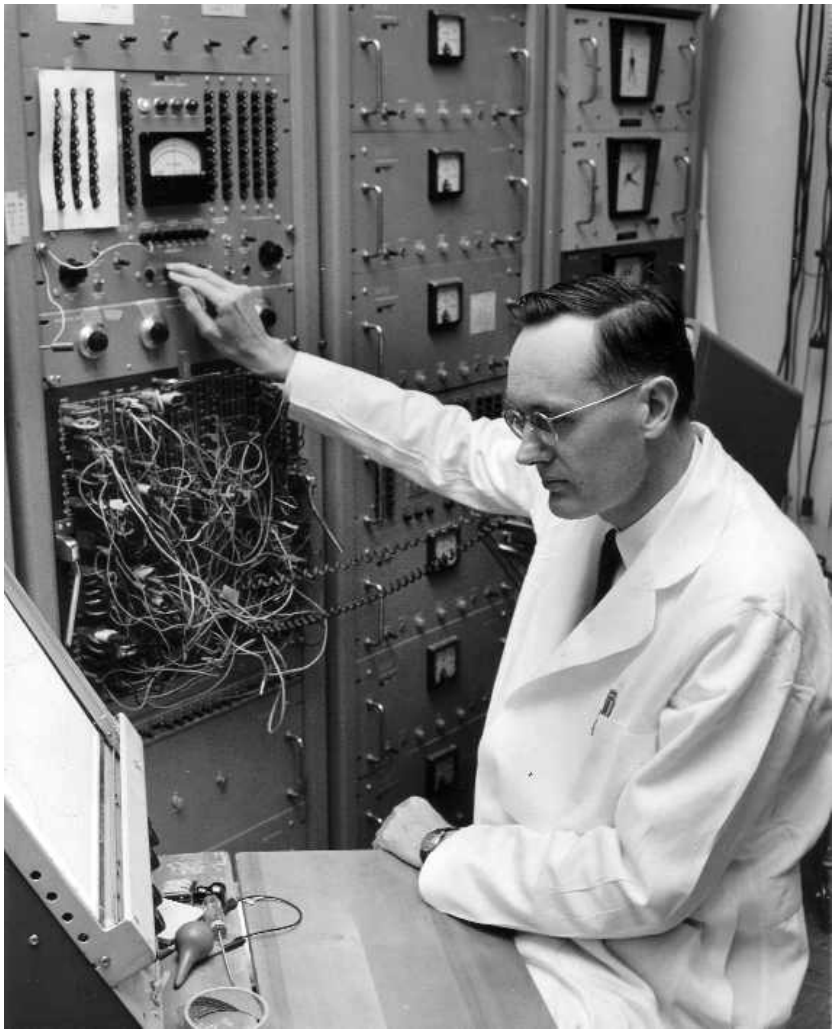


Polyphemus



In den späten 1930er Jahren entwickelte George A. Philbrick *Polyphemus*, einen Prozesssimulator für die chemische Industrie (siehe [Philbrick 1948, S. 108]).

Beckman



Richard FitzHugh löst die Hodgkin-Huxley-Gleichungen auf einem Beckman Analogrechner (Izhikevich E. M., FitzHugh R.).

EAI 231R Analogrechner am „Deutsches Elektronensynchrotron“, Hamburg, Bild: Inge Borchardt.

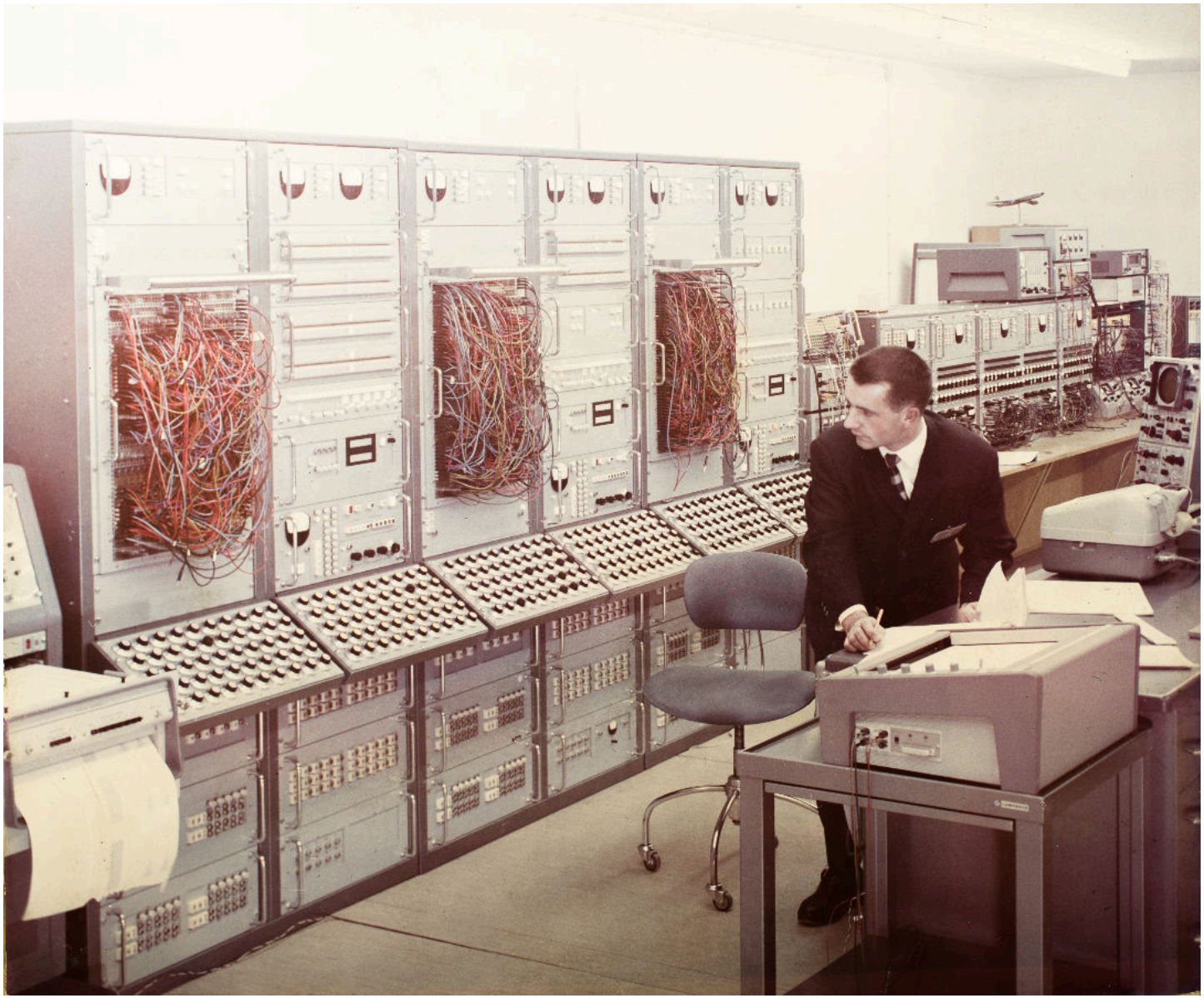
EAI im DLR

EAI Systeme im DLR (1970er Jahre), Bild: Dr. Jessika Wichner, DLR, Signatur FF-557.

Fünf RAT 700 und eine RA 800:

Ein Flugsimulator

Flugsimulator bei Boelkow, 1960 (VJ-101):



Bemerkung

An dieser Stelle sollten einige Dinge Erwähnung finden:

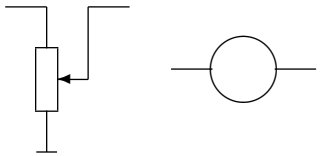
- Es geht *nicht* darum, solche Systeme wieder aufleben zu lassen! Sie gehören in ein Museum.[2]
- Das manuelle Patchfeld ist ein Relikt der Vergangenheit – so kann und will man sicherlich nicht im 21^{sten} Jahrhundert programmieren.
- Ein Digitalrechner kann – zumindest im Prinzip – jedes berechenbare Problem lösen, genügend Speicher und Zeit vorausgesetzt. Seine interne Struktur bleibt hierbei fest.
- Das kann ein Analogrechner nicht – er muss über soviele Rechenelemente verfügen, wie für die Modellbildung benötigt werden, d. h. er muss mit der Problemgröße mitwachsen!
- Dafür arbeitet ein Analogrechner mit im Wesentlichen konstanter Lösungszeit! Etwas, das der Digitalrechner nicht kann.

[2] `https://analogmuseum.org`

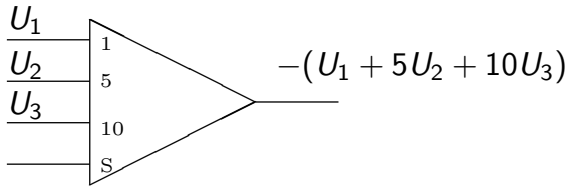
Programmierung

Typische Rechenelemente

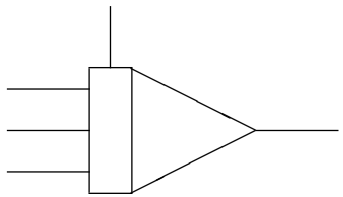
- Koeffizientenglieder (Spannungsteiler):



- Summierer, $-U_{\text{out}} = \sum_{i=1}^{n} a_i U_i$:



- Integrierer,[3] $-U_{\text{out}} = \int_0^t \sum_{i=1}^{n} a_i U_i(t)\,\mathrm{d}t + U(0)$

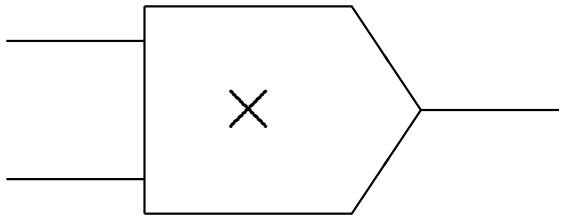


Ein Integrierer hat drei Betriebsarten: *Anfangswert* (*initial condition*), *Rechnen* (*operate*), *Halt*.

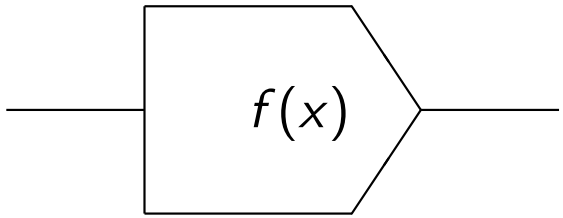
[3] Das „magische Element“– nahezu perfekte Integration mit einem Widerstand, einem Kondensator und einem Operationsverstärker.

Typische Rechenelemente

- Multiplizierer:



- *Offene Verstärker*, mit denen Umkehrfunktionen ($\frac{\cdots}{\cdots}$, $\sqrt{\ldots}$ etc.) gebildet werden können.
- Komparatoren für „Entscheidungen“
- Funktionsgeneratoren:



Die Programmierung eines Analogrechners ist sehr viel einfacher, als die eines Digitalrechners (auch, wenn es nicht so aussieht ;-)):

Die Verschaltung der Rechenelemente kann einfach aus den zu lösenden Gleichungen hergeleitet werden:

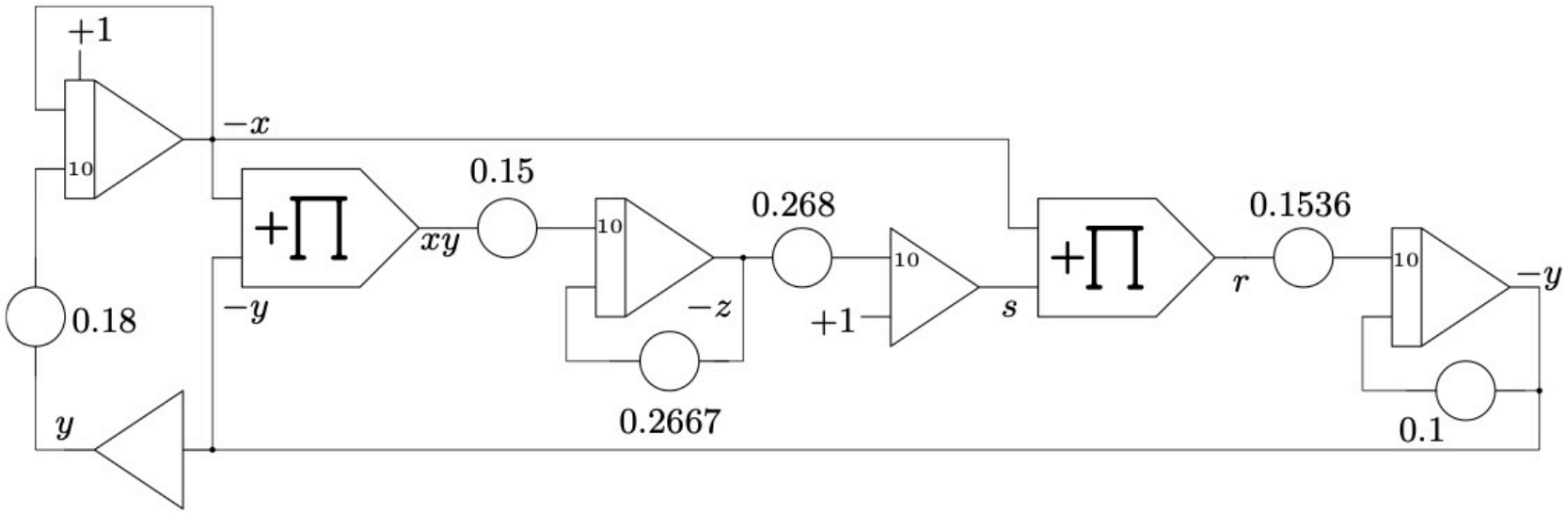
Dieses Verfahren geht auf Lord *Kelvin* zurück:

1. Löse die Gleichung(en) nach den höchsten Ableitungen auf.
2. Erzeuge alle benötigten niedrigeren Ableitungen durch Integrierer.
3. Baue die rechte Seite der Gleichung(en) zusammen. :-)
4. Dies entspricht gerade der höchsten Ableitung, kann also in den Eingang der Rechenschaltung *zurück geführt* werden.

Die folgenden Folien zeigen zwei Beispiele:

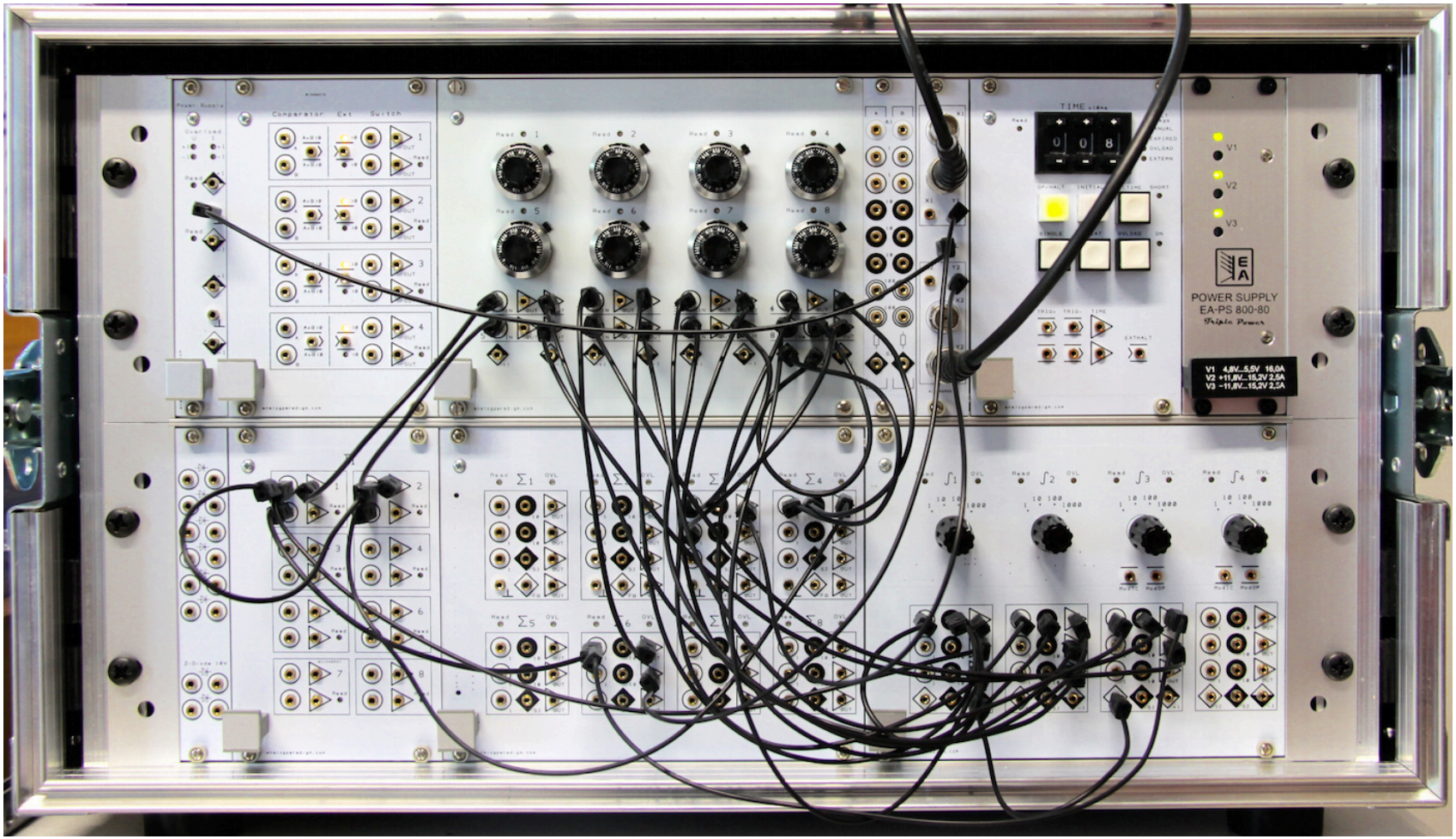
Das chaotische System überhaupt; 1963 von Edward Norton Lorenz entdeckt:[4]

$$\begin{aligned}\dot{x} &= \sigma(y - x)\\ \dot{y} &= x(\rho - z) - y\\ \dot{z} &= xy - \beta z,\end{aligned}$$



[4] Siehe [Lorenz, 1963].

Die klassische Implementation sieht so aus:

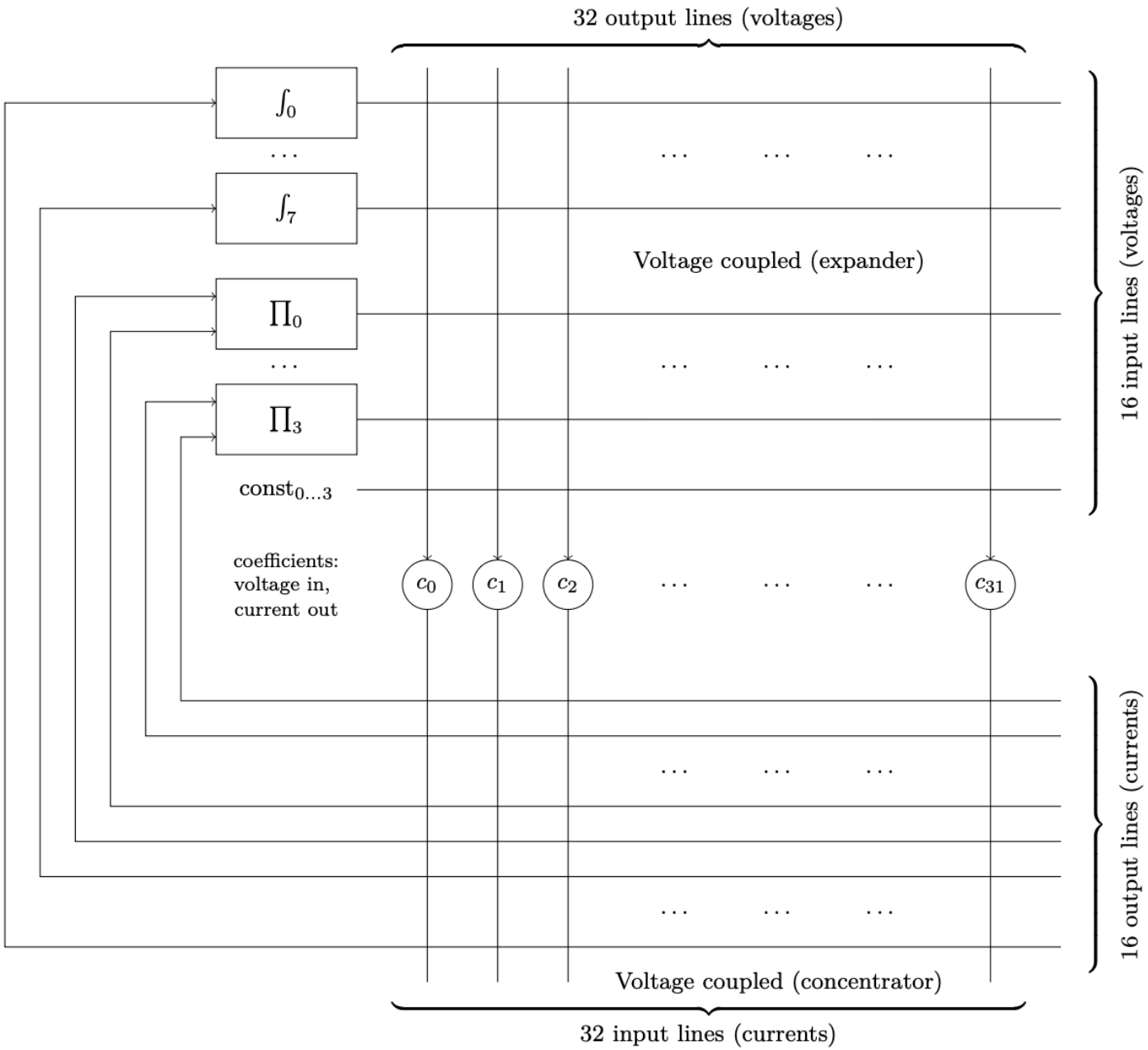


Autopatch

- Offensichtlich ist ein klassisches Patchpanel nicht allzu praktikabel. . .
- Es braucht ein *Autopatch*-System.
- Die Idee an sich ist nicht neu – schon der ROCKEFELLER Differential Analyzer hatte ein Autopatch-System, allerdings hatte er auch nur vergleichsweise wenige Rechenelemente.
- Problem: Eine $n \times m$-Schaltermatrix wird schnell unpraktikabel.
- Idee: Man kann die Tatsache ausnutzen, dass die meisten Schaltungen vergleichsweise dünn besetzt sind (analog zu einer klassischen Telefonvermittlung).

So sieht beispielsweise das Autopatch-System von anabrid aus:

Autopatch


32 output lines (voltages)
$\int_0$
$\int_7$
$\prod_0$
$\prod_3$
$\text{const}_{0\ldots3}$
Voltage coupled (expander)
16 input lines (voltages)
coefficients:
voltage in,
current out
c_0
c_1
c_2
c_{31}
16 output lines (currents)
Voltage coupled (concentrator)
32 input lines (currents)

Lorenz-Attraktor

Auf einem modernen, rekonfigurierbaren Analogrechner geht das schon sehr viel einfacher:

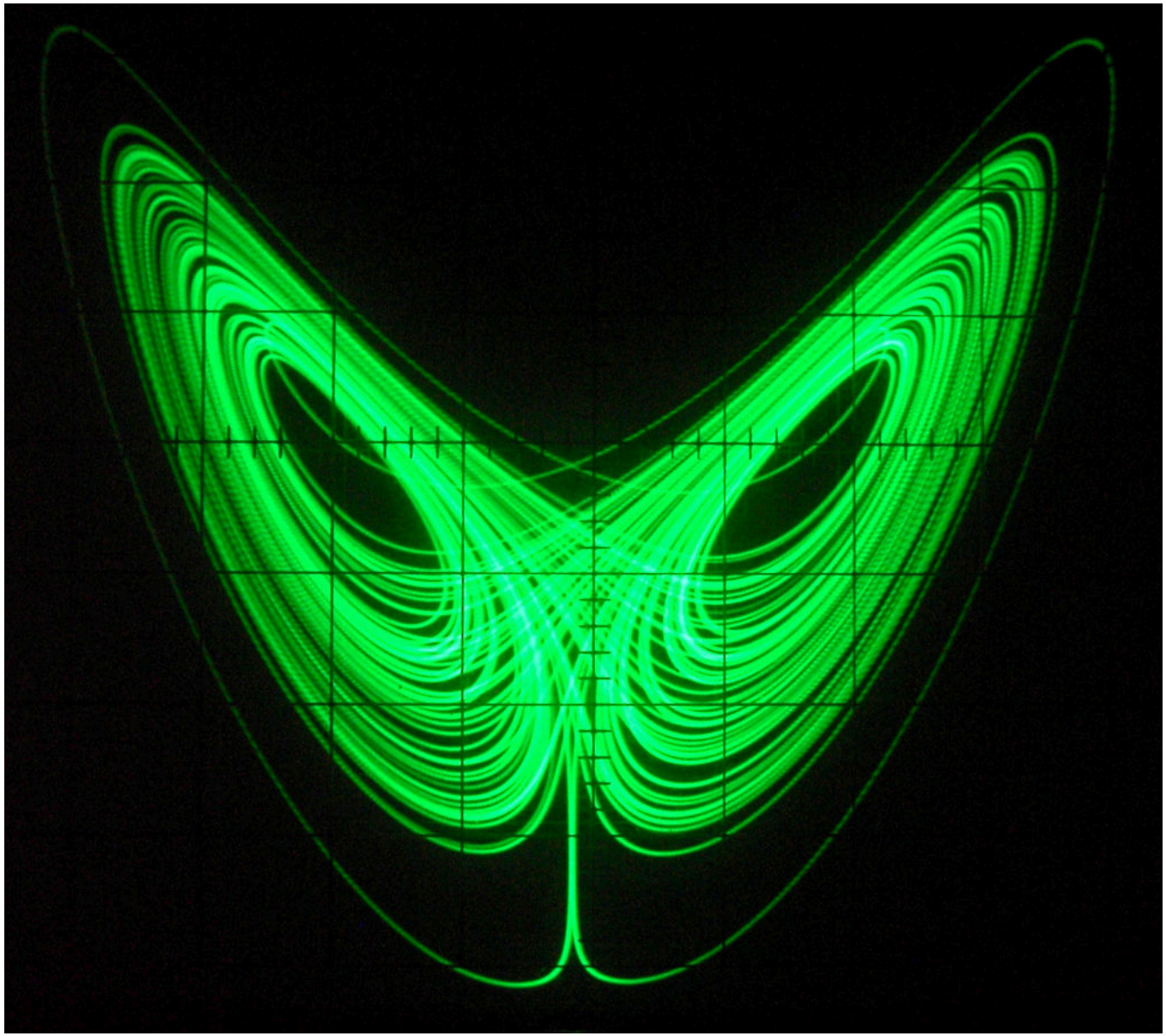
```
fn X(t);
fn Y(t);
fn Z(t);

let diff[X, t] = 1.8 * Y - X;
let diff[Y, t] = 1.56 * X * (1 - 2.678 * Z) - 0.1 * Y;
let diff[Z, t] = 1.5 * X * Y - 0.2667 * Z;

let X(t: 0) = 0.1;
let Y(t: 0) = 0.0;
let Z(t: 0) = 0.0;

plot(x: X(t), y: Y(t));
out X(t);
out Y(t);
```

Lorenz-Attraktor



Moderne Entwicklungen

Moderne Analog- und Hybridrechner müssen zunächst eine Reihe von Anforderungen erfüllen bzw. Probleme lösen:

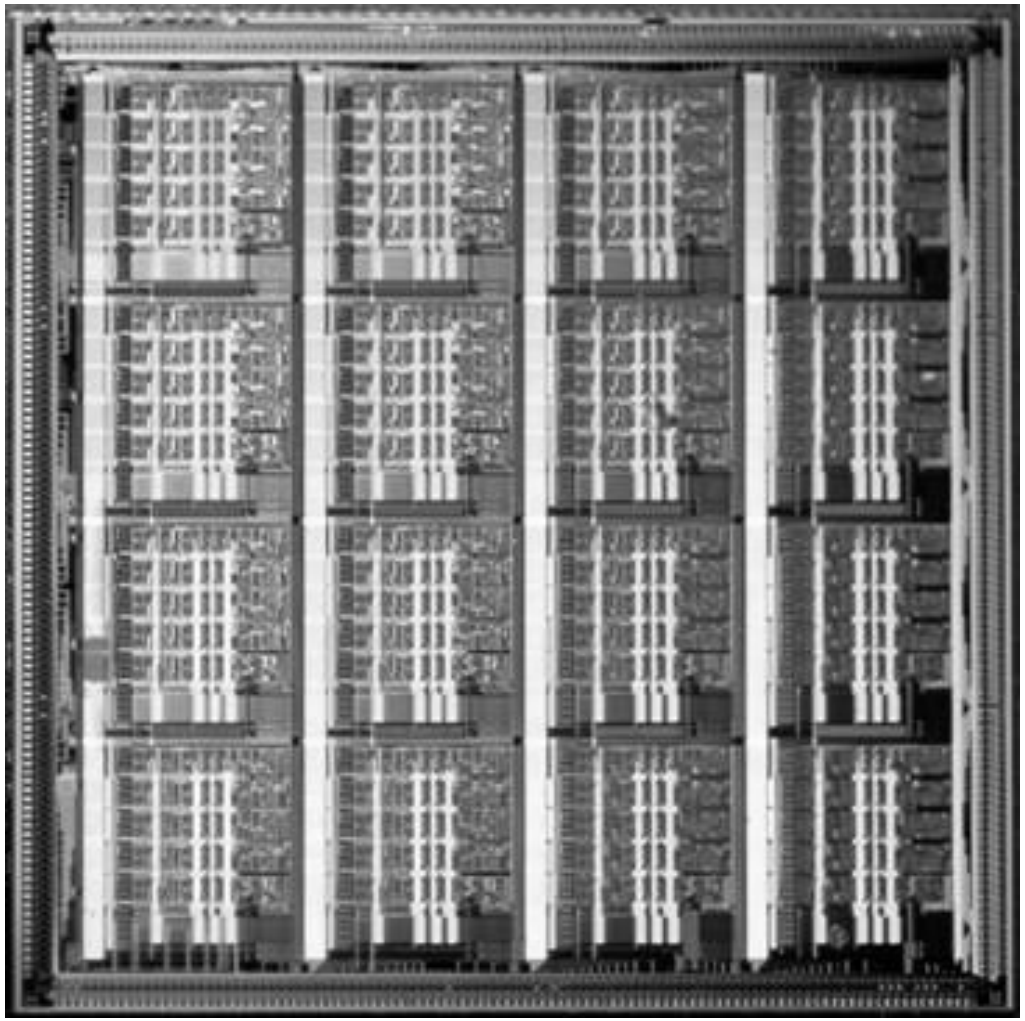
- Das Patchfeld gehört ins Museum – moderne Analogrechner müssen von einem Digitalrechner innerhalb maximal weniger Millisekunden rekonfigurierbar sein.
- Hierfür wird eine *Domain Specific Language* (*DSL*) mit Compiler, Entwicklungstools etc. benötigt, um Rechenschaltungen definieren zu können.
- Der Analogrechner muss „eng“ mit einem Digitalrechner gekoppelt werden, was
  - low-level Software (Devicetreiber etc. – Interruptlatenzen sind hier ein echtes Problem) sowie
  - Bibliotheken, neue numerische Verfahren etc. voraussetzt.
- Die Rechenelemente müssen eine hohe Bandbreite aufweisen (idealerweise einige 10 MHz bis 100 MHz), um extrem schnelle Lösungzeiten (max. einige μs) zu erzielen.

Die folgenden Folien geben einen kurzen Überblick über aktuelle Projekte im akademischen und kommerziellen Bereich:

Akademische Projekte

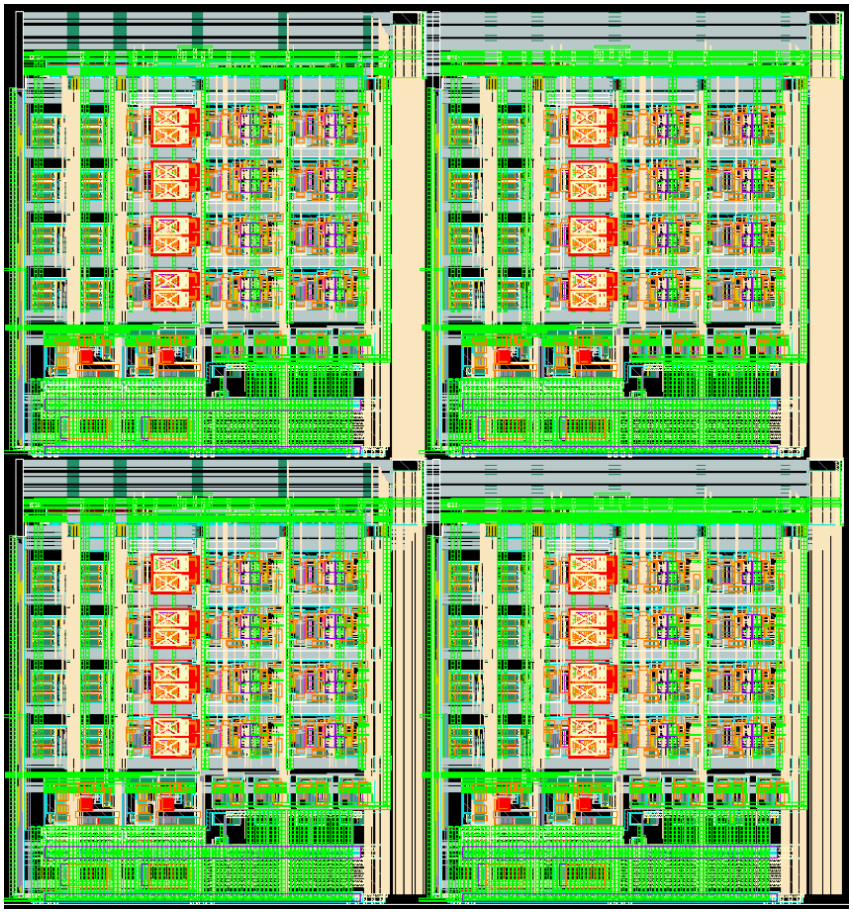
Glenn E. R. Cowan (2005)

Erster rekonfigurierbarer Analogrechner auf einem Chip:[5]



[5] Siehe [Cowan(2005)].

Weiterentwickelter Analogrechner auf einem Chip:[6]



(Bild: http://yipenghuang.com/wp-content/uploads/2016/07/v2_chip.png, abgerufen am 19.09.2019.)

[6] Siehe [Guo et al.(2016)].

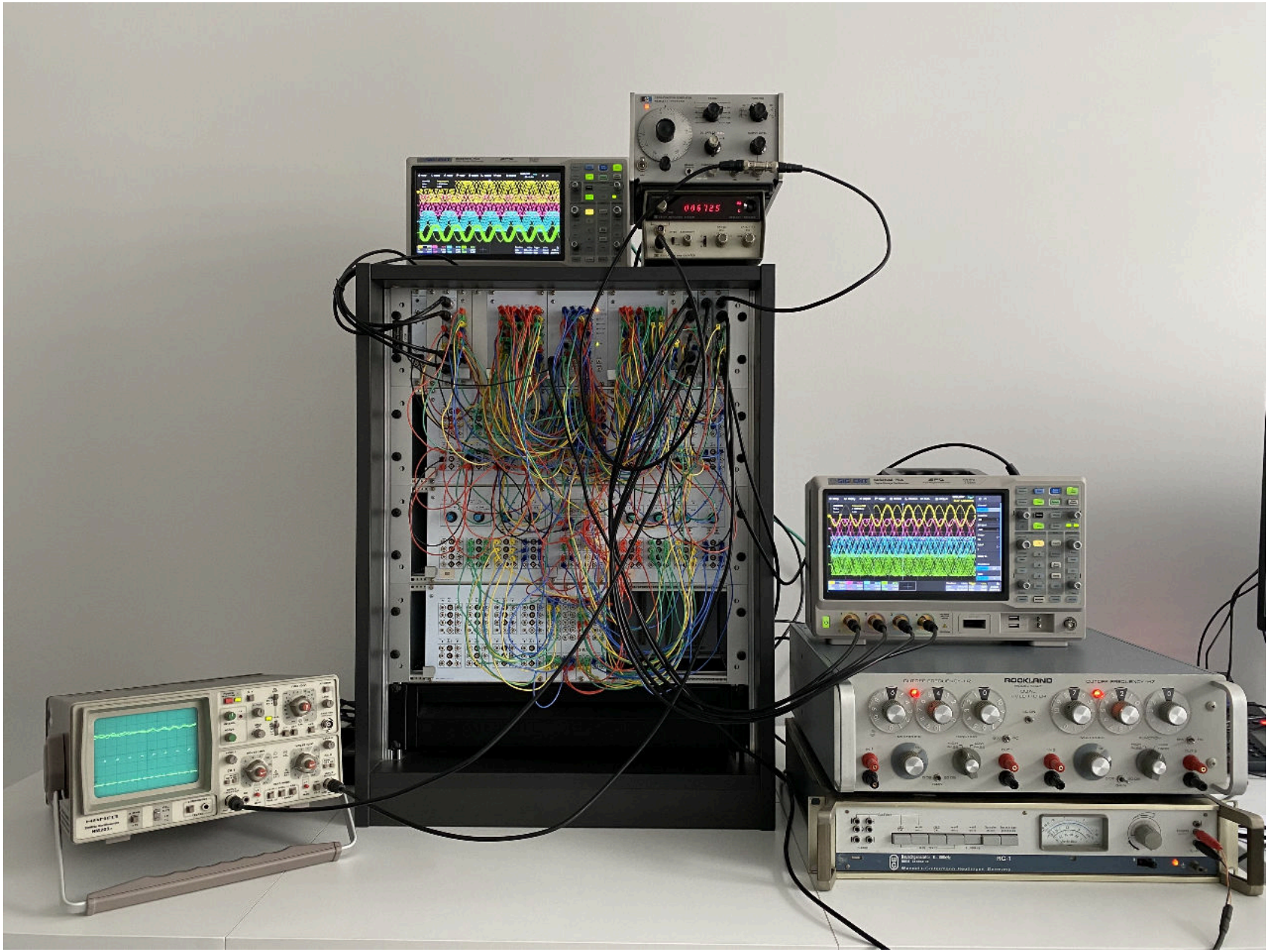
„Quantum inspired“ Analogrechnen

Osz.-basierte Ising-Maschinen

Mit Hilfe oszillatorbasierter ISING-Maschinen lassen sich eine Reihe „harter“ Probleme lösen.

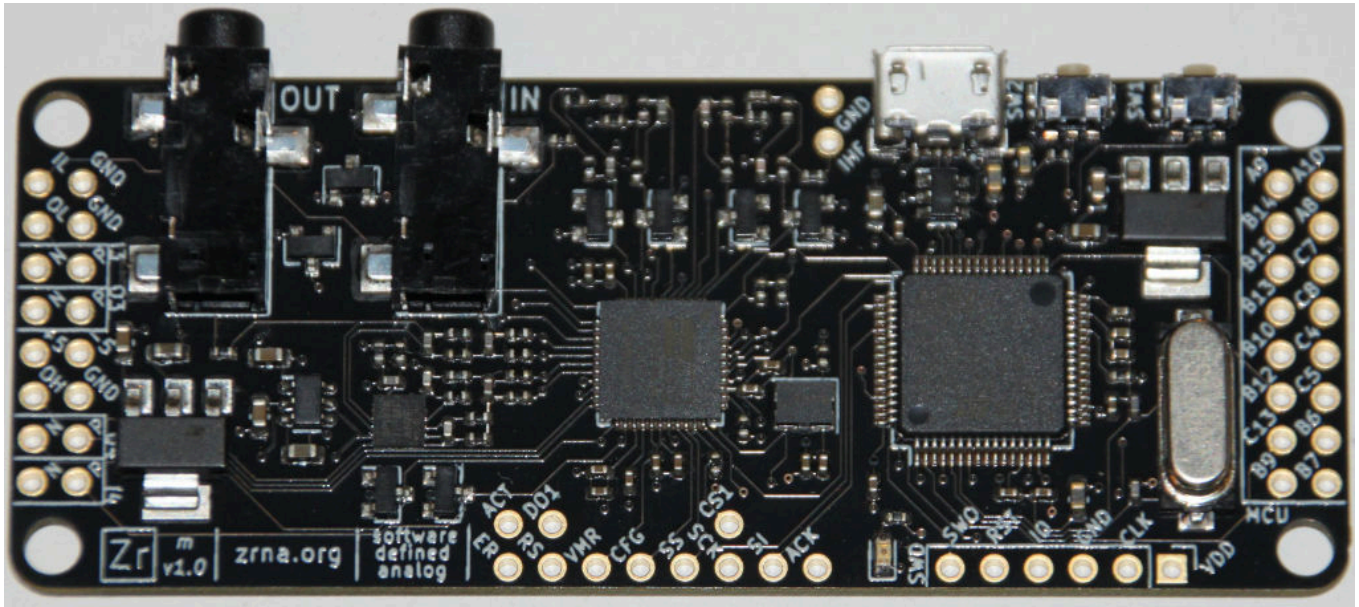
Die Idee ist hierbei, eine Vielzahl von Oszillatoren „schlau“ miteinander zu koppeln und ihre Phasenbeziehungen zur Repräsentation von Bits zu nutzen:

Osz.-basierte Ising-Maschinen



Anwendungsbereich *edge computing*

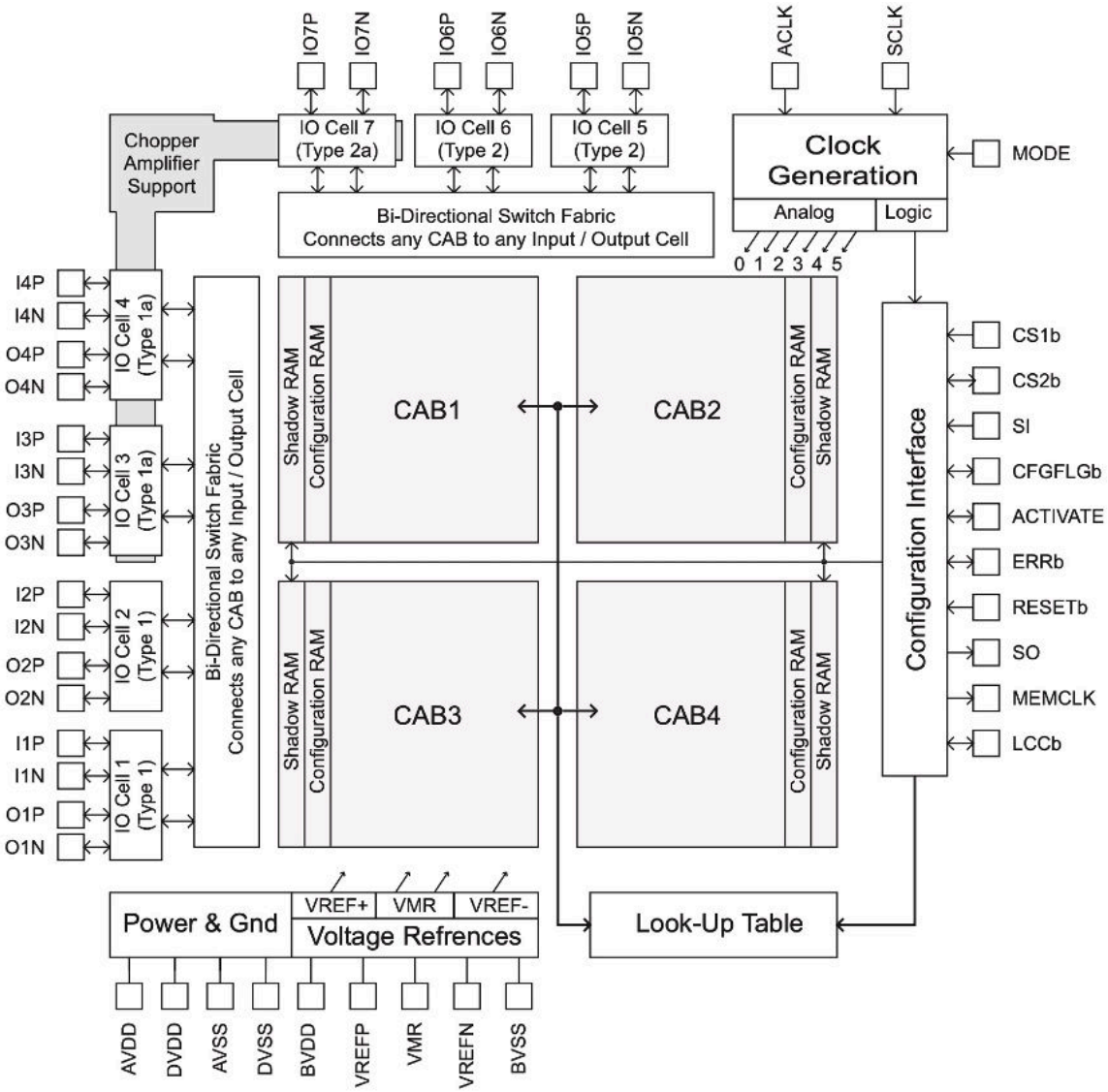
Field Programmable Analog Arrays (Anadigm): Rekonfigurierbare Filter etc., z. B. für Audioanwendungen:[7]



Das FPAA besteht aus *configurable analog blocks* (*CAB*s), die über *switched capacitors* miteinander verschaltet werden.

[7] Gezeigt ist ein Entwicklungsboard von ZRNA, `https://zrna.org`.

Okika



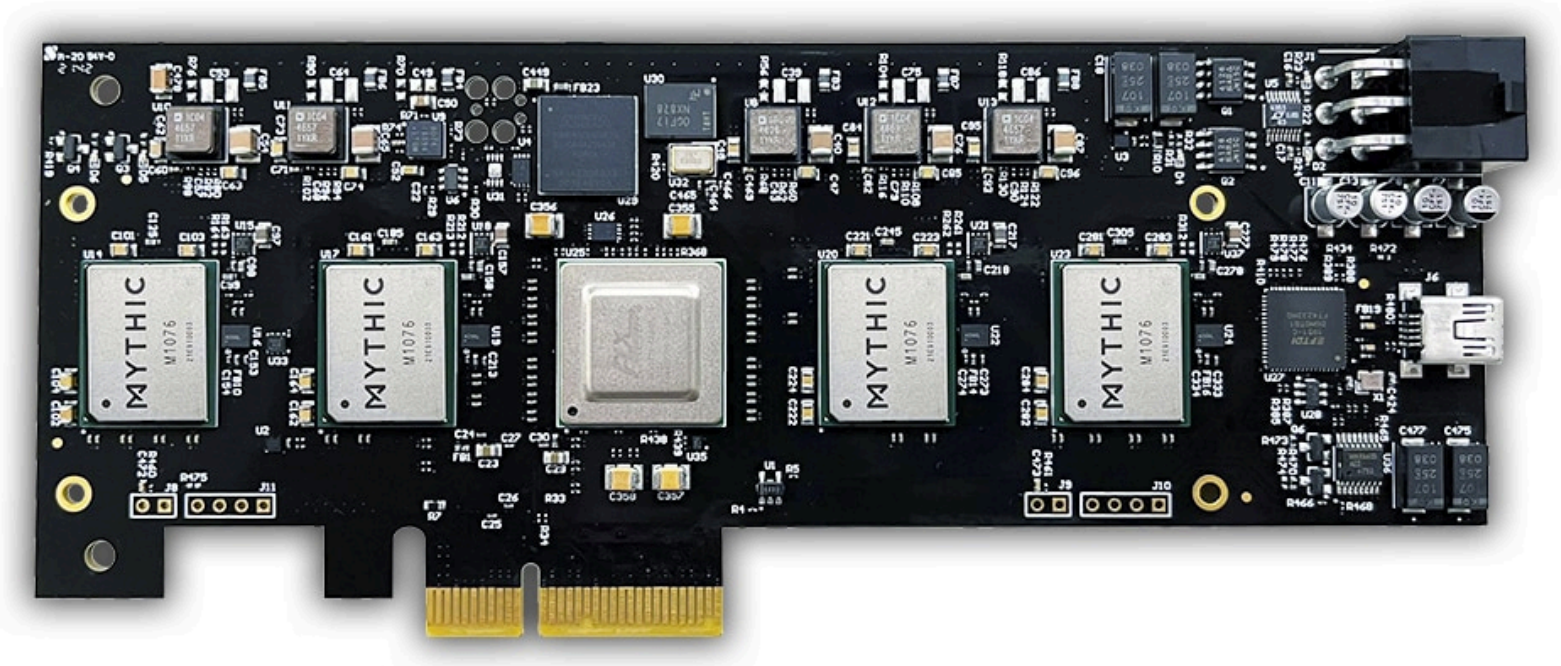
(Siehe. [Anadigm(2006)].)

Anwendungsbereich künstliche Intelligenz

Der gesamte Bereich der KI ist wohl der kommerziell interessante Anwendungsbereich für moderne Analogrechner:

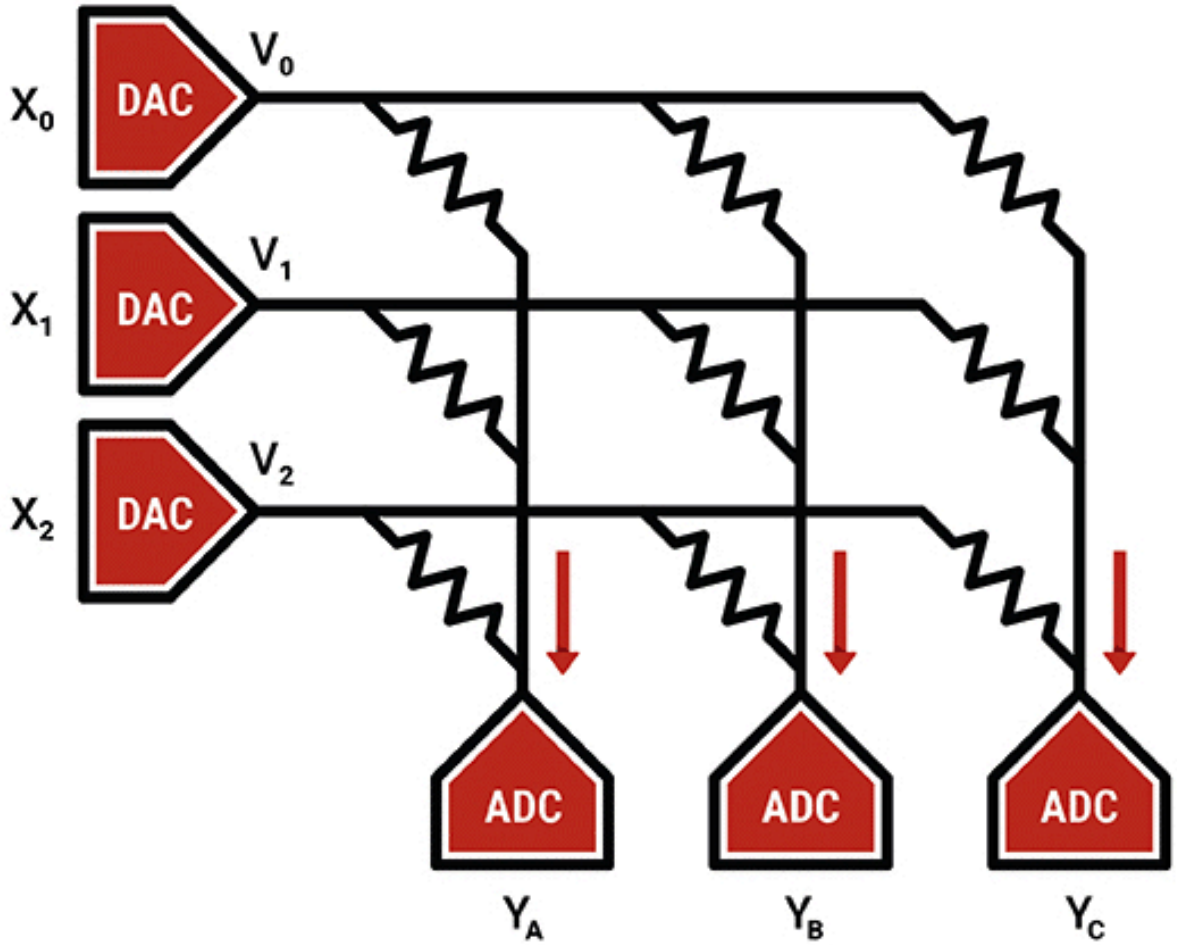
- Im einfachsten Fall lassen sich linearalgebraisch Operationen auf Matrizen und Vektoren mit Hilfe von Widerstandsnetzwerken implementieren (eher langweilig :-)).
- Alternativ kann man auch deutlich biologienähere Modelle von Neuronen analog implementieren, die beispielsweise spikendes oder burstendes Verhalten aufweisen (neuromorphic computing – schon deutlich interessanter :-)).
- Sehr aktuell sind Ansätze, Netzwerke aus gedämpften harmonischen Oszillatoren aufzubauen, um neuronale Netze zu implementieren (das ist ausgesprochen interessant :-)).

MYTHICs[8] *analog matrix processors* (*compute in memory*) für KI-Anwendungen:



[8] `https://mythic.ai`, abgerufen am 25.09.2022.

MYTHIC



Genutzt werden hier Flash-Speicherzellen für die Kopplungsgewichte.

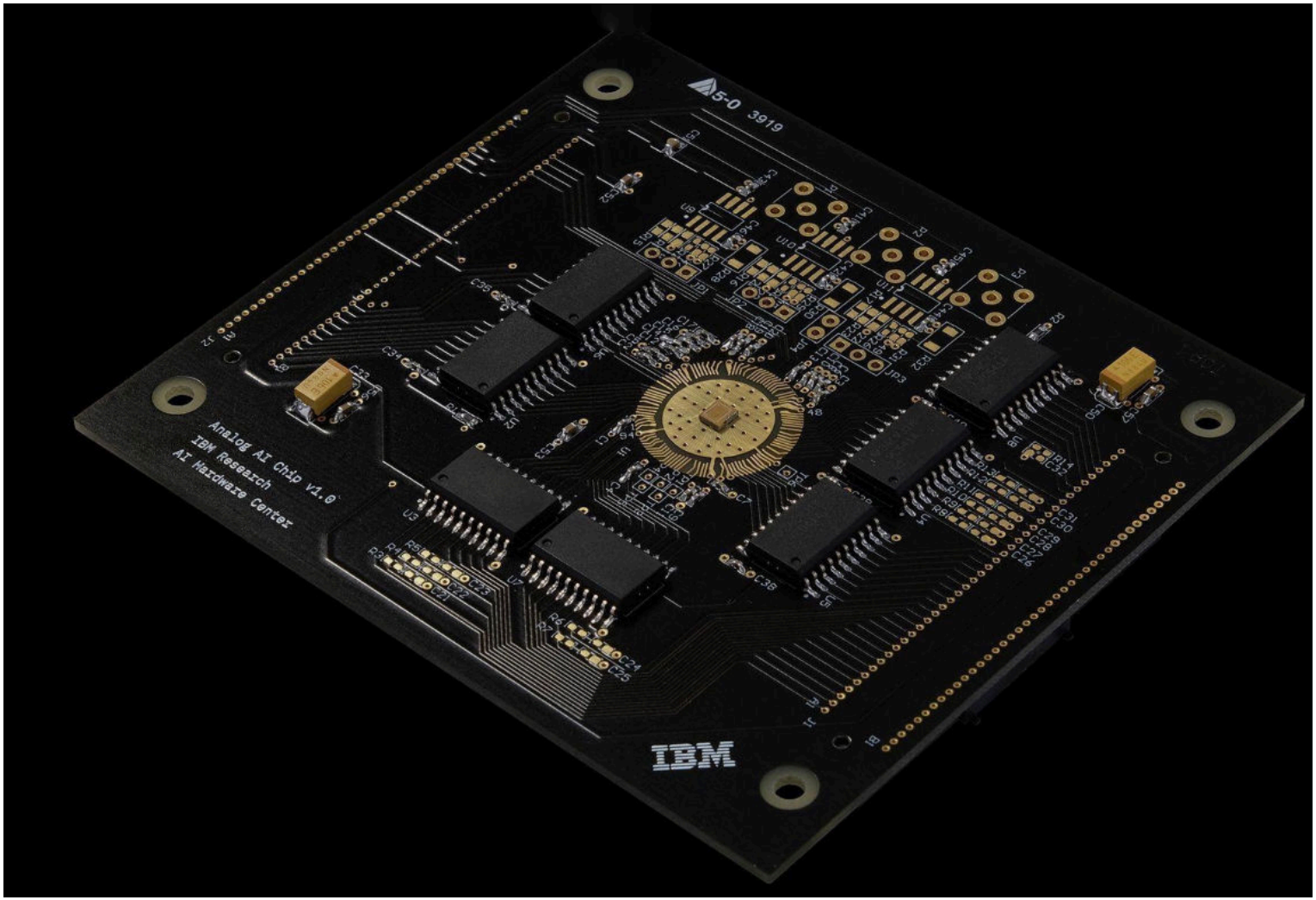
`https://mythic.ai/technology/analog-computing/`, abgerufen am 26.06.2023.

Auch IBM hat einen analogen Co-Prozessor für KI-Anwendungen entwickelt:[9]

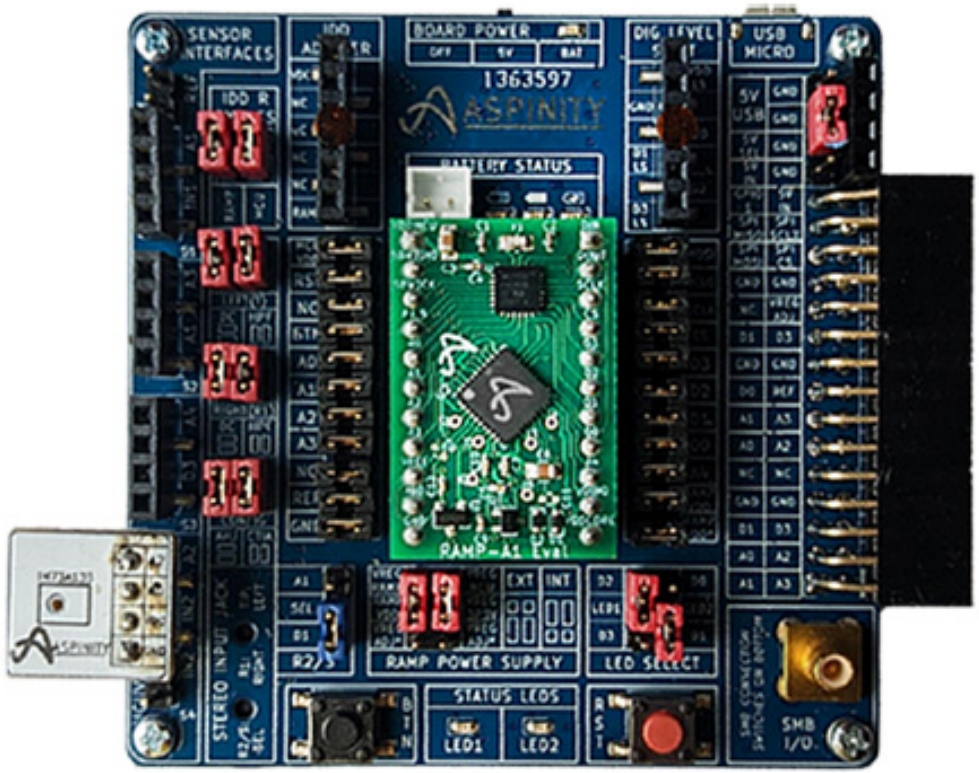
- Die synaptischen Gewichte werden hier als *Resistive RAM* (*ReRAM*) in Form von *phase change memory* implementiert (der Zustand der Zellen wird zwischen amorph und kristallin umgeschaltet).
- Dieser experimentelle Chip beinhaltet $35 \cdot 10^6$ solcher Speicherzellen.

[9] `https://research.ibm.com/blog/the-hardware-behind-analog-ai`, abgerufen am 25.09.2022.

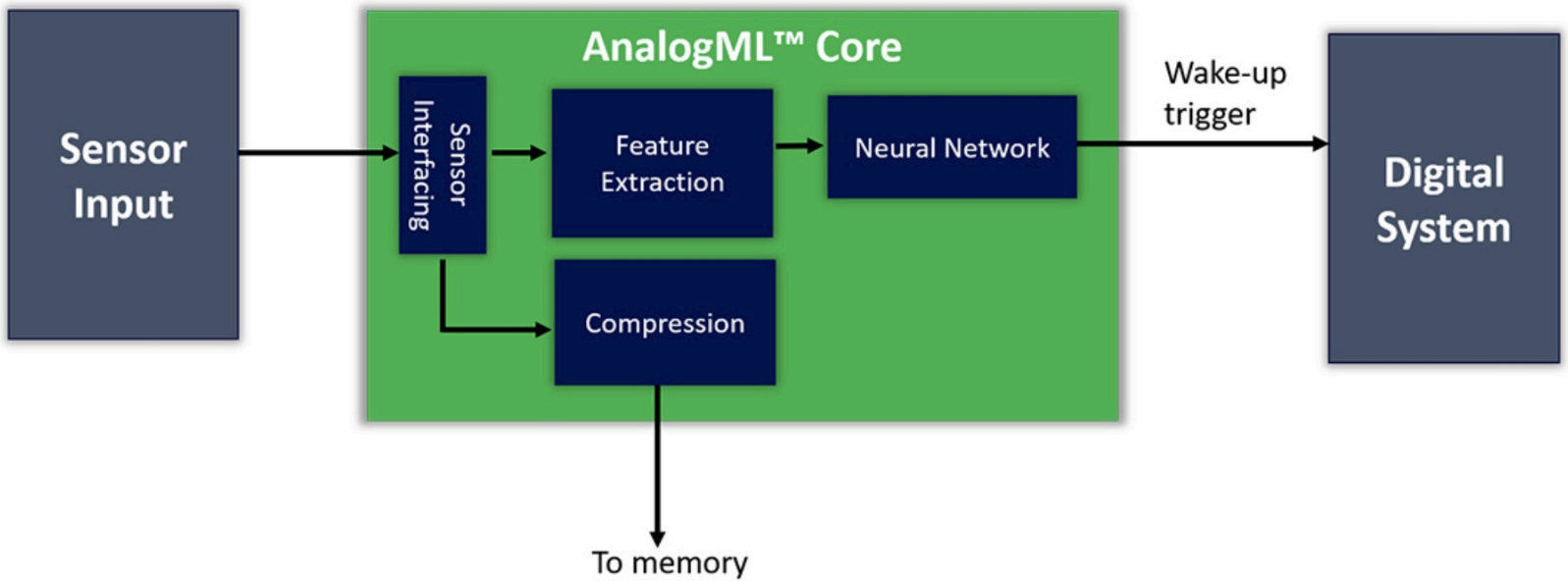
IBM



Triggerworterkennung, Glasbrucherkennung,Vibrationsanalysen etc.:

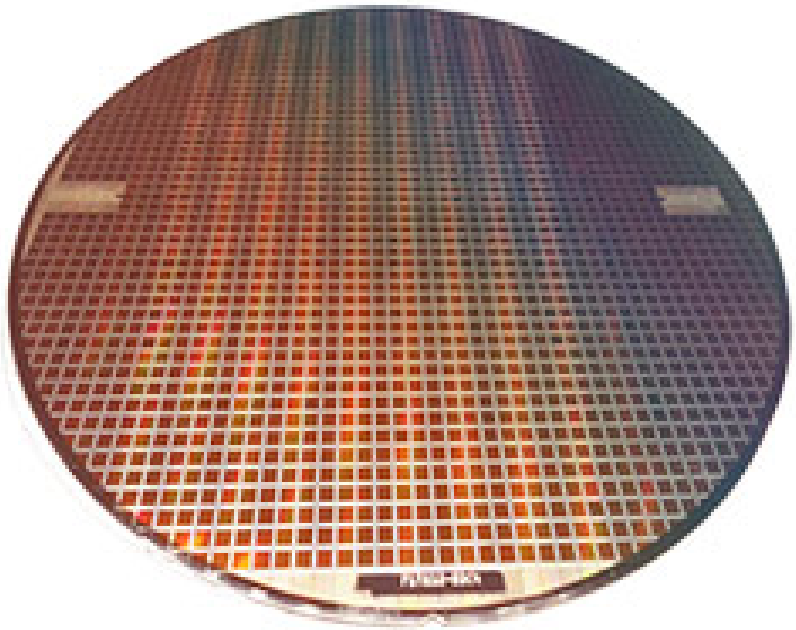


Aspinity



https://www.aspinity.com/AnalogML-Core, abgerufen am 26.06.2023.

Reconfigurable Analog Modular Processor (*RAMP*) für „neuromorphes Rechnen“[10]



[10] Siehe `https://www.aspinity.com/RAMP-Technology`, abgerufen am 26.06.2023.

Ein einfaches, aber dennoch interessantes Neuronenmodell geht auf HINDMARSH und ROSE zurück:[11]

$$\dot{x} = -ax^3 + bx^2 + y - z + I_{\text{ext}} \tag{1}$$

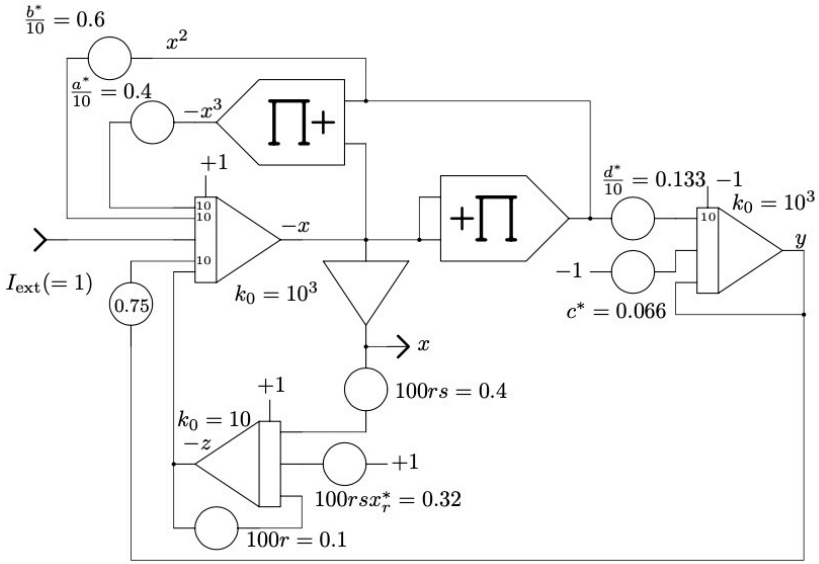
$$\dot{y} = -dx^2 + c - y \tag{2}$$

$$\dot{z} = r(s(x - x_r) - z) \tag{3}$$

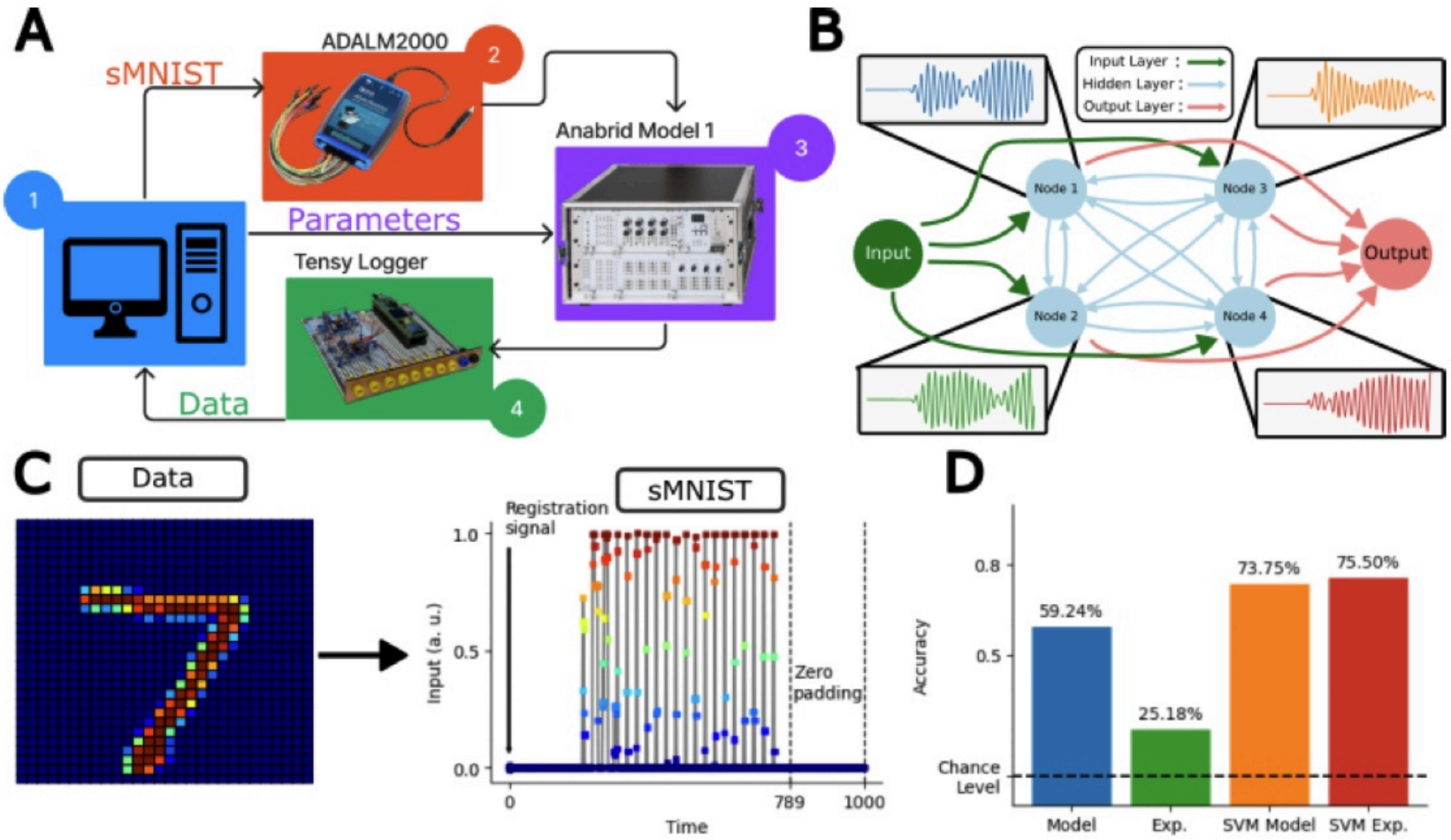
mit den Parametern $a = 1$, $b = 3$, $c = 1$, $d = 5$, $r = 10^{-3}$, $s = 4$ und $x_r = -\frac{8}{5}$.

[11] Siehe [HINDMARSH et al. 1982], [HINDMARSH et al. 1984], [IZHIKEVICH 2007, pp. 123 ff.] sowie [ULMANN 2021].

Spiking neurons



Siehe [Carvalho et al. 2025]:



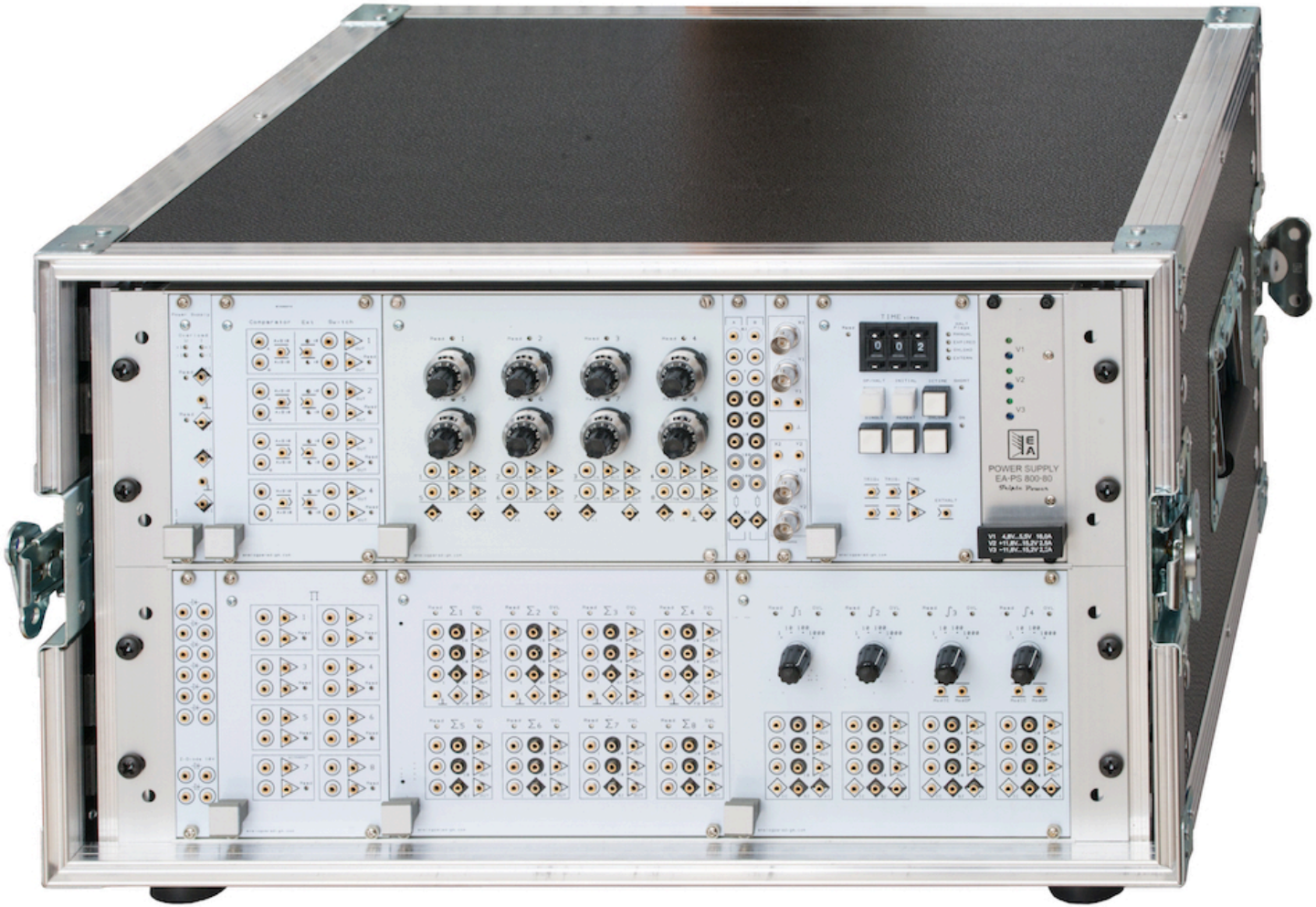
Anwendungsbereich *general purpose computing*

Die anabrid GmbH (Germany) wurde 2020 mit dem Ziel gegründet, moderne, frei programmierbare und hochintegrierte Analogrechner zu entwickeln.

Entwicklungen:

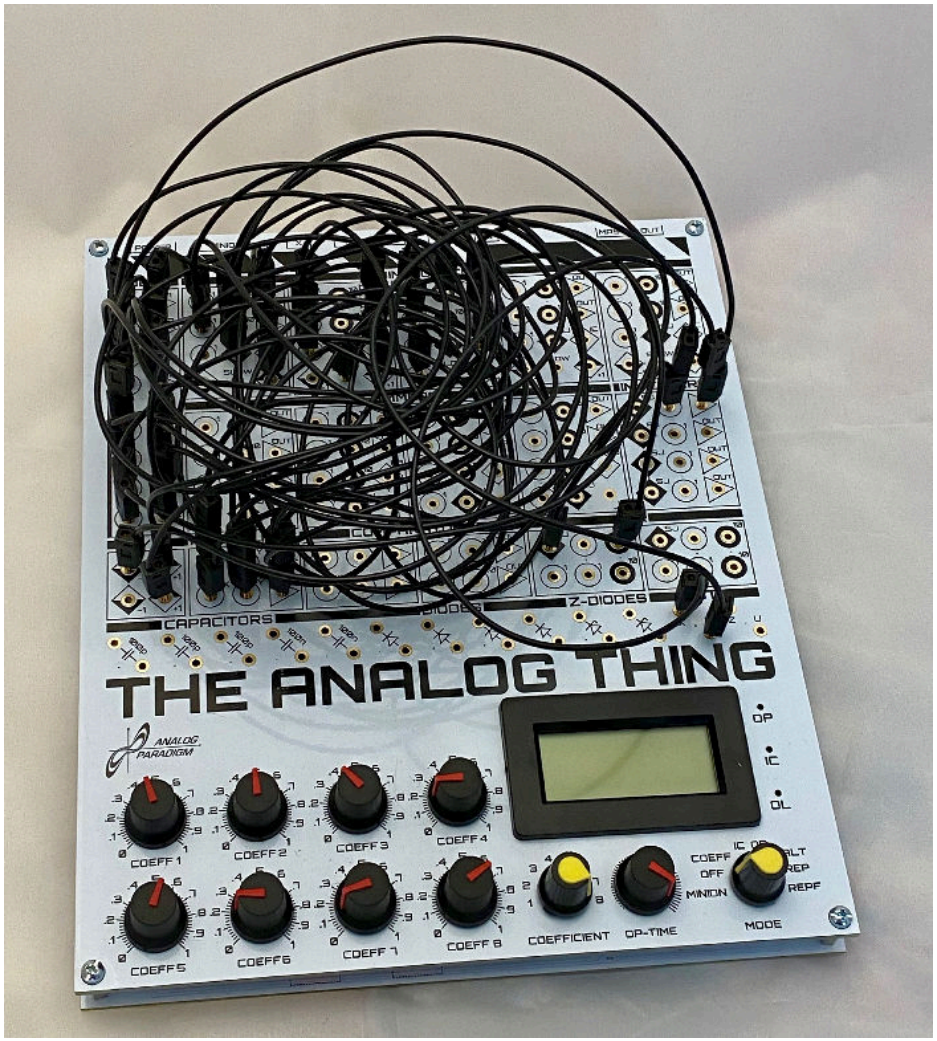
- Interconnect-Strukturen für große Analog- und Hybridrechner
- Hochsprache zur Programmierung mit Compiler und Softwarestack
- Hard- und Softwareunterstützung für High-Performance Hybridrechner
- Numerische Verfahren
- Quantum-inspired oszillatorbasierte Ising-Maschinen
- Rechenelemente in 65 nm CMOS-Technologie

Das anabrid Model-1 – ein klassischer Analogrechner für Lehr- und Forschungszwecke:[12]



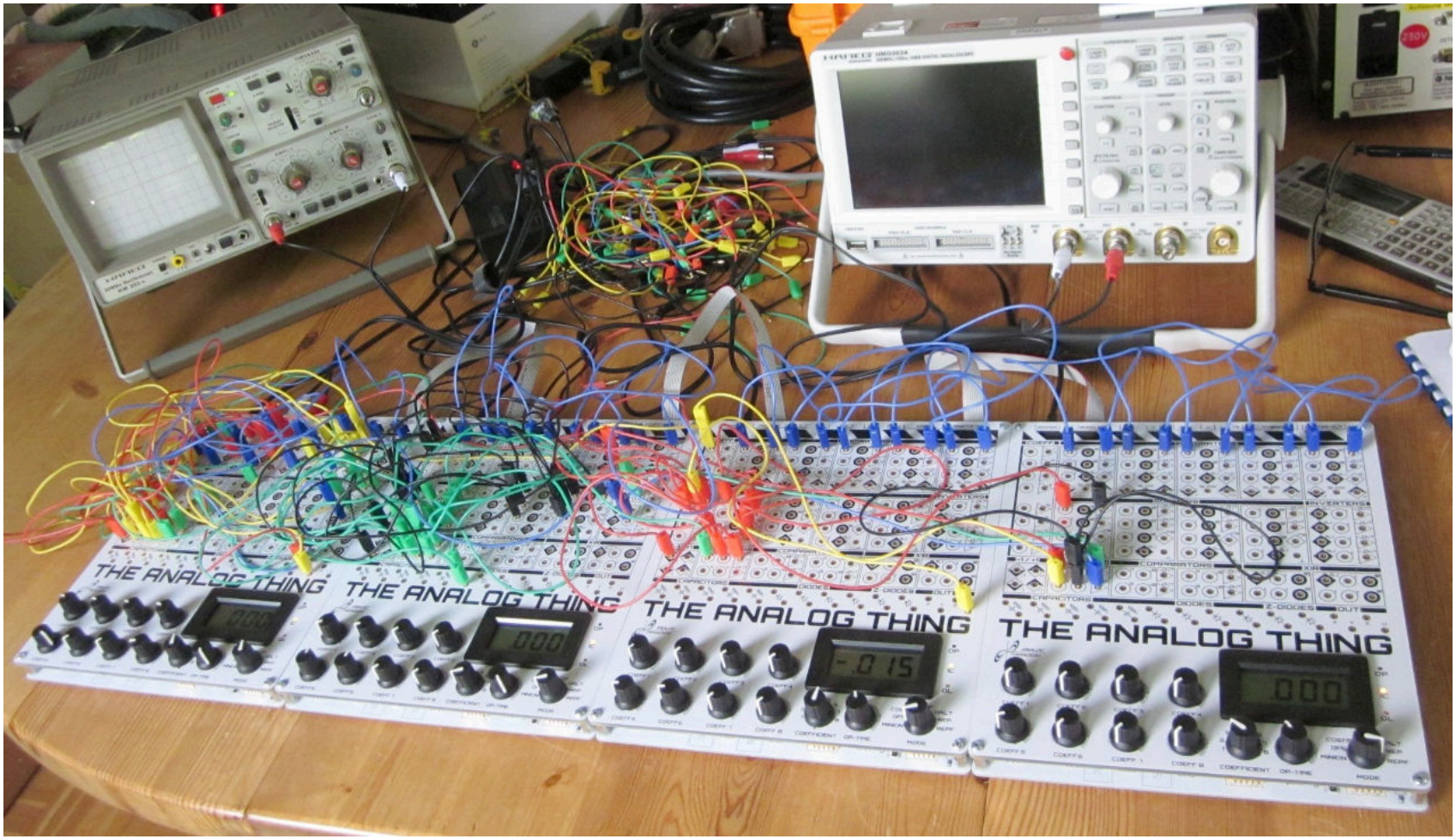
[12] Siehe [Ulmann(2019)].

Kleiner Analogrechner für Ausbildung und Lehre (*open hardware*):[13]



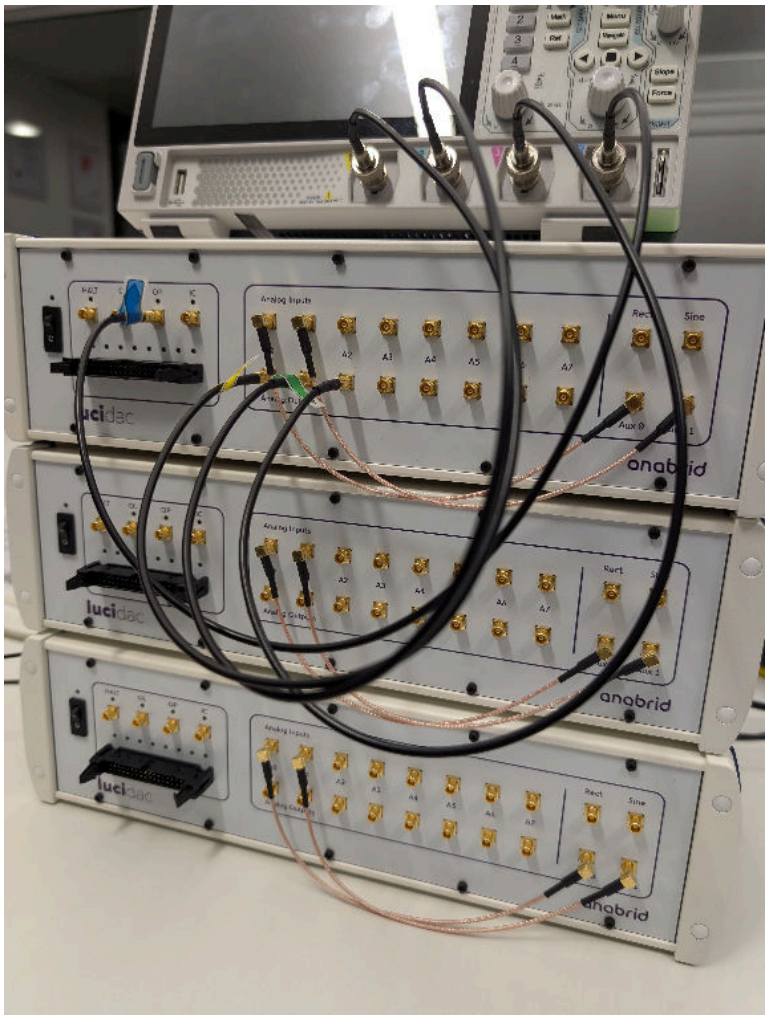
[13] anabrid GmbH

THE ANALOG THING



LUCIDAC

Rekonfigurierbarer Analogrechner, kein Patchfeld mehr erforderlich:[14]

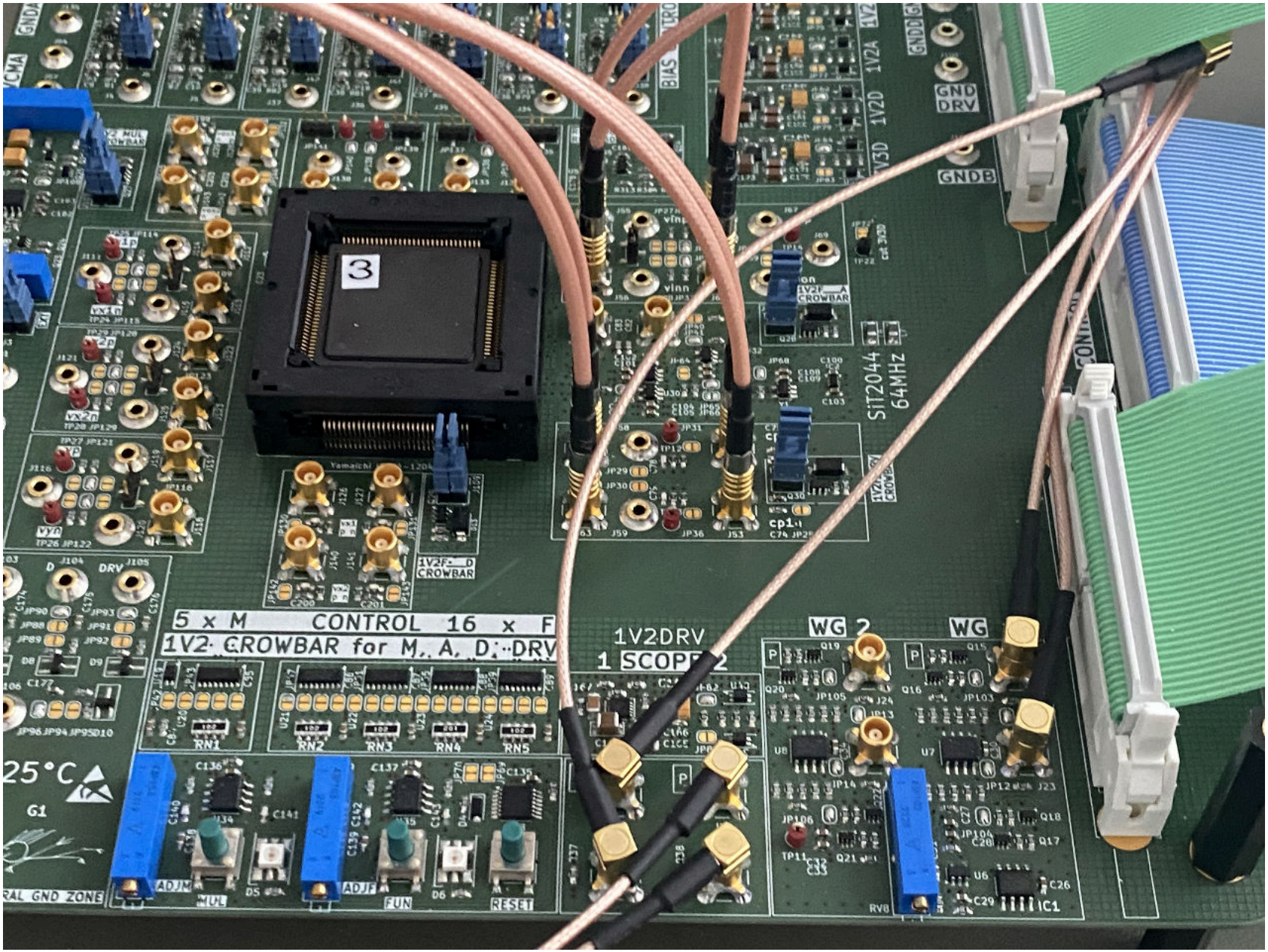


[14] anabrid GmbH

REDAC

REDAC – der analoge Supercomputer in der Cloud: 1000 Integrierer, 500 Multiplizierer, 4000 Koeffizienten, 100% rekonfigurierbar.

65 nm CMOS


3
5 x M CONTROL 16 x F
1V2 CROWBAR for M, A, D, DRV
1V2DRV
WG 2
WG
GNDB
GND DRV
SIT2044 64MHz
25°C
G1
ADJM
MUL
ADJF
FUN
RESET
RN1
RN2
RN3
RN4
RN5
IC1

Vielen Dank für Ihre Aufmerksamkeit!

Happy analog computing!

Der Autor ist unter folgender Adresse erreichbar:
ulmann@anabrid.com.

Bibliography

Bibliography


Anadigm, *AN13x/AN23x series, AnadigmApex dpASP Family User Manual*


Bureau of Ordnance Publication (Hrsg.), *Torpedo Data Computer, Mark 3, Mods. 5 to 12 inclusive*, June, 1944


Pedro Carvalho, Bernd Ulmann, Wolf Singer, Felix Effenberger, "An analog-electronic implementation of a harmonic oscillator recurrent neural network", preprint, `https://arxiv.org/pdf/2509.04064`


Greg Cope, *The Aizawa Attractor*, `http://www.algosome.com/articles/aizawa-attractor-chaos.html`, retrieved 15.12.2018


Glenn Edward Russell Cowan, *A VLSI Analog Computer / Math Co-processor for a Digital Computer*, Columbia University, 2005


Daniel Etiemble, *45-year CPU evolution: one law and two equations*, https://www.researchgate.net/publication/323510528_45-year_CPU_evolution_one_law_and_two_equations, abgerufen am 08.06.2023


N. Guo, Y. Huang, T. Mai, S. Patil, C. Cao, M. Seok, S. Sethumadhavan, and Y. Tsividis, "Energy-Efficient Hybrid Analog/Digital Approximate Computation in Continuous Time", in *IEEE Journal of Solid-State Circuits*, vol. 51, no. 7, pp. 1514-1524, July 2016


J. L. Hindmarsh, R. M. Rose, "A model of the nerve impulse using two first-order differential equations", in *Nature*, Vol. 296, 11 March 1982, pp. 162–164

Bibliography

- J. L. Hindmarsh, R. M. Rose, "A model of neuronal bursting using three coupled first order differential equations", in *Prov. R. Soc. Lond.*, B 221, 87–102 (1984)
- Eugene M. Izhikevich, *Dynamical systems in neuroscience: the geometry of excitability and bursting*, MIT Press, 2007
- Edward Norton Lorenz, "Deterministic Nonperiodic Flow", in *Journal of the Atmospheric Sciences*, Volume 20, pp. 130–141, March 1963
- Andrew Lucas, "Ising formulations of many NP problems", `https://arxiv.org/abs/1302.5843`, abgerufen am 05.06.2024
- [Meccano 1934/2] N. N., "Machine Solves Mathematical Problems – A Wonderful Meccano Mechanism", in *Meccano Magazine*, Vol. XIX, No. 6, June, 1934, pp. 442–444

Bibliography


[Philbrick 1948] George A. Philbrick, "Designing Industrial Controllers by Analog", in *Electronics*, June, 1948, pp. 108–111


Bernd Ulmann, *Model-1.1 – User manual*, `https://analogparadigm.com/downloads/handbook.pdf`, abgerufen am 21.01.2025


Bernd Ulmann, *The Hindmarsh-Rose model of neuronal bursting*, Analog Computer Applications, `https://analogparadigm.com/downloads/alpaca_28.pdf`, abgerufen am 14.01.2021

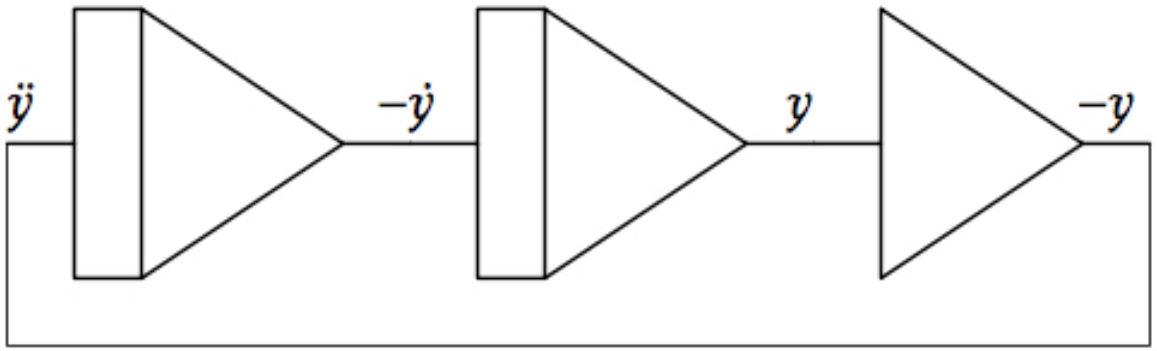

Bernd Ulmann, *Analog and hybrid computer programming*, DeGruyter, 2nd edition, 2023

„Hello world!“

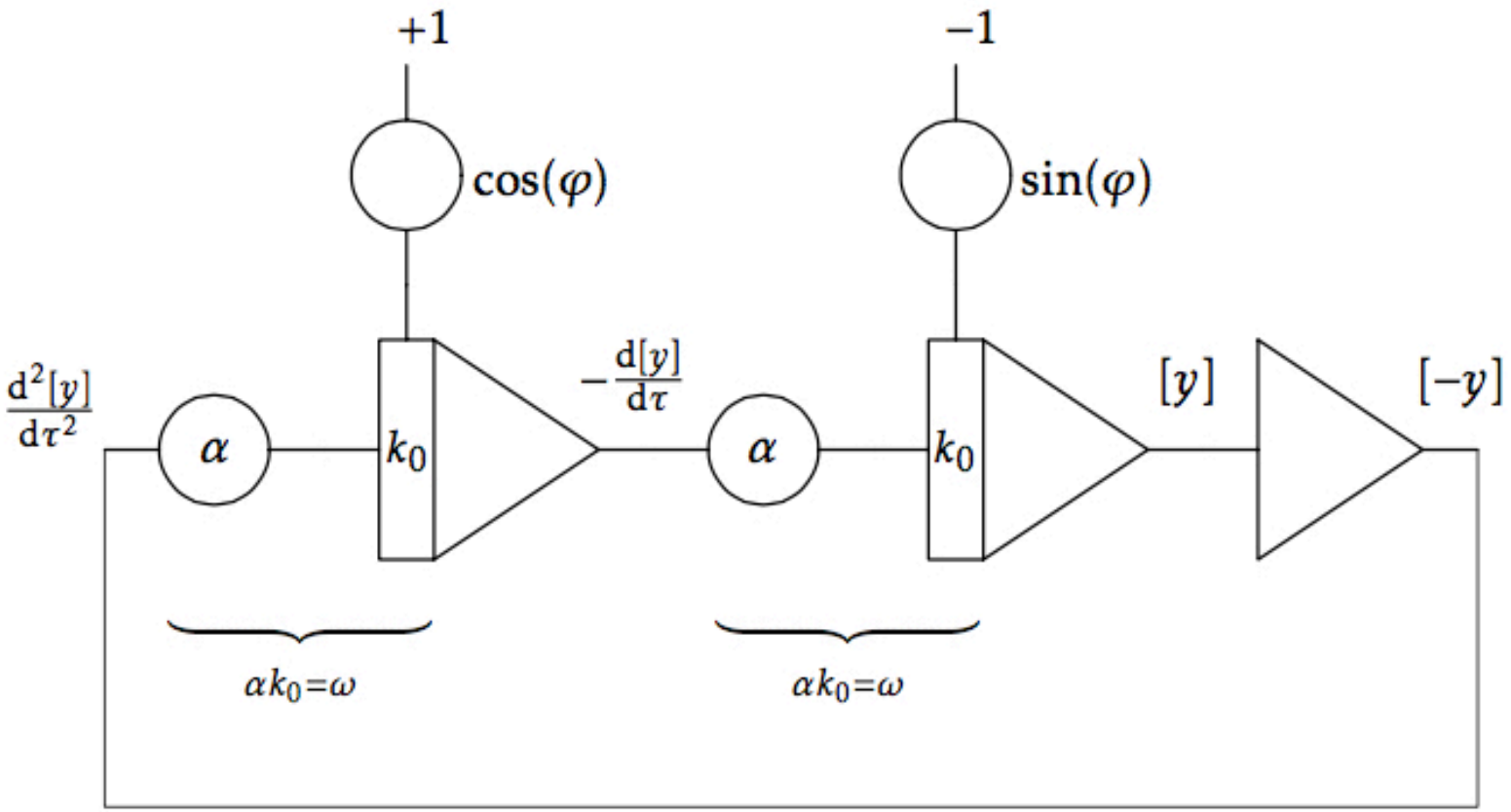
Das „Hello world!“ für Analogrechner berechnet Sinus und Cosinus als Lösung der Differentialgleichung

$$\ddot{y} = -y$$

mit geeigneten Anfangswerten, was zu folgendem ersten Programmansatz führt:

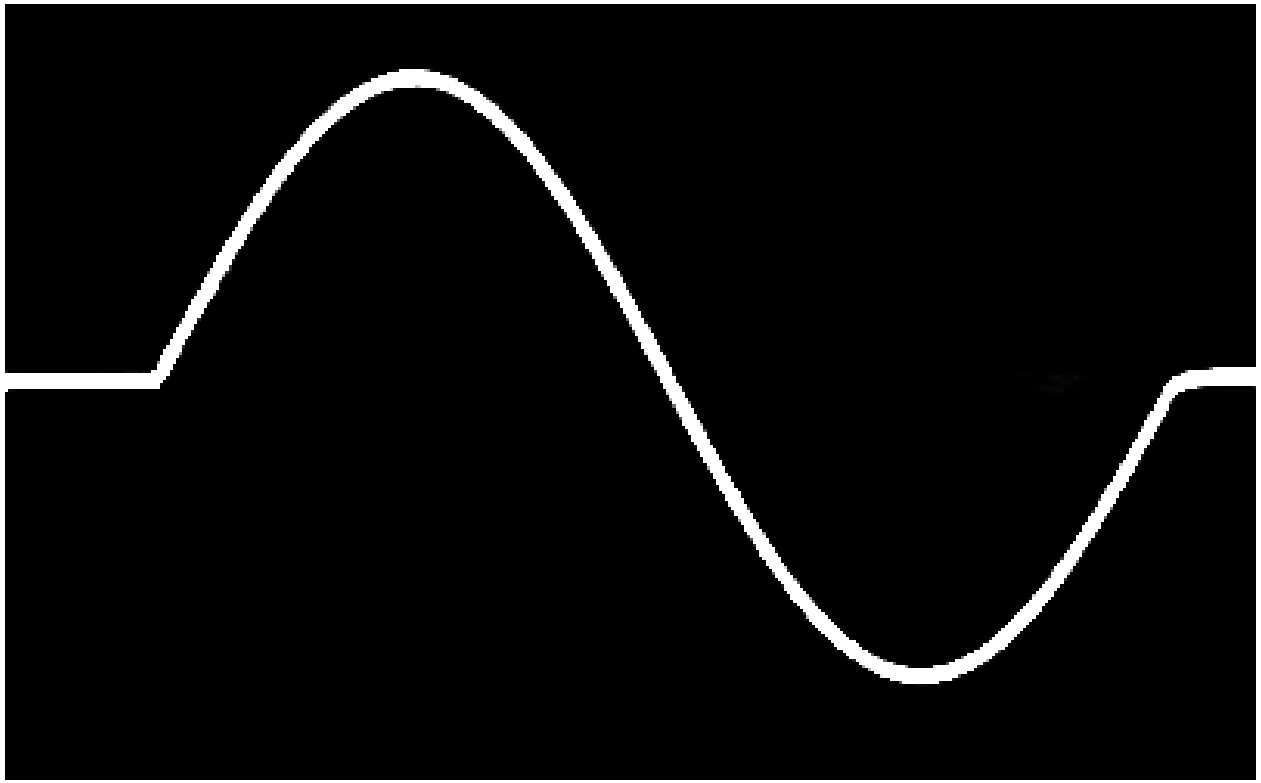


Nun benötigt man nur noch Anfangsbedingungen, um zu einem vollständigen Programm zu kommen:



Ergebnis

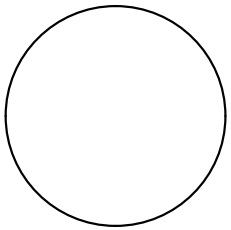
Dieses Programm liefert folgendes Ergebnis:

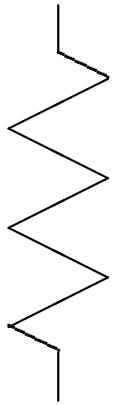


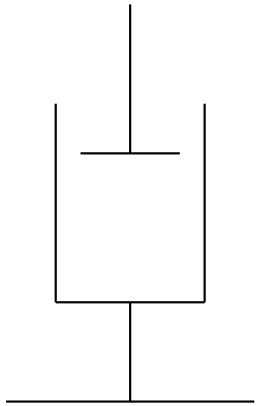
Masse-Feder-Dämpfer-System

Masse-Feder-Dämpfer-System

Ein Masse-Feder-Dämpfer-System ist zu simulieren:

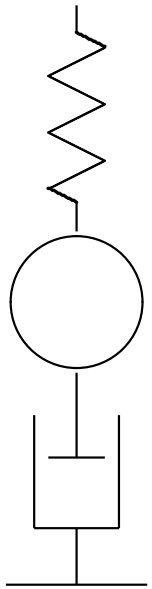






$$F_m = ma = m\ddot{y} \qquad F_s = sy \qquad F_d = d\dot{y}$$

Masse-Feder-Dämpfer-System



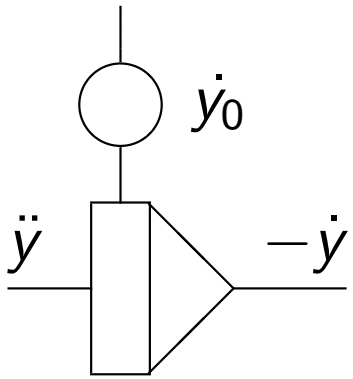
$$F_m + F_d + F_s = 0$$

$$m\ddot{y} + d\dot{y} + sy = 0$$

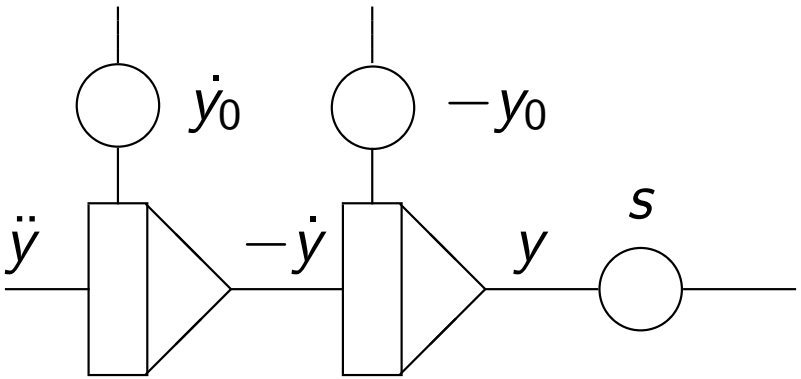
Daraus ergibt sich:

$$\ddot{y} = \frac{-(d\dot{y} + sy)}{m}.$$

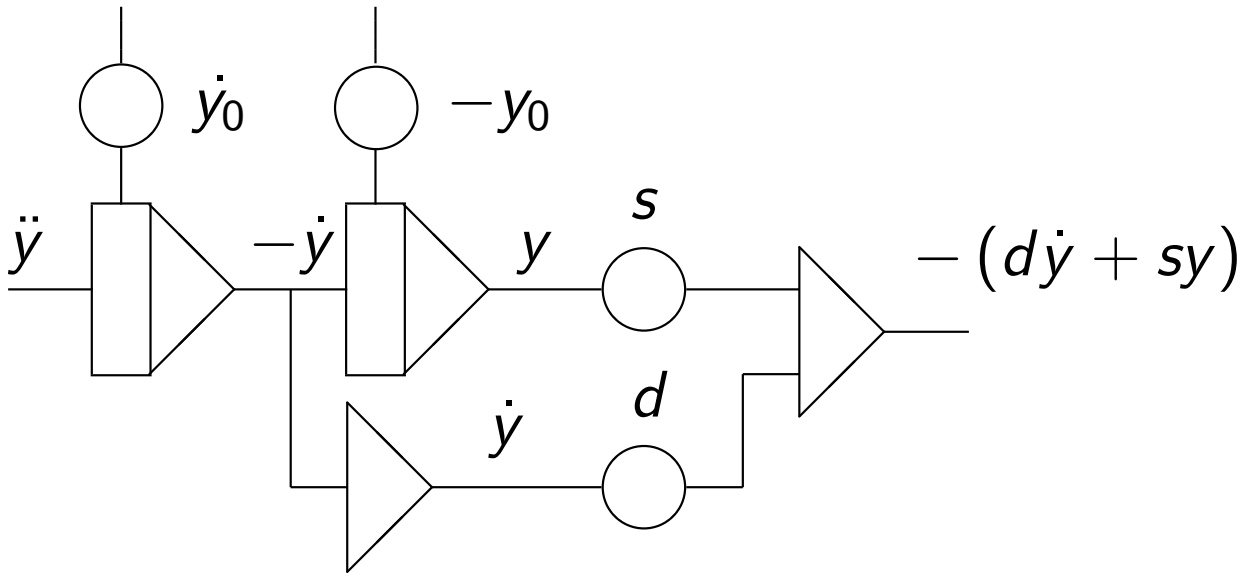
Eine erste Integration liefert:



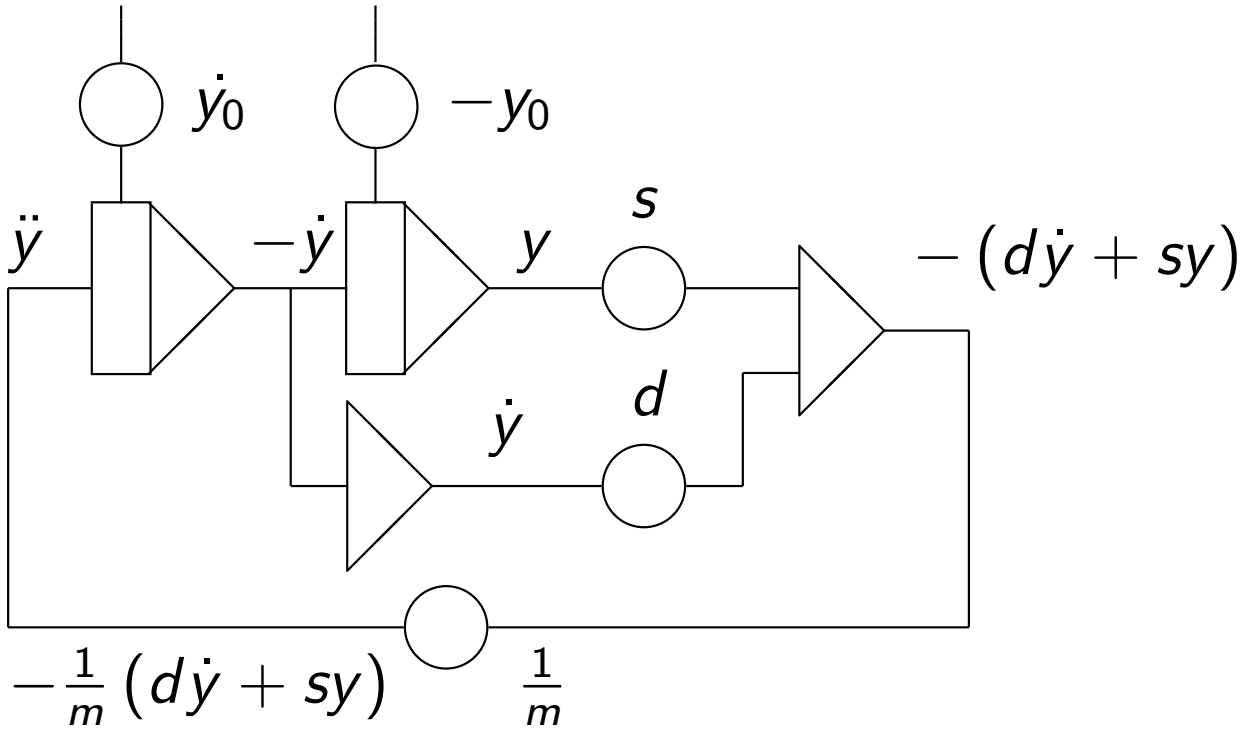
Erneutes Integrieren ergibt:



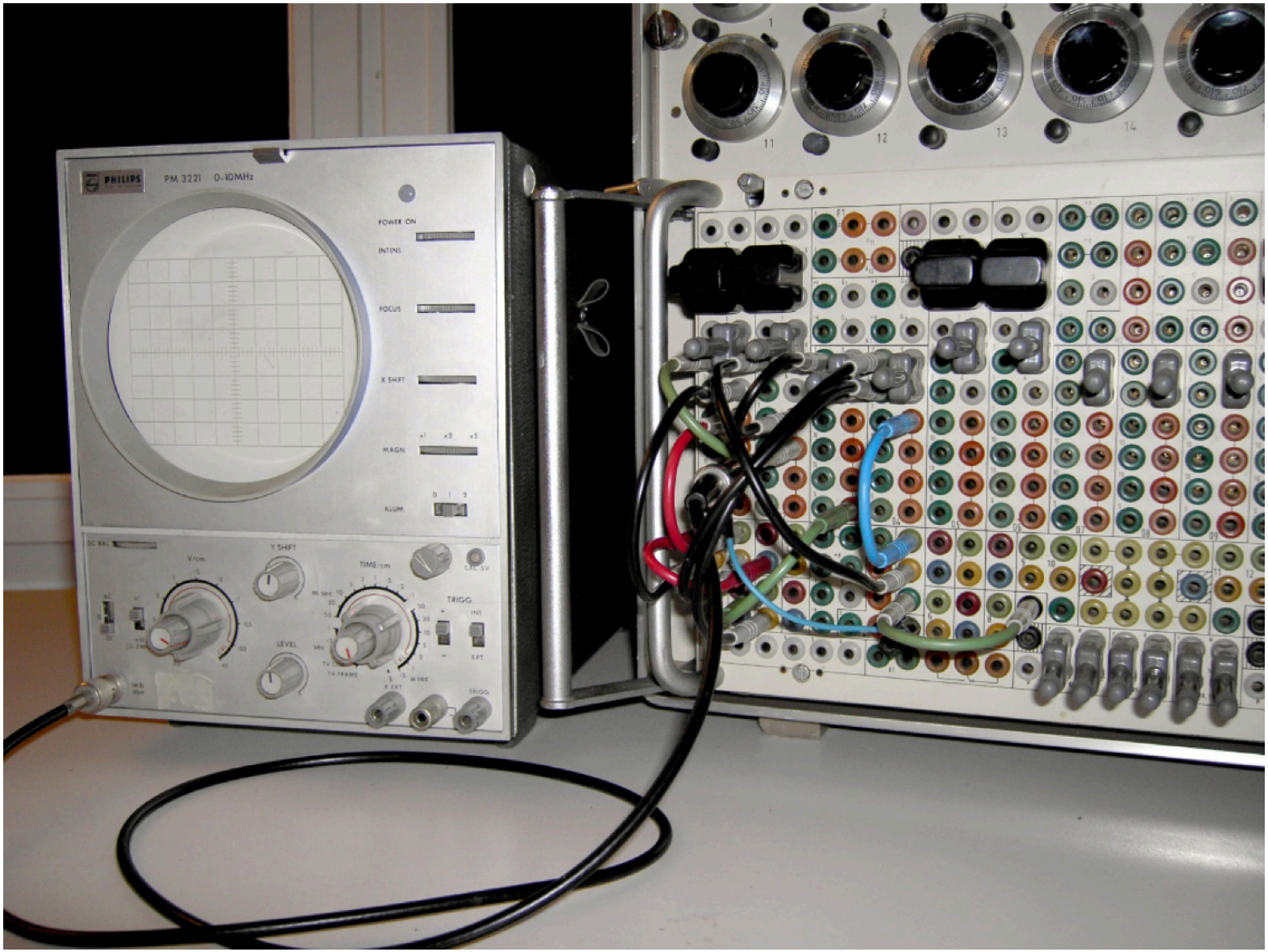
Erzeugen von $-(d\dot{y} + sy)$:



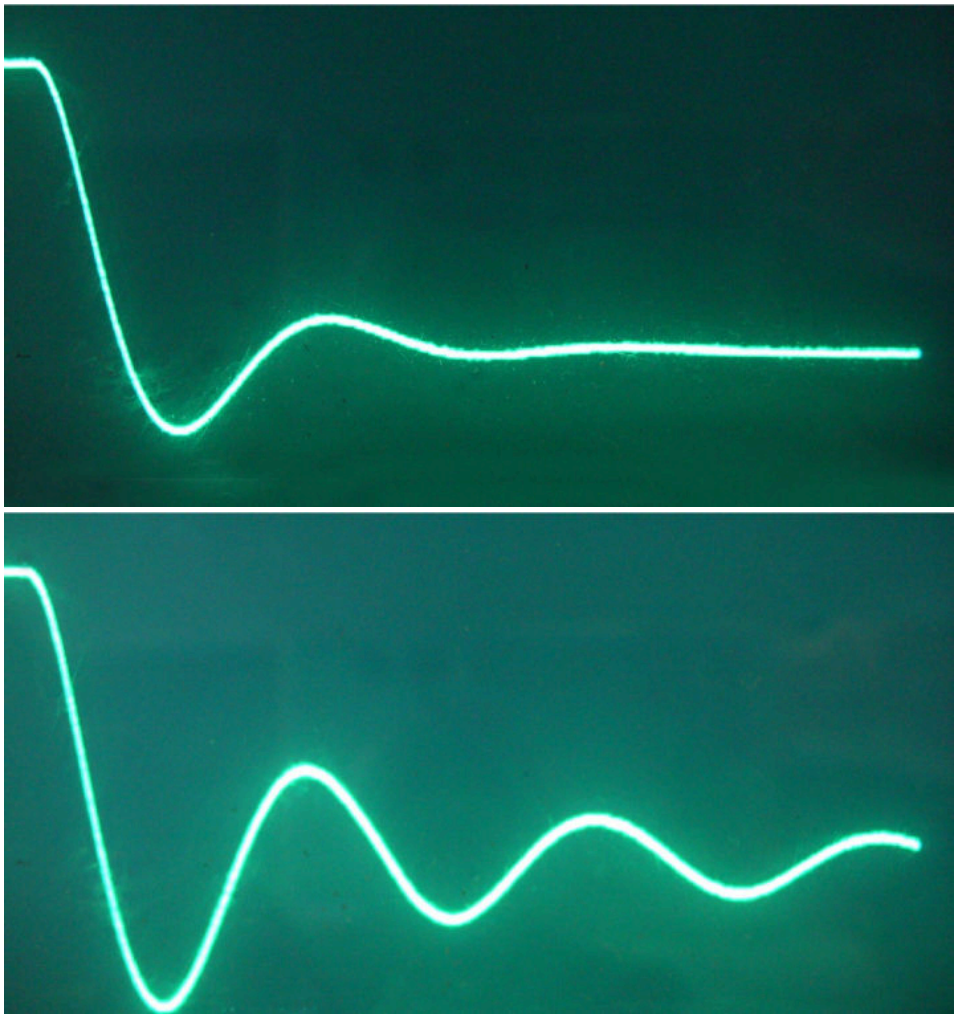
Fertiges Programm



Implementation



Typische Ergebnisse mit unterschiedlichen Dämpfungskoeffizienten:



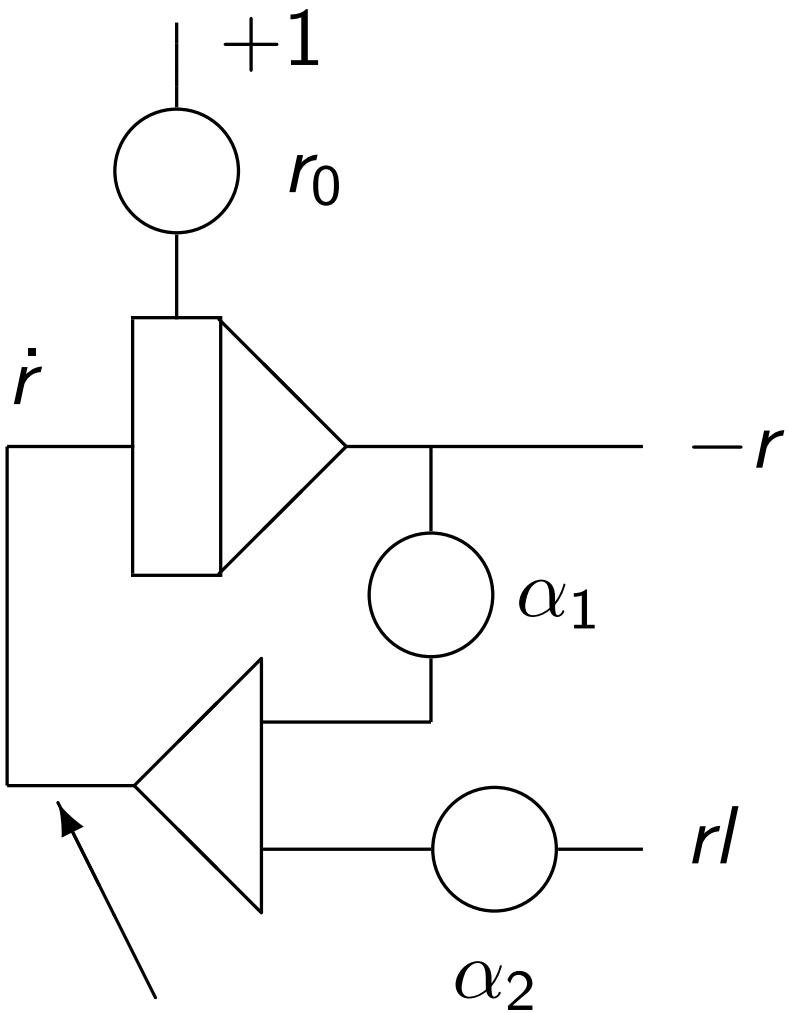
Räuber-Beute-Modell

Etwas komplexer ist ein Räuber-Beute-Modell (Lotka-Volterra-Gleichungen), das beispielsweise aus einem (halbwegs abgeschlossenen) Ökosystem mit Hasen und Luchsen besteht:

$$\begin{aligned} \dot{r} &= \alpha_1 r - \alpha_2 rl \\ \dot{l} &= -\beta_1 l + \beta_2 rl \end{aligned}$$

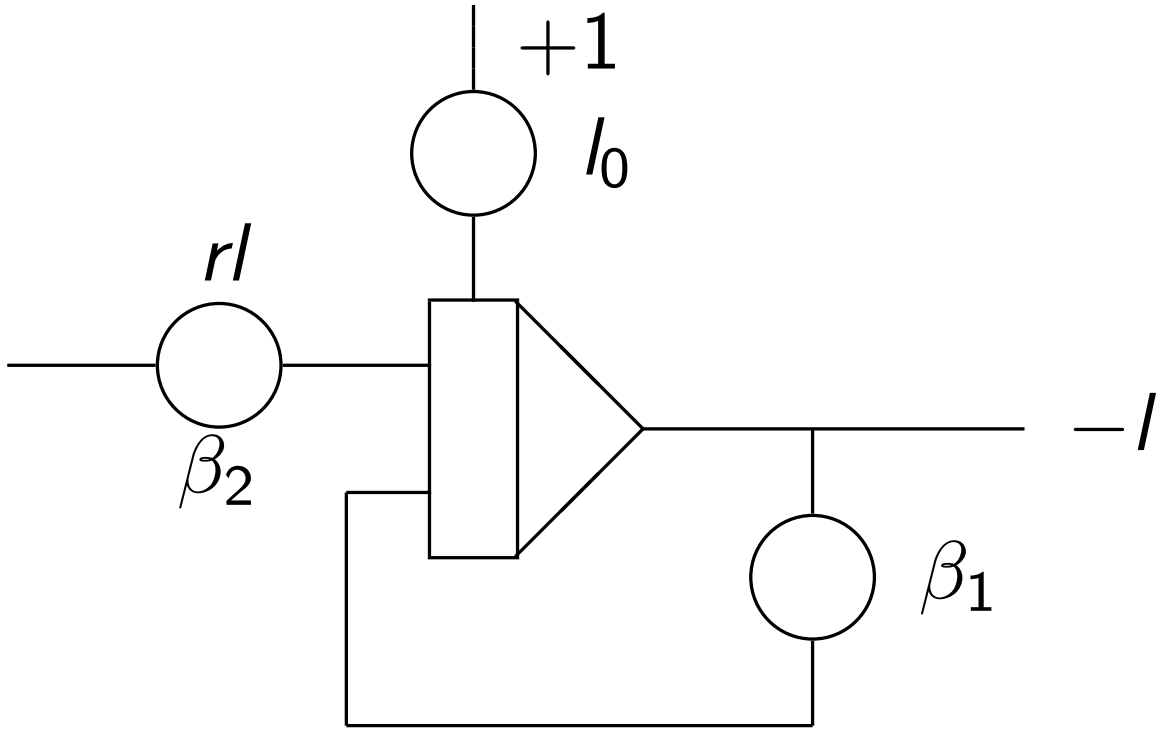
Hier ist α_1 die Geburtenrate der Hasen, β_1 ist die natürliche Sterberate der Luchse, α_2 ist die Rate der Hasen, die durch Luchse getötet werden und β_2 beschreibt den Zuwachs der Luchspopulation durch das Fressen von Hasen.

Erzeuge $-r$

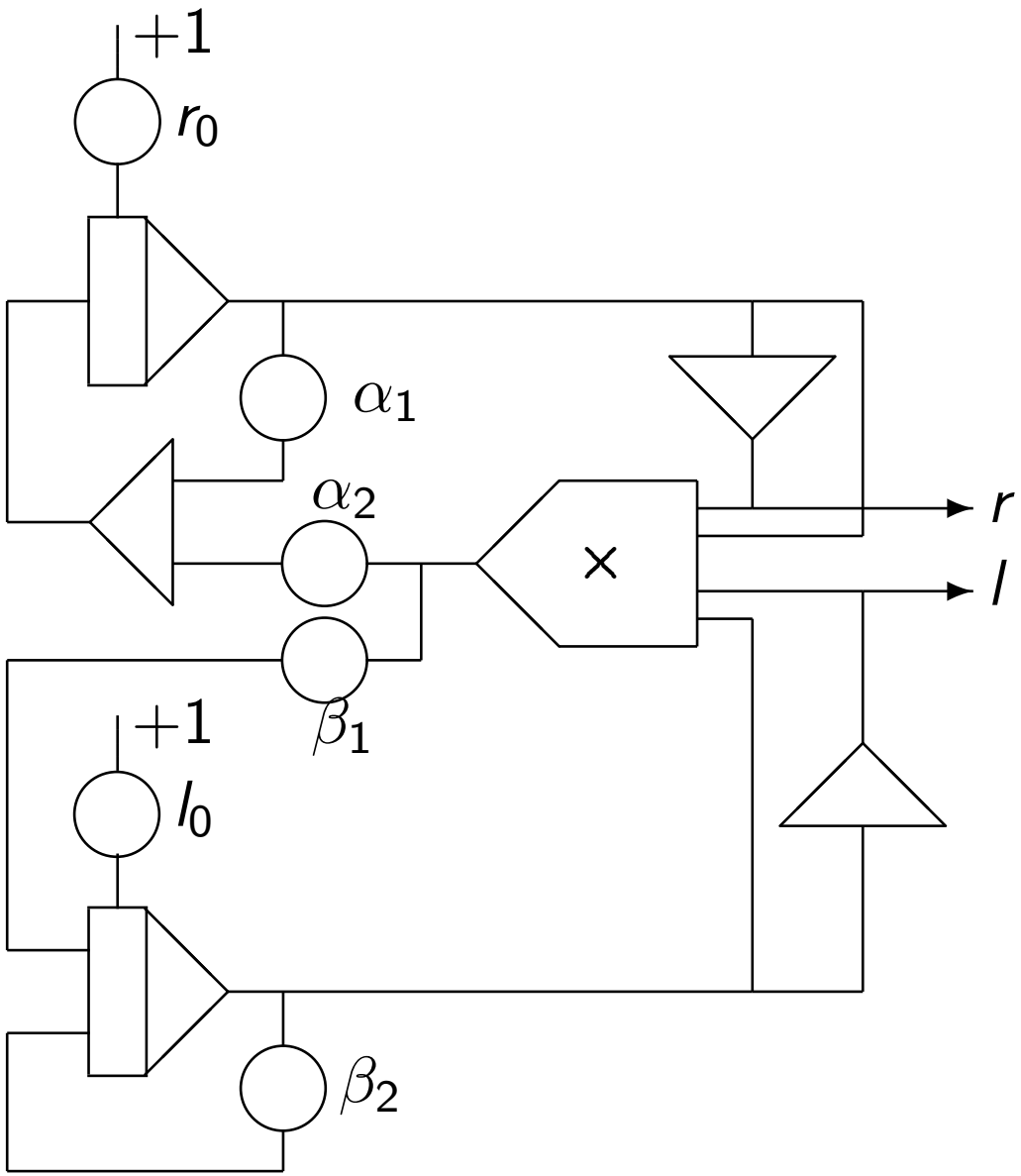


$$-\left(-\alpha_1 r + \alpha_2 rl\right) = \alpha_1 r - \alpha_2 rl$$

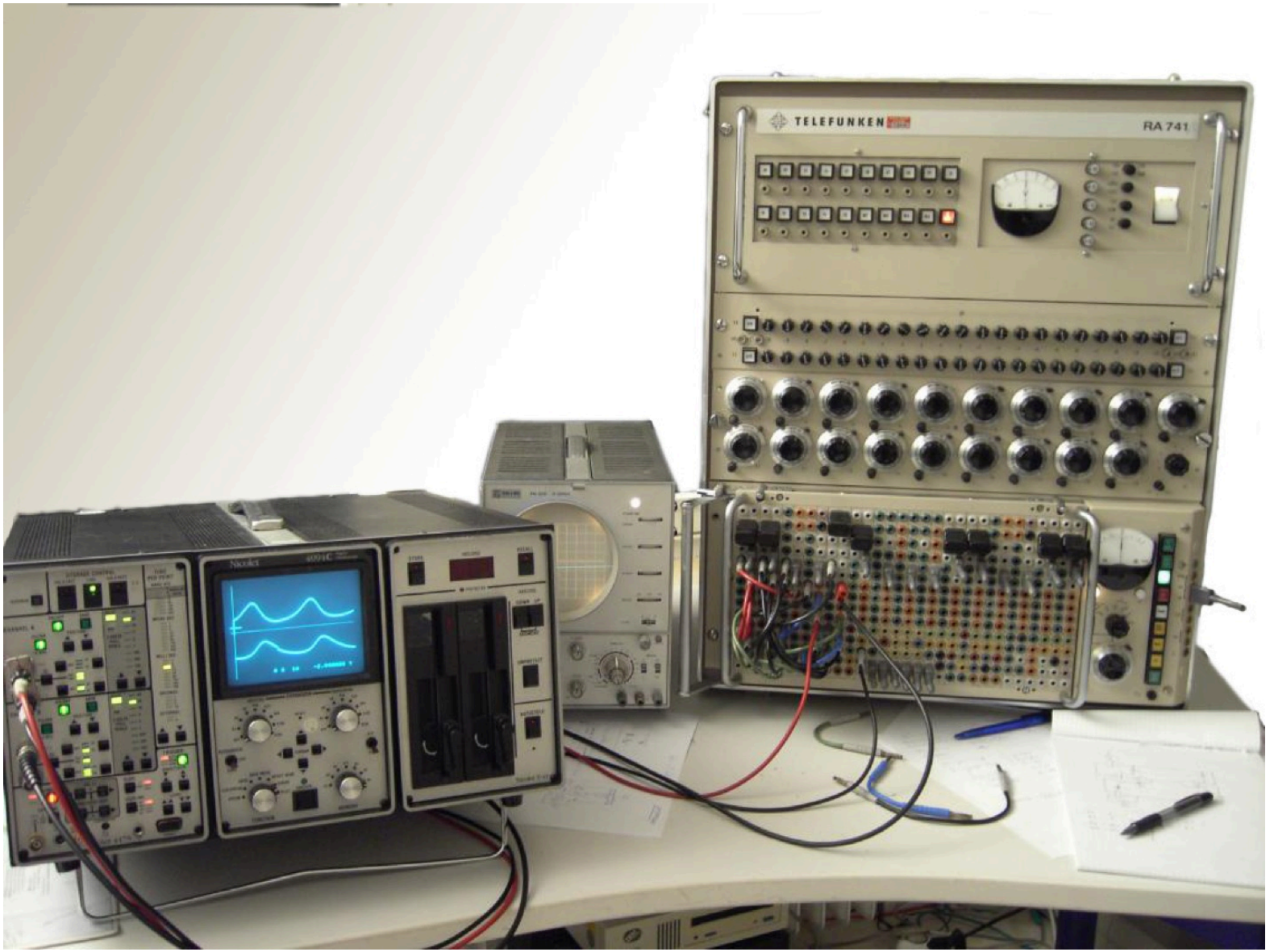
Erzeuge $-l$



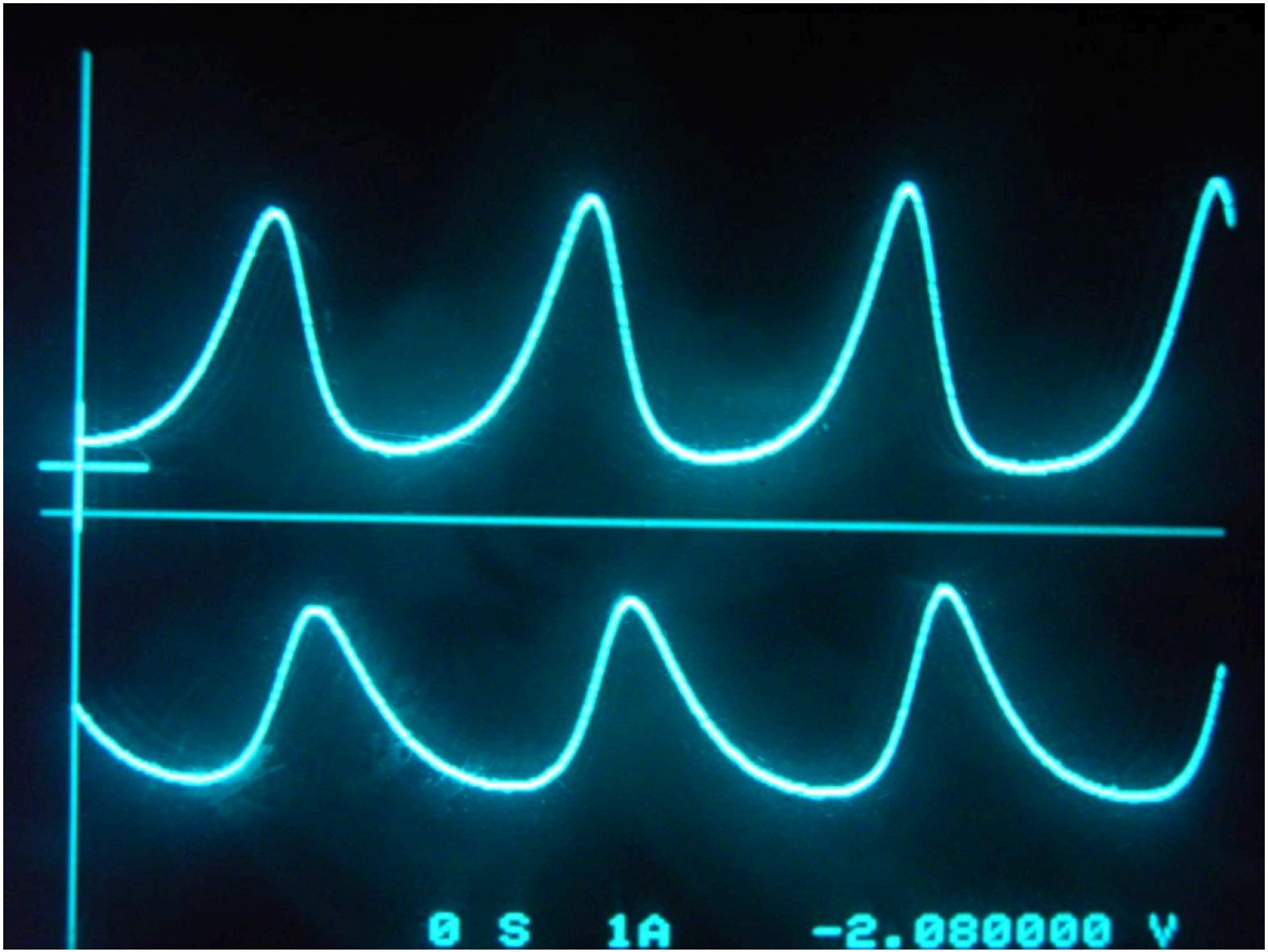
Programm



Implementation



Ergebnisse



van der Pol-Oszillator

Die van der Pol-Gleichung

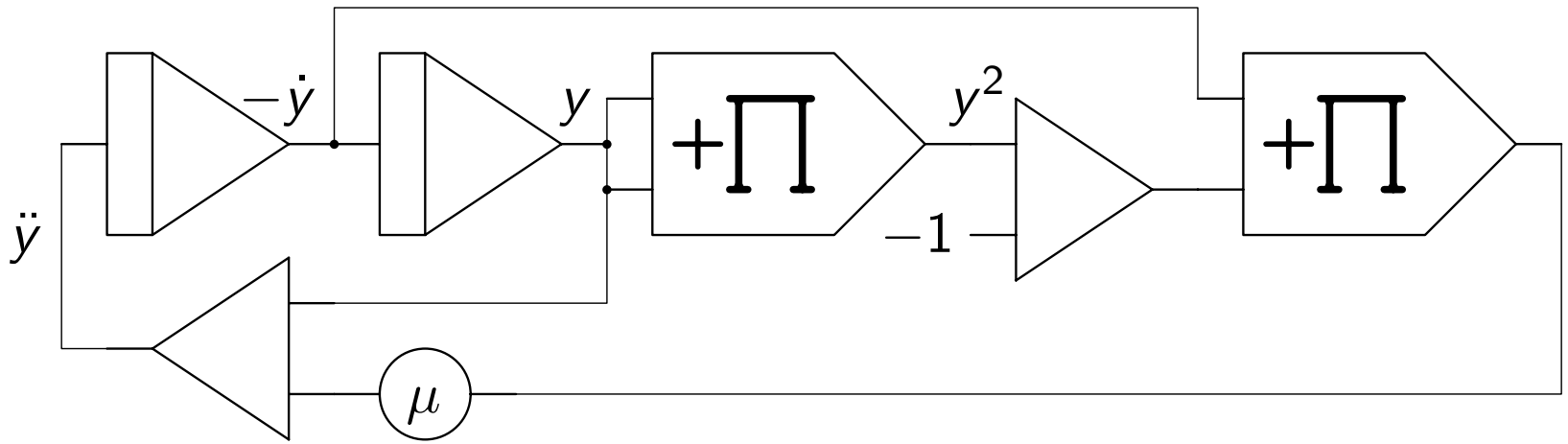
Noch etwas komplizierter ist die VAN DER POL-Gleichung:

$$\ddot{y} + \mu \left(y^2 - 1\right) \dot{y} + y = 0,$$

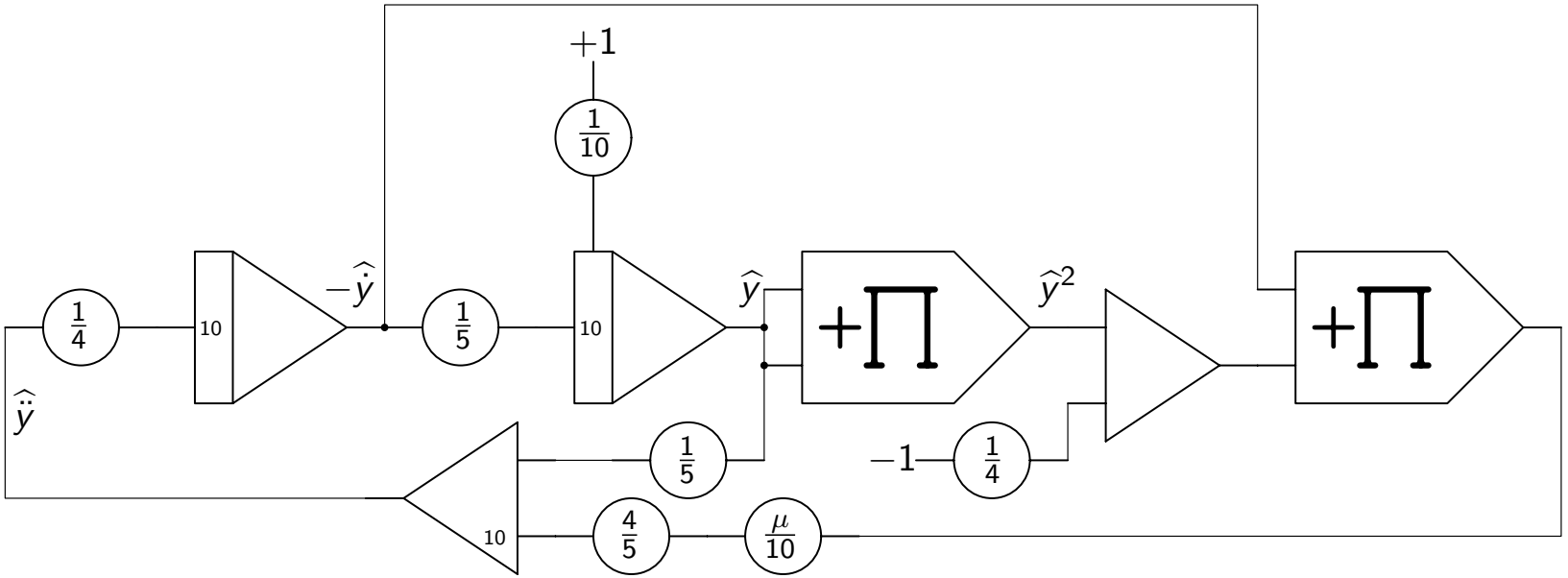
die nach $\ddot{y}$ aufgelöst werden kann:

$$\ddot{y} = -y - \mu \left(y^2 - 1\right) \dot{y}.$$

Hieraus resultiert das folgende, unskalierte Programm:

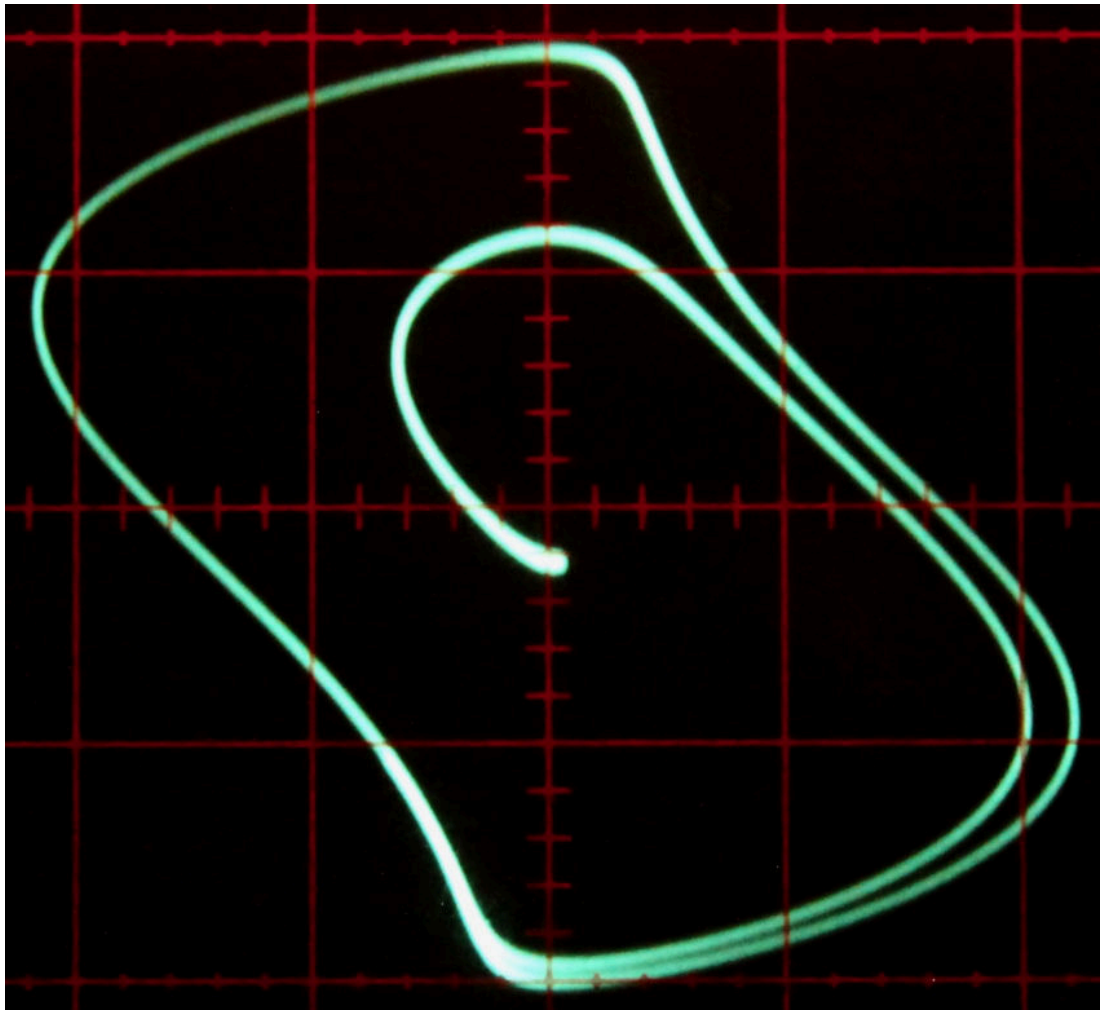


Die van der Pol-Gleichung



Ein typischer Phasenraumplot sieht wie folgt aus:

Die van der Pol-Gleichung



Partielle Differentialgleichungen

Partielle Differentialgleichungen sind „eklig“, weil Analogrechner in der Regel nur nach der Zeit integrieren können, man braucht also spezielle Ansätze, hier am Beispiel der Wärmeleitungsgleichung demonstriert:

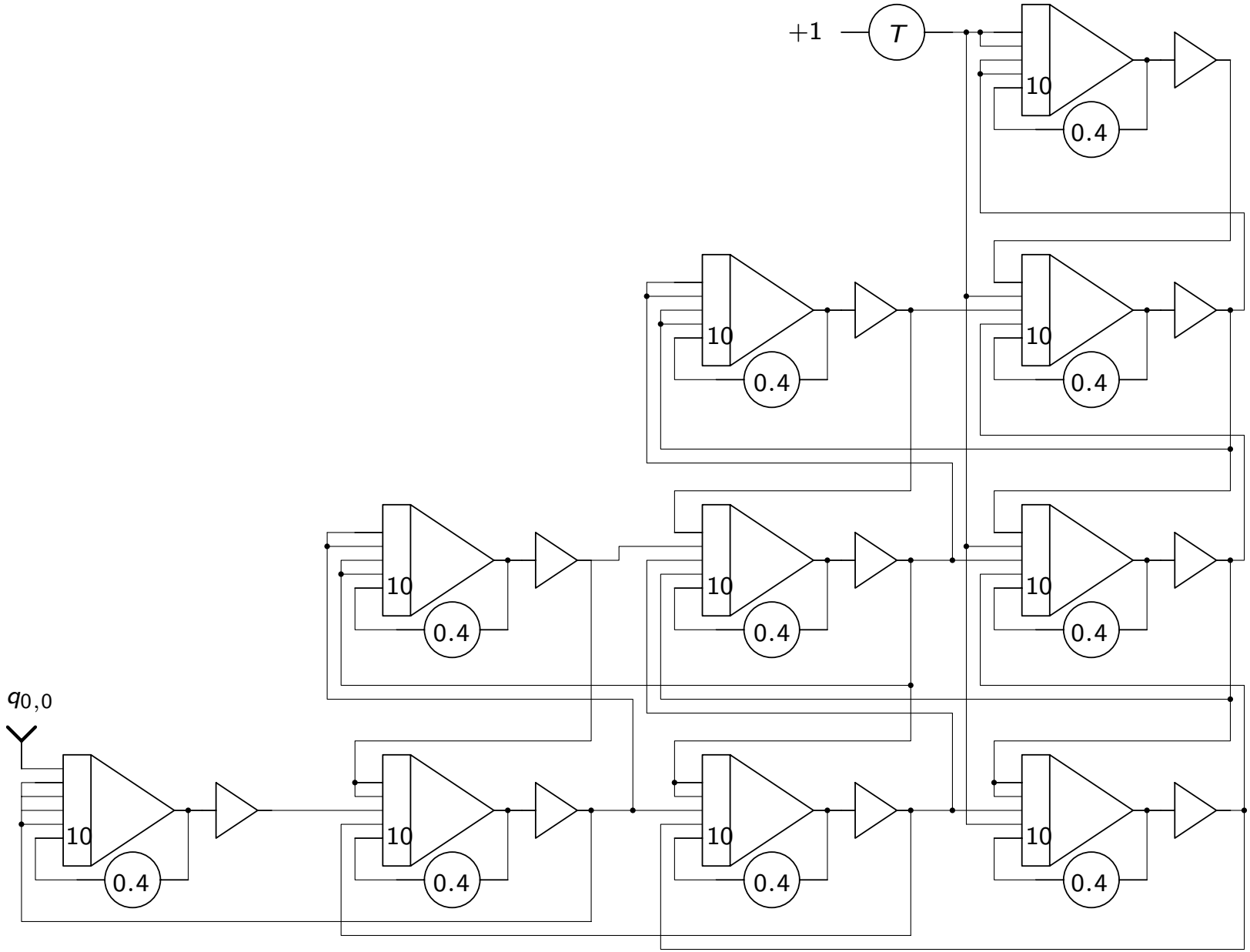
$$\dot{u} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right).$$

Ein Ansatz ist, die rechte Seite zu diskretisieren:

$$\dot{u}_{i,j} = \alpha \left(u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1} - 4u_{i,j} \right) + q_{i,j}.$$

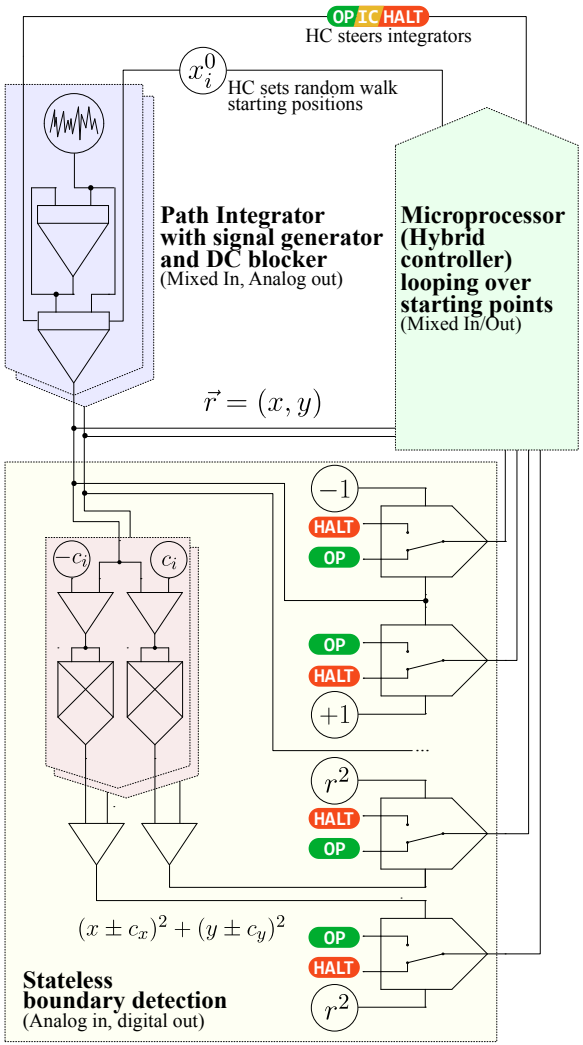
Das geht, ist aber aufwändig:

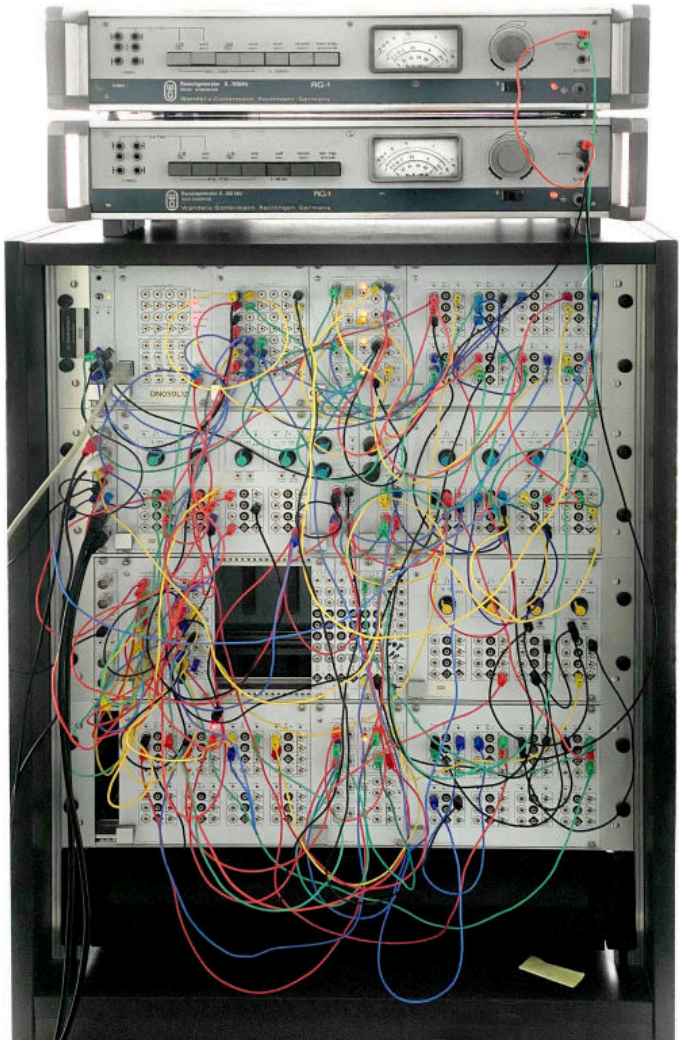
Die Wärmeleitungsgleichung



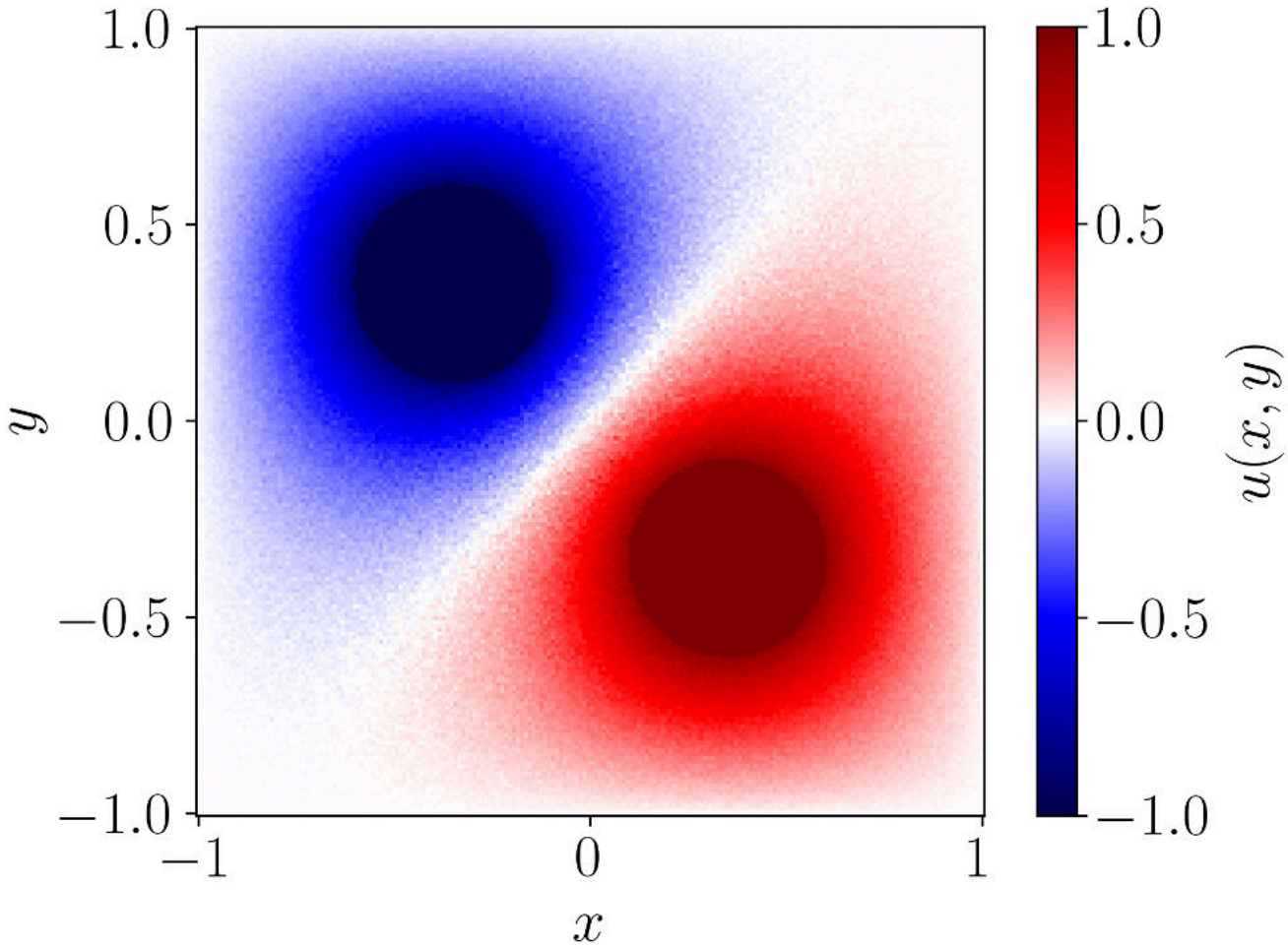
Feynman-Kac

Mitunter kann man hier z. B. Monte-Carlo-Ansätze verwenden (nach Feynman und Kac):



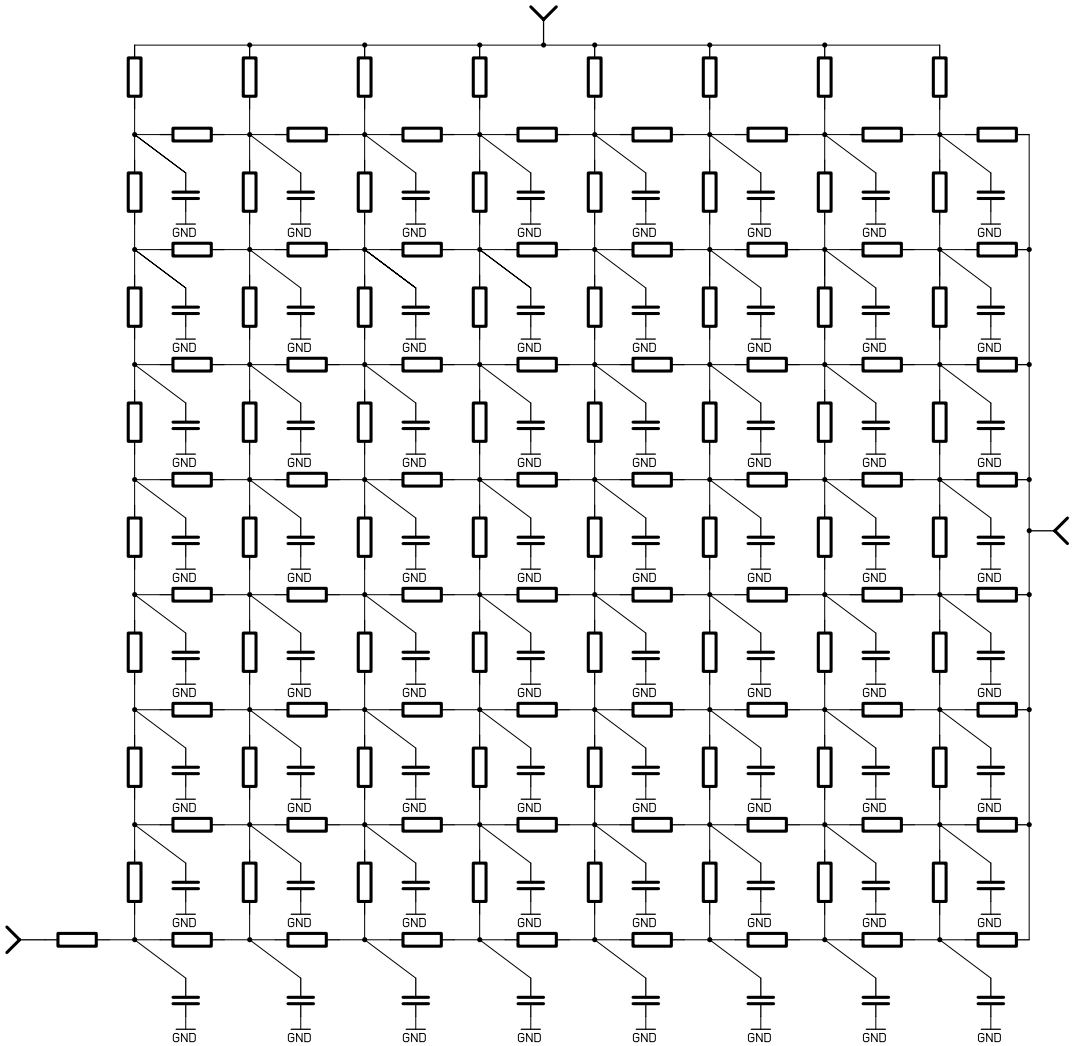


Das Nette hierbei ist, dass solche Ansätze ohne Gitter funktionieren und man plötzliche wieder Zeit quasi gegen Genauigkeit/Auflösung „tauschen“ kann:

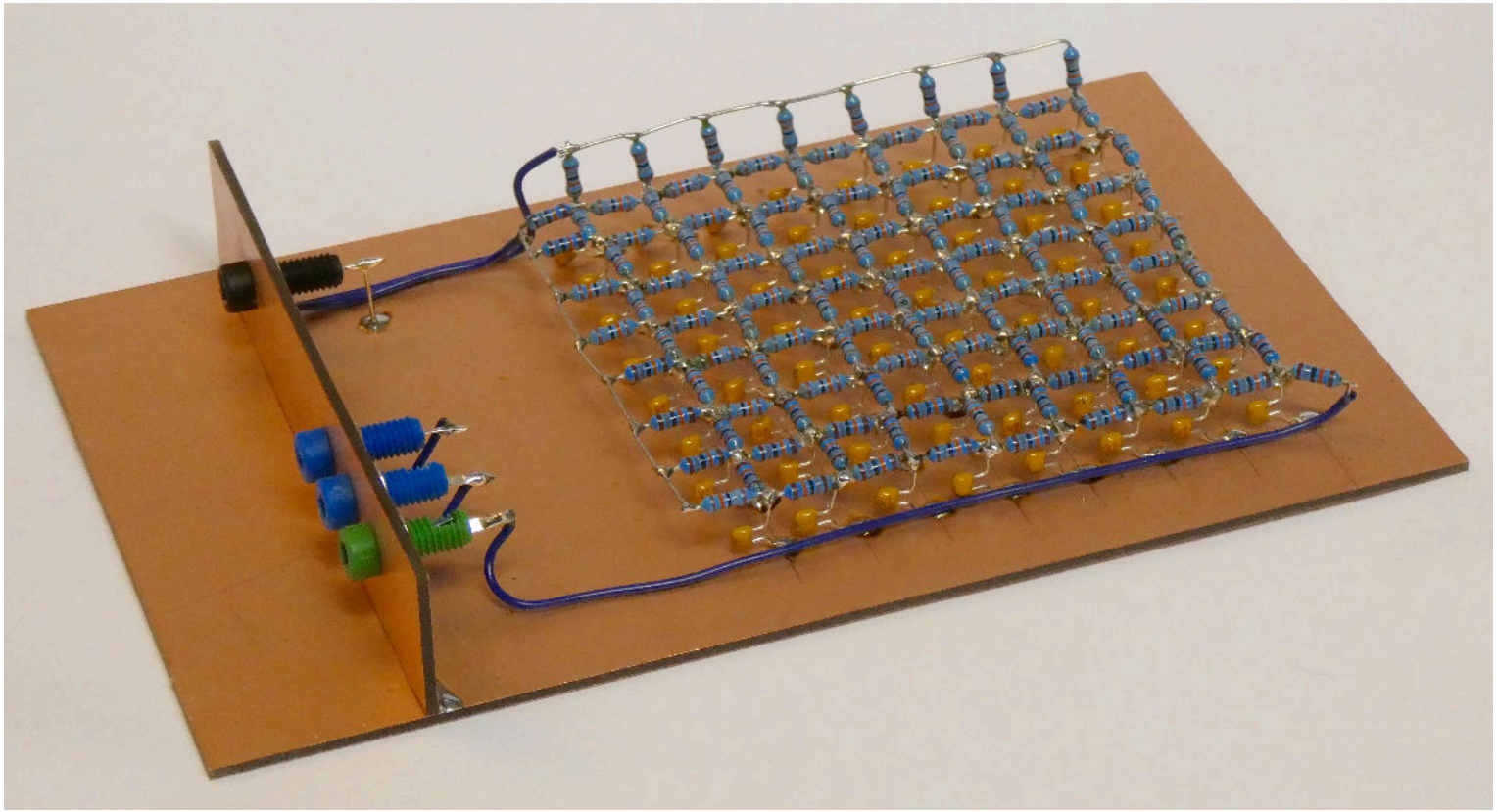


Wärmeleitungsgleichung

So etwas wie die Wärmeleitungsgleichung kann man sogar noch viel „direkter“ in Angriff nehmen:

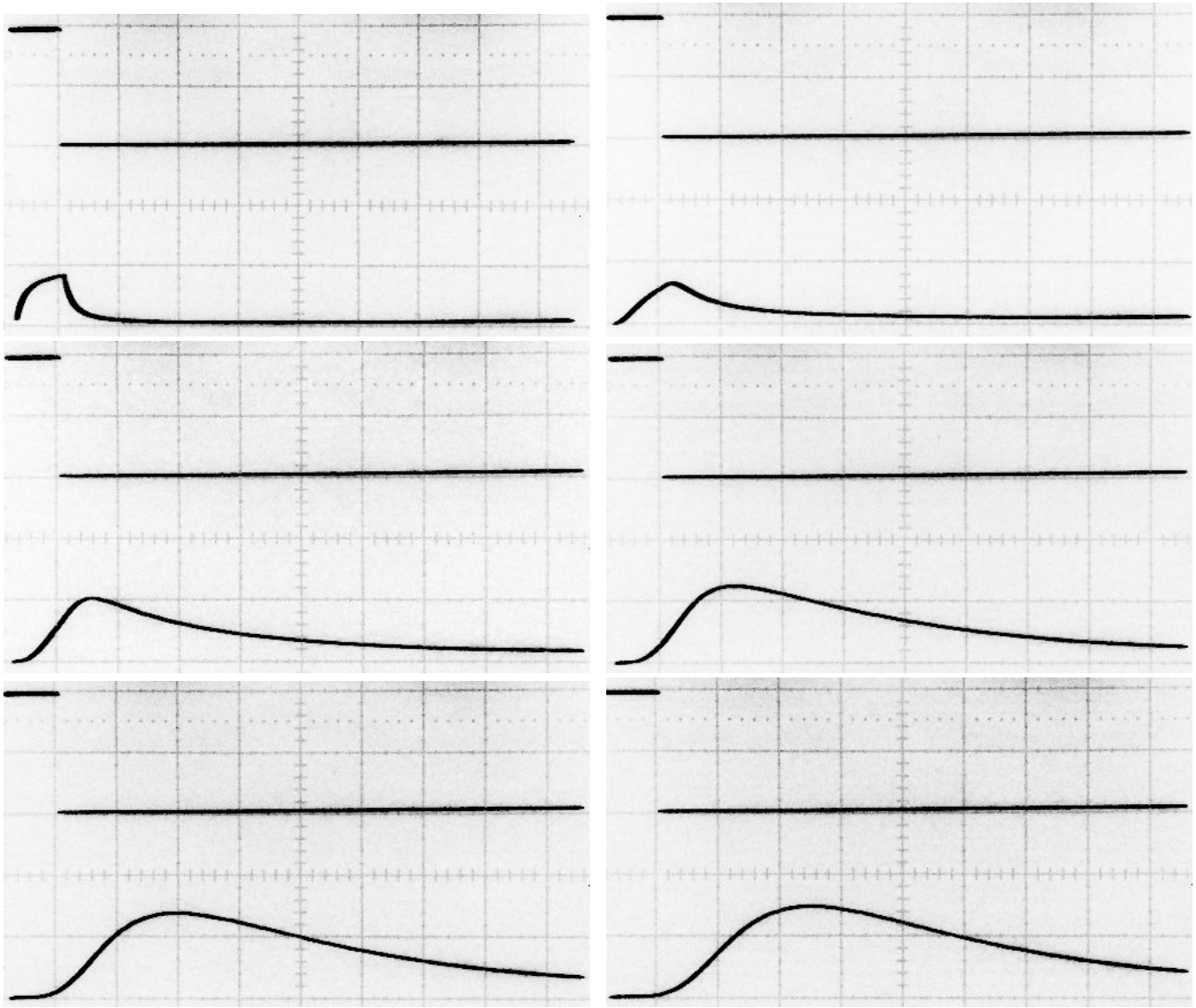


Ein passives Modell



Ein passives Modell

Ergebnis nach einem Eingangimpuls an der Stelle $u_{0,0}$:



Ein passives Modell

Stabile Lösung: