Beyond Zeros and Ones
Analog and Hybrid Computing for the 21st century

Prof. Dr. Bernd Ulmann, anabrid GmbH

21. September 2026





Introduction

After many incredibly successful decades of stored-program digital computing different computational paradigms are shifting into the limelight:

- Analog Computing and
- Quantum Computing (which we won't discuss in this talk).

What is an analog computer?

An analog computer solves problems by setting up a mathematically equivalent electronic model. It is a non-algorithmic machine!

But first: Why should we even care?

Problems of classic digital computers

Why analog/hybrid computing?

Although stored program digital computers are ubiquituous now, they have a number of drawbacks which are becoming more and more of a real problem for future applications. Some of these challenges are:

- High power consumption
- Clock frequencies are limited (energy consumption, pipeline length, etc.)
- On-chip structures can not be shrunk much further – atoms are quite big after all ...
- Parallelism is hard to exploit with digital computers (AMDAHL's law)
- Only a tiny fraction of the silicon used in a digital computer actually performs computations, the vast majority deals with data storage, data management, control, etc.

Size



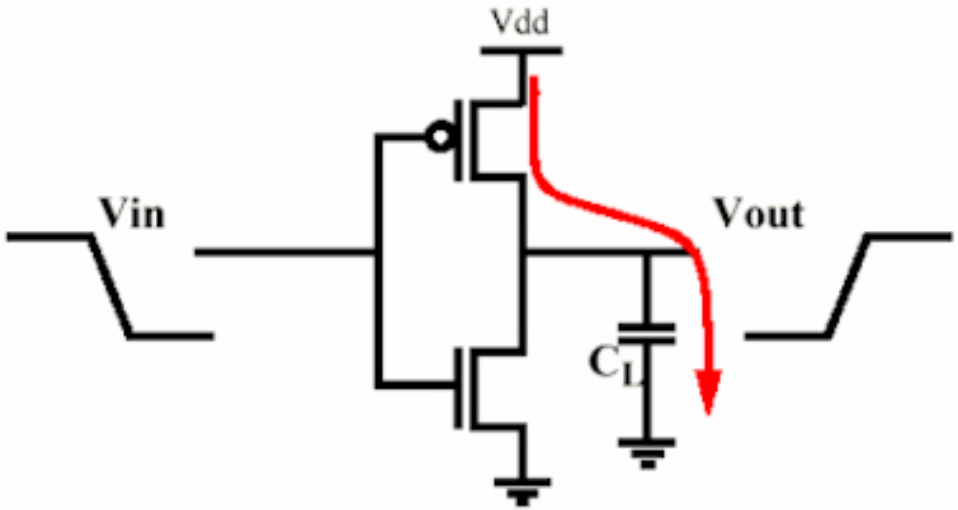
https://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Frontier_Supercomputer_%283%29.jpg/2560px-Frontier_Supercomputer_%283%29.jpg, retrieved 05.06.2024

Cooling



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

A typical output stage looks like this:



Problems:

- C_L gets charged/discharged at every change of the output.
- Both output transistors are switched on (very briefly) during a state transistion.
- The transistors are not perfect and exhibit leakage currents.
- $P_{\text{cpu}} = P_{\text{dyn}} + P_{\text{sc}} + P_{\text{leak}}$ is super-linear with respect to the clock frequency.

Power density

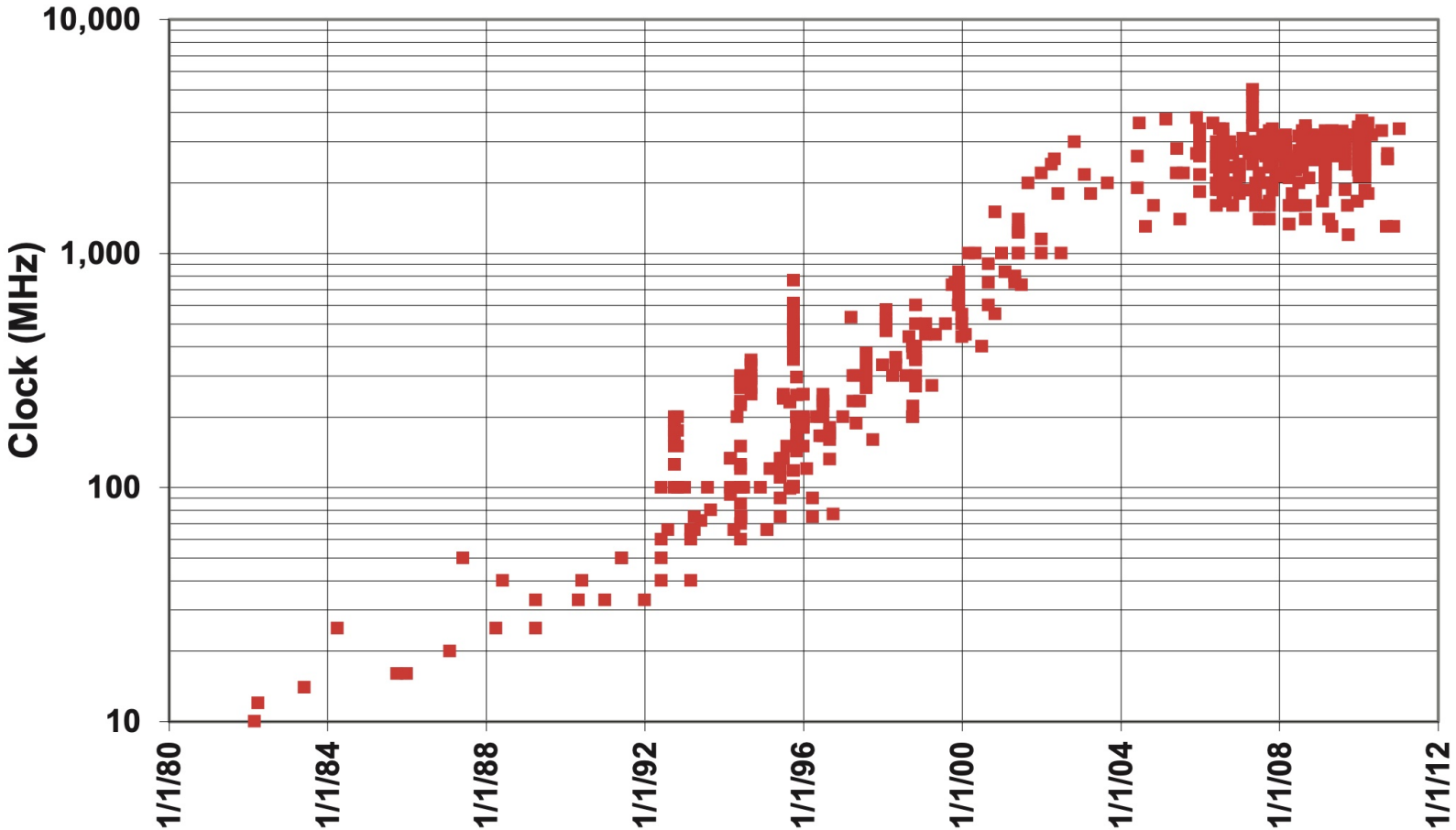


Power density





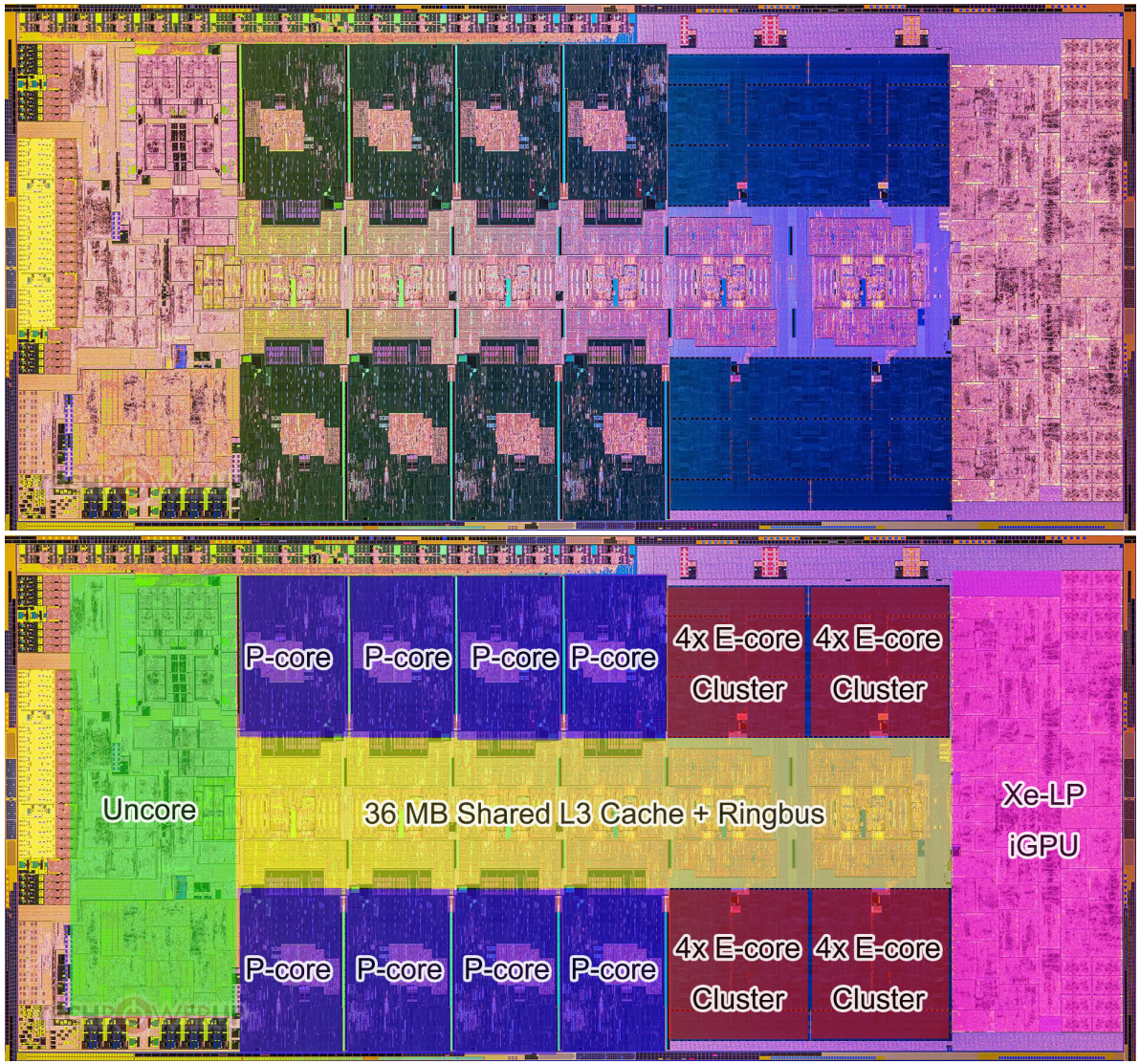
Clock frequencies



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

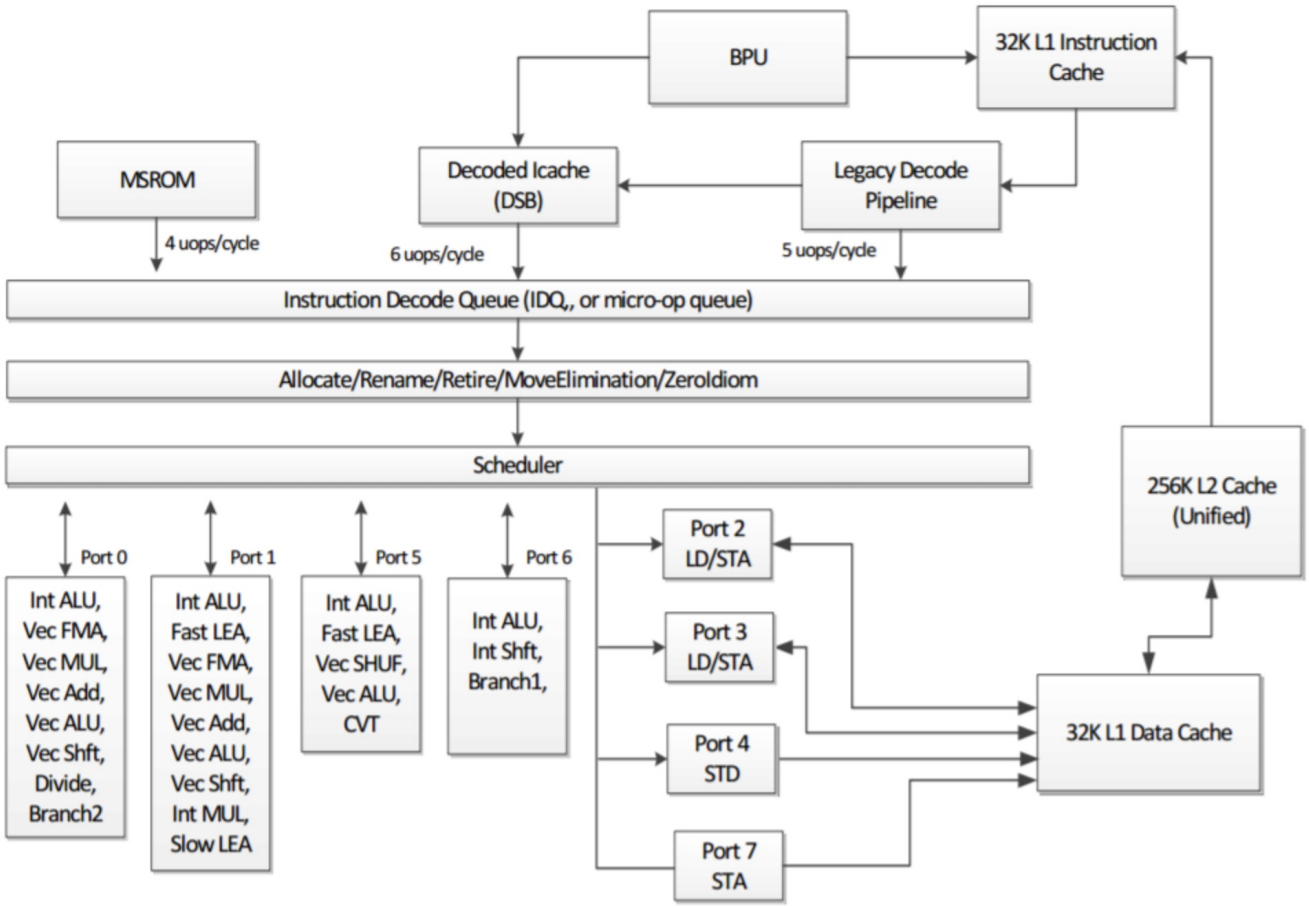
Typical chip structure

Intel Core i9-13900K:



https://www.techpowerup.com/review/intel-core-i9-13900k/2.html, retrieved: June 8th, 2023

Typical chip structure



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

the-intel-skylakex-review-core-i9-7900x-i7-7820x-and-i7-7800x-tested/3, retrieved: June 8th, 2023

A few numbers

Schnellster digitaler Supercomputer (`top500.org`, September 2026):

LineShine – Lingkun, Shenzen

- Rmax: 2198 PFLOPS
- 13 789 440 Kerne
- Ca. 42.22 MW
- About 52 GFLOPS/W

Human brain:

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

The “trick” of a biological brain is that is does not work algorithmically – it is an analog computer!

Problems of algorithms

- The main problem is that algorithms are basically sequences of instructions to be executed serially to solve a given problem.
- In contrast to this, reality is inherently parallel!
- A biological brain is an ideal example for this – it consists of an incredible number of basically rather simple cells, neurons, which work in full parallelism without any need for a central memory, central control circuitry etc.
- This is what distinguishes digital computers from analog and quantum computers:

Our current digital computers are *stored program* computers, while analog and quantum computers are not programmed in an algorithmic way but rather by setting up a *circuit* consisting of computing elements working in full parallelism to solve a problem.

Analog Computing

Analog Computers

- Analog computers are programmed by setting up a model, an *analogue*. There is no algorithm at all (this is a major advantage! :-))
- They are highly energy-efficient.
- Values are typically (but not necessarily so) represented as continuous voltages or currents.[1]
- They operate in an inherently parallel fashion and do not suffer from AMDAHL's law, etc.
- Analog computers adapt seamlessly to our analog world – no need for analog/digital conversions and vice versa.
- Analog computers excel at problems that can be described by (systems of) differential equations (most problems relevant in science and engineering fall into this category).
- Analog computers can even do some things typically attributed to quantum computers (oscillator based ISING machines).

[1] There are digital analog computers, *DDA*s, but we won't focus on these.

What is an analog computer?

An analog computer

- contains many computing elements such as integrators (sic!), summers, multipliers, etc. (Very small systems feature about a dozens computing elements, while large systems contain hundreds and thousands and potentially many more.)
- These computing elements are connected to each other in order to form an *analogue* for the problem to be solved (that's where the name stems from). So there is **no algorithm** and **no memory** at all. The program is a *directed graph* describing these connections. (In this respect, analog and quantum computers are quite similar.)
- All computing elements operate in a fully parallel fashion.

The notion

The notion of "analog computing" is tainted – it sounds old fashioned and only few people think of a model when hearing that term.

Better terms would be "natural computing" or "physical computing".

There is nothing old fashioned about analog computers!

	continuous values	discrete values
algorithmic	N. A.	classic digital computer
non-algorithmic	analog computer	digital differential analyzer

Digital computer

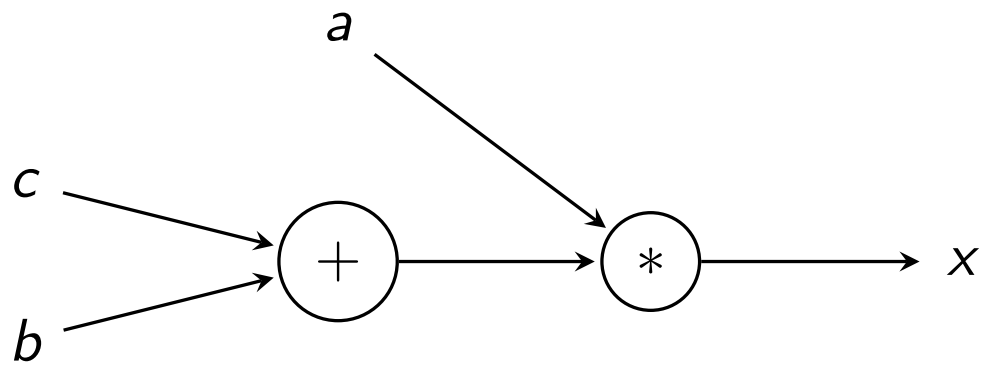
The following example shows the main differences between a digital computer and an analog computer:

Computing $a(b+c)$ on a stored program digital computer:

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

These are six instructions that have to be executed mainly sequentially.

Computing $a(b + c)$ on an analog computer:



Disadvantages

Of course, analog computers have some disadvantages, too (there is no such thing like a free lunch):

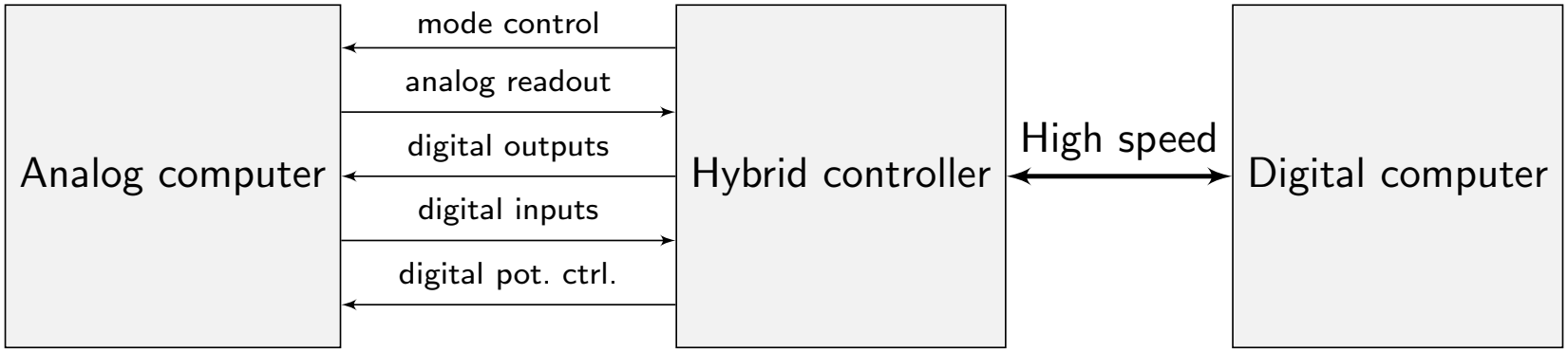
- Limited precision – typically about 3 to 4 decimal places (it turns out that this is often not a problem at all).
- Only time is directly available as free variable of integration, so solving partial differential equations requires some transformations/tricks.
- Generating arbitrary functions, especially those having more than one argument, requires special circuitry.
- Values are limited to the interval $[-1, 1]$, so problems must typically be *scaled* accordingly.

Most of these problems can be overcome by *hybrid computers*:

What is a hybrid computer?

- A hybrid computer is a combination of a traditional stored program digital computer and an analog computer.
- The analog computer offloads the digital computer from compute intensive tasks and thus acts as a co-processor.
- The digital computer controls the configuration and parametrization of the analog computer.
- The analog co-processor must be integrated seamlessly. This requires an extensive and complex software stack including an abstract programming language for the analog computer, associated compilers, libraries, etc.

What is a hybrid computer?



Areas of application

- High performance computing
- Artificial intelligence (implementation/training of ANNs, etc.) and machine learning
- Medical applications (cardiac pacemakers, brain pacemakers, insulin sensing/pump control) – the power consumption could be sufficiently low to power some implants with energy harvesting thus getting rid of clumsy batteries which limit life span and have to be recharged.
- Industrial control systems (basically stateless, not hackable in a traditional sense)
- Signal pre-/postprocessing for mobile devices and the like
- Trigger word detection for smart devices
- Monte-Carlo simulations, financial mathematics
- Optimization problems

A short history . . .

Gears from the Greeks

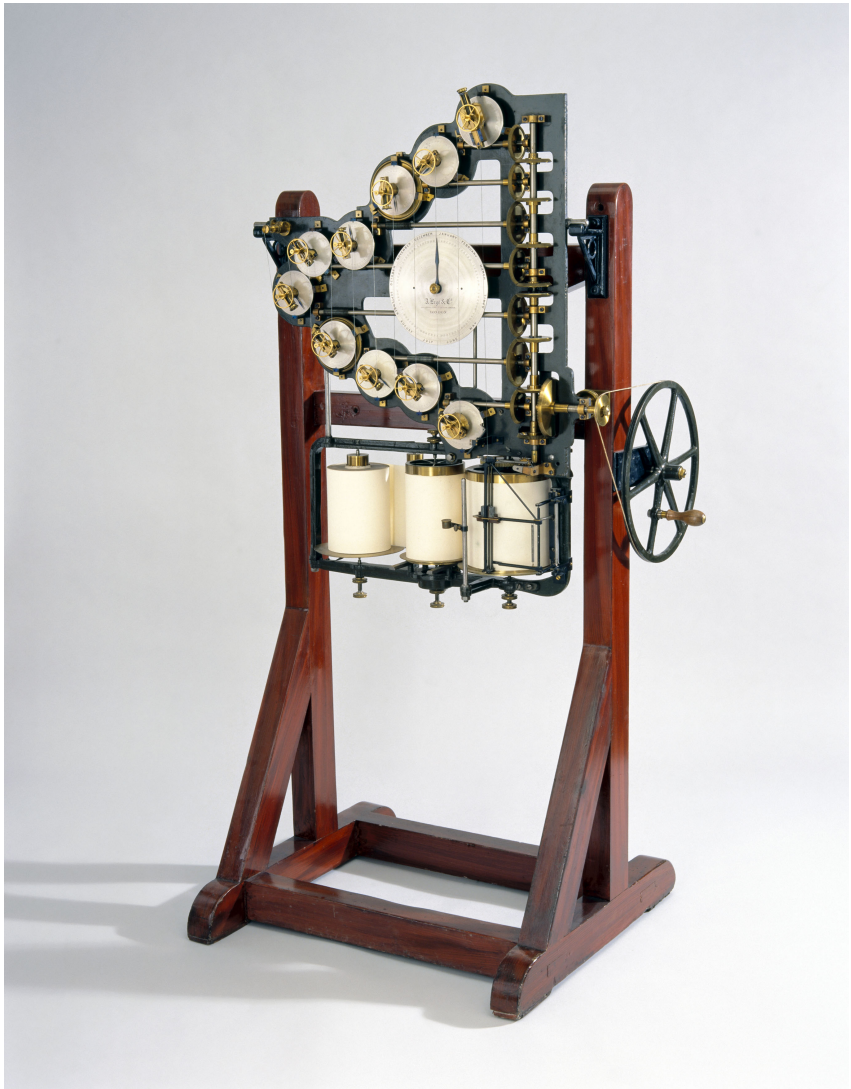
The *Antikythera* mechanism ($\approx$ 150 v. Chr.):



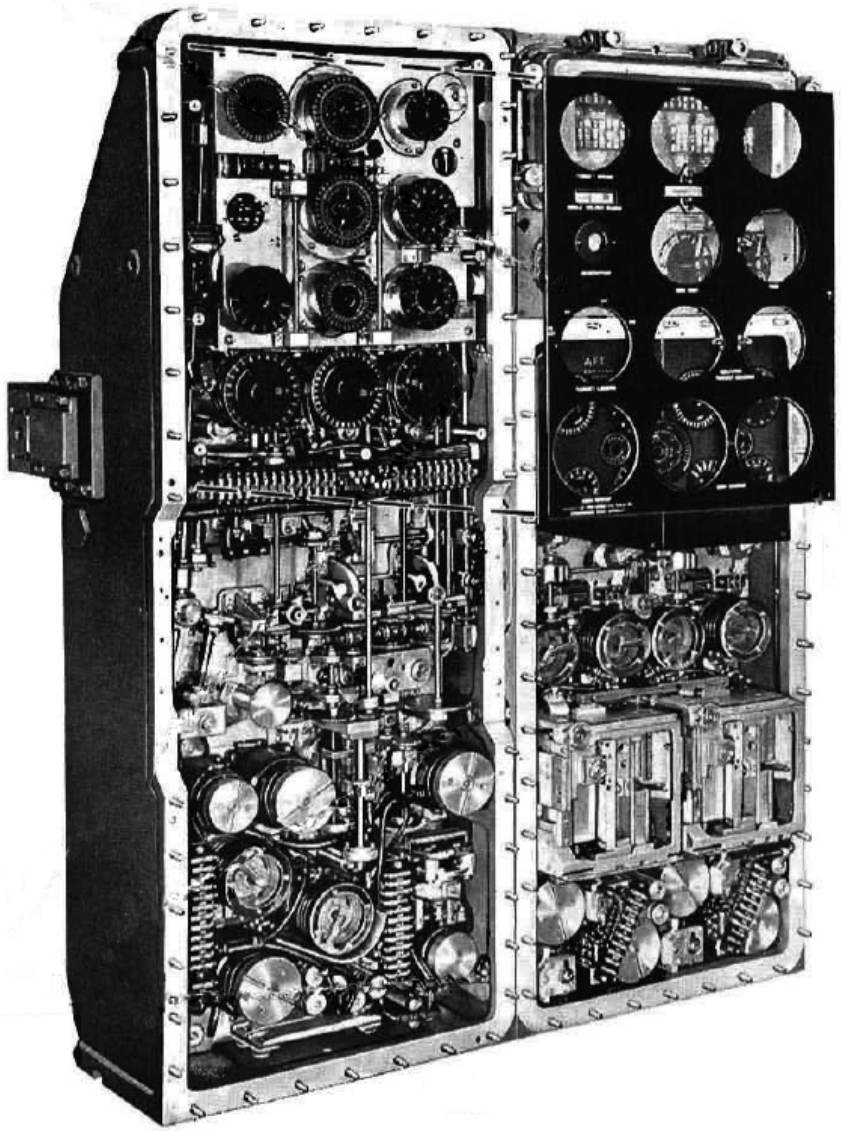
(http://upload.wikimedia.org/wikipedia/commons/6/66/NAMA_Machine_d%27Anticythre_1.jpg)

Kelvin's tide predictor

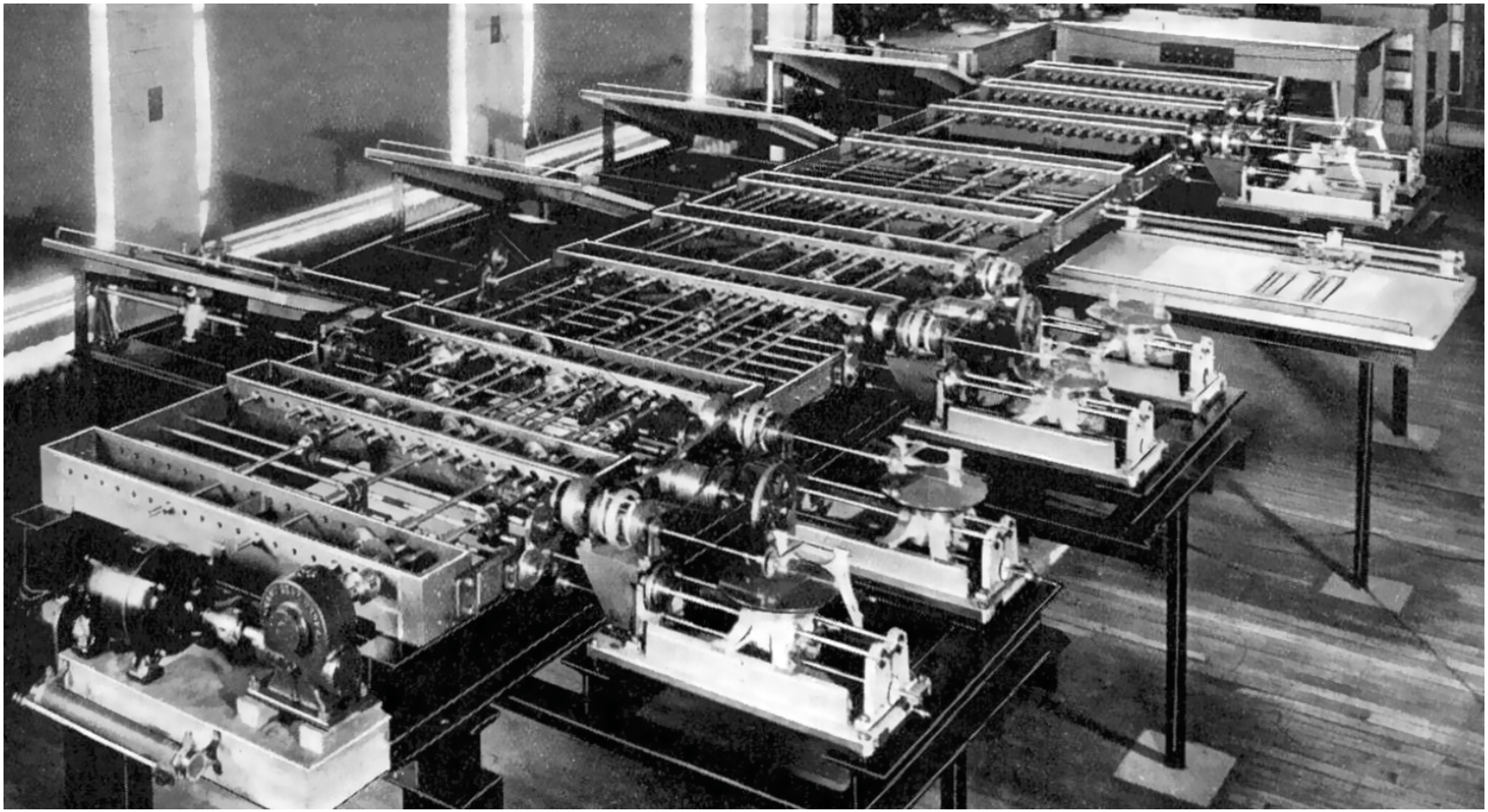
Harmonic synthesizer – 10 *partial tides*, 1876:



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

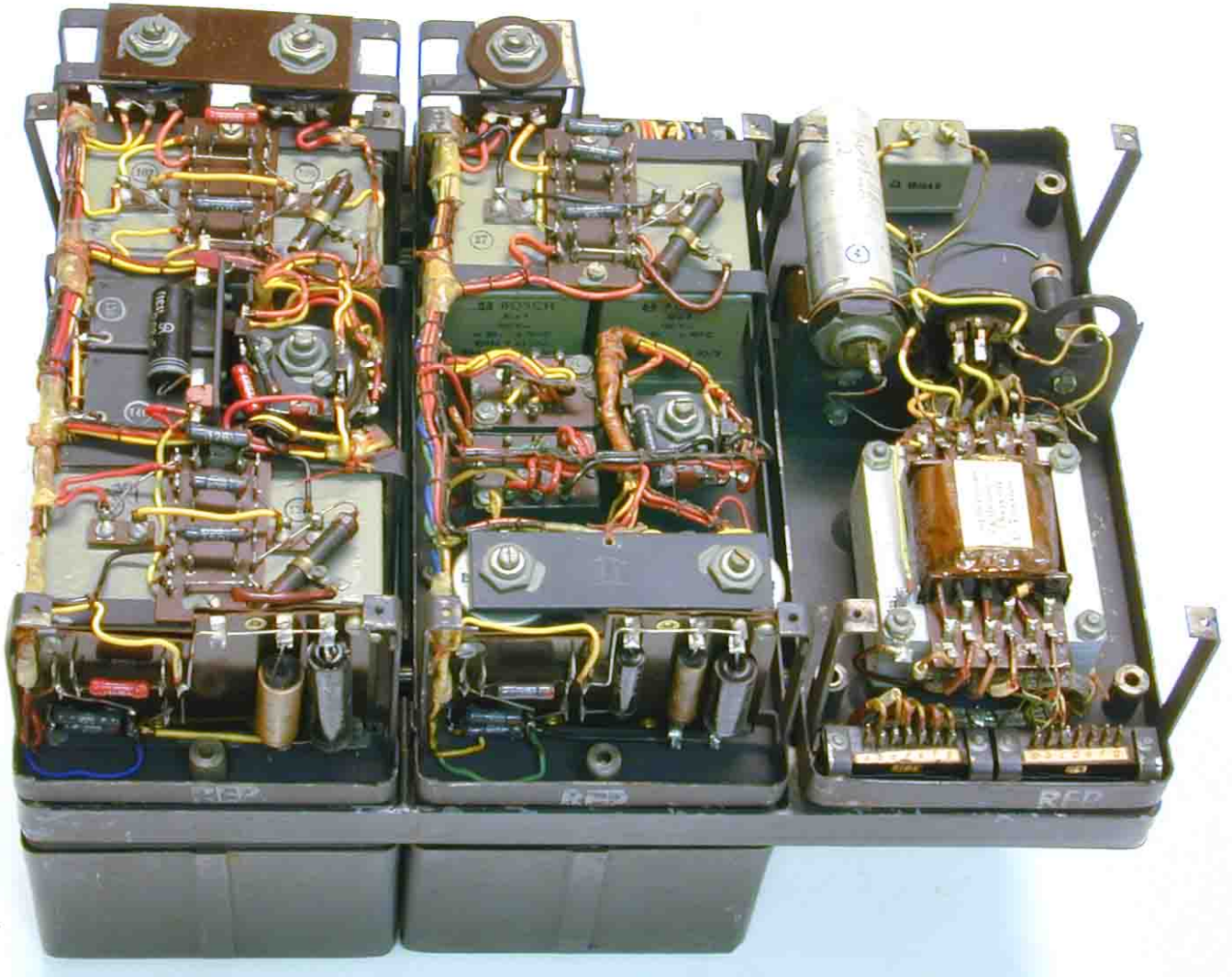


Vannevar Bush's DA, late 1920s:



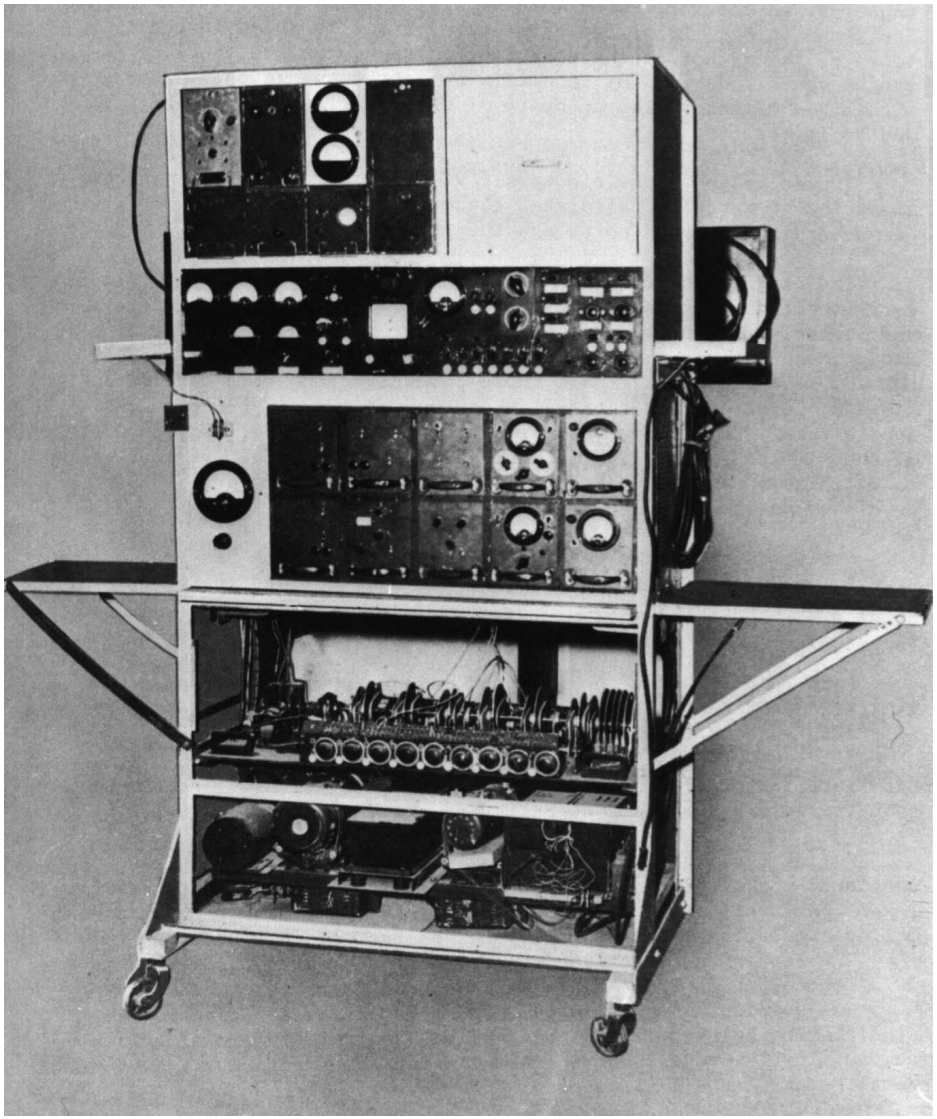
(See [Meccano 1934, p. 443].)

The *Mischgerät* (1940s):

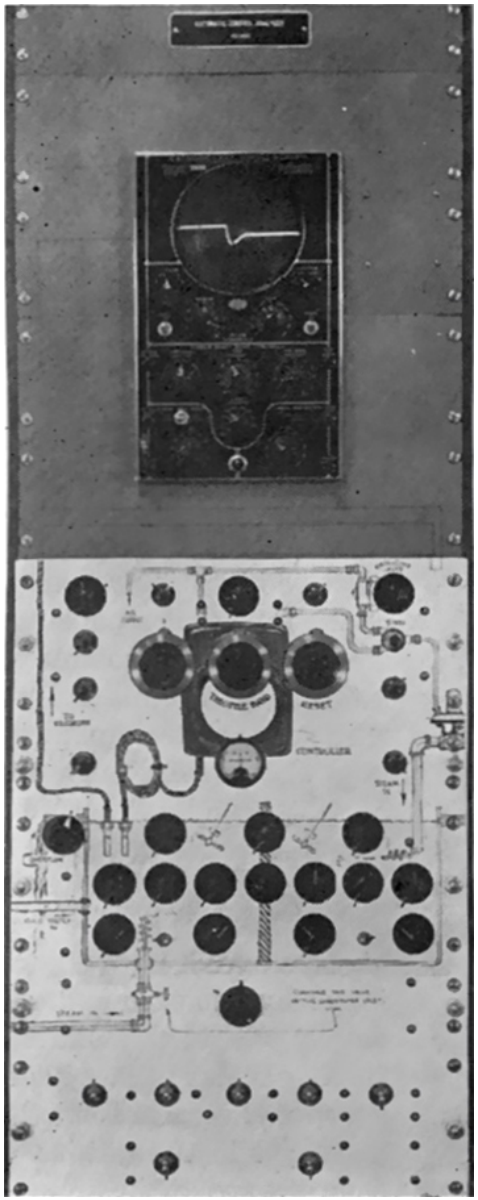


(Picture by Adri de Keijzer.)

HOELZER's general purpose analog computer:

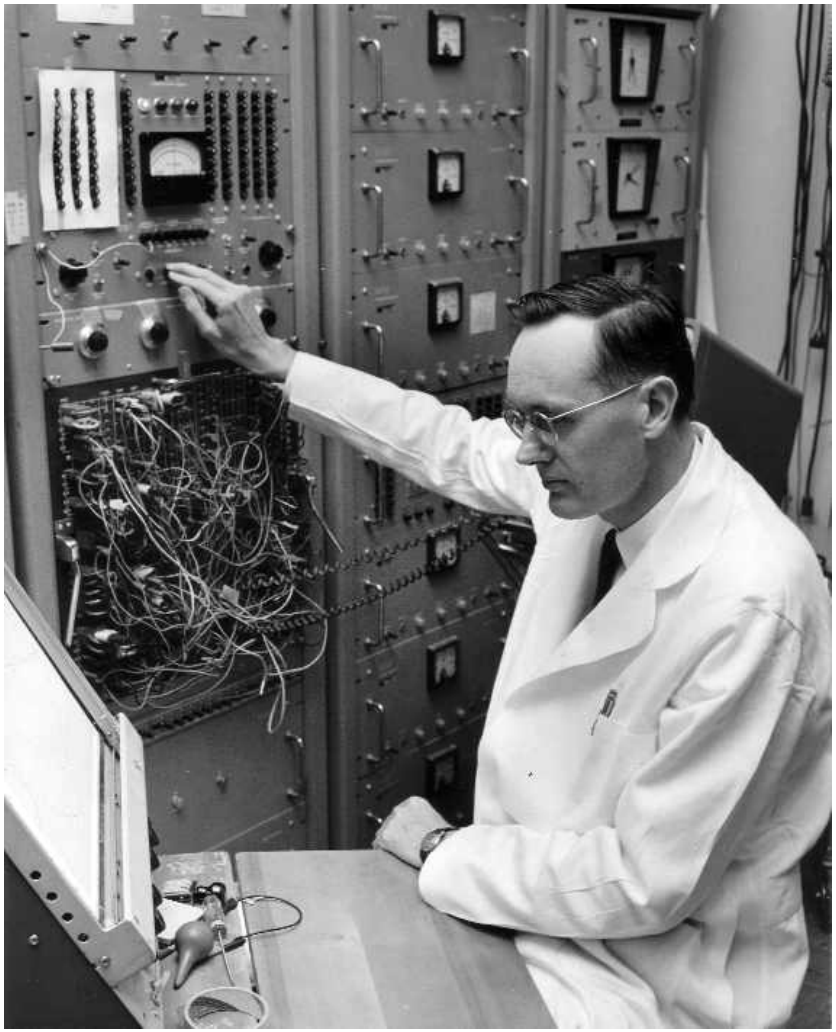


Polyphemus



George A. Philbrick developed *Polyphemus* in the late 1930s to simulate processed in the chemical industry (see [Philbrick 1948, p. 108]).

Beckman



Richard FitzHugh solves the Hodgkin-Huxley equations on a Beckman analog computer (Source: Izhikevich E. M., FitzHugh R.).

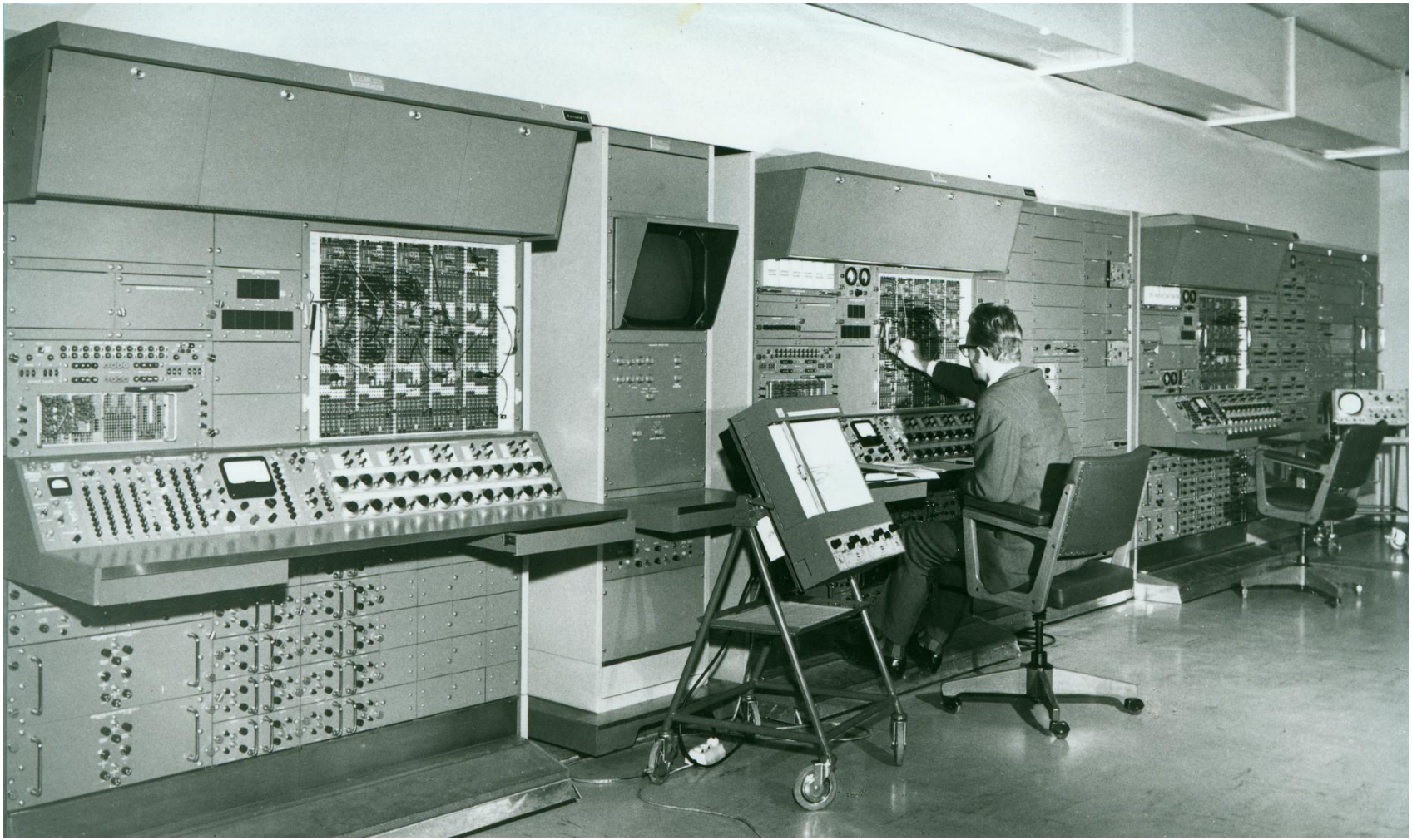
EAI 231R

EAI 231R installation at „Deutsches Elektronensynchrotron“, Hamburg, Picture: Inge Borchardt.



EAI im DLR

EAI installation at DLR (1970er Jahre), Picture Dr. Jessika Wichner, DLR, Signatur FF-557.

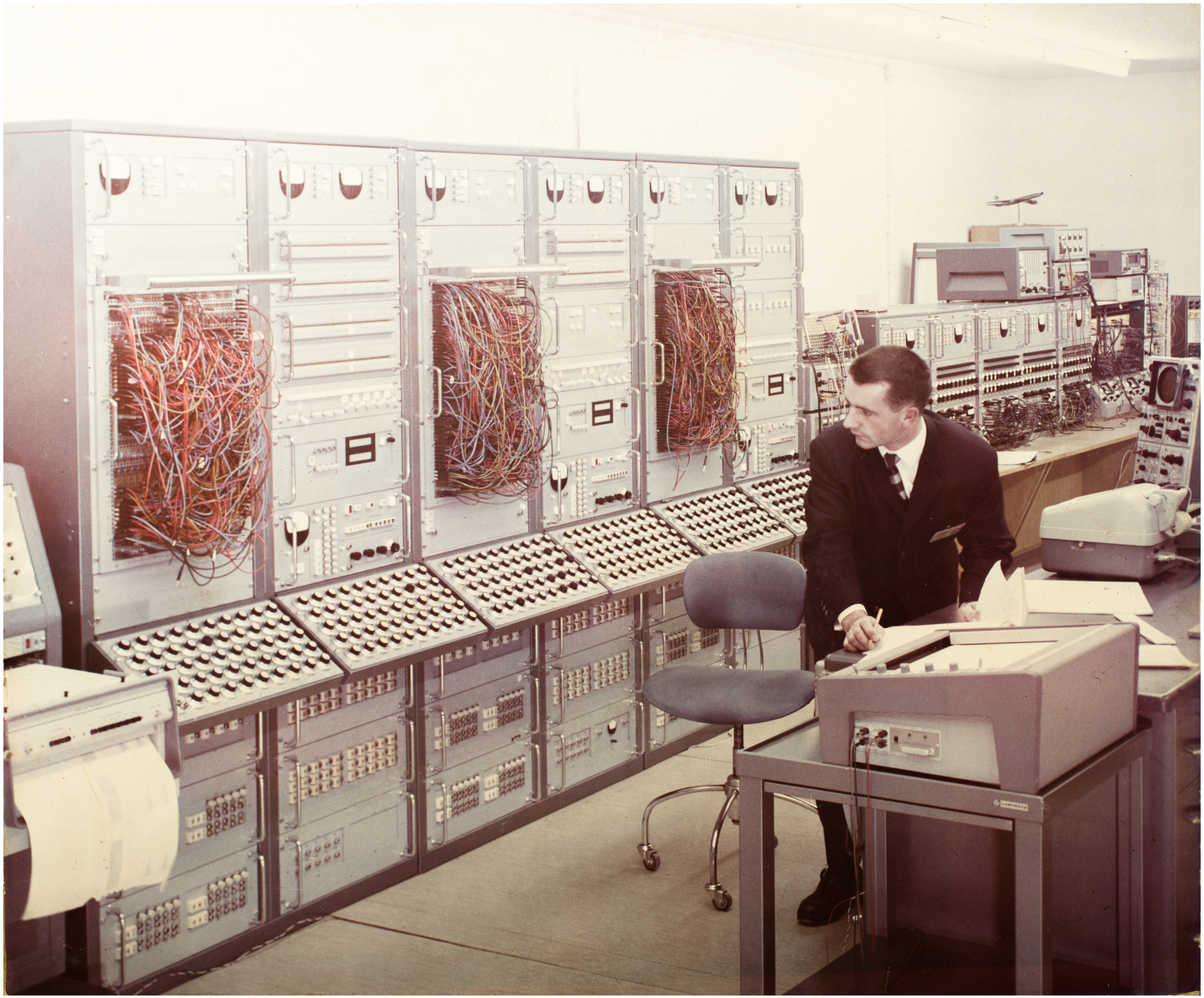


Five RAT 700 and an RA 800:



A flight simulator

Flight simulator at Boelkow, 1960 (VJ-101):



Remark

A few things must be noted:

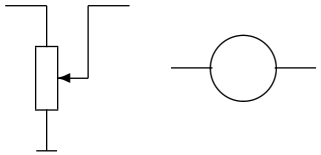
- I am **not** proposing building something like this in the future! Such systems belong into a museum.[2]
- **The patchpanel is a thing of the past!** The patching process could be quite cumbersome and time consuming.
- With a stored-program digital computer one can basically solve every problem given there are enough memory and time available. (This gives rise toe the *Big-O-Notation*.)
- In contrast to this and analog computer must have enough computing elements to implement a certain model. In contrast to a digital computer, this is very easy since there are no central memory, no complex control circuitry, etc.
- An analog computer basically has a constant time-to-solution, regardless of the problem size!

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

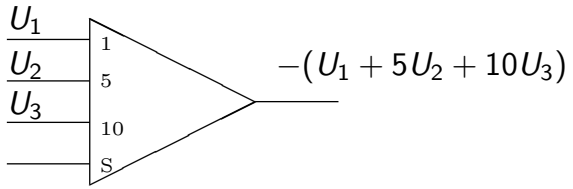
Basic programming – ODEs

Typical computing elements

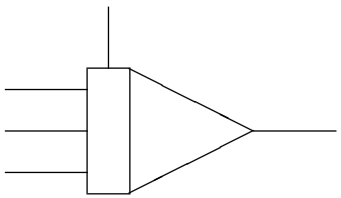
- Coefficient units (e. g. a voltage divider/multiplying DAC etc.)



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



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

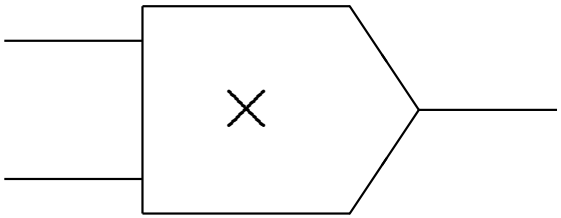


An integrator has three modes of operation: *initial condition*, *operate*, *halt*.

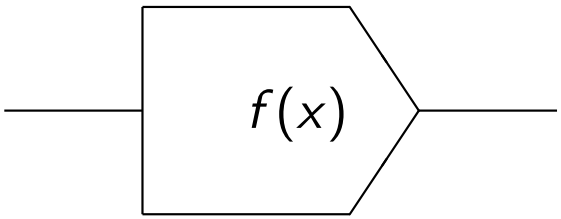
[3] The "magic component" – (near) perfect integration with a resistor, a capacitor and an operational amplifier.

Typical computing elements

- Multipliers



- *Open amplifiers* which can be used to implement inverse functions such as square root, division as well as special functions like limiters etc.
- Comparators with electronic switches
- Function generators (traditionally using polygonal interpolation, nowadays table-lookups using ADCs and DACs)



Programming

Programming an analog computer is much simpler than programming a stored program digital computer as there is nothing like an intricate algorithm controlling the system.

An analog computer programm, i. e. the interconnection scheme for its computing elements, is derived from the problem equations.

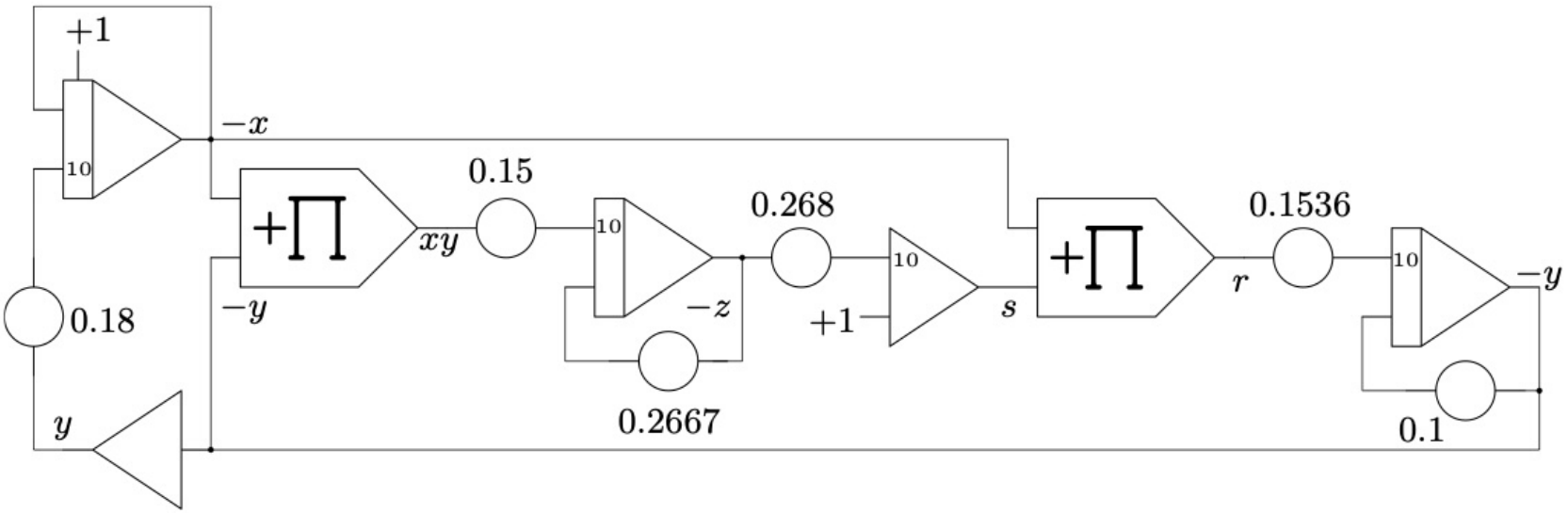
The classic programming technique is due to Lord Kelvin:

1. Solve the equation for the highest derivative.
2. Generate all lower derivatives by repeated integrations.
3. Derive all terms on the right hand side of the equation based on these lower derivatives.
4. Tie all these terms together. According to the equation this process started with, this must be equal to the highest derivative, so this value can be fed back into the circuit as this highest derivative.

The slides below show two examples:

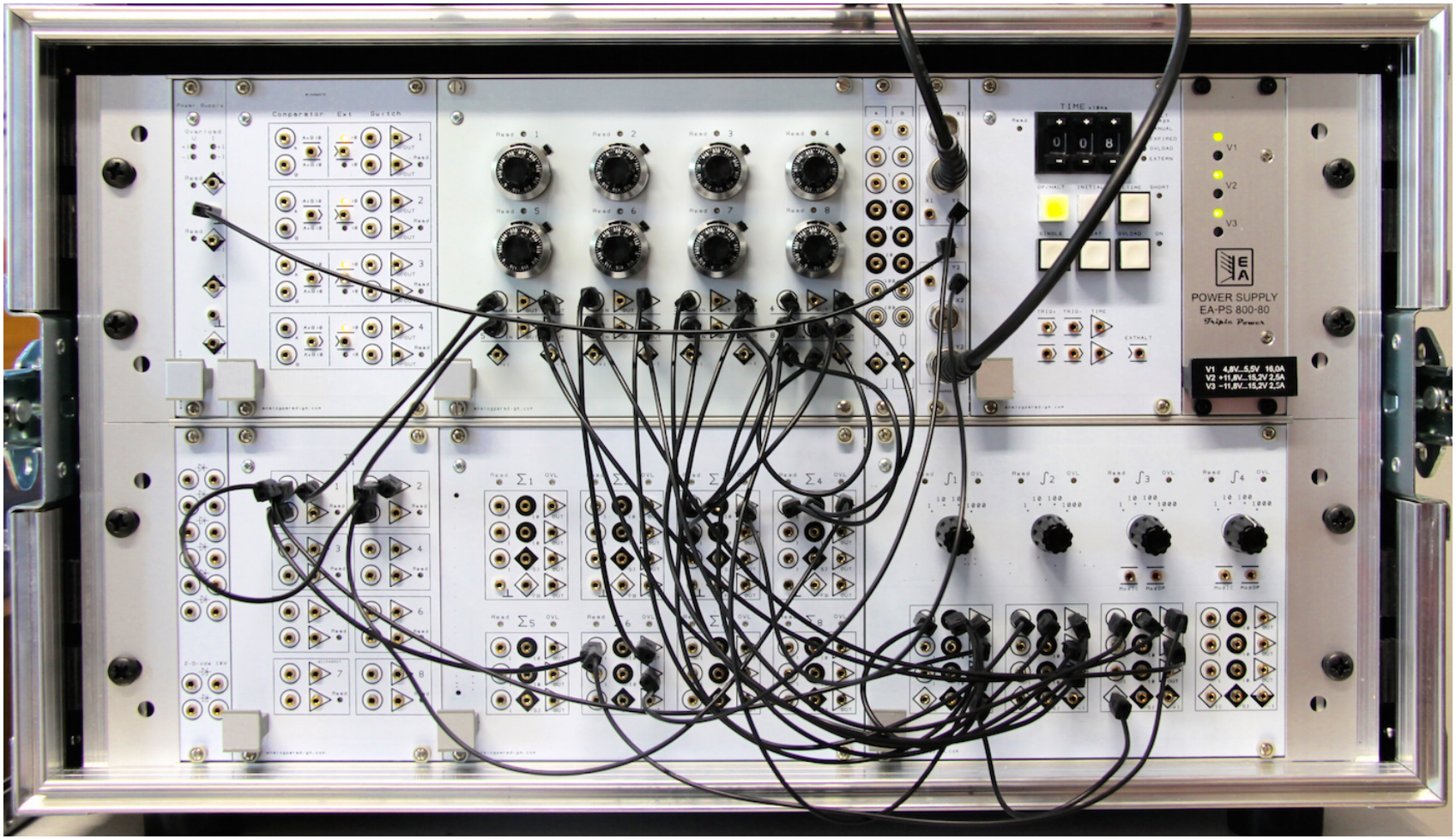
The archetypal chaotic system; discovered by Edward Norton Lorenz in 1963:[4]

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



[4] Siehe [Lorenz, 1963].

That is how an implementation on a classic analog computer looks like:

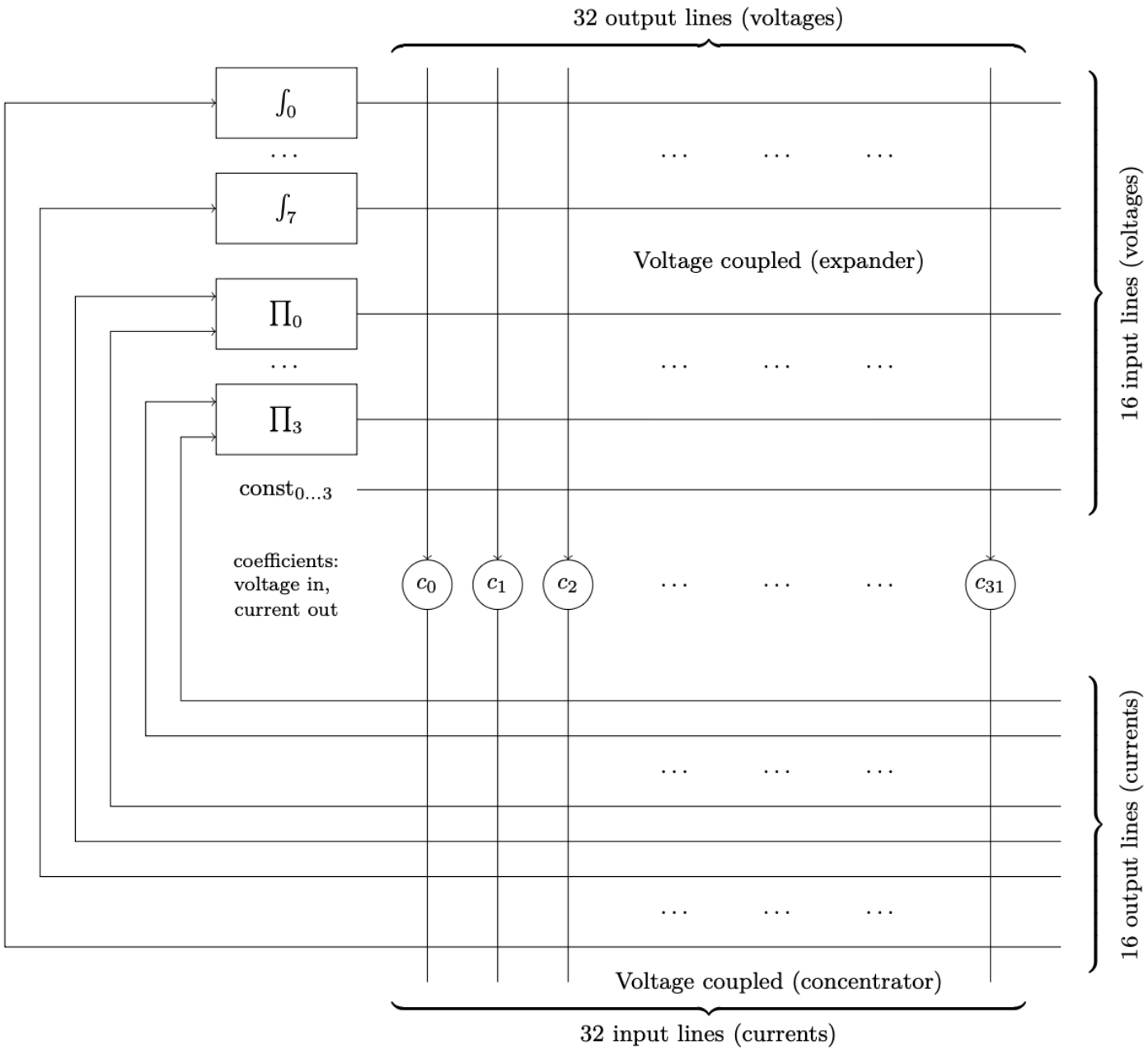


Autopatch

- Clearly patching something like this manually is not really scalable...
- We need an *autopatch*.
- This idea is rather old but was unfeasible until recently. Even the ROCKEFELLER electromechanical differential analyzer had an autopatch system, but it also had few computing elements in all.
- Problem: An $n \times m$ crossbar matrix does not scale!
- Idea: Utilize the fact that interconnects are rather sparse (like a classic telephone exchange system).

This is the basic idea of anabrid's autopatch system:

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-Attractor revisited

On such a modern reconfigurable analog computer things are much simpler:

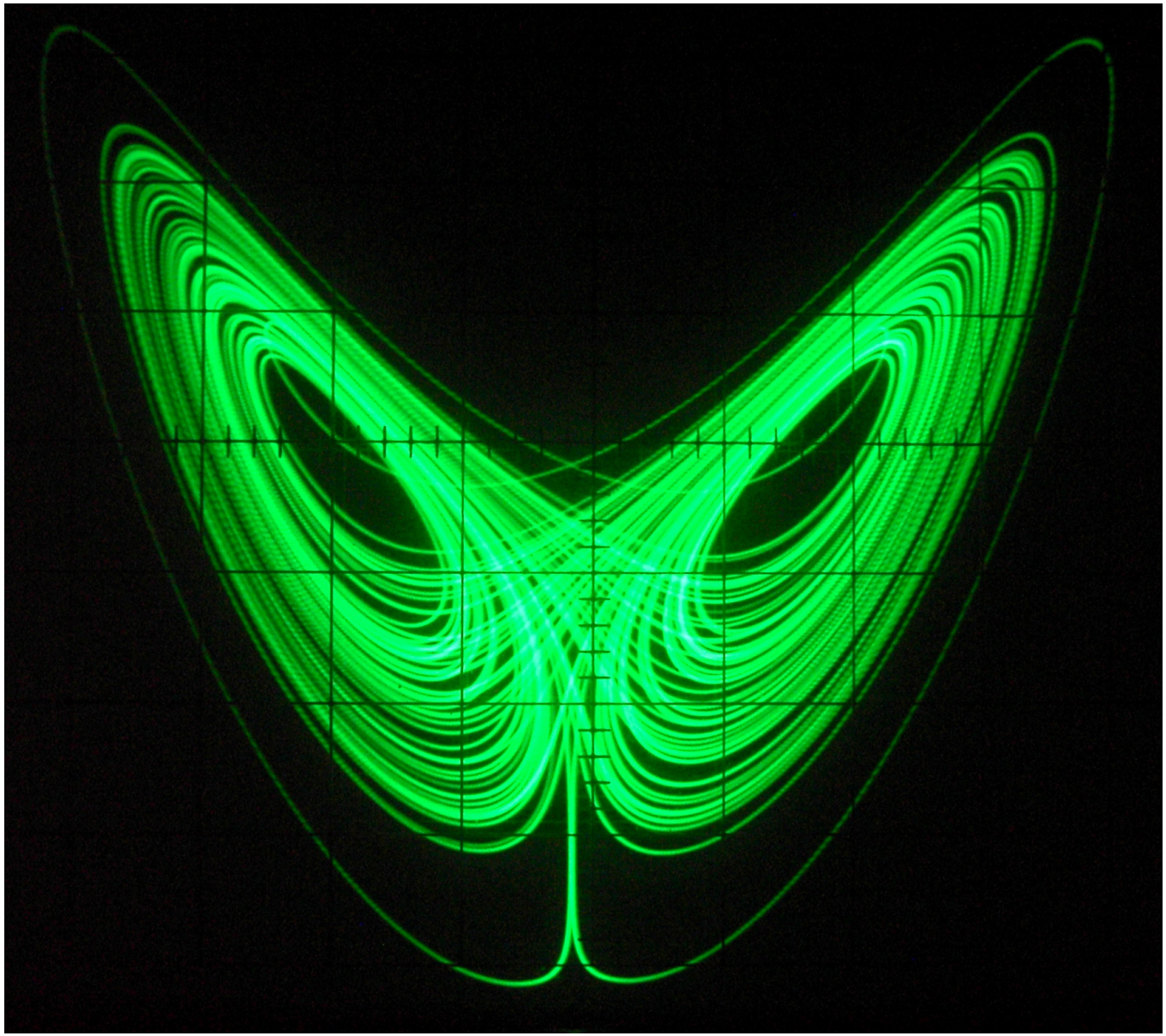
```
fn X(t);
fn Y(t);
fn Z(t);

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-Attractor



Modern developments

Modern approaches must address the following issues of classic analog computers:

- The patch panel belongs into a museum – modern analog computers must be automatically reconfigurable in a few milliseconds at most.
- This requires a *Domain Specific Language* (*DSL*) with associated compiler and libraries to specify an analog computer program which we have at anabrid.
- Tight integration of the analog computer in a digital environment is necessary which requires
  - low-level software (device drivers etc.) – one particular problem are slow interrupt latencies of modern operating systems
  - libraries, numerical algorithms suitable for hybrid computer approaches ...
- The computing elements should have very high bandwidth (ideally in the range of several 10 Mhz to 100 MHz) to achieve short solution times (a few μs at maximum).

Fast? Energy efficient?

Fast? Energy efficient?

How fast, how energy efficient are analog computers actually?

- That is not trivial to answer. It requires
- benchmarking realistic problems on classic digital systems (CPU, GPU), ideally using various numerical methods and then doing the same computation on an analog computer.
- To put the numbers obtained that way into perspective, question like precision and accuracy have to be answered, too.
- That is beyond the scope of this slide set but we are working on a comprehensive academic publication.
- However, the following slide shows one example problem from our suite of benchmarking problems:

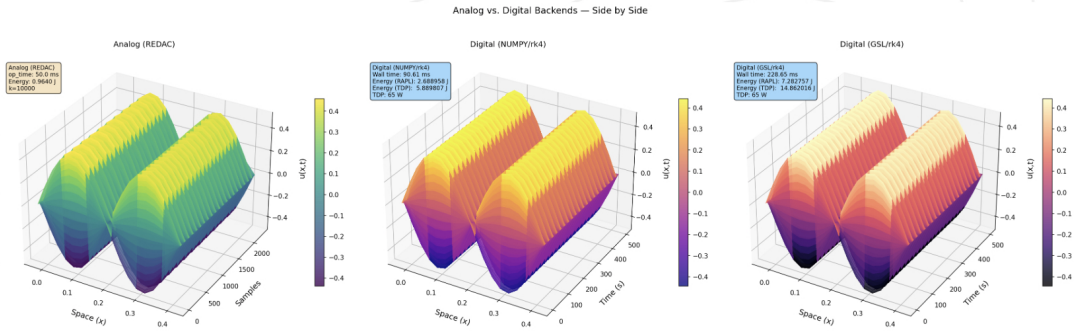
Fast? Energy efficient?

The problem. The 1D wave equation drives every loudspeaker, every guitar string, every transmission line and every torsional shaft mode. Solved on a 56-node analog mesh as a chain of integrators that physically “is” the wave.

Real-time vibroacoustic twins for active noise cancellation in EVs, aerospace skins, and structural-health monitoring on the edge.

	CPU (NumPy / GSL)	REDAC today	REDAC on-chip*
Time-to-solution	90.6 / 228.6 ms	50 ms	≈ 50 µs
Energy per run	2.69 / 7.28 J	0.96 J	≈ 0.96 mJ

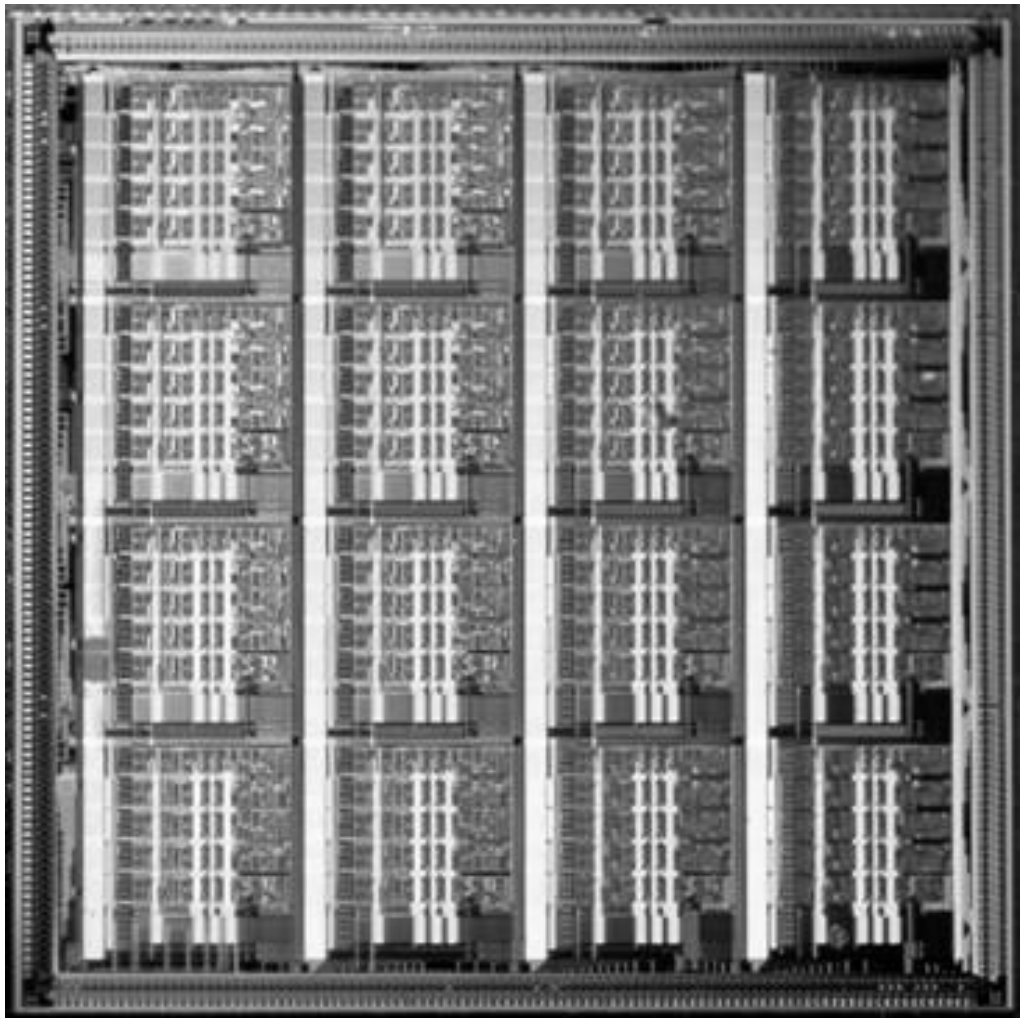
✔ Up to ≈ 7.6× less energy than CPU GSL; same standing-wave pattern, all three solvers.



Academic projects

Glenn E. R. Cowan (2005)

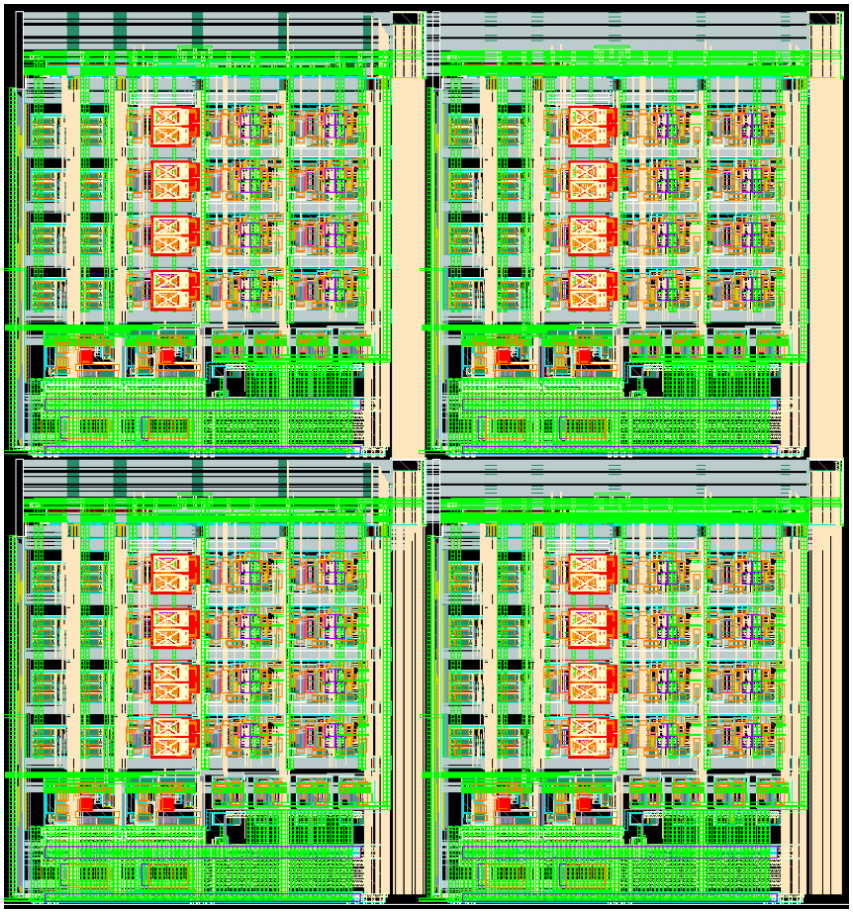
First fully reconfigurable analog computer on chip – mainly used for stochastic differential equations:[5]



[5] See [Cowan(2005)].

Ning Guo et al. (2016)

More advanced analog computer on chip:[6]



(Picture source: `http://yipenghuang.com/wp-content/uploads/2016/07/v2_chip.png`, retrieved 2019-09-19.)

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

Quantum inspired analog computing

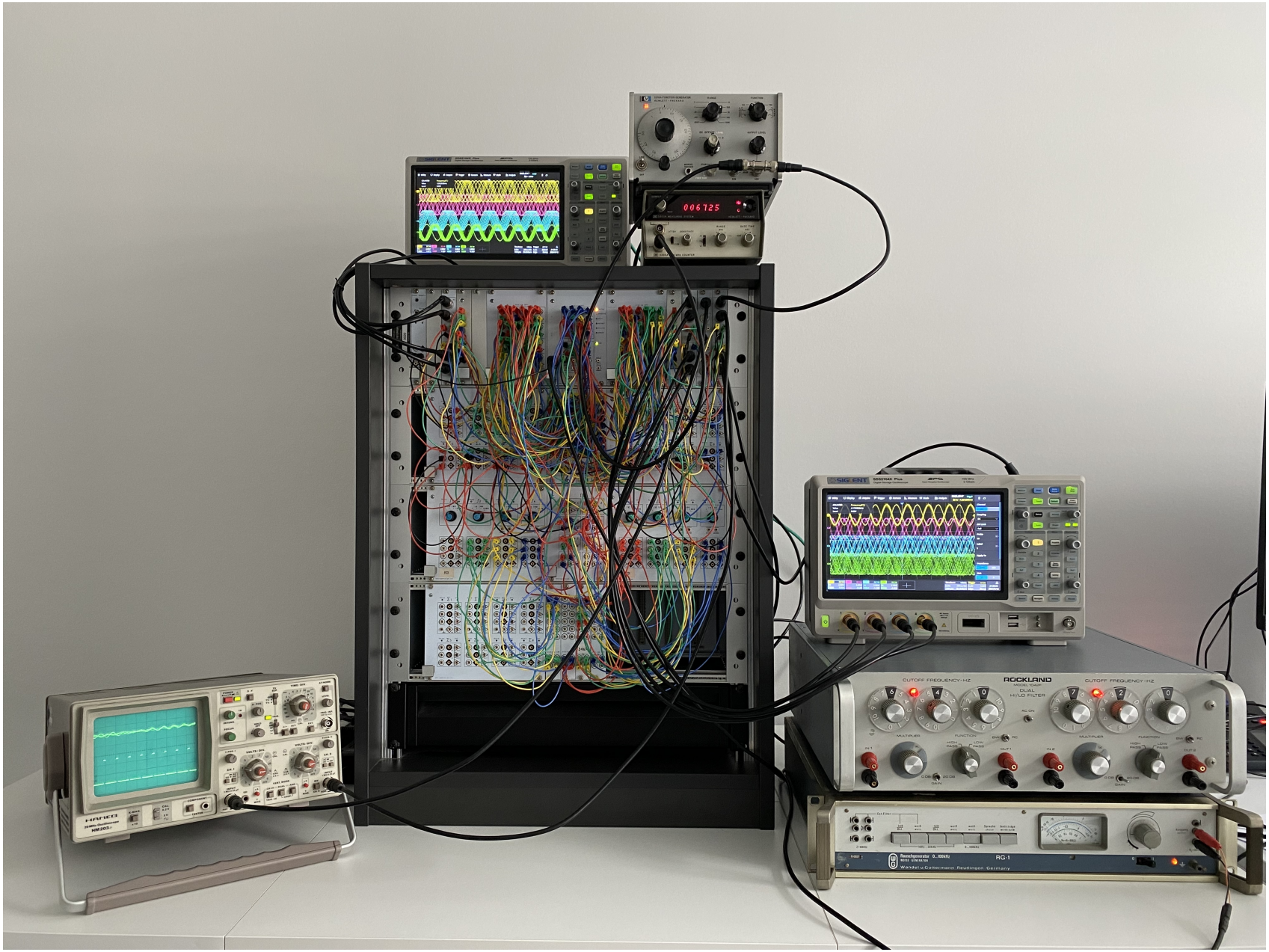
Oscillator based Ising machines

There are ISING formulations of many NP-complete and NP-hard problems (see [LUCAS(2014)]).

The basic idea is to implement an ISING model using coupled oscillators, representing information as the phase relationship of their respective output with respect to some reference oscillator.



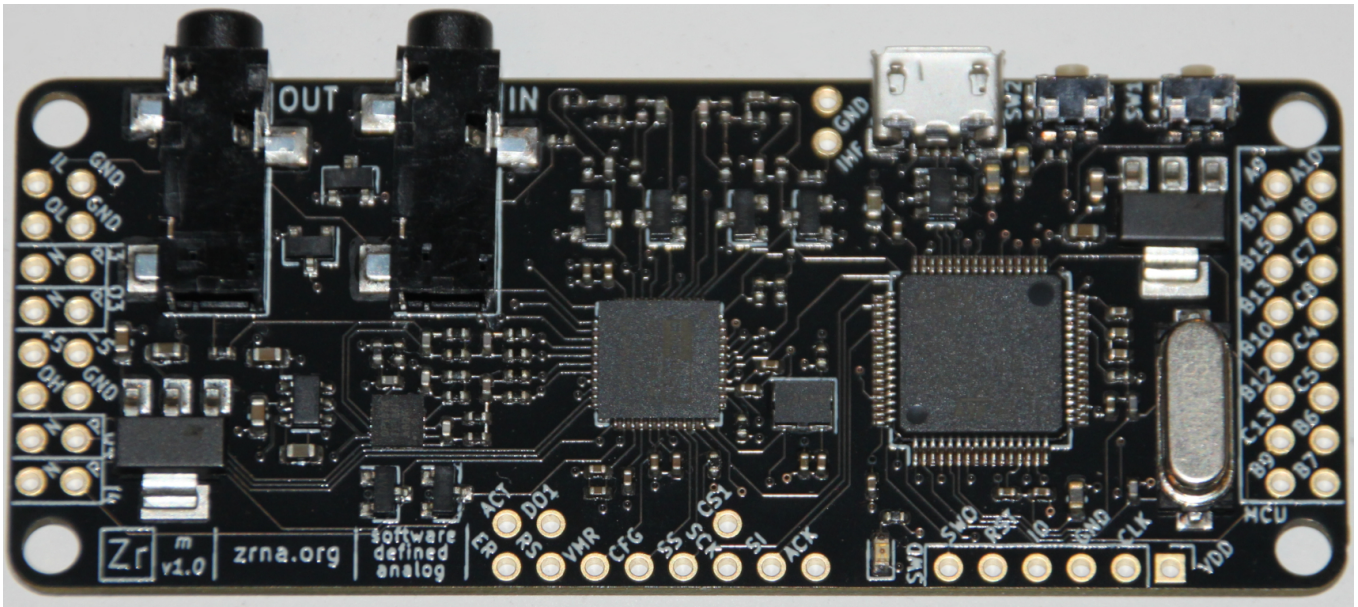
Oscillator based Ising machines



Application area: Edge processing

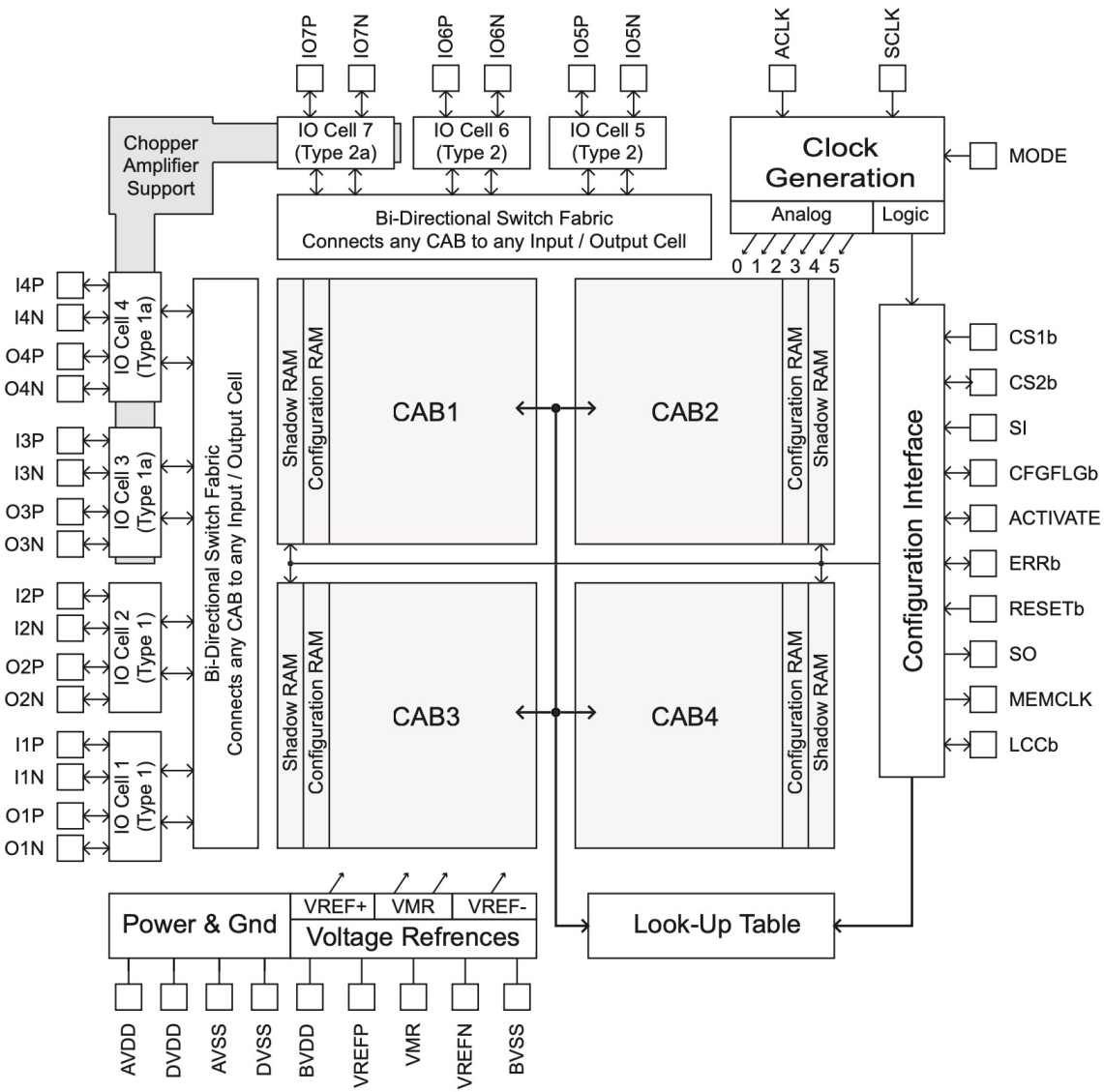
Okika

Okika (formerly Anadigm) manufactures FPAAs (using switched capacitors for the central interconnect structure) mainly aimed at the signal processing market. A typical application example is shown here (`zrna.org`):



At its heart are *configurable analog blocks* (*CAB*s) which are internally connected by switched capacitors.

Okika



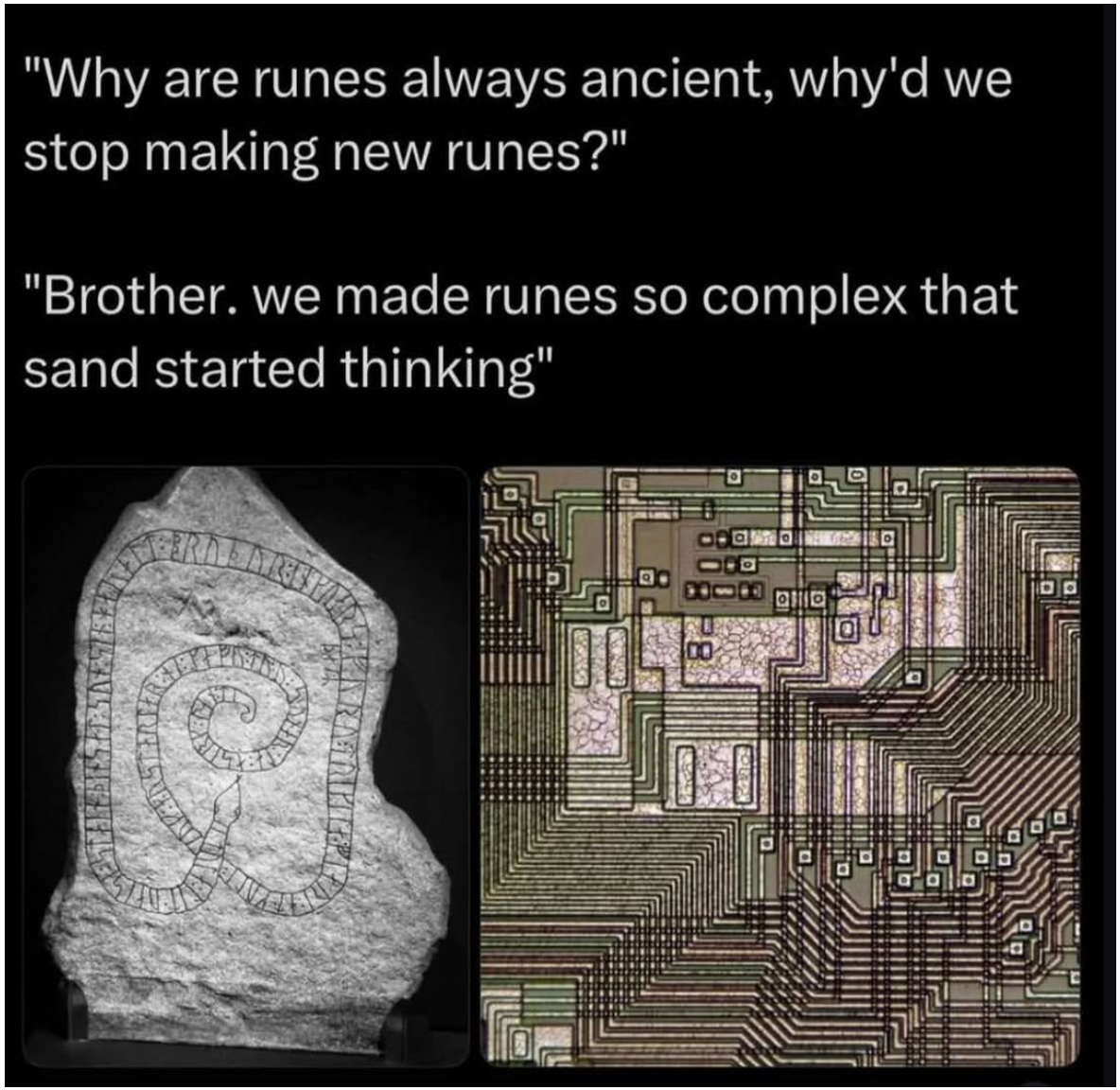
(Cf. [Anadigm(2006)].)

Application area: AI

AI might be the commercially most promising area of application for analog computers:

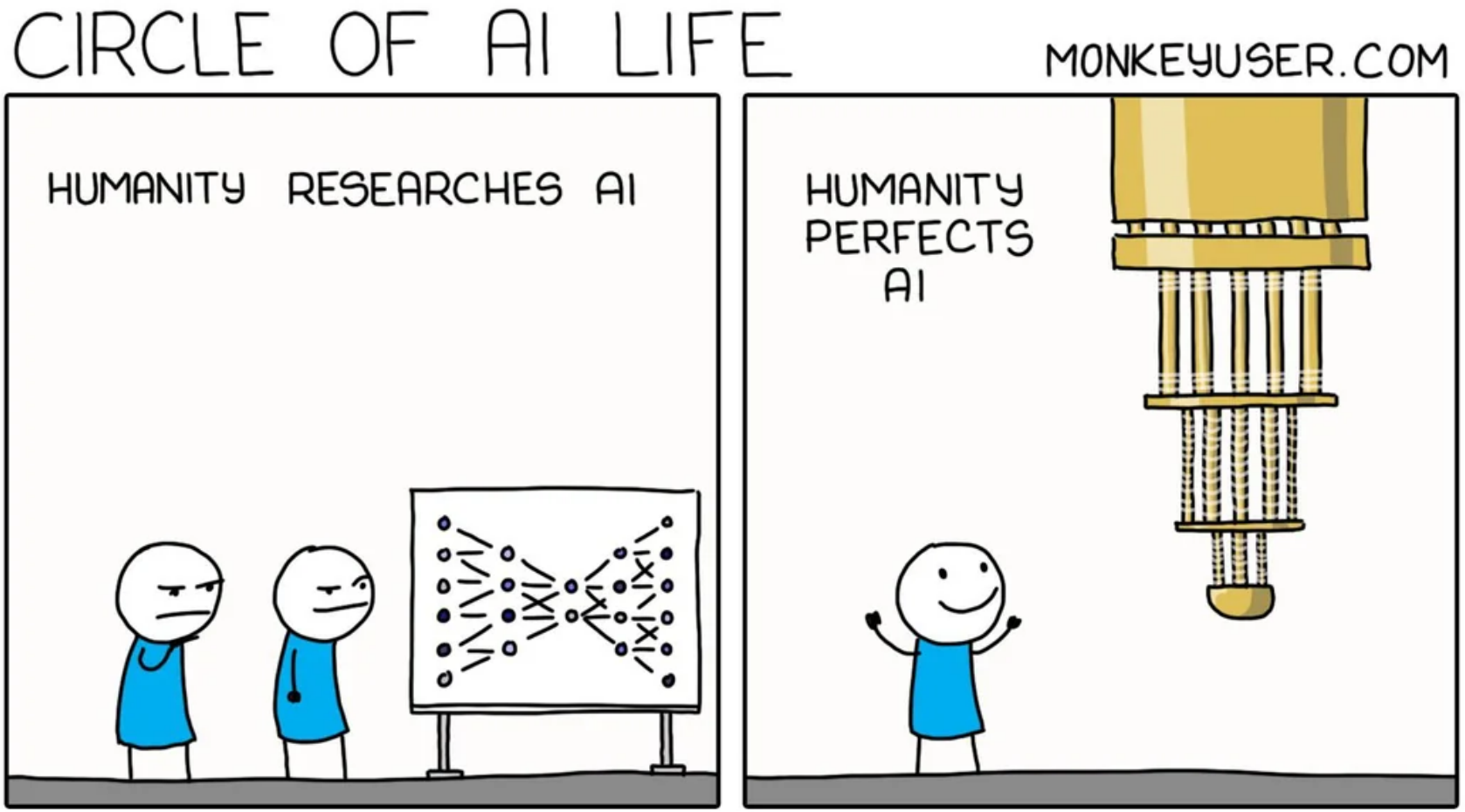
- In the most simple case one can implement linear algebraic operations on matrices and vectors using arrays of (configurable) resistors (pretty boring :-)).
- One can implement more biology-inspired systems mimicking the spiking and/or bursting behavior of actual neurons (neuromorphic computing – much more interesting :-)).
- Another approach would be to use networks of damped harmonic oscillators as the substrate for AI system (highly interesting :-)).

Runes



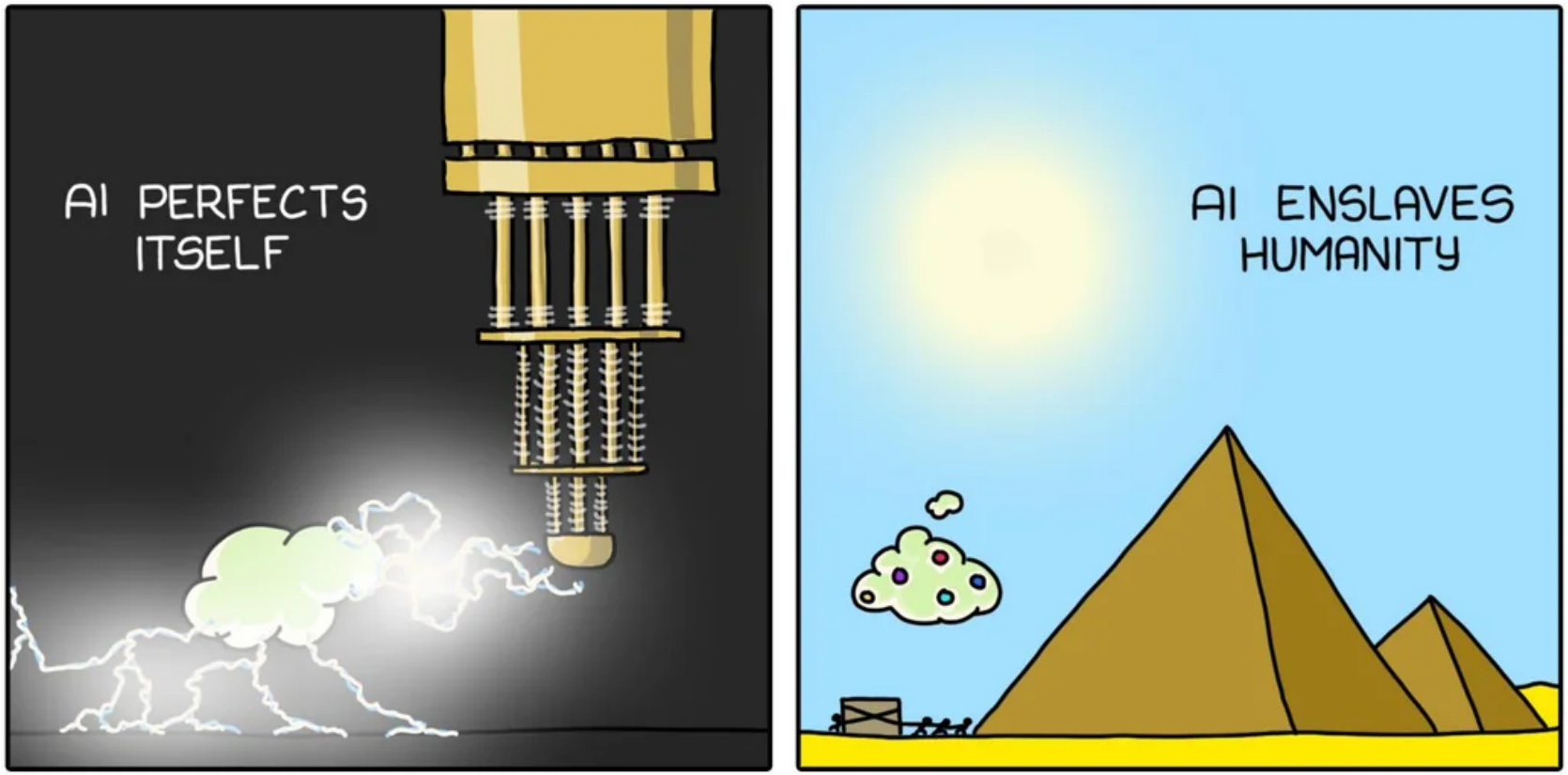
Unfortunately, I cannot find the source anymore...

Circle of AI life



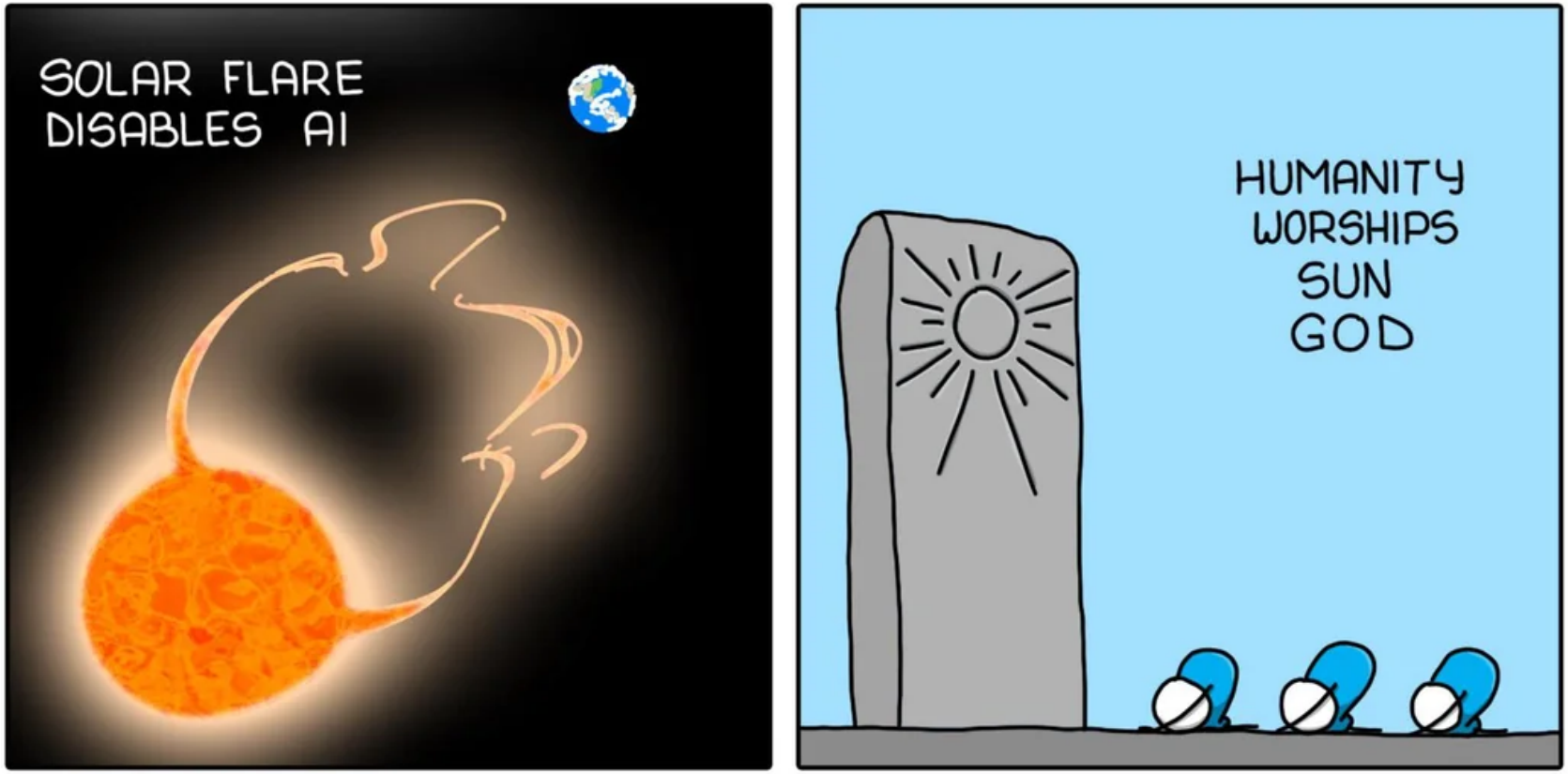
https://www.monkeyuser.com/2020/circle-of-ai-life/, retrieved 14.09.2026

Circle of AI life



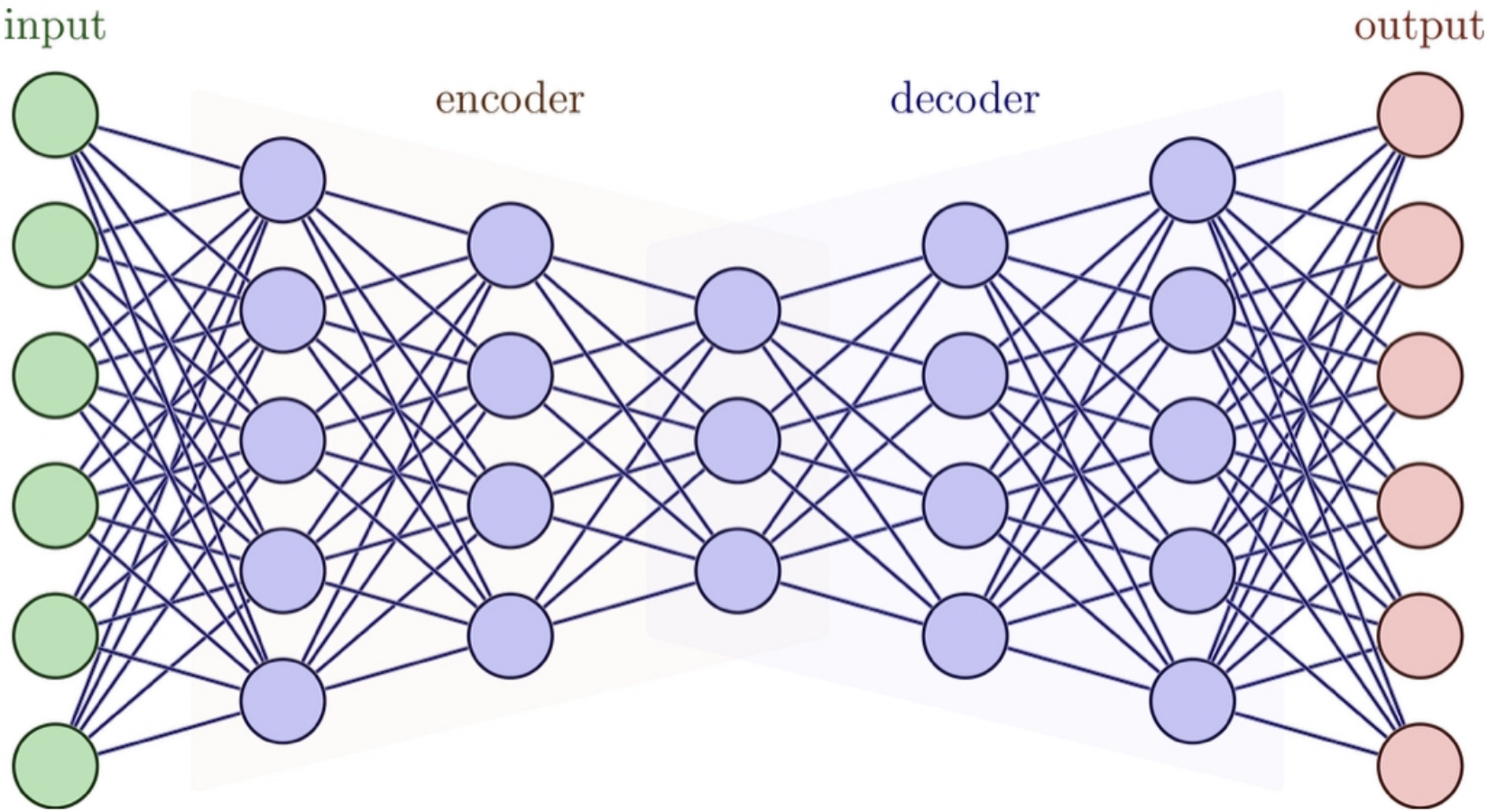
https://www.monkeyuser.com/2020/circle-of-ai-life/, retrieved 14.09.2026

Circle of AI life



https://www.monkeyuser.com/2020/circle-of-ai-life/, retrieved 14.09.2026

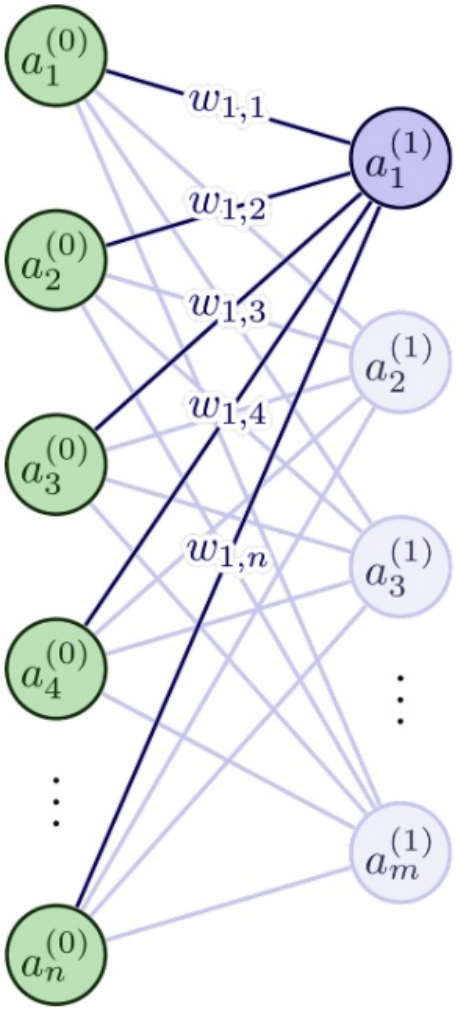
Basic idea



https://tikz.net/neural_networks/, retrieved 09.09.2026.

Basic idea

The is basically a linear algebraic nightmare requiring an incredible amount of energy for training, etc.



$$= \sigma\left(w_{1,1}a_1^{(0)} + w_{1,2}a_2^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$\mathbf{a}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{a}^{(0)} + \mathbf{b}^{(0)}\right)$$

https://tikz.net/neural_networks/, retrieved 09.09.2026.

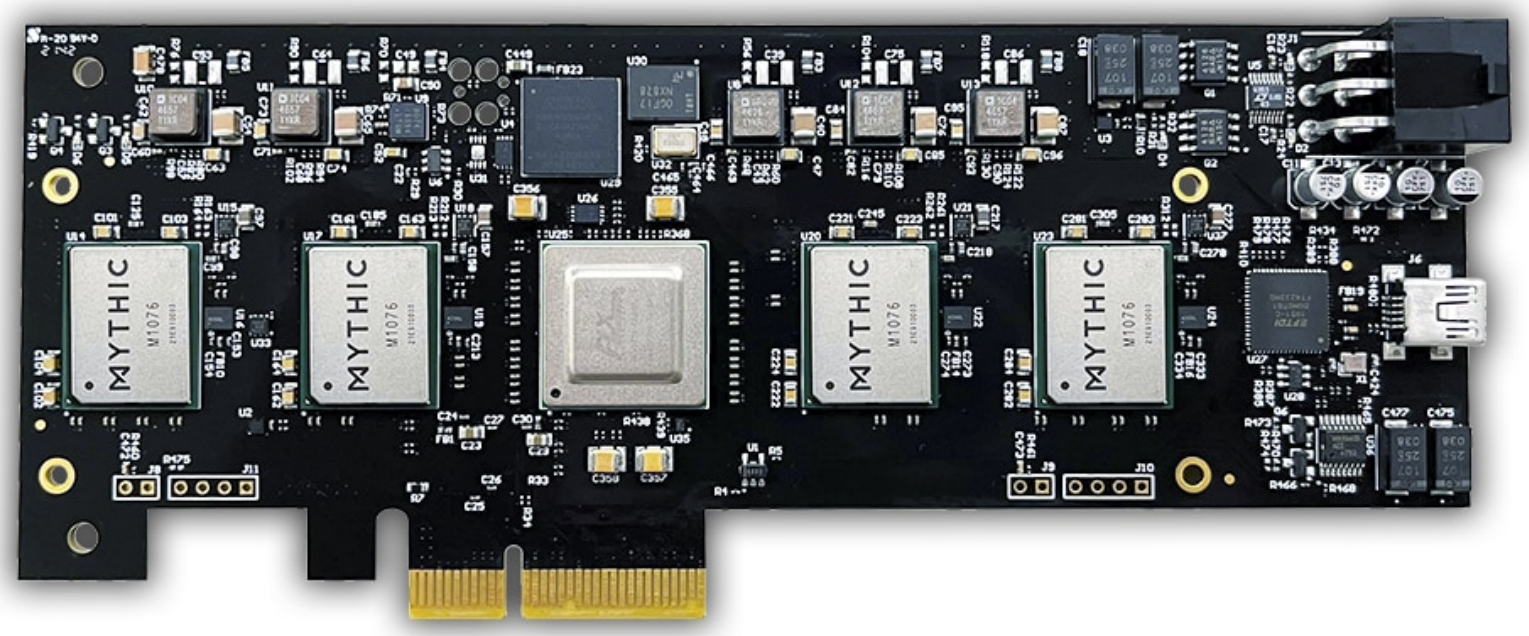
In general

There are various ways to implement AI systems on analog computers:

- Matrix/vector multiplication, etc., can be implemented using arrays of resistors.
- One could also (much more interesting :-)) use networks of coupled oscillators resembling neural networks.
- Even more biology related would be systems consisting of analog computer implementations of “realistic” neurons (neuromorphic computing).

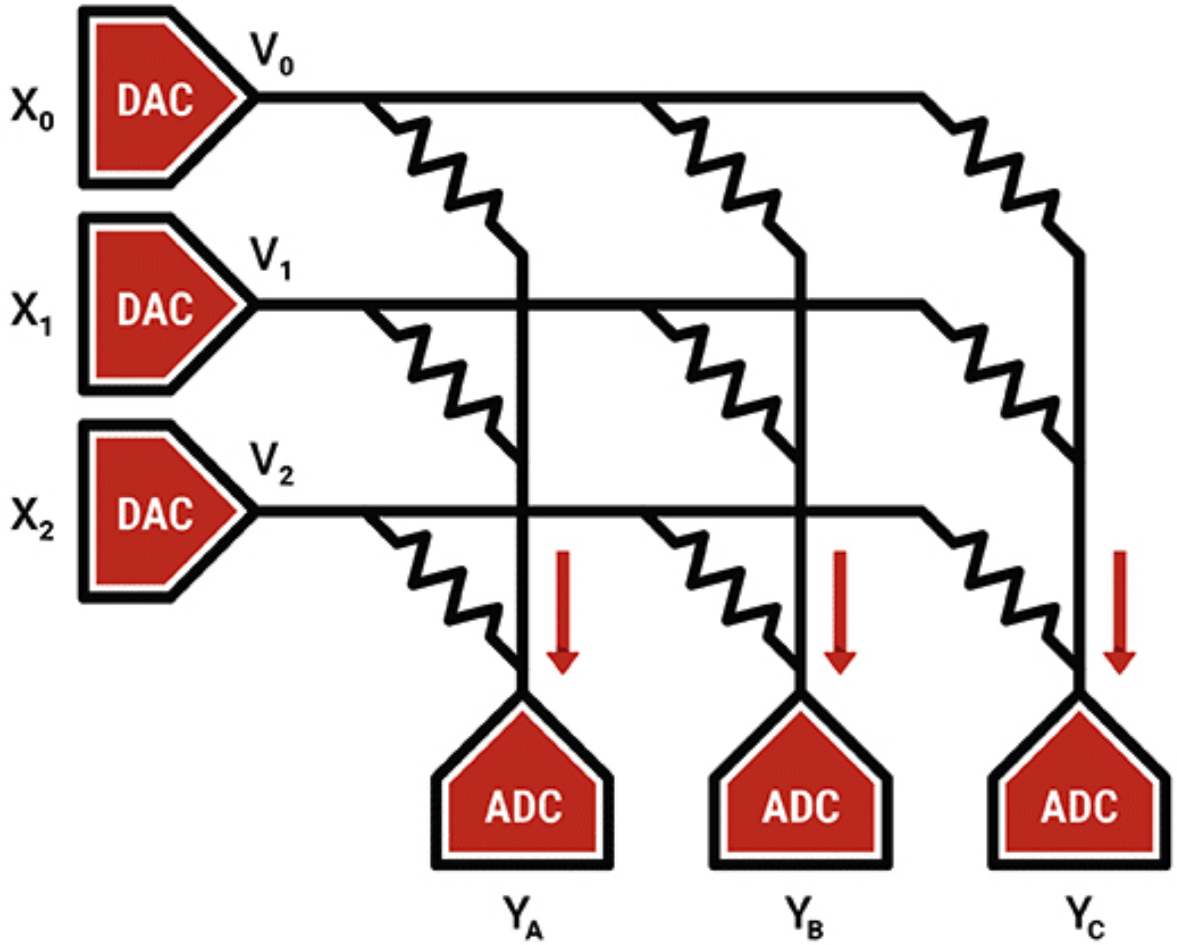
MYTHIC

MYTHIC[7] develops *analog matrix processors* (*compute in memory*) aimed at AI applications:



[7] `https://mythic.ai`, retrieved 25.09.2022

MYTHIC



At the heart of these matrices are Flash memory cells for the synaptic weights.

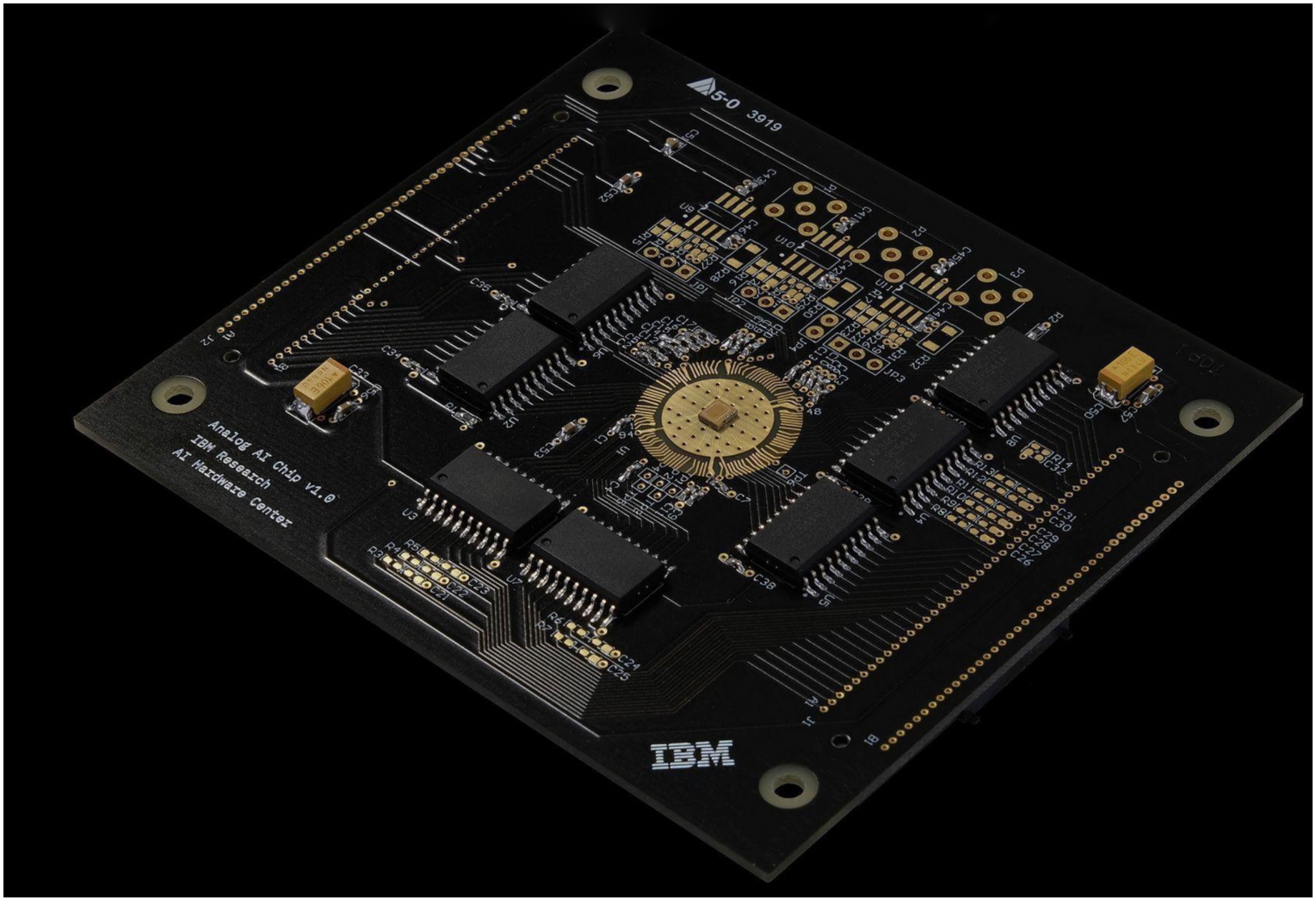
Source: `https://mythic.ai/technology/analog-computing/`, retrieved 26.06.2023

IBM has also developed an analog AI accelerator which was unveiled in 2022.[8]

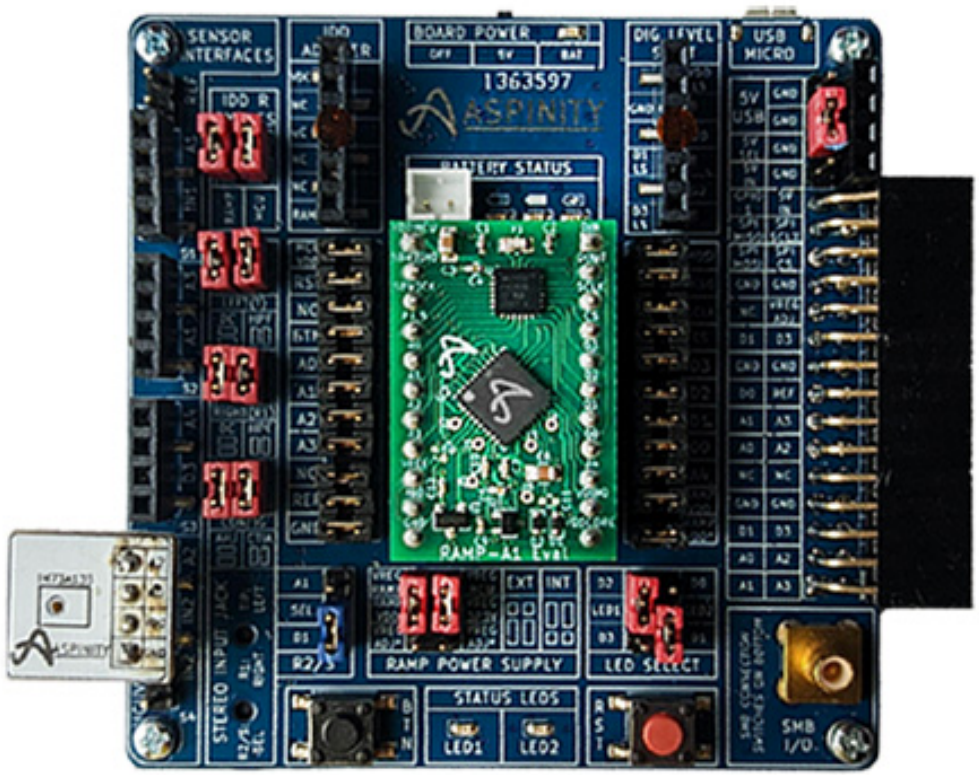
- *Resistive RAM* (*ReRAM*) in the form of *phase change memory* is used for the synaptic weights. The PCM state is switched between amorphous and crystalline.
- This experimental chip implements $35 \cdot 10^6$ PCM cells.

[8] `https://research.ibm.com/blog/the-hardware-behind-analog-ai`, retrieved 25.09.2022

IBM

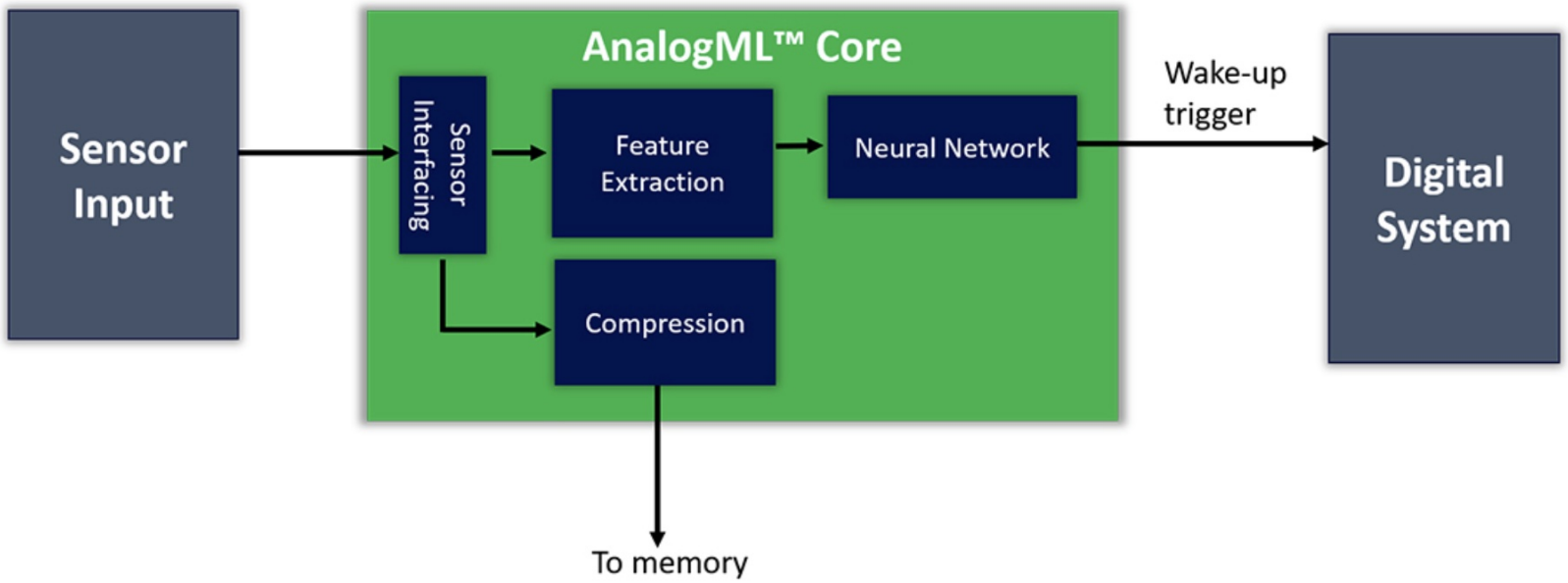


Aspinity[9] develops special analog processor chips for voice recognition, acoustic event and vibration classification:



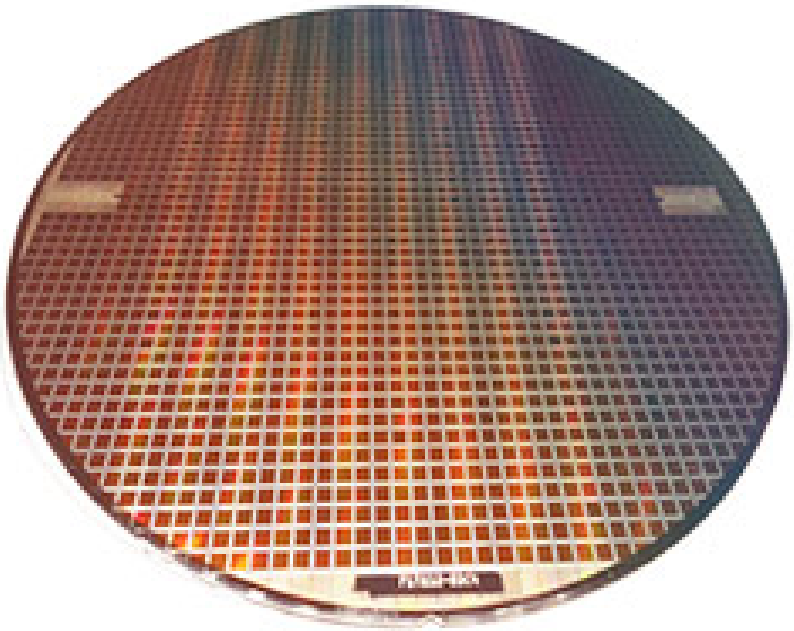
[9] https://www.aspinity.com, retrieved 25.09.2022

Aspinity



Source: `https://www.aspinity.com/AnalogML-Core`, retrieved 26.06.2023

Aspinity



Aspinity
has also developed the
Reconfigurable Analog Modular Processor
(*RAMP*) which is
aimed at neuromorphic
computing
(see `https://www.aspinity.com/RAMP-Technology`,
retrieved 26.06.2023).

Philosophical question

Here is a rather philosophical question:

Can a "real" artificial intelligence by implemented by simple linear algebra (with some nonlinearity but nevertheless...) only?

Why are biological neurons so complex? Why do they exhibit such a rich and dynamic behavior? There must be a reason for this, given the millions of years nature had time to optimize neural networks...

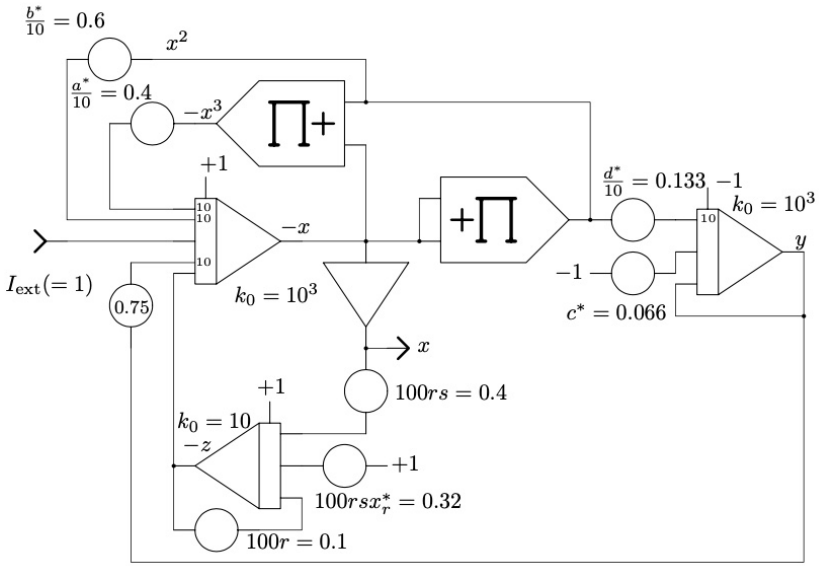
A rather good neuron model is due to Hindmarsh and Rose:[10]

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

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

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

with $a = 1$, $b = 3$, $c = 1$, $d = 5$, $r = 10^{-3}$, $s = 4$, and $x_r = -\frac{8}{5}$.





[10] See [Hindmarsh et al. 1982], [Hindmarsh et al. 1984], [Izhikevich 2007, pp. 123 ff.], and [Ulmann 2021].

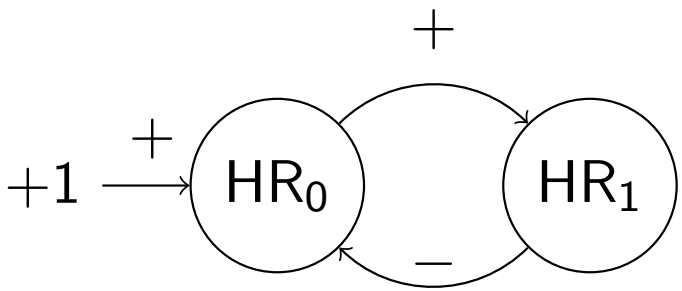
This model shows spiking and bursting behavior!

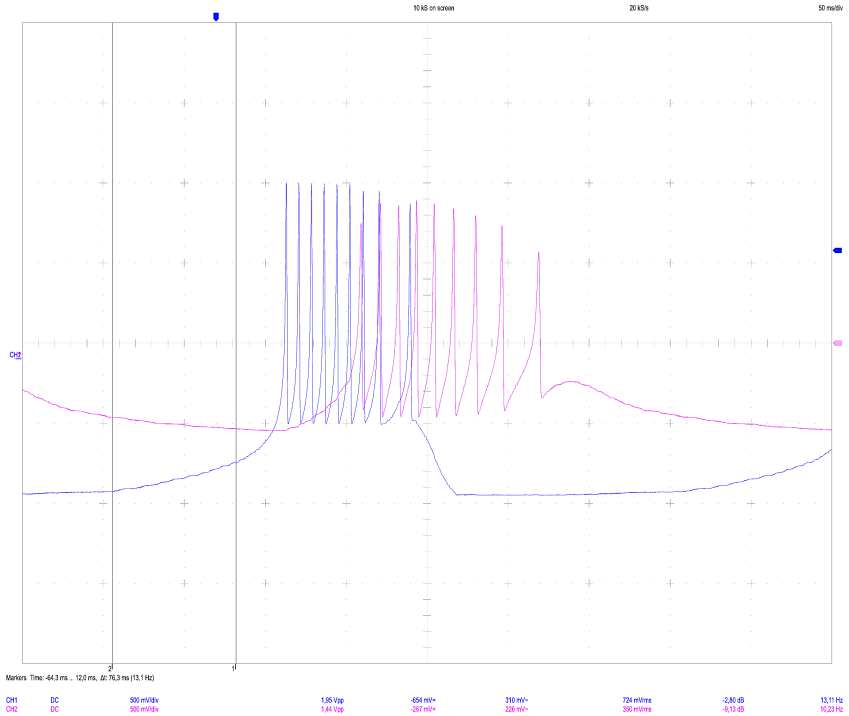
A digital implementation would have to solve three coupled nonlinear differential equations, a rather tedious and CPU intensive task. For an analog computer this is a breeze...

Using such neurons it is now possible to build more complex systems with very biology-like behavior:

A simple example

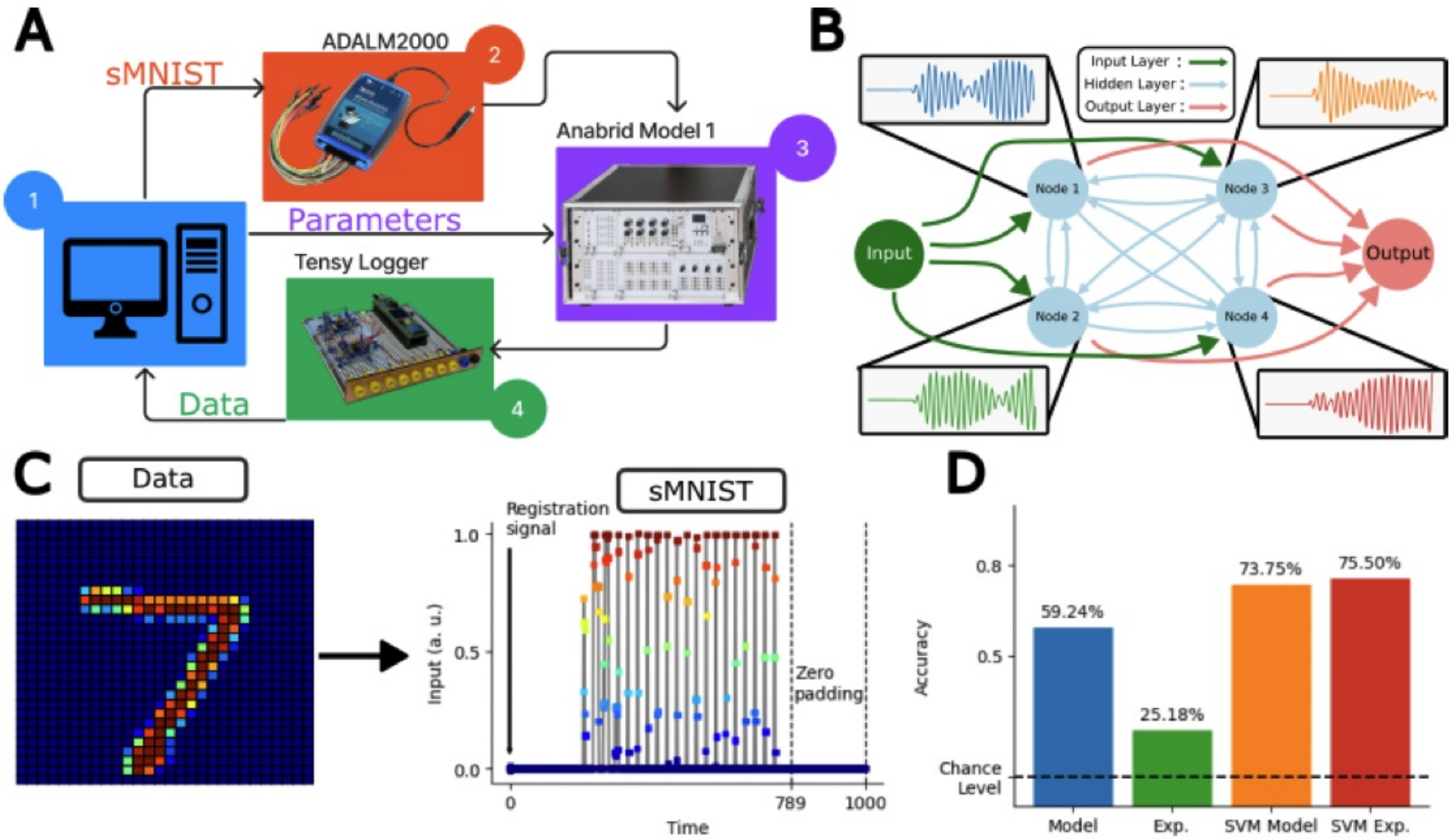
Two neurons coupled with each other, one connection is excitatory and one is inhibitory:





Why not use millions and billions of such neuronal models to create large networks resembling biological systems to a high degree?

See (e. g.) [Carvalho et al. 2025]:



Application area:

General purpose analog computing

anabrid GmbH (Germany) was founded in 2020 with the goal of developing, producing and marketing a high-performance reconfigurable general purpose analog computer on chip.

We work on and already have

- interconnect structures for large analog computers,
- a DSL with associated compiler,
- hard- and software-support for high-performance hybrid computers,
- numerical algorithms for hybrid computers, and
- quantum inspired oscillator based ISING machines as well as
- computing elements in 65 nm CMOS technology.

Model-1

Our first product was the *Model-1* modular analog computer which has been sold worldwide to a variety of universities and companies:[11]



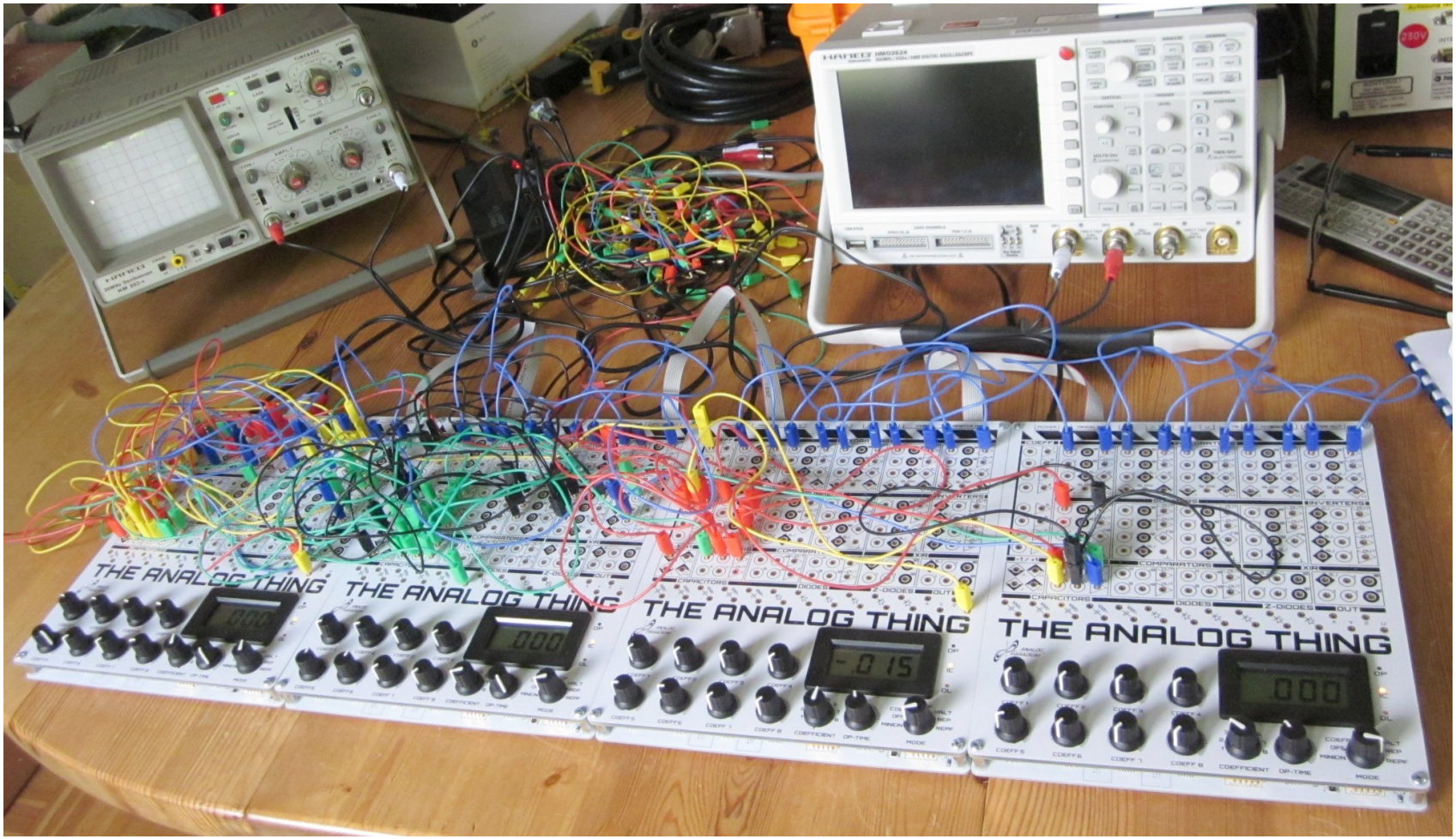
[11] See [Ulmann (2010)].

THE ANALOG THING

To bring analog computing to schools, universities and into the hands of hobbyists, we have created an open-hardware project:



THE ANALOG THING



LUCIDAC

Our latest system is already fully reconfigurable – no patch cables anymore – the *LUCIDAC*:

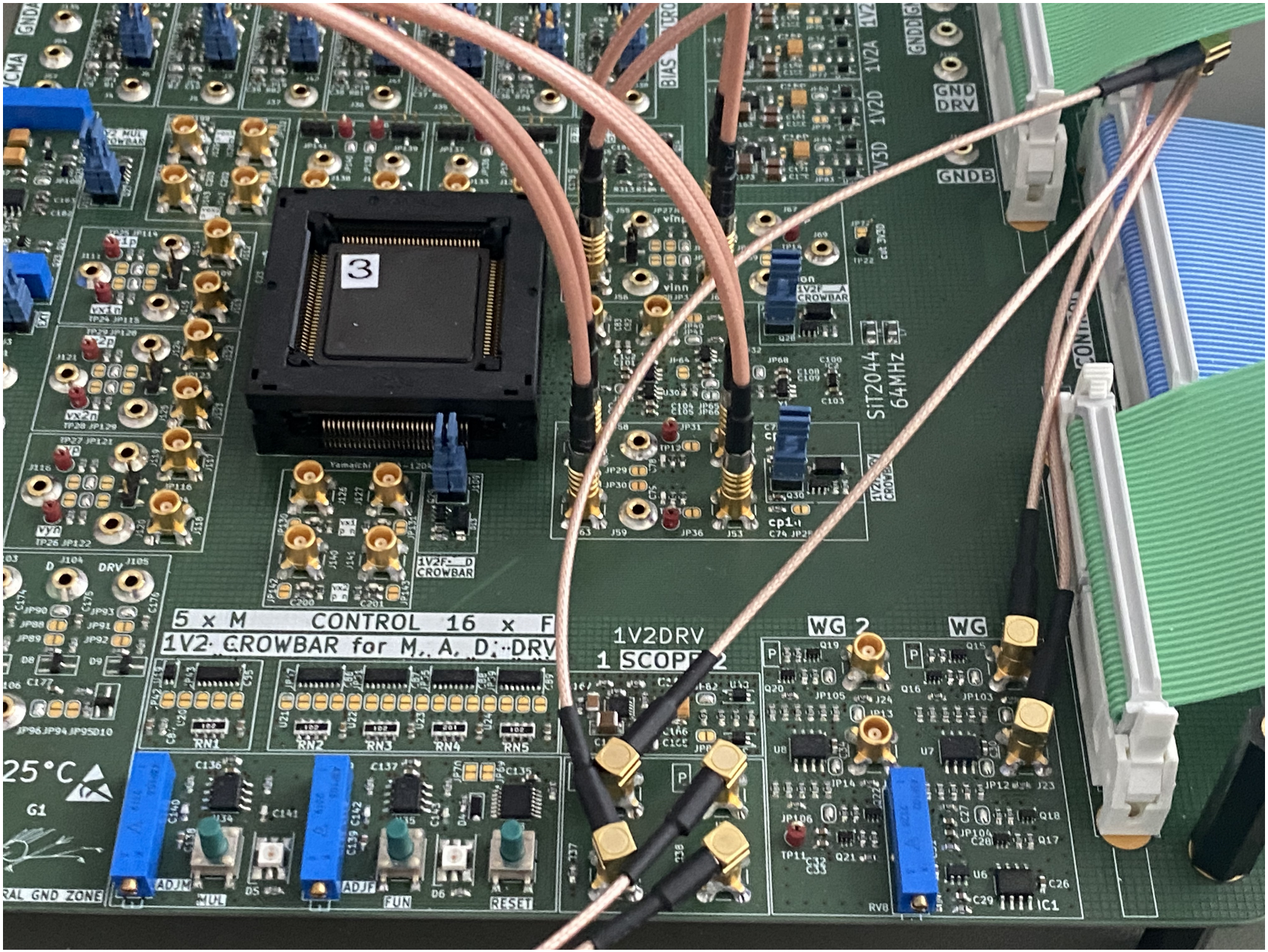


REDAC

REDAC – the analog supercomputer in the cloud: 1000 integrators, 500 multipliers, 4000 coefficients, fully reconfigurable.



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

Thank you!

Thank you for your interest and

happy analog computing!

The author can be reached at
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

Bibliography



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`, retrieved 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`, retrieved 21.01.2025
- Bernd Ulmann, *The Hindmarsh-Rose model of neuronal bursting*, Analog Computer Applications, `https://analogparadigm.com/downloads/alpaca_28.pdf`, retrieved 14.01.2021
- Bernd Ulmann, *Analog and hybrid computer programming*, DeGruyter, 2nd edition, 2023

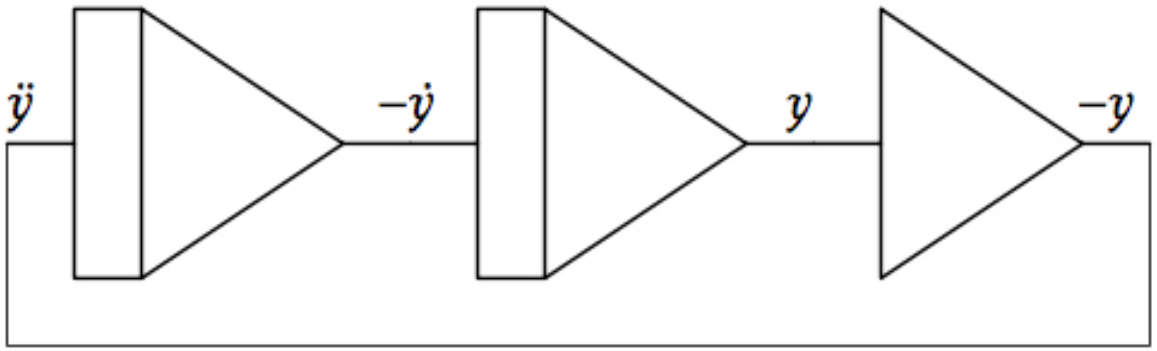
“Hello world!”

A simple example

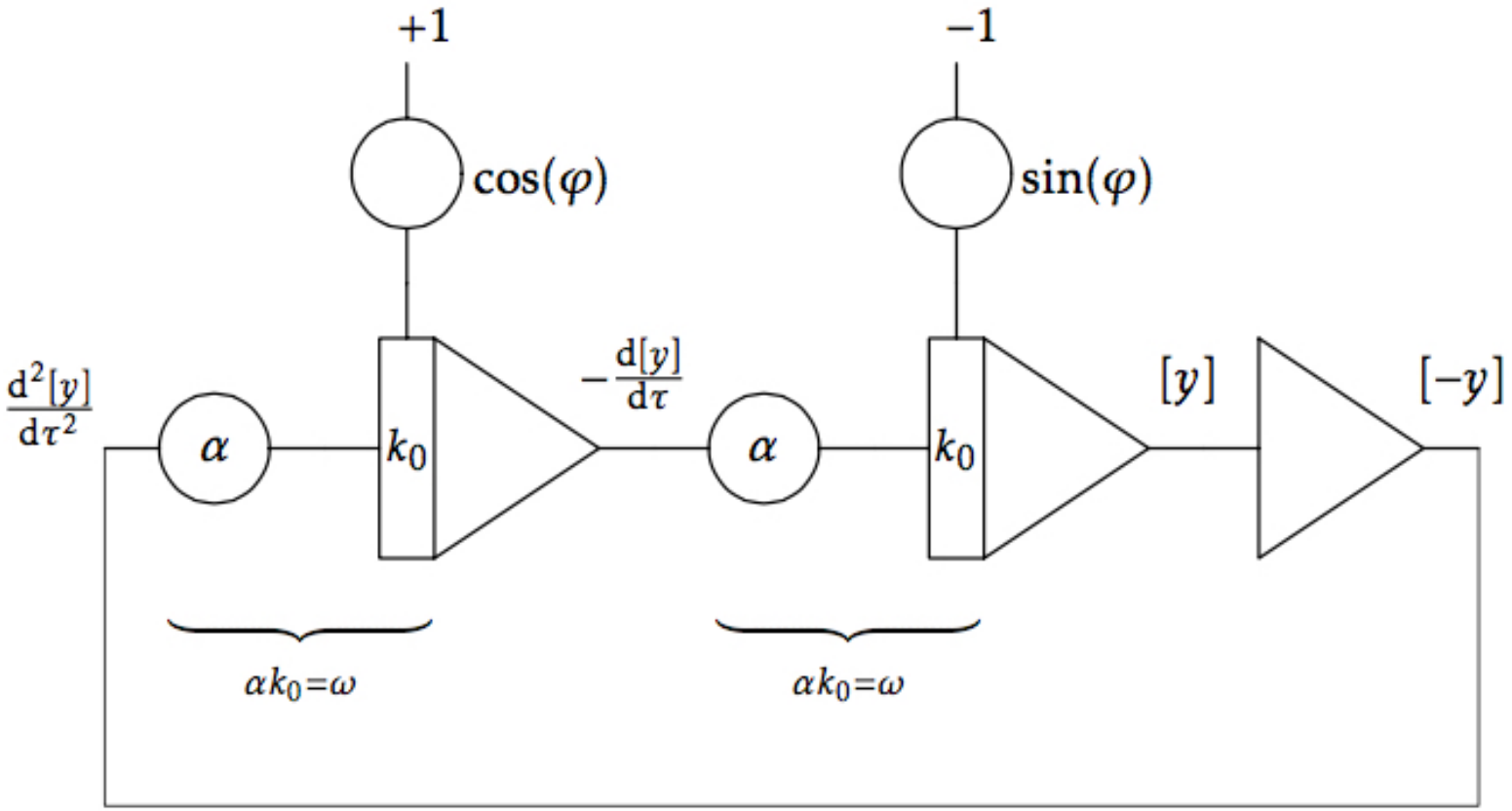
The "*Hello World*" of analog computing is a sine/cosine circuit based on ($\ddot{y}$ is already separated on the left hand side)

$$\ddot{y} = -y \tag{4}$$

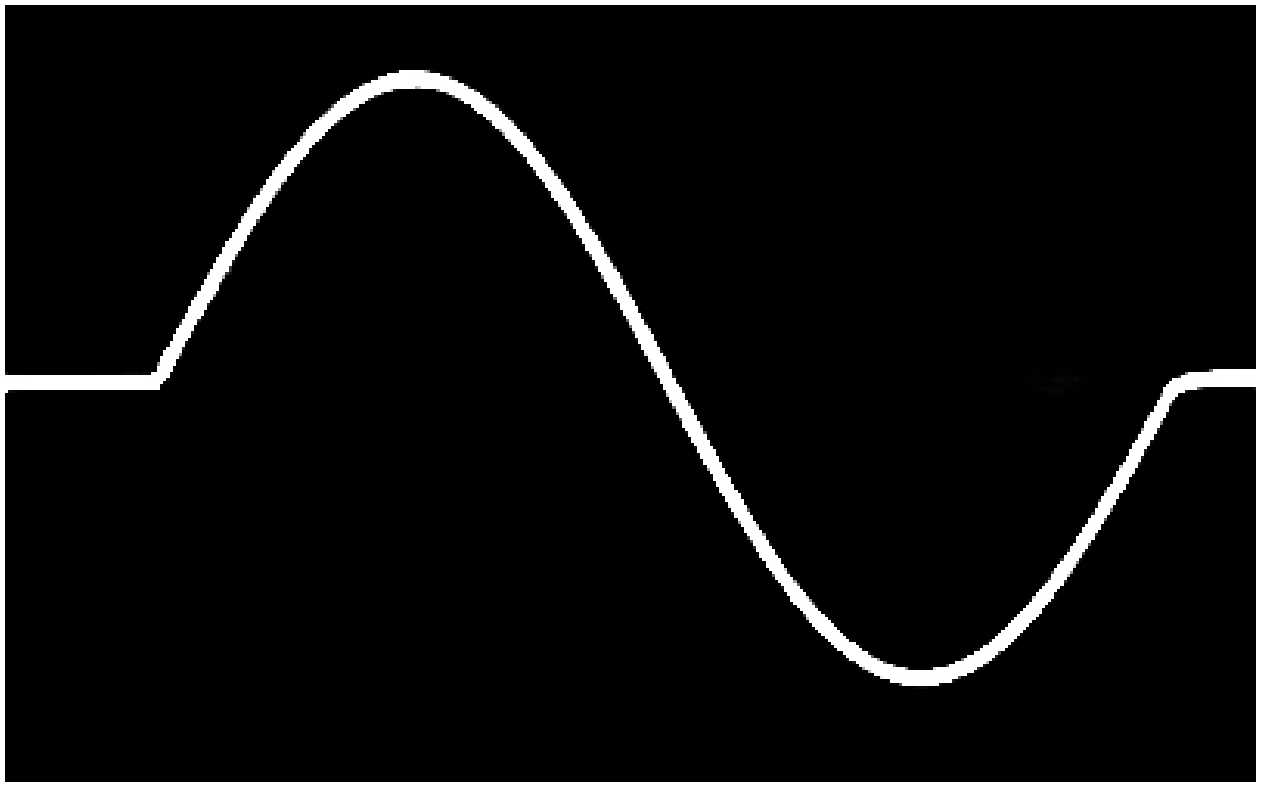
with suitable initial conditions. This leads to the following program sketch:



Taking things like the angular frequency ω as well as initial conditions into account yields the following complete program:



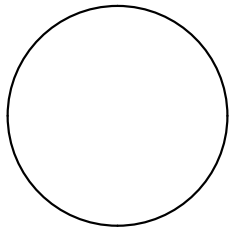
This yields the following output at the second integrator (actual screen shot – the analog computer was running for the time of one period as determined by ω):

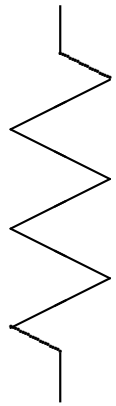


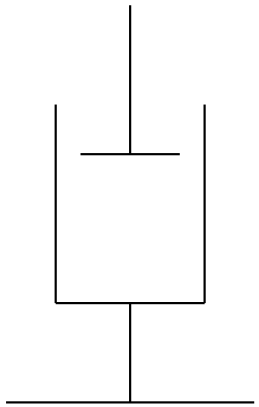
Mass-spring-damper system

Mass-spring-damper system

We will simulate a mass-spring-damper system consisting of these parts:

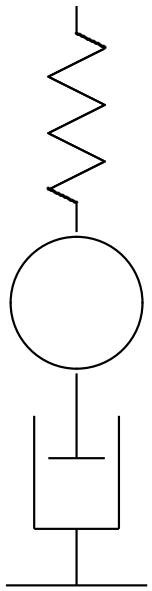






$F_m = ma = m\ddot{y}$ $F_s = sy$ $F_d = d\dot{y}$

Mass-spring-damper system



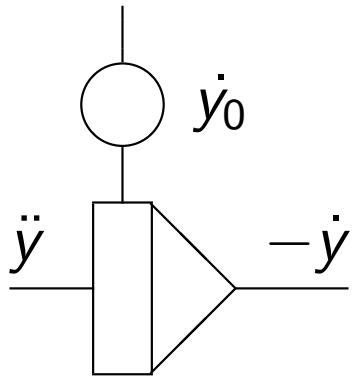
$$F_m + F_d + F_s = 0$$

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

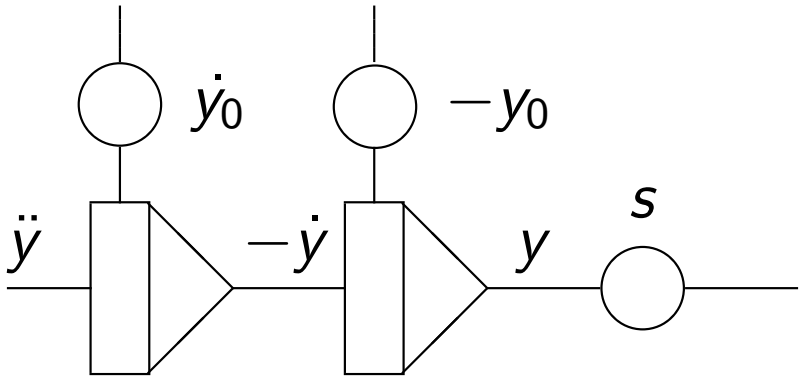
We have

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

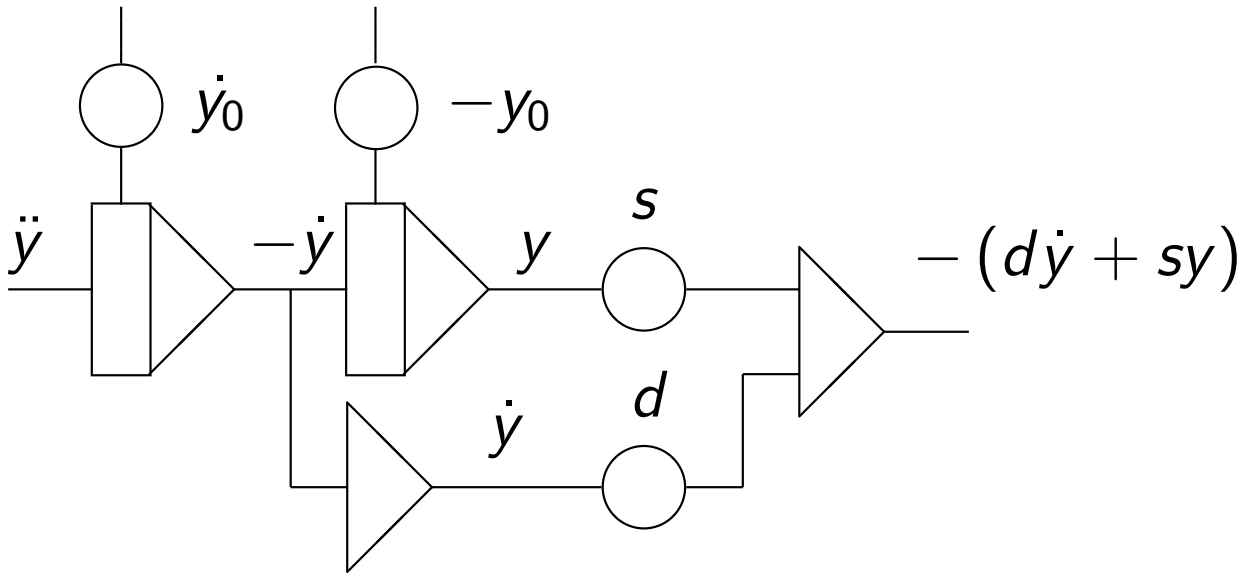
Integrating once yields:



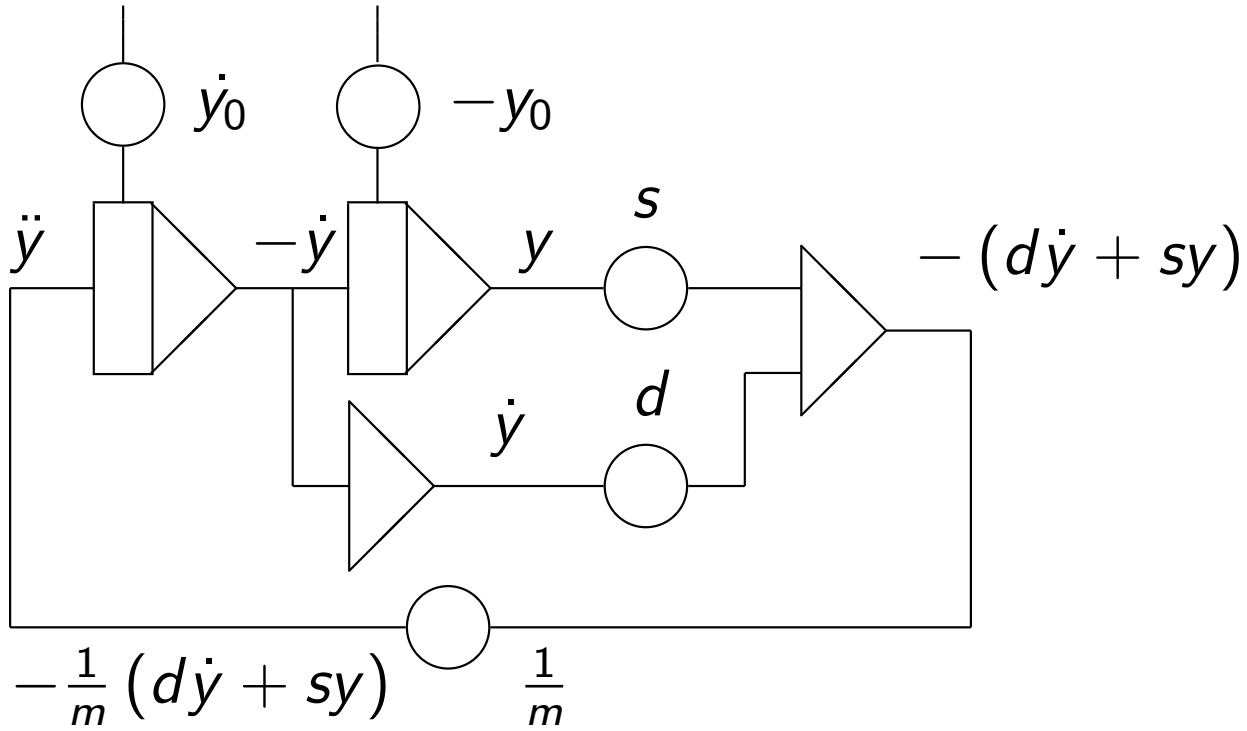
Integrating a second times results in:



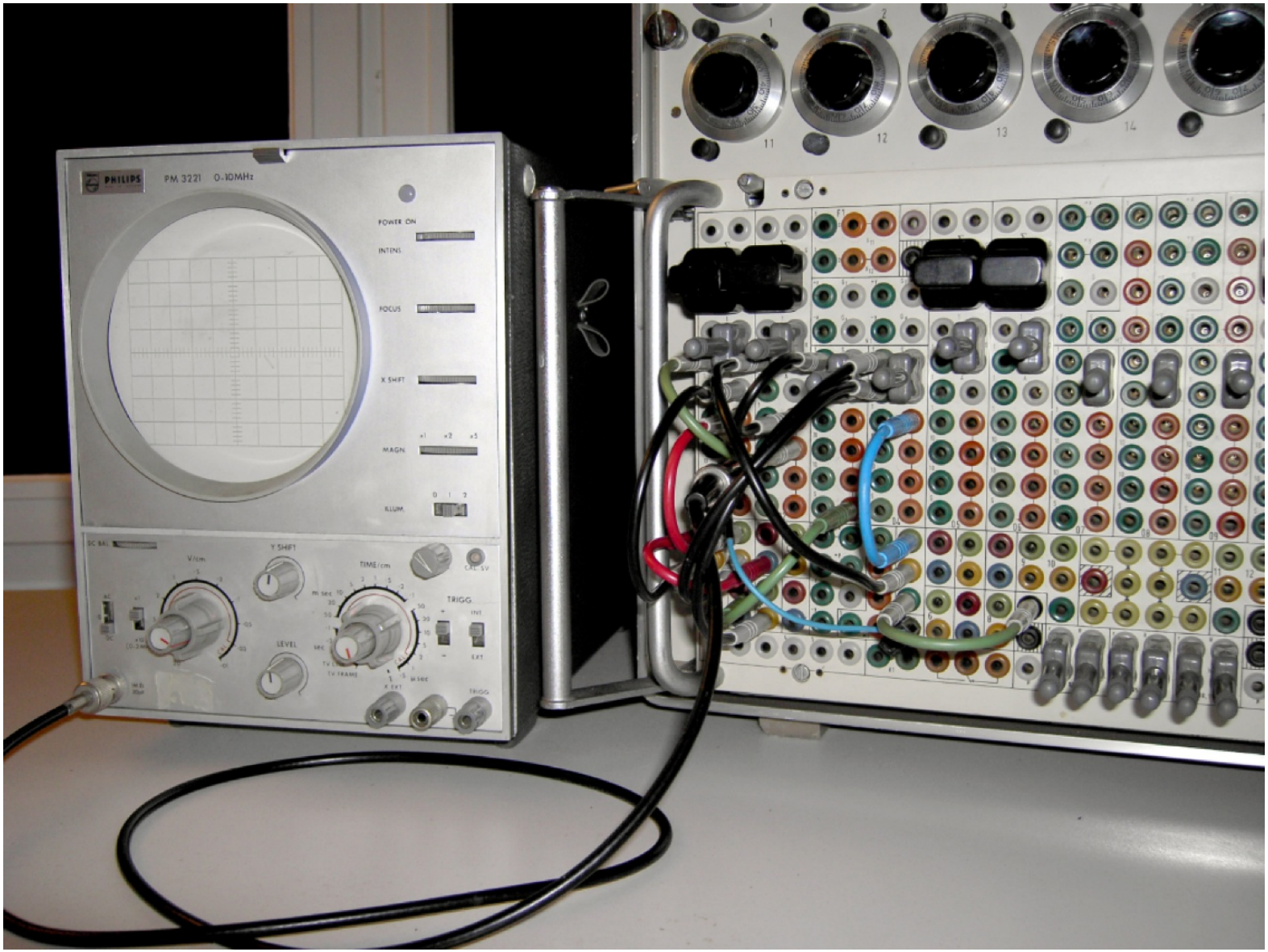
Generating $-(d\dot{y} + sy)$:



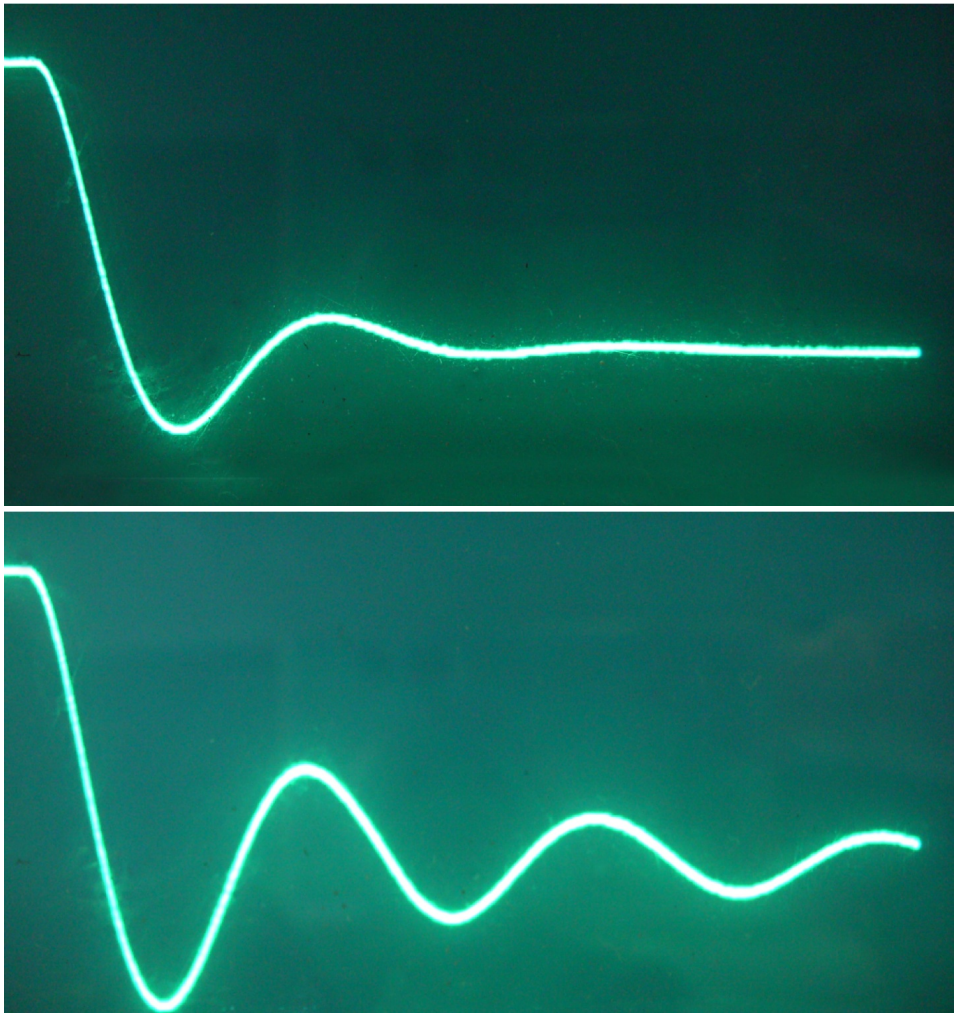
Final setup



Implementation



Typical results with different damping coefficients:



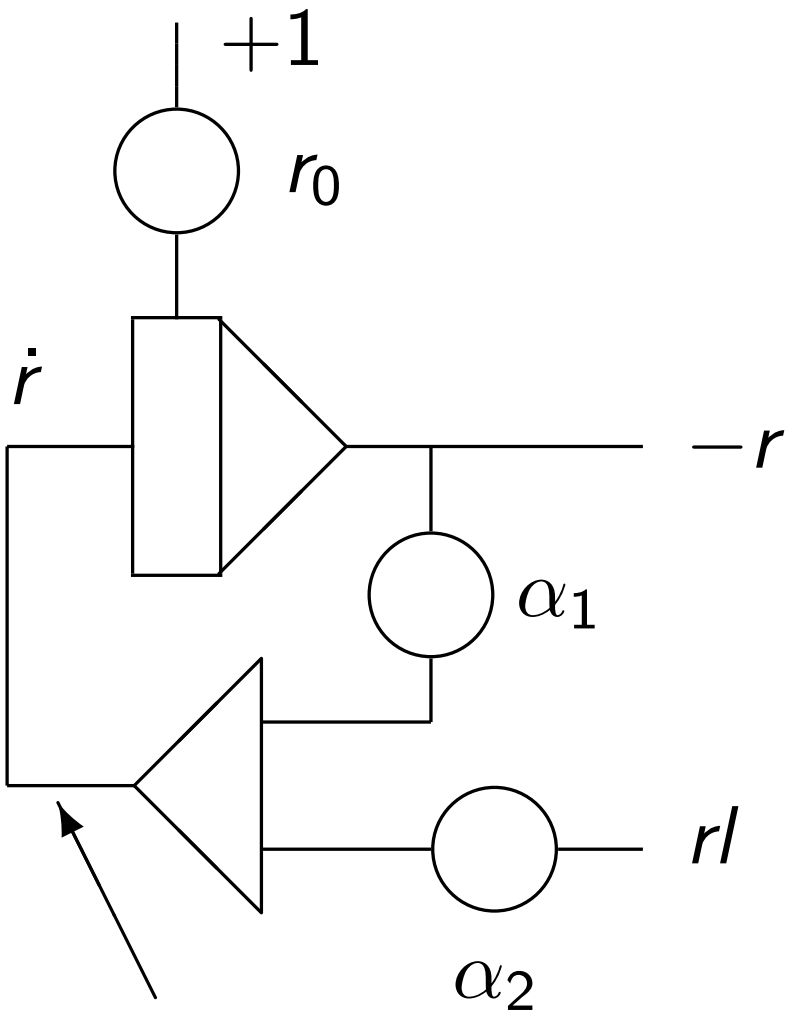
Predator-prey model

This example is a bit more complex as it involves two coupled differential equations, the LOTKA-VOLTERRA equations describing a simple eco system consisting of hares and lynxes:

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

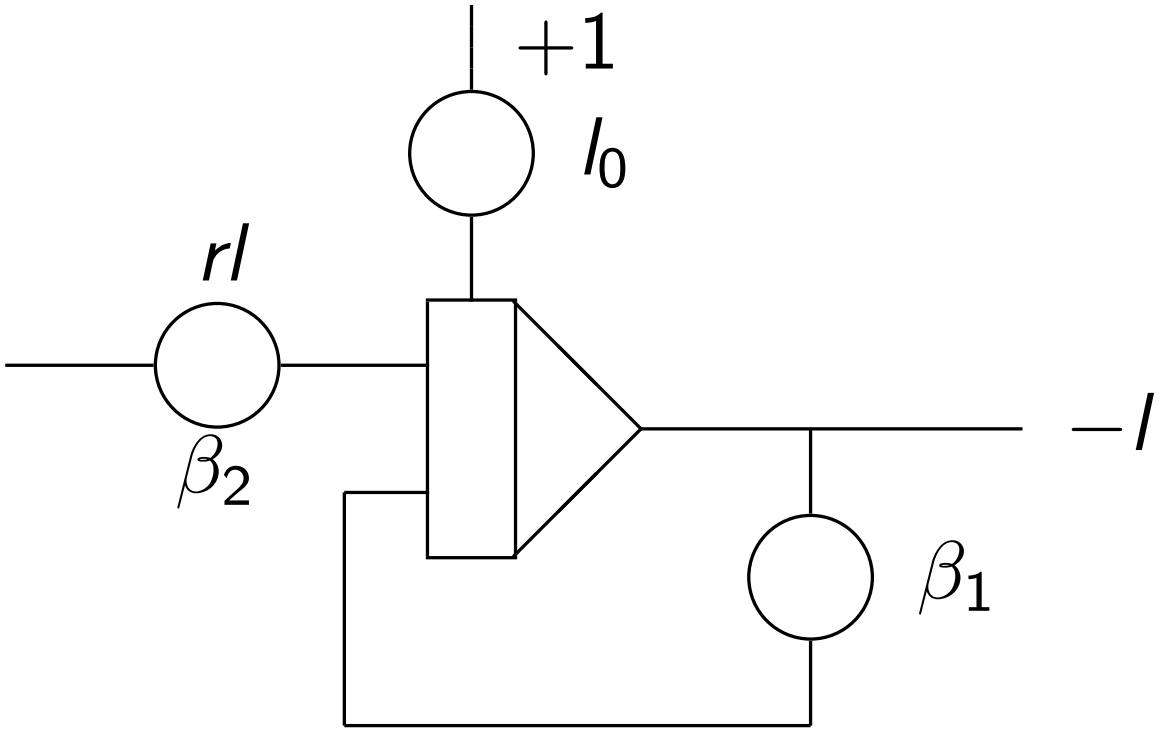
α_1 is the birth rate of hares, β_1 is the natural death rate of the lynxes, α_2 is the rate of hares killed by lynxes, and β_2 the increase of the lynx population due to killed hares.

Computing $-r$

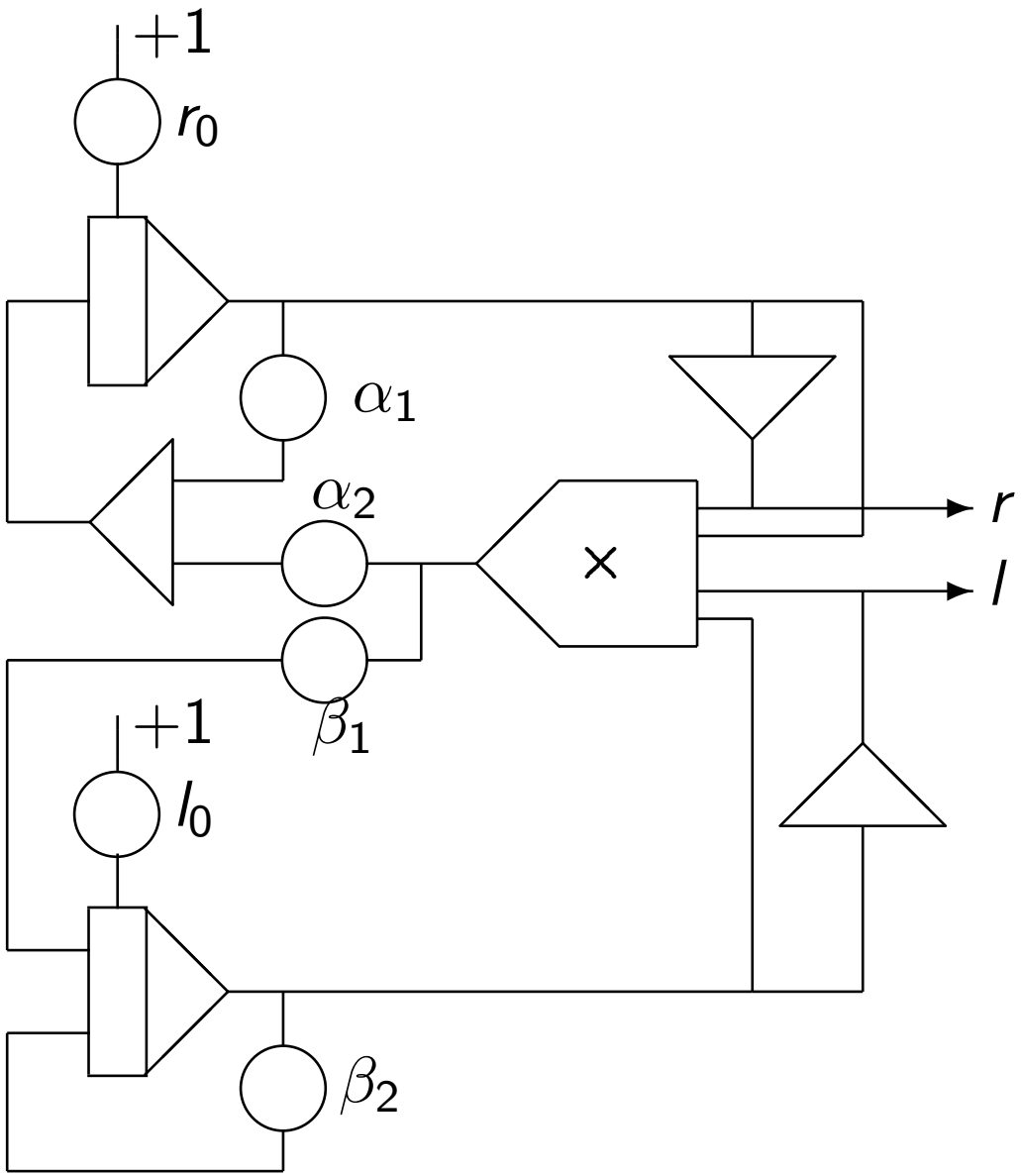


$$-\left(-\alpha_1 r + \alpha_2 rl\right) = \alpha_1 r - \alpha_2 rl$$

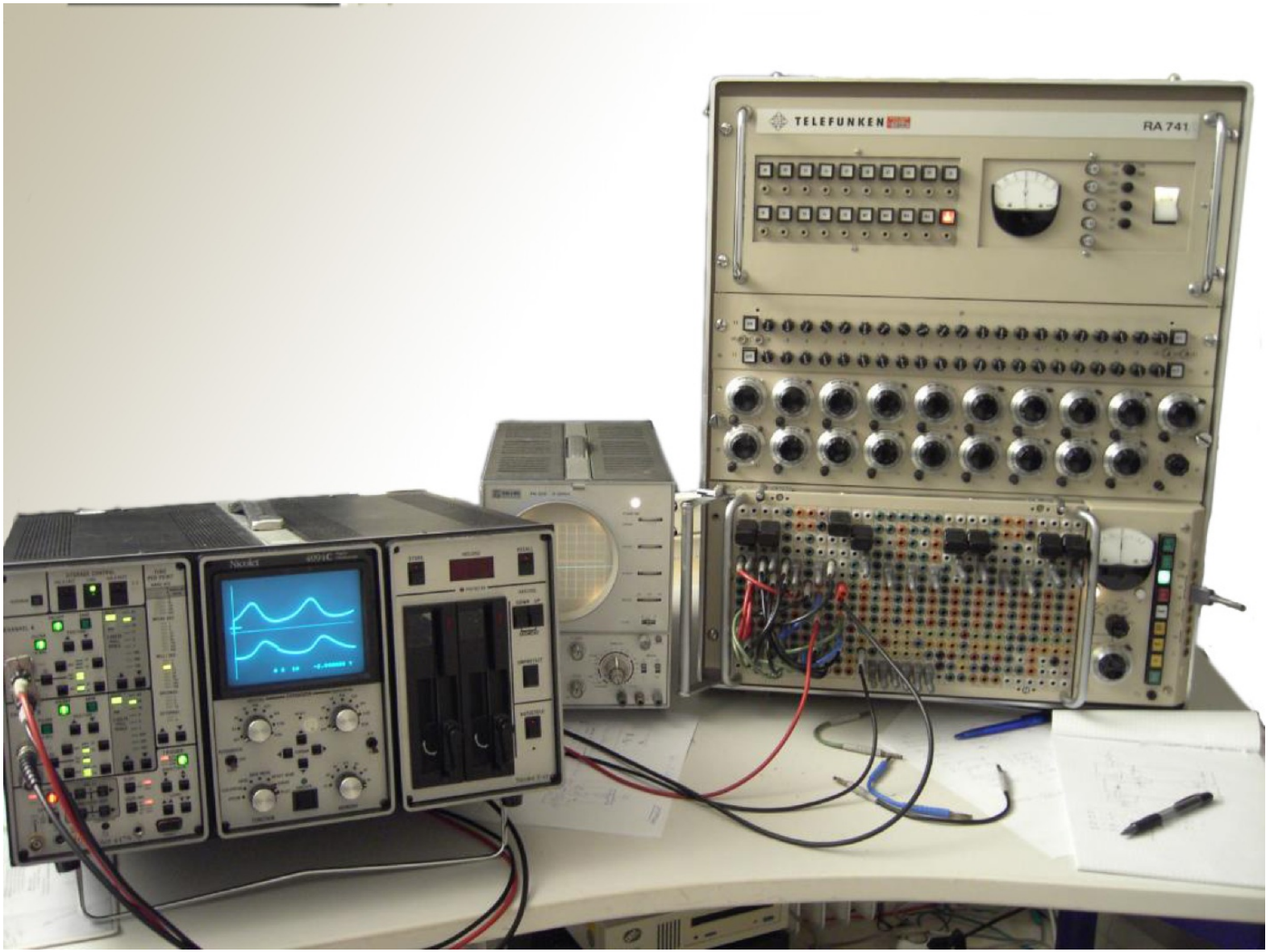
Computing $-I$



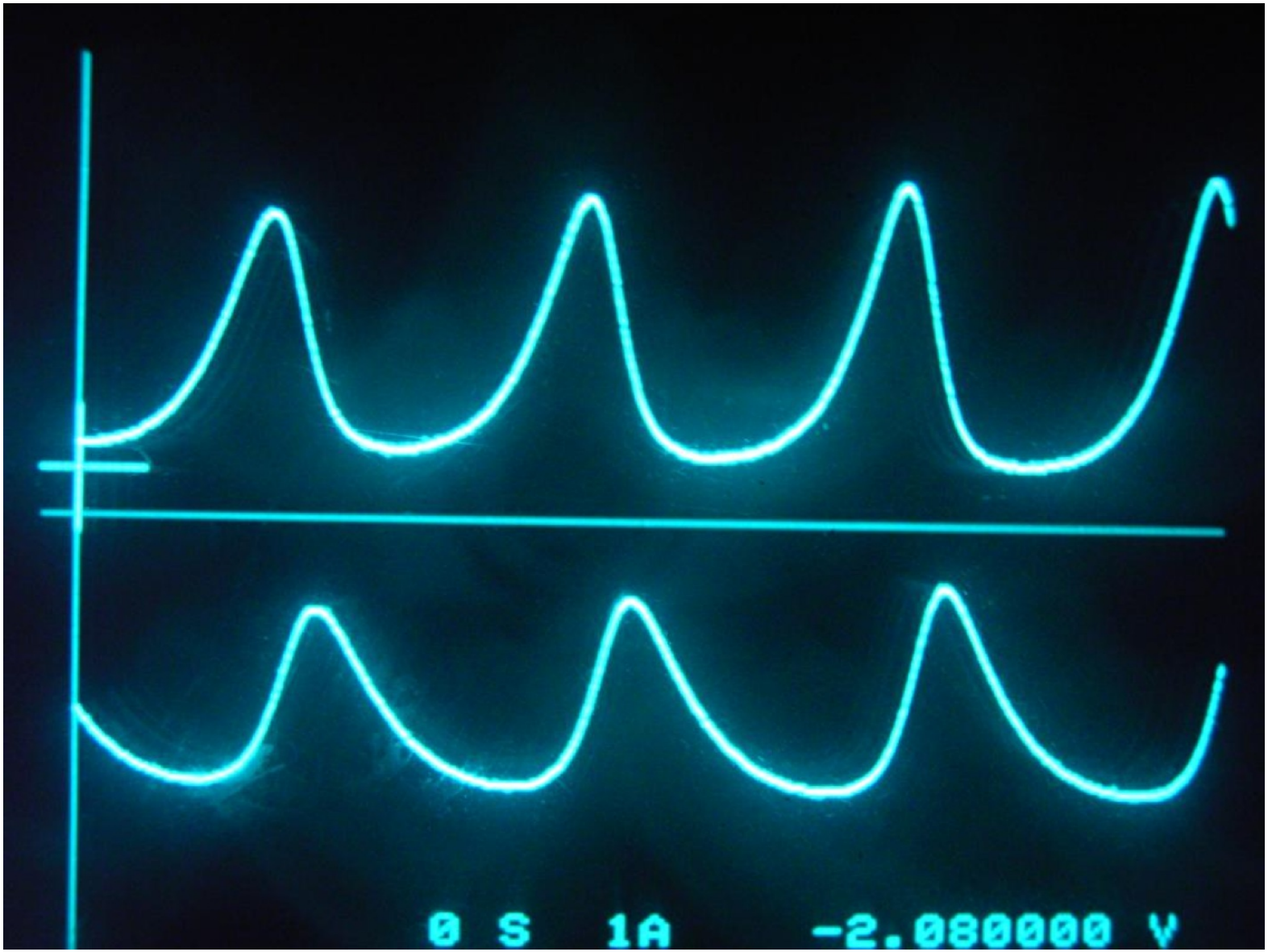
The overall setup



Implementation



Results



van der Pol oscillator

Van der Pol equation

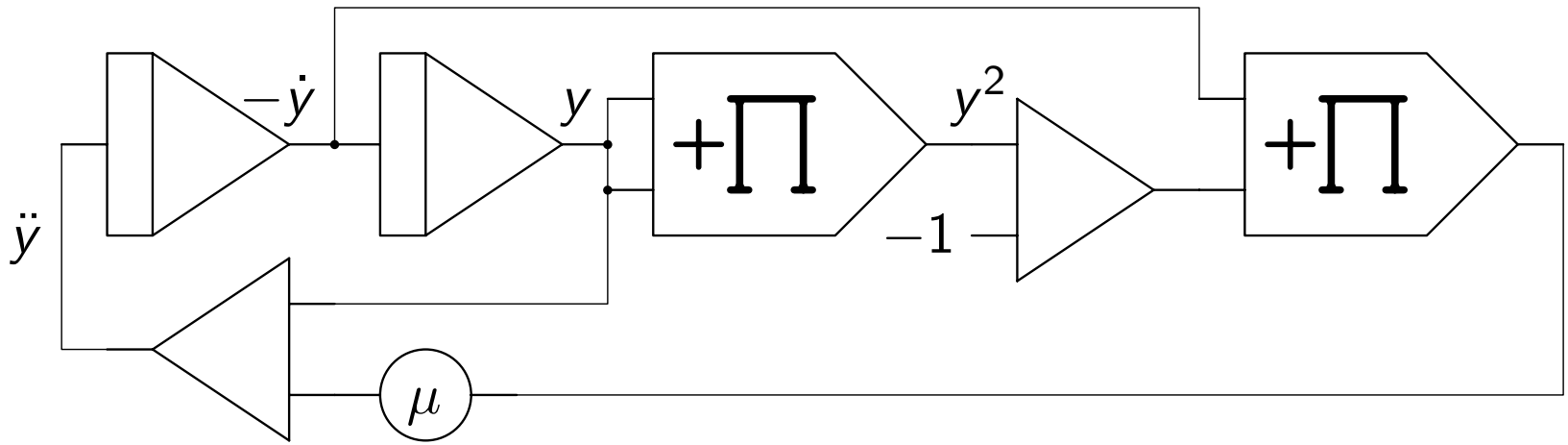
A little more complex example is the VAN DER POL equation:

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

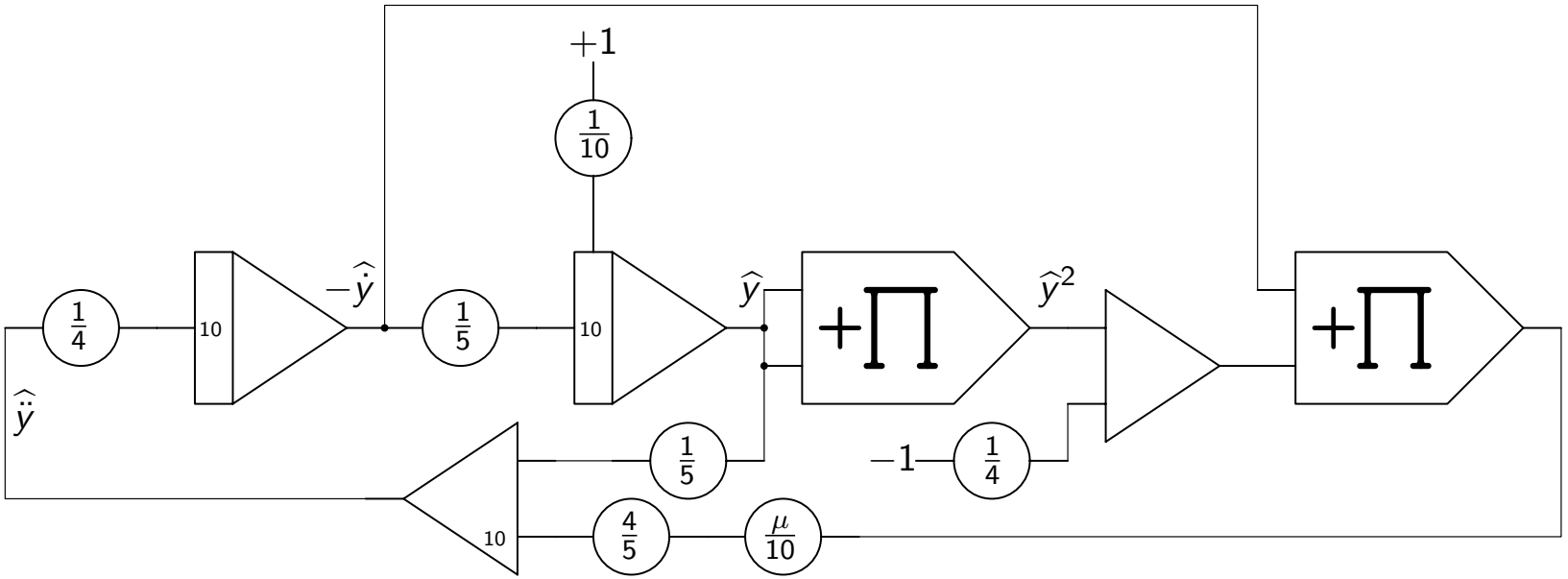
which can be rearranged, yielding

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

The unscaled analog computer program looks like this:

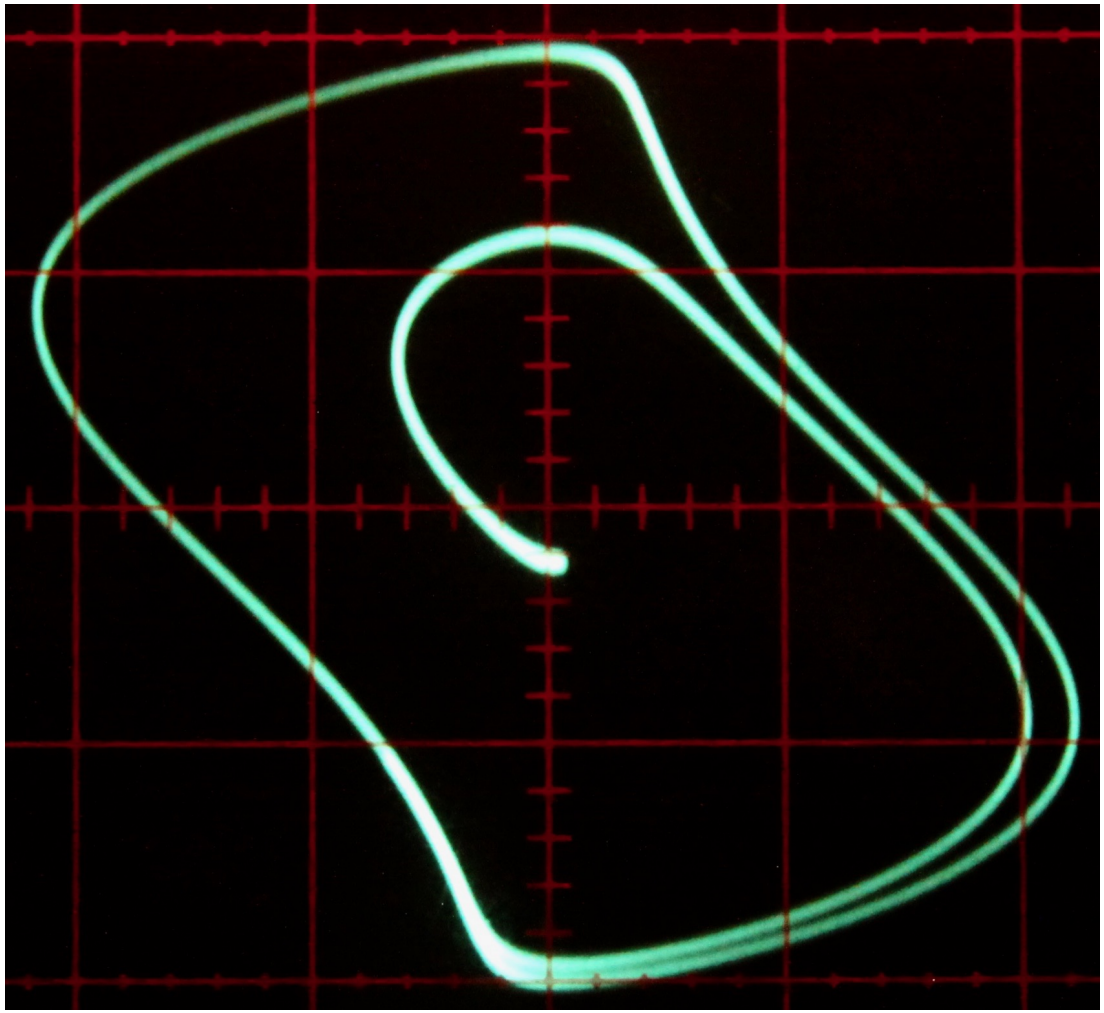


This problem must be scaled so that no variable exceeds the interval $[-1, 1]$ yielding the following program:



A typical phase space plot generated with this setup looks like this:

Van der Pol equation



PDEs

PDEs are challenging since time is the only free variable of integration ...

An analog computer setup for the two-dimensional heat-equation starts with

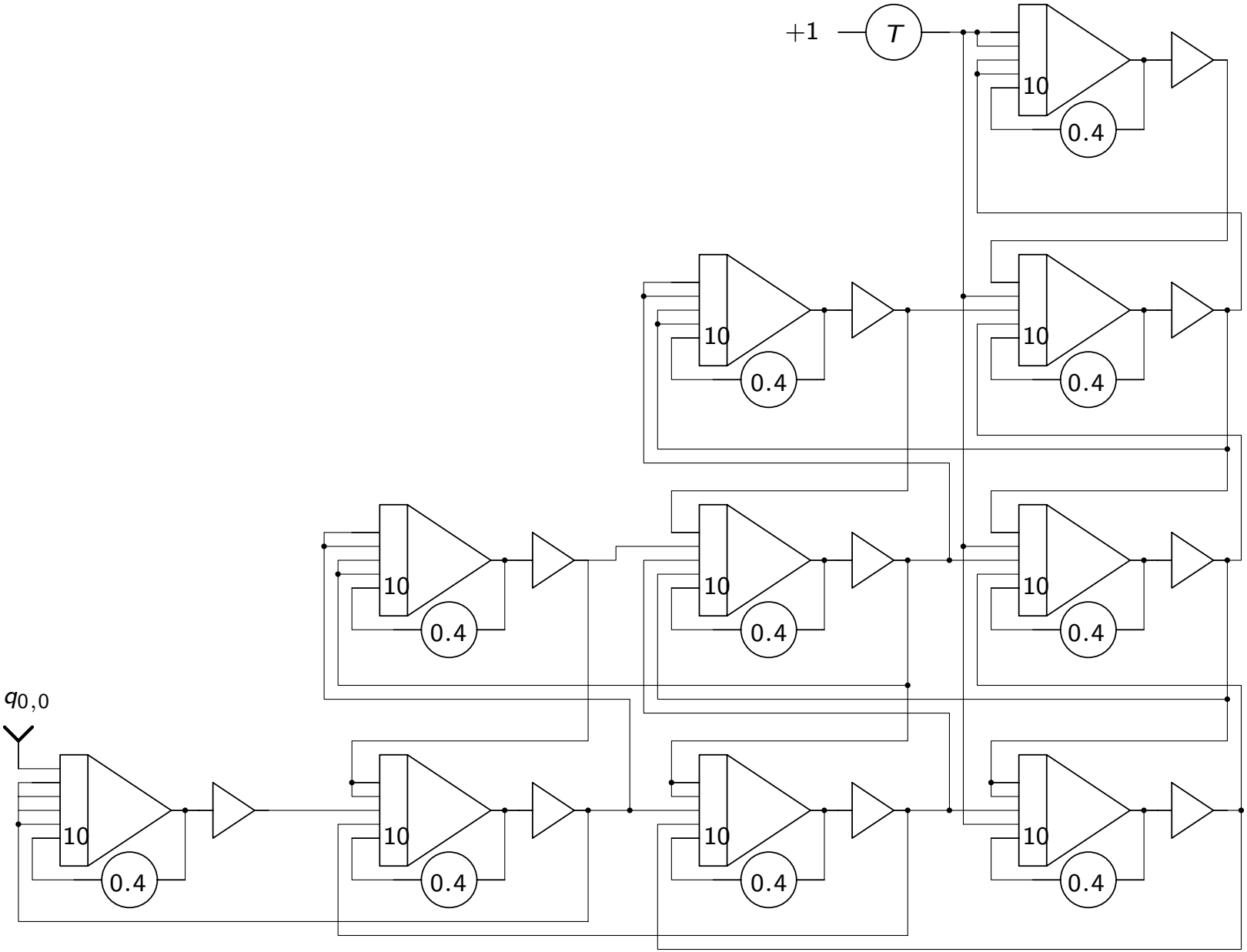
$$\dot{u} = \alpha \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right).$$

An obvious aproach is to replace the right-hand side of the above equation by a difference term:

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

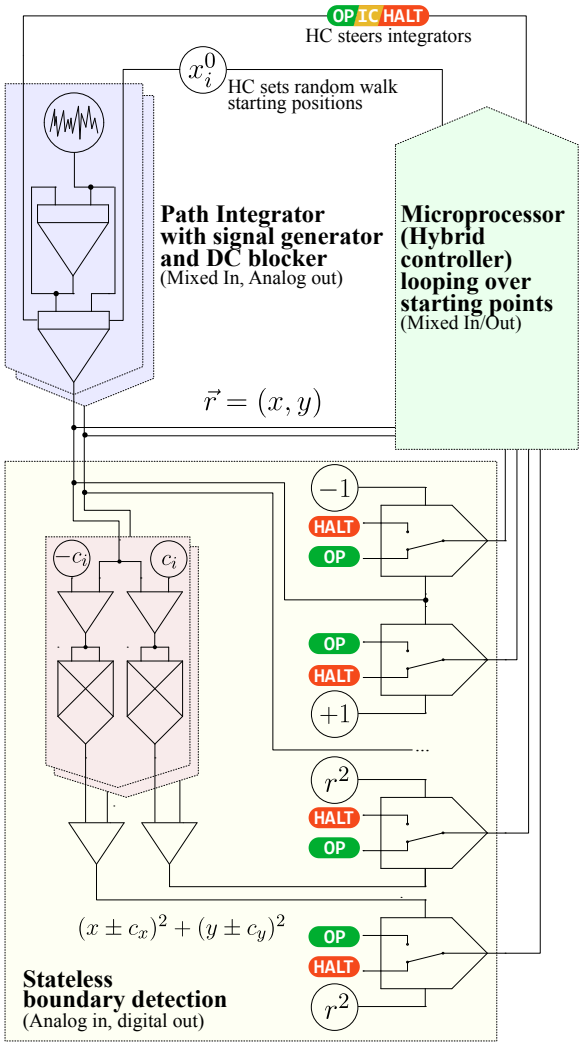
This can now be treated as shown before. In the following one octant of a square sheet is modelled (taking a symmetry argument into account):

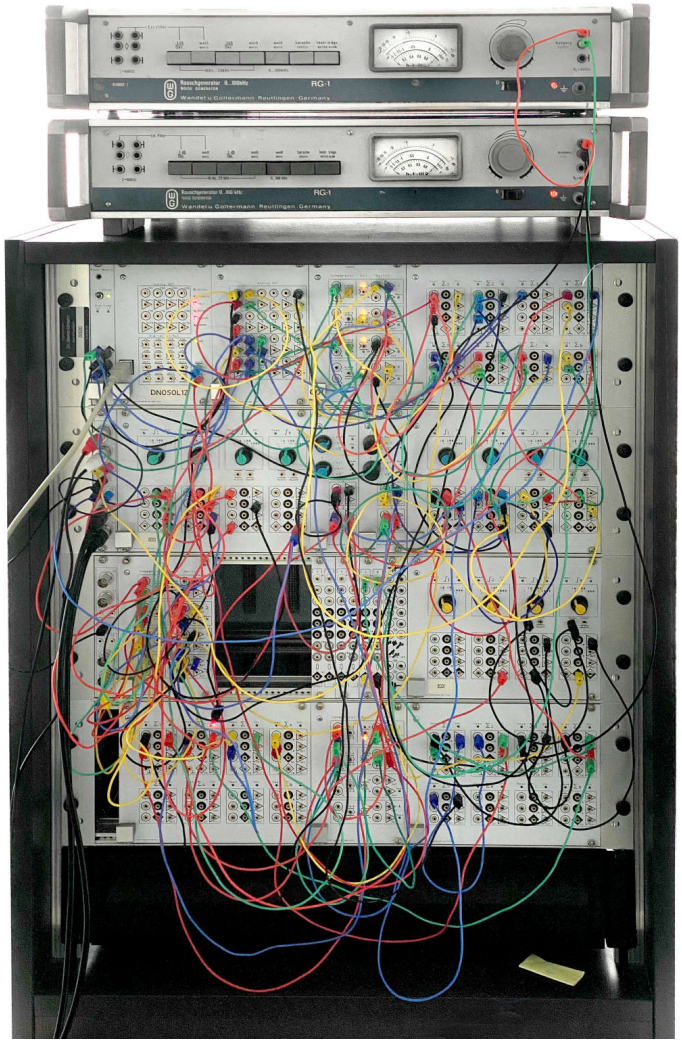
The heat equation



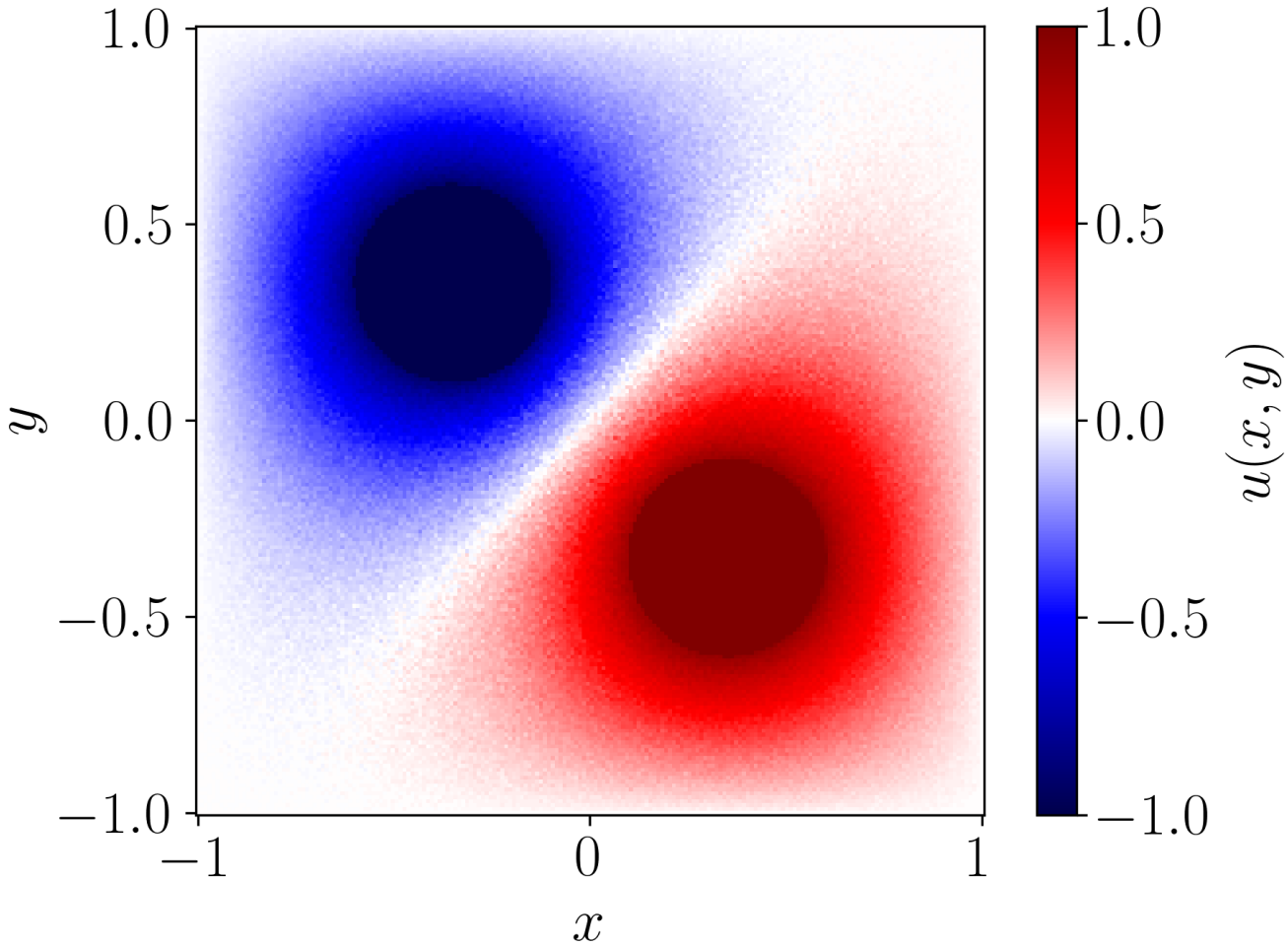
Feynman-Kac

Another, really interesting approach to solving PDEs is due to Feynman and Kac using a Monte-Carlo approach:



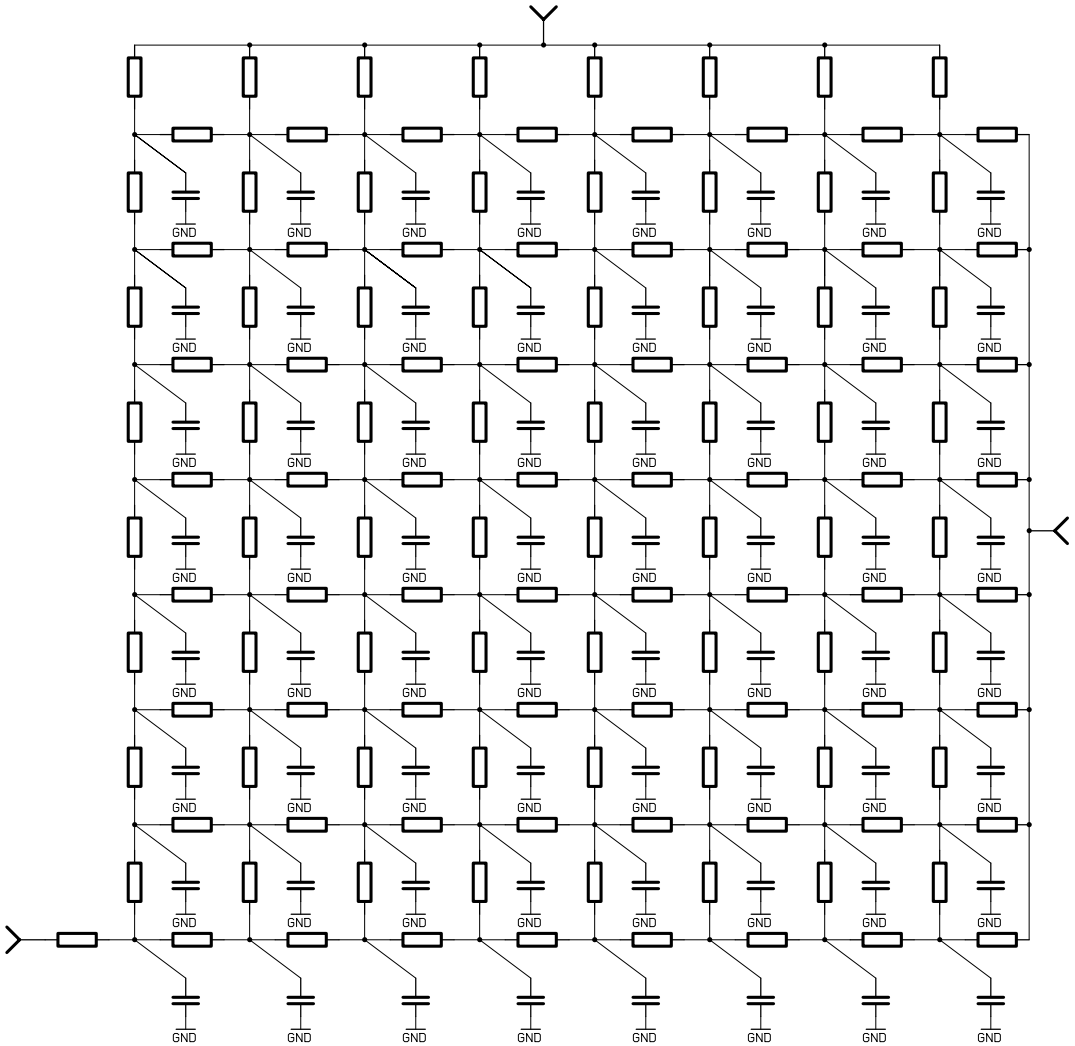


The charming characteristic of this approach is that no grid or discretisation of space coordinates in required. A typical result looks like this (in good accordance with numerical results):

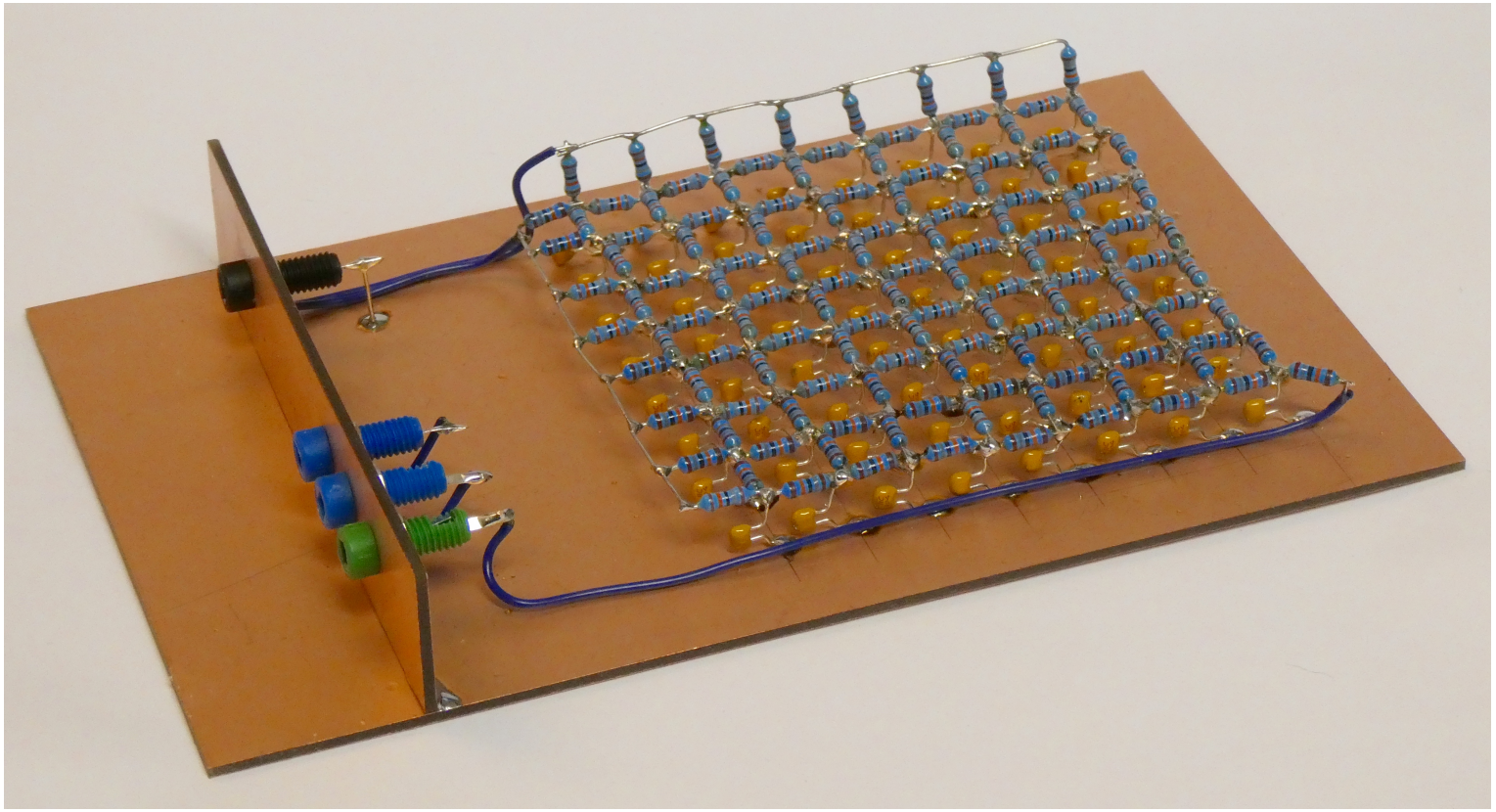


The heat equation revisited

Something as simple as the heat equation can also be solved by a passive model:

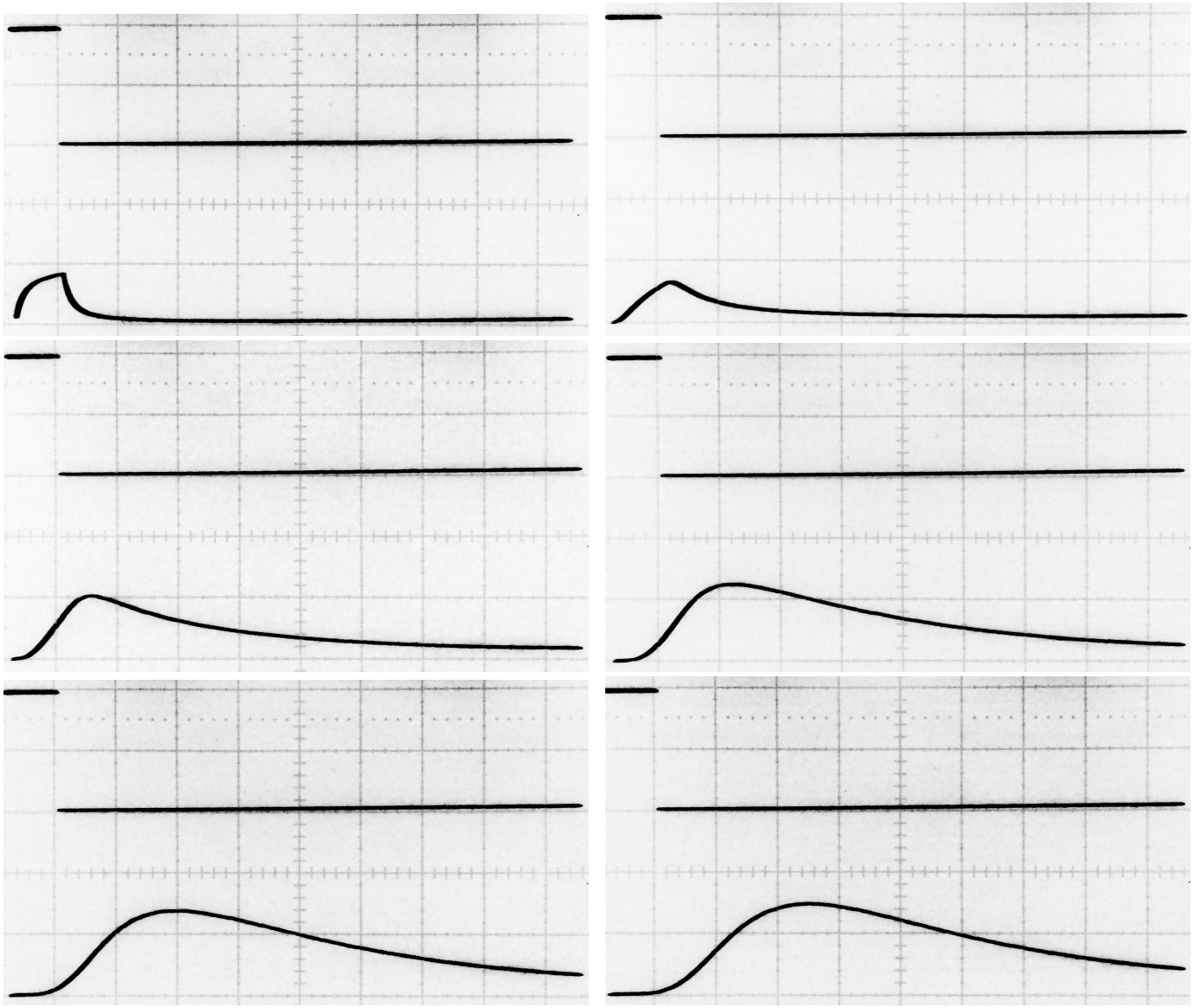


A passive analogue



A passive analogue

Response along the diagonal elements to an input pulse at $u_{0,0}$:



A passive analogue

Steady state solution:

