Analog Computing for the 21st Century

Bernd Ulmann

***Abstract*—Many people think of analog computing as a historic dead-end in computing. In fact, nothing could be further from the truth as analog computing – together with quantum computing – has the potential to bring computing to new levels with respect to raw computational power and energy efficiency. The following paper explains the limits of digital computers, gives a quick introduction to analog computing in general, and shows a number of recent developments that will change the way we think about computers in the next few years.**

***Index Terms*—analog computing, unconventional computing**

I. Introduction

After many decades of extremely successful application of stored-program digital computers (just called *digital computers* here) two alternative computing paradigms are shifting into the limelight: Analog computing and quantum computing. Both hold much promise for the future. The reasons for this forthcoming paradigm shift away from digital computers, the basic ideas of analog computing, application areas, etc. are described here.

It should be noted here that *analog computing*, despite its long history (much longer than that of digital computers) is a very active and promising area of research. One should not think of large classic analog computers in a museum but instead of modern integrated circuits and high performance computing. The 21st century needs analog computers on integrated circuits as co-processors employing the current power efficient CMOS technology, so we are talking about cutting-edge technology, not some arkane relics of the past.

II. Why we should care about analog computing

Despite their incredible success and versatility, digital computers are about to hit fundamental physical limits and face a variety of other challenges thus shifting the focus of current research to different computational paradigms.

The first problem to mention is the high power consumption of our digital infrastructure, a large part of which is caused by the very digital computers at its heart. The overall ICT (*Information and Computing Technology*) sector is estimated to have consumed between 4% and 6% of the global electrical energy in 2020 with data centers accounting for about one sixth of that total number [1]. In particular the demand for CPU power for training deep artificial neural networks is growing at an ever increasing pace, thus further advancing the energy demands of our current computing infrastructure. Apart from the obvious implications for future systems and data centers with respect to energy consumption, this creates an additional problem – that of heat removal. Modern top performing CPUs such as members of AMD's *Ryzen Threadripper* family have TDP (*Thermal Design Power*) values of up to 250 Watt. Cooling large scale systems based on such chips is no easy task.

Interestingly, clock frequencies of digital computers have not increased much at all during the last two decades which is largely due to the fact that the overall power consumption of digital circuits increases superlinearly with clock frequency. Admittedly the energy efficiency of these systems has risen several times due to advances in computer architecture and a trend towards many-thread-architectures coupled with more and more specialised subsystems but these speedups are neither easy to implement nor applicable to all applications.

Generally, parallelism is rather hard to achieve in digital computers as observed and described by Gene Myron Amdahl as early as 1967. Even if Amdahl's *law*, derived in this seminal publication, is an oversimplification and does not take the structure of modern processors with their intricate caching systems, etc., into account, it turns out to be a rather good tool for estimating the maximum speedup achievable by parallelising a task [2]. Admittedly, there is a variety of tasks which benefit rather directly from parallel processing as described by Gustafson's *law* [3] but this is not a general rule. To get near the theoretical peak-performance of a given system with a real-world application is not a simple task, often leaving large parts of a CPU sitting idle.

Current technology nodes of 2 nm are about to hit fundamental boundaries of integration densities, and of the many billions of transistor functions in a modern CPU, typically only a small part is actually performing computations while the vast majority implements infrastructure such as high-speed caches, intricate control circuits ranging from out-of-order-execution to complex pipeline control, register renaming, a plethora of uncore functions, bus and memory interfaces, and many more. At the very heart of these systems are

Bernd Ulmann, ulmann@anabrid.com, anabrid GmbH, c/o DLR Innovationszentrum, Wilhelm-Runge-Straße 10, 89081 Ulm, Germany.

comparatively few ALUs (*arithmetic logic unit*s) performing actual computations, which does not make really good use of the available silicon real estate.

A comparison of two state-of-the-art supercomputers might be interesting: Today's fastest digital supercomputer, *Frontier*, a HPE Cray EX system located at the Oak Ridge National Laboratory, achieves a staggering $1,194$ PFLOPS ($= 1.194 \cdot 10^{18}$ floating point operations per second, measured with the LINPACK benchmark) using $8,699,904$ cores and consuming about 22.7 MW of electrical power. This amounts to 52 GFLOPS per Watt which is really impressive but next to nothing compared with the contender, a human brain. The raw computational power of a brain is estimated to be in the ball park of 38 PFLOPS at an overall power consumption of about 20 Watt. This amounts to about 1.9 PFLOPS/W – many decades better than our current digital supercomputers.

How is that possible? What does nature do so differently from our classic approach to computation to achieve such an extraordinary high energy efficiency? It is mainly the fact that the brain does not execute an algorithm, i. e., there is no sequence of instructions to be executed in a step-by-step fashion. Instead a brain consist (vastly simplified) of billions of rather simple computing elements, neurons, which are interconnected in a suitable fashion to implement the many feats we are capable of. Thus, the *program* underlying a brain is not an algorithm but a directed graph describing the interconnection of neurons with their individual input weights etc. There is no central memory, no cache memory, no intricate control system, just a multitude of small computing elements all working in full parallelism.

From that perspective, a biological brain quite closely resembles an *analog computer* instead of a stored-program digital computer, as the interconnection of its computing elements is the actual program.

III. Analog computing

Now, what is an analog computer? In a nutshell, an analog computer consists of a number of computing elements, each capable of performing a basic mathematical operation such as summation, integration (this is a truly magic element), multiplication, etc., which are interconnected in a suitable way to form a model, an *analogue*, for a given problem to be solved. The term *analog computer* refers to the model building charactersitic and not to a certain representation of values.

Variables are typically represented by voltages or currents.[1] This representation simplifies the construction of an analog computer substantially since only a single wire (or a pair of wires) is required to connect two computing elements. Nothing is more true in computer science than the saying "there is no such thing like a free lunch" which also holds true in the case of analog computers where this characteristic comes at a cost: An analog computer typically offers only limited precision with typical resolutions being of the order of 10^{-3} to 10^{-4}. Analog computers capable of a precision of 10^{-4} are often called *precision analog computers*.

As negative as this sounds it isn't much of a problem for many if not most applications. Engineering problems typically do not require more than a few decimal places for a useful solution, biological neural networks only feature very limited resolution for their synaptic weights which thus also holds true for artificial neural networks, etc. In cases where higher precision is required, results generated by an analog computer could be used as starting points for numerical algorithms executed on an attached digital computer which can then enhance these results to the required degree of precision.

Using voltages or currents to represent values not only removes much complexity from a computer but also adds significantly to the energy efficiency of the system since there is no need to flip zillions of signal lines between 0 and 1 billions of times per second, each time charging or discharging tiny parasitic capacitors. Furthermore this representation facilitates interfacing to the surrounding world making analog computers ideal for signal pre- and postprocessing, etc.

Analog computers are ideally suited for problems that can be described as (systems of coupled) differential equations (most problems relevant in science and engineering fall into this category). There are also suitable approaches to tackle partial differential equations on analog computers. They can even implement oscillator based *Ising machines* and thus solve problems which are normally attributed to adiabatic quantum computers [4][5].

The following simple problem shows the main difference between programming a classic digital computer and an analog computer. Here, $x = a(b + c)$ is to be computed. A classic digital computer requires six instructions to accomplish this as shown in figure 1. The two arithmetic operations at the core of the task are surrounded by four instructions to load data from memory and store the result back to memory. Of course, this ratio gets better with increasing problem complexity and with clever register allocation schemes, but it illustrates the nature of a digital computer quite well.

Solving the same problem on an analog computer requires two computing elements, one summer and one multiplier, and a few connection between these elements. The input values a, b, and c represented by voltages or currents are connected to these ele-

[1] It should be noted that there are even "digital analog computers" called *DDAs* (*Digital Differential Analyser*s) which represent values in a binary fashion but also rely on a set of computing elements which are interconnected with each other to implement a program.

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

Figure 1. Computing $x = a(b + c)$ on a digital computer

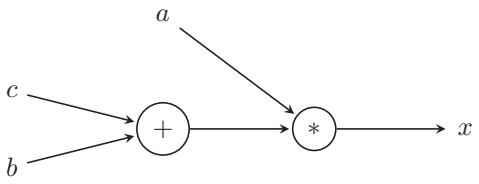

Figure 2. Analog computer setup for solving $x = a(b + c)$

ments, while the output of the summer is connected to one input of the multiplier. There is no algorithm in the classic sense of the word, instead the program resembles a directed graph describing how computing elements must be connected with each other to solve a given problem. Since there is no algorithm and no central memory storing instructions and values, all computing elements can work in full parallelism with no need for explicit synchronisation etc.

This leads to an interesting basic difference between digital and analog computers: In a digital computer it is always possible to trade problem complexity against solution time. The more complex a problem gets the longer it will take to solve it on a digital computer (always assuming there is enough memory to hold all instructions and variables for the problem). This often convenient tradeoff is not possible with an analog computer, which is a certain drawback. If the implementation of a problem requires more computing elements than a given analog computer contains, it cannot be solved (at least not directly) on this particular machine. On the other hand, the solution time for a problem on an analog computer does not increase with problem size, provided that there are enough computing elements available to implement the program.

Of course, there are some drawbacks of analog computers: In addition to the rather limited precision of analog computing elements, values have to be within the interval $[-1, 1]$.[2] Additionally, the generation of arbitrary functions as obtained by experiments or the like is also quite challenging for an analog computer especially when functions $f(x_1, \ldots, x_n)$ of more then one argument are required. Nowadays this can be easily overcome by using lookup tables stored in some memory with attached analog-digital- and digital-analog-converters (*ADC*s and *DAC*s). Regarding integration, being a basic operation of an analog computer, only time is available as the free variable of integration; this constraint requires the use of advanced techniques to tackle PDEs (*partial differential equations*).

Modern analog computers typically will be part of a *hybrid computer*, i. e., a combination of a digital computer with an analog computer as a specialised, closely attached co-processor. This co-processor excels at the high-speed, energy efficient solution of (systems of) differential equations while the digital computer allows for storage, decision making, function generation, parametrisation of the analog computer, etc.

IV. Programming

As arcane as analog computer programming may look to the uninitiated, it is much more straight forward than the algorithmic approach we were all taught and cherish. A simple example may show the basic approach which relies on an idea of Lord Kelvin who developed the Kelvin *feedback technique* [6]. in 1876 after the invention of a practical mechanical integrator by his brother [7, pp. 22 ff.].

This technique transforms a mathematical problem description into an analog computer program, i. e., a directed graph connecting suitable computing elements, in a series of five steps:

1) Organise the equation to isolate the highest derivative on the left hand side.
2) Assuming this derivative is known, all lower derivatives can be generated by repeated integrations.
3) Using summers, multipliers, and other computing elements, all terms on the right hand side of the equation obtained in the first step are derived.
4) These terms are now tied together to form the right hand side of the equation. Since this must be equal to the highest derivative this signal is then fed back into the circuit as the highest derivative which was assumed to be known in the second step.
5) This program typically needs scaling to ensure that no value lies outside the interval $[-1, 1]$. In addition to this, a good scaling process also ensures that ideally every variable will make best use of this interval, thus increasing the precision of the computation.

Suppose the 2nd-order differential equation $\ddot{y}+y = 0$ is to be solved with an analog computer. It first is solved for its highest derivative yielding $\ddot{y} = -y$. Starting from $\ddot{y}$ the lower derivatives $-\dot{y}$ and y are derived by a chain of two integrators. (Due to the actual implementation of such integrators, they typically perform an implicit sign flip. Summers behave similarly.) Since the right-hand side of $\ddot{y} = -y$ is negative, an

[2] It is best to always think in terms of this abstract interval instead of the actual minimum and maximum voltages or currents representing these values in order to simplify programming and scaling. Early vacuum tube based analog computers used voltages between ± 100 V to represent values. These voltages later dropped to ± 10 V in transistorised machines, while modern implementations use ± 1 V and even lower voltages at much increased bandwidths.

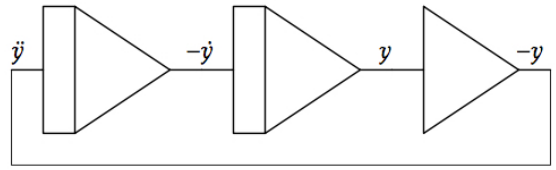

Figure 3. Basic analog computer setup for solving $\ddot{y} + y = 0$

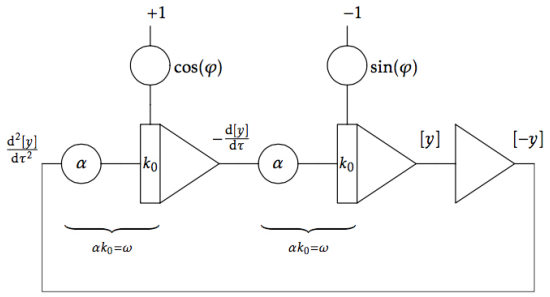

Figure 4. Analog computer setup for solving $\ddot{y} + \omega^2 y = 0$ with proper initial conditions

additional change of sign is required which is done by a summer with only one input, acting as an inverter. The output of this summer is then fed into the first integrator, thus completing a feedback loop as shown in figure 3. The triangular symbol on the right denotes a summer while the two symbols to its left represent integrators.

This program is missing one important detail – the initial conditions of the integrators, specifying which solution to obtain. A complete program for the solution of the equation $\ddot{y} + \omega^2 y = 0$ with initial conditions $y(0) = \sin(\varphi)$ and $\dot{y}(0) = \cos(\varphi)$ is shown in figure 4. The circles denote coefficients, k_0 denotes the *time-scale factor*[3] of the integrator with $\alpha k_0 = \omega$.

An analog computer typically features (at least) three modes of operation:

- *Initial Condition* mode (*IC*): The integrators are set to their respective initial condition values. This step must precede the following mode.
- *Operate* (*OP*): The integrators integrate with respect to time – the computer performs its actual computations.
- *Halt* mode (*HALT*): All integrators are halted, so that all variables within the program remain constant. This is typically used to read values when using slow ADCs. This mode may be followed by OP or IC.

Problems of a more realistic complexity can be transformed into analog computer programs employing the same rationale but typically need some (extensive) attention with respect to scaling. A recent and thorough introduction to analog and hybrid computer programming can be found in [8].

[3]This time-scale factor k_0 describes the speed at which an integrator runs. In the case of $k_0 = 1$ the output of an integrator will reach -1 with a constant input of $+1$ after one second. If $k_0 = 10^3$ this value will be reached after one millisecond, etc.

V. Problems of classic analog computers

While analog computers were the systems of choice when it came to the treatment of problems requiring real-time solutions of highly complex dynamic systems, they were basically replaced by digital computer in the early 1980s even though their digital rivals could not compete their computational power for another one or two decades to come.

The main reasons for this were that digital computer became quickly cheaper while analog computers continued to be rather expensive systems due to the required high precision components, etc. In addition to this, programming a classic analog computer required manually plugging hundreds and sometimes thousands of patch cables on a patch panel to implement a certain analog computer program, a time consuming and error prone process. Although such patch panels with their intricate maze of wires could be quickly replaced on medium to large scale analog computers, switching from one program to another still was a time consuming affair. Accordingly, most analog computers were used to treat a single problem for a prolonged time during which no other program could be run concurrently. In contrast to this digital computers offered time sharing access since the 1960s allowing many users to share the (limited) computing power of these machines. This characteristic alone often turned purchase decisions away from an analog computer.

Figure 5 shows a classic large scale analog computer installation at the German aerospace company Boelkow in 1960. This particular installation implemented a flight simulator for a vertical-takeoff-and-landing jet fighter. It shows clearly the fact that the analog computer's size must match the problem size. At its heart are three large Telefunken RA800 computers as well as a number of smaller scale table top analog computers, all being interconnected to form a single very large scale machine. It is quite impressive that it was possible to implement a realistic flight simulator controlling a hydraulic hexapod with a cockpit mounted on its top in 1960 using this approach, something digital computers needed many years to actually compete with.

At the same time this picture clearly shows how cumbersome programming these systems was back then. The patch panels are buried under heaps of patch cables connecting hundreds of computing elements with each other. Changes to such a complicated setup often required hours or even days and sometimes weeks of preparation and actual implementation. Changing parameters of a program required manually changing hundreds of precision 10-turn potentiometers, 300 of which are visible on the three large systems on the left in the figure with many more on the smaller machines on the right. A detailed history of analog computers can be found in [7].

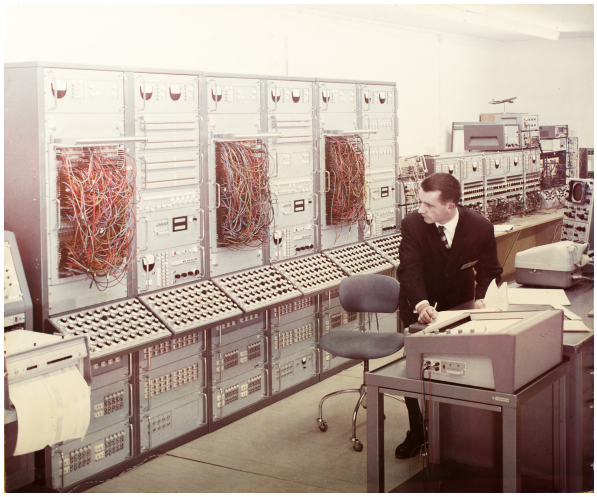

Figure 5. Classic large scale analog computer installation

VI. Areas of application

There is a wide variety of applications for modern analog computers. The most prominent of which are the implementation of artificial neural networks by means of analog electronic implementations of neurons, and high performance computing with applications such as molecular dynamics, computational fluid dynamics, Monte-Carlo simulations as often used in financial mathematics, optimisation tasks, and many more. These fields can benefit most from the high computational power of analog computers with their high energy efficiency in second place.

Another large application area where the high energy efficiency is of utmost importance are medical applications such as cardiac and brain pacemakers, blood sugar sensing and insulin pump control, in vivo instrumentation for cancer therapy, and many more. In some specialised applications it might be possible to power the implant by energy harvesting thus alleviating the need for bulky and inconvenient energy storage solutions, much to the benefit of the patient.

Since an analog computer does not execute an algorithm and has no traditional memory containing instructions or the like it is not prone to classic attack vectors, making it a worthwhile option for process control in critical environments and the like.

Used for signal processing, analog computers can perform trigger word detection for smart devices, vibration analysis, and other tasks for predictive and preventive maintenance, etc.

VII. Modern technology

Today, the main drawbacks of these classic systems can be overcome using modern CMOS technology. The patch panel is a thing of the past and belongs in a museum. Modern analog computers will be fully reconfigurable using large switch matrices (*crossbar switches*) under the control of an attached digital computer. Also the manual potentiometers have long since been replaced by digital potentiometers or multiplying DACs allowing the digital computer to change parameters in microseconds.

In addition to this, modern implementations of analog computers will offer much higher bandwidth computing elements, thereby allowing for much shorter solution times. Combined with automatic reconfiguration and reparametrisation this will make it possible to use an analog computer in a time-sharing like mode of operation.

One central problem that must be addressed in order to get widespread acceptance for analog and hybrid computing is the necessity of an abstract programming language, a *domain specific language* (*DSL*), that must hide most if not all of the underlying physical principles the analog computer relies on. The reason for this is the fact that programming analog computers requires a completely different mindset to writing an algorithm. Since the majority of programmers and IT professionals are trained to work in the algorithmic domain, the impedance mismatch between this approach and that of specifying the interconnection of individual computing elements must be minimised.

Furthermore, it is not that simple to tightly couple an analog computer with a digital system, especially since the analog computer offers solution times in the submicrosecond range. This requires very short interrupt latencies on the digital computer so that it can keep up with its analog co-processor without forcing it to idle most of the time waiting for some interrupt to be processed.

VIII. Modern developments

This section gives a quick overview over recent developments in the field of analog and hybrid computing including academic and commercial projects.

Terminating a long hiatus in the field of analog computing, Glenn E. R. Cowan developed and implemented a reconfigurable analog computer on a VLSI chip in 2005 [9]. Its development showed the basic feasibility of CMOS based analog computers. This was then followed by an enhanced VLSI analog computer chip developed and implemented by Ning Gou in 2016 [10]. Unfortunately neither of these academic projects resulted in commercial developments. The bandwidth of the computing elements was rather limited with sometimes large phase shifts caused by the interconnect matrices. Also the software support was quite limited.

An early commercial development was due to *anadigm*® who developed and market *FPAA*s (*Field Programmable Analog Arrays*) consisting of a number of *Configurable Analog Block*s (*CAB*s) and a number of input/output interfaces. These can be used to imple-

ment a variety of signal pre- and postprocessing tasks such as filters, general audio signal conditioning, etc.[4]

Aspinity has developed two major technologies: The *RAMP*™[5] *Technology Platform*, a programmable analog neuromorphic processor, and the *AnalogML*™*Core*, a machine learning co-processor. Typical applications are trigger word detection for smart devices, glass break detection, acoustic event detection, and vibration monitoring. Due to the high energy-efficiency these devices can be applied in always-on sensing applications.

Another startup working towards analog computing for artificial intelligence is *Mythic* (https://mythic.ai/). They pioneer a *compute-in-memory* approach where memory cells are implemented as variable resistors in the form of flash memory cells. A matrix of such cells can perform typical operations of linear algebra by employing KIRCHHOFF's law, being fed row wise by DACs with ADCs reading out the results of the implicit additions and multiplications performed by the resistor array.

IBM, too, has developed an analog AI accelerator, which was unveiled in 2022.[6] It uses *Resistive RAM* (*ReRAM*) made from *Phase Change Memory* (*PCM*) cells the state of which is switched between an amorphous and a crystalline state. This experimental chip implements $35 \cdot 10^6$ such PCM cells.

A very interesting development is the implementation of artificial neural networks in a true three-dimensional topology. Stacking of chips has been done for many years in the digital domain but is inherently limited in its extent due to the high power consumption and the problems of removing the excess heat from such a stack of chips. The very high energy-efficiency of analog computing approaches will make large scale three-dimensional topologies feasible.[7]

One might ask about the application of Memristors as they are often mentioned being "ideal" devices for implementing synaptic weights in analog neural networks, etc.[8] It may be doubted that Memristors based on filament conduction would be a good choice for implementing synaptic weights due to their typically limited number of state changes as well as the necessity for a device forming period prior to their actual use in a circuit. Approaches to Memristors not relying on filament conduction are relatively new and seem to have not yet matured into integration-ready devices on a large scale. However, using Memristors in analog artificial neural networks might be interesting as they might be able to implement self-learning systems based on assumptions such as "fire together, wire together".

Apart from these rather specialised and mostly AI centric analog computing applications and approaches, there is another startup, *anabrid* GmbH, based in Germany[9] pursuing general purpose analog computing with the ultimate goal of designing, implementing, and marketing a reconfigurable large-scale integrated analog processor. This device will act as a co-processor offloading compute intensive tasks from a classic digital processor but without being restricted to a narrow field of application, which is in contrast to the aforementioned developments.

Analog computing like quantum computing comes with the challenge that programming such machines is completely different to the classic algorithmic approach taught in schools and universities. Especially professional programmers have lots of difficulties addapting to a different computational paradigm like that of an analog computer. To bridge this gap at least two things are required: First, a cheap analog computer aimed at the educational market, at hobbyists, etc. Second, an abstract domain specific programming language which allows the seemless integration of analog co-processors into current digital computer systems. Ideally, a programmer should not have to think about the actual implementation of an analog computer or of intricacies such as scaling and the like.

Figure 6 shows *THE ANALOG THING*, an open hardware project[10] aimed at the educational market. This little analog computer contains enough computing elements for serious experiments in analog computing and can be easily interfaced to a digital computer, thus forming a hybrid computer.[11] It contains five integrators, four summers, four sign-inverters, two multipliers, two comparators, and eight manual coefficient potentiometers. Of this system more than 1000 have already been ordered showing a strong interest in analog computing in and for the 21ST century.

IX. CONCLUSION

It may be concluded that analog computing is making a comeback and will stay with us in the future. Analog computers, ranging from specialised devices to general purpose co-processors will substantially transform the way we compute in general. The following years will see huge leaps in analog computer implementations catching up with the developments in the digital domain during the last decades, eventually surpassing our classic computers for certain areas of application.

[4] A nice signal processing board based on such an FPAA was developed and is sold by NICOLAS STEVEN MILLER (https://zrna.org). These boards can be easily configured using a Python client.

[5] *Reconfigurable Analog Modular Processor*

[6] https://research.ibm.com/blog/the-hardware-behind-analog-ai

[7] See https://research.ibm.com/blog/vlsi-hardware-roadmap.

[8] ReRAM cells are typically not considered being Memristors due to their rather different behavior.

[9] The author is one of the founders of anabrid GmbH.

[10] See https://the-analog-thing.org and https://github.com/anabrid/the-analog-thing for schematics etc.

[11] See https://the-analog-thing.org/wiki/Hybrid_Computer

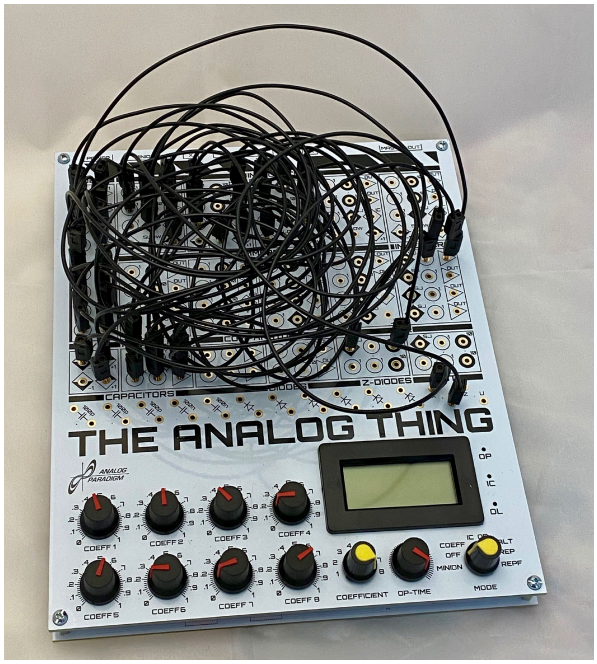

Figure 6. THE ANALOG THING

Bernd Ulmann Bernd Ulmann is co-founder of anabrid GmbH and professor for business computer science at FOM University of Applied Sciences for Economics and Management. He studied mathematics with a minor in philosophy at the Johannes Gutenberg university Mainz. His main interests are analog and hybrid computing and the simulation of dynamic systems. He also collects and restores classic analog computers and has a soft spot for chaotic systems and their implementation on analog computers.

REFERENCES

[1] UK Parliament Post, "Energy Consumption of ICT", in *POSTNOTE*, Number 677, September 2022

[2] GENE M. AMDAHL, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", in *AFIPS Conference Proceedings*, 30, doi:10.1145/1465482.1465560, pp. 483–485

[3] JOHN L. GUSTAFSON, "Reevaluating Amdahl's law", in *Communications of the ACM*, Vol. 31, Issue 5, doi:10.1145/42411.42415, pp. 532–533

[4] JEFFREY CHOU, SURAJ BRAMHAVAR, SIDDHARTHA GHOSH, WILLIAM HERZOG, "Analog Coupled Oscillator Based Weighted Ising Machine", in *Scientific Reports, nature research*, 15 October 2019

[5] MOHAMMAD KHAIRUL BASHAR, ANTIK MALLICK, DANIEL S. TRUESDELL, BENTON H. CALHOUN, SIDDARTH JOSHI, NIKHIL SHUKLA, "Experimental Demonstration of a Reconfigurable Coupled Oscillator Platform to Solve the Max-Cut Problem", in *IEEE Journal on Exploratory Solid-State Computation Devices and Circuits*, 23 September 2020, pp. 116-121

[6] Sir WILLIAM THOMSON, "Mechanical Integration of linear differential equations of the second order with variable coefficients", in *Proceedings of the Royal Society*, Volume 24, No. 167, 1876, pp. 269–270,

[7] BERND ULMANN, *Analog Computing*, 2nd edition, DeGruyter, 2023

[8] BERND ULMANN, *Analog and hybrid computer programming*, DeGruyter, 2nd edition, 2023

[9] GLENN EDWARD RUSSELL COWAN, *A VLSI Analog Computer / Math Co-processor for a Digital Computer*, Columbia University, 2005

[10] 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