Mecobo: A hardware and software platform for in materio evolution

Odd Rune Lykkebø[1], Simon Harding[2], Gunnar Tufte[1] and Julian Miller[2]

[1] The Norwegian University of Science and Technology
Department of Computer and Information Science
Sem Selandsvei 7-9, 7491 Trondheim, Norway
odd.lykkebo, gunnar.tufte@idi.ntnu.no
[2] Department of Electronics
University of York
Heslington, York, UK. YO10 5DD
slh@evolutioninmaterio.com, julian.miller@york.ac.uk

Abstract. Evolution in Materio (EIM) exploits properties of physical systems to compute. "Designs" are evolved instead of a traditional top down design approach. Computation is a product of the state(s) of the material and input data. Evolution manipulates physical processes by stimulating materials assessed in situ. A hardware software platform designed for EIM experimentation is presented. The platform with features especially for EIM is described together with demonstration experiments using carbon nanotubes in a thick film placed on micro electrode arrays.

1 Introduction

Unconventional computation and unconventional machines try to move beyond the Turing/von Neumann [1,2] concept of computing and computer architecture [3]. Evolution in Materio (EIM) [4,5,6] is such an unconventional approach where the underlying physical properties of bulk materials are explored and exploited for computation. In contrast to a traditional approach where a substrate, e.g. silicon, is meticulous designed, produced and programmed the essence of EIM is neatly phrased as "bulk processes" producing "logic by the pound" by Stewart [7] when introducing his experimental electrochemical system.

The concept of EIM is a truly unconventional approach. Computation is a product of intrinsic physical properties of some material. The computational function is not specified by a traditional sequential program, the program concept is replaced with configuration signals that induce physical processes enabling computation. Configuration signals for a sought computational task are a result of computer controlled evolution (CCE). An Evolutionary Algorithm (EA) explores and exploits intrinsic physical processes by applying some kind of configuration signals.

Figure 1 illustrates a possible scenario for an EIM experimental set-up. The configurable material can be seen as a black box. Incident data are applied, the response is measured and evaluated toward a predefined transferee function. The

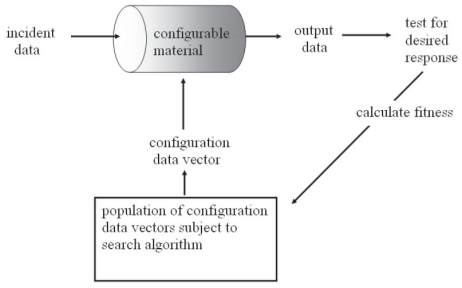


Fig. 1. Principle of evolution in materio.

search algorithm can manipulate physical properties of the material by applying configuration data vectors.

The format of input data, response and configuration data are material specific. As such, any experimental EIM set-up must be capable to produce configurations with properties capable of manipulating physical properties in the material. Incident data properties must be of a type capable of produce an observable response, i.e. output data.

In Thompson's work [8] unconstrained evolution of configuration data for a Field Programmable Gate Array (FPGA) was used to evolve a tone discriminator. The FPGA may be considered as the material. The input signal to this digital circuit was analogue, the response was digital sampling of a captured analogue measurement. The configuration data for the chip was a digital bit stream. Even if Thompson exploited the physical properties of the chip, the configuration vector was digital. Harding and Millers [9] did a similar experiment with liquid crystal as material. In contrast to Thompson's experiment the configuration data signal property for the liquid crystal was evolved, the configuration data was unconstrained with regards to signal type, e.g. analogue, digital and time dependency.

In most EIM work an intrinsic approach has been taken. An intrinsic approach to EIM allowing the physical artefact to be assessed in-situ is preferred to get access to all inherent physical properties of the material [6]. Intrinsic evolution requires an interface that can bridge the gap between the analogue physical world of materials and the digital world of EAs. We propose and demonstrate a flexible platform, Mecobo, designed to interface a large variety of materials. Flexible hardware allows for the possibility to map input, output and configuration terminals, signal properties and output monitoring capabilities in arbitrary ways. The platform's digital side, i.e. EA and software stack, is as important as the hardware. A flexible software platform including hardware drivers, support of multiple programming languages and a possibility to connect to hardware over the internet makes Mecobo a highly flexible platform for EIM experimentation.

Mecobo is part of a the NACSENCE project [10] targeting engineering of nano-scale units for computation. The demonstration experiments presented uses

single-walled carbon nanotubes mixed with poly(methyl methacrylate) (PMMA) dissolved in anisole (methoxy-benzene) as computational material.

The article is laid out as follows: Section 2 presents the nanoscale material and the physical electrical terminals. In Section 3 the architecture of the hardware of the interface is presented. Section 4 present the software of Mecobo. Experiments demonstrating the platform are presented in Section 5. Discussion and conclusion are given in Section 6.

2 Nano Material as Computational Resource

The demonstration experiments in Section 5 show computation in carbon nanotubes. The material samples used are all part of the ongoing NACENCE project. At present time micro electrode arrays are used to connect electrically to a thick film containing nanotube structures. Figure 2 show two different glass slides. At top a slide with 64 electrodes is shown. Left; the glass slide with contacts on the rim. At the Right a microscope image of the array covered with the thick film. On the bottom of Figure 2 a different glass slide type with 12 electrodes is shown. The micro electrode array slides was produced by Kieran Massey at the University of Durham by depositing a solution of carbon nanotubes onto a slide and letting the solvent dry out, leaving a random distribution of nanotubes across the probes in the slide.

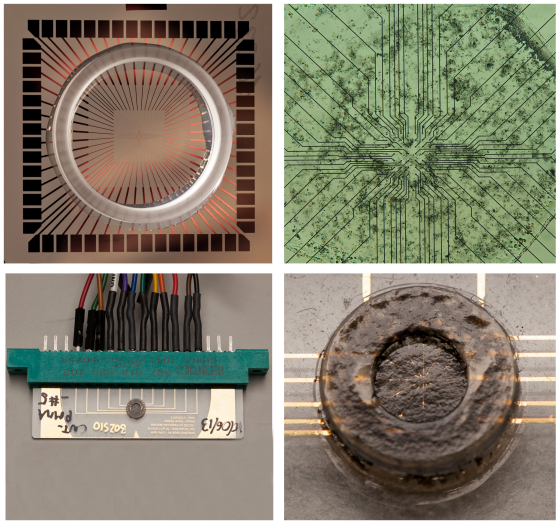


Fig. 2. Samples of materials. Top row: Micro electrode cup. Bottom row: Micro electrode slide.

In the demonstration of the interface in Section 5 the sample used was of the type shown at the bottom of Figure 2. As the main topic here is hardware

and software properties to interface a variety of materials, details regarding the physical properties are not presented in detail. See [11] for information regarding the material slides.

3 Hardware: Interfacing the Black Box

Evolutionary exploration of computation by manipulation of physical systems is an intrinsic [8] approach. If the system is considered as a lump of matter, as illustrated in Figure 1, the selection of signal types, i.e. inputs, outputs and configuration data, assignment to I/O ports may not relate to material specific properties. As such, any I/O port can be assigned any signal type. Further, signal properties, e.g. voltage/current levels, AC, DC, pulse or frequency, needed to unveil potential computational properties of different materials are unknown. To be able to explore and exploit a material's physical properties evolution must have access to explore as unconstrained as possible.

3.1 Interface

The interface is designed to handle all the physical/electrical properties as mentioned above. To be able to ease the process of providing input data to any computational problem the interface also provide the possibility to provide input data. That is, a set of input data signals can be defined as part of the experimental set up to simulate external signals.

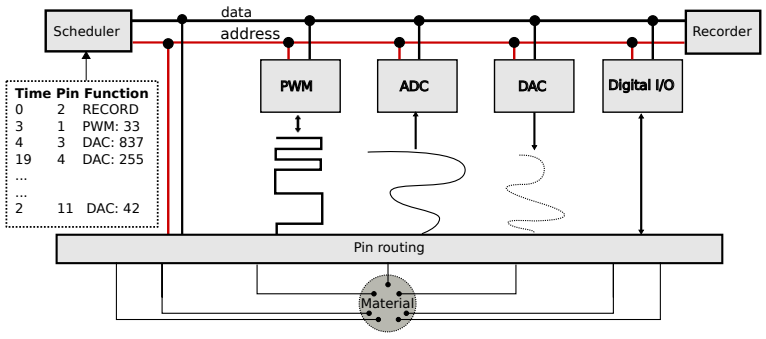


Fig. 3. Overview of the complete system.

Figure 3 show a principle overview of the hardware interface. In the figure an example set up is shown in the dotted box. The shown example genome defines pin 2 to be the output terminal, pin 1 to be the data input and pin 3 - 12 to be configuration signals. The architecture is controlled by a scheduler controlling the following modules: Digital I/O can output digital signals and sample responses. Analogue output signals can be produced by the DAC module. The DAC can be

configured to output statical voltages or any arbitrary time dependent waveform. Sampling of analogue waveforms from the material is performed by the ADC. Pulse With Modulated (PWM) signals are produced by the PWM module.

The system's scheduler can set up the system to apply and sample signals statically or produce time scheduled configurations of stimuli/response. The recorder stores samples, digital discrete values, time dependent bit strings, sampled analogue discrete values or time dependent analogue waveforms. Note that the recorder can include any combination of these signals.

As stated, in a bulk materials there is no specific defined input and output locations, e.g. in the carbon nano-tube PMMA samples is just distributed over the micro-electrode array. As such, the chose of data I/O and configuration terminals should be possible to be put under evolutionary control. In the interface all signals passes a crossbar, i.e. Pin routing. Pin routing is placed between the signal generator modules and the sampling buffer (PWM, ADC, DAC, Digital I/O and Recorder) making it possible to configure any terminal of a material to be input, output or configuration.

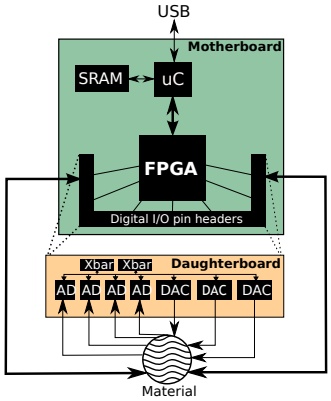
The presented material signal interface in Figure 3 support all our objectives. It is possible to evolve the I/O terminal placement. A large variety of configuration signals are available to support materials with different sensitivity, from static signals to time dependent analogue functions. The response from materials can be sampled as purely static digital signals, digital pulse trains or analogue signals. Further the scheduler can schedule time slots for different stimuli when time dependent functions are targeted or to compensate for configuration delay, i.e. when materials needs time to settle before a reliable computation can be observed.

3.2 Interface Physical Realization

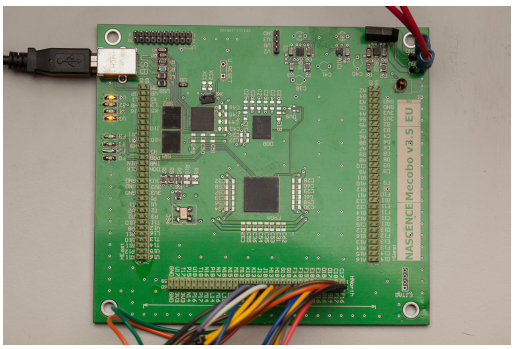
The described system shown in Figure 3 is implemented as an autonomous interface hardware platform. The platform can communicate with a host computer over USB. The host can run an EA or stand as a bridge (server) connected to the internet.

The hardware implementation of the interface, which we call 'Mecobo', is shown as a block diagram in figure 4(a). Mecobo is designed as a PCB with a FPGA as the main component. The system shown in Figure 3 is part of the FPGA design together with communication modules interfacing a micro controller and shared memory. As shown in Figure 4(a) the digital and analogue designs are split in two. All analogue components are placed on a daughter board; such as crossbar switches and analogue-digital converters. This split enables redesign of the analogue part of the system. Making it possible to alter the analogue specification without changing the digital part of the motherboard. The micro controller stands as a communication interface between the FPGA and the external USB port.

Figure 4(b) show the motherboard with the Xilinx LX45 FPGA, Silicon Labs ARM based EFM32GG990 micro controller connected to a 12 terminal material sample.



(a) Mecobo block diagram.



(b) Picture of Mecobo.

Fig. 4. Hardware interface implementation overview.

The motherboard is capable of controlling 80 digital I/O signals, which can be connected directly to a material sample or used for controlling resources on a daughter board. The FPGA drives the I/O pins at LVCMOS33 level, giving a minimum input voltage level for switching to digital 'high' at 2.0V, and maximum 0.8 sensing to 'low'.

The software interface sends commands to the scheduling unit (implemented in the micro controller). The scheduler takes care of controlling the various pin controllers. A pin controller is the abstract term we use to describe a unit that drives or sources a physical I/O pin. Each pin controller has a slice of the global address space of this bus and can be programmed individually by the scheduling unit by outputting the command and data on the bus.

The scheduler accepts a sequence of commands from the user software. Each such sequence item consists of parameters that describe the state of the pin at a given point in time. In Figure 3 for example, pin 2 is set as a 'recording' pin from time 0 (it also has a duration, and sampling frequency attached that is not shown here). Pin 1 is set to output a pulse width modulated version of the value 33, and pin 3 is set to output the analogue voltage level corresponding to 837, which could for instance map to analogue voltage level -2.3V relative to the daughter board analogue ground. In this case the scheduler would issue commands to one of the DAC controllers and to two of the PWM controllers.

4 Software

As there is no known, and hence no standard programming model for in-materio, we developed a system inspired by the track based model of music or video editing applications. An example of this is shown in Figure 5. Each track corresponds to an output pin of the FPGA, and on each track an action (or set of actions) are scheduled. Once the tracks are configured onto the FPGA, the sequence is

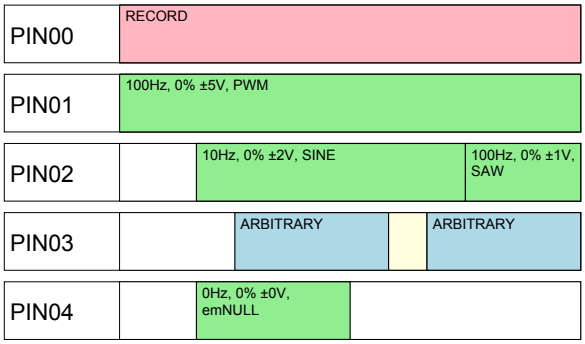


Fig. 5. Illustration of a genotype described in the 'track based' programming model. Each row is an output from the FPGA (and hence an input to the material). The horizontal axis is time. The model aligns closely with the hardware architecture, and also hints at a possible genotype representation.

'played' back. As can be seen in the illustration, 'recordings' can also be scheduled. A recording in this case is the data captured from an input of the FPGA.

From this model, an Application Programming Interface (API) was developed that allows users to interact with the hardware. The main purpose of the API is to expose the functionality of the EM in a consistent and easy to use manner. Additional APIs provide support for data collection and for processing the data itself.

Client applications (i.e. software for performing the evolutionary algorithm), connects to a the EM via control software running on a PC. The control software is responsible for communicating at a low level to the EM, and translating the track based model into the FPGA's internal model.

The control software implements the API as a Thrift Server. Thrift[3] is a technology maintained by Apache that is designed to allow applications running on different operating systems, written in different languages and running on different computers to communicate with each other. Thrift provides a language that is used to define the functionality exposed by the server. This language is then compiled by the Thrift compiler into skeleton code that contains all the functionality needed to act as a server and accept connections, but is missing the functional components. These are then added to complete the server implementation.

On the client side, the interface can be compiled by Thrift into a library that exposes all the methods in the API. Thrift is able to generate the client and

[3] `http://thrift.apache.org/`

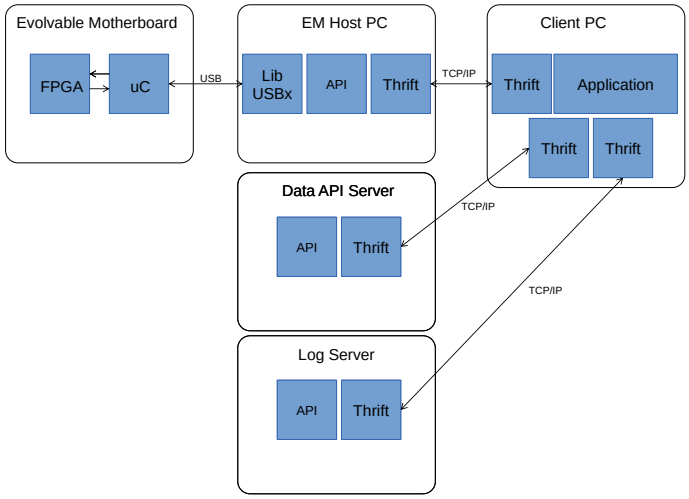


Fig. 6. Overview of the complete software architecture.

server code for many languages including C++, C#and Java. The client library is then connected via TCP (or shared memory if the client/server are both on the same PC) to the server. Client applications then only need to implement their functionality, and no knowledge of the underlying protocols or workings of the EM or server software is required.

As the communication between Thrift Server and Client applications is based on TCP, there is no necessity for all components to run on the same computer. We have successfully tested the API over the internet, and have found that it is feasible for one institute to run the evolutionary algorithm, and another to host the EM.

Figure 6 shows the complete software architecture for the system. On the left we see the hardware (i.e. Mecobo), on the right is the client application. In the middle we see the main API components. Although only one EM is shown, it is practical to add more. Client applications can connect to multiple servers (and hence Mecobos), and hence can control a number of systems in parallel. We envisage this as being useful for robustness testing, investigating repeatability, and allowing multiple Mecobos to work together on a single problem.

5 Initial Experiments

To demonstrate the Mecobo platform, two experiments are presented. The experiments are executed on the presented hardware/software platform using a 12 electrode array similar to the shown example in Figure 2. They demonstrate only a fraction of the capabilities of our hardware and software platform, and those that we expect the material to have.

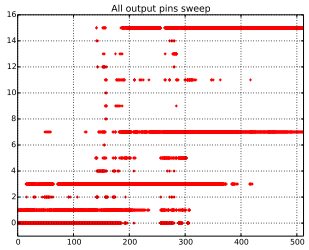
5.1 Exhaustive Sweep

If only digital time-independent logic is considered, then it is possible to run an exhaustive search mapping all possible configurations to the 12 pin sample. To take this into a more "computational" relevance, and to show the effect of interpreting the results when "programming" the material, we interpret two pins as input to a logic gates, the recording pin as gate output and the remaining 9 pins as configuration. To represent functions a *gate output sum* can be constructed. The gate sum is the output of the truth table as shown in in table 1.

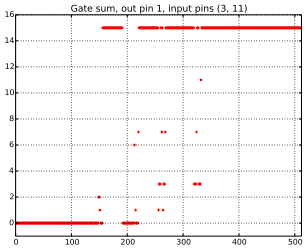
Input	Config	Output
0,0	1,0,1,1,0,1,0,1,1	0
0,1	1,0,1,1,0,1,0,1,1	1
1,0	1,0,1,1,0,1,0,1,1	1
1,1	1,0,1,1,0,1,0,1,1	0

Table 1. Gate sum mapping. XOR. 0110 binary or 6 in decimal.

A '1' represents 3.3V and '0' represents 0V. All possible pin combinations for input, configuration and output is tested and mapped to a functionality plot. If a gate is found it is plotted as a gate output sum, e.g. XOR: 0110 (6). An example of such a plot is presented in Figure 7(a). The plot show all found two input logical functions for all possible input output mappings.



(a) Sweep all I/O combinations.



(b) Sweep fixed I/O combination.

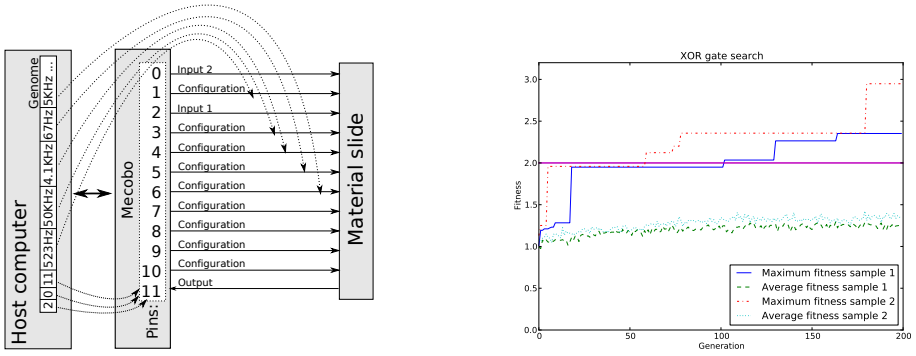
Fig. 7. Example of logic gates found using exhaustive sweep.

The gate output sums are represented in decimal on the vertical axis. The configuration vector is given (decimal) on the horizontal axis. Interesting cases include XOR (gate sum 6 (0110)) and NAND (gate sum 8) are present together with all other possible 2 input logic function. Figure 7(b) show one of the possible I/O configuration. Here all possible gate configurations are shown for one particular I/O mapping, i.e. pin 1 output and pin 3 and 11 as input.

5.2 Genetic Search for Logic Functions

In Section 5.1 it was shown that the nanotube sample was capable of producing logic gates. However, to be able to evolve a sought functionality the EA must be able to exploit and explore the genetic representation and the search space, i.e. evolvability [12] must be present.

To explore evolvability a Genetic Algorithm (GA) was used to search for stable XOR gates. The GA was quite standard; 25 individuals, two crossover points and tournament selection with 5 individuals as elite. Two different material samples was tested.



(a) Representation. (b) Evolved function.

Fig. 8. Representation and results from evolvability test.

To demonstrate the platform's possibility to generate time dependent signals a genotype that opens for dynamic PWM signals was chosen. The GA can adjust each configuration pin within a frequency range of 400Hz - 25MHz. The mapping of which material terminal to use as input, output or configuration was placed under genetic control. Figure 8(a) illustrates the representation. The first two genes assign the input signals, e.g. pin 0 and 2. The third gene gives the output, e.g. pin 11 is sampled by the recorder. The remaining nine pins are mapped to configuration genes specifying frequency values. A restriction must be added to ensure that the same pin is not used for input and output. To ensure "legal" gates, offspring with "illegal" pin mapping was not put back in the population. I/O and configuration was crossed over separately; hence two point crossover.

The GA was set up to search for a stable 2-input XOR logic gate. The response is measured by setting up the gate input pins (i.e. the two first fields of the genome) to constant voltage levels at 0V or 3.3V. The configuration genes are frequencies of square waves, remaining fixed over the 2^2 possible inputs. The response is sampled from the GA-chosen output pin for 100ms at 10KHz. Referring to table1, the number of correct samples in each sample period for each row is further multiplied by a constant, indicating how hard this case is to

find: (0,0): 0.3, (1,0): 0.5, (0,1): 0.5, (1,1): 1.15. Particularly promising cases are further given a 0.5 bonus, giving a total possible fitness of 2.95 for a 'perfect' gate, and 1.95 for a functioning gate.

Figure 8 shows the evolution of fitness for these experiments. The horizontal line at 1.95 indicate the threshold for a functioning XOR gate where the majority of the samples in a sample buffer is over 55%. Note that a functioning XOR was found in both material samples, and in material sample 1 a near-perfect gate was discovered after 150 generations. The difference between the elite best and the average case is quite large. This can be explained by the relatively few XOR gates in the material, which can also be observed in figure 7(a).

6 Discussion and Conclusions

The presented hardware and software platform for EIM experimentation show a tool enabling exploration and exploitation of materials for computation purposes. The flexibility in signal levels and types together with the possibility to put the mapping of input, output and configuration terminals under evolutionary control offers a possibility toward unconstrained material evolution.

The presented results demonstrate how the platform can be used. There are several interesting aspects that is worth a note. In the exhaustive sweeps presented in Section 5.1 the gate sum plots are a course mapping of possible computational properties of the material. The plot in Figure 7 show that this sample is capable of solving problems beyond simple threshold functions. As such, results for such sweep plots can be used to coarsely classify and as a measurement of closeness/distance between materials samples within a batch or batches with different physical properties.

Even if a material is capable of solving a function, like the XOR, it is not necessarily easy to evolve. Further, as indicated by earlier EIM work stability of found solutions may be problematic. The example given shows two important factors. Stability can (and should be part) of the problem definition. The change of representation shows the possibility to provide a verity of signal types. The material used in the example show computational properties for static and dynamic configuration data.

As stated in Section 1 this work is part of a bigger project. The platform is in use by several researchers in the Nascense project consortium, e.g. University of York [13] for function optimization.

Software and production files can be downloaded from: http://www.nascence.eu[4]. It is also possible for researchers to connect to Mecobo from: http://www.nascence.eu[4].

Acknowledgements

The research leading to these results has received funding from the [European Community's] Seventh Framework Programme ([FP7/2007-2013] [FP7/2007-2011]) under grant agreement no [317662].

[4] Will be available at publication date

References

1. J. von Neumann. First draft of a report on the edvac. edited by m. d. godfrey 1992. Technical report, Moore School of Electrical Engineering University of Pennsylvania, 1945.
2. A. Turing. On computable numbers, with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society 1936-37*, volume s2-42 of *2*, pages 230–265. London Mathematical Society, 1937.
3. C. Teuscher and A. Adamatzky. *Unconventional Computing 2005: From Cellular Automata to Wetware*. Luniver Press, 2005.
4. J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In *2002 NASA/DOD Conference on Evolvable Hardware*, pages 167–176. IEEE Computer Society Press, 2002.
5. S. L. Harding, J. F. Miller, and E. Rietman. Evolution in materio: Exploiting the physics of materials for computing. *Journal of Unconventional Computing*, 3:155–194, 2008.
6. J. F. Miller, S. Harding, and G. Tufte. Evolution-in-materio: evolving computation in materials. *Evolutionary Intelligence*, 7(1):49–67, 2014.
7. R. M. Stewart. Electrochemically active field-trainable pattern recognition systems. *IEEE Transactions on Systems Science and Cybernetics*, 5(3):230–237, July 1969.
8. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *1st International Conference on Evolvable Systems (ICES96)*, Lecture Notes in Computer Science, pages 390–405. Springer, 1997.
9. S. L. Harding and J. F. Miller. A tone discriminator in liquid crystal. In *Congress on Evolutionary Computation(CEC2004)*, pages 1800–1807. IEEE, 2004.
10. H. Broersma, F. Gomez, J. F. Miller, M Petty, and G. Tufte. Nascence project: Nanoscale engineering for novel computation using evolution. *International Journal of Unconventional Computing*, 8(4):313–317, 2012.
11. A. Kotsialos, M. K. Massey, F. Qaiser, D. A. Zeze, C. Pearson, and M. C. Petty. Logic gate and circuit training on randomly dispersed carbon nanotubes. *Submitted to: International Journal of Unconventional Computing*, 2014.
12. M. Kirschner and J. Gerhart. Evolvability. *Proceedings of the National Academy of Sciences of the United States of America*, 95(15):8420–8427, July 1998.
13. M Mohid, J. F. Miller, S. L. Harding, G. Tufte, O. R. Lykkebø, K. Massey, and M. Petty. Evolution-in-materio: Solving function optimization problems using materials. In *Submitted to: Unconventional Computation and Natural Computation*. 2014.