A Simulation Tool for Evolving Functionalities in Disordered Nanoparticle Networks

Ruud van Damme*, Hajo Broersma*, Julia Mikhal*, Celestine Lawrence*, and Wilfred van der Wiel*
*Faculty of Electrical Engineering, Mathematics and Computer Science
CTIT Institute for ICT Research and MESA+ Institute for Nanotechnology
University of Twente, The Netherlands
Email: {r.m.j.vandamme,h.j.broersma,j.mikhal,c.p.lawrence,w.g.vanderwiel}@utwente.nl

***Abstract*— Recently published experimental work on evolution-in-materio applied to nanoscale materials shows promising results for future reconfigurable devices. These experimental results are based on disordered nanoparticle networks, without a predefined design. The material is treated as a black-box, and genetic algorithms are used to find appropriate configuration voltages to enable a targeted functionality. To support future experimental work, we developed simulation tools for predicting candidate functionalities. One of these tools is based on a neural network model, but the one presented here is based on a physical model. The physical model describes the charge transport between the nanoparticles, which is governed by what is known as the Coulomb blockade effect. The new simulation tool combines a genetic algorithm with Monte-Carlo simulations that are based on this physical model. The code of the new simulation tool has been validated with known results on small deterministically designed nanoparticle networks from literature. The code has also been applied to simulate reconfigurable logic in small $k \times k$ grids of nanoparticles. The results show that the new approach has great potential for partly replacing costly and time-consuming experiments.**

I. Introduction and Motivation

Within the framework of a European FP7 project called NASCENCE [1], we have studied and exploited the dynamics of the charge transport in disordered nanoparticle (NP) networks that have been prefabricated without much control on their design. The aim was to apply evolution-in-materio [2] to evolve such NP networks after fabrication into functional units by fixing a number of input and output electrodes, and using another set of electrodes to change and control the state of the NP networks. The latter was achieved by applying artificial evolution in the form of a genetic algorithm (GA) in order to optimise the control voltage setting for a particular functionality. It has recently been shown experimentally [3] that it is possible to use this approach to evolve such fixed prefabricated disordered NP networks *in situ* into any of the 16 possible binary logic functions by fixing two input electrodes and one output electrode, and successively finding and setting the appropriate control voltages on six control electrodes, including a back gate. These results were produced with a high degree of stability and reproducibility, and have recently been published [3]. This experimental work presents a proof of principle that indicates very promising prospects for using this material to perform more complicated computational tasks.

In the next section, more details on the structure and fabrication of these NP networks will be given, together with an explanation of the approach to achieve the wanted functionality. It is highly desirable to have a simulation tool to explore the potential functionalities of such NP networks without the burden of spending many hours in the lab and wasting expensive resources to look for such functionalities experimentally. This was the main motivation for developing the current simulation tool. Next to that, simulations can yield useful information on the minimum requirements that are needed for obtaining the targeted functionality if we were able to produce these NP networks according to a predetermined design. This could lead to new devices for the digital industry, possibly replacing purpose-built assemblies of transistors. Moreover, simulations can provide evidence for the scalability of the approach. This could lead to future applications of these NP networks in the area of Cellular Neural Networks. Simulations can also give new insights into the dynamics of the charge transport that might lead to a better understanding as to why and how the NP networks reveal the functionalities we observe.

The rest of the paper is organised as follows. In Section II, some technical details are presented on the fabrication, structure and charge transport in the gold NP networks that were used in our experimental work, as well as how artificial evolution was applied to look for functionalities. Section III describes existing models and simulation tools for this type of NP networks, and the choices that form the basis for the developed simulation tool, combining a GA with Monte-Carlo simulations for charge transport. In Section IV, the newly developed simulation tools are validated by comparing the simulation results on small deterministically designed NP networks with known functionalities from literature. New simulation results that have been obtained for larger NP networks are presented in Section V. We finish the paper with some conclusions and an outlook to future work in Section VI.

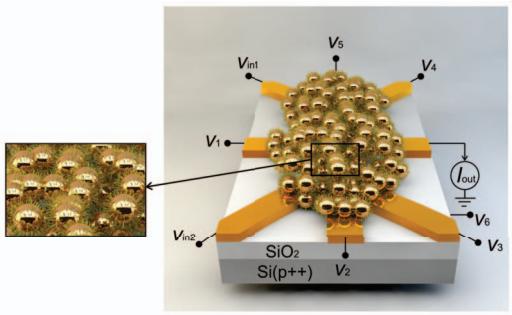


Figure 1. Artist's impression of a disordered NP network

II. NP Networks

In this section, some details regarding the fabrication, structure and evolution are given for the NP networks that have been used in the experimental work that led to [3]. These NP networks were produced in collaboration with the NanoElectronics group at the MESA+ Institute for Nanotechnology of

978-1-5090-0623-6/16/$31.00 ©2016 IEEE

the University of Twente. Figure 1 shows an artist's impression of such a network.

These NP networks consist of disordered networks of 2-20nm Au NPs functionalised by a layer of 1-octanethiols, acting as a tunnel barrier. The NPs are trapped by dielectrophoresis in between 8-12 Ti/Au electrodes on top of a highly doped Si/SiO_2 substrate, functioning as a back gate. The central circular region in between the electrodes has a diameter of about 200nm. This implies there are in the order of a hundred to several hundreds particles, depending on the size of the individual particles and the parameters used in the process of trapping. An example of a fabricated NP network is shown as an atomic-force microscopy (AFM) image in Figure 2. More details on the production process and its characteristics can be found in [3].

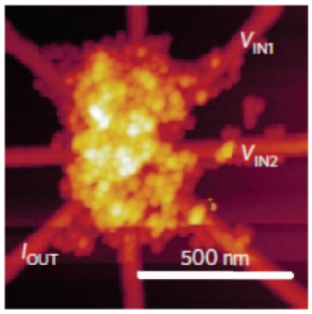


Figure 2. AFM image of one of the fabricated NP networks

At low temperature, where the NP's charging energy is larger than the thermal energy, the system of NPs behaves as a disordered Coulomb island network in which each NP acts as a single-electron transistor (SET) [4]. One electron at a time can tunnel when the charging energy is provided, either by applying a voltage across the SET or by electrostatically shifting its potential. Otherwise, the transport is blocked due to Coulomb blockade [5]. The system of randomly assembled NPs therefore provides a network of interconnected, robust, non-linear, periodic switches, as a result of the Coulomb oscillations of the individual NPs.

Two electrodes are used for applying time-dependent voltage signals (as indicated by V_{in1}, V_{in2} in Figures 1 and 2). All but one of the remaining electrodes are used for applying static control voltages (as indicated by $V_1, \ldots, V_5$ in Figure 1), and one is used for measuring the resulting time-dependent current (as indicated by I_{out} in Figures 1 and 2). In addition, a static voltage can be applied to the back gate (as indicated by V_6 in Figure 1). The strongly non-linear switching behaviour of the SETs and their mutual interactions give rise to functionality greatly depending on the control voltages.

Measurements in [3] indeed indicated that low-temperature electron transport is dominated by Coulomb blockade [5] and that its detailed behaviour strongly depends on the used input and output electrodes, as well as on the static voltages applied to the remaining electrodes. These characteristics lie at the basis of the realisation of logic gates.

As mentioned before, fixing two of the electrodes as inputs and one as an output, the remaining electrodes together with the back gate have been used to configure the NP networks for target functionalities, applying a GA. More details on this GA will appear in the next section, where the same GA in used for the simulations. It has been demonstrated experimentally in [3] that such NP networks can be configured as any of the basic Boolean logic gates with a very high degree of stability and reproducibility. Importantly, any changes made to the NP networks were completely reversible, so different configurations could be programmed into the same NP network one after the other. This proof of principle indicates very promising prospects for using this material to perform more complex tasks. It will be obvious from the above description that the experimental work of fabricating, testing and evolving the NP networks is time consuming and uses valuable resources. Moreover, there are many questions on the use of these NP networks that are difficult to answer experimentally, because there are serious challenges in fabricating NP networks with smaller central gaps or with more control electrodes using the same area. Therefore, it was desirable to develop a simulation tool specifically for exploring these large disordered NP networks.

III. Modelling and Simulation

In this section, the simulation tool is going to be described, together with the choices we made in developing the tool. This section contains a more detailed description of the physical and quantum phenomena that play a key role in the modelling and simulation of the above NP networks.

A. Monte-Carlo Method

The assembled investigated NP networks are large in terms of the number of NPs and disordered, and almost random in nature. The transport of electrons is governed by the Coulomb blockade effect: transport is blocked, except at almost discrete energy levels; there exactly one electron can jump. The dynamics of such a system is governed by stochastic processes: electrons on all islands (NPs) can tunnel through junctions with a certain probability. For such systems, there are basically two simulation methods to our disposal: Monte-Carlo Methods and the Master Equation Method [6], [7].

The main approach in the Master Equation Method is to obtain the set of relevant states the NP network under investigation can occupy, so that one can set up the rate matrix. The states and their transition rates lead to a usually large set of ordinary differential equations (ODEs), referred to as the Master Equations, which may be written in matrix form. For networks with relatively few NPs, i.e., with a limited set of possible charge states, this set of ODEs can be solved with numerical software [4]. The advantage of this method is clear: one obtains all probabilities in steady state averages and fluctuations of any variable. The problem of rare events can also be circumvented in this way [7]. The drawback is also clear. In principle, there are almost infinitely many states, as each NP can host many electrons. So a choice has to be made which states are incorporated in the computation. Secondly, even if this number of states for each NP is limited, say to three, a network of only 40 NPs boils down to a coupled system of ODEs of $3^{40} \approx 10^{19}$ variables. Indeed, the Master Equation approach is in practice only used for small networks in case one needs an accurate description [4], [7]. And even then the Master Equation Method uses an adaptive method, where in the course of solving the steady state some states are removed or added [6], [8].

Since the number of particles in our NP networks is relatively large, the Monte-Carlo Method is the best candidate. This method simulates the tunnelling times of electrons stochastically. To get meaningful results, one needs to run the algorithm in the order of a million times. Doing so, the stochastic process gives averaged values of the charges, currents, voltages, etc. In principle, it can handle an arbitrary network of any size, but of course computations take time and scalability is an issue if we want to simulate networks consisting of several hundreds of particles.

B. Simulation Tools

The simulation tool that is going to be described next is an extension of existing tools for simulating nanoparticle interactions, like SPICE [9] or SIMON [10]. The latter packages have been developed for the simulation of, usually small, networks consisting of components such as capacitors, inductors, resistors and extended objects such as diodes and transistors. They all have in common that they intend to describe these networks with a reasonably high accuracy, at the cost of high computation times. Therefore, scalability is a serious issue if we would consider applying these existing methods to the NP networks we are exploiting at the moment. Moreover, our approach to these networks is different. We do not need to have accurate results. We are not aiming at understanding these large disordered NP networks in great detail; to the contrary, we are interested in global properties of these networks, such that we can steer the experimentalists into the more interesting regions of design and exploitation. In fact, it is simply impossible to describe the precise structure and properties of the interconnected components, so we cannot hope for a high accuracy. A final objection against opting for the existing simulation tools is that the flexibility is limited. It generally will cost a lot of energy and time to add elements with different properties. This will become even more relevant as soon as we get a better understanding of the NP networks from a more theoretical point of view. This will be the case, for example, when we derive and want to test simplifications of our models. We are thinking ahead of potential possibilities for future models and simulations, for example in a discrete mathematical setting or of a more deterministic nature, as opposed to a fully stochastic Monte-Carlo simulation.

C. Components and Relationships

A single-electron circuit consists of branches (capacitors, tunnel junctions, voltage leads, and current leads) that start and end at nodes. The electrical description or characterisation of such a circuit consists of all node potentials, node charges, and branch currents. Usually only a part of the node potentials, node charges, and branch currents is known. The unknown voltages, charges and currents have to be calculated using Kirchhoff's laws. Next, we describe the different nodes we distinguish, followed by the relationships that are relevant for calculating the unknowns.

Voltage nodes. If the potential of a node is given, the node is called a voltage node. This is the case for a node that is connected to a grounded voltage source (like the node labeled by 1 in Figure 3), since the voltage source defines the potential of the node.

Internal nodes. Internal nodes are nodes with a known charge, and they are either nodes with a constant charge, or nodes the charge of which can change by an integer times the elementary charge. In Figure 3, the nodes labeled by 2, 4 and 6 are internal nodes. An internal node has a constant charge if it is connected to capacitors only, since in that case no charge can enter or exit, such as the node labeled by 6 in Figure 3. If an internal node is connected to at least one tunnel junction, its charge can change by an integer times the elementary charge due to electrons tunnelling onto and off the node.

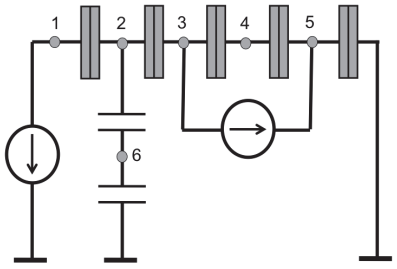


Figure 3. An example network with different types of nodes

Floating nodes. Not grounded voltage sources which are not connected to other grounded voltage sources produce floating nodes. A priori neither charge nor potential is known of such nodes. However, the potential differences of all floating nodes in a macro-node are known. For example the potential difference between the nodes labeled by 3 and 5 in Figure 3 is determined by a voltage source. In our experiments such nodes will not occur, so therefore we did not implement this option.

In the following we put subscripts v for quantities on voltage nodes and i for internal nodes.

The relationship between charges and potentials is given by the capacitance matrix C:

$$\mathbf{q} = C \cdot \mathbf{v}$$

This matrix C is symmetric. Known quantities are voltages on voltage nodes, $\mathbf{v}_v$, and charges on internal nodes, $\mathbf{q}_i$. From this one can compute the unknown quantities: charges $\mathbf{q}_v$ on voltage nodes and voltages on internal nodes, $\mathbf{v}_i$. The relationship between all these variables can be written down in the following alternative form:

$$\begin{bmatrix} \mathbf{v}_v \\ \mathbf{q}_i \end{bmatrix} = A \cdot \begin{bmatrix} \mathbf{v}_i \\ \mathbf{q}_v \end{bmatrix}, \tag{1}$$

where A is (usually) a sparse matrix.

Obtaining the unknowns requires that we solve the above system. We are in the fortunate position that the matrix A in most cases does not change (during the optimisation process in which we apply the GA this need not always be the case).

The usual standard approach to solving such linear systems is to make an LU decomposition of A once, at the start of the computation. This is rather easy to program, and it works satisfactorily well in case of small networks. It is an order N^2 method, with N the size of the matrix. For larger networks, say with $N \geq 50$ nodes, it pays off to turn the process into an order N process, at the cost of a large overhead due to preprocessing the data. We omit the details due to page limitations, but we plan to extend this article to a journal version which will contain the details. We have also developed a parallel algorithm for this, using CUDA on a GPU.

D. Electron Jumps

Up to now, the equations concerned a fixed network with capacitances only. So if the stimuli, e.g., voltages $\mathbf{v}_v$ are constant, the physical system will instantaneously be in a stationary solution. However, electrons might tunnel through junctions between neighbouring islands (nodes, NPs). The modelling of this phenomenon requires the phenomenological quantity 'tunnel resistance' R of a junction. Quantum mechanical processes are described by probabilities (in our case depending on the tunnel rates). Fermi's golden rule is the fundamental result that governs the processes: given two states i (initial) and f (final), with a change in free energy

$$\Delta F = F_f - F_i,$$

the tunnel rate $\Gamma(\Delta F)$ is determined by:

$$\Gamma(\Delta F) = -\frac{\Delta F}{e^2 R}\left(1 - \exp\left(\frac{\Delta F}{k_B T}\right)\right)^{-1}, \qquad (2)$$

where e is the elementary charge carried by a single electron, k_B is Boltzmann's constant, T is the temperature, and R is the empirical value of the resistance of the relevant junction. For $T \to 0$ this rate becomes

$$\Gamma(\Delta F) = \begin{cases} 0 & \Delta F \geq 0, \\ -\dfrac{\Delta F}{e^2 R} & \Delta F < 0. \end{cases} \qquad (3)$$

In a multi-junction setting, the Monte-Carlo method has to be applied to all junctions, in order to examine all possible tunnellings, to compute (estimate) the rates for all these events and pick an event with a certain probably, consistent with the tunnel rates Γ. The change in free energy is given by:

$$\Delta F_j = -e(v_j - v_{0,j}),$$

with

$$v_{0,j} = \frac{e}{2(C_j + C_{j,\text{eff}})}$$

These numbers can be substituted in (2) or (3) in the numerical process. The constant $C_{j,\text{eff}}$ can be computed as the effective capacitance of the Thévenin equivalent circuit. We omit the details.

Based on the above relationships, we can compute all probabilities (rates) of all possible transitions, so we are in the position to simulate the process by a Monte-Carlo approach.

E. Monte-Carlo Simulation Algorithm

Details of the simulation program are outlined in a pseudocode as follows [11].

Monte Carlo Algorithm

```
1. Preprocessing:

1.1  Take input description of the circuit;
1.2  Compute all possible tunnelling events
     and the capacitance matrix C;

2. The core: Repeat until statistics is OK

2.1 Compute unknown charges and voltages;
2.2 Compute the tunnelling rates;
2.3 Choose the next event;
2.4 Compute all changes.
```

As we explained, the electrostatic energy of an NP network can be calculated from its capacitance matrix C. If the charge configuration changes, node voltages change, and the electrostatic energy changes. The system tunnels from a state of higher electrostatic energy to a state of lower electrostatic energy. The change in electrostatic energy depends only on the difference in voltage between the initial and final nodes, plus a term that is independent of the charge state of the system.

Remark: An alternative but equivalent approach is deduced from a thermodynamical treatment, based on the so-called Gibbs free energy. Since it leads to exactly the same results, we do not explain it in this paper. For a detailed account, see [12].

The only thing we have not accounted for yet is the GA that will be used to find appropriate settings of the control voltages.

F. Genetic Algorithm

In a given optimisation problem which we are going to solve with an NP network, a GA is used to search for the solution of the NP network configuration. For a GA to run we need:

- an objective function;
- a genome which represents the free variables.

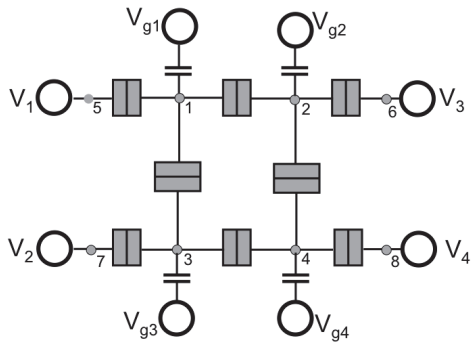


Figure 4. A circuit representing a small NP network

Consider a fixed NP network, e.g., the one corresponding to the circuit representation of Figure 4. It is described by

- the topology of the network (a graph); here we have 8 nodes (NPs), 8 junctions (the grey double-blocks in the figure) and 4 normal (gate) capacitors,
- the C and R values of the junctions,
- the C values of the gate capacitors,
- the lead voltages on the leads $(V_1, \ldots, V_4)$
- the gate voltages on the leads $(V_{g1}, \ldots, V_{g4})$

In the experiments, we can only change the lead and gate voltages. Moreover, the gate voltages are applied globally (so $V_{g1}, \ldots, V_{g4}$ represent one back gate, like V_6 in Figure 1), hence we take only one degree of freedom for that quantity.

Now assume we wish to evolve Boolean logic in this (unrealistically small) network, for example a NAND. We need two inputs and one output for the NAND function. So we are

left with only one of the leads $V_1, \ldots, V_4$ and the back gate which can be used to configure the network. In this case, the genome would consist of two variables. In general, the number of variables in the genome is the number of leads (including the back gate) minus the number of input and output leads we need for the functionality we want to evolve.

In our current code we use the following simple GA, with evolving populations of 20 individuals (genomes). Below, 'cross' means uniform crossover, with each of the genes in the offspring assigned randomly from either parent, with a mixing ratio of 0.5, followed by the randomisation of every gene with mutation probability 0.1. For more details, see [3].

The Genetic Algorithm

```
for k = 1 to 20
generate a random genome W[k]
compute its fitness
end
sort the genomes decreasing in fitness:
W[1],...,W[20]

while true (the core of the GA)
0) create empty list K
1) add W[1],...,W[5] to K
2) add mutated W[1],...,W[5] to K
3) cross W[m] * W[m+1], m=1-5,
and mutate; add to K
4) cross W[i] * W[j_i], i=1-5,
j_i random from 1-20,
and mutate; add to K
5) compute fitness of K[1],...,K[20]
6) sort K to form a new W
```

For our current purposes this simple GA is sufficient, but developing better GAs is an important topic for future investigation, in particular if we turn to more complex functionalities and larger networks. To complete the description of our method, we next explain which fitness function we used.

G. Fitness Function

We want to define the quality (fitness) of a function f (the measurements, currents) on $D = [0,1]^2$. This function f is 'the best' in case

$$f(x,y) = \begin{cases} 0 & \text{if } [x,y] \in A_0 \\ c & \text{if } [x,y] \in A_1 \end{cases},$$

with $A_0 \bigcup A_1 = D$, where the regions A_0 and A_1 depend on the specific target function. There is an ambiguity in this definition, as this f is a current for which we have no scale: c is unknown. We first approximate this f with a piecewise target function:

$$f(x,y) \approx C_0\phi_0(x,y) + C_1\phi_1(x,y),$$

with

$$\phi_0(x,y) = 1, \;\; \phi_1(x,y) = \begin{cases} 0 & \text{if } [x,y] \in A_0 \\ 1 & \text{if } [x,y] \in A_1 \end{cases},$$

and we use the L^2-norm:

$$\epsilon^2 = \min_{C_0,C_1} \int_D \left(f(x,y) - C_0\phi_0(x,y) + C_1\phi_1(x,y)\right)^2 \mathrm{d}x\mathrm{d}y$$

Now the question is: what definition do we use for quality? If ϵ is small, the quality is high, and if the ratio of a '1' as output compared to '0' is large, the quality is high (the on/off ratio is high). There are infinitely many choices possible. We experimented with a quality q of the form

$$q = \frac{1}{\sqrt{\epsilon}} \cdot (M - m) \cdot \left(\frac{M}{m}\right)^k,$$

$$\text{with } m = \max_{[x,y]\in A_0} |f(x,y)| \text{ and } M = \max_{[x,y]\in A_1} |f(x,y)|,$$

and the optimal value for k appears to be 1.

There may occur problems when both $m, M \to 0$. We adjusted the criterion to take this into account. For instance, when we know that $|c| > 10^{-12}$, we assigned $q = -\infty$ in case $M < 10^{-12}$.

H. Complete Simulation Package

The software that has been developed is based on the above Monte-Carlo approach for NP networks, including the GA. We wrote this simulation code in C++; it can handle arbitrary random networks. These networks can contain, in principle, thousands of components. Extensions to more complicated elements are also possible, without much effort: this was precisely the reason for opting for a self-made code, as we argued before. Next to this C++ code, we have two other software components that can be coupled to it:

A **Matlab code**, which enables the study of networks in 'slow motion', in order to get grip on the real performance of such networks. This is necessary for answering the question: how can we approximate NP networks, for example with a similar network of a more discrete nature, such as Ising models, percolation type of approaches, or others approaches [13] to be explored in the next stages of our research. This Matlab code uses a compiled version of our C++ code running in the background.

A **Labview code**, in which also the GA is incorporated. This Labview code is also directly linked to the experimental setup. So the Labview code enables experimentation and possible exploitation of the produced NP networks, as well as experimentation with the simulated networks and possible predictions of potential computational tasks.

IV. Validation with Known Examples

We validated our code with the simplest NP networks that exhibit rudimentary logic, and for which theoretical and experimental data are available from literature. We describe these small designed NP networks and refer to the bibliographic sources in the following subsections.

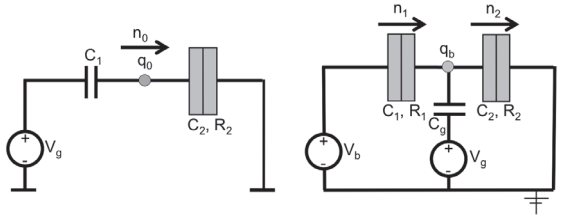


Figure 5. Left: a Single Electron Box; right: a Single Electron Transistor.

1) Single Electron Box: This device, that is illustrated in the left part of Figure 5, consists of just one small island (NP), separated from a larger electrode (electron source) by a tunnel barrier [6]. An external electric field may be applied to the island, using another electrode (gate), separated from the island by a thicker insulator which does not allow noticeable tunnelling. The external field changes the electrochemical potential of the island and thus determines the conditions for electron tunnelling. This device is a building block for larger devices, the most important of which is the Single Electron Transistor (SET) that is depicted in the right part of Figure 5.

2) Single Electron Transistor: This device is obtained by splitting the tunnel junction of the single electron box and applying a DC voltage between the two parts of the external electrode [5], [6], [14]. The resulting SET is probably the most important device in this field. It is reminiscent of a usual MOSFET, but with a small conducting island embedded between two tunnel barriers, instead of the usual inversion channel. The relevant parameters for the functioning of the device, as well as for the larger networks, are: the capacitances of the capacitors, C_g, and of the junctions, C_j, the tunnel resistances of the junctions, R_j, the initial charges at the islands, q_i, the temperature T, and the voltages of the sources (V_b, V_g). The physics is completely determined by the number of electrons that jump through the tunnel junctions (n_j).

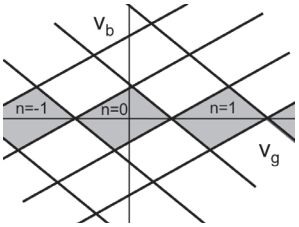


Figure 6. Simulated stability diagram of a SET.

In Figure 6, we show the results of our simulation tool for the stability diagram (Coulomb diamonds) of a SET [6], which shows a full correspondence. We omit the explanation.

3) Voltage State Logic – Inverter: As a third example, we show the logic network of an inverter, see Figure 7. This particular network, that is taken from [15], is also experimentally fabricated and characterised.

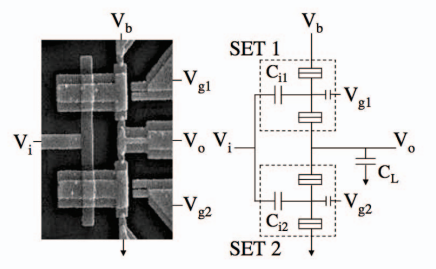


Figure 7. The inverter, taken from [15].

In Figure 8, we show simulation results obtained by the presented simulation tool for the V_{in} vs. V_{out} characteristics of the inverter of [15], which also shows a full correspondence.

V. New Results

As an example which is still relatively small and manageable, but shows interesting features, we tested our methods on the symmetric 4x4-grid consisting of the components shown in Figure 9. Currently, we do not know of any technology to actually fabricate such a grid, unfortunately.

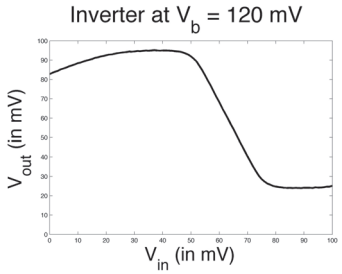


Figure 8. Results of simulations on the inverter of [15].

In Figure 9, the 16 green dots represent the NPs; in between are the tunnelling junctions with fixed C and R values. The two input leads and the single output lead are depicted as I_1 and I_2 and O, respectively. Voltages are applied on V_1-V_9 according to the GA, and also to a back gate that is connected through tunnel barriers (a silicon oxide layer) to all NPs (for convenience we have not shown the back gate in the figure).

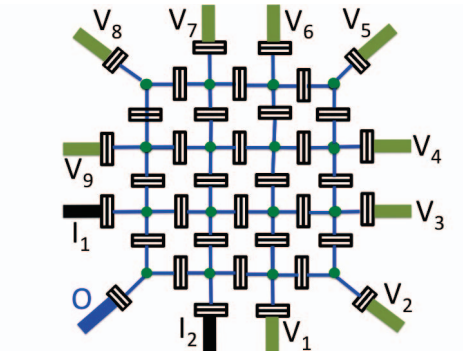
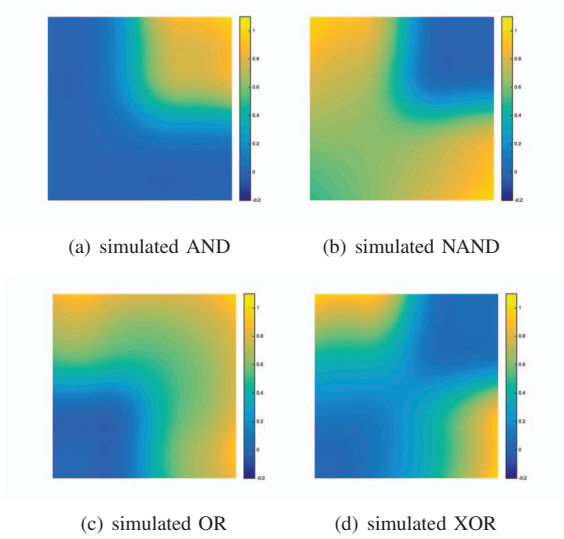


Figure 9. A symmetric 4x4-grid of 16 NPs (green dots) with leads

The fitness of the sets of configuration voltages is determined by how close the output for the four input combinations of the Boolean truth table is to the desired logic, as described in Section III. Applying our simulation tool to the small network of Figure 9, we were able to evolve all basic Boolean logic gates with different settings of the values of the free variables (the lead voltages and the back gate voltage). We illustrated the solutions for just four of the cases (AND, NAND, OR and XOR) as contour plots in Figure 10. The four plots are functions of the two input signals; the voltages of both inputs range from 0 to 10mV, horizontally as well as vertically (with 0,0 in the left bottom corner of each of the plots); the colour scheme ranges from blue for small values (close to a logical 0) to yellow for high values (close to a logical 1) of the output. A good correspondence can be seen from the plots between the target and simulated functionality.

From an electrical engineering point of view, these simulation results are very interesting for several reasons. First of all, the (theoretical) NP network of Figure 9 can be configured into any of the basic Boolean logic gates, using only 16 NPs of size 5-20nm. If we would like to design and build the same functionality with transistors, we would require at least 10 transistors. Secondly, in the designed circuit we would have

to rewire the input and apply it at different places, whereas in the (simulated) NP network we apply each of the input signals at exactly one place. Even with current transistor sizes below 20nm, the designed circuit would require the same or more space.



(a) simulated AND (b) simulated NAND

(c) simulated OR (d) simulated XOR

Figure 10. Contour plots of simulated evolved logic in the 4x4-grid.

It is interesting to note that in the experiments with the real material samples [3], we can also evolve all basic Boolean gates within an area of a diameter of around 200nm and with only six control voltages, but currently these samples consist of 100-150 NPs that are self-assembled into a disordered network. This shows the great potential for the approach, both in the bottom-up and top-down design regime. Currently, we are not aware of any production techniques for fabricating samples that come anywhere close to the 4x4-grid structure of Figure 9. To obtain deeper insight into the underlying currents and physical phenomena, and with the long term goal to fully understand what is going on in terms of electron jumps and currents through these NP networks, we have developed visualisation tools to analyse the processes that are taking place over time.

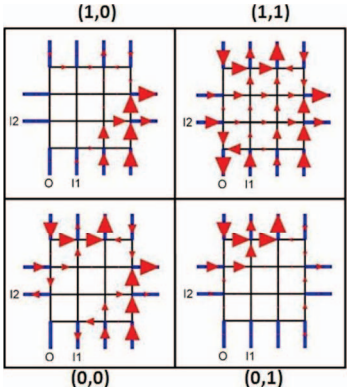


Figure 11. Current patterns for simulated AND in a 4x4-grid of 16 NPs.

In Figure 11, we presented some pictures visualising the currents through the network, which in this case, as an example, was configured as an AND. The current amplitudes are proportional to the area of the red arrows in the figure. The currents are all averaged over time, but we can also calculate variances, and show fast animations of the electron jumps. We have not been able to deduce a satisfactory explanation for the patterns and jumps that cause the 4x4-grid to behave as a logic AND (or one of the other basic Boolean logic gates). Figure 11 helps visualising how complex the traffic of electrons in such networks can become, and it can give us more insight as to why they behave as logic.

We also simulated the evolution of Boolean logic in $k \times k$ grids of NPs with $k \neq 4$, and for different symmetric positions of the two input leads relative to the output lead in the left bottom corner. We cannot discuss these results here in detail because of page number limitations, but we highlight some of the most striking results and conclusions next. Perhaps most strikingly, the evolved Boolean logic did not get much better in quality by increasing k to 5 or 6, or by adding diagonal junctions (such that the internal NPs have six instead of four neighbouring NPs). Figure 12 shows the best evolved NAND (upper three pictures) and XOR (lower three pictures) for $k = 4, 5, 6$ (from left to right).

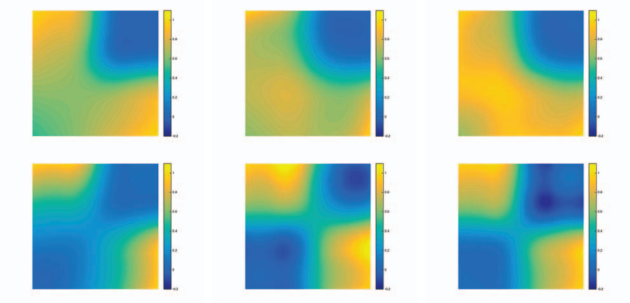


Figure 12. Upper three: the best evolved NAND for $k = 4, 5, 6$; Lower three: the best evolved XOR for $k = 4, 5, 6$.

For $k = 4$, in Figure 13, we plotted the best quality NAND (upper three) and XOR (lower three) that we evolved in case the input leads were chosen symmetrically at distance 1, 2 and 3 (from left to right in the figure) from the output lead along the two sides of the grid.

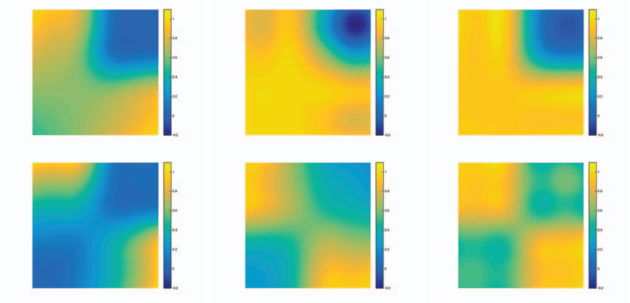


Figure 13. Best NAND, XOR for $k = 4$ with input leads at distance $1, 2, 3$ of the output lead.

Although reasonably clear logic functions could be evolved in all cases, there seem to be better overall results if the choice of input leads is such that they are closer to the output lead.

For $k = 3$, we were still able to simulate an evolved AND, XOR, and OR, as illustrated in Figure 14. In this case, for $k = 3$, a good quality NAND and NOR are not easy to evolve,

as we can conclude from Figure 15, and for the NXOR we did not get any reasonable result.

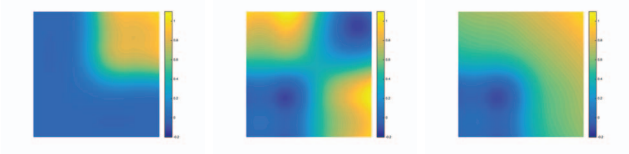


Figure 14. Best AND, XOR, OR for $k = 3$.

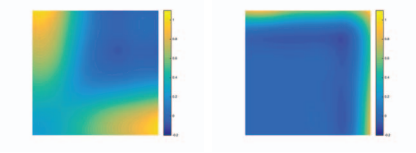


Figure 15. Best NAND, NOR for $k = 3$.

We even obtained some logic for $k = 2$, but since the NAND and NOR are already difficult to evolve for $k = 3$, it is of pure theoretical interest to study these cases. There is a lot more we could vary: different R and C values for the junctions, different sizes for the NPs, other choices for NP networks, and other functionalities. We hope to report on this in the near future.

VI. Conclusion

We have developed a simulation tool for exploring the capabilities of using evolution-in-materio to turn disordered NP networks into functional units. We have validated the code on small examples of designed NP networks that are known from literature, and we have illustrated how the code could be used to explore larger NP networks, in particular for turning $k \times k$ grids of NPs into logic gates, resembling reconfigurable assemblies of transistors. It was shown that there is a very good match between the experimentally assembled and measured NP networks and the simulated NP networks. Although the simulations indicate that the applied physical model seems to fit very well, it is not clear how accurate the simulations can predict the behaviour of larger disordered NP networks. On the other hand, given the many challenges and time-consuming production methods as well as the use of other resources, there is a clear advantage of having the simulation tools available. Moreover, the simulations are more flexible and much faster than the experiments, and could lead to fundamental insights as well as suggestions for more complex functionalities. It is clear that in the end we will need the NP networks to function as stand-alone devices. Although the simulated NP networks are scalable, they are still small compared to the NP networks that we have been able to investigate experimentally. It is a major challenge to narrow this gap, both experimentally and by optimisation and parallelisation of the simulation code. Experimentally, the main challenges for ongoing and future work are to scale down the central gap, to trap smaller particles, and to have a richer structure of electrode architectures to interface the NP networks. For the simulations, we are currently working on a parallelisation of the code to run on a GPU. For a system with N variables, theoretically we know it should result in an order $O(\log(N))$ process. Whether this is realistic and usable in practice has not been determined yet. There will be quite some overhead due to communication from and to the GPU. It might be necessary to consider the parallelisation of jumps of electrons that take place at mutually remote particles. Theoretically, these particles might interact and the jumps cannot be treated as independent events, but for the simulations we expect no major differences, if the model parameters are corrected after a number of simultaneous events. Finally, since the NP networks in [3] cannot be fabricated according to a predefined specific design, it is not possible to use an accurate physical model for such systems. In [16], we present an alternative approach. This novel approach is based on training artificial Neural Networks in order to model and investigate the NP networks.

Acknowledgment

We acknowledge financial support from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement No. 317662.

References

[1] H. Broersma, F. Gomez, J. Miller, M. Petty, and G. Tufte, "Nascence project: Nanoscale engineering for novel computation using evolution," International Journal of Unconventional Computing, vol. 8, no. 4, 2012, pp. 313–317.

[2] J. F. Miller and K. Downing, "Evolution in materio: Looking beyond the silicon box," in Proceedings of the NASA/DOD Conference on Evolvable Hardware, 2002, pp. 167–176.

[3] S. Bose, C. Lawrence, Z. Liu, K. Makarenko, R. van Damme, H. Broersma, and W. Van der Wiel, "Evolution of a designless nanoparticle network into reconfigurable boolean logic," Nature Nanotechnology, vol. 10, no. 12, September 2015, pp. 1048–1052.

[4] K. Likharev, "Single-electron devices and their applications," Proceedings of the IEEE, vol. 87, 1999, pp. 606–632.

[5] A. Korotkov, Coulomb Blockade and Digital Single-Electron Devices. Blackwell, Oxford, 1997, pp. 157–189.

[6] C. Wasshuber, Computational Single-Electronics. Springer-Verlag, 2001.

[7] ——, "Single-Electronics – How it works. How it's used. How it's simulated." in Proceedings of the International Symposium on Quality Electronic Design, 2012, pp. 502–507.

[8] N. Allec, R. Knobel, and L. Shang, "Adaptive simulation for single-electron devices," IEEE Transactions on Nanotechnology, vol. 7, no. 3, 2008, pp. 351–354.

[9] L. Nagel and D. Pederson, "Simulation program with integrated circuit emphasis," University of California, Berkeley, Memorandum ERL-M382, April 1973.

[10] C. Wasshuber, H. Kosina, and S. Selberherr, "A simulator for single-electron tunnel devices and circuits," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 16, 1997, pp. 937–944.

[11] A. A. Elabd, A.-A. T. Shalaby, and E.-S. M. El-Rabaie, "Monte carlo simulation of single electronics based on orthodox theory," in 29th National Radio Science Conference (NRSC 2012), 2012, pp. 581–591.

[12] R. H. Klunder and J. Hoekstra, "Energy conservation in a circuit with single electron tunnel junctions," in The IEEE international Symposium on Circuits and Systems, vol. I, 2001, pp. 591–594.

[13] R. J. Baxter, Exactly Solved Models in Statistical Mechanics. Academic Press, 1982.

[14] A. Korotkov, "Single-electron logic and memory devices," Int. J. Electronics, vol. 86, 1999, pp. 511–547.

[15] C. Heij, P. Hadley, and J. E. Mooij, "A single-electron inverter," Appl. Phys. Lett., vol. 78, 2001, pp. 1140–1142.

[16] K. Greff, R. van Damme, J. Koutník, H. Broersma, J. Mikhal, C. Lawrence, J. Schmidhuber, and W. van der Wiel, "Unconventional computing using evolution-in-nanomaterio: Neural networks meet nanoparticle networks," In: FUTURE COMPUTING 2016: The Eighth International Conference on Future Computational Technologies and Applications, 20-24 March 2016, Rome, Italy, 2016, pp. 15–20.