A study on emerging electronics for systems accepting soft errors

Joakim Alvbrant

Submitted to Institute of Technology at Linköping University in partial fulfilment of the requirements for the degree of teknologie licentiatexamen

Linköping University
Electrical Engineering
Integrated Circuits and Systems
SE-581 85 Linköping, Sweden

Linköping 2016
To my family and graduating members from the class of we made it\textsuperscript{1}.

\textsuperscript{1}To This Day by Shane Koyczan
Abstract

Moore’s law has until today mostly relied on shrinkage of the size of the devices in integrated circuits. However, soon the granularity of the atoms will set a limit together with increased error probability of the devices. How can Moore’s law continue in the future? To overcome the increased error rate, we need to introduce redundancy. Applying methods from biology may be a way forward, using some of the strategies that transforms an egg into a fetus, but with electronic cells.

A redundant system is less sensitive to failing components. We define electronic clay as a massive redundancy system of interchangeable and unified subsystems. We show how a mean voter, which is simpler than a majority voter, impact a redundant system and how optimization can be formalized to minimize the impact of failing subsystems. The performance at given yield can be estimated with a first order model, without the need for Monte-Carlo simulations. The methods are applied and verified on a redundant finite-impulse response filter.

The elementary circuit behavior of the memristor, “the missing circuit element”, is investigated for fundamental understanding and how it can be used in applications. Different available simulation models are presented and the linear drift model is simulated with Joglekar-Wolf and Biolek window functions. Driven by a sinusoidal current, the memristor is a frequency dependent component with a cut-off frequency. The memristor can be densely packed and used in structures that both stores and compute in the same circuit, as neurons do. Surrounding circuit has to affect (write) and react (read) to the memristor with the same two terminals.

We looked at artificial neural network for pattern recognition, but also for self organization in electronic cell array. Finally we look at wireless sensor network and how such system can adopt to the environment. This is also a massive redundant clay-like system.

Future electronic systems will be massively redundant and adaptive. Moore’s law will continue, not based on shrinking device sizes, but on cheaper, numerous, unified and interchangeable subsystems.
Acknowledgments

Many are those who have crossed my path in life. Some of them have helped me during the work of this thesis. I would like to take the opportunity to thank:

- My adviser and supervisor, Senior Lecturer J. Jacob Wikner, for his support and help during the years.
- Associate Professor Oscar Gustafsson and Professor Atila Alvandpour for giving me the opportunity for a Ph. D. position.
- Professor Mark Vesterbacka, for his support.
- My office companion Ph. D. Niklas U. Andersson for all kinds of discussions.
- Associate Professor Mikael Olofsson and Senior Lecture Martin Singull for their support with statistical derivations.
- Ph. D. Ola Leifler for his assistance with document template and hacking in \LaTeX.
- All current and former colleagues of Electronics Systems (ES) and Integrated Circuits and Systems (EKS).
- Emad Athari, for his life changing comment.
- Lars G., for his guidance.
- Dr. Armin Jalili, for his companionship in the office and help in finding words.
- Petter Källström, for our nerdy discussions about everything.
- Eya Gabriella Gelland for support and interesting discussions about life.
- Susanna von Sehlen, for joy and thoughtful insights.
- Associate Professor Mario Garrido, “Do more of the fun stuff”.
- Lena and Mikael Wigle with kids, for providing a stable port.
• Vahid Keshmiri, my bro.

• Ph. D. Christer Jansson, my wingman.

• The proof reading team: Susanne von Sehlen, Mikael and Lena Wigle, Vahid Keshmiri, Petter Källström, Vishnu Unnikrishnan, Armin Jalili, Christer Jansson, Martin Nielsen Lönö, and Carl Ingemarsson.

• My dear sister with family, for their support and joy.

• My dear father and mother, for their support through my life.
  I wouldn’t be here without you.
# Contents

Abstract ......................................................... v
Acknowledgments ............................................. vii
Contents .......................................................... ix
List of Figures ................................................... x
List of Tables ..................................................... xi
List of Acronyms ............................................... xiii

1 Introduction .................................................. 1
   1.1 Growth by continuous multiplication ....................... 1
   1.2 Order out of chaos ........................................... 3
   1.3 Electronic cells .............................................. 5
   1.4 The future of electronics .................................... 7

2 Electronic Clay .............................................. 9
   2.1 Introduction ................................................ 9
   2.2 System description ......................................... 11
   2.3 Performance prediction .................................... 17
   2.4 Finite-length impulse response filter design example .... 19
   2.5 Simulation setup .......................................... 22
   2.6 Simulation results ......................................... 23
   2.7 Discussion ................................................. 30

3 Memristors ...................................................... 33
   3.1 Introduction ................................................. 33
   3.2 Physical relationships ...................................... 35
   3.3 Memristor models .......................................... 37
   3.4 Design space ................................................ 44
   3.5 Simulation results .......................................... 48
   3.6 Applications ............................................... 53
   3.7 Conclusions ................................................. 58
4 Miscellaneous

4.1 Introduction ............................................ 61
4.2 Artificial neural networks ............................ 63
4.3 Wireless sensor networks ............................. 71
4.4 Conclusions ........................................... 78

5 Discussion ........................................... 81
List of Figures

1.1 A fertilized egg initially divides into intermediate cell types according to a predefined sequence. Some cells on the branches have a self destruct instruction which terminates that particular branch. ......................................................... 4
1.2 The cell division procedure of a fertilized egg and the imitated sequence on an electronic cell array (ECA). The point where the sperm has entered the egg determine the orientation of the cell division by gradient signaling substances. As the number of cells in the lump goes up, more types signal substances diffuses out from specific locations. In ECA the initial point can be set by applying a voltage at the desired orientation. .......................... 5

2.1 The basic lattice structure used in this chapter. The structure is based on a number of cascaded columns of functions, such as scaling and delays. Each column represents a matrix operation of the inputs from previous column.............................................. 12
2.2 Structure of redundant system. Each processing element (PE), marked by dashed boxes, contains a functional column of $F_n$. Each group of PE, forming a complete set of functional columns in $F_n$, are grouped and marked by solid line box, and then copied $x_n$ times into redundant columns. ................................. 14
2.3 With mean value voters the system becomes the original system, but with scaling factors due to the stochastic properties. This means that we can express the transfer function from $d_{ini}$ to $d_{outi}$ as the sum of all possible possible paths between input and output. ................................................................. 15
2.4 A finite impulse response filter implemented in the basic lattice structure. The input signal is $x[n]$, and the output signal is $y[n]$. The other input is tied to zero and the second output left open. 19
2.5 A general low-pass filter specification. The passband is specified from zero up to the normalized frequency, \( \omega_p T \), together with the maximum allowed deviation, \( \delta_p \), from the desired gain. The stopband is specified from the normalized frequency \( \omega_s T \) up to the sampling frequency. The maximum allowed gain in the stopband is given by the parameter \( \delta_s \). The relative performance measurement \( A_{att} \) is defined as ratio of the minimum gain in the passband and the maximum gain in the stopband.

2.6 Comparison of the filter coefficients when optimization is done without (left) and with (right) concern of the degree of redundancy. In this example the error probability is \( \epsilon = 1.0 \cdot 10^{-2} \) and the redundancy is \( X = 22 \). The filter coefficients (top) are in general slightly increased in amplitude, while the degree of redundancy (bottom) changes from a uniform distribution to a non-uniform distribution.

2.7 The yield of NOM, COPT, and CXOPT design procedures for \( X = 22 \) and \( \epsilon = 0.01 \). The optimization reduces the amount of high performance in favor of better yield at low performance, as illustrated by the arrows. The superfluous performance can be traded for increased yield at lower performance.

2.8 Performance of CXOPT procedure in Monte-Carlo simulation with the yield of \( \Phi = 0.99 \). With low error probability and a high degree of redundancy (lower left corner) the performance is good. Increasing the error probability, or reducing the degree of redundancy, will reduce the performance. Eventually the system breaks down (top right) with too high error probability and too low degree of redundancy.

2.9 A scaled version of Fig. 2.8, for comparison with Fig. 2.10.

2.10 The predicted performance of the CXOPT design case at the yield of \( \Phi = 0.99 \). The predictor gives in this case a general overestimation of the performance. However, at high error probability and low degree of redundancy (top right), it is underestimating the performance.

2.11 Yield loss of the SIMP case. The loss is expressed as \( \Delta \beta = \log_{10} (1 - \Phi) \), i.e., the power of the error probability of the hole system. In principle, the SIMP case fails for all error probabilities of PEs above \( 2.2 \cdot 10^{-2} \). Below that, the degree of redundancy influence as expected, low loss at high redundancy and vice versa. Due to numerical limitations the results in the figure are limited to \(-14\).

2.12 Comparison of yield loss between CXOPT case at \( A_{att} = 25 \, dB \) with SIMP case, expressed as \( \Delta \beta_{\text{CXOPT}} - \Delta \beta_{\text{SIMP}} \), where \( \Delta \beta = \log_{10} (1 - \Phi) \). Negative number means CXOPT case have less yield loss than SIMP case.
3.1 The pair-vice combinations of current, voltage, flux, and charge generates six relationships. The relationship which Chua pointed out are the memristor, which links charge with flux and has the unit of Ohm. The primary behavior of a memristor is that the resistance changes and the value can be remembered. The symbol could be interpreted as resistor in a memory box, the direction dependency is marked by the black bar.

3.2 The number of papers containing the word “memristor” according to Google Scholar. After Chua’s paper in 1971, the number of articles stayed on a handful each year. In the late 90’s the number started to increase and completely explode with HP Labs paper in 2008.

3.3 The linear memristor model by HP Labs. The memristor is modeled as two regions of doped and non-doped material. As current floats through the device, the cross section between the regions moves (marked with distance $d$). As a result, the linear combination of resistance, $R_{on}$ and $R_{off}$ changes and the memristor changes its resistance.

3.4 The circuit equivalent of the linear memristor model. The regions of doped and non-doped material are modeled as two variable linked resistors in series.

3.5 The Simmons tunneling barrier macro model of the memristor. Between two platinum plates there are two regions, a non-doped $\text{TiO}_2$ and a doped $\text{TiO}_2$. The doped region causes a series resistance, while the non-doped causes a barrier of length $w$, in which the electrons tunnel through.

3.6 Some Joglekar-Wolf window functions. By reducing the speed of the state close to the boundaries, the state $x$ is kept within the region $[0, 1]$.

3.7 Illustrative figure of the behavior of a Biolek window. Depending on the direction of the current trough the memristor, the window jumps between the two curves.

3.8 By adding an additional brick wall function to the window function, there is a safe region the simulator has to encounter. Thus can the non-stable region be avoided.

3.9 Circuit set up for amplitude characteristic. A memristor is driven by a sinusoidal current source with amplitude $I$ and frequency $f$.

3.10 **Dashed line:** Calculated amplitude characteristic of a clipped sinus inversely proportional to frequency. At $f_0$, the amplitude is exactly 0.5. **Solid line:** Behavior of the clipped sinus versus frequency.

3.11 Please see next page for caption.
3.11 Plots from simulation of the linear memristor model at five different frequencies. A sinusoidal current source with amplitude 10 $\mu$A drives a memristor and the current is plotted against the voltage across the memristor.

**Upper left ($\circ$):** $f_0 = 1$ kHz; the state of the memristor toggles between the two borders. Only a small fraction of the period is spent in transition.

**Upper middle ($\times$):** $f_0 = 12.6$ kHz; Most of the period is spent at the borders. However, the fraction spent in transition has increased.

**Upper right ($\Delta$):** $f_0 = 178$ kHz; The state makes complete transition between the borders during one period. Only a small fraction of the period, if any, is spent in pending at the borders.

**Lower left ($\ast$):** $f_0 = 316$ kHz; The state only moves between the initial condition and one border and the wings of the butterfly shrink.

**Lower middle ($+$):** $f_0 = 5.01$ MHz; Very small change in the state during one period. The memristor behaves like a resistor.

**Lower right:** The amplitude characteristic of the state versus frequency for the initial condition $x_0 = 0.5$. Corresponding points of the I-V plots marked with corresponding markers.

3.12 Amplitude characteristic of a memristor with a Joglekar-Wolf window function. The initial state values are $x_0 \in [0.01, 0.99]$. A secondary fold appears due to pending of the state sustained by the window function, but the mapping between charge and state is conserved.

3.13 Amplitude characteristic of a memristor with Biolek window function. The initial state values are $x_0 \in [0.01, 0.99]$. The surface is smooth and has no secondary folding other than the expected roll-off, only a small arc can be seen versus $x_0$. However the one-to-one mapping of charge and state is lost.

3.14 The crossbar structure of memristors in a two by two array. The selected memristor is addressed by applying a voltage (or current) between one column metal wire and one row metal wire. The current (or voltage) relates to the addressed memristors state, which encodes the information. In these applications the number of rows and columns are large.

3.15 The Hodgkin-Huxley model of the cell membrane. The capacitor models the charges on each side of the cell membrane. The branch of $E_{Na}$ and $R_{Na}$ models the potential of the sodium ions and their dedicated passage channels through the membrane. Similar branch is for the Potassium ions ($K$). The final branch models the leakage through membrane.
3.16 Solving a maze by using memristors. A network of memristors are connected by switches. The maze, illustrated by the wide light blue lines is defined by the position of the switches. An open switch indicates a wall and a closed indicates a possible path.

4.1 Basic topology and functions of a neuron. Input signals to the neuron are applied at the dendrites, where a chemical transfer signal (neurotransmitter) is converted to an electrical potential. All input potentials are summarized over the soma. If the potential at the axon hillock is above a threshold, this causes a wave of potential (also known as an action potential) to propagate over the axon. Finally the propagating action potential reaches the axon terminals, where the electrical potential is converted to chemical transmitters to dendrites of other neurons or cells. The axon may be warped by Schwann cells, which causes the action potential to jump between the nodes of Ranvier and thus propagates faster and more energy efficient.

4.2 The structure of a restricted Boltzmann machine. The input data is presented at the visible nodes $v_0: v_n$, which are connected to the hidden nodes $h_0: h_m$ by the connection matrix $W$.

4.3 The Boltzmann machine operates by using the input data to set the visible nodes ($v^1$), and then generates the state of the hidden nodes ($h^1$), which in turn generate new visible nodes ($v^2$) and so on. The movement of data from the visible nodes to the hidden nodes and back again is called a shake and is done a predefined number of steps. At the end, the final shake presents the result from the network, the belief.

4.4 The training, test, and reconstructed data of a restricted Boltzmann machine. Top: The training data, each pixel column represents one vector. Middle: The test data to be reconstructed. Bottom: The resulting reconstructed data according to test data and the knowledge stored in the network.

4.5 The original image which the neural network is trained on. In addition to the image data, two sets of potentials, depending on the pixel location, are used to form the input vector.

4.6 A reconstructed image from the network. Given the two sets of potentials, in x-direction and y-direction, each pixel is reconstructed by the restricted Boltzmann machine to complete a image similar to the original in Fig. 4.5.

4.7 In the learning of the Markov model, a piece is selected from the data and the occurrence of that particular symbol combination is learned. Then, another piece is selected and learned and so on.

4.8 The Markov model can be seen as a structural tree, where the branching is done on previous symbols in the piece. The leaves contains the probabilities of the different symbols.
4.9 Generating output data from the Markov model. The last symbols point out a subtree of probabilities for the next symbol and a new symbol is generated, and the piece index is incremented to generate another symbol.

4.10 A small example of a wireless sensor network with ten motes and one base station.

4.11 A system block diagram of a wireless sensor network (WSN) mote. A mote must have at least one function block in each category including: energy (green), communication (blue), measurement (beige-orange), and control (gray).

4.12 A small wireless sensor network with two base stations. The motes can have a potential to each base station and thus some spatial information can be deduced.

4.13 Map location of the base station and the hostile jamming source in the field of motes.

4.14 Left: Organization example of a grid of motes. Each pixel represents a mote and at the center left the sink is located. Right: Color encoding of potentials, from lowest (black) to highest (white).

4.15 Setup to calculate Friis' equations for the signals mote to mote and jamming source to mote.

4.16 Left: The network of motes is jammed by a hostile signal. The jamming source can only disable motes within the region where it can interfere enough with the signals of neighboring motes (the white circle). The rest of the network realign itself. Right: Color encoding of potentials, from lowest (black) to highest (white).
List of Tables

2.1 All possible combination of second moments in the sub-terms of transfer functions and their expectation values. The variables $a_1$, $a_2$, $m_1$, and $m_2$ are assumed to be non equal between themselves unless otherwise specified. The support function $C_n(p)$ is defined in (2.14). .................................................... 18

2.2 The optimization and implementation procedure of the three design cases. .......................................................... 24

2.3 Yield and performance improvements of COPT and CXOPT case compared to the NOM case. The results of $X = 3$ and $X = 5$ have been deemed as outliers. ................................................. 27

2.4 The $R^2$ model fit measurement of the performance prediction, $\hat{A}_{att, min}$, validated on Monte-Carlo result. The ratio of non-outliers data points in parenthesis. ................................. 28

4.1 The training data for a restricted Boltzmann machine in the simple vector recognition example. ................................. 66
## List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>AADD</td>
<td>aware adaptive directed diffusion</td>
</tr>
<tr>
<td>ADC</td>
<td>analog to digital converter</td>
</tr>
<tr>
<td>ANN</td>
<td>artificial neural network</td>
</tr>
<tr>
<td>CMOS</td>
<td>complementary metal oxide semiconductor</td>
</tr>
<tr>
<td>CPU</td>
<td>central processing unit</td>
</tr>
<tr>
<td>DAC</td>
<td>digital to analog converter</td>
</tr>
<tr>
<td>DC</td>
<td>direct current</td>
</tr>
<tr>
<td>DME</td>
<td>distributed modular equipment</td>
</tr>
<tr>
<td>DNA</td>
<td>deoxyribonucleic acid</td>
</tr>
<tr>
<td>DRAM</td>
<td>dynamic random access memory</td>
</tr>
<tr>
<td>ECA</td>
<td>electronic cell array</td>
</tr>
<tr>
<td>FFT</td>
<td>fast Fourier transform</td>
</tr>
<tr>
<td>FIR</td>
<td>finite-length impulse response</td>
</tr>
<tr>
<td>FPGA</td>
<td>field programable gate array</td>
</tr>
<tr>
<td>GPU</td>
<td>graphical processing unit</td>
</tr>
<tr>
<td>IC</td>
<td>integrated circuit</td>
</tr>
<tr>
<td>PE</td>
<td>processing element</td>
</tr>
<tr>
<td>RBM</td>
<td>restricted Boltzmann machine</td>
</tr>
<tr>
<td>RMS</td>
<td>root mean square</td>
</tr>
<tr>
<td>RNA</td>
<td>ribonucleic acid</td>
</tr>
<tr>
<td>SIMD</td>
<td>single instruction multiple data</td>
</tr>
<tr>
<td>SON</td>
<td>self organizing network</td>
</tr>
<tr>
<td>TEAM</td>
<td>threshold adaptive memristor</td>
</tr>
<tr>
<td>WSN</td>
<td>wireless sensor network</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

The question of the origin of life is a deep and delicate one. We are not going to answer it here. However, we will touch on the subject slightly, by discussing how an egg can become a chicken, by touching just a fraction of all mechanisms in the transformation, for any other animal, plant, or fungus. We are going to use some tools from the field of biology and use those in the field of electronics.

The technological development today does seem to be going in a rapturous speed, and also, seems to be increasing faster and faster. This is true, but the main cause is that the human brain has a problem to comprehend with large time spans, larger than our own life time. We simply do not understand eras longer than one life span. For instance, consider that agriculture first emerges only some ten thousand years ago. The introduction of the plow, about thousand years ago, increased production when animals could be used to cultivate the earth. And, about one hundred and fifty years ago, the first steam tractor increased the production further. Since then the number of people involved in food production has dropped significantly and released them to work with other tasks. Today, in most countries, only a few percent of the population is involved in food production.

1.1 Growth by continuous multiplication

The interest you get each year in a bank account is proportional to your savings. Normally, the interest you get from the bank is added to the savings, which generates even more interest next year. We may express the amount on the saving account as

$$m_t = m_0(1 + r)^t, \quad (1.1)$$

where $m_0$ is the initial amount, $r$ is the interest and $t$ is the number of years since the account was opened. The growth of money is a result of annual continuous multiplication. The general form of (1.1) is the exponential law

$$y(x) = \beta \phi^{x/\sigma},$$

where $\beta$, $\phi$ and $\sigma$ are constants from the application the law describes.

As seen with the agriculture example in the previous section, the time span seems to decrease with each innovation step. Agriculture seems to fol-
low an exponential law, at least historically. The question for the future is, if the fraction of the population in food production will continue to decrease. The number of farmers does seem to decrease while the size of the farms increase.

In 1938 Benford [1] presented a study on numerical quantities in about twenty different fields. He found that a histogram of the first digit, in any of the fields, follows a distribution of $P(d) = \log_{10}(1 + 1/d)$, which is approximately an exponential law. For instance, frauds in financial accounting can be detected, or at least raise suspicion, if there are deviations in the distribution of the first digit compared to Benford’s law. Fabricated numbers do not tend to follow Benford’s law. [2]

The Fibonacci numbers (developed to predict the reproduction of rabbits) also follows an exponential law. The definition of Fibonacci numbers $f_1 = f_2 = 1, f_n = f_{n-1} + f_{n-2}, n > 2$ gives somewhat of a hint of what is going on, but maybe not too obvious. If we instead look at the ratio of two Fibonacci numbers, we see that $f_{n+1}/f_n \to (\sqrt{5} + 1)/2 = \Phi, n \to \infty$ (which is the golden ratio). The next number is, not an addition of previous result, but rather a multiplication. Consider the agriculture example again, a fraction of the last harvest is going to be the seed for next year’s production. The effectiveness of the crop depends on the multiplication factor, which must have a “unity grain” higher than one, for that particular crop.

Another exponential law is Moore’s law, which states that the amount of functionality in a integrated circuit (IC) is doubled every 18 months for the same price. Actually, when Moore did his observation, about six years after the first IC by Kilby [3], He said the time period was every twelve months, but this time span has increased the last decade [4]. The importance is, that Moore’s law still continues, but for how long, and what will come after?

Some of the future trends are already visible today. The first is the halt of the hunt for even higher clock frequencies in the central processing unit (CPU). The author’s first computer ticked away at seven MHz, while today’s computers are clocked in speeds about five hundred times faster.

Beside this, today’s CPUs execute each commands in fewer clock cycles, and to push the performance even further, processes several commands in parallel in each core and have multiple cores. Today a common cell phone contains two to three CPUs where some of them are multi-core [5, 6].

The trend of multi-core structures is pushed even further with the graphical processing unit (GPU) where a large amount of parallel cores execute single instruction multiple data (SIMD) commands to process the data to calculate the pixels on the screen. However, not all of the tasks we want to solve with a computer system are of the type calculate-pixels-on-screen problem, which are easily to parallelize, many tasks rely on close previous results and this may be a major hurdle for further increase of performance.

The next step of parallelized systems is today’s super computer, which simply is a large pack of smaller computers with a very fast back bone network between them. At the writing of this thesis the current largest super computer is the Chinese Tianhe-2 with 3.120.000 cores running at 2.2 GHz.
producing a theoretical performance of 54 peta-flops at the cost of a power consumption of 17 mega Watts\(^1\).

The parallel power of the human brain is rather amazing. Let us consider the problem of separating pictures of dogs and cats. Simple, right? A cat has four pawns, and a dog has... wait. A cat has a nose and a tail, and a dog has a nose and a tail... The task of separating cats and dogs is not an easy task for a computer, but a kid can do it easily. To really show the difference, a computer runs at several giga-Hertz, while the human brain runs at about one hundred Hertz and only need about one tenth of a second to do the task. That is, the human brain can do the computation needed with about ten cycles, while a computer needs millions to even get close [7].

The reduction of the price of hardware has today gone so far that you actually can get a rather decent computer attached to the cover of a magazine\(^2\). Current retail price\(^3\) for a Raspberry Pi Zero is about $5, which implies a manufacturing cost less than that.

At the same time, the software market still grows. The introduction of applications or apps, has caused today’s cellular phone customers to be more focused on the experience of the device, the hardware underneath is since long of second or next to none in priority, it just has to work. The problem is, reliability will soon not be obvious, due to the finite size of atoms.

### 1.2 Order out of chaos

Chaos is all around us, but with a kind of order. Looking at the big scale, the structure of the universe is by its clusters and super clusters forming a homogeneous porridge. The same does the Milky way. Looking at the scale of our solar system, there seems to be a more order system by considering the organization of the planets. Our own planet, Tellus, does have a regular structure of water and land masses. Looking at even smaller scale, the geology of rocks and minerals show once again a kind of chaotic order. At the same scale clay, which we use to form e.g. pots, “looks” like a homogeneous substance. But looking at a smaller scale, visible by the naked eye, the different grains of sand can be observed. Furthermore, each grain in the pot can be interchanged with any other grain and the pot is still a pot. In fact, we may remove some fraction of the grains and we will still have a pot, as long the grains are removed at random. If you remove too many, a hole may occur and we lose the function of a pot, to keep things contained. A random removal of grains may however weaken the pot’s structure such that evaporation occurs faster and the pot preform less well in retaining humidity or liquid. What if, we could build electronics in the same way as clay builds up a pot or a brick. Imagine that every grain of sand in the pot is some kind of electronic building block. How should such a system be constructed? 

\(^{1}\)www.top500.org, accessed December 2015.
\(^{3}\)www.m.nu/dator-raspberry-zero-p-2506.html, accessed March 2016
higher degree of order has to be brought to the electronic system compared to the pot example. We need to bring order out of the chaos.

The biology has solved the issue of bringing order out of chaos. Or actually as Richard Dawkins states, stable constructions remain over time [8]. Have you ever wondered how an egg, which is a huge, single cell, can turn into a fetus? How does the egg know how to divide itself into different cells, and when? The answer to that is twofold. First, the entry point of the sperm serves as reference for diffusion of a signal substance in the egg. This is the coordinate system for further development and other substances to be diffused [9]. The second part is the state machine in the cell divisions, the first division generates the cells AB and P1, which are of an intermediate type of cells which only purpose is to generate the four next coming cells ABa, ABp, EMS, and P2, as seen in Fig. 1.1.

At the same time, more layers of signals substances are diffused from specific cells, increasing the dimension and precision of the coordinate system of the lump of cells. The spatial concentration of different signal substances together with the states within each cell, determines each cell’s development path [10].

We can use the strategy adopted by, the fertilized egg to organize and bring order in an electronic system.

1.3 Electronic cells

Now, consider an electronic cell array (ECA). In this constitution we have a fixed set of cells and now splitting, since they are fabricated all together in the same chip, can only be done by sub grouping of already existing cells.
1.3. Electronic cells

Figure 1.2: The cell division procedure of a fertilized egg and the imitated sequence on an ECA. The point where the sperm has entered the egg determine the orientation of the cell division by gradient signaling substances. As the number of cells in the lump goes up, more types signal substances diffuses out from specific locations. In ECA the initial point can be set by applying a voltage at the desired orientation.

We may connect several chips in a larger array, but the argument still holds, the set is fixed. With this fixed set we may now apply our knowledge of how the biological cells organize themselves.

Let us apply some kind of signal at one edge, the simplest one would be a constant current or voltage, which would be allowed to diffuse over a resistive network among the cells, as seen in Fig. 1.2. At the start, all cells are the same but depending on which potential they sense they choose to become cells A or B. This is the first step in dividing the ECA into two types.
1. INTRODUCTION

The division can then continue to the next step into further specialization. Remember that each specialized cell can now send out other signals on either the same network or another parallel network to refine the resolution of positions in the array. The refinement of potentials gives the possibility to further specialization of cells, which in turn gives greater refinement of the potentials. This process continues until the target configuration is reached. With a lot of spare cells, organization of the system can take faulty cells into account during the formation.

The faulty cells may however inflict the self organization operation. This may be avoided by introducing majority decision of the neighbors to “kill” or isolate the faulty cell, or we may rely on that the larger number of cells are behaving correctly as seen in chapter 2.

Since the programmer of such an ECA does not exactly know where the specific function is going to be. Physical on the chip for each instance of the device, there must be a way of opportunistically find a path for signals between block. In chapter 3 we take a look on the possibility to use the memristor as a maze solver between different regions in the ECA.

Another clay like application is artificial neural network (ANN), where a bunch of nodes (or neurons) are implemented either by software entities or as dedicated hardware. When the ANN is trained to learn specific features of the data, we do not know exactly which node’s connection that contains the specific data. The organization occurs opportunistically due to the initial randomization of the network and the data presented. We may figure it out by looking on the network, but there is no real way, unless we have precise knowledge of the initial state.

A WSN can also be considered as a clay like system. The network consists of small nodes which communicate wirelessly with other nodes. However, the major idea with WSN is the abundance of nodes and no particular order is applied when setting it up. You simply sprinkle an area of interest with nodes and they self organize. Once again, the key features are redundancy and opportunistic configuration.

Finally, some disassociated words about the concept of deoxyribonucleic acid (DNA) being the origin of life. It is probably not. Most indications point in the direction that ribonucleic acid (RNA) is the origin of life, due to its ability to both code information and operate upon other molecules [11]. Ribosomes, for instance, are built up by RNA [9]. However, DNA does store the code of life and somewhere along the evolution the information has been transferred, or more likely translated, over to the more stable DNA. The cell’s core, which contains the DNA, is just like any other organelles (cell organs) in the cell, it needs the support from other parts of the cell to operate. Only DNA in it self, is not life.
1.4 The future of electronics

So, how will the future of electronics be? Well, the guess would be that Moore’s law will continue to be valid for a while, making electronics a real commodity produced by a small fraction of the population, in the same way agriculture went. The continuation will be something completely else such as quantum computers, light based calculation or even biological cells processing information. [12]

Meanwhile we may make use of ECA, ANN and WSN at the cost of that we may have to relax the requirement on precision, for instance that $1 + 1$ is no longer $= 2$ but rather $\approx 2$, but this may be acceptable since even today’s computer does have truncation errors due to the limited number of bits in the representation.

Finally, we may have a moral dilemma. The systems may be on the border of consciousness as defined by [13]. In that case, do we have the right to switch the power off?
Chapter 2  
Electronic Clay

Assume that Moore’s law will continue to be valid also in the future, but at the cost of poor reliability. The components will be very small, like the fine grains in clay, but error-prone. Electronic clay are massive redundant systems, which are tolerant to loss of a fraction of the building blocks. The blocks should be interchangeable with each other. In ordinary clay we may lose or interchange some grains without any notice. The same thing should hold for electronic clay.

2.1 Introduction

Moore’s law (approximately) states that the amount of functionality and performance is doubled every 18th month for the same cost [14]. This trend is mainly, but not completely, dictated by the shrinking of the minimum-size transistor dimensions, we get more transistors per area and hence the price per transistor is lowered. Scaling of complementary metal oxide semiconductor (CMOS) geometries will indeed increase the complexity of systems since we can integrate more components. Typically, also the transistors become faster and consume less power individually. The accuracy is however affected. For example, the intrinsic gain of the transistor decreases and the matching properties that are inversely dependent on the transistor area gets worse. The reliability and yield of our IC decreases.

With such small devices, the electrical behavior will in the end be defined by a few layers of atoms in the crystal lattice. With so few atoms, even a handful of extra or less atoms may cause a great change in the circuit behavior. With smaller transistor dimensions, the deviation of the threshold voltage increases [15–17], but at the same time the supply voltage decreases. This can reduce the signal headroom to zero or below, causing the circuit to be in a locked position - on or off.

This means we can no longer talk about deterministic design, but rather stochastic designs, which Rabaey refers as “The Age of Randomness” [18]. As a consequence, the perspective of “always precisely right”, may need to be changed to the point of “close enough” or “good enough”. We may have to remind ourselves that Moore’s law is a financial law and not a physical
2. **Electronic Clay**

law, and hence we may have to sacrifice some area on securing performance and rely on the effects of mass production for future development.

The classical way to make a system more resilient is to introduce redundancy as von Neumann suggests [19] already in 1956. This is today used in very sensitive systems, such as primary flight computers for air and space aviation [20, 21] but also in transistor- and gate-level logic [22]. Furthermore, the trend in today’s research in avionics industry is to utilize distributed modular equipment (DME), which incorporates many functions on the same computational subsystem but may also implement the same function on different subsystems such as in the Scarlett-project [23] and the succeeding Ashley-project [24]. A soft real-time system may also be seen as a system with error-prone subsystems, where the ratio of missed deadlines essentially is the error probability.

From nature we get inspiration of how a redundant system can be built. Most likely was von Neumann inspired by nature. The brain consists of many parallel executing elements. These elements are unified (neurons), and are thus essentially identical and can be interchanged\(^1\). However, the connection between neurons differs, especially the strength of the connections [25, 26]. The unified elements are comparatively simple, or at least perform a reduced set of operations and this unification caters for redundancy in an easy way. Sensory inputs are redundantly acquired and processed. A touch is built up by several sensitive neural-ends and signals are processed in parallel in neocortex. If an action is required, a set of parallel neural signals will return to the muscles and they will in turn, in principle, average the neuron activity into strength of contraction [25].

There is a complexity in defining the correct operation, and the system itself might not know which subsystem(s) to rely on. A normal redundant system usually rely on majority voters, but with a high degree of redundancy the majority voting becomes complex and impractical, since the complexity increase drastically with the number of inputs; all must be compared to each and other. Attempt has been done to reduce the complexity [27]. In this work, instead of making complex judgment of working or non-working subsystems, we rely on the use of redundancy itself, and combine the results from all subsystems and weigh them equally together. This means that the massively redundant set of signals will make each signal less important and local “errors” will have less importance.

With these examples in mind, and the insight of Rabaey [18], a mean voter, taking the average of all votes, could perhaps be a good candidate when constructing systems, rather than the majority voter.

**Purpose**

The purpose of the work is to assess how massive redundant system, or electronic clay, can be useful for signal processing.

---

\(^1\) As long they are of same type.
2.2. System description

Demarcation

We study a redundant, regular, discrete-time, signal-processing system. In order to cater for errors in processing elements we add redundant paths to a certain order. We use a mean-value voter that takes the average of all the results for further processing. For the sake of simplicity, we are only looking on feed forward systems, that is, only functions of weighted sums of the inputs, and not any feedback system.

To evaluate the suggested system we use a finite-length impulse response (FIR) filter. Note that we in this work do not propose an FIR filter design guide. Instead, we use the filter as a reference to a linear system, with typical, unified processing elements, PEs, to illustrate our proposed methodology.

Chapter overview

We elaborate on a general structure of a mean value voter system. From a given error probability and with this structure we can formulate an expression for the expected deviation from the specification. We can then use this expression to optimize the system with respect to design coefficients and degree of redundancy to obtain yield. Furthermore, we present an expression for the expected overall performance for the system. Finally the methodology is applied to a design example and the result is evaluated.

Background

The field of biologically inspired electronics is wide. To start somewhere we chose a weighted sums system, similar to a neuron. We also decided to go for a general and simple Bernoulli error model to simulate and analyze the fundamental behavior. A Bernoulli variable is either zero or one.

2.2 System description

A number of PEs forms a lattice structure as shown in Fig. 2.1 with $M+1$ inputs$^2$ and outputs ($d_{\text{in}}$ and $d_{\text{out}}$). Each PE, outlined in dashed box, consists of arbitrary linear functions $f_{i,j,n}$ including scaling and delays. The system can be described by functional matrices as

$$H_F = F_N F_{N-1} \ldots F_0,$$

where

$$F_N = \begin{bmatrix} f_{0,0,n} & f_{0,1,n} & \cdots & f_{0,M,n} \\ f_{1,0,n} & f_{1,1,n} & \cdots & f_{1,M,n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{M,0,n} & f_{M,1,n} & \cdots & f_{M,M,n} \end{bmatrix}.$$  

$^2$0, 1, ..., $M$ is $M + 1$ elements.
Figure 2.1: The basic lattice structure used in this chapter. The structure is based on a number of cascaded columns of functions, such as scaling and delays. Each column represents a matrix operation of the inputs from previous column.

Note that each column of (2.2) represents one PE, i.e., a functional column. Each stage, \( F_n \), of (2.1) can be made redundant by using \( x_n \) copies in parallel. This is illustrated in Fig. 2.2 where a number of copies has been introduced in the structure. (No details are shown in the copies.) This means we have both functional columns and redundancy columns (assumed if not specified).

The outputs from each column are processed by voter circuits (illustrated by the diamond-shape operator in Fig. 2.2) in each PE of the next column. This construction also makes the voting redundant. Traditionally, the voter circuits implement majority-decisions on a bit-level, as described in [21]. We instead use a mean-value voter as also suggested by [28, 29], such that all inputs are weighted equally and the voters are easy to implement. The major drawback is the loss of error masking and we cannot detect the "traitors" in the Byzantine generals problem [30]. Faulty information is simply embraced by the average.
2.2. System description

The output of the system is a set of signals from the PEs in the last column. These are gathered in a termination node, which is the same kind of voter as in the rest of the system, but assumed to be ideal for evaluation purposes. Similarly the input from each \( d_{inj} \) may either be a single line input, or a vector of inputs from previous subsystems in the signal chain.

### Probability of error

To each PE, a Bernoulli distributed variable is associated. A one, 1, implies correct operation and a zero, 0, denotes incorrect operation. We simply assume that the PEs either work or do not work (producing a constant 0). The probabilities of error are assumed to be independent and identically distributed for all copies within a column, that is

\[
\Pr(R(m, n) = 1) = p_{m,n} = 1 - \epsilon_{m,n},
\]

where \( \epsilon_{m,n} \) are the error probabilities for each PE, \( m \) is the functional column and \( n \) is the redundant column of PE in Fig. 2.2.

### Impact on the system when using mean-value voters

With mean-value voters together with the fact that all PEs in each column process the same data from the previous column, we can rearrange and collect all the redundant terms and redraw Fig. 2.2 into Fig. 2.3. This structure is similar to the original but with new scaling factors depending on the number of working PEs. The scaling factors are given by the functions

\[
\gamma_{m_1,m_2,n} = \frac{1}{\xi_n} \sum_{k=1}^{\xi_n} R(m_1, k, n) R(m_2, k, n).
\]

The scaling factors can be formed in a matrix as

\[
G_n = \begin{bmatrix}
\gamma_{0,0,n} & \gamma_{0,1,n} & \cdots & \gamma_{0,M,n} \\
\gamma_{1,0,n} & \gamma_{1,1,n} & \cdots & \gamma_{1,M,n} \\
\vdots & \vdots & \ddots & \vdots \\
\gamma_{M,0,n} & \gamma_{M,1,n} & \cdots & \gamma_{M,M,n}
\end{bmatrix}.
\]

The system is still linear, and superposition is applicable, which means we can write the complete redundant system as

\[
H = (F_N \odot G_N) (F_{N-1} \odot G_{N-1}) \cdots (F_0 \odot G_0),
\]

where \( \odot \) is the element-wise multiplication.

### Target optimization function

The goal of the optimization is to reduce the excessive performance in favor of a higher yield and make sure we do not introduce too much redundancy if
Figure 2.2: Structure of redundant system. Each PE, marked by dashed boxes, contains a functional column of $F_n$. Each group of PE, forming a complete set of functional columns in $F_n$, are grouped and marked by solid line box, and then copied $x_n$ times into redundant columns.
2.2. System description

Figure 2.3: With mean value voters the system becomes the original system, but with scaling factors due to the stochastic properties. This means that we can express the transfer function from $d_{\text{inj}}$ to $d_{\text{outi}}$ as the sum of all possible possible paths between input and output.
not needed. We could apply evolutionary or swarm algorithms to optimize the system, such as [31–34], but if a deterministic target function can be formulated, the optimization procedure will most likely be more computational efficient, since no stochastic process is used.

Given that (2.1) is able to describe, at least approximately, the desired system behavior between input to output, \( D_{ij} \), we can form a target function as the maximum weighted expected square deviation from the specification

\[
\min_{\bar{x}, \bar{f}} \max_{\vec{\omega}} \sum_{i,j} E \left[ (H_{ij} - D_{ij})^2 \right] W_{i,j},
\]

(2.7)

where \( \bar{x} \) and \( \bar{f} \) are the design parameters sets. All the independent variables to be considered during optimization are contained in \( \vec{\omega} \). To emphasize and/or equalize the optimization, the functions \( W_{ij} \) are used to adjust the weight in the optimization. The target function is convex and we want to find the minimum value of the largest value that components in \( \vec{\omega} \) can produce. The terms in (2.7) can be expressed as

\[
E \left[ (H_{ij} - D_{ij})^2 \right] = E \left[ H_{ij} H_{ij}^* \right] + D_{ij} D_{ij}^*
\]

(2.8)

where * denotes the complex conjugate.

The transfer function from one input to an output is the sum of all possible paths between them and can hence be written as

\[
H_{ij} = \sum_{m_{i,j}=0}^{M-1} \sum_{m_{i,j}=0}^{M-1} \ldots \sum_{m_{i,j}=0}^{M-1} (f \circ g)_{i,m_{i,j},N} \left( \prod_{n=2}^{N} (f \circ g)_{m_{i,j},m_{i,j-1,n}} \right) (f \circ g)_{m_{i,j},0,n},
\]

(2.9)

where \( (f \circ g)_{i,j,n} = f_{i,j,n} g_{i,j,n} \). For example, a small system of \( M = 2 \) and \( N = 2 \), the transfer function from \( d_{i,0} \) to output \( d_{i,1} \) can be written as

\[
H_{1,0} = (f \circ g)_{1,0,2} (f \circ g)_{0,0,1} (f \circ g)_{0,0,0} + (f \circ g)_{1,0,2} (f \circ g)_{0,1,1} (f \circ g)_{1,0,0} + (f \circ g)_{1,1,2} (f \circ g)_{1,0,1} (f \circ g)_{0,0,0} + (f \circ g)_{1,1,2} (f \circ g)_{1,1,1} (f \circ g)_{1,0,0}.
\]

(2.10)

The expected value of the total transfer function \( H_{i,j} \) from input \( j \) to output \( i \) becomes

\[
E \left[ H_{i,j} \right] = \sum_{m_{i,j}=0}^{M} \sum_{m_{i,j}=0}^{M} \ldots \sum_{m_{i,j}=0}^{M} f_{i,m_{i,j},N} E \left[ g_{i,m_{i,j},N} \right] \left( \prod_{n=2}^{N} f_{m_{i,j},m_{i,j-1,n}} E \left[ g_{m_{i,j},m_{i,j-1,n}} \right] \right) f_{m_{i,j},0} E \left[ g_{m_{i,j},0} \right],
\]

(2.11)
2.3. Performance prediction

where

\[
E[g_{m_1,m_2,n}] = \begin{cases} 
p_{m_1,n}, & m_1 = m_2 
p_{m_1,n}p_{m_2,n}, & m_1 \neq m_2.
\end{cases}
\] (2.12)

In a similar way, the expected value of the second order term can be expressed by

\[
E[H_{i,j}H_{i,j}^*] = \sum_{m_N=0}^{M} \sum_{m_{N-1}=0}^{M} \cdots \sum_{m_1=0}^{M} \sum_{a_N=0}^{M} \sum_{a_{N-1}=0}^{M} \cdots \sum_{a_1=0}^{M} f_i,m_{N}\, f_i,a_{N} \, f_{m_1,0} \, f_{a_1,0} \\
\left( \prod_{n=2}^{N} f_{m_n,m_{n-1},n} \, f_{a_n,a_{n-1},n} \right) \\
E[S_{i,m_{N},N} S_{i,a_{N},N}] \, E[g_{m_1,0} \, g_{a_1,0}] \\
\prod_{n=2}^{N} E[g_{m_n,m_{n-1},n} g_{a_n,a_{n-1},n}],
\] (2.13)

where the expected values of the sub-terms \( E[g_{m_1,m_{N-1},0} g_{a_1,a_{N-1},0}] \), under different conditions on the index variables, are found in Table 2.1. The derivation has been omitted. The function

\[
C_n(p) = p^2 + p(1-p)\, x_n,
\] (2.14)

should be interpreted as the squared first moment plus the variance, which carries information about the degree of redundancy, i.e. the term \( x_n \).

Finally, we can assemble all parts of (2.7) to get an expression that is swift and easy to calculate during the optimization. The execution time of this function is critical since it is in the innermost loop, and thus is going to be executed intensively in the search algorithm.

2.3 Performance prediction

Since the system may be complex and potentially require extensive time consuming stochastic simulation, we would like to be able to estimate the performance of the proposed redundant system early in the design procedure.

We will use a first-order linearized model. The model is obtained by differentiating the system transfer function with respect to the stochastic scaling factors \( g_{m_1,m_2,n} \). We define the linearized terms as

\[
\frac{dH_{i,j}}{\partial g_{a_{n},n}} = \sum_{k=0}^{M} \sum_{r=0}^{M} E[\partial H_{i,j} / \partial g_{k_{r},n}] \, E[\partial g_{k_{r},n} / \partial g_{a_{n},n}].
\] (2.15)

The first term of the double sum expresses how sensitive the system is to a change in the scaling factor \( g_{k_{r},n} \). In the example given in (2.10) the sensitivity of \( g_{0,1,1} \) is

\[
E[\partial H_{i,0} / \partial g_{0,1,1}] = f_{i,0,2} p_{1,2} p_{0,2} f_{0,1,1} f_{i,0,0} p_{1,0} p_{0,0}.
\] (2.16)
Table 2.1: All possible combination of second moments in the sub-terms of transfer functions and their expectation values. The variables $a_1$, $a_2$, $m_1$, and $m_2$ are assumed to be non equal between themselves unless otherwise specified. The support function $C_n(p)$ is defined in (2.14).

<table>
<thead>
<tr>
<th>Case</th>
<th>Condition</th>
<th>$E [g_{m_1,m_2,n}g_{a_1,a_2,n}]$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$a_1 = a_2 = m_1 = m_2$</td>
<td>$C_n(p_{a_1,n})$</td>
</tr>
<tr>
<td>2</td>
<td>$a_1 = a_2 = m_1$</td>
<td>$C_n(p_{a_1,n})p_{m_2,n}$</td>
</tr>
<tr>
<td>3</td>
<td>$a_1 = a_2 = m_2$</td>
<td>$C_n(p_{a_1,n})p_{m_1,n}$</td>
</tr>
<tr>
<td>4</td>
<td>$a_1 = m_1 = m_2$</td>
<td>$C_n(p_{a_1,n})p_{a_2,n}$</td>
</tr>
<tr>
<td>5</td>
<td>$a_1 = a_2, m_1 = m_2$</td>
<td>$p_{a_1,n}p_{m_1,n}$</td>
</tr>
<tr>
<td>6</td>
<td>$a_1 = a_2$</td>
<td>$p_{a_1,n}p_{m_1,n}p_{m_2,n}$</td>
</tr>
<tr>
<td>7</td>
<td>$a_1 = m_1, m_2 = a_2$</td>
<td>$C_n(p_{a_1,n}p_{a_2,n})$</td>
</tr>
<tr>
<td>8</td>
<td>$a_1 = m_1$</td>
<td>$C_n(p_{a_1,n}p_{a_2,n})p_{m_1,n}$</td>
</tr>
<tr>
<td>9</td>
<td>$a_1 = m_2, a_2 = m_1$</td>
<td>$C_n(p_{a_1,n}p_{a_2,n})$</td>
</tr>
<tr>
<td>10</td>
<td>$a_1 = m_2$</td>
<td>$C_n(p_{a_1,n}p_{a_2,n})p_{m_1,n}$</td>
</tr>
<tr>
<td>11</td>
<td>$a_2 = m_1 = m_2$</td>
<td>$p_{a_1,n}C_n(p_{a_2,n})$</td>
</tr>
<tr>
<td>12</td>
<td>$a_2 = m_1$</td>
<td>$p_{a_1,n}C_n(p_{a_2,n})p_{m_2,n}$</td>
</tr>
<tr>
<td>13</td>
<td>$a_2 = m_2$</td>
<td>$p_{a_1,n}C_n(p_{a_2,n})p_{m_1,n}$</td>
</tr>
<tr>
<td>14</td>
<td>$a_1 = m_2$</td>
<td>$p_{a_1,n}p_{a_2,n}p_{m_1,n}$</td>
</tr>
<tr>
<td>15</td>
<td>All other conditions</td>
<td>$p_{a_1,n}p_{a_2,n}p_{m_1,n}p_{m_2,n}$</td>
</tr>
</tbody>
</table>

Note that in our proposed system, some of the outputs of each PE are dependent on other PEs, which implies that the signals are dependent on multiple PEs. Going back to (2.4) we notice that the underlying $R(m_1,k,n)$ will be common for many scaling factors.

The second term of (2.15) is the connection between the scaling factor and the PEs and is

$$E \left[ \frac{\hat{g}_{k,n}}{\hat{g}_{r,a,n}} \right] = \begin{cases} 
1 , & k = a = r \\
p_{r,n} , & k = a \neq r \\
p_{k,n} , & k \neq a = r \\
0 , & k \neq a \neq r 
\end{cases} \quad (2.17)$$

Finally, assuming that the degree of redundancy is high, making use of the central limit theorem, and applying the triangle inequality, we may define a deviation from the absolute transfer function as the combined root-mean-square of all deviations as

$$|\Delta H_{ij}|^2 = \sum_{a=0}^{M} \sum_{n=0}^{N} \left| \frac{\hat{H}_{ij}}{\hat{g}_{a,a,n}} \right|^2 \sigma_{a,a,n}^2 \Phi^2, \quad (2.18)$$

where $\Phi = \text{erf}^{-1}(\Phi)$, and $\Phi$ is the desired yield, e.g. 0.99. The variance $\sigma_{a,a,n}^2$ can be shown to be

$$\sigma_{a,a,n}^2 = p_{a,n}(1 - p_{a,n}) \frac{1}{x_n}. \quad (2.19)$$
2.4 Finite-length impulse response filter design example

Figure 2.4: A finite impulse response filter implemented in the basic lattice structure. The input signal is $x[n]$, and the output signal is $y[n]$. The other input is tied to zero and the second output left open.

The expected system performance for $H_{i,j}$ at given yield $\Phi$, can then be expressed as

$$H_{i,j}(\Phi) = \sqrt{|H_{i,j}|^2 \pm |\Delta H_{i,j}|^2}. \quad (2.20)$$

The choice of plus or minus sign depends on if the upper or lower limit is of interest.

2.4 Finite-length impulse response filter design example

To illustrate our method, we consider the design and optimization on a finite-length impulse response (FIR) filter, which has been used as an example before [35] and proved to give some performance. The filter contains simple building blocks, which fit the PEs and can be implemented using the suggested structure in Fig. 2.1. The FIR filter is formed as seen in Fig. 2.4 by selecting

$$f_{0,0,n} = z^{-1} \quad f_{0,1,n} = 0 \quad f_{1,0,n} = c_n \quad f_{1,1,n} = 1, \quad (2.21)$$

for $i = [0, \ldots, N]$. A delay, i.e., time-shift in the discrete-time domain, is indicated as $z^{-1}$. The $c_i$ are the filter coefficients.

Filter specification

We use a low-pass FIR filter specification with a unity gain at direct current (DC). This means that the specification in its simplest form is given by the pass- and stop-band normalized frequencies, $\omega_p T$ and $\omega_s T$, together with the desired amplitude/attenuation in each band, see Fig. 2.5. The target specification for a low-pass filter of order $N$ can be written as [36]

$$D_{10}(\omega T) = \begin{cases} e^{-j N \omega T/2}, & \omega \leq \omega_p \\ 0, & \omega \geq \omega_s. \end{cases} \quad (2.22)$$
Figure 2.5: A general low-pass filter specification. The passband is specified from zero up to the normalized frequency, $\omega_p T$, together with the maximum allowed deviation, $\delta_p$, from the desired gain. The stopband is specified from the normalized frequency $\omega_s T$ up to the sampling frequency. The maximum allowed gain in the stopband is given by the parameter $\delta_s$. The relative performance measurement $A_{att}$ is defined as ratio of the minimum gain in the passband and the maximum gain in the stopband.

We introduce a performance measurement as

$$A_{att} = \left( \min_{\omega_p \in \Omega_p} |H_{10}(\omega_p T)| \right) / \left( \max_{\omega_s \in \Omega_s} |H_{10}(\omega_s T)| \right), \quad (2.23)$$

that quantifies the quality and functionality of the filter. As indicated in Fig. 2.5, the $A_{att}$ is a relative measure and finds the minimum gain difference between pass- and stop-band, which captures the very essential performance property of the filter.

The way the performance is measured may differ with application. For instance, if the system should perform a fast Fourier transform (FFT), an appropriate design specification would contain specifications of the values of the output frequency bin when samples of a specific coherent sinus are applied to the inputs. The relative measurement would be a ratio between the lowest amplitude of desired frequency bin and the highest undesired.

**Deterministic mean and quadratic error**

Since all elements in the FIR filter contains the same processing blocks, we may reasonably argue that error probabilities are the same. Hence, let us assert the probabilities in (2.3) as

$$p_{0,r} = p_r, \quad p_{1,r} = p_s. \quad (2.24)$$
2.5 Simulation setup

From (2.11) we get the expected transfer function from $x[n]$ to $y[n]$ as $H_{10}$, with $z^{-1} = e^{-j\omega T}$

$$E[H_{10}] = \sum_{n=0}^{N} \left( \prod_{p=0}^{n-1} p_r \right) e^{-j\omega T} c_n p_r p_s \left( \prod_{m=n+1}^{N} p_s \right), \quad (2.25)$$

which can be interpreted as a summation over all possible paths, similar to how an ordinary FIR would build up the output.

The paths consist of the first product of upper PEs and their probability to function. The middle term is the crossing from upper PE to lower PE with the gain of filter coefficient $c_n$. And the terms of the last product are the probability of the lower PEs to function. The quadratic error is given by expanding all the terms from (2.13), and substituting the terms from Table 2.1, and finally putting it all into (2.8) which gives

$$E\left[|H_{10} - D_{10}|^2\right] = \sum_{n=0}^{N-1} \sum_{b=a+1}^{N} c_a c_b 2 \cos((b - a)\omega T) \left( \prod_{p=0}^{n-1} p_r \left( p_r + \frac{1 - p_s}{x_p} \right) \right) p_s \left( \prod_{k=a+1}^{b-1} p_r p_s \right) p_r$$

$$+ \sum_{n=0}^{N} c_n^2 \left( \prod_{p=0}^{n-1} p_r \left( p_r + \frac{1 - p_s}{x_p} \right) \right) p_s p_r \left( p_r p_s + \frac{1 - p_s p_r}{x_n} \right) \left( \prod_{q=n+1}^{N} p_s \left( p_s + \frac{1 - p_r}{x_q} \right) \right) + D_{10} D_{10}^*$$

$$- D_{10} \sum_{n=0}^{N} \left( \prod_{p=0}^{n-1} p_r \right) e^{-j\omega T} c_{n} p_r p_s \left( \prod_{m=n+1}^{N} p_s \right)$$

$$- D_{10} \sum_{n=0}^{N} \left( \prod_{p=0}^{n-1} p_r \right) e^{j\omega T} c_{n} p_r p_s \left( \prod_{m=n+1}^{N} p_s \right), \quad (2.26)$$

where the first component is the sum of the cross-correlation between all different paths. The next component is the sum of all quadratic terms, and the last three components are the plain expectation values of the transfer function.

2.5 Simulation setup

According to von Neumann [19], the upper limit of PE error probability for a triple redundant systems is 1/6. von Neumann also states a maximum redundancy of about $x_i < 20000$. This we have deemed as too much in our simulations and instead we chose the number of parallel branches from

$$X \in 3, 5, 7, 10, 15, 22, 33, 46, 68, 100, 150, 220. \quad (2.27)$$

The set starts with degrees of redundancy usually used for majority voters, and continues with exponentially growing numbers up to where it is thought the impact of one faulty PE would be only a fraction of a percent ($\delta \approx 1/220$).

The PE error probabilities are selected from a set, $\epsilon$, containing values swept in 14 steps from 0.1% to 15%. On top of all the different combinations
of $X$ and $\epsilon$, we have created three different design procedures, which are: NOM, COPT, and CXOPT, as compiled in Table 2.2 and explained below.

All design procedures utilize optimization (2.7) to find the best filter coefficients $c_i$ values, but with different constraints. This is what we refer to as the coefficient design phase. After that, in the implementation phase, the $c_i$ values are used to implement the filter where the degree of redundancy is either set by $X$ or, as provided in the coefficient design phase, set by $x_i$, depending on the design procedure. The design procedures are:

- **NOM**, illustrates an ordinary design procedure where no concern is taken to the error probability and redundancy during the coefficient design phase, this is simply done by letting $p_r = 1$, $p_s = 1$, and $x_i = 1$, which reduces (2.25) to an ordinary FIR filter transfer function. In the implementation phase, however, all columns have the same degree of redundancy set by $X$.

- **COPT**, utilizes the knowledge of the error probability, i.e., $\epsilon_r$ and $\epsilon_s$ from the set $\epsilon$, and a uniform redundancy from $X$ throughout each column.

- **CXOPT**, utilizes a full freedom of coefficients and degree of redundancy during the coefficient design phase, based on knowledge of the error probability. The results are then implemented without any modifications.

In all three design procedures, we use a fixed filter specification, $\omega_p T = 0.2 \pi$, and $\omega_s = 0.4 \pi$. The ripple in the passband is set to 1 dB (a comparatively large value) and minimum attenuation in the stopband is set to 45 dB, which gives a 15-th order filter. We have chosen weights such that the values in (2.7) of pass- and stop-band become approximately the same. i.e. 1 and 100.

### Optimization

The optimization problem of 2.7 is a mixed-integer and non-linear. A branch-and-bound [37] meta-optimizing algorithm is applied to handle integer optimization, whilst a min-max solver with constrains solves the non-linear sub problems. The meta-optimizer first finds the optimal solution without integer constraints. From this initial point, a first-order estimate [38] of the function gradients for each integer variable, i.e. $x_n$, is done. This extra search does initially consume more computational time, but is beneficial later when cutting the search tree.

### Monte-Carlo simulation setup

The full set of all possible states with working and non-working processing elements can be very large and simulating all possible states is impractical.
2.6 Simulation results

Table 2.2: The optimization and implementation procedure of the three design cases.

<table>
<thead>
<tr>
<th>Case</th>
<th>Coefficient design phase</th>
<th>Implementation phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOM</td>
<td>free</td>
<td>$c_i$</td>
</tr>
<tr>
<td>COPT</td>
<td>free</td>
<td>$x_i$</td>
</tr>
<tr>
<td>CXOPT</td>
<td>free free</td>
<td>$c_i$</td>
</tr>
</tbody>
</table>

Instead, we simulate a subset of stochastic realizations, i.e., using Monte-Carlo and gather the result in bins $h_i$ in a histogram. The cumulative sum of the histogram will give an estimate of the yield. With the best performance in the highest bin, the yield can be expressed as

$$\Phi(n) = 1 - \sum_{i=1}^{n} h_i,$$  \hspace{1cm} (2.28)

where the center bin $n$ corresponds to a certain performance. Given a desired accuracy of the bins, say $\alpha$, there should be about $1/\alpha$ realizations in each bin in the histogram [39]. It is especially hard to correctly estimate the yield, or loss of yield, when the error probability of non-working PEs is very small. That means, we need to dynamically scale the number of simulations to get enough states with non-working PEs.

The probability of having at least one PE to be non-working is the complementary probability of all PEs working, that is, $1 - (1 - \epsilon)^{x_{\text{tot}}}$, where $x_{\text{tot}} = \sum_{i=0}^{N} x_i$. Hence the number of simulations must at least be

$$N_{\text{runs}} = \frac{n_{\text{bins}}}{\alpha(1 - (1 - \epsilon)^{x_{\text{tot}}})},$$ \hspace{1cm} (2.29)

to cover $n_{\text{bins}}$ number of bins in the histogram. The maximum value of (2.29) is located at low redundancy and low error probability. Early simulations show that most results are within a 15 dB range. and, with a resolution of 0.1 dB, we select $n_{\text{bins}} = 150$. Given this, together with $\epsilon = 10^{-3}$, $\alpha = 10^{-3}$, and $x_{\text{tot}} = (15 + 1) \cdot 5 = 80$ we are required to run $N_{\text{runs}} \approx 2 \cdot 10^6$ simulations to get the desired accuracy. To achieve efficient simulations, a dedicated multi-threaded C-program executes the Monte-Carlo simulations.

2.6 Simulation results

The system is optimized and simulated with all the three design procedures in Table 2.2 and is described below.

Optimization

The optimization mainly affects the parameter space by, in general, increasing the magnitude of the coefficients in order to compensate the gain due
Figure 2.6: Comparison of the filter coefficients when optimization is done without (left) and with (right) concern of the degree of redundancy. In this example the error probability is $\epsilon = 1.0 \cdot 10^{-2}$ and the redundancy is $X = 22$. The filter coefficients (top) are in general slightly increased in amplitude, while the degree of redundancy (bottom) changes from a uniform distribution to a non-uniform distribution.

Monte-Carlo simulation results

Monte-Carlo simulations show that the optimization in the COPT procedure, compared to the NOM procedure, typically reshape the yield curves to emphasize on higher yield for lower performance. See Fig. 2.7 for an example of this. Even more interesting is the results from the CXOPT procedure, where the yield curve has improved even more in the low performance region, but also in total, shifted towards higher (right in figure) performance.

For each combination of the sets of $X$ and $\epsilon$, the intersect points of NOM for $\Phi = 0.70, 0.95$, and $0.99$ are decided. From these points, either the increased yield at same performance, $\Delta \Phi$, or the increased performance at same yield, $\Delta A_{att}$, are determined. However, for a low degree of redundancy, i.e. $X = 3$ and $X = 5$, the performance of the optimized procedures are often much worse than the NOM procedures. These cases can be discarded as outliers with the argument that they would most probably be designed with majority voters. Therefore, the mean values and standard deviations in Table 2.3 are calculated without these cases. As seen in Table 2.3, the yield-mean
2.6. Simulation results

Figure 2.7: The yield of NO\textsubscript{M}, CO\textsubscript{PT}, and CXO\textsubscript{PT} design procedures for $X = 22$ and $\epsilon = 0.01$. The optimization reduces the amount of high performance in favor of better yield at low performance, as illustrated by the arrows. The superfluous performance can be traded for increased yield at lower performance.

of the CO\textsubscript{PT} procedure increases poorly compared to the yield-mean of the NO\textsubscript{M} procedure. The small improvements are smaller than, or in the same range as, the standard deviation. Hence, we cannot claim any improvement. Same issue can be observed with the mean of performance improvement $A_{att}$. Detailed studies of the results show that CO\textsubscript{PT} actually performs rather well for low error probabilities and high degrees of redundancy. Regression indicates that

$$\epsilon < X^{0.75} \Phi^{3.35} / 272,$$

(2.30)

should be met to guarantee improvement in the CO\textsubscript{PT} procedure.

CXO\textsubscript{PT} procedure is better than CO\textsubscript{PT}. It reduces the mean-yield loss compared to the NO\textsubscript{M} procedure by about 4% at $\Phi = 95\%$, and about 0.9% at $\Phi = 99\%$, which means that the yield loss has been reduced by about 80% relative in those two cases. Here the standard deviations are small enough to support a suggested improvement. The CXO\textsubscript{PT} shows less improvement at $\Phi = 70\%$ than for the higher yield cases. This is due to, the optimization in some points of $\epsilon$ and $X$, emphasize on yields higher than $\Phi = 70\%$.

A collection of all the $A_{att}$ performance data for the CXO\textsubscript{PT} procedure at $\Phi = 0.99\%$, is found in Fig. 2.8. The performance trend is rather dis-
2. **Electronic Clay**

![Figure 2.8: Performance of CXOPT procedure in Monte-Carlo simulation with the yield of $\Phi = 0.99$. With low error probability and a high degree of redundancy (lower left corner) the performance is good. Increasing the error probability, or reducing the degree of redundancy, will reduce the performance. Eventually the system breaks down (top right) with too high error probability and too low degree of redundancy.](image)

Table 2.3: Yield and performance improvements of COPT and CXOPT case compared to the NOM case. The results of $X = 3$ and $X = 5$ have been deemed as outliers.

<table>
<thead>
<tr>
<th>$\Phi$ [%]</th>
<th>Procedure</th>
<th>$\Delta \Phi$ [%]</th>
<th>$\sigma_{\Delta \Phi}$ [%]</th>
<th>$\Delta A_{\text{att}}$</th>
<th>$\sigma_{\Delta A_{\text{att}}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>COPT</td>
<td>-0.71</td>
<td>14.18</td>
<td>-0.02</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>CXOPT</td>
<td>12.53</td>
<td>9.42</td>
<td>0.89</td>
<td>0.90</td>
</tr>
<tr>
<td>95</td>
<td>COPT</td>
<td>1.30</td>
<td>2.28</td>
<td>0.42</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>CXOPT</td>
<td>4.02</td>
<td>0.82</td>
<td>1.82</td>
<td>0.84</td>
</tr>
<tr>
<td>99</td>
<td>COPT</td>
<td>0.38</td>
<td>0.41</td>
<td>0.48</td>
<td>0.59</td>
</tr>
<tr>
<td></td>
<td>CXOPT</td>
<td>0.89</td>
<td>0.12</td>
<td>1.99</td>
<td>0.89</td>
</tr>
</tbody>
</table>

The case of high redundancy and low error probability gives the best performance. From there the performance decreases, either by reduced redundancy or increased error probability.

**Performance prediction result**

As we define the performance in (2.23) as the attenuation ratio between the pass- and stop-band, the expected performance can, given the yield $\Phi$, be predicted from (2.20). Expressing the equation in logarithmic dB scale gives

$$
\hat{A}_{\text{att,min}} = 10 \log_{10} \frac{\min\left(\frac{|H_{1,0}(\omega_s)|^2 - |\Delta H_{1,0}(\omega_s)|^2}{\max\left(|H_{1,0}(\omega_s)|^2 + |\Delta H_{1,0}(\omega_s)|^2\right)}\right)}{\max\left(|H_{1,0}(\omega_s)|^2 + |\Delta H_{1,0}(\omega_s)|^2\right)}, \quad (2.31)
$$
2.6. Simulation results

where $\omega_p$ and $\omega_s$ are swept over the pass- and stop-band, respectively.

To verify the prediction estimates, the filter cases where optimized for different degrees of redundancy and error probabilities. For each combination the filter was simulated by Monte-Carlo with two million realizations to achieve enough accuracy. In Fig. 2.9 the results from CX$_{OPT}$ at $\Phi = 0.99$ are found. The prediction measurement is calculated according to equation (2.31), where the underlying sensitivity is calculated according to (2.15). The results of the CX$_{OPT}$ case are presented in Fig. 2.10.

Comparing the two figures shows that the prediction does overestimate the performance at low error rates and underestimate at higher error probability, especially at lower degree of redundancy. The model is compared with Monte-Carlo simulation result using the $R^2$ measure [40] and the result of the three different cases can be found in Table 2.4. In the table the values in parenthesis are the ratios of non-outliers data points. Outliers are predictions and simulations that generate not-a-number (NaN) results, see Fig. 2.10. For low degree of redundancy, the performance prediction generally is inaccurate, due to the assumption of Gaussian distribution. We can also see a trend that the performance is a little overestimated at low error probability.

However, generally the performance prediction fit the data of all three cases with a $R^2$ around 90%, and only a few percent of outliers data, which is to be considered as reasonably fit.

Comparison to majority voter

The comparison in this section can be considered as comparing apples to pears, not the same but neither too far fetched. Actually if you think about it, the apple doesn’t fall far away from the pear tree.

Figure 2.9: A scaled version of Fig. 2.8, for comparison with Fig. 2.10.
Figure 2.10: The predicted performance of the CXOPT design case at the yield of $\Phi = 0.99$. The predictor gives in this case a general overestimation of the performance. However, at high error probability and low degree of redundancy (top right), it is underestimating the performance.

Table 2.4: The $R^2$ model fit measurement of the performance prediction, $A_{att,\min}$, validated on Monte-Carlo result. The ratio of non-outliers data points in parenthesis.

<table>
<thead>
<tr>
<th>$\Phi$ [%]</th>
<th>NOM</th>
<th>COPT</th>
<th>CXOPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.70</td>
<td>0.88 (1.00)</td>
<td>0.91 (1.00)</td>
<td>0.88 (1.00)</td>
</tr>
<tr>
<td>0.95</td>
<td>0.91 (0.99)</td>
<td>0.93 (0.99)</td>
<td>0.90 (0.99)</td>
</tr>
<tr>
<td>0.99</td>
<td>0.89 (0.97)</td>
<td>0.91 (0.97)</td>
<td>0.87 (0.96)</td>
</tr>
</tbody>
</table>

A system in which every voter in Fig. 2.2 is a majority voter would be very reliant, but at the cost of many complex voters. But also an unfair comparison to a mean voter system. Instead, we create a design case we call SIMP, which is based on the original system, as in Fig. 2.1, but we make a number of redundant equal branches which are eventually gathered in one single majority voter at the end. This would be a fair comparison in the sense that either a single complex majority voter is used or many simple mean voters. However, we are comparing apples to pears.

The error probability of the SIMP case is shown in Fig. 2.11, where the we have to use logarithmic scale to visualize the result. The yield versus performance curves for a system of majority voters only consists of two points, either enough PEs in the circuit works and certain $A_{att}$ is obtained, or not enough PEs work which results in no performance at all. On the other hand, a mean voter based system will, as seen in Fig. 2.7, have a smooth transition in yield when sweeping $A_{att}$. 
2.7. Discussion

In this chapter we have elaborated on, formalized and simulated a system of clay electronics based on mean voters. Starting with a general lattices structure, we introduce redundancy, due to the error-prone building blocks. Assuming the error probability for the components are known, we form a deterministic expression on the expected mean and deviation, which is used for optimization. Furthermore, we also derive a first order prediction model of the expected yield.

To the general lattice structure an FIR filter is applied. The filter is made redundant. The coefficients and the degree of redundancy is optimized. Finally, the design example is run by Monte-Carlo simulations. We saw the

Comparing the majority voter with the mean-voter directly tends to favor the former. However, if the requirement of $A_{att}$ is relaxed in favor for a higher yield, there are conditions when the mean voter actually behaves better.

The SIMP case gives yield performance comparable to mean voter designs. For low error probabilities the performance is decent, but the design starts to provide low yields around 1% error probability and above. As seen in Fig. 2.12 there are some combination’s of $X$ and $\epsilon$ such that the mean voter implementation have less yield loss compare to the SIMP case, but as asserted, the cost to pay is a reduction in performance in $A_{att}$.

2.7 Discussion

Figure 2.11: Yield loss of the SIMP case. The loss is expressed as $d\beta = \log_{10}(1 - \Phi)$, i.e., the power of the error probability of the hole system. In principle, the SIMP case fails for all error probabilities of PEs above $2.2 \cdot 10^{-2}$. Below that, the degree of redundancy influence as expected, low loss at high redundancy and vice versus. Due to numerical limitations the results in the figure are limited to $-14$. 

$\begin{array}{cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
yield curve did change behavior due to the optimization. The curve shifted down the ratio of high performance realizations in favor for higher yield at lower performance. i.e. the design become more stable against errors. Furthermore, the optimization also showed that the degree of redundancy should not be unified distributed.

In our FIR filter it was shown that the redundancy depends also on the coefficients, i.e., the weighting of the signal at different stages throughout our system. Also, although the FIR filter coefficients are symmetric through the signal path, the amount of redundancy is not necessarily symmetric.

The model fit of the performance prediction is sufficiently good for the FIR filter example, but is inadequate for low degree of redundancy.

The mean voter structure can outperform the majority voter in some cases, when the degree of redundancy is rather high and the components are error-prone, but of the cost of a reduced performance.

Assume that Moore’s law, even in the future, continues based on shrinking of components mainly. A fair share of those components will malfunction, which in the end will mean the end of Moore’s law, since any manufactured chip will contain errors.

If we instead contemplate the situation and construct systems adapted to the situation, such the clay system with mean voters, Moore’s law does not have to end.
Chapter 3
Memristors

The memristor has a high potential of becoming the next enabler to keep up with Moore’s law. The major reason for this is the small feature size, combined with the interesting property of being a non-volatile memory function. In addition, the suggested structures enables computation and storage at the same location, similar to how neurons operate.

3.1 Introduction

Late professor Leon Chua indicated in 1971 the existence of the memristor as a fundamental circuit component by the paper “Memristor - The Missing Circuit Element” [41]. In his paper he showed how the four quantities current (i), charge (q), voltage (u), flux (ϕ) can be combined in six different pairs and each pair has a relationship described by fundamental circuit equations. Five of those equations where known to science, but the sixth, between charge and flux, was new, see Fig. 3.1.

According to the paper, the sixth relationship does have a memory in the equation, and the unit is resistance, or to be precise Ohm. This chapter are about that device with memory and resistance, the memristor.

Purpose

The purpose and aim of this chapter is to acquire knowledge how the memristor behaves as a device, and see where and how it can be used in applications which could be used to extend Moore’s law beyond continuing shrinking of transistor dimensions.

Demarcation

Since our interest lies in the fundamental principles of the functionality, we limit ourselves to investigate simpler models of the device only. Even though a lot of effort is done on the modeling of the memristor, the basic understanding can still be captured with a simpler model. Similar to how transistors are modeled with basic circuit theory during design, such as the operation point.
and h-parameter model of a bipolar transistor, we do think a simple model is accurate enough for our purpose.

Chapter overview

In this chapter we cover the basic physics and relationships of the memristor, some of the available models of the transistor and their additional window functions. Furthermore, we elaborate upon the available design space the memristor has, introduce amplitude characteristic and simulation results. Finally the chapter is concluded by some applications and conclusions.

Background

The number of papers on the memristor after Chua’s paper in 1971 [41] was very sparse until 2008, when Hewlett Packard Labs (HP) presented a real working device [42, 43]. After this presentation of a real existing component the number of articles on the subject more or less exploded1, as seen in Fig. 3.2. The small dip of the graph at 2015 can either be due to the fact that year 2015 is not yet completed at the time of writing, or an early indication that the graph follows the Gartner hype cycle.

3.2 Physical relationships

Figure 3.2: The number of papers containing the word “memristor” according to Google Scholar. After Chua’s paper in 1971, the number of articles stayed on a handful each year. In the late 90’s the number started to increase and completely explode with HP Labs paper in 2008.

Probably similar to Dr Emmett L. Brown’s flux capacitor, the memristor device has the appealing property of both being able to compute and store information at the same time. In addition the physical size is a very appealing factor, the dimensions are on par with via- and cut-contacts which interconnect between the layers of metal and substrate in a IC. That is, the size is at least about a half or one power smaller than transistors at the time. With structures as suggested by Strukov [44] calculations has been made, showing that a memristor-based thumb drive would be able to hold the equal amount of the total information stored by library of congress.

The content of this chapter is mostly based on the work of the author and co-authors in [45, 46].

3.2 Physical relationships

Let us look closer at the discovery Chua made. In Fig. 3.1 the six different combinations of current ($i$), charge ($q$), voltage ($u$), and flux ($\phi$) are arranged in the corners.

---

2Movie character in Back to the Future
If we start with the most fundamental combination, the relationship between current and voltage, we see that the relationship between them is

\[ u(t) = Ri(t), \quad (3.1) \]

which we recognize as Ohm’s law. This is indicated in Fig. 3.1 by the \( R \) arrow.

The relationship between voltage and charge is capacitance. From basic circuit theory we get

\[ \frac{du(t)}{dt} = \frac{i(t)}{C}. \quad (3.2) \]

Integration after minor rearrangement gives

\[ q(t) = u(t)C, \quad (3.3) \]

which is the \( C \) arrow in Fig. 3.1.

The third elementary component, the induction coil, has the relationship between current and voltage by the equation

\[ u(t) = \frac{di(t)}{dt}L, \quad (3.4) \]

and integration gives

\[ \int_{-\infty}^{t} v(\tau)d\tau = \phi(t) = Li(t), \quad (3.5) \]

where the terms on left and right side of the left equality sign is the relation between voltage and flux. The terms besides the right equal sign give us the relationship between flux and current.

Furthermore, the integration of current over time is by definition the charge

\[ q(t) = \int_{-\infty}^{t} i(\tau)d\tau. \quad (3.6) \]

Now, one relationship is remaining, the link between charge and flux. Let us assume the relationship has the same form as previous relationships and for the general expression

\[ \phi(t) = M(q(t))q(t), \quad (3.7) \]

where \( M(q) \) is the memristive function. Writing out (3.7) with (3.5 and 3.6) gives

\[ \int_{-\infty}^{t} v(\tau)d\tau = M(q(t)) \int_{-\infty}^{t} i(\tau)d\tau. \quad (3.8) \]

Differentiating both sides, however slightly crude on right hand side, finally gives

\[ v(t) = M(q(t))i(t), \quad (3.9) \]

where we now see that the unit of memristance is also Ohm.

In short, the memristor is a resistor that changes resistance as the current flows. Imagine a river bank that erodes as the water current flows and that the process is reversible.
3.3 Memristor models

There are a number of different models developed for the memristor in the literature [43, 47–56]. They differ in complexity and accuracy and we will cover some of them. However, since our interest is in the fundamental behavior of the memristor, we choose to explore and investigate the simplest one, which is the linear drift model.

**Linear drift model**

With the presentation of a real physical tangible memristor, HP Labs also included the linear drift model [43]. The idea is to identify the memristor as two regions of different resistive characteristics. The resistivity is set by the amount of oxygen vacancies, i.e., the doping by the positive ions.

In Fig. 3.3 a model of a memristor of depth $D$ is illustrating the two regions of doped and non-doped material. The orientation of depth $D$, length $L$, and width $W$ can be chosen arbitrary, but since the memristor is most likely to be implemented in a memory, as a via-contact in a metal layer stack, this conceptual orientation looks to be the most reasonable [44, 57, 58].

The abstract (since it is not real, only conceptual) intersection between the two regions is located at $d$, where we have the constraint $0 \leq d \leq D$. This is the state of the memristor. To constrain the state, we will later introduce different window functions.

The resistance the memristor asserts is the linear combination of the two regions of doped and non-doped regions, as seen in Fig. 3.4. To abstract state further and remove the dimension property, we assert $x = d/D$ and hence the constraint/borders becomes $0 \leq x \leq 1$. The total resistance, or
Figure 3.4: The circuit equivalent of the linear memristor model. The regions of doped and non-doped material are modeled as two variable linked resistors in series.

Memristance, is thus

\[ M(x) = xR_{on} + (1 - x) R_{off}. \]  

(3.10)

The speed of the state was, quite reasonably, as suggested by HP Labs to follow the mobility equation for electrons \[ [49, 59] \]

\[ v = \mu E, \]  

(3.11)

where \( \mu \) is the charge mobility and \( E \) is the strength of the electrical field, which is the voltage over the device divided by the depth. As a conservative approximation, the voltage over the terminals of the memristor is according to Ohm's law: \( v_m(t) = R_{on}i_m(t) \). Hence, the speed of the state can be expressed as

\[ v = \frac{dd}{dt} = \mu E = \mu \frac{R_{on}i_m(t)}{D}. \]  

(3.12)

With the derivation of the substitution above, \( dx = dd/D \), we can express the velocity as

\[ D \frac{dx}{dt} = \frac{\mu R_{on}}{D} i_m(t). \]  

(3.13)

With a slight rearrangement we get

\[ \frac{dx}{dt} = \frac{\mu R_{on}}{D^2} i_m(t), \]  

(3.14)

and integration of both sides gives

\[ \int_0^t \frac{dx}{d\tau} d\tau = x + x_0 = \mu \frac{R_{on}}{D^2} \int_0^t i_m(\tau) d\tau = \mu \frac{R_{on}}{D^2} q(t). \]  

(3.15)

The initial state \( x_0 \) must have the same boundaries as the state itself, i.e., \( 0 \leq x_0 \leq 1 \). Since the memristor is a memory, the initial state is defined by any operation the memristor had prior to \( t = 0 \).

The (3.15) does resemble the equation of a capacitor, behold the equation

\[ u_C(t) + u_{C_0} = \frac{1}{C} \int_0^t i_C(\tau) d\tau, \]  

(3.16)
where $u_C(t)$ and $i_C(t)$ is the voltage and current through the capacitor and $u_{C_0}$ is the initial condition. From this we may already suspect a frequency dependency of the memristive behavior.

To determine the initial condition in terms of resistance, the (3.15) is combined with (3.10). Rearranging the terms gives

$$M(q(t)) = \frac{\mu R_{on}}{D^2} (R_{on} - R_{off}) q(t) - x_0 (R_{on} - R_{off}) + R_{off}. \quad (3.17)$$

Finally, assume that the initial resistance is known at $t = 0$ and the accumulated charge at that point is zero, i.e., $M(q(t = 0)) = R_{init}$, which gives

$$x_0 = \frac{R_{off} - R_{init}}{R_{on} - R_{off}}. \quad (3.18)$$

Before continuing with the linear drift model, some other models are briefly covered.

**Non-linear ion drift**

The non-linear ion drift model was proposed by Lehtonen et al. [56] based on the work by Yang et al. [54]. The current through the memristor is given by

$$i(t) = x(t)^n \beta \sinh(\alpha v(t)) + \chi \left(e^{\gamma v(t)} - 1\right), \quad (3.19)$$

where $n, \alpha, \beta, \gamma$, and $\chi$ are parameters to minimize the error to experimental data. The first part of the equation, the sinh, is a symmetrically exponential growing function which is crossing the origin. The second part is the diode equation.

The speed of the state is given by

$$\frac{dx}{dt} = aH(x)v(t)^m, \quad (3.20)$$

where $a$ is a material constant, $m$ an odd integer, and $H(w)$ is a window function assuring stability, as described below.

The requirement on $m$ may seem odd, but an even $m$ would only drive the state in one direction. If an even $m$ is desirable, the function must also include a sign function.

The non-linear drift model complicate the I-V relationship and the state movement behavior, which gives a progressive response to the applied voltage, i.e. we get second order effects that may inflict on the basic principal behavior.

**Simmons tunneling barrier**

Based on the Simmons tunneling model, Picket et al. [52] formed a set of equations for the memristor. Further development was done by Abdalla and
3. Memristors

Figure 3.5: The Simmons tunneling barrier macro model of the memristor. Between two platinum plates there are two regions, a non-doped $\text{TiO}_2$ and a doped $\text{TiO}_{2-x}$. The doped region causes a series resistance, while the non-doped causes a barrier of length $w$, in which the electrons tunnel through.

Pickett [60] which is presented here briefly. The structural concept of the model can be seen in Fig. 3.5. The memristor is still constituted between two platinum plates, and the doped region is as before modeled as a resistor, in this case the resistor $R_s$. However, the non-doped region is modeled as a barrier of width $w$, where the electrons tunnel through. An electrical structure of the Simmons tunneling barrier model of the memristor is also shown in Fig. 3.5, where the relation between current and voltage is given by

$$v_m = v_g + i_m R_s,$$  \hspace{1cm} (3.21)

where $v_m$ is the voltage over the memristor, $v_g$ the voltage over the barrier, and $i_m$ is the current through the memristor.

The state speed, in distance per second, is given by

$$\frac{dw}{dt} = \left\{ \begin{array}{ll}
    f_{\text{off}} \sinh \left( \frac{i_m}{i_{\text{off}}} \right) \exp \left[ - \exp \left( \frac{|i_m| - \frac{|i_{\text{off}}|}{|i_{\text{c}}|}}{w_{\text{c}}} \right) - \frac{w}{w_{\text{c}}} \right], & i_m > 0 \\
    f_{\text{on}} \sinh \left( \frac{i_m}{i_{\text{on}}} \right) \exp \left[ - \exp \left( - \frac{|i_m| - \frac{|i_{\text{on}}|}{|i_{\text{c}}|}}{w_{\text{c}}} \right) - \frac{w}{w_{\text{c}}} \right], & i_m < 0
\end{array} \right.,$$  \hspace{1cm} (3.22)

where $f_{\text{off}}, f_{\text{on}}, i_{\text{off}}, i_{\text{on}}, a_{\text{off}}, a_{\text{on}}, b$, and $w_c$ are model parameter to be fitted. The relation between $v_g$ and $i_m$, beside (3.21), is given by

$$i_m = \frac{j_0 A e}{(\Delta w)^2} \left\{ \phi_1 \exp(-B \sqrt{\phi_1}) - (\phi_1 + |v_g|) \exp(-B \sqrt{\phi_1 + |v_g|}) \right\},$$  \hspace{1cm} (3.23)

where $A$ is the cross section area of the tunnel and $e$ the charge of the electron. Furthermore, we have

$$j_0 = \frac{e}{2\pi h}, \quad w_1 = \frac{1.21}{\phi_0}, \quad \Delta w = w_2 - w_1,$$  \hspace{1cm} (3.24)

where $h$ is the Planck constant, $\phi_0$ barrier constant in volts, and $w_1$ model constant in nm. The remaining parameters are calculated as

$$\phi_1 = \phi_0 - |v_g| \left( \frac{w_1 + w_2}{w} \right) - \frac{1.15 \lambda w}{\Delta w} \ln \left( \frac{w_2 (w - w_1)}{w_1 (w - w_2)} \right),$$  \hspace{1cm} (3.25)
Memristor models

\[ B = \frac{4\pi \Delta \omega 10^{-9} \sqrt{2m}e}{h}, \quad (3.26) \]

\[ w_2 = w_1 + w \left( 1 - \frac{9.2\lambda}{3\phi_0 + 4\lambda - 2|v_g|} \right), \quad (3.27) \]

and

\[ \lambda = \frac{e \ln(2)}{8\pi \epsilon_0 \omega 10^{-9}}, \quad (3.28) \]

where \( m \) is the mass of the electron.

The Simmons tunneling barrier model gives a relative good fit to measured data [52]; but the drawback is the rather complicated equations which are implicit defining the relationships between \( i_m, v_g, \) and \( w \), which causes longer simulation time.

Threshold adaptive memristor model

The Simmons tunnel barrier model can according to [50] be simplified. The suggested threshold adaptive memristor (TEAM) model has a simpler state change equation given by

\[
\frac{dx}{dt} = \begin{cases} 
    k_{on} \left( \frac{i_m}{i_{on}} - 1 \right)^{a_{on}} f_{on}(x), & i_m < i_{on} < 0 \\
    0, & i_{on} < i_m < i_{off} \\
    k_{off} \left( \frac{i_m}{i_{off}} - 1 \right)^{a_{off}} f_{off}(x), & 0 < i_{off} < i_m,
\end{cases} \quad (3.29)
\]

where \( k_{on}, k_{off}, a_{on}, a_{off}, i_{on}, \) and \( i_{off} \) are model constants. As seen in (3.29), state only moves towards the on region for negative currents below the threshold \( i_{on} \) and towards the off region for positive currents larger than \( i_{off} \).

The powers of \( a_{on} \) and \( a_{off} \) are used to approximate the exponential function; and the functions \( f \) are typically windows function as described below.

The current voltage dependency is expressed by the function

\[ v_m = R_{on} i_m \exp \left( \frac{\lambda (x - x_{on})}{x_{off} - x_{on}} \right), \quad (3.30) \]

where

\[ \lambda = \ln \left( \frac{R_{off}}{R_{on}} \right). \quad (3.31) \]

The TEAM model is simpler and faster, at the cost of a mean error of only 0.2% against the Simmons tunneling barrier model [50]. The model is very flexible, by choosing the window functions carefully a large group of models can for example be implemented, such as the linear drift model and Simmons tunneling barrier [61]. In principle the TEAM model is a wrapper, containing all the other models.
Window functions

Major part of the models uses window functions, i.e. the linear, Non-linear ion drift, and TEAM model. The window function should limit the state to be within the boundary of $[0, 1]$. Hence we must introduce some non-linearity to keep the state variable within the borders.

Perhaps, the most obvious way is to simply add two conditional statements that moves the state back to either border as soon as the state is out of bound. A more elegant solution is to introduce a window function on the speed of the state such that it never can go out of the defined region. The window function is applied by insertion into (3.14) by

$$\frac{dx}{dt} = H(x)\mu \frac{R_{on}i_m(t)}{D^2}.$$  \hspace{1cm} (3.32)

Still the most intuitive solution is to simply add a brick wall constrains, i.e.,

$$H_t(x) = \begin{cases} 
1, & 0 \leq x \leq 1 \\
0, & \text{otherwise} \end{cases}.$$  \hspace{1cm} (3.33)

However by doing so, every time the state hits the boundary, the relation between charge and state redistributed, i.e. all information of former current is lost in (3.15).

Joglekar-Wolf suggested a group of window functions

$$H_2(x) = F_p(x) = 1 - (2x - 1)^{2p},$$  \hspace{1cm} (3.34)

where the parameter $p$ sets the sharpness of the window, see Fig. 3.6, where a few of the functions are plotted. As $p$ increases the more of brick wall function the window becomes. The Joglekar-Wolf window function makes the state to be contained within the borders by simply reducing the speed of the state. Since the speed of the state continuously decrease the closer to the border it gets, will the state converge towards 0 or 1 for a (reasonable; limited, non-Dirac) constant valued input current. From this follows that all (reasonable) input currents are mapped on the state positions between 0 and 1. This implies that the relationship between charge and state position is sustained.

The assumption of the state having the same speed towards the border as from it in the close proximity of the border, requires the same amount of current and time to get the state out of the region as into it (actually it is the same charge in and out, but for most of us is more convenient to talk about current and time).

As seen within the other models, it is likely that the speed towards the borders are some sort of exponential function. Furthermore, it is most reasonable to assume that as soon as the direction of the current changes, a new initial condition and new exponential function holds. To get a more practical memristor, Biolek suggest the following set of window functions

$$H_3(x) = G_p(x) = 1 - (x - \theta(-i_m(t)))^{2p},$$  \hspace{1cm} (3.35)
3.3. Memristor models

Figure 3.6: Some Joglekar-Wolf window functions. By reducing the speed of the state close to the boundaries, the state \( x \) is kept within the region \([0, 1]\).

\[
\theta(x) = \begin{cases} 
0, & x < 0 \\ 
1, & x \geq 0
\end{cases}.
\] (3.36)

As seen in Fig. 3.7, the Biolek window function in principle consists of two over layered functions, symmetrical around 0.5. Similar to the Joglekar-Wolf window function, the Biolek window function causes the speed of the state to slow down as it moves closer towards the borders. However, the selection between the two different functions are determined by the direction of the current. This means, the state is contained within the borders by slowly...
converging when moving towards the border. On the other hand, when the current changes direction, the other function is chosen and the speed of the state towards the mid-region are done with ease.

These equations is now enough to make a simple model for simulations in tools such as P-spice or Spectre in Cadence. However, the Joglekar-Wolf and Biolek window do contain a flaw, the functions are negative outside the region \([0, 1]\). This means when the simulator do a time step and extrapolate current differential equations it may sometimes end up with a negative valued window function and this causes a divergence which makes the simulator to halt. To overcome this problem, the tolerances of the simulator can be tightened to the cost of extended simulation time. This may be exaggerated costly if the design is large and only a few of the components are in need of such high precision.

To avoid this problem, we can insert a safe region close to the borders, as seen in Fig. 3.8. At this safe region the window function is zero. When the simulator steps into this region, it will discover the discontinuity and steps back. We can add the safe region by an additional window function

\[
S(x, V_m) = \theta(V_m)\theta(x_{max} - x) + \theta(-V_m)\theta(x - x_{min}),
\]  

(3.37)

where \(\theta\) is defined in (3.36), and \(x_{max}\) and \(x_{min}\) defines the borders of the safe region. With the safe region term, (3.32) becomes

\[
\frac{dx}{dt} = G_p(x)S(x, V_m)\mu \frac{R_{on}i_m(t)}{D^2}.
\]

(3.38)

It is now possible to trade the size of the safe region, against simulation speed. With less precision and thus less simulation time, the safe region needs to be bigger. If the simulator diverge, you now have the choose of increase either the precision or the safe zone of the window.

### 3.4 Design space

Regardless within field you are as a designer, the context sets limitations on your choices. A subset of these limited choices are the design parameters, for instance you can’t have infinitely small gate length of a transistor due to the restrictions in manufacture process and eventually the granularity of atoms. You may also have requirements of quantified steps, either by the design rules directly or indirectly by requirements of symmetry.

To understand the design space for the memristor, we have to break down terms in equations (3.10). Rather obvious does the physical dimensions of the device sets the behavior, such as width \((W)\), length \((L)\), and depth \((D)\). However, most likely, the resistivity are not something eligible for the designer, but decided by doping concentrations. In principle we can express the two parameters of (3.10) by

\[
R_{on} = \frac{\rho_{on}D}{WL},
\]

(3.39)
3.4. Design space

Figure 3.8: By adding an additional brick wall function to the window function, there is a safe region the simulator has to encounter. Thus can the non-stable region be avoided.

\[ R_{\text{off}} = \rho_{\text{off}} \frac{D}{WL}, \]  

(3.40)
where \( \rho \), (in \( \Omega \cdot m \)) is given by the material conductivity defined by the manufacture process.

Another aspect of the design space may be that one of the dimensions may be locked to a certain value. For memristors implemented between different metal layers, the depth \( D \) may only be the thickness of the filler material between two metal layers, or possibly in best case between any two layers, giving slightly more options. For a lateral oriented memristor the “length” (\( L \)) in Fig. 3.3 most certainty will be locked to a specific dimension specified by the process. On the other hand, may the “depth” (\( D \)), be freely chosen given a wider range of options.

Furthermore, the mobility, \( \mu \), is given by the process and can not be changed. This finally means that the state speed is given from (3.39) into (3.14) is

\[ \frac{dx}{dt} = \frac{\mu_{\text{on}}}{WL} i_{\text{in}}(t), \]  

(3.41)
which sums up the relation between the design parameters and the behavior of the memristor.

**Amplitude characteristic**

Consider a sinusoidal current source driving the memristor, as in Fig. 3.9. Furthermore, let us assume that the state of the memristor can be described
3. Memristors

Figure 3.9: Circuit set up for amplitude characteristic. A memristor is driven by a sinusoidal current source with amplitude $I$ and frequency $f$.

by a full swing sinus between 0 and 1 with the frequency $f_0$, that is

$$x(t) = a_0 + a_1 \sin(2\pi f_0 t). \quad (3.42)$$

The amplitude of the oscillating state with period of $T = 1/f_0$ can be condensed to a root mean square (RMS) entity. Therefore it is appropriate to define

$$x_{\text{rms}} = \sqrt{\frac{1}{T} \int_0^T (x(t) - \bar{x})^2 \, dt}, \quad (3.43)$$

and $\bar{x}$ defines the mean value of the state over one period,

$$\bar{x} = \frac{1}{T} \int_0^T x(t) \, dt. \quad (3.44)$$

Now, we have a quantity to describe the change, or in other word the modulation of the state variable. A larger value means more modulation than a smaller value.

Maximum sinusoidal swing of $x(t)$ in (3.42) are given for $a_0 = 0.5$ and, $a_1 = 0.5$. In the circuit in Fig. 3.9, the amplitude $a_1$ will be proportional to the integrated current over a period for a fixed value of the current amplitude. At frequency $f_0$ the amplitude is exactly 0.5 and for higher frequencies it is below that. For lower frequencies we just, for now, assumes that $x(t)$ is clipped outside $[0, 1]$.

A numerical implementation of the state $x(t)$ over different frequencies was done in Matlab, where the result of $x_{\text{rms}}$ is plotted as dash lines in Fig. 3.10. In addition, a illustrative plot of the behavior of $x(t)$ versus frequency is also plotted. This plot should be interpreted as: at a given frequency the time domain curve would look like the plot in the neighborhood of that frequency, i.e. low frequencies causes a square wave like form of $x(t)$, and high frequencies generates a small amplitude but oscillating form on $x(t)$.

At the frequency $f_0$ the modulation is precisely between 0 and 1 and we define this as the cut-off frequency. To find the cut-off frequency for a given current amplitude, as defined in Fig. 3.9, we have

$$i_m(t) = I \sin \left( \frac{2\pi t}{T} \right). \quad (3.45)$$
If we for the sake of simplicity still ignore the window function, the current $i_m(t)$ will move the state from 0 to 1 during one half of its period and back in the other period, we can formulate this from equation (3.15) as

$$x_{p-p} = x \left( \frac{T}{2} \right) = \frac{I_m R_{on}}{D^2} \int_0^{T/2} \sin \left( \frac{2\pi}{T} t \right) dt = \frac{I_m R_{on}}{D^2 \pi} T = 1. \quad (3.46)$$

Setting $f_0 = 1/T$, gives the cut-off frequency as

$$f_0 = \frac{\mu R_{on} I}{D^2 \pi}. \quad (3.47)$$

Now we see that the cut-off frequency is linearly dependent to the amplitude of the current through the memristor as well for the on resistance $R_{on}$.

So, the expected behavior of the state variations for a memristor driven by a constant amplitude sinusoidal current source can be organized in three regions:

- **Digital modulation**: The state toggles between 0 and 1, and spends most of its time there, for frequencies bellow $f_0$. The value of the state transfer function is about $20 \log_{10} 0.5 \approx -6 \text{ dB}$.

- **Cut-off frequency**: The state oscillates exactly between 0 and 1. The modulation of the state is at the maximum without any clipping. The value of the state transfer function is about $20 \log_{10} (0.5/\sqrt{2}) \approx -9 \text{ dB}$.

- **Analog modulation**: The modulation is the same as at the cut-off frequency, but the amplitude decreases as the frequency increases. The roll-off should be about $-20 \text{ dB/decade}$.
3. Memristors

However, the use of a window function will cause distortion in these regions, by compressing the state space.

3.5 Simulation results

The linear model with Joglekar-Wolf and Biolek window functions is simulated with Verilog-A language with Spectre simulator in Cadence. In addition to the original window functions, the safe region is added to the models, to ensure stability and speed of the simulations.

How the values of the parameters in equation 3.38 are to be chosen depends on the physical implementation in the manufacturing process. In the simulations done here we simply assume some, for us, reasonable values of the parameters. The actual values does not change the fundamental behavior, but only shifts the result by orders of magnitude. In the simulations the mobility is set to $\mu = 1 \text{ nm}^2/\text{Vs}$. The device depth is set to $D = 10 \text{ nm}$. The values of $R_{\text{on}}$ and $R_{\text{off}}$ are chosen to be 10 kOhms and 110 kOhms respectively. The memristor is driven by a sinusoidal current source with $I = 10 \mu\text{A}$ at frequency $f$. To give a comparable results the Joglekar-Wolf $F_2(x)$ window is chosen to match the Biolek $G_4(x)$ window function. Finally, the safe region is chosen such that $x \in [0.001, 0.999]$.

Butterfly graphs

To get a grasp on how the memristor behaves, one of the most common ways is to plot the current through the memristor versus the voltage across it, as seen in Fig. 3.11. Depending on the frequency of the current source the state moves with different magnitude. We can group the different plots of Fig. 3.11 depending on their topology (and with a little bit of imagination) into: bent nails ($\circ$, $\times$), butterflies ($\Delta$, $\ast$), and finally, straight sticks ($+$).

In the first group, the bent nails, the state is pending at the borders for the most time of the period of the sinus. The resistance of the memristor is either $R_{\text{on}}$ or $R_{\text{off}}$ which causes the the two lines. Only a small fraction of the period is spent in the transition between the two states. This transition causes the small needle eyes ($\circ$). As the frequency increases moderately, and right initial condition is set to place the state properly to avoid pending. The fraction of spent time in the transition time between the borders increases, which causes the needle eyes to grow in size ($\times$).

In the second group, the butterflies, the state is in transition for most part of the period, and thus spends less time at the borders. This can be seen by the fact that the slope of the edges of the butterfly wings in $\Delta$ is the same as for the bent nails. By further increase of the frequency, even less time is spent close to the borders, as seen in $\ast$. Looking at the slope of the edges on this butterfly, we see that the memristor spends some part of the period at $R_{\text{on}}$ ($x$ close to, but less than 1), but never at $R_{\text{off}}$, which means that the state was initially set such the oscillation is asymmetric around $x = 0.5$. In fact, the initial state was indeed $x = 0.5$, but when applying a sinusoidal current,
the state wanders off to one side, since the current is positive in the first half period, and then returns in the second half period by the negative current. That means that the initial state is not the same as the average state for a memristor driven by a sinusoidal current. To achieve that, the source must be a cosinus.

Finally, at high frequencies well above the cut-off frequency, the state does not hardly move anything during one period (+). The wings of the butterfly has collapsed and all that remains is a stick close to an ordinary line as a normal resistor.

The lower right part of Fig. 3.11 shows the amplitude characteristic of the state in all five cases. As seen, for the lower frequencies the amplitude is at maximum since the state is most of the time at the borders. For the cut-off frequency ($\Delta$) the state amplitude characteristic has dropped about 3 dB compared to $\phi$. (Recall that the root mean square of a sine wave is $1/\sqrt{2} = -3$ dB of a square wave.) With further increased frequency the amplitude characteristic roll-off. The odd shape of the curve after cut-off frequency is discussed below.

The simulation results are also included as a flip-book animation in the lower left margin.

**Amplitude characteristic results**

In the linear drift memristor model described above, the amplitude characteristic is calculated from transient simulations. The initial conditions are set to $x_0 \in [0.01, 0.99]$, while the frequency is swept. Some of the simulations are not immediately in a steady state behavior the first couple of periods. An ocular inspection of the cycles of $x$ versus the current shows a converging behavior after about six to ten cycles, i.e. the plots of the cycles starts to overlay each other. Hence the eleventh period is chosen for the calculations of the amplitude characteristic. Of course other periods after the eleventh period can be used.

From (3.47), the cut-off frequency should be expected at

$$f_0 = \frac{1 \text{ nm}^2/\text{Vs} \times 10 \text{ kOhm} \times 10 \mu\text{A}}{\pi(10 \text{ nm})^2} = 0.32 \text{ MHz}. \quad (3.48)$$

However, the simulations shows that the actual cut-off frequency, where the amplitude characteristic has lost 3 dB, is about 0.22 MHz for the Joglekar-Wolf window function (with $x_0 = 0.5$). This discrepancy is annoying and not fully understood by the author, but assumed to be linked with the usage of window function which (3.47) does not consider.

In Fig. 3.12, the amplitude characteristic for Joglekar-Wolf window function is plotted over the frequencies $f \in [10^3, 10^6]$ Hz and for the initial condition $x_0$. The initial condition must be considered since it will also determine the amplitude due to the scaling by the window function. The shape of the result has a somewhat complicated structure. The basic shape is as expected, a flat surface for low frequencies and a roll-off region starting at the cut-off
Figure 3.11: Please see next page for caption.
3.5. Simulation results

Figure 3.11: Plots from simulation of the linear memristor model at five different frequencies. A sinusoidal current source with amplitude 10 µA drives a memristor and the current is plotted against the voltage across the memristor.

**Upper left** (•): $f_0 = 1$ kHz; the state of the memristor toggles between the two borders. Only a small fraction of the period is spent in transition.

**Upper middle** (×): $f_0 = 12.6$ kHz; Most of the period is spent at the borders. However, the fraction spent in transition has increased.

**Upper right** (∆): $f_0 = 178$ kHz; The state makes complete transition between the borders during one period. Only a small fraction of the period, if any, is spent in pending at the borders.

**Lower left** (◦): $f_0 = 316$ kHz; The state only moves between the initial condition and one border and the wings of the butterfly shrink.

**Lower middle** (+): $f_0 = 5.01$ MHz; Very small change in the state during one period. The memristor behaves like a resistor.

**Lower right:** The amplitude characteristic of the state versus frequency for the initial condition $x_0 = 0.5$. Corresponding points of the I-V plots marked with corresponding markers.

frequency, or in other words, the surface is folded down. However, there are two additional distortions of the surface.

The first distortion is seen at the cut-off frequency. We see a second fold which depends on the initial condition $x_0$ and this is as described above, due to the asymmetrical behavior between $x_0$ and the sinusoidal current source which causes the initial roll-off to be steeper than $-20$ dB/decade. If the polarity of the current source is reversed, the secondary fold gets mirrored in the plot along the $x_0$-axis.

The second distortion is the arc structure shown at the edge of high frequencies, as seen in Fig. 3.12. This is due to the pending occurring at the borders. The current over a half period is so small that the state never leaves the border.

The amplitude characteristic of the Biolek window of $G_4$ is found in Fig. 3.13. The roll-off at cut-off frequency is rather clear without any major distortion. The secondary folding effect is apparent to be gone. The only remaining distortion can be seen at the edge of the surface at high frequencies, where a small arc still remains. However, the smooth surface of amplitude characteristic is traded at the cost of losing the one-to-one mapping between charge and state.

The cut-off frequency for the Biolek window is about 280 kHz regardless of the initial state, which is closer to the theoretical cut-off frequency, but still a little bit away.
3. Memristors

Figure 3.12: Amplitude characteristic of a memristor with a Joglekar-Wolf window function. The initial state values are \( x_0 \in [0.01, 0.99] \). A secondary fold appears due to pending of the state sustained by the window function, but the mapping between charge and state is conserved.

Figure 3.13: Amplitude characteristic of a memristor with Biolek window function. The initial state values are \( x_0 \in [0.01, 0.99] \). The surface is smooth and has no secondary folding other than the expected roll-off, only a small arc can be seen versus \( x_0 \). However the one-to-one mapping of charge and state is lost.
3.6 Applications

Mazumder [62] nicely structure and order the different uses of memristor into a taxonomy. The first selection is based on whether the memristor is used as a discrete component in the circuit or it is connected in a crossbar array.

In classical terms, a discrete component is referred to a basic circuit element in a single package performing one operation. All of the device terminals are accessible on the package. In contrast, an IC contains a number of devices with interconnections between them. The circuit operation is more complex and only a few of the devices has terminals on the package.

In this case, the term “discrete” is used for all other applications than crossbar arrays. A crossbar circuit uses memristors as interconnects between a horizontal layer of wires and a vertical layer of wires as seen in Fig. 3.14, where a two by two array is illustrated. The second selection is base on whether the memristor(s) are used for analog or digital processing. Analog processing, makes use of the continuous function the state variable constitutes. In digital processing, the exact position of the state is less important, the only thing that matters is if the state is above a certain limit or below another limit, or in some cases, just above or below one limit.

Figure 3.14: The crossbar structure of memristors in a two by two array. The selected memristor is addressed by applying a voltage (or current) between one column metal wire and one row metal wire. The current (or voltage) relates to the addressed memristors state, which encodes the information. In these applications the number of rows and columns are large.
Discrete applications

Perhaps the most obvious application of the memristor is to use it as a memory cell. The state of the memristor can be used to code either one or more bits. However, to read out the state value, a current has to be applied, and this changes the state. To set things back, a reverse current can be applied, but due to uncertainties the state may drift away after a number of readouts. Today a common implementation of memories are done by storing a charge in a capacitor. The drawback of this implementation is leakage of the charge and hence the need of the memory to be constantly refreshed. By replacing the capacitor by a memristor, we can avoid the need to constantly refreshing. An secondary effect is that the memory is non-volatile.

The memristor has been proven to be able to perform logical operations [63–65]. This may very well be suitable, but maybe a more appealing implementation would be to store the state of each flip-flop in a digital circuit with a memristor [66]. By doing so, a computer would not be harmed by a sudden power failure. As soon as you unplug your computer it will freeze and continue at the exact instance when the power returns (“Instant computer”).

Since the value of memristance can be modified, we can use it as a programmable resistance in generalized analog circuits, such as [67], but also in digital field programable gate array (FPGA) [57, 65]. Similarly, an analog filter is defined by the topology and the coefficients of the components R, C, and L. With the memristor, we may design an analog filter which can be adjusted by the resistance similar to [68].

Another application of the memristor is the modeling of the cell membrane of neural cells. The membrane is permeable to Ca and K ions and can be modeled with an electrical circuit as suggested by Hodgkin and Huxley [69]. The model is especially appropriate for propagation of an action potential over the axon, where the wrapping of Schwann cells around the axon creates a lumped structure. Alan [70] and Sah et al. [71] showed that the memristor can be used in Hodgkin and Huxley’s electrical model of the cell membrane, as seen in Fig. 3.15, by replacing the $R_{Na}$ and $R_K$ with memristors with the right parameters. In real neuron cells, the cell membrane is a two dimensional surface where the permeability of sodium and potassium ions are distributed. The model above assumes that the complete cell membrane as one piece and has to be divided into smaller blocks, which gives a lumped model. This can now be used to simulate an action potential (a propagating wave over the axon).

Crossbar structure

Since the pitch of the elements in a crossbar structure, as seen in Fig. 3.14, is very much smaller than a traditional memory, the crossbar structure shows great potential for storing information. A memristor is about the same size as a diffusion metal contact at any terminal of a transistor. Furthermore, there is no need for a selection transistor which would take space. Furthermore,
3.6. Applications

Figure 3.15: The Hodgkin-Huxley model of the cell membrane. The capacitor models the charges on each side of the cell membrane. The branch of $E_{Na}$ and $R_{Na}$ models the potential of the sodium ions and their dedicated passage channels through the membrane. Similar branch is for the Potassium ions ($K$). The final branch models the leakage through membrane.

Strukov and Williams [44] showed that with the right alignment, a standard CMOS process will be capable of driving a large number (100) of crossbar structures of memristors. The expected capacity would be around 100 terabits per cm$^2$. However, there is a requirement on the memristor for the structure to work; the ratio $R_{off}/R_{on}$ must be high (about 1000) according to the delegates at 1st Symposium on memristors and memristive systems (UC Berkeley, 2008), which should be within grasp since Pickett (2009) reported a ratio of 500 [52]. If the ratio is too low, the current between the addressed row and column may leak through parasitic paths and affect other memristors too much. Another issue is the rather long writing time for a memristor memory, up to 250 ns compared to conventional dynamic random access memory (DRAM) of less than 0.5 ns [72].

In a traditional computer the data is stored in a memory, located elsewhere to the computing location, this means that the data has to travel over the bus before computation can occur. Some speed up can be done by caching, but does not give speed up in one hundred percent of the time. The stacked crossbar structure gives an interesting feature to compute and store the data at the same location [73]. We may see one layer of the crossbar as a matrix and apply values (voltages) on one metal layer with a digital to analog converter (DAC) and read the voltages with an analog to digital converter (ADC) on the other metal layer. This semi-analog computer stores and computes at the same location on the same devices, similar to neurons.

The semi-analog computer structure can be driven even further, towards emulation neurons in the brain. The memristors in the crossbar structure will in this application encode the strength of each receiving dendrite on one cell. Each column, or row depending on the orientation, will be a weighted sum of the inputs. The same summation operation is done over the cell membrane over the major body (soma) of a neuron. The output of each column can be sent to a amplifier that integrates and fires back to the corresponding row. A number of different implementations has been shown in [56, 58, 74, 75].
3. Memristors

It should be mentioned that there exists a patent on crossbar structures, which maybe need some consideration [76].

Maze solver

The memristor can be used to solve the shortest path, or actually most of the possible paths between two points in a maze [77–79]. A very simple maze can be found in Fig. 3.16, where the actual maze is defined by the wide light blue lines. The memristor network is built by two basic cells in a checker pattern. Each cell contains two memristors and two switches. The only difference between the two basic cells are the orientation of the memristors. The walls in the maze are defined by the switches, an open switch means there is a wall and the current can not pass through, while a closed switch means it can.

Now, let’s say we want to find the path between A and B. Let initially all memristors be in off-state, i.e. $R_{\text{off}}$. A positive voltage is then applied on the memristor terminal A and terminal B is grounded. The memristors in the closest path in the right direction will change state to a lower resistance. Alternating the polarity, terminal A is grounded and a positive voltage is applied to terminal B. This will cause the memristors in the closest path, but previously in the wrong direction to change state to a lower resistance. Further alternating repetition will cause all the memristors in the closest paths to reduce their resistance and eventually the closest path can be found by reading out the states.

The suggested structure [77] will find the paths between one pair of end points, the one-to-one, by toggling the polarity of the voltage at the end points.

Can the same structure be used to find paths between several pairs of end points, such as two one-to-one paths in the same maze? The toggling can be seen as a complementary code-set sent by the pair. Maybe with the appropriate selection of codes, only the memristors in the shortest paths between pairs of end points will change state.

Let us assume a simple example with desired connection between terminal A and B, and C and D. Assume the voltages of the endpoints to be time discrete sequences $A = x^1$, $B = x^2$, $C = x^3$, and $D = x^4$ where

\[
\begin{align*}
  x^1 &= x^1_1 x^1_2 x^1_3 x^1_4 x^1_5 \cdots \\
  x^2 &= x^2_1 x^2_2 x^2_3 x^2_4 x^2_5 \cdots \\
  x^3 &= x^3_1 x^3_2 x^3_3 x^3_4 x^3_5 \cdots \\
  x^4 &= x^4_1 x^4_2 x^4_3 x^4_4 x^4_5 \cdots ,
\end{align*}
\]

(3.49)

and where $x^i_j$ is a real number.
At the start all the memristors are in their off-state. Furthermore, the change of the memristors between point \(i\) and \(j\) can be assumed to be

\[
\frac{dx}{dt} \propto (x_i' - x_j'),
\]

(3.50)

where we have assumed that the change of the state is small for each time step, and thus the memristance only changes marginally.

Inside the paths in the pairs A-B and C-D, the memristors state have to change, which can be expressed as

\[
\sum_{n=1}^{N} x^1_n - x^2_n \neq 0
\]

\[
\sum_{n=1}^{N} x^3_n - x^4_n \neq 0.
\]

(3.51)

At the same time we must make sure that the memristors in the paths between pairs should not be affected. We do not want to detect any false paths. Hence we can express the requirement for paths A-C, A-D, B-C, and
3. Memristors

B-D, as the group of conditions

\[
\begin{align*}
\sum_{n=1}^{N} x_n^1 - x_n^3 &= 0 \\
\sum_{n=1}^{N} x_n^1 - x_n^4 &= 0 \\
\sum_{n=1}^{N} x_n^2 - x_n^3 &= 0 \\
\sum_{n=1}^{N} x_n^2 - x_n^4 &= 0.
\end{align*}
\] (3.52)

From this we can deduct

\[
\sum_{n=1}^{N} x_n^m = k_0 \quad m = 1, 2, 3, 4.
\] (3.53)

At the same time from (3.51), we have that

\[
\begin{align*}
\sum_{n=1}^{N} x_n^1 &\neq \sum_{n=1}^{N} x_n^2 \\
\sum_{n=1}^{N} x_n^3 &\neq \sum_{n=1}^{N} x_n^4,
\end{align*}
\] (3.54)

which leaves us with a contradiction. Hence linear equations can not be used to solve the maze problem for multiple pairs of connections, if we at the same time require that the parasitic paths should not be affected.

Furthermore, the one-to-one maze solver does require some kind of non-linearity, such as the Biolek window function, to get the state out of pending the off-state.

3.7 Conclusions

In this chapter we have looked at the memristor, or if you prefer “the missing link” between charge and flux. We have described a number of different models and some commonly related window functions.

Based on the linear drift model, we have elaborated upon the possible design space and showed that the state speed is proportional to the current and inversely proportional to all three dimensions, i.e. width, height, and length. Furthermore, the memristor frequency behavior has been characterized with the introduced amplitude characteristic function. Hopefully this will give a clearer view to the reader of how the memristor works in a system. The memristor is a frequency dependent component with a cut-off frequency which is dependent on current, process and dimensions. On the margin of this chapter there is a flip-book animation of the frequency dependency.

In this work we have also briefly touched on some of the numerous possible applications the memristor could be used in. Everything from data storing by digital memory, to store-and-compute structures such as neural networks, to the emulation of the cell membrane functionality.

Finally, the memristor can be used to solve complex problems such as finding the shortest way between two points in a maze, i.e. the one-to-one...
problem. However, the linear model is not adequate to solve two simultaneous one-to-one problems in the same maze, due to the parasitic paths. The memristor is a tricky two-terminal frequency component, with great potential for storing and computing.
Chapter 4
Miscellaneous

Our brain consists of a huge number of neural cells (neurons) and they are the reason you can think and reflect. Today, we can mimic some simpler neural and adapting systems. But the whole brain is still to be understood.

4.1 Introduction

You may sometime wonder; what is intelligence? There is no entirely obvious or clear answer to that question, and neither are we going to answer it here. But the most common explanation is something like this; Intelligence is the ability to store and use information to predict future events (or at least the probability of the events).

According to that definition the computer that defeated Kasparov in chess, Deep Blue\(^1\), would be intelligent. Similar to the victorious computer (cluster), Watson would be considered intelligent. However, if you ask those computers to solve another task, they would most probably perform poorly. They only excel in one task. Actually, even a control system for a missile may appear to be intelligent in its hunt for a fleeing target [8].

As Dawkins brutally states, intelligence is a bi-product in the strive for the genes to survive [8]. This is a rather sobering insight. Furthermore, an interpretation of Dawkins [8] book could be; you are merely a survival machine for a set of genes, that happens to assert the most stable configuration, given the context.

The human brain consists of neurons, each connected to other neighboring neurons, but also to neurons far away.

As seen in Fig. 4.1 the main blocks of a neuron are: the soma, dendrites, axon, and axon terminal. The connection between neurons are electrochemically in small gaps between the axon terminal and dendrites. The axon terminal transmits to the dendrites. The dendrites are located on the soma (body) of the neuron and all the signals from the dendrites are summarized over the surface of the soma. If the sum of all signals are above a threshold at the axon hillock, an action potential (a spike) travels along the axon to the axon terminal via other neurons. [25]

Figure 4.1: Basic topology and functions of a neuron. Input signals to the neuron are applied at the dendrites, where a chemical transfer signal (neurotransmitter) is converted to an electrical potential. All input potentials are summarized over the soma. If the potential at the axon hillock is above a threshold, this causes a wave of potential (also known as a action potential) to propagate over the axon. Finally the propagating action potential reaches the the axon terminals, where the electrical potential is converted to chemical transmitters to dendrites of other neurons or cells. The axon may be warped by Schwann cells, which causes the action potential to jump between the nodes of Ranvier and thus propagates faster and more energy efficient.

During development of the brain, the network of neurons are formed opportunistically. The brain seems to be very resilient. Even if you kill some thousands of brain cells by a really wet night at the pub, you will be able to function to some extent the next day.

In this chapter we will look on some imitations of neural networks.

**Purpose**

The purpose is to get an understanding of how neural networks operate and their possible applications for electronic systems. Especially, if they are applicable for self organization in ECA or similar structures.

**Demarcation**

This work touches the field of ANN and WSN, with no deeper mathematical investigation in regards of, for instance, sets, stability and so on.

**Chapter overview**

In this chapter we will cover two different ANN examples, the restricted Boltzmann machine (RBM) and the Markov chain network. Finally we will look into an WSN example where a malicious radio tries to disturb our network.
4.2 Artificial neural networks

Background

Almost any time you discuss the topic of nature-inspired electronics or computing, the function and properties of the neuron will appear in the discussion. Most people would agree that the function of the human brain in some respect is involved in the appearance of our intelligence. Hence, the field of ANN seems to be an obvious area to familiarize oneself with before trying to build intelligent electronic systems.

4.2 Artificial neural networks

Artificial neural networks (ANN) is one of the collective terms for mimicking the behavior of neurons. The degree of detail differs a lot, all from simple binary networks such as Hopper field networks [26] to ambitious full detail simulations such as the Blue brain project [80]. In addition there is a lot of research going on to understand the brain such as the Human brain project [81].

ANN can learn by regression of the data (to be learned). The update of network coefficients, i.e. the connection strength between the neurons, can be done by many different methods, among them are the Hebbian learning and the delta rule.

Hebbian learning is based on biology and can be delightfully explained by the famous experiment Pavlov did on dogs. In short, the experiment goes like this. Dogs that see food, have an increased tendency to drool. Ringing a bell in the presence of the dog without any food present, does not change its tendency to drool. However, if the bell rings, every time the dog sees food, there will be an association between food and the bell. Eventually, only the bell is enough to make the dog to have an increased tendency to drool. On a neuronal level, the connection strength between neurons for drool and the bell are getting stronger in the simultaneous of exposure to food and bell. Eventually the bell neurons are enough to signal to the drool neuron. [26]

The delta rule simply uses the difference of what the ANN produces and the expected correct result, i.e. the training data, and updates the connection strengths in network by a quantity corresponding to a fraction of the difference. [26]

ANN can be used in many different applications, in [82] the network is used for distance estimation in images of a stereoscopic feed of frames.

The image feed from each camera is processed one frame at the time with a set of electronic neurons. A sub-window is swept over the frame in both horizontal and vertical position. The image data from the sub-window is applied to the neurons, which each and every one, calculate the sum of the absolute distance between the input data and previously learned data. The label of the neuron with the lowest sum, shows what have been found, and the position of the sub-window where it has been found. The processing is done on both cameras frame and the difference in position is used to calculate the distance to the object.
Figure 4.2: The structure of a restricted Boltzmann machine. The input data is presented at the visible nodes \( v_0 : v_n \), which are connected to the hidden nodes \( h_0 : h_m \) by the connection matrix \( W \).

**Restricted Boltzmann machine**

The connections and nodes in an restricted Boltzmann machine (RBM) are shown in Fig. 4.2. The RBM is a reduced form of the normal Boltzmann machine. Connections are only allowed between visible and hidden nodes. No connections are present between any two hidden nodes or any two visible nodes. [83]

The nodes \( v_i \) and \( h_i \) are binary and the connection matrix \( W \) is initialized with random numbers, preferably normal distributed with a variance of about 0.01. Please note that \( v_0 \) and \( h_0 \) are constant ones, 1, to feature bias terms.

The connections between the visible nodes and hidden nodes are indirect by probability functions. The probability that the hidden node \( h_i \) is set to one at time instance \( k \) is given by

\[
Pr(h^k_i = 1) = \frac{1}{1 + \exp(-\sum_{j=0}^{n} W_{ij} v^k_j)}.
\]  

(4.1)

And similarly, the probability of node \( v_i \) to be set to one, 1, at time instance \( k + 1 \) is

\[
Pr(v^{k+1}_j = 1) = \frac{1}{1 + \exp(-\sum_{i=0}^{m} W_{ij} h^k_i)}.
\]  

(4.2)

A visualization of an execution of a RBM is shown in Fig. 4.3. At the first time instance, the input data is presented at the visible nodes. The information is then transferred via the connections to the hidden nodes. From the hidden nodes, the information is transferred back to the visible nodes.
4.2. Artificial neural networks

Figure 4.3: The Boltzmann machine operates by using the input data to set the visible nodes \( (v_j^1) \), and then generates the state of the hidden nodes \( (h_i^1) \), which in turn generate new visible nodes \( (v_j^2) \) and so on. The movement of data from the visible nodes to the hidden nodes and back again is called a shake and is done a predefined number of steps. At the end, the final shake presents the result from the network, the belief.

The procedure of transferring information from the visible nodes to the hidden and back again is called “a shake”. After a number of shakes the model belief can be found at the visible nodes. The number of shakes is a parameter of appropriate design selection between one and infinity. Learning is done by grouping data, in any order, into batches and update the connection strengths according to

\[
\Delta W_{ij} = <v_j^1 h_i^1> - <v_j^0 h_i^0>
\] (4.3)

and

\[
W = W + \alpha \Delta W,
\] (4.4)

where \(<x>\) denotes the mean of \(x_n\) in a batch.

The (4.3) can be seen as the difference between real data and what the model “believes”. A common nomenclature is \(\Delta W_{ij} = <v_j^1 h_i^1>_{\text{data}} - <v_j^0 h_i^0>_{\text{model}}\). Another common practice is to use the same variable for visible and hidden nodes, that is \(\Delta W_{ij} = <s_j^1 s_i^1>_{\text{data}} - <s_j^0 s_i^0>_{\text{model}}\), which is really confusing when entering the field. As seen, the RBM uses both Hebbian learning and the delta rule.

From a biological point of view, the RBM is not plausible, since each pair of hidden and visible nodes uses the same weight (strength).

**Simple vector recognition example**

A restricted Boltzmann machine of 16 visible nodes and five hidden nodes is trained on four different symbols, as seen in Table 4.1, where each symbol have all bits in a nibble group set to one. The restricted Boltzmann machine is trained in batches of 500 samples each in 1000 steps, which may have been a little too extensive in retrospect. The trained RBM is then presented with data...
to reconstruct. In our example, all piece of presented data is reconstructed 500 times and averaged. This may also have been a little too extensive. The training data vectors, the vectors to reconstruct and the reconstructed vectors can be found in Fig. 4.4, where each pixel column represents a vector. The network seems to reconstruct the missing data quite well.

**Image reconstruction example**

The restricted Boltzmann machine can also be used to store and reconstruct images. To provide information about location in the image to the network of nodes, two sets of potentials are used; one set of 20 potentials for the x-direction and one set of another 20 potentials for the y-direction.

\[
pot_i(x) = \frac{1}{1 + \exp(-k(x - x_i))}.
\]

(4.5)

The parameter \(k\) determines the sharpness and \(x_i\) the location of each potential. Image data and potentials are then associated together in one vector

\[
\text{data} = [R][G][B][\text{pot}_1(x) \ldots \text{pot}_n(x)][\text{pot}_1(y) \ldots \text{pot}_n(y)].
\]

(4.6)

Training of the network is done with the image seen in Fig. 4.5, which has the size of 30 by 30 pixels, and is performed in 500 batches of 500 vectors each. Reconstruction is performed in 128 batches, each containing 40 data points. The reconstructed image can be found in Fig. 4.6, where fuzziness and minor deformation can be seen.

To get a neural network to learn and reconstruct data often requires some kind of tweaking and testing to get reasonable results. There is always some parameter that has to be adjusted.

In the example above, there is only one pixel associated with each input vector. This can, quite possibly, be increased to include neighboring pixels as well. But then, the initial values of the reconstructed image, must most probably be considered, i.e., initial the image is mid gray, to make sure of proper converging.

**Markov-lyrics**

Let us assume that we have some kind of sequenced data of a specific kind, but there is not enough for our particular application. Hence, we would

<table>
<thead>
<tr>
<th>Nibbles</th>
<th>(v_1 : v_4)</th>
<th>(v_5 : v_8)</th>
<th>(v_9 : v_{12})</th>
<th>(v_{13} : v_{16})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vector1</td>
<td>1111</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>Vector2</td>
<td>0000</td>
<td>1111</td>
<td>0000</td>
<td>0000</td>
</tr>
<tr>
<td>Vector3</td>
<td>0000</td>
<td>0000</td>
<td>1111</td>
<td>0000</td>
</tr>
<tr>
<td>Vector4</td>
<td>0000</td>
<td>0000</td>
<td>0000</td>
<td>1111</td>
</tr>
</tbody>
</table>

Table 4.1: The training data for a restricted Boltzmann machine in the simple vector recognition example.
Figure 4.4: The training, test, and reconstructed data of a restricted Boltzmann machine. **Top:** The training data, each pixel column represents one vector. **Middle:** The test data to be reconstructed. **Bottom:** The resulting reconstructed data according to test data and the knowledge stored in the network.
Figure 4.5: The original image which the neural network is trained on. In addition to the image data, two sets of potentials, depending on the pixel location, are used to form the input vector.

like to find a method to generate more data with similar properties. Then a Markov model could be of use.

From a set of data we train the model by looking at short, fixed size, pieces of data. The model assumes that the symbols before the last symbol (or so) implies which the last symbol can be in that piece, i.e., the last symbol in the piece correlates with the previous symbol in the piece. As seen in the example in Fig. 4.7, looking at the piece “lo wo” in “Hello world” tells the model that there is a probability of a ‘o’ after the sequence “lo w”.

The next piece is generated by increasing $n$ by one, and now the model learns that there is an increased probability of having an ‘r’ after “o wo”.

One way to implement this model is to use a tree structure as in Fig. 4.8. Each letter in the current piece points at a branch for each level. At the end, leaves counters keep track of the occurrence of the different symbols. After the last piece has been presented to the model, each leaf can be assigned a probability linear to the value of the counter of each leaf compared to the sum of all the counters of leaves in that subtree, or in mathematical terms

$$
Pr\{S_n = s_n | S_{n-1} = s_{n-1}, ..., S_{n-m} = s_{n-m}\}. \tag{4.7}
$$

That is, given the sequence of previous symbols, what is the probability of the next symbol?
Figure 4.6: A reconstructed image from the network. Given the two sets of potentials, in x-direction and y-direction, each pixel is reconstructed by the restricted Boltzmann machine to complete an image similar to the original in Fig. 4.5.

Figure 4.7: In the learning of the Markov model, a piece is selected from the data and the occurrence of that particular symbol combination is learned. Then, another piece is selected and learned and so on.
To generate data from the Markov tree, we need to start with an initial seed, which can be either the first symbols or any piece of the data learned, or any random path in the tree. The seed cannot be completely randomly generated directly since there is nothing to guarantee that this particular symbol sequence exists in the model.

Given the seed, which is now pointing out a subtree, a new symbol is generated randomly given the different probabilities in that subtree, and that symbol is added to the output, as seen in Fig. 4.9.

To generate the next symbol, the last symbols of the output now points out another subtree which generates the next symbol. In this manner the model can produce any desired amount of data.

The size of the pieces that the model work with determines how random the output will be compared to the input data of the model. If the model work with very short pieces there is not enough information to capture the behavior of the data and the model only produces almost random noise. If the piece length is too long, the subtrees becomes more deterministic, since only one or a few leaves in the subtree has a probability higher than zero. In the absolute worst case, only the input data itself come out of the model.

As an example, we used a major part of the Swedish lyrics from the music group Kent as input. The piece length was chosen to six letters, which gave
4.3 Wireless sensor networks

A WSN can be any network of sensors that is communicating wireless. However, here we focus on a subset. Smart dust is the term for an WSN with physical small nodes, also referred as motes, which are distributed in wast volumes.

The motes are spread out over the area of interest, as seen in Fig. 4.10, where one base station collects the data from the network. The placement can either be deliberate, at exact positions, or random. When the motes are placed deliberately, the location of each mote can be specified exactly. When placing the motes randomly, such as sprinkled from an air vessel, the locations can not be determined and the network must therefore self organize its structure. The sprinkle technique should be an effective way to cover large areas fast.

Each mote consist of at least four parts: Energy storage, signal receiver/transmitter in any media (radio or laser) [84], one measurement

www.universalmusic.se, accessed 2016-01-22
Figure 4.11: A system block diagram of a WSN mote. A mote must have at least one function block in each category including: energy (green), communication (blue), measurement (beige-orange), and control (gray).

The application fields for a mote network are probably many more than we can imagine, but suggested applications have been: surface exploration of other planets [85], drought warning, monitoring of NO\textsubscript{x}-concentrations in crop fields, monitor the amount of pedestrians or animals in an area [86], border surveillance, finding missing/lost people in the woods and fire alarm over large woods.

The mote may behave differently depending on the available energy in the storage and/or harvester. In the case where a lot of energy is available the mote can do more measurements, act more frequently as a relay station for messages from other motes and so on. In the case of low available energy the mote will be more restrictive in dispatching measurements and messages. All to sustain the network for as long as possible.

Since the network of motes is a swarm network solution, where any mote may sleep or be disconnected indefinitely (i.e. dead or stolen by thieves), the intelligence must originate from the network itself.

One way of creating intelligence is to reduce the information. The information reduction can be done by simple rules such as majority voting with a certain threshold. For instance, if a mote receives messages from three other motes that there is a fire going on, it should start sending the message of fire. This will cause an avalanche of messages in the network and the information will soon reach the base station for further dispatch to the fire department.

A slightly more intelligent way of processing data would be to extract virtual objects and send information of those instead. For instance, a front...
4.3. Wireless sensor networks

of drought, which propagates through the landscape, can be summarized by the properties of location, speed, direction, and width. The motes cluster themselves to one or many of those virtual objects and only one or a few messages are sent towards the base station.

Organization

So how should the network organize itself to arrange a message to go towards the base station and how should one make sure that the motes closest to the base station do not suffer from energy depletion?

Well, for the organization part, the suggestion is what is called directed diffusion [86, 87] or the waterfill model. The principle is very simple. In an area with motes a base station broadcasts its presence with a zero, 0, level identification. All motes that manage to communicate in both directions with the base station will acquire level 1. All motes that can communicate in both directions with motes on level 1, acquire level 2, and so on. For a mote network on a grid, with indices $i$ and $j$, this can be expressed as

$$ pot_{i,j} = \min_{m=i+\{-1,0,1\},n=j+\{-1,0,1\}} (pot_{m,n}) + 1. $$

Eventually all motes that can communicate, are attached to the network. Messages in the network will now flow, like water down the slope of levels, or directed by diffusion, towards the base station.

For a system designer of a such network, the hard question will be, how often the network should reassign the level of each mote? This will be determined by the frequency of mote losses and the degree of overlap in their communication radii.

To reduce the total energy consumption in a WSN the motes in the network may be assigned different roles in a hierarchical substructure. For example as in [88, 89], the motes are gathered into small local clusters. In each cluster there is one controlling mote that is handling all the routing. In assistance, there are relaying motes to communicate with controlling motes in other clusters. All other motes communicate only with the controlling mote in the cluster. Since some motes in the cluster are doing more communication than others, they may suffer from energy depletion, and then other motes must step in and take over their role.

According to the classification in [90], the WSN described above is a self organizing network (SON) with dynamic network protocol with multiple hop (multihop) communication.

In the work of [90], a small network is simulated in the programming language Erlang. The communication protocol and behavior of each mote where controlled by a finite state machine. In addition a energy model of the mote where also included. The work also acknowledge the major issue of limited energy available for each mote, in the range of a few Joules per day, compared to the energy needed to fulfill spectrum requirement in radio broadcasting.
4. MISCELLANEOUS

Figure 4.12: A small wireless sensor network with two base stations. The motes can have a potential to each base station and thus some spatial information can be deduced.

Networks with multiple base stations

With the water fill/directed diffusion approach, each mote will get an identification by the levels to the base station. This identification also tells us a little bit about the geographical distance to the base station, assuming all motes are similar and evenly distributed. By adding identification of the base station and second base station at another location, as seen in Fig. 4.12 each mote will have two address codes, one to each base station. This may be mapped to geographical points, if we assume that the motes are somewhat equally distributed over the area.

The additional base station also provides protection against hijacking of the network. For a single base station network, a hostile takeover can simply be done by adding a malicious base station. This will rearrange the network to send all messages to the false base station.

If the network contains two or more base stations and the motes send messages to them alternate or randomly, a malicious base station will only partly reduce the success rate of passing the information.

Equalization

In the simplest form of directed diffusion, no concern is taken to the energy levels of the battery of each mote. The algorithm is greedy and uses the shortest way to the sink, and hence the motes in the shortest path may die due to depletion of energy. A way to overcome this is to use the current available energy in each mote and select an appropriate alternative path, as suggested
4.3. Wireless sensor networks

by [91, 92] which they call aware adaptive directed diffusion (AADD). The problem with AADD is the need for central optimization.

Instead, the (4.8) can be adapted by considering the energy level available in the mote

\[ pot_{ij} = \min_{m=i+\{-1,0,1\},n=j+\{-1,0,1\}} (pot_{m,n}) + 1 + \exp \left( \frac{E_{max} - E}{\lambda E} \right) , \] (4.9)

where \( E_{max} \) is the maximum energy storage capacity, \( E \) the current energy stored, and \( \lambda E \) a design parameter.

By doing this, drained motes will be higher up in the diffusion potential and the network will restructure to create other paths for the messages.

Another possible adaptation is to monitor the potential values of the mote neighbors, and try to adapt to be in the middle; similar to how birds and fish try to be in the middle of their flock/shoal.

However, such mid way adaptation must have an adaptation gain less than one to ensure stability, for instance

\[ pot_{ij} = \min_{m=i+\{-1,0,1\},n=j+\{-1,0,1\}} (pot_{m,n}) + 1 + \exp \left( \frac{E_{max} - E}{\lambda E} \right) + \frac{w \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} pot_{m,n} / 9,} {w \sum_{m=i-1}^{i+1} \sum_{n=j-1}^{j+1} pot_{m,n} / 9}, \] (4.10)

where \( w << 1 \).

**Jamming of wireless sensor networks**

As an implementation example we assume a quadratic field of one square km. The motes are placed in a 5 meters grid and the base station is placed as in Fig. 4.13. An interfering signal source, is also placed in the field, trying to jam the signals in the network.

The motes self organize according to directed diffusion by (4.8) and the different potentials of the motes can be seen in Fig. 4.14. Motes close to the base station or with low potential are denoted in black. Motes further away with high potential are denoted with white. The intermediate motes are denoted with gray.

**Link budget**

To understand how a jamming source degrades on the signal between motes, we need to look at the energy in the link budget.

We make the following assumptions:

- The allowed frequency span is 2300-2450 MHz, and output power of the antenna is 1 mW.
- Each channel is 25 kHz wide.
- The distance between motes is 5 m, in grid pattern.
- Motes only communicate with its eight closest neighbors.
The jamming power is 200 W, spread through the whole spectrum. The power into each receiver can be calculated by Friis’ equation [93, 94] by

\[ P_{rx} = \frac{P_{tx} G_{tx} G_{rx}}{L_{tr}}, \quad (4.11) \]

where \( P_{tx} \) is the output power of the transmit amplifier, \( G_{tx} \) is the antenna gain in transmitter and \( G_{rx} \) is the receiver antenna gain. The propagation loss is calculated by

\[ L = \frac{(4\pi)^2}{\lambda}, \quad (4.12) \]

where \( \lambda = c/f \).

If we now look at the situation where one mote is transmitting to another mote, while at the same time, the jamming source sends its signal, as seen in Fig. 4.15. The revived power from the mote is

\[ P_{mote,in} = \frac{P_{mote, sent} G_{tx} G_{rt}}{L_{mote,mote}}, \quad (4.13) \]

while the transmitted power from the jamming source is

\[ P_{jam,in} = \frac{P_{jam, sent} G_{jx} G_{rt} \Delta Ch}{L_{jam,mote} \Delta B}, \quad (4.14) \]
4.3. Wireless sensor networks

Figure 4.14: **Left:** Organization example of a grid of motes. Each pixel represents a mote and at the center left the sink is located. **Right:** Color encoding of potentials, from lowest (black) to highest (white).

where $\Delta B$ is the frequency bandwidth of the jamming source, and $\Delta Ch$ is the bandwidth of one channel. The mote only uses one of the available channels.

Assuming a bit error rate of $10^{-3}$ the signal to noise ratio should be about 20 dB or higher [95], and therefore we set $SNR_{min} = 20$ dB. Now, similar to how the equalization consider stored energy, we may form an expression that consider the signal to noise conditions, that is

$$pot_{ij} = \min_{m=i\pm1, n=j\pm1} (pot_{m,n}) + 1 + \exp\left(-\frac{P_{mote, in}(dB) - P_{jam, in}(dB) - SNR_{min}}{\lambda_{SNR}}\right),$$

where $\lambda_{SNR}$ determine the sharpness of the function and have to be chosen by the designer. The associated cost is the need for a signal to noise estimation in each mote.

In Fig. 4.16 the potentials of the motes in the network are shown. The white circle is the motes the jamming source has managed to knock out of the network. The parameter $\lambda_{SNR}$ has been set to 0.1, which gives a rather sharp and distinct transition between non-jammed and jammed motes. We also see in Fig. 4.16 how the network adapts and wraps around the jammed area, at the cost of having the upper right corner motes at a higher potential.
Figure 4.15: Setup to calculate Friis’ equations for the signals mote to mote and jamming source to mote.

Figure 4.16: **Left:** The network of motes is jammed by a hostile signal. The jamming source can only disable motes within the region where it can interfere enough with the signals of neighboring motes (the white circle). The rest of the network realign itself. **Right:** Color encoding of potentials, from lowest (black) to highest (white).
As seen above, a WSN, which uses directed diffusion, can adapt to changes in the environment such as energy levels and as in the example, jamming of the network. The adaptation time does of course depend on the size of the network and how often energy is spent on communicating the potentials.

4.4 Conclusions

We have looked at some systems that mimic the behavior of neurons or neural systems. There are many different types and applications for ANNs and we have only looked at a few.

The RBM system can learn a pattern and reproduce it, even if there are some errors in the input. Furthermore, the RBM system can be used together with potentials to store organization of cells. We could use this, to organize the cells of an ECA which implement the clay system of the redundant FIR filter, described in chapter 2. Furthermore, the RBM is basically opportunistic since the initial connection matrix is randomized.

A neural network with lowest sum neurons where applied in image recognition and distance estimation of stereoscopic image streams. Furthermore, any ANN can be implemented with the memristors described in chapter 3, since a crossbar structure in principle is a connection matrix.

We have also seen how a Markov system can be used to mimic patterns in sequential data such as lyrics. Although the produced lyrics can not be found the original lyrics, they appear to have the same pathos. The Markov system offers a simple way to generate abundant amount of data with a particular inherent property, for testing systems (like the WSN).

Finally, we have seen how a WSN system can self organize by simple rules and adapt to the given context, in our case a malicious radio broadcast aimed to jam the system. The system behaves like clay, since it consist of wast degree of redundancy of unified building blocks, i.e. the motes. Furthermore, each mote is nameless until engaged to the network and assigned a potential.

A mote can be seen as a neuron. A computing device that communicate with other neurons. Possibly can a mote simulate several neurons. The network can be formed opportunistically in the same way as the neurons connects described in thesis intro. Both ANN and WSN are redundant systems of unified subsystems, i.e. clay, where losses of subsystems is not a catastrophic failure.
Chapter 5
Discussion

The power laws discussed in the introduction will live on. Exactly how long Moore’s law will apply, only the future can tell. One thing is for sure, the current form will end and something else will take over. The size of the atoms is a hard limit with today’s technologies. Development of electronics will continue, but in a different way. One way could be the one described in this thesis. We may very well see self organizing cellular based structures that can cope with the errors induced by very small devices, which induces high deviations and reliability issues to such extent that current error correction strategies are not possible.

First, such a cellular array structure has to be organized by any means. That can be done with the strategy laid out in chapter 1, with successive division of the array together with internal state machines inside each cell, and at least one potential gradient. Another way is to use a neural network in each cell together with a higher number of potential gradients, such as described in chapter 4. However, instead of determine the color, the desired cell function at each location can be selected. One may also ponder the idea of combining both methods.

Second, since there is an error probability, we have to use redundancy to ensure sufficient functionality. As we saw in chapter 2, it is possible to design simple mean voter systems that do perform, but not as excellent as a major voter system does.

Third, once the cell functionality is decided in the cellular array, they have to find paths for communication among themselves. In chapter 3, we elaborated on the possibility to use the memristor in solving a maze problem. That does work fine as long as there is only one path to be found. However, in real applications there is strong reason to believe that several paths have to be solved simultaneously. There may be a way to do this, but here we didn’t find it.

The memristor shows interesting properties. This device is frequency dependent for a fixed current amplitude. Since it is a two terminal device, both the input and the output have to be applied to the same pins, which makes the device tricky to use in circuits. However, the device does have great
5. DISCUSSION

potential to be the next memory solution, due to the small feature size and ability to retain the content without power (non-volatile).

Beside the plasticity of the neural networks, we have also looked briefly into the plasticity of wireless sensor networks, where we saw the network ability to reshape and realign due to the context of the network and energy situation. Finally, the patos of Kent lyrics was captured in a Markov model, which where able to generate new lyrics.

The combined topics above may very well play part in the future development of electronics. So, Moore’s law will continue. However, the exponential growth of future systems will not mainly rely on the shrinkage of the transistors and other devices. Instead the growth will build on usage of abundance of subsystem, where the cost of each individual system is reduced.

Future electronics will be like clay since it will be abundant and redundant, due to the error probability Unified cells may overtake each other place, like the smart dust WSN and somewhat the redundant FIR filter. Future electronics will be like clay since it will be opportunistic, adapting and plastic. Forming paths or adjusting strength in connections as the environment changes, both the internal environment, such as which nodes or PEs are working, and the external environment, such as the available energy or the input data or any possible competition for resources.

As errors occurs on the system, we will have a softly degrading system and may have to accept loss in performance or accuracy.

The exact location of any function will be unknown to the designer, and the organization will be autonomous based on any combination of potentials, neural networks, or sequencing schemes in the same way as egg turn into a fetus. In fact, the transition to clay electronics has already started to some extent. Today’s super computer does have a clay like appearance from a programmers perspective. The computer is built by a large number of smaller computers, interchangeable to each other. The programmer does not know exactly which computer is going to do which calculation, or which node is going to be the main node. All the programmer, or exeriser of the program, knows, is the number of nodes.
Bibliography


Moore’s law has until today mostly relied on shrinkage of the size of the devices in integrated circuits. However, soon the granularity of the atoms will set a limit together with increased error probability of the devices. How can Moore’s law continue in the future? To overcome the increased error rate, we need to introduce redundancy. Applying methods from biology may be a way forward, using some of the strategies that transforms an egg into a fetus, but with electronic cells.

A redundant system is less sensitive to failing components. We define electronic clay as a massive redundancy system of interchangeable and unified subsystems. We show how a mean voter, which is simpler than a majority voter, impact a redundant system and how optimization can be formalized to minimize the impact of failing subsystems. The performance at given yield can be estimated with a first order model, without the need for Monte-Carlo simulations. The methods are applied and verified on a redundant finite-impulse response filter.

The elementary circuit behavior of the memristor, “the missing circuit element”, is investigated for fundamental understanding and how it can be used in applications. Different available simulation models are presented and the linear drift model is simulated with Joglekar-Wolf and Biolek window functions. Driven by a sinusoidal current, the memristor is a frequency dependent component with a cut-off frequency. The memristor can be densely packed and used in structures that both stores and compute in the same circuit, as neurons do. Surrounding circuit has to affect (write) and react (read) to the memristor with the same two terminals.

We looked at artificial neural network for pattern recognition, but also for self organization in electronic cell array. Finally we look at wireless sensor network and how such system can adopt to the environment. This is also a massive redundant clay-like system.

Future electronic systems will be massively redundant and adaptive. Moore’s law will continue, not based on shrinking device sizes, but on cheaper, numerous, unified and interchangeable subsystems.