THE EVOLUTION OF ROBUST, REVERSIBLE, NANO-SCALE, FEMTO-SECOND-SWITCHING CIRCUITS

Hugo de GARIS[1,2] Thayne BATTY[1]

[1] Brain Builder Group, Computer Science Dept.,
[2] (Theoretical) Physics Dept.,
Utah State University, Logan, Utah, 84322, USA.
tel. +1 435 797 0959, degaris@cs.usu.edu, tsbatty@cc.usu.edu
http://www.cs.usu.edu/~degaris

Abstract

This paper introduces conceptual problems that will arise in the next 10-20 years as electronic circuits reach nanometer scale, i.e. the size of molecules. Such circuits will be impossible to make perfectly, due to the inevitable fabrication faults in chips with an Avogadro number of components. Hence they will need to be constructed so that they are robust to faults. They will also need to be (as far as possible) reversible circuits, to avoid the heat dissipation problem if bits of information are routinely wiped out during the computational process. They will also have to be local if the switching times reach femto-seconds, which is possible now with quantum optics. This paper discusses some of the conceptual issues involved in trying to build circuits that satisfy all these criteria, i.e. that they are ***robust***, ***reversible*** *and* ***local***. *We propose an evolutionary engineering based model that meets all these criteria, and provide some experimental results to justify it.*

1. Introduction

In 2002, the first author and others published a paper [deGaris et al 2002] that attempted to evolve a Boolean network that was reversible, so that if it were implemented at molecular scales, it would be less likely to suffer from the classic heat dissipation problem created when circuits routinely wipe out bits of information when computing irreversibly [Landauer 1963], [Bennet 1971]. We evolved Boolean circuits that were reversible and capable of performing useful functions. However, this paper lacked a vital component. It unconsciously assumed that the circuits implementing these reversible Boolean networks were perfectly fabricable. This paper attempts to address this shortcoming by proposing an evolutionary engineering based model that is not only nano-scale based, and reversible, but is also robust, i.e. that it still functions when the inevitable faulty electronic components are present in the nano circuits. The ideas that serve as the basis of this new model are tested in simulation and the results are presented in this paper as well.

The contents of the remainder of this paper are as follows. Section 2 is a discussion of the conceptual problems involved in creating nano-scale, local, reversible, robust electronic circuits. Section 3 presents a brief summary of our previous work on the RENN (Reversible Evolvable Neural Network) model [de Garis et al 2002] that serves as the basis of our new robust version of the same ideas. Section 4 presents our new RENARO (Reversible Evolvable, Nano-scale, Robust) model. Section 5 summarizes experimental simulation results of this RENARO model. Section 6 summarizes, and poses some ideas for future work.

2. Conceptual Issues Facing the Construction of *Reversible, Evolvable, Nano-scale, Robust,* Electronic Circuits

This section discusses conceptual issues that will arise when attempting to construct nano-scale electronic circuits that switch in femto-seconds, that are robust to fabrication faults, and that are reversible, so as to overcome the heat dissipation problem inherent with 3D irreversible circuits.

We discuss these topics in the following order –

a) Femto-Second Switching
b) Evolvable Circuits
c) Reversible Circuits
d) Robust Circuits

a) Femto-Second Switching

Moore's Law states that the capacity (e.g. component density) of electronic chips is doubling every 18 months or so. If this rate continues, then by the end of the second decade of our new century, electronic components will have reached the size of molecules, which will raise a host of conceptual and implementation problems. For example, how will it be possible to design electronic systems with a trillion trillion (the number of molecules in a human sized object such as an apple) components? Will this gargantuan number of components have to self-assemble in an artificial embryological manner, a theme important to the field of evolvable hardware? How can a team of human designers design top-down-style a trillion or more components? It will obviously be impossible. Much greater levels of automated design will be needed, a topic to be addressed in greater detail in the next subsection.

What are the consequences of femto-second switching, that molecular electronics will allow? Today it is already possible to switch in femto-seconds. This has been achieved with quantum optics [Walls & Milburn, 1994]. However, femto-second switching will have profound effects on traditional computing methodologies, effectively killing them. Traditional computing architectures use a "fetch-execute" model [Hennessy & Patterson, 1996] with a central memory containing the computer program's instructions. This basic architectural model works in the nano-second domain, but will fail in the femto-second domain. It is a widely known rule of thumb that light travels about a foot (30 cms) in a nano-second. In a femto-second, light travels a millionth of that distance, i.e. 0.3 micron, or about 3000 Angstroms. Hence computer instructions (if we continue to use such a methodology) will have to be distributed throughout the computing medium, i.e. be local to where they are to be used.

Computing will have to be distributed, local and emergent. Electronic components will only be influenced by their neighbors, and not by components that are "far" away, in femto-second switching terms.

Hence it is likely that future computation will be "crystalline" in nature [Margolus 2002]. Component "modules" of molecular scale, will be placed at "grid points" as in 2D and 3D crystalline structures.

We now attempt to calculate roughly how many atoms one could realistically put into such a module. If these modules were cubical, and were "packed" together like a tower of child's blocks, then each block (i.e. cube) could be maximum 3000 Angstroms on a side, otherwise subcomponents (atoms/molecules) near one face of the cube could not influence another subcomponent (atom/molecule) near the opposite face, before the latter switches again (at femto-second speeds).

If we assume in the limit that each atom can perform the action of a simple logic gate, then how many such atoms could one fit into a 3000 Angstrom cube? If we assume that the atoms are carbon or silicon etc based, then let these atoms have a diameter of about 10 Angstroms, so that 300 atoms could fit into one side of the cube, hence about 30 million such atoms per cube. This number should be sufficient to perform many complex functions per node. Nevertheless, it will ultimately pose a limit to what one can do with each node. This should be remembered.

This crystalline or cellular automata [Margolus 2002] like structure will be the future of molecular based computation. Nodes will be influenced only by their neighbors. Instructions will need to be sent to each node from the surfaces of the cube, prior to execution, but only if one insists on using such an approach.

An alternative, is to use a form of "emergent computation", whereby the mutual interaction of many such locally interconnected nodes results in a desired functionality of the whole. However, this raises the question of how one programs such a system. This is the topic of the next subsection.

b) Evolvable Circuits

The short answer to the above question is – "by evolving it". It is extremely difficult to program using an emergent style. The complications of a 3D cellular automata like, or crystalline like computational process are severe, so really all one can do is to set up such a system, with its local instructions and let it run and observe the results. To improve upon it, one can make random changes to its structure and observe the results. One sees that this line of reasoning results in the use of an "evolutionary engineering" paradigm. Really, the only effective way of coping with such massive levels of complexity is to use nature's method, i.e. applied evolution. This paper presents one particular evolutionary model that may be suitable for such a purpose.

c) Reversible Circuits

In the 60s and 70s, theoretical physicists considered the theoretical limits of computation, and founded a specialty called "the physics of computation" [Feynman 1996], [Hey 2002], [Leff & Rex 1990], [Leff & Rex 2003]. One such question discussed was concerned with whether it would be possible to compute without generating heat. Landauer's Principle [Landauer 1961] states that the source of the generation of heat in computation is due to the destruction of bits of information, not their transformation. If a computational system and its environment are considered as a closed system (i.e. no energy is allowed in or out), then the second law of thermodynamics states that the entropy (a measure of the disorder of a system) cannot decrease, and usually increases. If x bits of information in a system are wiped out, the size of the state space of the system is reduced from 2^N to 2^{N-x} bits, so that its entropy decreases (i.e. that there is less scope for disorder, because the state space is smaller). Hence the entropy of the environment increases (i.e. its temperature rises) so as to keep the total entropy from decreasing.

Landauer looked at the computing of his day, and noticed that computers used irreversible logic gates (e.g. NAND gates, with their 2 bits in and 1 bit out, thus necessarily destroying 1 bit of information at each gate per calculation. He concluded from this observation that computing necessarily implied the generation of heat because its logic gates were irreversible. It did not occur to him to ask if it were possible to use reversible gates. In the 70s, various people began thinking about reversible logic gates (e.g. [Fredkin & Toffoli 1982], [Toffoli 1984]) and noticed that with reversible gates (i.e. one can deduce the output from the input and vice versa, due to the 1 to 1 nature of the gate) it was possible to devise reversible computers using reversible gates, so that one could input a bit string at one end of the computer, make a copy of the answer at the other end, then send the answer back thru the computer to result in what one started with. No bits have been wiped out, so no heat is generated. This is the basic idea of reversible computation.

In section 3 we will give a brief summary of such a reversible computing style, using Boolean neural networks.

Since 3D crystalline nano-scale circuits cannot be irreversible (otherwise so much heat would be generated, that these circuits would explode [Hall 1993]), reversible computing becomes essential.

d) Robust Circuits

It is well known from high school chemistry that there are a trillion trillion atoms in one “mole” of material. (A mole is by definition the number of atoms of a given type contained in a lump of that material whose mass is equal in grams to that atom’s atomic weight, e.g. there are a trillion trillion (actually, $6.023*10^{23}$) atoms in 12 grams of carbon (where 12 is carbon’s atomic weight).

With a trillion trillion components in a human scale molecular circuit, it will be effectively impossible to fabricate such circuits without faults. Hence methods must be found to build nano-scale circuits that are fault tolerant. For example, the neural circuits in our brain, function fault tolerantly because of massive redundancy. Each neuron in the brain is connected to roughly 10,000s of others, so that if one neuron dies, the impact of its death on the neurons it connects to is negligible, due to the dominating influence of the other 10,000s.

It may be necessary to evolve fault tolerant nano-scale circuits based on similar neural network ideas, with the artificial neurons having large numbers of artificial synapses, or connections. By the same logic, if one artificial neuron or synapse is faulty, it will not matter.

But, what about the impact of faults upon reversibility? One of the conceptual challenges of this paper is finding an answer to this question. If a reversible circuit is perfect (i.e. fault free) then what impact will a fault have upon its reversibility? Could it be that there will be a trade off between reversibility and fault tolerance? Or, can one have both? If a trade off is necessary, then perhaps one can evolve circuits with as little irreversibility as possible for a given level of robustness? Perhaps numerical laws can be found expressing this relationship? We will consider such issues in the discussion of our new RENARO model as presented in section 4.

3. Summary of the Older RENN (Reversible Evolvable (Boolean) Neural Network) Model

This section gives a brief description of the older RENN model, upon which the newer RENARO model is based.

The basic idea employed in this reversible evolvable network architecture is to use a Boolean network, where the nodes are Boolean function truth tables, and the links are connections of single bit streams taken from one of the output columns in a node’s truth table to one of the input columns of (usually) another node’s truth tale. The number of bits input to, and output bits exiting from, a node must be equal, to avoid many-to-one, or one-to-many mappings between the N input bits and the M (not equal to N) output bits (i.e. some of the rows in the truth table will be the same in either the input or the output side). Also the 2^N rows in the output side must all be different. If these conditions are satisfied, then we have reversibility, meaning that for any one of the 2^N different inputs, one can consult the node’s truth table to find the corresponding output string of N bits. Similarly, if the signaling were reversed (for example, by winding the clock backwards) then the incoming "output bits" would generate the corresponding outgoing "input bits". Since no bits are lost (i.e. not wiped out) then no (or very little) heat should be generated.

A Boolean network of this type can have any number M of nodes. There will thus be a total of N*M output lines between all the nodes, where N is both the number of input bits and the number of output bits in a node's truth table (for all nodes in the network). An output line from one node becomes another node's input line (although an output line can loop back to the same node). These one bit input and output lines are connected randomly (under genetic control) between the nodes.

Methodology

To start signaling, all nodes are given an initial input bit string signal of N 1s. Arbitrarily choose a single output node T. Arbitrarily choose one of its output bits B. Assume the signaling is clocked (i.e. synchronous). The (time varying) bit-string output of this node's bit B is used as the output of the network, from which the fitness (in genetic algorithm terms) is measured. In earlier work [Korkin et al 1998] this bit string can be converted into a time dependent (digitized) analog signal by convoluting the bit string with a digitized convolution window, so that this converted analog signal can then be compared with an analog target signal for evolution. The fitness of the network can then be defined by how closely the (converted) output signal matches (i.e. follows) the target output signal curve over time.

The chromosomes (in GA terms) will contain two sets of information –

a) The truth tables of the M nodes in the network (if there are N in/output bits in (all) the tables, then these truth tables will have 2^N input and output rows).

b) A connection matrix CM(Npq, Nrs) where Npq is the qth output line of node p, is connected (or not) to the sth input line of the rth node. (p can be equal to r). This matrix can be represented as an (M*N)*(M*N) integer array. A 1 in this matrix CM(i,j) indicates a connection between the appropriate output line i and input line j (i.e. "from, to").

Genetic Operators

i) Truth Table Mutations

Choose randomly a node in the network and then chose randomly two different truth table output rows for that chosen node. Swap the two output rows. (One cannot simply flip a random bit in the output row because this would then generate a row equal to some other row in the output side of the truth table, so swapping is necessitated.

ii) Connection Mutations

Randomly choose any two columns (rows) in the connection matrix. Observe the two row (column) positions in which the 1s occur. Swap the two 1s in those two row (column) positions.

Experimental Result

In this experiment, a dynamic "Fourier like" target curve was employed, which had the formula S(t) = 300 + 100 sin($2\pi t/100$) + 50sin($2\pi t/80$) +30sin($2\pi t/70$) – 20sin($2\pi t/60$). The convolution window chosen for this experiment was {8, 16, 26, 35, 44, 52, 59, 64, 65, 64, 61, 57, 52, 45, 37, 29, 21, 13, 7, 4}. Fig. 1 shows that the output curve evolved moderately well, which implies that the RENN model is evolvable.

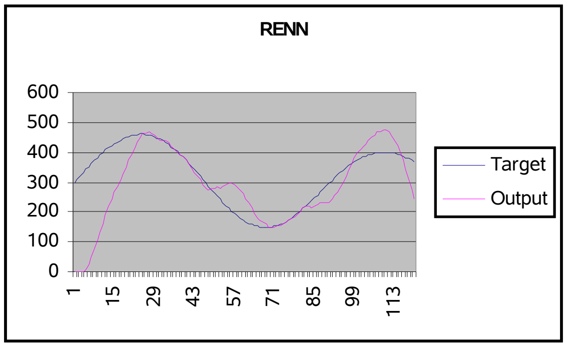

Figure 1RENN with 0 Errors

4. Introducing the New RENARO (<u>R</u>eversible <u>E</u>volvable <u>Na</u>no-scale <u>Ro</u>bust) Model

The RENARO model is essentially a modified RENN model, with extra features that make it robust to faults (fault tolerant), with a slight compromise towards irreversibility. The basic idea of the RENARO model is to use a form of 3-fold redundant circuitry, in which each RENARO node is fitted with 3 RENN nodes, which we will call sub-nodes from now on. Fig. 2 shows the RENARO node structure. 3 copies of each bit are sent (along 3 separate lines) between two nodes, instead of just 1 bit. For each bit position in the N bit output bitstring, the 3 bit values are compared using a “majority vote” truth table, shown in Fig. 3. If 2 or 3 of the input bits (A, B, C) have a given value, the 3 output bits (X, Y, Z) are all given the same majority value (e.g. if A=1, B=1, C=0 then X=1, Y=1, Z=1; if A=1, B=0, C=0 then X=0, Y=0, Z=0). The 3 output bits (X, Y, Z) from the sub-neurons are then sent through another et of voting gates to catch any errors in the LUTs. The signals then move onto the next neuron where the same sequence takes place.

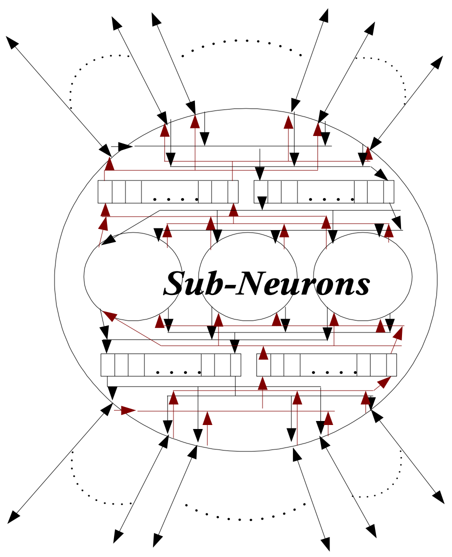

Figure 2RENARO node, the upward path inside the neuron shows the signaling when the network is reversed

A	B	C	X	Y	Z
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

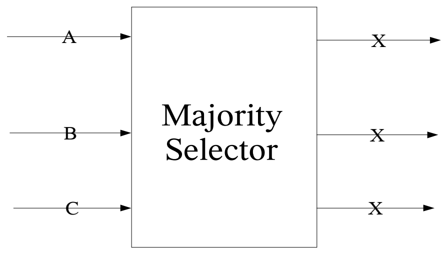

Figure 3Majority Selector Gate and truth table

Note this truth table is irreversible, but at least any (single) fault occurring inside a RENN sub-node, or along a bit transmission line is corrected before it can cause further damage.

Hence there will be N majority vote gates (LUTs) for the 3N bits coming in one direction, and a further N gates for the 3N bits coming in from the opposite direction. Thus the 3 sub-nodes are sandwiched on both sides by majority decision gates. Thus for a given direction of signaling, the input gates would detect signal line faults, and the output gates would detect faults in the sub-nodes. In the opposite signaling direction, the gates would reverse the above roles.

Since in the vast majority of cases, there will be no errors, the inherent irreversibility in these majority vote gates will generate very little heat.

5. Simulation Results of the RENARO Model

Two main tests were administered to gauge the reliability and robustness of the RENARO model. Both cases used the same “Fourier-like” curve used in the RENN tests (namely, S(t) = 300 + 100sin($2\pi t/100$) + 50sin($2\pi t/80$)+30sin($2\pi t/70$)–20sin($2\pi t/60$).) with 120 ticks. In the first sequence a 3-way voting scheme was employed while a 5-way voting methodology was introduced in the second sequence. In both tests, we aimed to find the maximum threshold (i.e. the maximum number of errors introduced into the Boolean circuit) at which errors could be introduced, detected, and corrected. In both sets, errors were introduced to both the LUTs and the lines. Each error could arrive in three forms: stuck-at-0, stuck-at-1, and inverted (i.e., a 1 becomes a 0, and a 0 becomes a 1).

The first test, which utilized a 3-way voting gate, had a bitstring length of 615000. Intuitively, as long as errors were introduced at a rate such that concurrent errors did not appear, the model would *catch every error*. That is to say, as long as two invalid bits did not pass through a single voting gate at the same time, no errors would permeate the system. Logically, it is possible for concurrent errors to exist with no impact on performance, so long as the invalid segments were not used. Using this information it is easily shown that concurrent errors cannot exist in the lines and still have a fault tolerant system.

At a 1% error rate, on average, 58.4 concurrent errors would exist in the entire system. Experimentation showed that this was too large an error rate. Using the logic established earlier, at least one of those concurrent errors existed on the line, or on a used segment of the LUT because significant deterioration in the output curve was recorded. Further experiments showed that approximately 3000 errors in the system proved to be the maximum number that could be tolerated. Even at 3000 errors, some concurrent errors still existed, but to a significantly lesser extent than in the 6000 error case, and the errors were usually never encountered as the signals progressed through the network. Considering that RENARO has roughly three times the number of bits as RENN, the corresponding run on RENN was done using only 1000 errors. Now that the two networks have been handicapped accordingly, meaningful comparisons can be made. Qualitatively, RENARO shows significant improvements over RENN in the area of robustness and fault tolerance. In fact, when compared to the base case, shown in fig. 1, there is no visible trace of errors existing in the system, indeed, no error exists in the actual output.

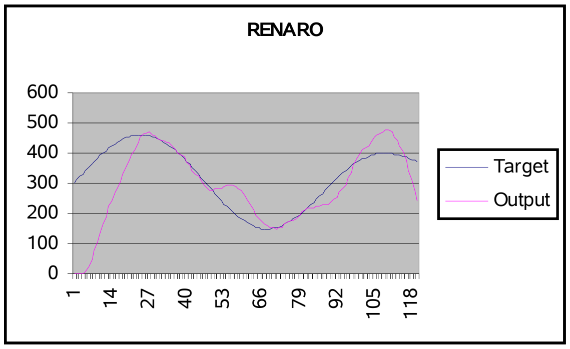

Figure 4RENARO – 3-way voting gate - 3000 errors

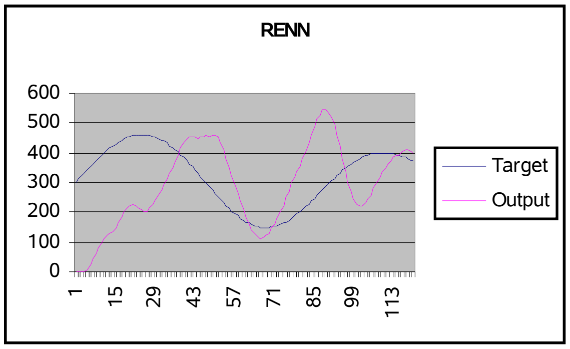

Figure 5RENN - 1000 errors

With these two results we are now able to make a slightly more quantitative analysis of RENARO. The fitness of a run was defined as the sum of the inverse of the square of the difference between the actual output and the desired output plus one i.e., $fitness = \sum \frac{1}{(actual - target)^2 + 1}$. Both the base case and the 3000 error version of RENARO returned a fitness value of 10.03, while RENN with 1000 errors returned a value of 0.37. It is now easily concluded that RENN suffers considerably when errors are introduced to the system. Meanwhile, RENARO is still able to function perfectly below a certain threshold error rate.

Next, a 5-way majority voting scheme was employed, that can of course, cope with two simultaneous errors at the same gate. In this case the bitstring is now 1,025,000 bits long. This time in order to have concurrent errors fool the voting gate three or more incorrect bits must go through at once. As expected the ceiling for total errors rose, but of particular interest is that the ceiling for error rate also rose slightly. With a 3-way voting gate the percentage of total errors allowed was around 0.487%, however, with a 5-way gate that percentage rose 0.39% to 0.878%. While this result is somewhat intuitive, it remains interesting nonetheless. In both cases the experiments show that the gates must be able to catch all errors or the invalid data will cascade through the entire network and severely corrupt the output. In addition, the use of a convolution window only serves to accentuate the error. The same base case was used in this experiment, and a RENN run with 1600 errors was used to perform comparisons for the same reason as cited above.

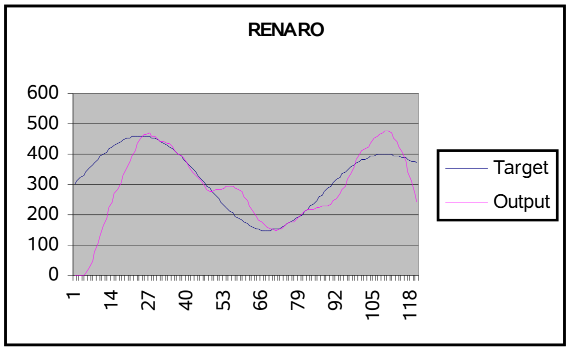

Figure 6RENARO - 5-way voting gate - 9000 errors

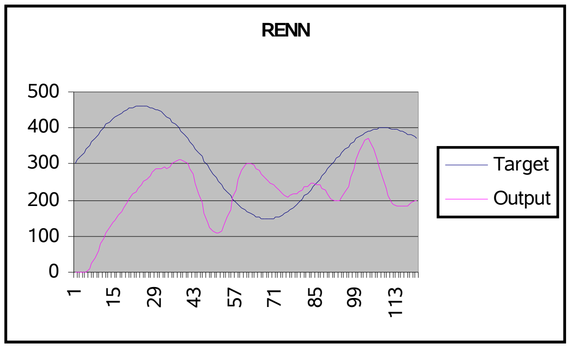

Figure 7RENN - 1600 errors

Using the same fitness definition, a 5-way RENARO implementation returns the same value of 10.03 as in the 3-way and base case. RENN this time returned a value of 0.46. This fitness is higher than the 1000 error case – an example of how the placement of errors comes into play and of the choice of fitness definition. When these two results are taken together, a solid qualitative measurement shows the superiority of RENARO when errors are involved.

Now that we have established an experimental foundation, we work backwards in an attempt to form a mathematical basis for our findings. Using the hyper-geometric distribution on our 3-way voting scheme with 3000 errors, we can conclude that a given set of three signals has a 7.11e-3% chance of having two or more errors. With a total of 205200 redundant sets-of-three in the network, the binomial distribution shows us that 14.59 errors are expected to exist in the system. Experimentation shows that this number is, on average, 14.18. Now we move onto how many errors are actually encountered during the run. Given that voting gates are used 20 times (two rows of ten inputs/outputs) in each neuron, there are 20 neurons, and the network goes through 120 iterations, voting gates are utilized a total of 48000 times. Using this as the number of trials in the binomial distribution, we arrive at an expected value of 3.41 concurrent errors during a single run on 3-way voting RENARO network. This result is consistent with our findings. The hyper-geometric distribution assumes that each set of three voters is equally likely to occur. However, significant portions of the total number of bits are not used as the signals progress through the network. Besides this fact, 3.41 must also be viewed as an inflated estimate, because 7.11e-3% represents the odds of having an error in any set-of-three signal, not necessarily of the set going through the voting gate. Also, as was stated previously, with 3000 errors, RENARO always had concurrent errors present. However, those errors were never realized as previously cited. In addition, because 3000 errors represents the ceiling, RENARO has been known to fail, on occasion, when 3000 errors were present.

6. Summary and Future Work

While RENARO certainly has limits relevant to the number of errors that can be corrected, it represents a significant improvement over RENN in terms of robustness. Beneath a certain threshold level of error rate, all errors are corrected through the voting gates. At error rates greater than the threshold rate, the errors can overload the gate, providing invalid information to the rest of the network which will cascade unchecked for the remainder of the run.

For future work, many possibilities lie open. For example, one could estimate the amount of heat generated by the irreversible majority vote gates in the RENARO model. Perhaps other approaches to creating robustness to faults could be envisioned, e.g. error correcting codes. Another topic which needs addressing is how to construct such circuits. If the type of circuits envisioned in this paper can be made to generate little heat, then 3D circuitry becomes possible. How would one program such circuits, e.g. how to fill the LUTs in the sub-nodes? One idea might be to send in signals from 2 faces of the cube. When they collide, a bit is stored at the collision point. Investigating the details of such a proposal would be worthy of another research paper.

References

[Bennet 1973] "Logical Reversibility of Computation", C. H. Bennet, IBM J. Res. Dev. Vol. 17, 1973, pp 525-532.

[de Garis et al 2002] "Reversible Evolvable Networks : A Reversible Evolvable Boolean Network Architecture and Methodology to Overcome the Heat Generation Problem in Molecular Scale Brain Building", Hugo de Garis, Jonathan Dinerstein, Ravichandra Sriram, Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware, 15-18 July, 2002, Alexandria, Virginia, p274, http://www.cs.usu.edu/~degaris/papers/RENN.pdf or .ps

[Feynman 1996] R.P. Feynman, "Feynman Lectures on Computation", ed. A.J.G. Hey & R. W. Allen, Addison Wesley, 1996.

[Fredkin & Toffoli 1982] Edward Fredkin & Tommaso Toffoli, "Conservative Logic", Int. Journal of Theoretical Physics, 21, 219, 1982.

[Hall 1993] "An Electroid Switching Model for Reversible Computer Architectures", PhysComp 92, IEEE Press, Los Alamitos, CA, 1993.

[Hennessy & Patterson, 1996] John L. Hennessy & David A. Patterson, "Computer Architecture : A Quantitative Approach", Morgan Kaufmann, 1996.

[Hey 2002] ed. A. J. G. Hey, "Feynman and Computation", Westview Press, 2002.

[Korkin et al 1998] "A Spike Interval Information Coding' Representation for ATR's CAM-Brain Machine (CBM)", Michael Korkin, Hugo de Garis, Norberto Eiji Nawa, Int. Conf. on Evolvable Systems, Sept. 24-26, Lausanne, Switzerland, http://www.cs.usu.edu/~degaris/papers/ices98-final.pdf

[Landauer 1961] “Irreversibility and Heat Generation in the Computing Process”, R. Landauer, IBM J. Res. Dev. Vol. 5, 1961, pp 183-191.

[Leff & Rex 1990] eds. H.S. Leff & A. F. Rex, “Maxwell’s Demon : Entropy, Information, Computing”, Princeton University Press, 1990.

[Leff & Rex 2003] eds. H.S. Leff & A. F. Rex, “Maxwell’s Demon 2: Entropy, Classical and quantum Information, Computing”, Institute of Physics Publishing, 2003.

[Margolus 2002] “Crystalline Computation”, Ch. 18 in book “Feynman and Computation : Exploring the Limits of Computers”, ed. A.J.G. Hey, Westview Press, 2002..

[Toffoli 1984] R. Toffoli, “Comment on “Dissipation in Computation”, Phys. Rev. Let. 53. 1204, 1984.

[Walls & Milburn 1994] D. F. Walls, G. J. Milburn, “Quantum Optics”, Springer Verlag, 1994.