Achieving Complexity and Reliability on Unreliable Platforms

Pauline C. Haddow and Morten Hartmann
The Norwegian University of Science and Technology
Department of Computer and Information Science
Sem Sælands vei 7-9, 7491 Trondheim, Norway

pauline@idi.ntnu.no,mortehar@idi.ntnu.no

Abstract

As the density of electronic devices increases, traditional design techniques fail to fully utilise the expanding resource. How can the design community utilise the design capacity that technology is offering and at the same time ensure its correctness? A further problem facing the design community is that yield rates are falling due to the increasing complexity of modern chips. How can we improve yield rates whilst achieving even more complex chips?

This paper presents our approach to the problem of achieving complex and reliable designs. First we consider our approach with respect to one of today's technologies, Field Programmable Gate Arrays, and present our current results. We then present our ideas with respect to solutions to this problem based on future distributed technologies.

1 Introduction and Motivation

To find solutions to the problem of developing large complex circuit designs new design paradigms are required [Sem01, AN95]. Our approach focuses on efficiently searching a larger design search space than that searched using traditional design solutions. This approach is within the field of evolvable hardware, applying evolutionary algorithms to the design of digital and analogue circuits.

Complexity

Using evolutionary algorithms, some of the design constraints built into traditional design techniques and tools are removed. In addition, evolution removes limitations imposed by the human designers thought process. However, is this enough to enable us to evolve complex reliable designs? Unfortunately the answer is no. Although evolution is a powerful tool it suffers from a number of weaknesses. The main weakness is the resource greedy nature of the methodology.

To reduce resource usage and thus improve evolution, we need to expand the methodology to include concepts that will aid simplification of the genotype (genetic algorithm

representation of the circuit). One possible solution to this problem is to turn to biology. Biological development illustrates how a simple description may be used to develop a complex organism. We, therefore, wish to introduce a similar method to the evolution process i.e. specify a simple genotype that can be evolved efficiently and developed to the complex circuit. More information on this work may be found in [HTvR01].

A further improvement in resource usage may be achieved by reducing the size of the phenotype itself i.e. by reducing the configuration bits required to program the device. A simplified Virtual FPGA, termed Sblock has been developed to achieve this goal. A brief description of this architecture is given in section 2 and a more detailed description may be found in [HT01].

Robustness

The above solutions have focussed on evolving large complex designs. What about robust designs? Why do we need robustness? In a conventional, non fault-tolerant technology, unforeseen events may occur which prevent proper operation of an implemented design. To achieve robustness, a number of techniques have been developed to detect and repair faults.

A number of noteworthy efforts within fault detection and repair based on the principles of biological development, are currently under development. In the embryology work conducted at York [BOST00] and Ecole Polytechnique Fédérale de Lausanne (EPFL) [MSST00], experiments have been conducted using FPGAs with extended Configurable Logic Blocks (CLBs) to contain a complete genotype of the circuit. Through repeated cell divisions a circuit develops from a single cell into a full-grown phenotype. An interesting approach was taken in [BT01] where principles of biological immune systems were adopted to attain fault tolerance. Work on achieving tolerance to temperature changes includes [TL00] and [SKZ01].

However, as designs get larger can we really rely on being able to detect and repair all faults? Can we expect high fault coverage as designs grow in complexity? Again the answer is most certainly no. Although these techniques will still be very essential, we need to be able to tolerate faults. This has a two-fold advantage. First, we can tolerate faults that may not be detectable or provided for by the detection and repair system and secondly we can tolerate a less reliant technology. Even with today's technologies this would mean that we can improve yield by accepting chips that would otherwise be discarded. On the other hand, it opens the path for new technologies based on less reliable or less controllable mediums. Our approach to fault tolerance is termed Messy Gates and is described in section 3.

Future Technology

So far we have discussed both the advantages and weaknesses of evolution to achieve large complex robust designs with regards to today's technologies. However, what about the future? Can we really exploit evolution on today's technology platform? Can evolution also be a more futuristic design methodology for newer technology platforms? Nanotechnology may be said to be a future faster technology but still limited by the design challenge. What characteristics should a future technology have so as to reduce the design challenge and enable fault tolerant designs?

A good place to start is to consider digital design itself. Can we free a digital design

from its rigid structure — direct communication between gates and digital thresholds, so as to enable new freer technologies? What about a distributed fault tolerant technology? Moving in this direction, inspiration may be gained from a newer research area —amorphous computing [Aea99]. Looking even further towards the future we could be talking about fault detection and repair from within the technology.

The remainder of the paper is structured as follows. Section 2 briefly outlines the Virtual EHW FPGA. Section 3 presents a fault tolerant methodology for today's and tomorrow's technologies along with a selection of results to illustrate its viability. Section 4 discusses robust computing. Also, the amorphous computing concept is presented, as developed at MIT, along with a discussion as to how we can achieve similar characteristics in a future technology. This discussion is taken a step further in section 5 where possible avenues for implementation of a digital design in such an environment are discussed. Further, reliability is taken to an even more futuristic level considering fault detection and repair from within the technology. Finally in section 6, we summarise the contribution of this work.

2 Sblock Architecture

The Virtual EHW FPGA contains blocks — *Sblocks*, laid out as a symmetric grid where neighbouring blocks touch one another. There is no external routing between these blocks except for the global clock lines. Input and output ports enable communication between touching neighbours. Each Sblock neighbours onto Sblocks on its 4 sides.

Each Sblock consists of both a simple logic/memory component and routing resources. The Sblock may either be configured as a logic or memory element with direct connections to its 4 neighbours or it may be configured as a routing element to connect one or more neighbours to non-local nodes. By connecting together several nodes as routing elements, longer connections may be realised.

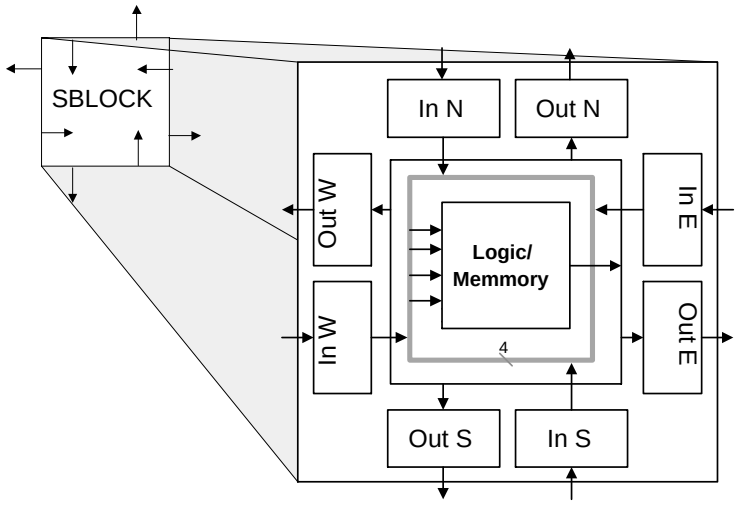

Figure 1: Sblock — Routing and Logic/Memory Block

Figure 1 illustrates the internal routing of the Sblock. Each Sblock is responsible for processing input signals and routing signals to its neighbouring Sblocks. There are 4 pairs of unidirectional wires at each interface. The input wires are attached through

routing logic to the input routing channel which provides a dedicated wire for each input. This channel is then fed into the logic/memory block. Output from the logic/memory block is a single wire which feeds the output routing ring. All the 4 outputs of the Sblock are attached to this output routing ring through the output routing logic.

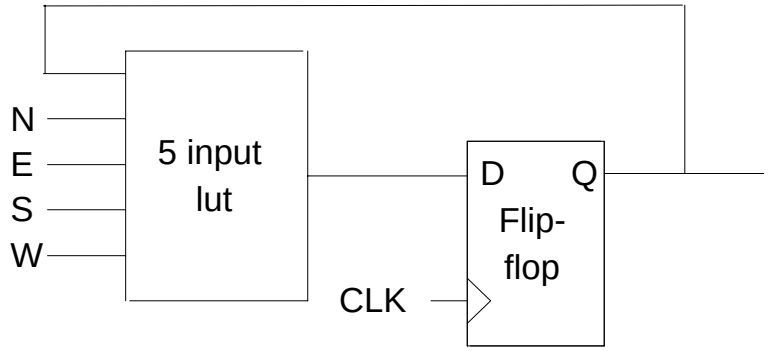

Figure 2: Sblock Logic

A more detailed view of the logic/memory block is shown in figure 2. Inputs from neighbouring Sblocks and a feedback from the output are connected to a 5 input look up table (LUT). The LUT may be configured to hold a function. When Don't Care (DC) bits are placed at a given input, then that neighbouring Sblock is not connected to it. In this way the LUT is programmed not only for functionality but external connectivity of the Sblocks. Therefore, to alter the functionality and connectivity of an Sblock only the LUT content need be reprogrammed.

When the Sblock is used for routing, all inputs apart from the incoming input to be routed are don't care inputs. The LUT is thus set up as shown in figure 3. In this example the west input is to be routed to the north by the routing module R and the result is to be ORed the *or* module's north input. The output of R reflects the values of the west input, thus implying don't cares at the other inputs. However, the output values will appear at all outputs not just the north output. It is up to the Sblock to the north to read its input at this port i.e. its south input, as shown. The output of or reflects the OR of its South and North input, thus implying don't cares at the other inputs.

3 Messy Gates

Seeking to investigate designs that do not rely upon every single component functioning correctly, a concept termed messy gates was introduced in [MH01a]. As such, the messy gate concept may be said to be a fault tolerant rather than a fault detection and repair methodology. This tolerance is a tolerance to a less reliable technology.

The work considers digital designs using gates that do not necessarily confer with the strict signal voltage range requirements of regular digital systems. That is, messy gates are able to operate out with the digital thresholds, their non-perfect digital outputs being allowed to propagate through the circuit. As such, the messy gate concept not only tolerates less than perfect gates but in fact uses evolution to exploit this *messiness*.

Circuits evolved using messy gate components have evolved architectures that are tolerant to faults. These architectures may incorporate some intricate redundancy where

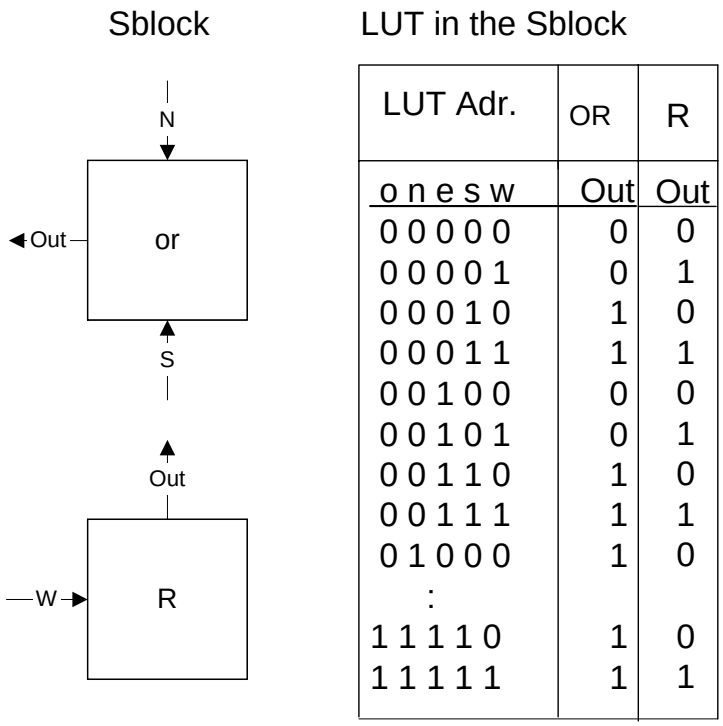

Figure 3: Sblock through Routing

the importance of individual gates is reduced whilst the priority and timing of signals along certain pathways are prioritised. As such, a given gate such as an AND gate may not have completed its function, due to unstable input values, before its output is transferred to another gate. This exploitation of ”messiness” can off course create circuits that cannot be implemented on traditional technologies. However, our long term goal is to establish design methods for achieving robustness for future technologies where such non perfect digital gates exist and analogue signals may be allowed to propagate through the design.

The initial work approached the fault tolerance issue from an abstract viewpoint [MH01a, MH01b] without regard to the underlying technology and may be said to be a proof of concept approach. The promising results of the initial work led to an in-depth MOSFET technology specific investigation of the messy gates concept [HEHM02, HHE02].

Figures 4 and 5 illustrate the power of the messy gate concept to generate functionally correct circuits in spite of noise and gate failures. The results are based on simulation of 5V CMOS, using NOT, NOR and NAND gates to evolve a multiplier. The graphs plot the average number of generations (used by the evolutionary algorithm) against increasing noise and gate failures respectively. The noise percentage indicates noise fluctuations relative to the complete signal voltage range. The gate failure percentage signifies the probability of a gate failing (on a per gate basis).

The results show that not only can evolution create a functionally correct circuit in spite of noise percentages of up to and including 50% but that in fact evolution seems to make a virtue of the applied noise. This is indicated by the lack of a trend towards increasing generations as noise increases. Gate failure is, not surprisingly a great challenge. In the results shown, however, evolution is able to handle gate failure up to 10%. Further details of these and other relevant experiments and verification of the evolved circuits may be found in [HEHM02, HHE02].

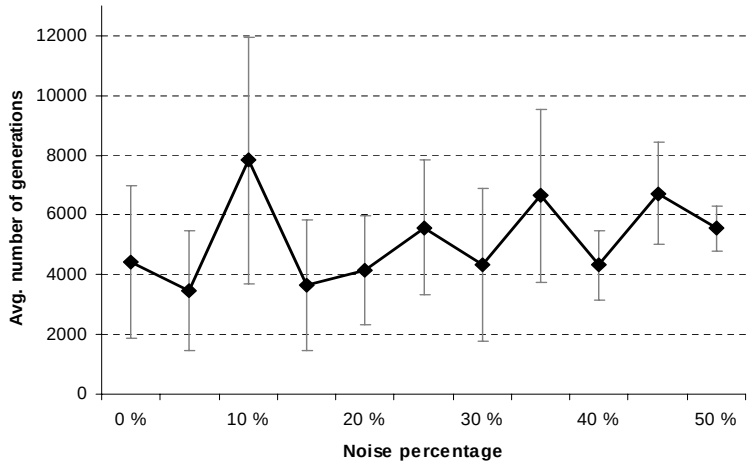

Figure 4: Evolution of Correct Circuits in Noisy Environments

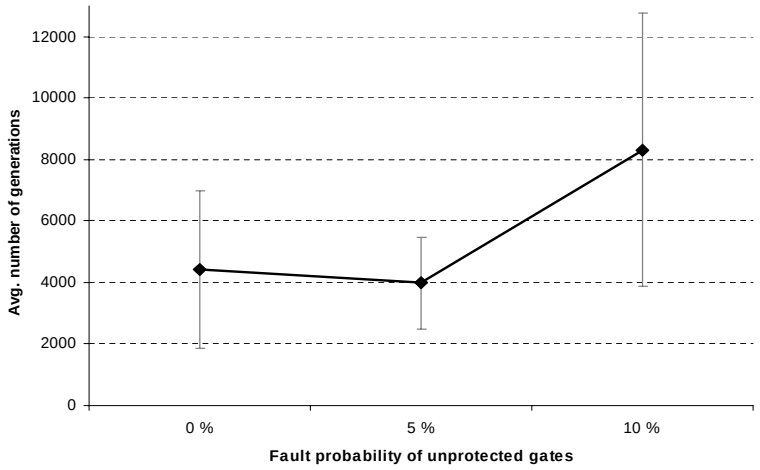

Figure 5: Evolution of Correct Circuits in Environments Where Gates Fail

4 Robust Ways of Computation

In the messy gates approach, noise and gate failures in a circuit environment are tolerated. Moving away from circuit design for a moment, we can consider a more general fault tolerant approach to computation. That is that of amorphous computing where correct functionality will be attained despite the possibility of a limited amount of malfunctioning units. In other words, the computational nature of amorphous computer has built-in fault-tolerance. It will not repair a broken unit but distribute functionality in such a way that the failure does not interfere with an implemented applications behaviour.

The main features of an amorphous computer are presented in section 4.1 and in section 4.2 these features are discussed in the light of both today's technologies and that which is achievable in the near future.

4.1 An Amorphous Computer

Amorphous computing [Aea99] was developed in anticipation of the fast evolving fields of micro-fabrication and cellular engineering. In these fields large collections of small computational units or cells can easily be produced. Amorphous computing is a computational paradigm for such collections and is based on the following assumptions:

- large number of computational units
- limited computational power per unit
- unreliable units
- units are not perfectly aligned geometrically
- wireless connections
- only local communication within a certain range
- asynchronous operation
- possibly movable units
- no global system knowledge

Figure 6 illustrates a collection of processing units in an AComp. The darker unit in the centre has a communication radius **r**. As highlighted, all units either within or overlapping the circle defined by **r** are within broadcasting distance of this unit.

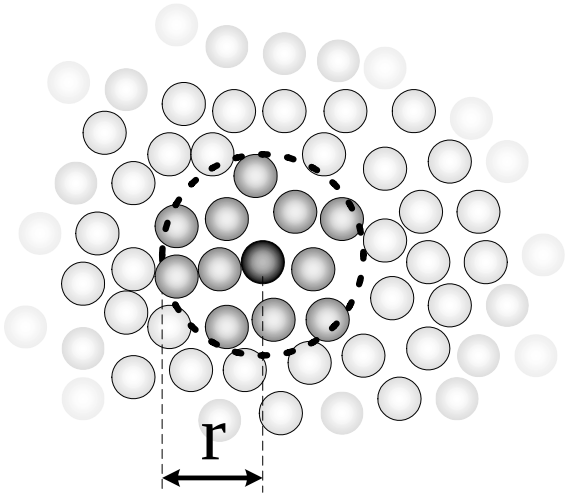

Figure 6: Communication Radius in an Amorphous Computer

While producing a system composed of such units is within reach, as yet there exists no programming paradigms applicable to it. Amorphous computing aims to fill the gap between the construction and the programming techniques required for an amorphous computer. More concretely, the important task in amorphous computing is the identification of appropriate organising principles and programming methodologies for obtaining predefined global behaviour through local interactions.

4.2 Moving towards a Distributed Fault Tolerant Platform

Considering the amorphous computing paradigm, we see that some of these features are present in today's reconfigurable technologies. Other features are not present but desirable.

Today's FPGAs consist of a large number of computational units termed Configurable Logic Blocks (CLBs). However, FPGAs are designed in a way so as to expect that each CLB involved in a design is a task or sub-task of the function being performed. In the amorphous computing paradigm the requirement for a large number of computational units refers to the fact that the number of units available to each task or subtask is well in excess of the number required to compute that task or subtask. This does not preclude an FPGA from being used for this purpose but would involve distribution of tasks at a level above that of individual CLBs. This is currently under investigation but this methodology is not the focus of this paper.

Each CLB may be regarded as having limited computational power and as such appropriate for an amorphous computing type computation. Although CLBs cannot be said to be reliable since CLBs may fail, the digital paradigm using it assumes a reliable technology. This is due to the fact that defects which occur during manufacturing, if detected, cause the chip to be discarded. Other defects or failures are then handled through fault detection and repair mechanisms. However, following the amorphous computing paradigm which assumes unreliable units, a methodology is required which will tolerate defects (arising during manufacturing) or faults (arising during the chip's lifetime). This is, of course, the purpose of our messy gate concept.

Although units non perfectly aligned geometrically, may be desirable in the future, it is believed that a distributed fault tolerant platform may also be achieved on a more traditional grid-style platform. The grid provides, for example, 4 local neighbours thus providing local communication within a given range. This, however, will not provide a large number of local units but is perhaps a good start to implementing a near amorphous like structure on today's technology.

Communication is restricted to only local communication with no global system knowledge. This restriction is not present in today's FPGAs since many different routing possibilities exist in addition to local routing. However, in our simplified Virtual FPGA presented in section 2, routing is restricted to local routing only thus providing a technology for further investigation of the amorphous concept.

FPGAs are inherently a synchronous technology and although they may be used as an asynchronous technology, asynchronous designs are difficult to implement. Again, our simplified Virtual FPGA is suitable for asynchronous design.

5 Digital Design on a Distributed Environment

Before moving the digital design era to a more distributed environments, we should first observe the relevant properties required for digital designs. These requirements are described in section 5.1. Some early thoughts on how to satisfy these requirements in a distributed amorphous computer environment are given in section 5.2. It should be noted that it is also important to consider the distributed environment in the light of the methodology i.e. the ability of evolution to exploit this environment. A discussion as to the suitability of future technologies for evolution and development is given in [HvR01].

5.1 Digital Design Requirements

Digital design as opposed to analogue design does not use the full properties of the underlying technology. It works at an abstract level where signal values above or below given thresholds are categorised as logic values. This means that circuit design is easier to achieve since slight variations in the low-level signal values do not effect the logic values. The earlier mentioned messy gate approach may be viewed as somewhere in between, in that it is close to digital but with propagating analogue signals.

With digital designs being robust to small signal variations, the digital abstraction does require that a produced signal be given time to stabilise before the correct value is used by the next unit. Synchronous or asynchronous design techniques are used to achieve this timing stability. In asynchronous design, a particular challenge is to attain stability in sequential circuits since these incorporate feedback loops.

An additional requirement of digital design is the possibility to send a directed message from one output to a designated input over some communication path. The communication paths are fixed for a design, whether we consider ASICs or FPGAs. These paths are fixed at design-time, or in the case of FPGAs, one might say at configuration-time. Usually, only one single unit is allowed to generate a unique signal at a given point in time. That is, no other units can contribute to that single digital signal.

A digital design implicitly assumes that the implementation platform is 100% functional and that each gate has a crucial role to play in the overall behaviour of the circuit. A support mechanism is, therefore, required to attain this flawless behaviour in the presence of faults.

The properties discussed above place requirements on the underlying technology. It assumes the presence of an abstraction mechanism and methodologies to control it.

5.2 Achieving a Complex Distributed Fault Tolerant Design

In order to achieve a distributed digital design one needs to consider the above mentioned requirements. The control of the system needs to be decentralization and only local interactions should be available. This rules out global clocks and synchronous designs. Instead it calls for asynchronous behaviour and methods for achieving asynchronous control suited for systems where communication is based solely on local broadcasts.

Let us start out by imagining a very simple system. Initially the system is based on the earlier mentioned Sblock architecture and thus feasible to implement using existing technology. The Sblock architecture allows local broadcasting where each unit has four neighbouring units.

Circuit design may be initiated by an input unit broadcasting a signal. This signal is thus made available to the four neighbouring units. The choice of unit to receive the signal is controlled by evolution. The chosen unit then performs a function of the received value. The function itself is selected from the available unit functions by the evolution process. On completion of the function, the unit broadcasts a new signal — the result of the performed function. This signal is available to its four neighbouring units. This process is repeated until the signal reaches the output unit.

The circuit design process described could design circuits that perform simple operations such as inverting an input value and routing it from the input to the output unit. The process itself seems at first quite simple, but presents many implementation challenges. Issues such as how to design the selection process which evolution uses to choose both the next unit in the path and its function need to be addressed.

The process described can be extended to allow for a much wider range of behaviour. By allowing a unit to have more than one input other logic functions may be achieved for a single unit i.e. 2 and 3-input logic gates or in fact any logic function for a given number of inputs. This implies that a unit needs input from more than one broadcast source before it can compute its function. On the other hand, a unit should be able to feed its output to more than one other gate i.e. a broadcast may have more than one recipient. Enabling these features enables parallel operations. If a unit initiates a broadcast and more than one of its neighbouring units are configured to receive the signal, those receiving units will compute their function based on the incoming signals and initiate broadcasts. Combining these features with multiple inputs and outputs yields a system that is able to perform complex tasks such as binary addition or multiplication.

With the increased complexity of the system, more design issues arise. These may include, for example, race conditions and unstable chaotic behaviour due to parallel operations and asynchronous control of functionality due to varying input arrival times.

Could the system described also enable fault tolerant designs? If we take into consideration the messy gates approach described in section 3, we see that fault tolerance could be gained by utilizing the parallel features of the system i.e. intricate redundancies could emerge. To achieve this, the messy gates approach would require to be tuned to the new technology.

So far, our system is still possible to implement on existing technology, using the virtual Sblock architecture. If we, however, adopt more of the messy gate properties and allow analogue signal values to propagate through the circuit, the current Sblock architecture comes short. An Sblock-like architecture with less mechanics for forcing the signals towards the digital endpoints of the scale would allow a more messy complex behaviour. Such an architecture seems technologically plausible today although not implementable on a traditional FPGA chip.

As the functional complexity of the system increases, so does the complexity of designing a functionally correct system. Evolutionary approaches are able to come up with surprising solutions. However, as discussed in section 1, evolution is limited through its resource usage and we can expect to turn to new supporting methodologies such as artificial development.

6 Summary and Future Directions

This article has focussed on a more futuristic view to fault tolerance. The messy gate approach, although implementable to a limitable extent in today's technology, is aimed at a more futuristic technology. Inspired by the amorphous computing concept, possible future technologies for fault tolerant circuits has been discussed as well as the problems inherent in achieving more complex fault tolerant circuits.

Looking even more to the future we might see more of the amorphous computing style principles through myriads of unreliable, non-geometrical processing units, including large neighbourhoods and processing units for a given broadcast.

Considering the importance of robustness, perhaps fault detection and repair and fault tolerance methodologies are still not enough. Perhaps we will also be looking to a technology that can detect and repair certain failures automatically.

Imagine a mixture of chemical substances in a reservoir, together with a circuit-implementation surface. The surface is made of individual cellular structures, comparable

to a CLB in an FPGA, but with the quantity and lack of geometric alignment of the amorphous computing principles. The total of the chemical substances and the computational units on top instantiate a certain chemical equilibrium. If a computational unit fails, this equilibrium is disrupted and a new working unit is spontaneously re-instantiated in order to re-attain the equilibrium. State information for a computational unit is redundantly stored on its neighbours and fed back onto the new unit to ensure continuous operation. The entire task of detecting a failure and replacing the part is taken care of using chemical principles. Again, this seems technologically improbable in the near future, but might be a starting point for experts in chemistry and materials to extend this to a working prototype.

References

[Aea99] H. Abelson et al. Amorphous computing. Technical report, Massachusetts Institute of Technology, 1999.

[AN95] R. Aaserud and I.R. Nielsen. Trends in current analogue design. *Analogue Integrated Circuits and Signal Processing*, 7(1), 1995.

[BOST00] Daryl Bradley, Cesar Ortega-Sanchez, and Andy Tyrell. Embryonics + immunotronics : A bio-inspired approach to fault-tollerance. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 215–224, 2000.

[BT01] D.W. Bradley and A.M. Tyrrell. The architecture for a hardware immune system. In *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pages 193–200, 2001.

[HEHM02] Morten Hartmann, Frode Eskelund, Pauline Haddow, and Julian F. Miller. Evolving fault tolerance on an unreliable technology platform. In *Proc. The Genetic and Evolutionary Computation Conference, GECCO-2002*, pages 171–177. Morgan Kaufmann Pub., 2002.

[HHE02] Morten Hartmann, Pauline Haddow, and Frode Eskelund. Evolving robust digital designs. In *Proc. The 2002 NASA/DoD Conferance on Evolvable Hardware, EH'02*, pages 36–45. IEEE Computer Society, 2002.

[HT01] P. C. Haddow and G. Tufte. Bridging the genotype-phenotype mapping for digital FPGAs. In *3rd NASA/DoD Workshop on Evolvable Hardware*, pages 232–239, 2001.

[HTvR01] P.C. Haddow, G Tufte, and P. van Remortel. Shrinking the genotype: L-systems for EHW? In *4th International Conference on Evolvable Systems: From Biology to Hardware (ICES01)*, pages 129–139, 2001.

[HvR01] Pauline C. Haddow and Piet van Remortel. From here to there: Future robust ehw technologies for large digital designs. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pages 232–239. IEEE Computer Society, 2001.

[MH01a] J.F. Miller and M Hartmann. Evolving messy gates for fault tolerance: some preliminary findings. In *The 3rd NASA/DoD Workshop on Evolvable Hardware*, pages 116–123, 2001.

[MH01b] J.F. Miller and M Hartmann. Untidy evolution: Evolving messy gates for fault tolerance. In *The 4th International Conference on Evolvable Systems: From Biology to Hardware, ICES2001*, pages 14–25, 2001.

[MSST00] Daniel Mange, Moshe Sipper, André Stauffer, and Gianluca Tempesti. Toward self-repairing and self-replicating hardware : the embryonics approach. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 205–214, 2000.

[Sem01] Semiconductor Industry Association. *The International Technology Roadmap for Semiconductors, 2001 edition*, 2001.

[SKZ01] A. Stoica, D. Keymeulen, and R. Zebulum. Evolvable hardware solutions for extreme temperature electronics. In *Proc. The Third NASA/DoD Workshop on Evolvable Hardware, EH 2001*, pages 93–97, 2001.

[TL00] Adrian Thompson and Paul Layzell. Evolution of robustness in an electronics design. In *Evolvable Systems: From Biology to Hardware. Third Int. Conf., ICES 2000*, volume 1801 of *Lecture Notes in Computer Science*, pages 218–228, 2000.