From Here to There : Future Robust EHW Technologies for Large Digital Designs

Pauline C. Haddow * Piet van Remortel †

* The Norwegian University of Science and Technology
Department of Computer and Information Science
O.S. Bragstadsplass 2E, 7491 Trondheim, Norway

† Vrije Universiteit Brussel
COMO - Department of Computer Science
Pleinlaan 2, 1050 Brussels, Belgium
pauline@idi.ntnu.no,pvremort@vub.ac.be

Abstract

Fault-tolerance may be expected to gain more and more importance in the future. Extremely harsh and changing environments, like outer space, already force us to think about this issue today, but issues like production of large-scale devices might put the same requirements on the devices of tomorrow.

Imagine a mixture of chemical substances in a reservoir, together with a circuit-implementing shell that has self-repairing properties based on the maintainance of the chemical equilibrium. Could this type of solution be the basis for a robust future technology for evolvable hardware?

A long term goal of evolvable hardware is to evolve large complex designs for large devices. However, both evolving large complex designs and manufacturing large reliable devices is technologically out of reach due to the resource greedy nature of GAs and low device yield rates.

In this article we explore the technological requirements of digital design, design by evolution and development and the reliability issue in the light of today's digital evolvable hardware technology, FPGA and a proposed fault tolerant technology, Amorphous Computers. Considering the limitations of these platforms, we project these findings towards possible future technology platforms.

1 Introduction

During the last 3-5 years the field of Evolvable Hardware (EHW) has matured considerably. One of the goals of EHW has been to evolve large designs, not achievable with traditional design methods. This goal is still beyond our reach. A further traditional problem, not due to the methodology but the technology itself, is achieving large scale devices with reasonable yields. In this work we consider the problems of evolving large digital designs on larger reliable platforms.

Apart from being suited for implementing digital designs, a new technology should be compatible with the evolutionary design methodology. This issue has been considered with respect to today's technologies in our work presented in [7]. Facing the challenge of larger and more complex designs, adopting more powerful ways of expressing circuit layouts is fundamental. This might involve some form of biological development principles incorporated into the design methodology [4].

A general property of future technologies is that they should offer some kind of robustness against faults thus ensuring continuous operation after critical events or unit failures. This may be achieved by robust ways of computation or by an underlying fault-detection and repair from within the technology. As a source of inspiration for a new EHW fault tolerant platform, we consider a fault tolerant platform proposed by MIT termed Amorphous Computers (AComp) [1].

A number of noteworthy efforts have already been conducted within fault detection and repair based on the principles of biological development. In the embryology work conducted at York [5] and EPFL [12], experiments have been conducted, using FPGAs with extended 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 [5] where principles of biological immune systems were

[0] The order of the author's names is purely alphabetical

adopted to attain fault-tolerance.

The text is organised as follows. In section 2 we present a short description of todays digital EHW technology, FPGA and MIT's proposed AComps. We then consider the requirements of digital circuits in section 3. In section 4 we highlight the technological requirements imposed by design by evolution and evolution with developmental principles. We then consider fault-tolerance in section 5. To conclude, in section 6, we extract a summary of the main ideas of the earlier sections and suggest possible future implementation platforms.

2 Technology Platforms

In this section we present two technologies — FPGAs and Amorphous Computers representing two fundamentally different ways of approaching computation. FPGAs are well-known electronic devices for implementing digital circuits. An AComp is a proposed platform for amorphous computing which is based on biological development principles. Although both are based on a cellular architecture and thus quite similar at first, they are fundamentally different in many ways.

2.1 Field Programmable Gate Arrays (FPGAs)

In digital EHW, FPGA's may be seen to be the target technology as they provide a re-configurable platform and chips are commercially available. The main elements of FPGA chips are configurable logic blocks (CLBs) connected together in a grid format and configurable routing resources. In addition, configurable input/output blocks (IOBs) are connected to the grid at the perimeter of the chip as shown in Figure 1.

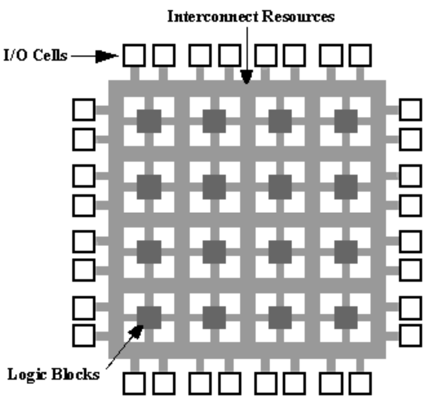

Figure 1. Typical Topology of Today's FPGA

To offer the possibility for complex designs, FPGAs are both expanding in size i.e. increasing the number of combinational logic blocks per chip, increasing the complexity of these blocks and introducing vast and varied routing resources.

Configuration can be categorised into non-partial or partial configuration. The former may be applied to any FPGA and enables the complete chip to be configured. The latter is offered on some devices and allows a portion of the chip to be reconfigured whilst the remaining parts of the chip retain their current configurations.

Illegal configurations are avoided by constraining configurations. To date two different methods to solve this problem may be seen. Either the logic blocks themselves have an architecture where it is not possible to configure them illegally (as seen in XILINX 6200 series) or the users themselves do not have direct access to the routing resources during the evolution process (such as in XILINX Virtex series).

2.2 An Amorphous Computer (AComp)

Amorphous computing [1] 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 2 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 the unit.

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

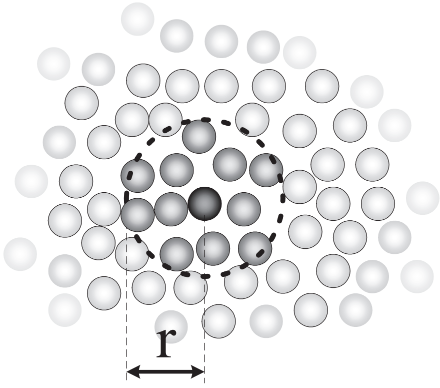

Figure 2. Communication Radius in an Amorphous Computer

required for an AComp. 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.

Sciences such as biology and physics have been adopted as a source of metaphors for the field of amorphous computing, which amongst others led to the development of the *Growing Point Language* (GPL) [6]. This is a language that is loosely based on botanical growth. A GPL Program is compiled to a finite state machine which is identical for all the units in the amorphous computer. The units are able to "differentiate" to different types, thus forming the predefined pattern intended by the programmer.

The basic unit in GPL is a "growing point", which is a shared activity and state of a group of cells which can be propagated to an overlapping neighbourhood. Growing points can split, die or merge with other growing points. Where a growing point passed through, the states of the processors are adjusted to represent the "material" left behind. Growing points move around following the rules of a programmed tropism. This can be towards a source of a diffusing message, away from it or in such a way that the concentration of this "pheromone" remains more or less constant.

To illustrate GPL, as a means to grow low-level electronics, we can consider the example given by Coore [6] of the generation of a pattern representing a connected series of CMOS inverters.

Figure 3 illustrates the schematic of a CMOS inverter with different colours representing different physical materials. To represent such a schematic on an AComp, growing points will move across the collection of processing units dropping AComp 'materials' along their path. Pheromones are secreted from key locations to guide i.e. attract, the

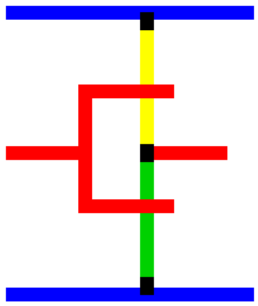

Figure 3. Model of a CMOS Inverter

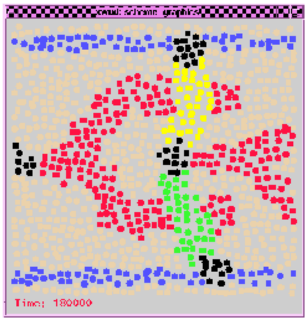

Figure 4. Resulting CMOS inverter [6]

growing points. As an example, a pheromone secreted from locations on the upper and lower blue material-lines (source and drain potential for the inverter) instantiates a north-south coordinate system, used to draw the vertical parts of the inverter. Figure 4 illustrates a sample inverter from the resulting inverters. Note that all inverters are not exactly identical, as they are grown on a non-uniform substrate of cells where some cells may be malfunctioning. The shared GPL program, however, assures a robust generation of fundamentally alike patterns of inverters.

3 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. However, a produced signal must be given time to stabilise before the correct value can be used by the next unit. Today we talk about synchronous or asynchronous design techniques used to achieve this stability. In asynchronous design a particular challenge is to attain stability in sequential circuits, since these incorporate feedback loops.

A digital design assumes the possibility to send a directed message from one output to a designated input over some communication path. Inconsistencies in signal levels can arise if a such a communication path e.g. a wire, is simultaneously driven by more then one signal.

The properties discussed above place requirements on the underlying technology. It assumes the presence of an abstraction mechanism and methodologies to control it. Considering FPGAs, programming the technology is based on the abstracted logic levels i.e. a string of 1's and 0's. An on-chip oscillator may be provided, but it is up to the designer whether he uses it to control the design. Compared to ASIC technology, FPGAs constrain the design as the circuit has to be mapped onto a fixed structure of logic and routing resources. On the other hand, it makes the designers job much easier. Synthesis and mapping tools are used to move the designers high level circuit representation to an FPGA representation, thus atomising a part of the design process. It is at these latter stages that technological constraints are met. The same design may be implemented on an ASIC platform and achieve better performance. However, whether using traditional or evolutionary design methods, the reconfigurability feature of FPGAs makes design changes much easier and thus provides an advantage over ASIC design.

Whether we consider ASIC or FPGA design, we are considering fixed communication, either decided at design time (ASIC) or at mapping time (FPGA). What if we free ourselves from the rigidity of ASIC and especially FPGAs and move towards an AComp style structure? Here we have non-uniformly distributed processors in a 2D plane, an increased number of possible neighbouring processors and a freer communication medium. Can we design efficient digital designs on such a technology? Which limitations related to digital circuits will this extra freedom impose?

Looking again at the digital requirements we need some form of control. In an AComp like structure, a clock is not the way to go i.e. placing a centralised control on an inherently decentralised system. Therefore, control would have to be asynchronous and requires an asynchronous methodology suitable for local broadcasts. Using local broadcasts, we also require a methodology for preventing collisions.

By using radio communication as the medium for the local broadcasts, as suggested in [1], larger neighbourhood sizes would result. This opens new possibilities for implementing a digital circuit on an AComp. A task or subtask may be broadcast over the large neighbourhood, looking for a processing unit to pick it up. In this way no fixed mapping of tasks to processing units is required. As such, the designer takes advantage of the flexible communication medium as well as the redundancy of the processing units.

To achieve stability on an AComp, it will be important to abstract from the properties of the processing unit technology such that a logic '1' or '0' can again be above or below some kind of threshold. This off course assumes that the digital design is implemented using a standard digital computation model. Moving towards distributed AComp style computation may give improved results.

4 Suitability for Evolution and Development

For a technology to be suitable for digital design, a methodology has to be found to aid the design process. The design productivity gap in the electronic industry is a well known fact. How can the design community utilise the design capacity that technology is offering and at the same time ensure its correctness? To find solutions to the problem of developing large complex designs new design paradigms are required [3][2].

By using evolutionary techniques in the development of hardware, a larger design space may be searched than that searched by more traditional techniques. This is achieved by removing some of the design constraints built into traditional design techniques which are often built into traditional design tools. In addition, evolution removes constraints imposed by the human designer. These constraints are limitations to the designer's thought process which limit the ability to think out all possible solutions for a given design.

If evolution aims to reduce the constraints of traditional design, is it possible to reduce the constraints of the technology itself by introducing technologies better suited to evolution? In digital EHW today, FPGAs may be seen to be the main target technology. They not only provide a reconfigurable platform, essential for realistic evaluation of individuals but also chips are commercially available. However, are today's FPGAs "evolution friendly"? Do they exhibit features which enable us to fully utilise the power of evolution?

To answer this question we need to look at the characteristics of evolvable hardware. Four important characteristics are the size of the phenotype required, flexibility and speed of configuration and robustness of the evolved design.

When we consider programmable technology, the actual phenotype is a result of the direct translation of a program for the technology. In the case of FPGAs we are talking about configuration data. Today's FPGAs offer more and more resources, requiring more and more configuration data. Using a typical one-to-one mapping from genotype to phenotype implies that the size of the genotype required is getting larger and larger. A large genotype increases resource requirements for the evolution process and, as such, is a disadvantage for evolution.

In intrinsic evolution we wish to test each individual on the technology platform itself to obtain a more realistic fitness evaluation than that obtained through simulation i.e. extrinsic evolution. Considering today's FPGAs and using the Xilinx Virtex series as a case study we see that configuration of a chip takes approximately 20 to 30 seconds. To evolve more complex design solutions we can expect that the combination of population size i.e. the number of individuals to be evaluated per generation, and the number of generations required to reach a solution will make evolution of a solution to the problem infeasible due to this slow interface.

Fitness evaluation is often the bottleneck of the evaluation process. In addition to this slow interface, the large phenotype resulting from the large genotype also slows down the evaluation phase. Fitness evaluation is particularly important when dealing with real time solutions which require that fitness evaluation be faster than the incoming data rate.

FPGA features such as partial-reconfiguration may be used to speed-up the evaluation phase as a smaller amount of configuration data is required to program the device for the new individual. Another feature in the fitness evaluation is the ease in which test data may be fed into the design and results extracted. This is possible with FPGAs but requires planning. The required signals must be mapped onto output pins since access to internal data is not possible.

To fully exploit evolution, the evolution process should be able to fully exploit the underlying technology parameters i.e. push the circuit behaviour beyond that of a typical digital circuit. However, taking advantage of the physical parameters may give chip dependence problems and/or reliability problems.

To summarise, we need a technology where complex circuits may be implemented without requiring a large amount of program data, fast programmability and reprogrammability perhaps with some kind of partial programming, access to evolution data in the phenotype technology and more robustness.

If we look to amorphous computer technology do we have these features? This is of course hard to say since an amorphous computer doesn't exist today. However, from the principles of amorphous computing and the proposed amorphous computers we can attempt to answer this question. It will of course be programmable and therefore reprogrammable.

The computer is designed around the notion of an abundance of processing units. Initially this suggests a very large phenotype. However, most of the phenotype information on an FPGA is routing data due to the vast amount of routing resources. Note this pressure on the configuration data is reduced drastically in an FPGA structure such as that proposed in [7] and extended in [8]. In the AComp, although there is a greater routing freedom, not having a physical abundance of routing means that no configuration data is wasted on configuring unused routing resources. Not being able to identify a particular processing unit also indicates that unused processing units will not have to be configured. As such, intuitively, an AComp style structure should reduce the size of the phenotype.

Access to particular data is likely to be a problem. The amorphous computing paradigm itself relies on the fact that the user does not have access to particular processing units. As such, access to information within the design will not be possible. This also means that the feature of partial programmability will not be feasible. This feature relies on the fact that you can access the exact part of the design required so as to only make changes to the parts of the design that differ from the individual implemented. One main feature of the amorphous computers, however, is robustness and is discussed in detail in section 5.

Evolutionary techniques are seen to be limited due to their resource-greedy nature i.e. limited to small non-complex circuits. Firstly this is due to the problem related to genotype/phenotype mapping. As the circuit complexity increases so does the genotype, assuming a one-to-one mapping from genotype to phenotype. Secondly, we have a slow evaluation phase due to the large phenotype required for today's technology. We know that partial evaluation may ease the slow evaluation since configuration is speeded up. However how can we improve the genotype/phenotype problem?

If we can shrink the genotype, we reduce the resource requirements of the evolution process but move the complexity over to the mapping process. It should be noted that a bridging evolution friendly FPGA may be used to simplify the mapping process as proposed in [8]. This assumes that the genotype is mapped to this bridge and the bridge is mapped further to the phenotype.

We see two possible ways to shrink the genotype involving different aspects of biological development. As a smaller step towards replicating biological development, we may include some form of growth in the genotype representation. An example of such an abstraction are growth rules based on L-systems [10][11]. A first attempt at achieving growth based on L-systems for digital design may be found in [8]. Here rules fire when their conditional clause is matched to produce additions and/or changes to the growing phenotype. A problem here is that it is hard to control the size of the phenotype eventually grown. Can it actually be implemented on the resources offered by the implementation technology? That is, the underlying technology places constraints on the grown phenotype to within the limits of the technology.

Considering FPGAs we are talking about a finite amount of CLBs and routing resources and a non-infinite number of ways of mapping the design to the technology. An AComp style technology also has a limited amount of resources but with the freer structure, achieving routing may be seen as almost infinite and with redundant processing elements there is a non-finite but expandable amount of computational resources. As such, an AComp style structure may offer an advantage for this type of growth method over an FPGA.

As a next step, the developmental process can be taken closer to the principles of real biological cellular development. This would then not only incorporate the increase in size (real biological "growth"), but also add the possibility to describe enormous complexity in the phenotype. This description would be based on a limited language and using local messaging, the intricate interaction of cellular states would be achieved.

In general, An initial unit holds the building plan (DNA). As opposed to a descriptive plan, this plan is generative, meaning that it describes what lower-level, local actions have to be performed to build the global phenotype. During the building process units can communicate with other local units, move, change shape, multiply or die. All units hold the same building plan, have an internal state and can exhibit group-wise behaviour patterns as a result of shared states and signalling. Again this form of development may lead to a large phenotype. This method would also require that each cell holds the building plan for the design i.e. a large genotype is stored in every cell. In FPGA technology this would be a problem as there is very little memory in a single CLB. On the other hand, an amorphous computer is designed based on developmental principles and each processing unit should be built with sufficient memory to hold the genotype.

5 Fault Tolerance

Why do we need fault tolerance? As a first and general fact, it makes a designed system operate more reliably. This is true for a system designed using traditional techniques, but also for results of non-traditional design methods such as evolution. For the domain of EHW, it is interesting to investigate the advantages that a fault-tolerant platform might offer over a non-fault tolerant one.

In a conventional, non fault-tolerant technology, unforeseen events may occur which prevent proper operation of an implemented design. For a technology to be fault-tolerant, the implemented design should continue to operate undisturbed by the events, requiring no fault detection mechanism from the design itself. Ideally, this property should be present, independent of the design that will be implemented.

In an amorphous computer, correct functionality will be attained despite the possibility of a limited amount of malfunctioning units. In other words, the computational nature of an AComp 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.

When considering amorphous computers, we are interested in finding out which mechanisms are behind this fault-tolerance. In order to answer this, we have to abstract the basic properties of GPL and relate them to the properties of the implementation layer i.e. the AComp.

The first property is that a task is defined at a level of groups of units instead of singular units. This automatically means that for each subtask of the task (for example drawing a line segment), a multitude of underlying units of the AComp are involved. Another way to put this is that the level of abstraction at which the task is specified is higher than the individual units of the implementation platform. This redundancy thus ensures robustness against processing unit breakdown.

A second property is that communication in GPL directly uses the underlying communication model of local broadcasts. In GPL, an example communication is the long-range pheromone broadcast. This is implemented using local broadcasts. Since there are many neighbours for a given processing unit, there are many neighbours that may forward the local broadcast. As such, we have an inherent local communication redundancy exploited for long-range communication. This redundancy ensures the fault-tolerant arrival of a long-range message.

Although not commercially available, producing an AComp in the form of a device like a digital chip is within reach. Compared to the properties of traditional digital devices the fault-tolerant nature of the AComp has interesting advantages for the manufacturer. Indeed, since production of 100% functional devices is a difficult task due to low yield rates, producing a device that is tolerant of limited deficiencies is an interesting prospect. Assuming a similar view to manufacturing a digital chip, we would obtain a considerable increase in yield rates. In addition, and in our view more interesting, this offers the possibility of building larger devices, not constrained by the obligation to produce them 100% functional.

One thing to note about a device manufactured without a 100% functional guarantee is that the exact specification that we are used to today, will be replaced by a non-exact specification. Specifications will not provide the designer with guaranteed data figures or ranges of figures but will provide the designer with guidelines for achieving continuous operation. In the case of an AComp this might for example be a minimal neighbourhood size to adopt in order to include enough working processors.

A digital design implicitly assumes that the implementation platform is 100% functional and 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. This may be required over a longer lifetime or in difficult environmental conditions.

6 Putting It All Together

Let's first consider today's digital EHW platform — FPGAs. As stated, this platform does not fully exploit evolution. Neither does it support any form of genotype expansion mechanism like growth rules or developmental principles. It is, however, very suitable for digital design but does not protect the design against technology failures such as CLB or routing switch failures.

Taking a step towards a more evolution friendly FPGA, our group proposed a simplified FPGA architecture (sblock architecture) [7]. An essential feature of a realistic platform for EHW is that it is available. This architecture is not likely to be mass produced due to its limited routing resources which, although a benefit to design by evolution, is a disadvantage for traditional design methods.

Much research is looking at new methods of representation including the development principles discussed. These new representations will help to reduce the complexity of the genotype but increase the complexity of the genotype-phenotype mapping. The sblock FPGA architecture is further proposed as an evolution friendly bridge to today's FPGA chips e.g. Virtex. This both simplifies the genotype-phenotype mapping problem and enables new representations to be tested out on a virtual evolution-friendly FPGA. Further mapping to a physical FPGA is available, if required. A further description of this virtual FPGA and the mapping solution may be found in [9]. Intuitively, this sblock architecture may be said to be developmentally friendly in its communication structure as routing is restricted to local neighbour communication — a feature of biological development. This issue is further discussed in [8].

A next step, to both bridge the genotype-phenotype gap and introduce a mechanism usable in fault detection and repair, is to incorporate a much more complete version of biological development in the technology i.e. to provide the ability to re-grow a circuit. One requirement is to provide sufficient memory at each CLB to store the full building plan. A second requirement is a two layer communication system. The first layer, the digital layer is for communicating logic signals i.e. typical design signals. The second layer handles the biological signals, simulating biological cell-to-cell signalling in order to let interpretation of genes be influenced by the neighbourhood the cell is in. A local highly connected system is appropriate with e.g. 8 neighbours per cell, communicating over multiple parallel wires.

Going further in the direction of inherent fault-tolerance, an AComp offers some fundamental advantages. Rather than having to re-grow the circuit itself, the distributed nature of the Amorphous Computing computational model makes a design less vulnerable. This is due to the abundance of units which protect the design from being dependent on individual units. Clearly the price to be paid here is the apparent incompatibility of digital circuits and the Amorphous Computing way of implementing a design i.e. different computational models.

In section 3, we proposed a way of mapping a digital design to an AComp. However, instead of trying to translate from a digital to an Amorphous Computing model, we could consider moving towards a new computational model for digital design similar to that of Amorphous Computing. A reason for this move is that fault-tolerance is expected to gain more and more importance in the future. Moreover, since AComps are based on biological cellular systems, they are quite well suited to implement developmental principles, already mentioned under the need to evolve larger designs. In this way, both large designs and large devices seem more attainable on an AComp like device. The major bottleneck in producing large-scale AComps seems to be the implementation of the communication system. Suggestions like RF-communication were already made by MIT [1]. Other possibilities might be communication through light or ultra-sound.

Although AComps might offer reasonable evolution-friendliness (section 4) together with fault-tolerance, it is rewarding to investigate the architecture of a platform that would offer almost all properties wanted and moreover be suited to implement a digital circuit. Although this seems technologically unattainable at the moment, we propose the general principles of an implementation platform that exhibits these properties.

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. 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.

This mechanism is related to the fault-detection and repair architecture with two communication layers proposed above. In the architecture discussed here, however, the entire task of detecting a failure and replacing the part is taken care of using chemical principles, which makes re-growing around broken parts no longer necessary. 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.

7 Acknowledgements

- The authors would like to thank Ellie D'Hondt for her time and input in the discussions related to Amorphous computers.
- Piet van Remortel is funded by a PhD grant from the Flemish Institute for the promotion of Innovation by Science and Technology in Flanders (IWT).

References

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

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

[3] S. I. Association. *The National Technology Roadmap for Semiconductors*. 1997.

[4] P. J. Bentley and S. Kumar. Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem. In *Genetic and Evolutionary Computation Conference (GECCO '99)*, pages 35–43, 1999.

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

[6] D. N. Coore. *Botanical Computing : A Developmental Approach to Generating Interconnected Topologies on an Amorphous Computer*. PhD thesis, Massachusetts Institute of Technology, 1999.

[7] P. Haddow and G. Tufte. An evolvable hardware FPGA for adaptive hardware. In *Congress on Evolutionary Computation(CEC00)*, pages 553–560, 2000.

[8] P. Haddow, G. Tufte, and P. van Remortel. Shrinking the genotype: L-systems for EHW? In *submitted to 4th International Conference on Evolvable Systems (ICES01)*, 2001.

[9] P. C. Haddow and G. Tufte. Bridging the genotype-phenotype mapping for digital FPGAs. In *submitted to the 3rdt NASA/DoD Workshop on Evolvable Hardware*, 2001.

[10] A. Lindenmayer. Mathematical Models for Cellular Interactions in Development. *Journal of Theoretical Biology*, 1968.

[11] A. Lindenmayer. Developmental Systems without Cellular Interactions, their Languages and Grammars. *Journal of Theoretical Biology*, 1971.

[12] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward self-repairing and self-replicating hardware : the embryonics approach. In *The 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 205–214, 2000.