Technical report
Hardware morphogenetic developmental system

Daniel Roggen, Dario Floreano
Autonomous Systems Laboratory
EPFL, Lausanne, Switzerland
http://asl.epfl.ch
name.surname@epfl.ch

November 1, 2004

Abstract

Indirect genotype to phenotype mappings in the form of developmental systems may allow better scalability to larger phenotypes in evolvable hardware. This report reviews developmental systems used in evolvable hardware and it proposes a new classifications based on hardware characteristics. It then describes a genetic encoding and developmental system called the *morphogenetic system* which has been designed for multi-cellular circuits. This morphogenetic system is inspired upon gene expression and cell differentiation but it focuses on efficient hardware implementation. An hardware implementation on the dynamically reconfigurable POEtic circuit is described. It uses serial arithmetics and time-multiplexing to minimize ressource use.

1 Introduction

Scalability to larger circuits is a problem frequently encountered in evolvable hardware [55, 17, 15, 12]. Genotype to phenotype mappings based on a developmental process have been proposed to improve the scalability of the evolutionary approach to larger phenotypes [57, 19].

This report first reviews developmental systems used in evolvable hardware. It proposes a classification of those developmental systems which is based on characteristics linked to their hardware implementation.

The report then describes the hardware implementation of a genetic encoding and developmental system based on gene expression and cell differentiation called the *morphogenetic system* [41]. This morphogenetic system has been developed for a multi-cellular reconfigurable circuit called POEtic [53]. In particular it focuses on low computational complexity and efficient hardware implementation. Previous experiments have shown that it performs well compared to a direct genetic encoding when evolving neural networks to do pattern recognition or robot control [41] and that it may scale better than a direct genetic encoding when evolving phenotypes to resemble specific 2D patterns [40]. The hardware implementation is a multi-cellular (distributed) implementation

which uses serial-arithmetics and time-multiplexing to reduce ressource uses, yet development is still achieved at relatively high speed.

This report is organized as follows. Developmental systems used in evolvable hardware are reviewed and classified in section 2. The architecture of the POEtic chip is summarized in section 3. The hardware implementation of the morphogenetic system is described in section 4. Finally the results are discussed in section 5.

2 Developmental systems

Over the last 40 years, several scientists have proposed mathematical models of development, notably Turing's morphogenesis [52], Lindenmayer's L-Systems [26] and Kauffman's random boolean networks [16].

In the early 1990s, Kitano proposed a graph generation system inspired upon L-Systems to generate the connection matrix of neural networks [18]. Mjolsness et al. proposed another evolutionary developmental model which consisted in rules formed of recursive applications of growth equations [38] with the objective of improving the scalability and generalization capabilities of neural networks by reducing the size of the search space. Vaario et al. used L-Systems to control the growth of neural networks [54].

In his 1993 review of the evolution of artificial neural networks, Yao distinguished between direct encoding schemes, where all connection parameters are individually encoded in the genetic string, and indirect encoding schemes, where the details of the architecture, such as the connections, are left to a training scheme or to developmental rules. In particular he mentioned that indirect encodings allow a more compact representation of the network's connectivity and may be more biologically plausible [57].

De Garis, in his 1992 PhD thesis, predicted that the combination of evolution and development would be applied to electronic circuits and he called this field Artificial Embryology [3].

The following section reviews developmental systems used in evolvable hardware. Afterwards, a classification of developmental systems used in evolvable hardware is introduced.

2.1 Growing electronic circuits

Developmental systems may improve evolvability or scalability in evolvable hardware, or to they can introduce fault-tolerance and adaptivity in hardware. Research in this field is reviewed below.

Circuits represented in a Hardware Description Language (HDL) can be evolved by a developmental process consisting of rewriting rules following a HDL grammar, evolved by "production" genetic algorithms [37]. This approach is motivated by the need for automated hardware design processes for the evolution of large circuits. Rewriting is controlled by a tree-structured chromosome where each node contains a production rule that is recursively applied to a starting symbol. Evolved HDL circuits are simulated to evaluate their fitness. The approach has been used to generate controllers for artificial ants that must follow a possibly interrupted food trail. The authors suggest that this approach

exploits regularities in the phenotype and therefore may be scalable to more complex problems [13].

Motivated by the scalability problem which is encountered with direct genetic encodings, Haddow and Tufte explored an indirect genetic encoding based on L-Systems to evolve electronic circuits on a custom "virtual" FPGA. The virtual FPGA [10] has the features deemed necessary for unconstrained evolution [9]. It does not exist as a custom chip, but is implemented over a Xilinx Virtex FPGA. The genetic encoding is an adaptation of L-Systems [26] for the evolution of hardware. It consists of a set of rewriting rules recursively applied to a "growing" string starting from an initial axiom (starting string). Two types of rule are used. Change rules replace a part of the configuration string with another one of equivalent length. Growth rules are used to allocate new logic elements (called *s-blocks* in the virtual FPGA terminology) on free space around the s-block which triggered the rule. The starting axioms and development rules are evolved using a standard genetic algorithm. The system was used to evolve circuits with specific distribution of s-blocks on a 16x16 cell array with moderate success [11]. Subsequent work considered the restriction of s-block configuration to specific s-block types to help evolution find specific types of circuits [50] and L-System rules were extended to be contextual and to control cell death. Applications included the growth of structure from a single starting cell (achieving a specific target size with a limited number of developmental steps), the differentiation of cells (a number of different cell types should be present after the developmental process) and the formation of patterns. In the last case, a symmetrical pattern of cells had to be found within a limited number of developmental steps. Those circuits were not implementing a particular functionality and had limited size: 3x3 for the growth and differentiation, and 4x4 for the pattern formation. The developmental mechanism was implemented in hardware on a dedicated processor [51] and the genetic algorithm was executed in software on a standard desktop computer.

Gordon et al. explored a model of biological development for the evolution of electronic circuit [8], which is akin to a minimalistic gene regulatory network with binary protein concentrations in a locally interconnected 2D array of cells. Cells implement a binary function of 4 inputs by means of a lookup table (LUT). Development rules have preconditions indicating which proteins must be present or absent for the postcondition to occur. Postconditions can either generate a protein or change the functionality of the cell which occurs either through a change of the cell's input and output connectivity or a change of LUT. Cells were implemented in a Virtex FPGA and unconstrained evolution was performed to evolve simple arithmetic functions. While the evolvability of the system was lower than a direct encoding, more regular and repeatable structures appeared in the content of the look-up tables. The authors argued that this is a key point of developmental systems and they suggested that evolving larger adders would become easier because they can be implemented with very regular structures such as the ripple carry adder. In further work the model was enriched by introducing diffusion and a finer detection of protein concentrations in neighbouring cells. The 2-bit adder could be evolved with this new developmental model when the genetic algorithm was replaced with a hill-climbing algorithm [7].

Developmental Cartesian Genetic Programming (DCGP) is a biologically motivated cell-based developmental model [36]. In DCGP a cell implements

both a functional part (the phenotype) and a developmental part. The functional part corresponds to the logic gates. The developmental part is a program that has as inputs the cell's connections, function and position in the circuit and defines the new connections, function and whether the cell will duplicate in the next developmental step. Development starts from a single initial cell and all the cells of an organism share the same developmental program. The developmental program of the cell is evolved. DCGP has been used to evolve binary adders and even-parity functions. It was shown that a moderate degree of generalisation is possible, although rare, by increasing the number of developmental steps to obtain a logic function of an additional input. The authors remarked that the DCGP genotypes are less evolvable than direct encodings, and that, although developmental systems may pay off with larger phenotypes, the issue may be more complex than simply looking for a way of reducing the genotype length.

De Garis approached the evolution of large-scale neural networks by implementing them in a large scale cellular-automaton (CA) executed in a custom hardware architecture designed for fast simulation [4]. The neural connections are grown according to an evolved program. During development, growth signals are sent along synaptic connections. When they reach the extremity of connections they induce growth, possibly altering the synaptic direction or creating branchings. Whenever a synapse reaches another one or a neuron, a connection is established. The sequence of growth signals is genetically encoded. This system has been used to produce waveforms of arbitrary shapes, to halve the frequency of an input signal, to solve the XOR problem, to detect a moving line and to detect a frequency or signal strength [5].

Embryonics (Embryological Electronics) is a large scale integrated circuit that displays properties associated with living organisms, such as self-repair and self-replication, by exploiting a simple developmental process [32, 30, 31]. Inside the Embryonics chip (i.e. a new type of FPGA) a subset of identical rectangular cells forms an "organism" with self-repair and self-replication properties. Cells are made up of reconfigurable logic elements called *molecules* which provide self-testing mechanisms through spatial redundancy. As in living organisms, Embryonics cells differentiate to execute different tasks. All the cells are structurally identical and run the same "program" (i.e. the genetic code of the entire organism). However, depending on the coordinates of the cell in the organism different execution paths are followed resulting in differentiation of the cell functionality. As all cells contain the complete genetic description of the organism, mechanisms such as self-reproduction and self-repair become possible. Self-repair is an alteration of the coordinate system in which faulty cells are avoided by the developmental process.

Miller described a method to evolve the development program inside cells to construct phenotypes of arbitrary size [34]. Cells, organised as a 2D array, have a type and contain internal variables akin to chemical concentrations. The cell program maps input conditions (chemical concentration and type of the cell and of its immediate neighbours) to output behaviours: change of chemical production, of cell type, death or growth of a new cell. The cell program is encoded using Cartesian Genetic Programming [35] and evolved. The system is able to evolve organisms with specific patterns of differentiated cells. Self-repairing properties were illustrated by removing cells and letting the developmental process recover the pattern of differentiated cells. The system is also capable of

adaptation by showing different developmental pathways in function of external environmental signals. Hardware implementation of the developmental model and its use for applications requiring self-repair and adaptation are under-way [27].

Gene regulatory networks (GRNs) describe the interactions among genes and proteins which may in turn activate or repress the expression of other genes. They are the basis of development in living organisms, allowing to build very complex structures from the comparatively small amount of DNA that those organisms have [2, 56]. With the objective of improving the fault-tolerance of bio-inspired electronic circuits, Koopman developed a simplified gene regulatory model that is suited for cellular implementation in electronic circuits [21]. Each cell contains a gene regulatory network that interprets an artificial genome containing the rules of activation or repression of the production of proteins. The protein production rules take into account the state of existing proteins in the cells. The system was used to evolve circular patches of specific target sizes and the fault-tolerance of the system was shown to be better than that obtained with a direct genetic encoding. A compact hardware implementation using fixed-point and bit-serial arithmetic is suggested for implementaion on the POEtic circuit [53].

Models of development are also interesting to understand the complex regulatory pathways in organisms, for example to design drugs. Hardware implementations of gene regulatory networks may allow faster simulations than their software counterpart and compact implementations may be achieved by exploiting analogies between CMOS circuits and GRNs [46]. While this implementation focuses on biological applications, it would be possible to evolve its configuration string and use it as a developmental system for evolvable hardware, with the advantage of the very compact implementation and high speed of the analog implementation.

Koza et al. showed that genetic programming can be used to evolve circuits [22]. In Genetic programming the chromosome is a tree representation of a computer program or of a circuit (each node of the tree represents an element of a circuit). The process by which the tree-representation of the chromosome is decoded to form a circuit may be considered a minimalistic form of a developmental system. Koza and others showed that genetic programming can be enhanced by using automatically defined functions which provide modularity and gene reuse [24, 23]. This method has been successfully applied to design low-pass filters, two-band crossover filters, amplifiers, etc [25].

Lohn et al. explored a linear representation of an analog circuit coupled with an unfolding process [28, 29]. This developmental system is used to evolve analog circuits composed of two- or three-terminal elements (resistors, capacitors, inductors and transistors). The chromosome consists in a list of bytecodes that are executed sequentially. Each bytecode indicates which new element to add to the circuit and how to interconnect it. The authors argued that this encoding generates many topologies which are seen in hand-designed circuits, even though not all the topologies can be encoded.

Mattiussi et al. proposed a genetic encoding which is decodable even after major reorganization by genetic operators, and at the same time which allows for gradual changes in phenotype under certain genetic operators. It has been demonstrated by evolving analog circuits [33]. The chromosome is scanned for substrings which identify components, components values (e.g. capacitor

values) and terminal labels. Interconnections among component terminals are implemented as resistors whose values are inversely proportional to the degree of similarity among the strings identifying the terminals. This genetic encoding can always generates legal phenotypes. The authors noted that the genetic encoding is also suited for the evolution of neural networks and gene regulatory networks.

Developmental systems are also used to evolve neural networks and morphologies. For reviews of those fields see [1, 20, 45].

2.2 Classification

To classify developmental systems which are employed in evolvable hardware, we propose a to look at key characteristics of their implementation, described below.

Extrinsic or intrinsic developmental system: Inspired from the difference between intrinsic and extrinsic evolution in evolvable hardware [4], which distinguishes the physical implementation of an evolved circuit from its simulation, we want to distinguish similarly between the execution of the developmental system in software or in hardware.

By extrinsic developmental system we mean that the developmental mechanism is executed in software on a PC. The execution of the developmental system converts the genotype into the phenotype (a circuit) which can then be implemented physically or simulated.

By intrinsic developmental system we mean that the developmental system is implemented in the same hardware as the circuit which is evolved (e.g. the same chip).

Centralized or distributed/cellular developmental system: In the case of an intrinsic developmental system we wish to distinguish between a centralized implementation or a distributed or cellular implementation.

In a centralized implementation a single hardware unit is in charge of running the developmental mechanism in the same hardware as the evolved circuit. This can be a CPU or a dedicated coprocessor in a system on a chip.

In a distributed approach the developmental system is executed by many independent but communicating units. In a multi-cellular electronic circuit this may be called a cellular implementation: each cell implements the developmental process in addition to its normal functionality.

The latter implementation may faster, more scalable, more biologically plausible, and possibly more robust than a centralized implementation, at the expense of more space.

Online or offline development: In online development the developmental process is running continuously to decode the genotype into the phenotype. It may also react to changes in the environment. Online development is biologically plausible and could allow for characteristics which are seen in living organisms such as growth or self-repair.

In offline development the developmental process decodes the genotype into the phenotype in one step. Once the phenotype is obtained the developmental process can be stopped. Although this is biologically less plausible, this may save hardware resources when implementing the developmental system.

In this review most developmental models are extrinsic [13, 7, 36, 34, 22, 21, 25, 33] and few are intrinsic [46, 31, 5, 51]. Among those, some model biological

gene regulatory networks in hardware but do not have the objective to develop circuits (although it could be a basis to do so) [46]. Embryonics [31] employs a direct genotype to phenotype mapping, however evolution is not yet used to find working circuits. To the author's knowledge the only intrinsic evolutionary developmental systems are the CA-based growth of neural networks of de Garis et al. [5] and the L-System-based encoding of Haddow and Tufte [51].

Embryonics and the CA-based growth of neural networks use a cellular implementation of the developmental system, while the L-System-based encoding uses a centralized implementation.

All the developmental systems in this review operate offline, with the exception of Miller's cellular program which showed that intrinsic development can lead to fault tolerant or adaptive development [34].

3 POEtic chip

Evolvable hardware is the creation of electronic circuit by using artificial evolution [14]. However evolution is only one of the three sources, or axis, of biological inspirations that the POE model distinguishes [43, 44]. The three axis are: Phylogeny (evolution) which describes how organisms change over the course of several generations, ontogeny (development) which is the development of an organism starting from a single cell, and epigenesis (learning) which is the adaptation or learning that an organism is capable during its lifetime.

Combining those three axis may lead to hardware with better properties. Development may improve the evolvability of circuits or may lead to self-repairing and self-reproducing circuits [31]. Learning, the process by which the synaptic connections changes under specific stimuli, may allow a controller to improve its response in function of past events or to adapt to new environments [6].

The three-year European project called POEtic [53] set to explore the combination of those three axis in hardware. For this purpose a novel reconfigurable circuit, the POEtic chip, has been developed as a generic platform to implement bio-inspired systems. POEtic is a common project of the Swiss Federal Institute of Technology in Lausanne, the University of Lausanne, the University of York, the University of Glasgow and the Technical University of Catalunya in Barcelona.

The POEtic chip [39, 48, 49, 47] has been developed to be a flexible substrate to implement bio-inspired mechanisms in hardware. One of the objective of this chip is to favour the implementation of mechanisms of self-repair, self-reproduction, development, learning, and evolution [53]. Keeping with the bio-inspired terminology, those circuits are sometimes referred to as *organisms*. The POEtic chip consists in reconfigurable logic which can be *configured* to implement the desired bio-inspired mechanisms, in a way similar to FPGAs.

The features of the POEtic chip are the following.

POEtic system on a chip: The POEtic chip is a SoC (System on a Chip) composed of a CPU, reconfigurable logic and I/O peripherals. Bio-inspired mechanisms that are better suited for a software implementation (e.g. evolutionary algorithms) or communication with I/O peripherals (e.g. to send data out of the chip) can be implemented in the CPU. Organisms are implemented in the reconfigurable logic. I/O peripherals allow communication with the chip's outside (e.g. with a robot or sensors).

Direct and fast CPU access to configuration bits: The configuration bits of the reconfigurable logic can be accessed directly and very fast from the CPU. This is important as evolutionary techniques may need to test lots of circuit configurations before finding suitable solutions. Commercial FPGAs usually require an external interface which slows down reconfiguration.

Documented configuration bits: The POEtic chip offers a documented configuration string. Therefore evolvable hardware experiments are possible and the resulting circuits can be analysed by looking at the evolved configuration string. Recent FPGAs do not offer such possibility anymore.

Hardware self-reconfiguration: Hardware self-reconfiguration of one logic element by another one may allow adaptive hardware (e.g. a logic elements is reprogrammed to act differently depending on environmental stimuli), self-repairing or developing hardware (e.g. logic elements reconfigure spare logic elements with a copy of their configuration bits to recover functionality or instantiate new functional elements).

Support for evolutionary algorithms in CPU instruction set: Evolutionary algorithms are stochastic search algorithms and therefore they make extensive use of pseudo-random number generation. They also manipulate chromosomes (bit strings) at the bit level. The POEtic CPU instruction set provides instructions of that kind to speed up evolutionary processes.

Dynamic routing: The POEtic chip has a novel concept of dynamic routing through which connections between components can be built automatically and at run-time by the hardware in a distributed way. This enables dynamic reorganization of routing within the chip. This feature is deemed necessary for self-repair, self-reproduction and also to react to changing environments (e.g. changes in location of sensors or actuators). In conventional FPGAs physical connections must be planned at design-time and cannot be changed at run-time.

The POEtic chip is illustrated in figure 1. It is composed of two subsystems: the *environmental* subsystem which contains a CPU and its peripherals and the *organic* subsystem which is reconfigurable logic.

In a typical application the environment subsystem is in charge of configuring the organic subsystem, interfacing with the outside of the chip through the numerous I/O peripherals which are available, and also running evolutionary algorithms and measuring the circuit fitness. The organic subsystem is used to implement the actual circuits which have e.g. learning, development or self-repair capabilities.

The environment subsystem is centered around a 32-bit, five-stage pipeline RISC microprocessor with 32 registers and a Harvard architecture. The instruction set contains specific instructions for evolutionary algorithms: random number generations and bit manipulation. An AMBA bus is used to interface with peripherals and the organic subsystem. The peripherals are: 2 UARTs (serial lines), I2C and SPI interface, an 8-bit parallel port, 2 16-bit timers, a 16x16 bit hardware Booth multiplier, and the organic subsystem interface. The latter allows access to the configuration bits and to the status and control bits of the organic subsystem.

The organic subsystem is organized as two layers as illustrated in figure 2. The lower layer is composed of an array of reconfigurable logic elements called *molecules* which are locally interconnected. This is where cells or functional blocks are implemented in the organic subsystem. The above layer is an array of dynamic *routing units (RU)* which are used for long distance connections

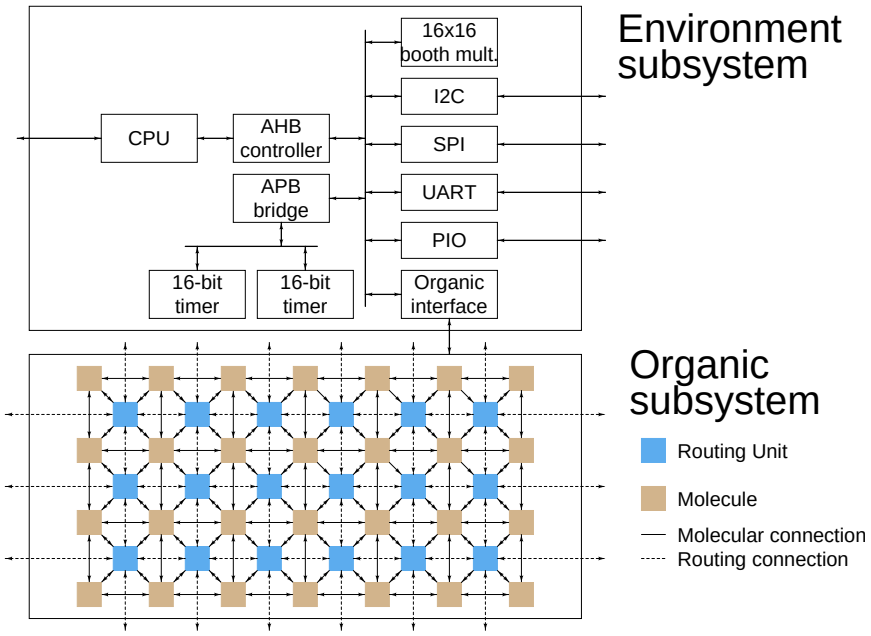

Figure 1: The POEtic chip is composed of two subsystems. The environment subsystem is composed of a CPU and several peripherals. In a typical application it is used to interface with the outside of the chip, program the reconfigurable logic and run the evolutionary algorithm. The organic subsystem is the reconfigurable logic where circuits which may have capabilities to learn or develop are implemented. The organic subsystem is composed of two functional elements: molecules, which are programmable logic elements, and routing units which are capable of dynamic routing.

among cells or functional blocks. A RU is shared by a group of 4 molecules underneath.

Molecules are composed of a 16-bit memory, a flip-flop and a switch box for local communication (see figure 3). A molecule can operate in different modes according to its configuration. In the 4-LUT mode the molecule is a 4-input lookup table. In the 3-LUT mode the molecule is used as two 3-input LUT. This mode allows efficient implementation of arithmetic operations (e.g. comparison, additions). In the memory mode the molecule implements a 16-bit shift memory. The input and output modes are used to send/receive signals to/from the routing layer. A trigger mode is used to synchronize the routing layer. The configure mode allows self-reconfiguration by transferring the content of the molecule memory into the configuration of a neighbouring molecule. Eventually in the communication mode the 16-bit memory is split in an 8-bit shift memory and in a 3-input LUT. This mode may be used for packet-based communication.

The routing array is composed of locally connected routing units which are capable of dynamic routing. The routing array is used for long distance connections. A routing unit is connected to a group of four underlying molecules. The functionality of the routing unit is given by the input or output molecule in this group. Routing units are passive when there is no underlying input or

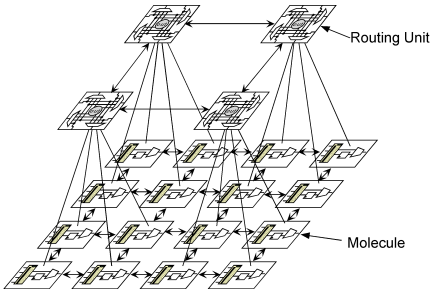

Figure 2: The organic subsystem is composed of two layers on top of each other. The lower layer is an array of molecules (reconfigurable logic elements) and the upper layer is an array of routing units which are used for long distance communication and dynamic routing.

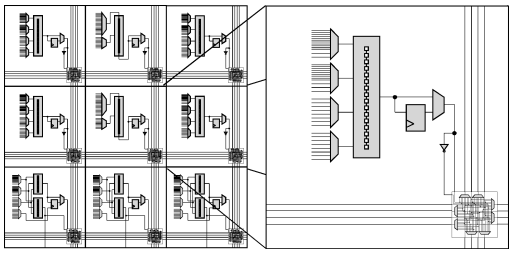

Figure 3: Moleculecular array (left) with close-up on the molecule (right) which reveals the 16-bit memory (middle), the flip-flop and the switch-box for local interconnection.

output molecule.

Routing units situated at the border of the array are connected to pins of the chip. Therefore signals on the chip pins can be accessed from the molecular array via the routing layer. Furthermore the chip is designed in such a way that several chips can be interconnected pin-to-pin to achieve a larger array of reconfigurable logic. In this case routing can be done across several chips transparently.

Routing units are capable of dynamic routing: connections between the routing units are built automatically at run-time. Dynamic routing relies on identifiers which are used as the addresses of the routing units. Those identifiers are stored in the 16-bit register of the corresponding input or output of molecule. A breadth first search algorithm implemented in the routing units is used to find a path between source and target routing units which have the same identifier. Path can also be changed, added or removed at run-time by locally reconfiguring the input or output molecules. For instance, by changing the content of the 16-bit register containing the identifier, the source or target of a path can be changed. Therefore new connections can be made even if they are not initially planned.

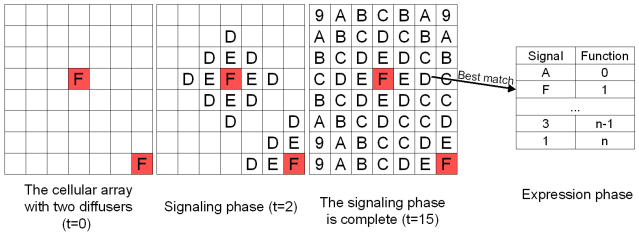

Figure 4: The three arrays on the left are snapshots of the signaling phase with one type of signal and two diffusers (gray cells) at the start of the signaling phase (left), after two time steps (middle) and when the signaling is complete (right). The number inside the cells is the intensity of the signal in hexadecimal. The expression table used in the expression phase is shown on the right. In this example the signal D matches the second entry of the table with signal F (smallest Hamming distance), thus expressing function F_1.

4 POEtic implementation of the morphogenetic system

The morphogenetic system is a genetic encoding and developmental system for multi-cellular circuits which has been designed with compact hardware implementation in mind [41]. As such the computational and memory requirements have been kept to a minimum. Only shift operations, decrements, increments and comparisons are needed. In particular none of the costly operations of multiplication or division operations are necessary. After summarizing the operation of the morphogenetic system, an implementation optimized for space is described for the POEtic circuit.

4.1 Summary

The morphogenetic system assigns a functionality to each cell of a multi-cellular circuit from a set of predefined functionalities. The process works in two phases: first a signalling phase then an expression phase. The signalling phase relies on the ability of the cellular circuit to exchange signals among adjacent cells to implement a diffusion process. The expression phase finds the functionality to be expressed in each cell by matching the signal intensities in each cell with a corresponding functionality stored in an expression table. Evolution is applied to the location of the diffusing cells, and to the content of the expression table.

Inter-cellular communication allows the exchange of signals between adjacent cells. A signal is a simple numerical value (the signal intensity) that the cell owns, and that adjacent cells are able to read, akin to a chemical concentration. Signals may be of different types (i.e of a different chemical nature). The intensity of signal s in cell i is noted by C_s^i.

Special cells, called *diffusers*, own a signal of maximum intensity. Signalling ensures that the signal intensity in the neighbouring cells decreases linearly with the Manhattan distance to the diffuser. The signalling algorithm is illustrated in table 1. First, all the signals are initially tagged as uninitialized, except for diffusers. Signals of each type are processed independently, without interactions among them (i.e. as if they were in different chemical layers). The intensity of

1	Tag in each cell the intensity of the signals as uninitialized. Decode the chromosome to find the location of the diffusers. Initialize the intensity of the corresponding signal to the maximum value of 15.
2	For each signal s and each cell i do:
3	If the cell is a diffuser of signal s, or if the intensity of signal s is already set then skip this cell.
4	Compute the intensity of signal s. It is the value of any initialized signal in a neighbouring cell minus one. If all existing neighbouring cell are uninitialized, then the intensity of signal s remains uninitialized.
5	Repeat step 2 to 4 15 times to complete the diffusion mechanism.

Table 1: Implementation of the signalling phase of the morphogenetic evolutionary system.

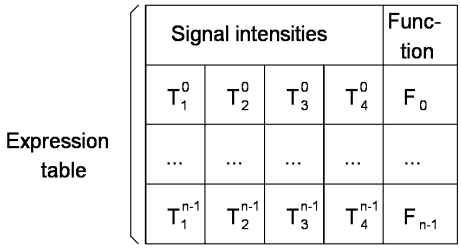

Figure 5: Top: expression table T with n entries. In this case four chemicals are used therefore each entry is 16 bit long.

uninitialized signals is then computed. It is equal to the value of the corresponding signal in any of the neighbouring cell minus one, if that signal is initialized (if it is not, then the signal remains uninitialized). This process is repeated until no more changes occurs in the signals. In the current implementation there are 4 type of signals and the signals are represented by a 4-bit number . Therefore, after 16 steps ($2^4 = 16$) the developmental process is completed. Figure 4 shows an example of the signalling phase in the case of a single type of signal, with two diffusers placed in the cellular circuit.

The expression phase assigns a function to each cell by matching the signal intensities inside that cell with the entries of an expression table T (fig. 5) stored in the genetic code. Each entry of the table contains the intensities of the four signals and the function to express in case of match. The intensity of signal s in the entry j of the table is noted by T_s^j. A cell i is said to match an entry j of the expression table when the distance $d = \sum_{s=1}^{4} DOp(C_s^i, T_s^j)$ is minimum. The distance operator DOp is the Hamming distance.

4.2 Hardware implementation

The hardware implementation uses 4 type of chemicals and 4 type of cells since those are the settings used in most experiments done with the morphogenetic system [41]. As an additional motivation, 4 signals can be efficiently stored in a single memory molecule. Note however that more type of cells can be handled with minor modifications.

The implementation consists in an array of locally interconnected *morphogenetic elements* which implement the signalling and expression mechanisms

within cells.

In the 3-layered view of POEtic cells which distinguishes the genotype, mapping and phenotype layers [53], the morphogenetic element would belong to the mapping layer and to the genotype layer because it contains the expression table which is part of the genotype. The genotype layer is completed by the POEtic CPU which configures the diffusers in morphogenetic elements prior to operation.

To keep the implementation generic, the phenotype layer is not part of the morphogenetic element as it is application dependent. Instead an output of the morphogenetic element indicates the functionality (cell type) that the phenotype layer has to implement.

Serial operations are favoured over parallel operations, therefore trading speed for smaller size. In particular bit-serial arithmetic (one bit of result is computed at each clock cycle) is used. The molecules of the POEtic circuit are well suited for this: shift memory molecules store efficiently 16-bit numbers and a serial addition, substration or comparison can be done with a single 3-LUT molecule (i.e. one LUT of the molecule computes the sum while the other computes the carry, which is memorized by the flip-flop for the next clock cycle).

Although the number of clock cycles for serial arithmetics depends of the bit length of the operands, it is best to extend the operation to a multiple of 16 clock cycles which corresponds to a complete rotation of values stored in shift memories. We define the term *molecular cycle* as 16 clock cycles.

Morphogenetic elements need to exchange signal intensities and *valid* flags which indicate whether signals are initialized (i.e. whether signal intensities are valid). The dynamic routing mechanism of the POEtic chip is used for this purpose and serial communication (data is transferred bit by bit) is used to minimize the number of communication lines. Eventually a single connection is used to send the 4 signal intensities and the 4 valid flags to connected morphogenetic elements.

The morphogenetic element has 4 inputs and one output which are connected to the 4 neighbouring morphogenetic elements by the dynamic routing mechanism at run-time. Plus it has another output which is the functionality the phenotype should take. The block schematic of the morphogenetic element is illustrated in figure 6.

It is composed of two main parts: the signalling block and the expression block. The signalling block handles I/O, the diffusion mechanism, and provides the signal intensities for use in the expression block. The `cellular input` receives the signal intensities and the valid flag from neighbouring elements over the dynamic routing layer. The valid signal is selected by `input select` and then decremented by `decrement-compare`. `Normalize` renormalizes the signal if it is invalid or below zero. Eventually the signal intensity and the valid flag is stored in `diffusion memory`. It is then provided to the expression block and afterwards it is sent to the `cellular output` at the next developmental step.

The expression block provides the `function` output of the element by sequentially comparing the signal intensities with each of the entries of the expression table. The content of the shift memory `expression table` is compared to the signal intensities in the cell with the `Hamming distance`. This distance is compared to the current `shortest distance` by `compare distance`. If the current distance is shorter the shortest distance is updated and the `best function`

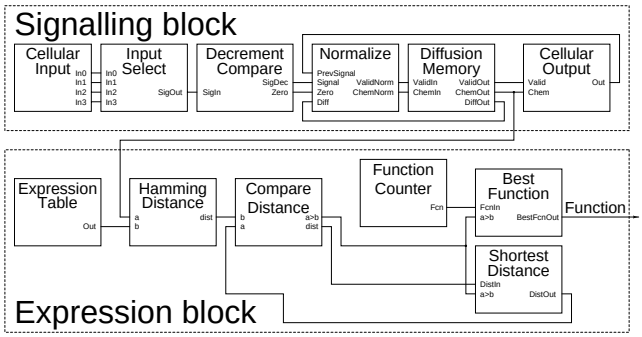

Figure 6: The morphogenetic element is composed of two main blocks: the diffusion block handles input, output and implements the diffusion mechanism, whereas the expression block finds the functionality to express in the cell according to the signal intensities. The `cellular input` and `cellular output` subblocks exchange signals with the connected morphogenetic elements via the dynamic routing layer of the POEtic chip. `Function` is the output intended for the phenotype of the cell: it indicates the functionality that the cell must take.

found until now is updated with the ever increasing value of the `function counter`.

The morphogenetic element continuously executes the developmental program. One developmental step consists of the sequential execution of the signalling and expression blocks. Figure 7 illustrates this and indicates the time necessary to complete each operation. The signalling block operates in two modes. During the first 5 molecular cycles it sends/receives the signal intensities to/from the neighbouring morphogenetic elements and updates the signal intensities according to the signalling rules. In the following 11 molecular cycles, the signalling block continuously provides at each molecular cycle the 4 4-bit signal intensities (i.e. a sequence of 16 bits) to the expression block to implement the matching process. The expression block alternates between two phases which implement the expression mechanism of the morphogenetic system. Two molecular cycles are required for each entry in the expression table because computing the Hamming distance and comparing it to the previously stored Hamming distance (i.e. to know if a better match has been found) are operations taking one molecular cycle. The sequence of operations is described below.

1. I/O: The signalling block sends the intensity of its signals and its valid bits to connected morphogenetic elements. At the same time it receives those of its neighbours. Each signal is sent and received during one molecular cycle: the first bit represents the *valid* bit (i.e. if the signal is initialized), the following 4 bits are the signal intensity, and the remaining 11 bits are zeros.

2. DIFF: At each molecular cycle the intensity of one signal is updated according to the diffusion rules. After 4 molecular cycles all the 4 signals have been updated. Afterwards the newly computed signal intensities in the morphogenetic element are provided to the expression block for the expression phase.

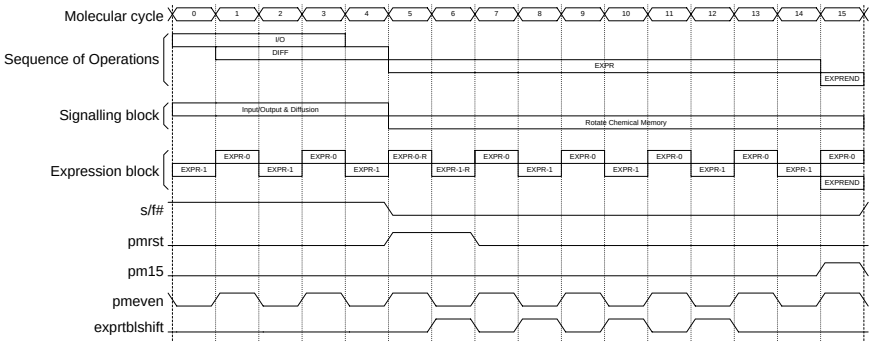

Figure 7: Sequence of operations for one developmental step, with the corresponding control signals.

3. EXPR: The expression phase is executed to find the index of the entry in the expression table best matching the intensities of signals in the morphogenetic element. Expression consists of two molecular cycles for each entry in the expression table, plus two molecular cycles to initialize the process. Those molecular cycles are EXPR-0 and EXPR-1. During phase EXPR-0 the Hamming distance is compared with the shortest stored distance. During phase EXPR-1 the Hamming distance is computed, the shortest distance and the best entry are updated if needed, and the index in the expression table is incremented. EXPR-0-R and EXPR-1-R correspond to the initialization of the expression phase.

4. EXPREND: The index is transferred outside of the morphogenetic element for use by the phenotype layer of the cell.

A complete developmental step takes 16 molecular cycles, with 5 molecular cycles used for I/O and diffusion, 10 molecular cycles used for expression (2 molecular cycles are required per entry in the expression table, and 2 additional molecular cycles are required to initialize the process) and the last molecular cycle outputs the functionality of the cell.

Complete development, which requires 16 developmental steps, takes $16 \cdot 16 \cdot 16 = 4096$ clock cycles. In contrast to a software implementation, this time is independent of the phenotype size.

Several control signals (see fig. 8) manage the sequencing of the signalling and expression blocks. Control signals are efficiently implemented by memory molecules configured as rotating memories.

The complete schematic of the POEtic implementation is given in figure 8 and the molecular layout of the implementation is illustrated in figure 9.

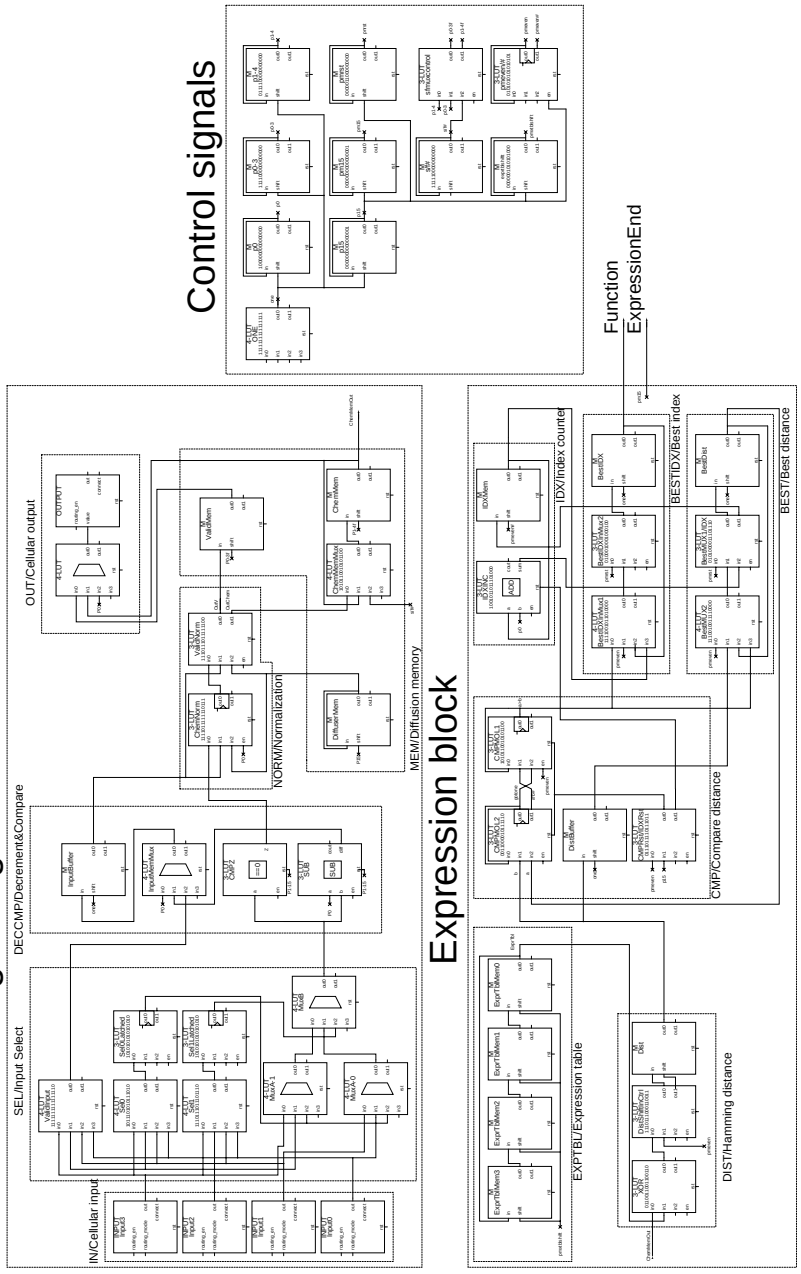

Figure 8: Schematic of the molecular implementation of the morphogenetic element.

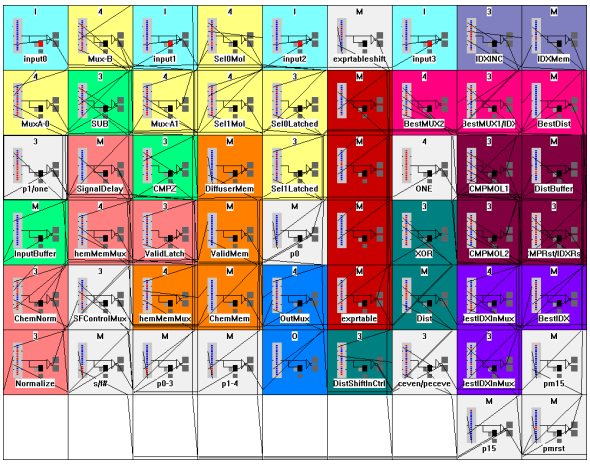

Figure 9: Molecular layout of the morphogenetic element: 56 molecules are required to implement the morphogenetic element.

The morphogenetic element is implemented in 56 molecules. They are distributed as follows: 12 implement control signals (in comparison to the schematic two molecules implement the same control signal in the layout to reduce the routing ressources), 25 implement the signalling block (5 are used as inputs and outputs via the dynamic routing layer) and 19 implement the expression block (4 of them are memories to store the expression table).

A 3 by 3 array of morphogenetic element after interconnection by the dynamic routing mechanism is illustrated in figure 10. Extensive simulations were done to ensure that the behaviour of the hardware implementation is identical to the software implementation.

5 Discussion and conclusion

The morphogenetic system, a developmental system and genetic encoding with indirect genotype to phenotype mapping, has been implemented in hardware on the POEtic circuit.

The low computational complexity of the morphogenetic system translates in compact hardware implementation. A cellular hardware implementation of the morphogenetic system on the POEtic circuit uses 56 molecules per cells to implement the morphogenetic process. As a comparison, POEtic molecules are approximately equivalent to a a Logic Element in Altera families of FPGA. Because of the cellular implementation, complete developmental takes 4096 clock cycles whatever the size of the phenotype is. Assuming a reasonable operating frequency of 10MHz of the chip this means that development of about 2000 individuals/sec is possible. The morphogenetic system is relatively compact and fast and it still has very interesting properties of evolvability and scalability compared to direct genetic encodings in several classes of applications [41, 40].

As a comparison, other developmental systems have been implemented in the POEtic circuit. A multi-cellular evolving and developing circuit with a direct genetic encoding has been implemented using 40 molecules per cell [42]. This

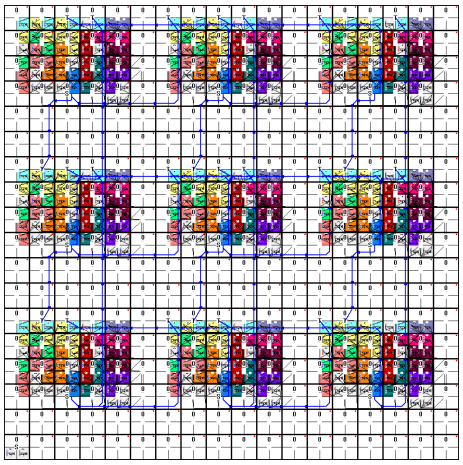

Figure 10: A 3x3 array of morphogenetic elements. Interconnections are done at run-time by the dynamic routing mechanism of the POEtic chip.

system exploits features of the POEtic chip to implement growth and differentiation mechanisms: cells, which are initially undifferentiated and unconnected, connect to each other and differentiate by expressing a corresponding part of the genetic code and thereby taking a specific functionality in the circuit. However, this developmental system does not seek to address the issue of scalability encountered in evolvable hardware, but it demonstrates the dynamic reconfiguration features of the POEtic chip. As a consequence a direct genetic encoding is used, which explains the smaller cell size. Also, hardware friendly genetic regulatory networks (GRNs) have been implemented in the POEtic circuit [21]. A preliminary implementation of a cell requires about 200 molecules. GRNs have the advantage of having intrinsic spatial and temporal dynamics which can be exploited to implement the development and the functionality of cells. For instance oscillatory circuits evolve easily and robustness to faults was evidenced. However this comes at a price: the size of the GRN cell is much larger than that of the morphogenetic element.

Comparison with other intrinsic developmental systems is difficult due to the disparity in hardware platforms. The L-System-based developmental models of Haddow and Tufte is intrinsic and centralized, implemented in a coprocessor dedicated to the development mechanism [50]. The developmental system of Embryonics does provide self-repair and self-replication however a direct genotype to phenotype mapping is employed, and no evolution has yet been used to find working circuits [31]. The cellular-automata-based growth of neural networks of de Garis was used in relatively simple tasks but seems to require a lot of computing power [5]. Interestings results in extrinsinc developmental systems were obtained in terms of fault-tolerance and adaptivity [34] and an intrinsic implementation is considered [27].

The morphogenetic system is an intrinsic, online and cellular developmental system, which is where we believe most of the benefits of developmental systems lie. Intrinsic development means fast genotype to phenotype mapping and close interaction of the developing circuit and its environment. Together with online development it may allow adaptation to the environment or fault-tolerance.

Finally a distributed development is faster, may be more robust than a centralized one and is more scalable. Still, the full potential of online development is not yet achieved as the hardware implementation of the morphogenetic system does not yet address the issue of adaptation or fault-tolerance. This remains the subject of future work, however interesting preliminary results in terms of fault-tolerance were obtained in software simulations [40].

References

[1] P. J. Angeline. Morphogenic evolutionary computations: Introduction, issues and examples. In J. R. McDonnel, R. G. Reynolds, and D. B. Fogel, editors, *The Fourth Annual Conference on Evolutionary Programming*, pages 387–401, Cambridge, MA, 1995. MIT Press.

[2] E. Coen. *The art of genes*. Oxford University Press, New York, 1999.

[3] H. de Garis. *GENETIC PROGRAMMING: GenNets,Artificial Nervous Systems,Artificial Embryos*. PhD thesis, Brussels University, 1992.

[4] H. de Garis. Growing an artificial brain with a million neural net modules inside a trillion cell cellular automaton machine. In *Proceedings of the Fourth International Symposium on Micro Machine and Computer Science*, pages 211–214, 1993.

[5] H. de Garis, L. de Penning, A. Buller, and D. Decesare. Early experiments on the CAM-brain machine (CBM). In A. Stoica et al., editors, *1st NASA/DoD Workshop on Evolvable Hardware*, pages 211–219, Los Alamitos, California, 2001. IEEE Computer Society Press.

[6] D. Floreano and J. Urzelai. Evolution of plastic control networks. *Autonomous Robots*, 11:311–317, 2001.

[7] T. Gordon. Exploring models of development for evolutionary circuit design. In *CEC2003, the Congress on Evolutionary Computation*, pages 2050–2057. IEEE Press, 2003.

[8] T. Gordon and P. Bentley. Towards development in evolvable hardware. In J. Lohn et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 241–250, Los Alamitos, California, 2002. IEEE Computer Society Press.

[9] P. C. Haddow and G. Tufte. An evolvable hardware fpga for adaptive hardware. In *Proceedings of CEC'00*, pages 553–560, 2000.

[10] P. C. Haddow and G. Tufte. Bridging the Genotype-Phenotype Mapping for Digital FPGAs. In D. Keymeulen et al., editors, *3rd NASA/DoD Workshop on Evolvable Hardware*, pages 109–123, Los Alamitos, California, 2001. IEEE Computer Society Press.

[11] P. C. Haddow, G. Tufte, and P. van Remortel. Shrinking the Genotype: L-systems for EHW? In Y. Liu et al., editors, *Proc. of the 4th Int. Conf. on Evolvable Systems (ICES 2001)*, pages 128–139, Berlin, 2001. Springer.

[12] P. C. Haddow and P. van Remortel. From Here to There: Future Robust EHW Technologies for Large Digital Designs. In D. Keymeulen et al., editors, *3rd NASA/DoD Workshop on Evolvable Hardware*, pages 232–239, Los Alamitos, California, 2001. IEEE Computer Society Press.

[13] H. Hemmi, J. Mizoguchi, and K. Shimohara. Evolving large scale digital circuits. In C. Langton and K. Shimohara, editors, *Proceedings of Artificial Life V*, pages 213–218. MIT Press, 1996.

[14] T. Higuchi et al. Evolving hardware with genetic learning: A first step towards building a darwin machine. In J.-A. Meyer, H. Roitblat, and S. Wilson, editors, *Proceedings of the Second International Conference on Simulation of Adaptive Behaviour*, pages 417–424, Cambridge, MA, 1993. MIT Press-Bradford Books.

[15] T. Kalganova. Bidirectional incremental evolution in extrinsic evolvable hardware. In J. Lohn et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 65–74, Los Alamitos, California, 2000. IEEE Computer Society Press.

[16] S. A. Kauffman. Metabolic stability and epigenesis in randomly constructed genetic nets. *Journal of Theoretical Biology*, 22:437–467, 1969.

[17] S. Kazadi, Y. Qi, I. Park, N. Huang, P. Hwu, B. Kwan, W. Lue, and H. Li. Insufficiency of piecewise evolution. In D. Keymeulen et al., editors, *3rd NASA/DoD Workshop on Evolvable Hardware*, pages 223–231, Los Alamitos, California, 2001. IEEE Computer Society Press.

[18] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems, 4(4):461–476*, 1990.

[19] H. Kitano. Challenges of evolvable systems: Analysis and future directions. In T. Higuchi et al., editors, *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, pages 125–135, Berlin, 1996. Springer.

[20] J. Kodjabachian and J. Meyer. Evolution and development of control architectures in animats. *Robotics and Autonomous Systems, 16(2-4)*, 1995.

[21] A. Koopman and D. Roggen. Evolving Genetic Regulatory Networks for Hardware Fault Tolerance. In *Proceedings of Parallel Problem Solving from Nature VIII*, pages 561–570, Berlin, 2004. Springer.

[22] J. R. Koza. *Genetic Programming.* MIT Press, Cambridge, MA, 1992.

[23] J. R. Koza, F. H. Bennet III, D. Andre, M. A. Keane, and F. Dunlap. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128, 1997.

[24] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane. Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming. In T. Higuchi et al., editors, *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, pages 312–326, Berlin, 1996. Springer.

[25] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, , and G. Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[26] A. Lindenmayer. Mathematical models for cellular interactions in development. *Journal of Theoretical Biology*, 18:280–299, 1968.

[27] H. Liu, J. F. Miller, and A. M. Tyrrell. An Intrinsic Robust Transient Fault-Tolerant Developmental Model for Digital Systems. In K. Deb et al., editors, *Proceedings of WORLDS workshop at GECCO 2004*, Berlin, 2004. Springer.

[28] J. D. Lohn and S. P. Colombano. Automated Analog Circuit Synthesis using a Linear Representation. In M. Sipper et al., editors, *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, pages 125–133, Berlin, 1998. Springer.

[29] J. D. Lohn and S. P. Colombano. A circuit representation technique for automated circuit design. *IEEE Transactions on Evolutionary Computation*, 3(3):205–219, 1999.

[30] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand. Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware*, pages 197–220, Berlin, 1996. Springer.

[31] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. *Proceedings of the IEEE*, 88(4):516–541, 2000.

[32] P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand. Embryological development on silicon. In R. Brooks and P. Maes, editors, *Proceedings of the Fourth International Conference on the Synthesis and Simulation of Living Systems*, pages 365–370. MIT Press, 1994.

[33] C. Mattiussi and D. Floreano. Evolution of analog networks using local string alignment on highly reorganizable genomes. In R. Zebulum et al., editors, *2004 NASA/DoD Conference on Evolvable Hardware*, pages 30–37, Los Alamitos, California, 2004. IEEE Computer Society Press.

[34] J. Miller. Evolving developmental programs for adaptation, morphogenesys, and self-repair. *Proceeding of ECAL 2003, 256–265*, 2003.

[35] J. F. Miller and P. Thomson. Cartesian genetic programming. In R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, editors, *Proceedings of EuroGP'2000*, pages 121–132, Berlin, 2000. Springer Verlag.

[36] J. F. Miller and P. Thomson. A Developmental Method for Growing Graphs and Circuits. In A. M. Tyrrell et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 93–104, Berlin, 2003. Springer.

[37] J. Mizoguchi, H. Hemmi, and K. Shimohara. Production genetic algorithms for automated hardware design through an evolutionary process. In *Proceedings of the First IEEE Conference on Evolutionary Computation (ICEC'94)*, pages 661–664. IEEE Press, 1994.

[38] E. Mjolsness, D. H. Sharp, and B. K. Alpert. Scaling, machine learning, and genetic neural nets. *Advances in Applied Mathematics*, 10:137–163, 1989.

[39] J. M. Moreno, Y. Thoma, E. Sanchez, and G. Tempesti. Hardware realization of a bio-inspired poetic tissue. In R. Zebulum et al., editors, *2004 NASA/DoD Conference on Evolvable Hardware*, pages 237–244, Los Alamitos, California, 2004. IEEE Computer Society Press.

[40] D. Roggen and D. Federici. Multi-cellular development: is there scalability and robustness to gain? In *Proceedings of Parallel Problem Solving from Nature VIII*, pages 391–400, Berlin, 2004. Springer.

[41] D. Roggen, D. Floreano, and C. Mattiussi. A Morphogenetic Evolutionary System: Phylogenesis of the POEtic Tissue. In A. M. Tyrrell et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 153–164, Berlin, 2003. Springer.

[42] D. Roggen, Y. Thoma, and E. Sanchez. An Evolving and Developing Cellular Electronic Circuit. In *Proceedings of Artificial Life IX*, pages 33–38, 2004.

[43] E. Sanchez and D. Mange. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. In T. Higuchi et al., editors, *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, pages 35–54, Berlin, 1996. Springer.

[44] M. Sipper, E. Sancher, D. Mange, M. Tomassini, A. Perez-Uribe, and A. Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation*, 1(1), 1997.

[45] K. Stanley and R. Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003.

[46] I. Tagkopoulos, C. Zukowski, G. Cavelier, and D. Anastassiou. A custom FPGA for the simulation of gene regulatory networks. In *Proceedings of the 13th ACM Great Lakes symposium on VLSI*, pages 132–135, New York, NY, USA, 2003. ACM Press.

[47] Y. Thoma. *Description of the Organic Subsystem of the POEtic Tissue.* Swiss Federal Institute of Technology, Logic Systems Laboratory, Lausanne, Switzerland, 2004.

[48] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti. A dynamic routing alogrithm for a bio-inspired reconfigurable circuit. In *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*, pages 681–690, Berlin, 2003. Springer Verlag.

[49] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti. Poetic: An electronic tissue for bio-inspired cellular applications. In *Proc. 5th Int. Workshop on Information Processing in Cells and Tissues (IPCAT 2003)*. To be published, 2003.

[50] G. Tufte and P. C. Haddow. Building knowledge into developmental rules for circuit design. In A. M. Tyrrell et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 69–80, Berlin, 2003. Springer.

[51] G. Tufte and P. C. Haddow. Biologically-Inspired: A Rule-Based Self-Reconfiguration of a Virtex Chip. In M. Bubak, G. D. van Albada, P. M. A. Sloot, et al., editors, *Proc. of the 4th Int. Conf. on Computational Science (ICCS 2004)*, pages 1249–125, Berlin, 2004. Springer.

[52] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London*, 237(641), 1952.

[53] A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. Villa. POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware. In A. M. Tyrrell et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 129–140, Berlin, 2003. Springer.

[54] J. Vaario, S. Ohsuga, and K. Hori. Connectionist modeling using lindenmayer systems. In *Information Modeling and Knowledge Bases: Foundations, Theory, and Applications*, pages 496–510, 1991.

[55] V. K. Vassilev and J. F. Miller. Scalability problems of digital circuit evolution: Evolvability and efficient designs. In J. Lohn et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 55–64, Los Alamitos, California, 2000. IEEE Computer Society Press.

[56] L. Wolpert. *Principles of Development*. Oxford University Press, Oxford, 1998.

[57] X. Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:203–222, 1993.