Evaluation of POEtic as a Purpose-Built Evolvable and Growing Hardware Platform

Crispin Cooper, Andy Tyrrell*, *SeniorMIEEE, FIEE*

Abstract

Currently there are few configurable logic chips suitable for evolvable hardware experimentation, and even fewer which can support the recent research trend of biological development. This paper concerns the new POEtic chip, which is designed to support both intrinsic evolution and growth on one platform. The chip is evaluated from the point of view of research application development: the implementation process for an evolving digital waveguide mesh is presented in detail, and a 'growing' application is also discussed. The chip is found to provide a very flexible platform for future experiments. The ontogenetic features exhibit extensive research potential.

Index Terms

Evolvable hardware, intrinsic evolution, phylogenesis, ontogenesis, growth, signal processing, programmable logic, bit serial, optimisation, development tools.

The authors are with the Department of Electronics, University of York, UK. e-mail: amt@ohm.york.ac.uk (corresponding author), crispin@cantab.net. phone: +44 1904 432361, fax: +44 1904 432335.

Evaluation of POEtic as a Purpose-Built Evolvable and Growing Hardware Platform

I. Introduction

IT is probably annoying to some researchers that Xilinx no longer manufacture the XC6200. Starting with Thompson's work on intrinsic evolution in 1996 [1] the field of Evolvable Hardware benefited from a chip that firstly had a transparent, open architecture and secondly had a routing plane that, implemented with multiplexers, could not be easily broken by a random configuration.

On the other hand, research has moved on and perhaps the XC6200 would be of limited use today. A problem with encoding a circuit directly as a configuration bitstream, as Thompson did, is that the genetic algorithm search space increases exponentially with the size of the circuit. Thus, the complexity of problems which can be solved by evolving an XC6200 configuration is limited. For this reason, evolvable hardware research has now segmented into two approaches: one is to work with a greater level of abstraction, evolving parameters such as filter coefficients rather than configuration bits, while the other is to study the biological concept of development (also known as ontogenesis [2]). The latter is of course the solution nature uses to evolve a complex system: a small cell is evolved, but this then grows to create a larger organism [3]. The POEtic chip [4] is a new evolvable hardware platform which has been developed to support both types of research.

The contribution of this paper is to discuss the POEtic chip from a user's point of view, the design flow of applications developed on the platform and the advantages and disadvantages of working in this way. Section II briefly describes the design of the chip and the tools available to developers. In Section III a novel application is presented in which abstract parameters concerning the shape of the circuit, rather than logic configuration bits, are evolved to create a waveguide model of the human vocal tract. In Section IV the design of an ontogenetic application is discussed, and Section V concludes.

II. The POEtic Chip

The POEtic chip was designed in 2003-4 [4], based on a bio-inspired concept originally described in [5]. This section summarizes the relevant features of the chip as seen from the perspective of the application developer, instead of the hardware designer.

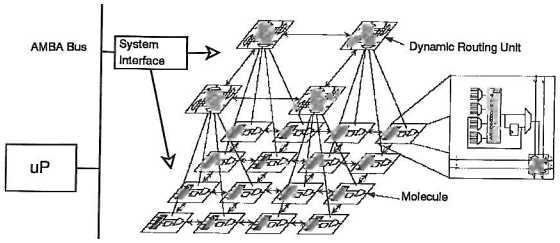

Fig. 1. Structure of a single POEtic chip as described in [4].

A. Overview

Internally, the POEtic chip consists of a microprocessor connected to an area of FPGA-like configurable logic called *POEtic tissue* (Figure 1). All tissue configuration and data bits are directly accessible to the microprocessor via memory mapping. The processor can also control the tissue clock, with single-stepping capability. Thus, it is possible to observe in detail the operation of any circuit running on the tissue. Also, the processor can be used to run a genetic algorithm and intrinsically evolve circuits. This is facilitated by the inclusion of a hardware random number generator.

The tissue is divided into *molecules*, each of which can operate in one of several modes. Standard FPGA functionality is achieved using *logic* and *memory* modes, where the molecule functions as (for example) a 4-bit LUT and flipflop, or a 16-bit shift register. However, a number of special modes also exist to assist with biological-inspired experimentation.

144 molecules are present on each chip. This number seems small in comparison to commercial FPGAs, but POEtic chips can be seamlessly chained together. If several chips are connected then *all* the tissue on *all* the chips is automatically mapped into the memory of *all* the microprocessors, allowing for easy extension of a large design.

B. Routing System

There are two different routing systems on the POEtic chip, *(inter)molecular* (short-range) and *(inter)cellular* (long-range). This terminology derives from considering ontogenetic applications which consist of a set of *cells*; these often require short-range routing inside a cell and long-range routing between cells. However, there is no requirement that an application be divided in this way - "cellularity" is an abstraction which the developer is free to use, but is not necessary for either type of routing. Still, the routing systems behave in dramatically different ways.

Molecular Routing: is implemented with a switchbox in each molecule (Fig. 2). Any of eight local inputs can be routed to any of eight local outputs, or to the molecule's internal logic - the output

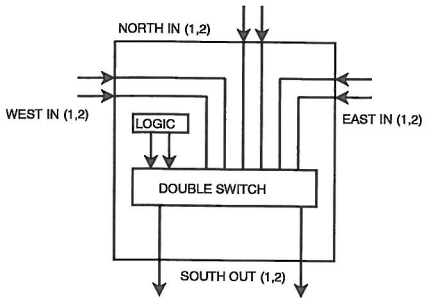

Fig. 2. Example *molecular* routing switchbox for the two south outputs of a single molecule. A number of similar switches exist in each molecule, to handle the north, west and east outputs as well as the four inputs to the logic. Thus, up to 8 signals can be passed through the molecule in addition to the molecule performing its own function.

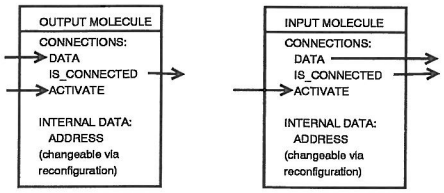

Fig. 3. Synopsis of the I/O molecules used for *cellular* routing, showing their connections to the *molecular* layer.

of which is also available for local routing. Molecular routing cannot cross the boundaries between different chips.

Cellular Routing: is implemented with the special *input* and *output* molecule types (Fig. 3). These molecules can be assigned arbitrary 16-bit addresses which are connected, on request and at runtime, to other I/O molecules (Fig. 4). This is achieved with a hardware implementation [6] of Dijkstra's routing algorithm [7]. Output molecules are connected to the nearest free input with a matching address, while input molecules are connected to the nearest matching output. This is also known as *dynamic routing*.[1] Cellular routing connections can cross the gap from one chip to another.

Note that neither type of routing, in any configuration, allows two outputs to be connected together. This ensures that random circuits created by a GA will not damage the chip.

C. Reconfiguration Capabilities

In order to support circuits modelled on biological development, where *cells* can change their functions at runtime, the tissue is extensively reconfigurable:

1) the microprocessor can change molecule configurations at any time, and

[1] A *static routing* mode is also available in which cellular connections are treated in the same way as molecular connections.

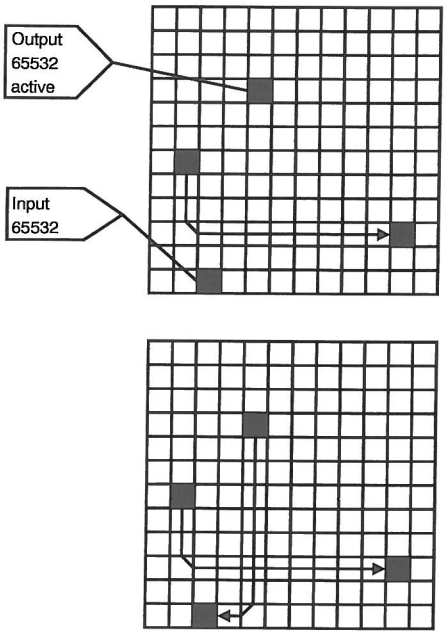

Fig. 4. Example of the *cellular* routing process. When an output molecule declares itself 'active', it is automatically connected to the nearest input molecule with the same address.

2) molecules can reconfigure one another.

This allows for distributed models of development and fault tolerance. Each molecule is initialised with a *configuration input direction*, which specifies a neighbour from which it will accept serial reconfiguration data. Large configuration chains can thus be set up across the tissue. A *partial reconfiguration* option also allows parts of each molecule to be designated non-reconfigurable. Thus, *logic, data, clock synchronisation, reset actions, molecule modes* and *routing* can all be reconfigured independently.

D. Reconfiguration of Routing

Reconfiguring the *molecular routing* of one molecule is likely to break the circuit, as the reconfiguration process has no respect for existing data sources and destinations. However, if the functionality of an entire block of circuitry is to be changed, the block reconfiguration data can include the correct molecular routing.

Reconfiguring the *cellular routing* is a more robust process. The routing address is simply rewritten, and a *re-route* process triggered, thus connecting the reconfigured molecule to the appropriate input or output automatically - regardless of physical location.

E. Development Tools

Two design tools have been developed which assist with the design of circuits on POEtic tissue.

POEticMol: allows direct editing of all the configuration bits of molecules on the tissue. It allows single-stepping and running of a tissue simulation and includes a tool to automatically create molecular routes, without configuring the switchboxes by hand.

SchemEd: is a schematic editor which compiles designs to files readable by POEticMol.

A microcode assembler and C compiler have also been produced for the microprocessor. The applications described in this paper were created with POEticMol.

III. Evolution of Parameters: the Digital Waveguide Mesh Application

This section discusses the implementation of a parameter-evolving application on POEtic hardware. The audio and GA aspects of this work are discussed in [8] and [9], while this paper examines the hardware as a case study of the POEtic design process. It is also a novel type of hardware evolution in which the circuit shape is changed by the GA.

A. Background

Although electronic speech and singing synthesis has existed since the 1960s, research still continues into creating realistic, humanlike sounds. It has been shown in [10] that digital waveguide meshes (DWMs) are a promising area of research for tackling this problem. DWMs allow the creation of acoustic objects, such as a throat, simply by specifying their physical properties i.e. shape and boundary characteristics. A simulated throat can then, for example, be excited at the input (larynx) end with white noise and a whispered sound heard at the output (mouth). This process is effective in one and two dimensions even though the space being modelled is three-dimensional.

However, the simulation needs real data on the shape of the human vocal tract while in use, and this is not easily available. fMRI techniques have been used [11] but these do not provide good temporal resolution and are expensive to undertake for every speaker and singer we wish to recreate electronically. X-rays have also been used, but there are issues with safe radiation dosage levels [12].

As an alternative approach, we evolve the shape of 2-d electronic vocal tracts until the output sound matches that of a given singer. As digital waveguide meshes are computationally expensive, and genetic algorithms compound this problem, it was decided that a hardware implementation of this on the POEtic platform would be useful for voice research as well as a good test of the capabilities of POEtic.

B. Designing for the POEtic tissue

Our simulation of the vocal tract consists of a number of *elements* each of which perform a simple computation [13] on 16-bit two's complement integers. Scattering Elements have four inputs and one output, and at each timestep of length Δt perform the computation described in Equation 1.

$$p_{out}(t) = \frac{1}{2}\sum_{i=0}^{i=3} p_{in}^{i}(t - \Delta t) - p_{out}(t - 2\Delta t) \quad (1)$$

Reflective Elements have one input, one output and a coefficient of reflection λ. At each timestep they perform the computation described in Equation 2.

$$p_{out}(t) = (1 + \lambda)p_{in}(t - \Delta t) - \lambda p_{out}(t - 2\Delta t) \quad (2)$$

It was decided to use cellular routing for inter-element connections (which can thus bridge the boundaries between chips) and molecular routing for local, intra-element connections. This also means that the mesh elements can be considered as "cells" in a cellular application. In this way, a simulated vocal tract can be configured by creating cells of the appropriate type in a position on the tissue which maps directly to the 2-d structure they represent. The individual "cell types" were designed in POEticMol.

C. Optimisation

Careful hand-optimisation of the designs is needed in order for the finished circuit to fit onto a reasonable number of POEtic chips. This is described in detail as for large applications, it is a key part of the POEtic design process. Optimisation efforts can be categorised as follows:

1) Use of bit-serial representation
2) Reduction of I/O overheads
3) Optimisation of arithmetic
4) Optimisation of logic
5) Use of genetic optimisation

1) Bit Serial Arithmetic: As the *logic* molecule types support at most 4 inputs and 2 outputs, by far the most efficient means of transferring and storing data is the bit serial format. A standard parallel arithmetic system would use many more molecules (see Table I). It would also run at 16 times the speed - however, this extra speed is not needed. Even a 1MHz clock would support real-time creation of 48 KHz audio data, and the POEtic clock can run faster than this. Serial implementation is eased by the availability of the *memory* molecule type which provides a 16 bit shift register.

TABLE I

ESTIMATED NUMBER OF POETIC MOLECULES PER DIGITAL WAVEGUIDE MESH ELEMENT.

Elements/Cell	Parallel arithmetic	Serial arithmetic
1	88	18
4	64	15
9	66	n/c
16	62	n/c

TABLE II

WORD ALIGNMENTS USED IN FIGS. 5 AND 6.

Offset	Meaning
0	Word aligned with cell input
3	Word aligned with arithmetic output
16	Word aligned with cell output (Fig. 5)
19	Word aligned with cell output (Fig. 6)

2) I/O overheads: The option of combining several mesh elements into one "cell" was considered. This cuts down on inter-cell connections, which need two extra molecules per signal (one input and one output). Also, the bit serial arithmetic in each element requires five molecules of control logic, and this logic can be shared between elements if they are combined into one cell. However, such attempts at efficiency greatly increase the complexity of the genetic algorithm, as we must now define a large number of different cells (representing various combinations of different types of elements), instead of just two cell types (one for each element). Thus the final design uses only one element per mesh cell.

3) Optimisation of Arithmetic: Note in Equation 2 that we must multiply arbitrary numbers by λ and $1+\lambda$, and this potentially requires a large amount of logic. However the specification is flexible: values of λ close to ± 0.9 are required,[2] and we implement $\lambda = \pm 0.875$. This allows calculation of Equation 2 with only subtraction and bit shifting, as $0.875x = x - x/8$ and $x/8$ is simply x shifted by 3 places.

[2]The outer end of the vocal tract has a negative coefficient of reflection, while the sides have a positive one.

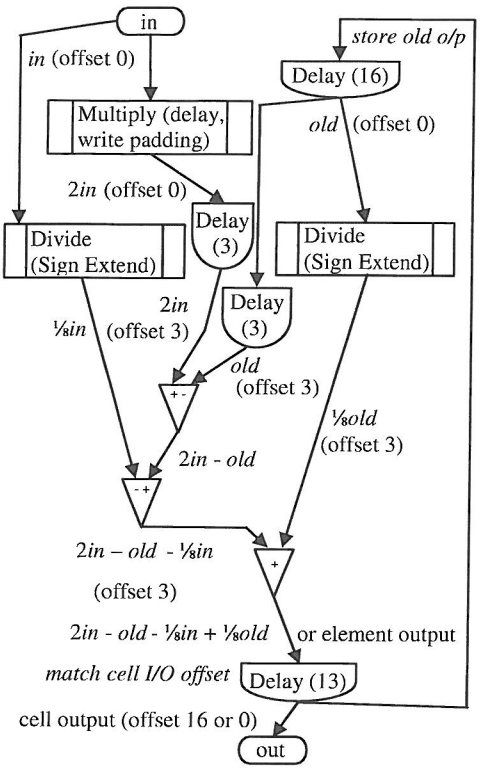

Fig. 5. Un-optimised Logic for a Reflective Cell. As the data is in bit-serial form, signals are annotated with a mathematical value and an offset for the correct reading of that value. See Table II for a summary of the offsets used.

4) Optimisation of Logic: The logic implementation is bit-serial, as described in [14]. However, due to the limited quantity of hardware available, it was crucial to make designs as small as possible, so some standard logic optimisations were performed by hand.

- Where possible, delays in the signal path have been combined as mentioned in [15]. For example, in Figure 5, the calculation of $2in - out - 1/8in$ requires the signals representing $2in$ and out to be delayed by 3 clock cycles each in order to match the word alignment of $1/8in$. This would require 6 POEtic molecules. But by calculating $2in - out$ before delays are applied (Figure 6), the single signal representing the result can be delayed with only 3 molecules.
- Multiplication by 2 (left shifting) has been made more efficient by specifying that there will be no overflow protection. (Figure 7).

Some nonstandard optimisation techniques are used as well. One of these is, as far as the authors know, novel and so is described here for completeness. It concerns the multiplication by -0.875 and +1.875, which is achieved with a special case of the serial-parallel multiplier [16] using only shift operations. It is found that resource usage can be reduced by the introduction of an extra bit of

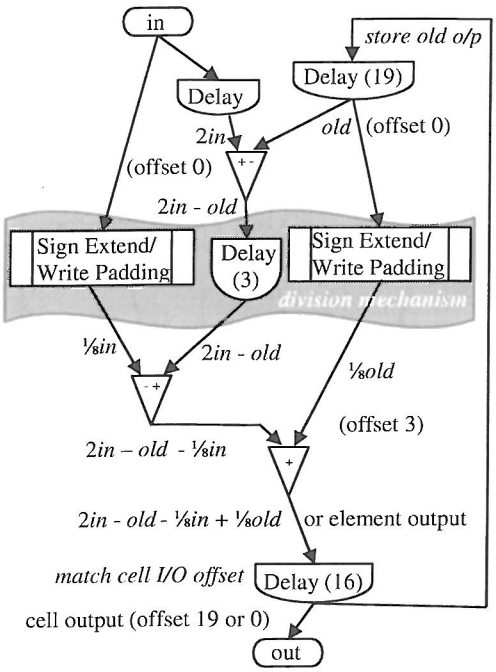

Fig. 6. Optimised Logic for a Reflective Cell. As the data is in bit-serial form, signals are annotated with a mathematical value and an offset for the correct reading of that value. See Table II for a summary of the offsets used. Logic optimisation has been achieved by (i) combining delay paths, (ii) removing overflow protection, thus replacing multiplication with a delay, (iii) introducing padding between words, which allows division to be implemented with a single flipflop and (iv) increasing the word length in order to replace the 13-step delay, which is costly to implement on POEtic, with a 16-step delay.

padding between words - contrary to intuition, which would suggest that the extra storage flipflops required would increase resource usage. This situation arises as follows:

1) The circuit contains logic for division or 'right shifting', which works by delaying all signals *not* being divided, while extending the sign of the numerator signal. As the signals have now been delayed by different amounts, we risk corrupting the previous output word when we subsequently add the result of the division to the output (the output's previous MSB overlaps with our former LSB). This can be prevented by disabling the adder on cycles where the overlap occurs, but only at the expense of extra hardware.
2) Instead, we circumvent the problem by introducing padding between words, thus ensuring the previous word's MSB remains uncorrupted. However we must guarantee that this padding will be set to zero (Fig. 8) as the left-shift operation relies on it (Fig. 9). We make no resource gain in this step as the problem of disabling an adder has been replaced by the problem of setting a padding bit to zero.

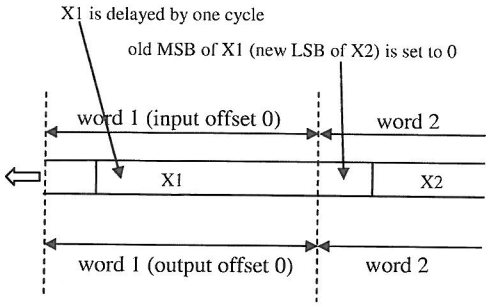

Fig. 7. Multiplication (left shifting) of signal X (words X1, X2) by 2. Input and output alignment is the same. Note that if we specify no overflow protection, we can assume that the old MSB of X1 is zero, so do not need hardware to set this.

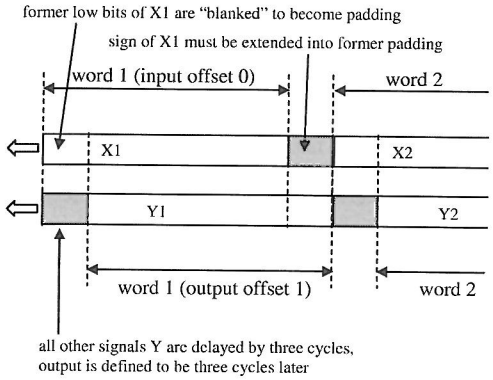

Fig. 8. Division (right shifting) of signal X (words X1, X2) by 2. Input and output alignments differ.

3) However, it is possible to implement both *sign extension* and *setting to zero* with the same flipflop: in one case, the flipflop stores the value of the extended sign and in the other case, it stores a zero. In both cases, the stored value is written to the signal: the flipflop must simply be reset every word cycle, in order to replace the 'sign' value with zero. The arithmetic carry flipflops are already reset in this manner so the existing reset control signal can be used.

So we save on one POEtic 'molecule' per mesh element - a reduction of approximately 6% - at the cost of a 6% gain in calculation time (from processing the padding), which we can afford. This is potentially applicable to any bit-serial application on programmable hardware in which

- both left and right shifts are used,
- a word-cycle reset signal already exists, and
- it is preferable to minimise flipflop usage at the expense of increased calculation time.

In fact, the final design (Fig. 6) uses *three* bits of padding between words, and this still represents a resource saving over the no-padding implementation. This is because of the efficiency of implementing

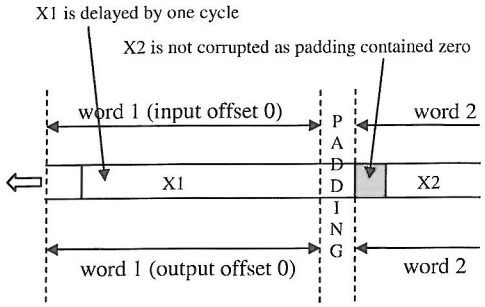

Fig. 9. Multiplication (left shifting) of signal X (words X1, X2) by 2, with padding. Input and output alignment is the same.

delays of certain lengths on the POEtic hardware (see Table III). In the case of the reflective cells, $w-3$ bits of serial storage are required to hold the cell output, for word length w. This is because the remaining 3 bits of each word are held in flipflops on the calculation chain. Thus, with $w=19$ we can store the result in a 16-bit shift register, requiring only one molecule, whereas $w=17$ would require a 14-bit shift register, using six molecules.

5) Genetic Optimisation: Finally, it is possible to reduce the size of the overall design by restricting the number of reflective mesh elements (which are costlier to implement than scattering elements). While a 'unit' of mesh boundary should reflect on up to three edges (requiring three elements), it is possible to reflect on only two out of three and still obtain reasonable audio results. Priority is given to edges that reflect the sound back towards its source, and then to those that create tangental standing waves. As is usual with GAs, evolution adapts to find a good solution regardless of underlying limitations. It is satisfying to note that on a biologically inspired platform, the best means of optimisation may also be biologically inspired.

D. Conclusions on Parameter Evolution

A system for evolving simulated vocal tracts on POEtic hardware has been developed. The evolutionary approach to this problem currently produces acceptable audio output (Table IV, Fig. 10, [8], [17]). The logic and routing layers of a sample mesh are shown in Figure 11.

However, the limited hardware resources available, even on a set of $\approx$ 20 POEtic chips, make the platform non-ideal for research of problems such as this, which require a moderate amount of hardware (the evolving mesh performs several hundred operations comparable to an integer add, every 19 clock cycles). While implementation may be possible, much time-consuming custom optimisation is necessary to ensure the circuit can be made small enough to fit on the system.

TABLE III

NUMBER OF POETIC MOLECULES NEEDED TO IMPLEMENT A SHIFT REGISTER OF VARIOUS LENGTHS.

Length	Molecules
1	1
2	2
3	3
:	:
7	7
8	1
9	1
10	2
:	:
15	7
16	1
17	1
18	2
19	3

TABLE IV

GA PARAMETERS.

Generations	50
Population	50
Proportion of search space explored	0.2%
Sampling	Universal Stochastic
Selection	Rank
Mutation	Governed by '1/5' rule
Crossover	0.2

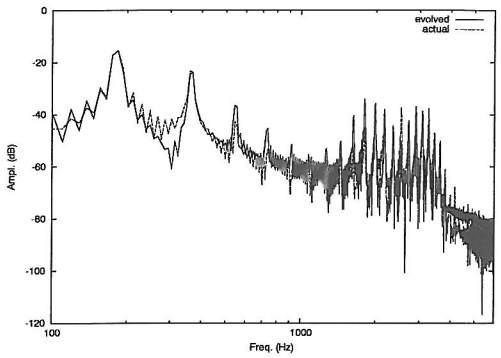

Fig. 10. Output of a mesh evolved to produce the 'ii' vowel, compared to the target 'ii' sound recorded in the studio. The mesh was excited by a real electrolaryngograph signal, which has acoustic properties similar to the excitation produced by the vocal folds.

The optimisation process has been described in detail in order to illustrate the difficulty of automating this process. The authors know of no hardware compiler which would, for example, explore the option of increasing word length and introducing padding between words in order to reduce resource usage. This is perhaps a worthwhile area of research in itself, if such drastic efficiency measures are required for any bit-serial programmable logic device.

While the application discussed here has pushed the POEtic architecture to the upper limits of resource usage, there are plenty of similar evolvable hardware applications which would not do so. For example, digital filter coefficients (implemented as register values) have been evolved on a standard FPGA [18]. POEtic offers the designer an advantage over this approach, as wider features of a circuit (such as the shape), as well as individual coefficients, can be evolved. For applications well within the resource limits, POEtic offers quick prototyping, easy debugging and above all, fast online evolution.

IV. Biological Growth: Design of an Ontogenetic Application

This section briefly describes the development of a 'growing' circuit on POEtic tissue. A growing *and* evolving circuit, consisting of cells which select their functions at runtime such that the circuit performs appropriate operations, has already been developed in [19]. In contrast, the application presented here does not use growth and differentiation in a biological sense but instead uses ontogenetic features for *resource allocation*. The task achieved is synthesis of the sound of a plucked string: a waveguide string [20] of a certain length is needed, and is configured to 'grow' onto the POEtic tissue until the required length is reached. The intention is to test and demonstrate the support for this provided by the POEtic architecture.

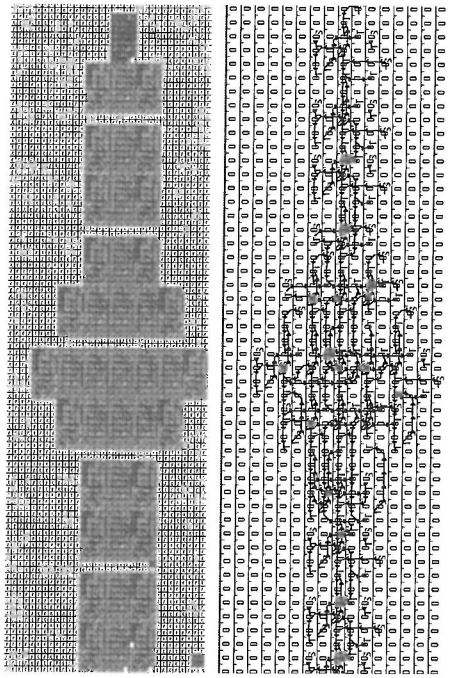

Fig. 11. Logic and routing layers of a mesh evolved to produce the 'ii' vowel. Little detail can be discerned at this level: the figure is included primarily to illustrate the POEticMol development tool. The logic layer (left) contains 42 cells (each 4x8 molecules in size); the gaps between areas of logic are included so that no cell is divided by a chip boundary. The structure of the routing plane, which has emerged at runtime from the addresses set in the cellular routing molecules, is also visible (right).

Of course, actual online growth of silicon transistors is not possible with current technology. All the models of growth presented here concern *information growth*, in which spare, inactive hardware exists which active hardware can reconfigure and use to 'grow' its function.

A. Generic Design for Growth

The fundamental mechanisms provided by POEtic to support all models of ontogenesis are

- self-reconfiguration of the POEtic tissue, and
- the ability to request a connection to the nearest suitable, free input.

Figure 12 illustrates a sample growth process using these features. It is important to note that in this case, all cells have the same initial address on the dynamic routing plane. The growth step is always achieved by requesting a connection to this address - which will connect the existing circuit to the nearest *unconnected* (i.e. inactive) cell. The new cell may then be reconfigured with a different functionality and/or address.

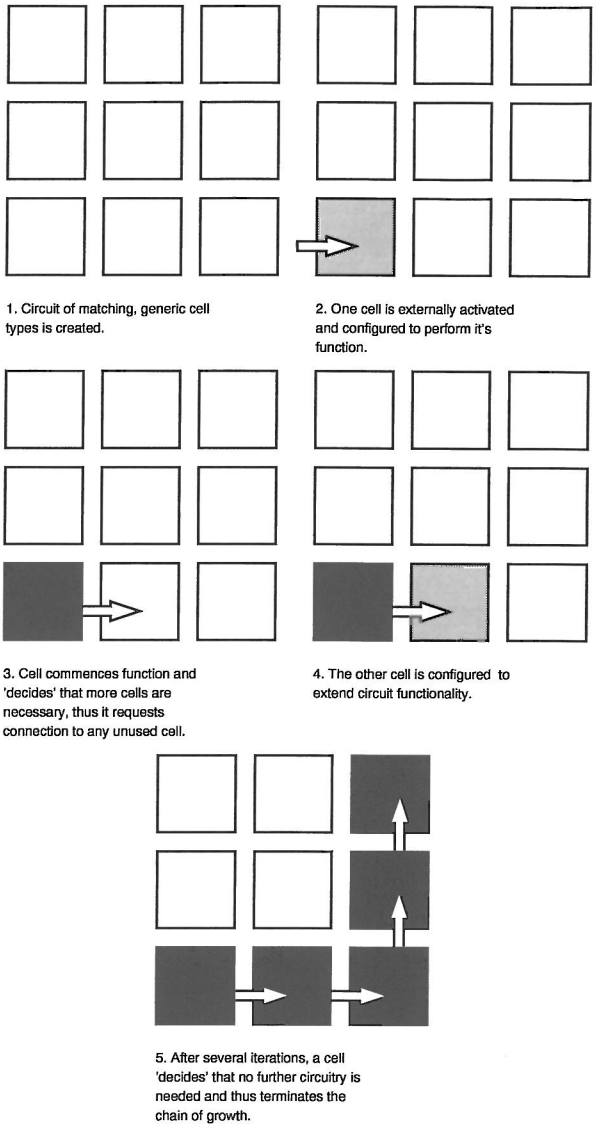

Fig. 12. Illustration of the generic hardware growth process. Steps 2 and 4 are enabled by the reconfiguration capabilities of the tissue, while step 3 is enabled by the dynamic routing system.

Several variations on this example are possible - for example, the cells could connect to more cells *before* deciding on their own function, with each cell thus going through three stages of life: connection, differentiation and function. This is the approach taken in [19]. Also, it is possible to evolve the cells rather than design them.

However, it is not compulsory to design in this manner. We choose instead to simply make use of a key feature of dynamic routing: that the location of a cell performing any particular function is irrelevant, as it can be automatically connected to other cells as appropriate.

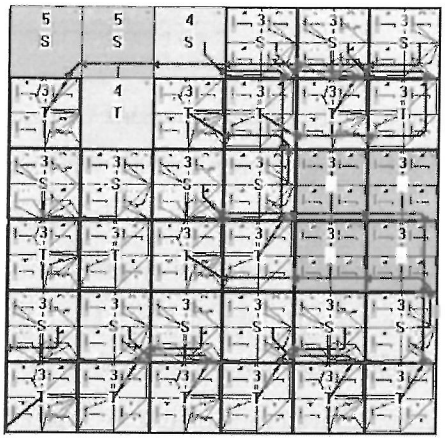

Fig. 13. Screenshot of the growing waveguide string as seen in POEticMol. The string starts on the bottom left. Thick arrows represent cellular connections. Growth takes place by forming forward connections, and cells form backwards connections once they have been activated and the address of their predecessor downloaded. The grey region on the right is intended to simulate a broken cell, which the growth process does not use (connections pass over it but do not end there). The light region at the top left represents the routing process searching for the next active cell.

B. The Waveguide String Application

In contrast the processes described in Section IV-A, the waveguide string application includes no cellular differentiation. Cells simply activate one another in turn, each passing an integer i to the next cell activated, which represents

1) the number of cells still needed, and
2) the address of the current cell.

Each activated cell calculates the value of $i - 1$ to use as its own address. It reconfigures its routing molecules to form backwards connections to the cell which activated it, and if $i > 0$, requests an ongoing connection to continue the chain of cells - down which it sends the new value of i. If $i = 0$ the chain is terminated and circuit function can begin.

This is by no means the simplest way of simulating a plucked string! However, the approach generalises to other problems of resource allocation. Also, the resource allocation is *distributed*, and this is a useful feature if cells are built to include self-test: we can then prevent faulty cells from activating. This greatly reduces the number of potential points of failure in the circuit (though it should be noted that the current implementation of the POEtic tissue contains a number of possible single-points-of-failure, such as a global clock). A full study of fault-tolerant mechanisms on the POEtic tissue has been conducted by Barker [21].

C. Conclusions on Ontogenesis

The application has been shown to be feasible (a snapshot of string growth in progress is given in Figure 13). One point to note is debugging circuits that grow in this way is a difficult task: perhaps generic tools to support analysis and debugging of this kind of application would be helpful.

The growth process is extremely rapid. It would be possible, for example, to create a synthesiser which individually 'grows' the functionality required to play every requested note, in real time. However, there is a difficulty with this approach: resources must be de-allocated as well as allocated, and the methods for doing this are not so well supported. Deallocation requires the breaking of routing connections, and currently, dynamic connections can only be broken by a global re-route. This would reconnect *all* input and output molecules with matching addresses. The process is not only inefficient, but also breaks certain types of circuit.

In particular, the growth mechanisms mentioned here rely on non-unique I/O addresses, which are connected on a nearest-free-neighbour basis. If the state of any cells change, and then a global re-route is triggered, we cannot guarantee that the same nearest-free-neighbours will still exist. Thus the re-routed circuit may undergo unintended functional change - effectively destroying the application.

It is, nevertheless, possible to design applications which are robust to the re-routing process (at the price of increased resource usage). But the authors suggest that it would be useful for future hardware systems supporting 'cell growth' to consider also supporting its biological counterpart, 'cell death' or the breaking of dynamic connections.

V. Conclusions

POEtic has been found to be a useful tool for evolution of parameters in hardware, allowing not only the changing of register values by a GA but also wider configuration aspects of the system such as the shape or connectivity of a circuit. Although chips can be seamlessly chained together, the hardware resources available on POEtic are limited - as with any hardware device. Optimising resource-hungry circuits to fit onto the hardware is difficult, indeed it is not clear how to design automated tools for this task. However, many applications do not require vast amounts of resources and will benefit from the rapid online evolution and debugging offered by the POEtic platform.

Also, the features implemented to support growth (dynamic routing and reconfigurability) are found to function well. Such systems are in their infancy and the platform represents a considerable advance. However, it is found that a process analogous to biological 'cell death' - i.e. the ability to break as well as form dynamic connections - may be of advantage on next-generation systems of this type.

Currently, there is no standard development paradigm for ontogenetic circuits. It is not clear how concepts such as dynamic routing and self-reconfiguration can be supported inside the traditional

development framework of schematic editors and hardware description languages. The POEtic development tools allow use of these features only by direct manipulation at the hardware configuration ('molecular') level: the schematic editor developed on the project does not support use of reconfiguration and dynamic routing features, but is only intended to assist with the design of traditional logic components. Incorporation of these novel features into the traditional hardware development environment may be an important area of future research.

A very interesting potential application of the POEtic platform is in unconstrained evolution. In 1996, Thompson [1] evolved remarkably efficient hardware designs which would likely never have been conceived by a human engineer. However, such circuits were limited in complexity by the mapping from genotype (circuit representation) to phenotype (circuit) - as the configuration bits are stored directly in the genome, the genetic search space increases exponentially with the size of the circuit. POEtic's ontogenetic features, combined with evolution, may be able to overcome this limitation, either through hand-designed growth mechanisms inspired by nature or by allowing unconstrained evolution to exploit the routing and reconfiguration features in its own way. Such work may push the complexity of automatically designed electronics to the next level.

ACKNOWLEDGMENT

This project is funded by the Future and Emerging Technologies programme (IST-FET) for the European Community, under grant IST-2000-28027 (POETIC). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

REFERENCES

[1] Thompson, A. "Silicon Evolution" *Genetic Programming 1996: Proc. 1st Annual Conf (GP96)* MIT Press, Cambridge, Massachusetts, USA, 1996, 444-452.

[2] Sipper, M., Sanchez, E., Mange, D. et al "A Phylogenetic, Ontogenetic and Epigenetic View of Bio-Inspired Hardware Systems" *IEEE Transactions on Evolutionary Computation*, 1:1, April 1997, 83-97.

[3] Wolpert, L. *Principles of Development* Oxford University Press, 1998, ISBN 0-19-850263-X.

[4] Thoma, Y., Tempesti, G., Sanchez, E., Moreno Arostegui, J.-M. "POEtic: An Electronic Tissue for Bio-Inspired Cellular Applications" *BioSystems* 74: 1-3, 2004, 191-200.

[5] Tempesti, G., Roggen, D., Sanchez, E., Thoma, Y., Canham, R., Tyrrell, A., Moreno, J.-M. "A POEtic Architecture for Bio-Inspired Systems" *Proc. 8th Int. Conf. of Artificial Life VIII*, MIT Press, Cambridge, Massachusetts, 2002, 111-115.

[6] Thoma, Y., Sanchez, E., Moreno Arostegui, J.-M., Tempesti, G. "A Dynamic Routing Algorithm for a Bio-Inspired Reconfigurable Circuit" *Proc. of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)* Springer Verlag, 2003, 681-690

[7] Dijkstra, E.W. "A Note on Two Problems in Connexion with Graphs" *Numerische Mathematik* 1, 269-271, 1959.

[8] Cooper, C., Murphy, D., Howard, D., Tyrrell, A. "Singing Synthesis with an Evolved Waveguide Mesh Model" Awaiting review for *IEEE Trans. Speech and Audio Processing*, 2005.

[9] Cooper, C., Howard, D., Tyrrell, A.: "Using GAs to Create a Waveguide Model of the Oral Vocal Tract," *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004* Coimbra, 280-288, 2004.

[10] Mullen, J., Howard, D.M. and Murphy, D.T., "Digital waveguide mesh modelling of the vocal tract acoustics," *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, 119-122, 2003.

[11] Story, B.H., Titze, I.R., and Hoffman, E.A., "Vocal tract area functions from magnetic resonance imaging," *Journal of the Acoustical Society of America*, 104, (1), 471-487, 1996.

[12] Story, B.H., "Using imaging and modelling techniques to understand the relation between vocal tract shape to acoustic characteristics," *Proceedings of the Stockholm Music Acoustics Conference*, SMAC-03, 435-438, 2003.

[13] Van Duyne, S., Smith, J.O., "The 2-d digital waveguide mesh," *Proceedings of IEEE WASPAA*, NY, USA, September 1993.

[14] Denyer, P. and Renshaw, D: *V.L.S.I. signal processing: a bit-serial approach,* Addison-Wesley, 1985, ISBN 0201144042

[15] Isshiki, T. and Dai, W.: "High-Level Bit-Serial Datapath Synthesis for Multi-FPGA Systems," *ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, Monterey, CA, 1996, 167-173.

[16] Andraka, R. "FIR filter fits in an FPGA using a bit serial approach," *Proceedings of the Third Annual PLD Design Conference and Exhibit*, March 1993. Available online: `http://www.andraka.com/files/fir.pdf`

[17] Availible online: `http://www.bioinspired.com/users/cc26/poeticaudio/`

[18] Zhang, Y., Smith, S., Tyrrell, A. "Intrinsic Evolvable Hardware in Digital Filter Design" *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004* Coimbra, 2004.

[19] Roggen, D., Thoma, Y., Sanchez, E. "An Evolving and Developing Cellular Electronic Circuit" *Proc. Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)* MIT Press, Cambridge, Massachusetts, USA, 2004, 33-38.

[20] Smith, J. "Physical Modeling using Digital Waveguides" *Computer Music Journal*, 16, 3, 74-91, 1992.

[21] Barker, W., Tyrrell, A. "Fault Tolerance using Dynamic Reconfiguration on the POEtic Tissue" to be submitted to *IEE Proc. Computing and Digital Techniques*, 2005.