Using Feedback to Regulate Gene Expression in a Developmental Control Architecture

Kester Clegg
Dept. of Computer Science
University of York
York, UK. YO10 5DD
kester@cs.york.ac.uk

Susan Stepney
Dept. of Computer Science
University of York
York, UK. YO10 5DD
susan@cs.york.ac.uk

Tim Clarke
Dept. of Electronics
University of York
York, UK. YO10 5DD
tim@ohm.york.ac.uk

ABSTRACT

We present what we believe is the first attempt to physically reconstruct the exploratory mechanism of genetic regulatory networks. Feedback plays a crucial role during developmental processes and its mechanisms have recently become much clearer due to evidence from evolutionary developmental biology. We believe that without similar mechanisms of interaction and feedback, digital genomes cannot guide themselves across functional search spaces in a way that fully exploits a domain's resources, particularly in the complex search domains of real-world physics. Our architecture is designed to let evolution utilise feedback as part of its mechanism of exploration.

Categories and Subject Descriptors

B.7 [**Hardware**]: Integrated Circuits

General Terms

Design

Keywords

Evolvable hardware, adaptation/self-adaptation, signal processing, gene regulatory networks, developmental processes

1. INTRODUCTION

A research agenda calling for evolutionary computation to abandon its "restricted and dated understanding of natural evolution" has recently appeared [1]. That article asks the field to challenge its long held assumption that there is a "one-way flow of information, from DNA to proteins" that forms the basis of solution discovery by evolutionary search algorithms. The view prevalent among practitioners of evolutionary computation is that genetic material is essentially symbolic rather than physical. But ignoring the physical aspects of gene translation may have led the field to underestimate the importance of developmental processes on issues like scalability and re-use. Advances in developmental biology have given us fresh insights into how evolution explores a functional domain and the constraints it operates under. Criticisms of this nature have appeared elsewhere [9] but have had little impact on the field, which continues to be dominated by efforts to optimise evolutionary search.

One agenda to investigate *how* evolutionary algorithms find solutions and *what* they are capable of finding was set in motion by Adrian Thompson in the mid 1990s [14]. His *in silico* experiments were designed to encourage as much innovation from the evolutionary process as possible. By allowing free access to the physical nature of the search domain, Thompson discovered that evolution was capable of finding solutions in areas that humans would find difficult or impossible to operate (see discussions in [5, 10, 6]). An outstanding task for evolutionary computation — for those who want to pursue Thompson's aims — is to find ways of introducing the equivalent richness of real world physics into virtual environments.

However, the introduction of richer resources does not guarantee their accessibility. We propose that in order to access interesting physical properties in evolved solutions, we need a physical embodiment of the developmental mechanisms employed by nature. This requires a two-way flow of information that allows a genotype to explore a functional domain in a manner controlled by the genome.

A crucial ability of the developmental process is to sense environmental inputs and respond. Things grow in accordance with their surroundings, using a feedback mechanism that tells cells when to start producing certain proteins or inhibit the production of others. Evidence from the study of gene regulatory networks suggests that evolution has exploited developmental mechanisms to allow the re-use of "useful" genes in different contexts [2, 3].

The issue of re-use prompts another criticism of current models of evolutionary computation, namely the "single solution genome". This artifact results from of the proximity between genotype and phenotype, the translation process between them being so direct a mapping as to make them often indistinguishable. In nature, a genome controlling the developmental process selects from many potential responses, according to the developmental context. A gene used in one place will have a different role somewhere else, roles that are separated by time and space. The repeated morphological features we witness throughout nature are the product of developmental processes. By contrast, in evolutionary computation, selection is carried out on "instant" phenotypic solutions, randomly mutating from one genera-

GECCO'07, July 7–11, 2007, London, England, United Kingdom.

tion to the next, in a process that has no natural mechanism for exploring the search space, or conserving and re-using useful genes (see discussion in §2.2). Such solutions are single, fixed answers to static environments.

Our proposed architecture addresses some of these issues by reproducing the physical exploration of genetic regulatory networks. It uses feedback to enable a developmental control system. In the following sections, we explain the biological inspiration for our architecture. We then discuss the components of our "evo-devo" platform and detail how we bind a digital genome to the physical complexity of the real world to enable developmental exploration. Finally we look at progress and issues with the platform, and its potential for future work.

2. BACKGROUND BIOLOGY

The biological inspiration for our developmental platform is based on the *cis*-regulatory regions or "switches" employed by gene regulatory networks (GRNs) that determine the contexts in which a particular gene is expressed or inhibited from producing proteins in the nucleus of a cell. These "switches" allow a gene to be re-used in a variety of contexts, which means that the protein a gene encodes will have the opportunity of interacting with different sets of proteins according to the cell's location in the organism. For example, the different bones in our body are not created by different genes encoding separate proteins for particular bones, but by the same gene being used in different contexts to create the bone material protein for a rib, a sinus, an outer ear and so on (see Fig. 1).

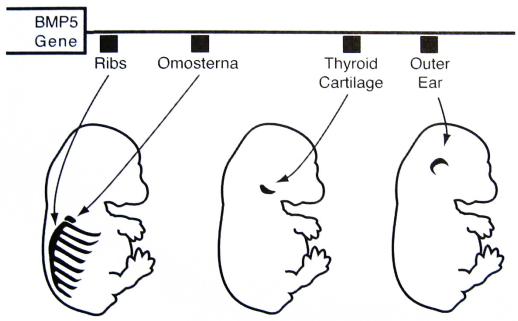

Figure 1: Different switches cause a gene for bone material to be expressed in different locations in a mouse embryo.
Image taken (with permission) from [2]

Gene switches work by certain proteins being able to *bind* to small sections of DNA material upstream of where the gene is located. These transcription factors or "binding proteins" act on DNA to inhibit or promote gene expression, and whether a transcription factor is present or not in a particular cell type is determined by that cell's location in the embryo. For example, in Fig. 2, a promoter for a gene is distributed in vertical stripes that extend to the horizontal axis of an embryo. However, the presence of inhibiting transcription factors for the same gene in the lower third and back half of the embryo results in a net expression of the gene as a series of dots along the horizontal axis.

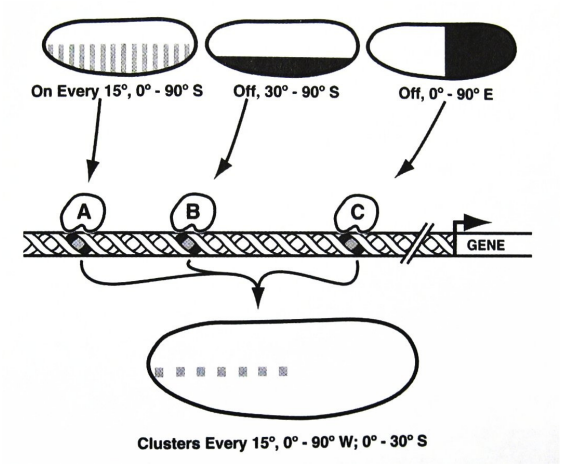

Figure 2: Gene switches acting on a drosophila embryo results in the gene being expressed as a series of dots extending halfway along the horizontal axis.
Image taken (with permission) from [2]

2.1 Gene Expression and Re-use

Pattern formation is the basis of all gene re-use: the same gene is turned on or off according to the presence of the transcription factors that occur in cells distributed throughout the embryo. Research suggests that hardly any morphological features are created *de novo* from new genes [2, 19], instead different morphological features between species are the result of the same genes (usually one of the four Hox clusters) being employed in different contexts. The Distal-less gene, essential for the formation of appendages, such as limbs or wings, is one example. In butterflies, this gene has evolved an additional "switch". The switch provides a new context for the Distal-less gene to be expressed — in this case, the location is on the wing. In this new context, rather than forming a limb bud, the gene results in an entirely different morphological feature and creates a spot of colour (see Fig. 3).

The gene switch mechanism allows re-usable, configurable instances of a gene to be expressed in the different contexts of embryo development. Repeated use of a gene in different contexts is called *modularity* by biologists, and gives rise to repetitive morphological structures such as vertebrate backbones, thorax segmentation, rib cages, leaf and wing venation, limbs, etc. Such structures are common in nature, but it has taken researchers a long time to understand the link between switches and gene expression. An important part of this interaction is how the switches work to allow the binding process some degree of flexibility.

2.2 Binding Signatures

Transcription factors attach to stretches of DNA by recognising signature sequences of base pairs. For example, a single switch for a gene may consist of several hundred base pairs (bp), lying perhaps several thousand bp upstream of the gene. Within the gene switch, there are usually 6–20 signature sequences (each ∼6–9 bp in length) that affect the expression of the gene concerned (a gene contains ∼1000 or so bp, and a chromosome contains thousands of genes, so

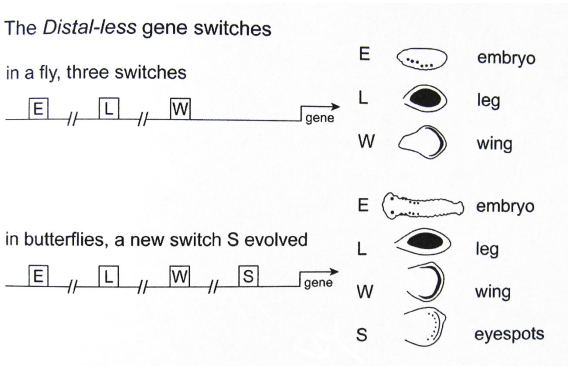

Figure 3: Switches in the Distal-less gene control expression in the embryo, larval legs and wing in both flies and butterflies, but butterflies evolved an additional switch to control expression of eye spots.
Image and text taken (with permission) from [2]

millions of bp). Even a short signature length has a huge number of possible combinations [2].

Signature sequences sometimes require exact matches for every position, sometimes they contain wildcards. Wildcard positions can be filled by all four nucleic acids (Cytosine, Thymine, Adenosine, and Guanine) but are more often limited to pairs of alternatives (e.g T or A, C or G, etc.). For example, Tinman, a gene related to heart development in most species, is highly specific. However, Pax-6 (the gene supposedly controlling the development of sight across species) and the gene Dorsal use wildcards in their binding signatures, represented by **K** (G or T), **Y** (C or T), **M** (C or A), **W**, etc. (this example is taken from [2]):

`Tinman`	`TCAAGTG`
`Pax-6(eyeless)`	`KKYMCGCWTSATKMNY`
`Dorsal`	`GGGWWWWCCM`

Thus Pax-6 has a signature with only 6 specific sites out of 16 possible bp combinations, indicating that it could bind at a variety of locations. This is borne out by experimental evidence that shows eyes can be "grown" in other contexts — such as on wings or legs — by altering the transcription factors present at those locations [2].

Binding signatures and proteins permit the genome to maintain a *set* of solutions from which it selects how to explore its functional domain. The action of "binding" is one of feedback: the information fed back to the DNA determines which genes will be expressed in that context. By exploring the functional search space in this way, developmental processes have a fundamental impact on which genes are conserved (see discussion on Hox-gene clusters in [18]). The most highly conserved genes turn out be a handful of "super" genes that evolved early on and have since been maintained across species [2]. Mutation of these genes is usually fatal, forcing evolution to preserve them. But their continued presence means that they are also more likely to be re-used by evolution during developmental processes.

3. OUR ARCHITECTURE

Our approach similarly uses a *set* of solutions that can be exploited by a phenotype in different contexts. The exploration of the functional domain is governed by interaction with the genome.

3.1 Overview

In order to pursue Thompson's aims for *in materio* evolution [15], we want to look at systems that incorporate a sufficient degree of complexity or "richness" that human engineers find challenging to design. An area that has traditionally provided this is analogue electronics. Evolutionary computation has a long track record in this area, with notable success by John Koza and his colleagues [8]. Physical implementation of analogue circuits is slow, and has meant that the evolutionary design process and fitness evaluations have run as software simulations. Although testing software simulations of analogue circuits is generally several factors slower than testing the same circuit in real-time on hardware, simulations have the advantage of being able to test many circuit configurations without having to physically implement any of them. Another drawback with software simulations of analogue hardware is the accuracy of simulated behaviour. Software is constrained to work within certain limits (e.g. number of components or parameter ranges). Outside those parameters, the accuracy of predictable behaviour goes down. This immediately hampers one of the most interesting capabilities of *in materio* evolutionary computation — which is to exploit parts of the design space that are too difficult or complex for human engineers. A software simulation has had these removed by necessity.

Recent integrated circuits (ICs) based on switched capacitor technology offer the opportunity to implement hardware based analogue circuits via software configured silicon. These ICs are similar in operation to Field Programmable Gate Arrays (FPGA), where logic functions can be downloaded onto the IC to run natively in hardware. Analogue ICs called Field Programmable Analogue Arrays (FPAA)[1] offer the convenience of implementing an analogue circuit by downloading its design onto an IC. This makes the implementation of analogue circuits fast enough to consider testing the circuits in hardware. Anadigm, a manufacturer of these ICs, has produced an FPAA capable of *dynamic* reconfiguration: changes to part of a circuit or to an entirely new circuit can be made within a single clock cycle without a full IC reset. The combination of dynamic reconfigurability, within the complex design space of analogue circuits, provides us with a good platform to see if there are benefits attached to introducing developmental control mechanisms into *in materio* evolutionary exploration.

3.2 The FPAA technology

A typical FPAA application might be described as a set of analogue circuits, with some host application controlling when the IC should reconfigure to a new circuit. Rather than evolve a single solution, this gives us the opportunity to evolve a set of solutions, specified and controlled by the genome.

The Anadigm AN221E04 FPAA is capable of implementing dynamically reconfigurable analogue circuits. Its architecture is split into 4 CABs (Configurable Analogue Blocks). Each CAB can support one or more Configurable Analogue

[1]There is a variety of these ICs with alternative names, such as Field Programmable Transistor Arrays (FPTA) or Dynamically Programmable Analogue Signal Processors (*dp*ASP).

Modules (CAMs). Anadigm supplies a range of about 40 CAMs with various configuration options. CAMs can contain circuits functioning as filters, multipliers, integrators, differentiators and so on. The CAMs are configured by setting specific options, floating point parameters and clock speeds using the AnadigmDesigner2 software[2].

The software allows analogue circuit designers to create and test circuits (either in simulation or by downloading onto the FPAA) using a "drag-and-drop" graphical user interface. The software can also be controlled by an API "wrapper" referencing the `ad2.dll` file. The build up of circuits and the reconfiguration process can then be controlled by an external application. The low-level commands (i.e. the bitstream) to trigger reconfiguration are not provided by the API interface. Instead, circuits can built up via software and saved as AHF (Ascii Hex Format) configuration files, which can be downloaded onto the FPAA as needed.

3.3 Gene Expression

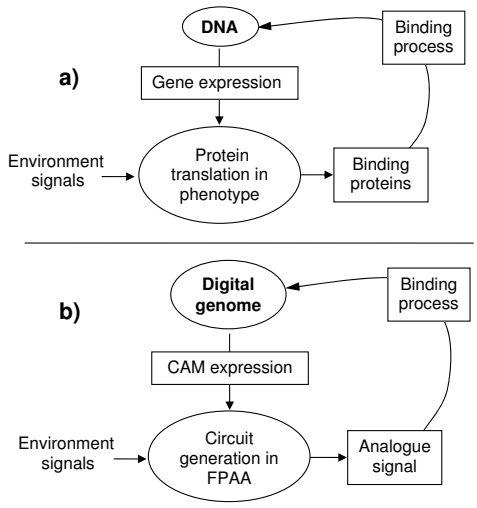

Figure 4: Diagram (a) shows natural gene expression where expression is affected by the presence of binding proteins in the cell. (b) shows our implementation of the analogous process.

In our architecture, the configuration of CAMs in a circuit is specified by the corresponding genes in a genome, and the context in which a CAM is deployed is determined by the "binding signature" for that gene (see Fig. 4). As CAM behaviour depends on the context of its deployment (i.e. its placement and connections within the circuit), one of our aims is to demonstrate that evolution will re-deploy "useful" genes in different contexts. What dictates whether a CAM is expressed or not depends on evolution being able to manipulate the binding signatures to its advantage. This ability is tied to the need to have a feedback mechanism as part of the genome. Finally, despite the fact that a genome contains the specifications and signatures for many CAMs, only a few are likely to be expressed in a phenotype "stage" of development.

[2]This software is free to download from www.anadigm.com.

3.4 Encoding the genome

The genomes and circuit specifications they encode are an adapted form of Cartesian Genetic Programming (CGP), developed by Miller and associates [11]. The genotype in CGP is represented as a list of integers that encode the function and connection of each node in a feed-foward, directed graph. In our version, we allow feedback loops between nodes as these provide interesting physical behaviour for analogue circuits. The genotypes are of fixed length but the numbers of nodes that can be realised in a circuit is dependent on what is expressed in the circuit at runtime, how the connections wire up the nodes and what the nodes represent. Fig. 5 shows the process of decoding the genome and the expression of developmental "stages" in the phenotype.

The genome representation is the part subject to mutation and its length is defined by the number of nodes. As in CGP, the number of inputs to each node is fixed. In our case, as the maximum number of inputs on any of the pre-configured CAMs supplied by Anadigm is 5, we use this and later trim the excess inputs (Fig. 5 shows only 4 inputs for clarity). The encoding links each input to an output (either from another node or itself). It is possible that one output on a node is linked many times and another not at all. In this case, the unlinked output is simply left "open". However, the process ensures that every input is connected and no inputs are left open.

Each node has an ID that specifies its CAM type and configuration. There are almost 200 CAM "primitives" if different configuration options are taken into account.[3] Within each primitive configuration, a large range of behaviour is possible as most CAMs take floating point parameter values. We encode a maximum of 4 of these, with those that are unused being ignored. Parameter values in CAMs are dynamic, i.e. setting certain options, or changing the values of other parameters, can affect the range of values acceptable to the CAM in that configuration. The encoding is interpreted as a percentage value, which is then applied as a function of the parameter range when the configured CAM is added during the circuit translation.

The final part of a gene specification is the binding signature for each CAM. This is a string representing the 4 nucleic acids in DNA. A single FPAA can fit around four to seven CAMs depending on their resource usage, therefore the signature length is generally proportional to the expected expression rates, i.e. choosing a very short binding signature might result in the expression of too many CAMs for most contexts. We will allow evolution to choose the signature length, but in our current prototype this is fixed.

Once the directed graph is generated, it is necessary to translate the node IDs into the specifications of the corresponding CAM. This means that most nodes have their inputs trimmed to the number on the corresponding CAM. For example, in Fig. 5, node 2 has 4 inputs and 1 output, but the CAM corresponding to its ID has 2 inputs and 2 outputs. Both of node 2's inputs (shown in the specification as 3:2, 3:1, etc) are linked to the first and second outputs of node 3. However, the CAM that node 3 represents has itself

[3]The actual number is larger. However, for reasons involving the design of the API it easier to build up the CAM library omitting some of the option configurations.

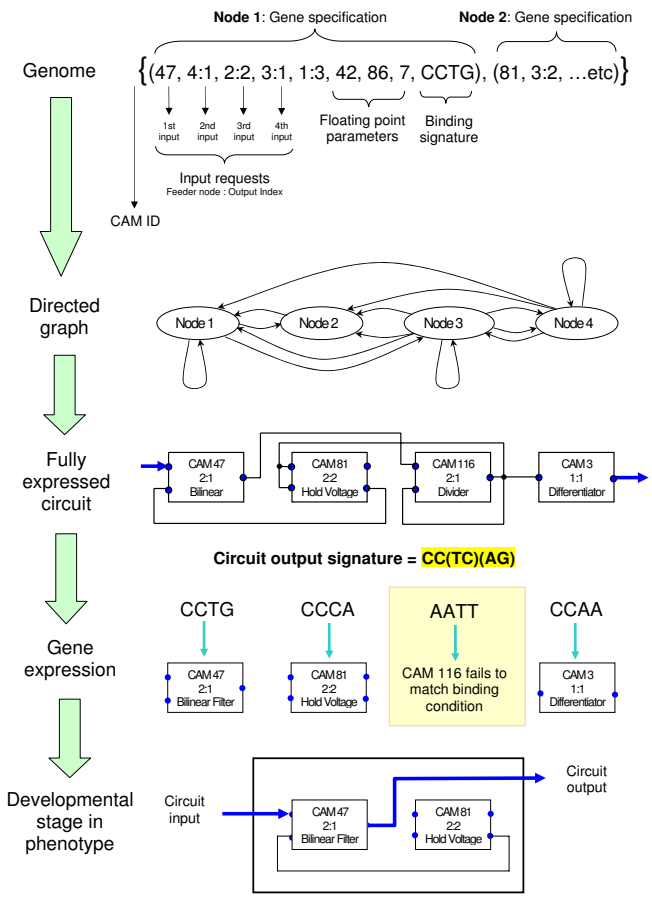

Figure 5: Genome to phenotype: decoding a genome into a developmental stage of the circuit phenotype. When implemented with a changing input signal, circuit output is fed back to the genome which reacts by expressing those CAMs that match the binding signature, allowing the genome to respond to dynamic conditions.

only one output, therefore this output is used to feed both of node 2's inputs. For node 2's outputs on the other hand, while node 1 requests the second output, no other node links to the first output, and so that output is left "open". The process of re-aligning inputs and outputs to the actual values of the CAMs concerned means that although all inputs will be connected, some outputs are likely to be left open (the figure averages around 20% of the number of nodes).

The next part of the process involves the expression of subsets from the genome. A subset could contain any combination of CAMs and each subset is turned into a circuit configuration (using the wiring connections of the fully expressed circuit). The total number of potential subsets quickly gets large as the number of nodes in the genome increases (see issues, §6). As each subset is generated, many with missing CAMs, the circuit wiring specification from the genome no longer holds good for all inputs. For example, Fig. 5 shows a binding condition in which node 3 fails to be expressed. This leaves nodes 1, 2 and 4 to create a circuit configuration. However, node 1 has only one output connected to node 3. As node 3 has not been expressed in this stage of the phenotype, the circuit is split in two.

In cases such as these, the wiring algorithm defaults to using the first CAM as the circuit input, even if that means that much of the circuit that has been expressed cannot be used. The wiring algorithm then traces through the circuit to the first "open" output that will enable the circuit to have both an input and output. In this case, the first open output it finds happens to be on the first CAM, and so the other CAMs (2 and 4) fail to have any impact on the circuit. The final developmental stage of the phenotype circuit is shown with only node 1 making up the circuit. Node 2 has been left in as a connection can still be made to the first input and although no input goes to node 2, the connection may still affect the fitness for this developmental stage and we do not make any judgement on how "sensible" a circuit is from the viewpoint of conventional design.

Maintaining the fully expressed circuit wiring specification allows a context in which all CAMs in a genome can be expressed. If this is the case, at least all inputs for that expression will be satisfied and some form of circuit utilising all of the CAMs can be realised. However, it also allows us to investigate the influence of evolution on the interaction between the context of an expression and the binding signatures of individual CAMs. If a CAM has connections specified as coming from another CAM that is never expressed in the same context, then evolution is forced to work on the binding signatures that could allow those connections to be realised. A later mutation to a binding signature may allow expression of the missing CAM in that context, it may re-wire connections or it may even change the CAM completely. What is important is that having a genome containing a particular CAM does not result in a predetermined circuit behaviour. Rather like the re-use of genes in different developmental contexts, under our schema different signal contexts are likely to result in a CAM behaving differently as its environment changes.

This indirect mapping from genotype to phenotype allows us to investigate *phenotypic stages of development*, as a phenotype may get high fitness in one context, low in another. We can assign progressively difficult tasks for contexts, and look to see what "useful" genes are conserved to allow the genome to "bootstrap" over generations. We can also alter fitness criteria for a particular context after a genome has already achieved good fitness for other contexts and see what impact this has on the re-organisation of the genome. This aspect of the architecture is discussed further in §7. A broader overview of the architecture is shown in Fig. 6.

4. DIGITISING THE SIGNAL

To access the complexity and richness of the physics within analogue hardware, we need some means of processing that environment digitally so that it can interact with evolutionary processes on the host application. The first part of this is the decoding of the genome into phenotype circuits. The second part is the binding process that allows the genome to receive feedback from its functional domain. Translating the physical complexity of the analogue output signal from the FPAA into a digital form is done via a wavelet transform. The process of matching this signal (i.e. the "context") to a set of a binding signatures in the genome circuit specification is done separately and is covered in §4.2.

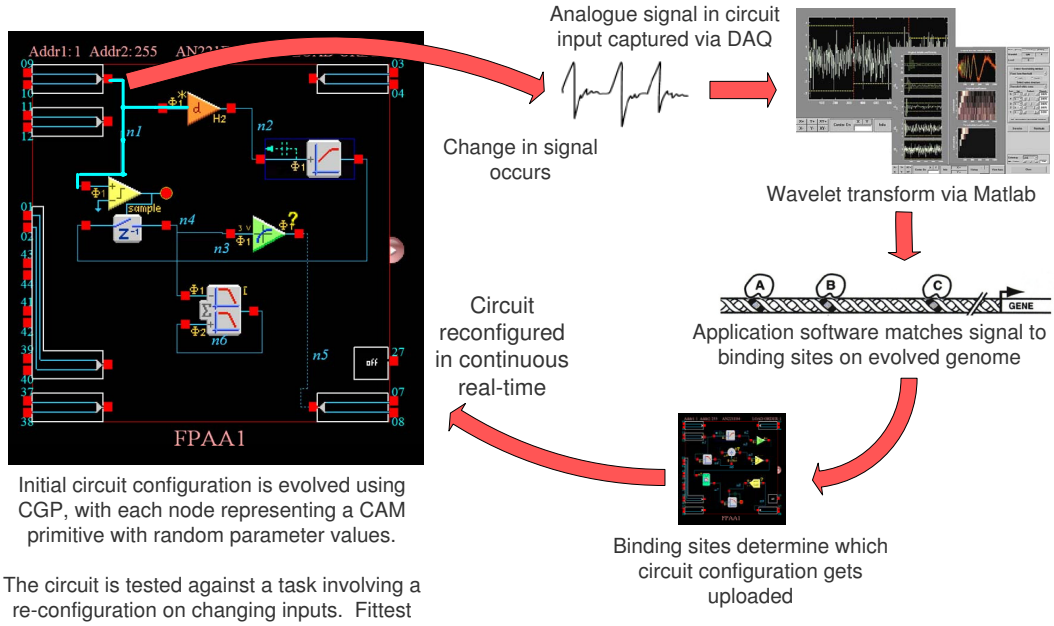

Figure 6: The developmental FPAA architecture, showing how feedback from conditions in the circuit wiring "bind" onto parts of a digital genome. When binding occurs the CAMs selected are expressed as a new circuit configuration.

4.1 An Overview of Wavelets

Wavelets grew out of the requirements of a group investigating new techniques for locating oil from the return of impulses applied to the ground [7]. The return signal was cluttered. To process it properly required the ability to resolve simultaneously in time and frequency. The Fourier Transform [13] defined in (1) provides excellent resolution in the frequency domain, but time domain information is lost. Truncation of the signal produces conflicting artifacts (spectral spreading and leakage) related to the length and shape of the time window over which the signal is expressed.

$$F(\omega) = \int_{-\infty}^{\infty} f(t) e^{-i\omega t} dt \tag{1}$$

There are methods for improving this trade-off between frequency and time. One such method is the Short Time Fourier Transform (STFT) [12], given in (2):

$$STFT_f(\omega, \tau, \gamma) = \int_{-\infty}^{\infty} f(t)\gamma(t-\tau)e^{-i\omega t} dt \tag{2}$$

The STFT gives the frequency components within different windows defined by the windowing function $\gamma(t-\tau)$. This function allows the frequency distribution within a time band to be measured. Whilst this is an improvement on the time-frequency resolution of the Fourier transform, it is still limited because of the window size. The smaller a window is, the better a high frequency component can be located in time. However, with smaller windows few cycles of low frequency components will be observed and so there is a loss of resolution [7]. Wavelets are designed to remove these artifacts. We can use them to analyse the output from our analogue circuit without distortion. The continuous wavelet transform, $W_f(a, \tau, t)$, effectively "cuts up data or functions or operators into different frequency components, and then studies each component with a resolution matched to its scale" [4]:

$$W_f(a, \tau, \psi) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(t)\psi\left(\frac{t-\tau}{a}\right) dt \tag{3}$$

The wavelet transform provides a time-frequency localisation that is not available using the STFT. By calculating the correlation between the desired signal and a wavelet function then scaling, dilating and shifting the wavelet function before repeating the process, an accurate representation of the frequency variability with time can be built up. The transform repeatedly compares the similarity between the wavelet and the signal under inspection at different dilations and shifting positions.

In general, the energy contained within a signal, the average of the square of the signal, will be preserved by the wavelet transform and so the original signal can be recovered. The function $\psi(t)$, the "mother wavelet", is unscaled, undilated and unshifted. The dilation is inversely linked to the frequency band that the wavelet will detect. At a high dilation the wavelet is stretched over a wide time period and so is more suited to low frequency signals. At low dilations the wavelet is compressed into a smaller time frame and so will pick up high frequency signals. The term, a, in the wavelet transform is the dilation. In addition the a factor ensures that the energy in the transform is normalised [12]. There is a wide range of wavelets that have been created and each has varied properties. When performing the wavelet

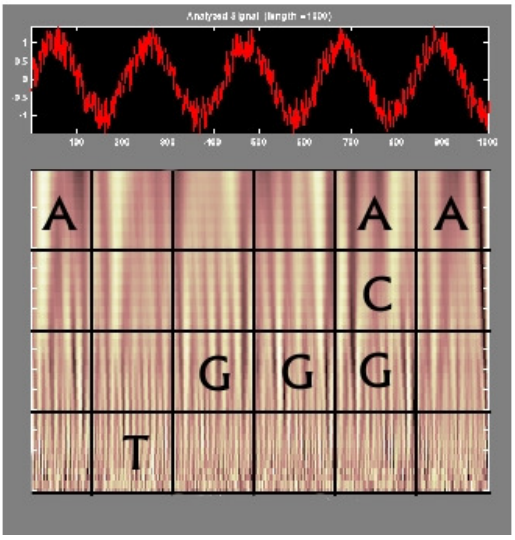

Figure 7: The circuit output behaviour analysed as a wavelet using Mathworks' Wavelet Toolbox. A grid is placed over the transform that matches the signal to the evolved binding sequence in the genome. Here, the binding signature would be ATGG{ACG}A, with a wildcard position in column 5.

transform a critical factor in obtaining an adequate result is that the appropriate wavelet function is selected. For simplicity, we next give an idealised example of a wavelet transform and how we would use it in the binding process.

4.2 The binding process

We take the wavelet transform of a signal, such as that shown in Fig. 7. This plots time (shifting) along the x-axis, frequency (dilation) along the y-axis, and the degree of correlation between the dilated and shifted wavelet and the signal at that time as an intensity value. Onto this we overlay a grid representing the 4 bases (A,C, T, G) as 4 rows. The grid works almost as a truth table, in that we inspect the average of the correlation values (wavelet coefficients) in a grid cell to see if it falls above a threshold or not. If the answer is positive, that square equates to that base being present at that position in the signature. The grid is read column by column. More than one value in a column means that this signature has a wildcard at that position. If no square in a column is above the threshold, then that position is ignored. As many positions are read as required to match a signature. The grid can thus be extended to meet the total number of CAMs / nodes in the genome as needed. Fig. 7 indicates (figuratively) how the scheme works — the exact details of the threshold values and which wavelet transform to use are still being worked on.

5. APPLICATION AREAS

Evolving a control system capable of responding to many circuit states is an interesting task and there are potentially many application areas. However, the notion of a dynamically reconfigurable analogue circuit is relatively new and therefore alien to most analogue engineers. When looking at traditional domains for signal processing for example, most applications have tried to put the "intelligence" into the digital signal processing after the sensor. However, there are occasions where analogue processing could bring benefits.

Analogue processing is fast, requires no processing overhead and works with low power. Areas such as robotics and ubiquitous computing would benefit from adaptable analogue circuits. An important aspect of analogue signal processing is filters. Analogue filters are used mainly for signal conditioning and for discriminating wanted from unwanted signal frequencies. In communications, filters are used to reduce or eliminate adjacent channel interference and reduce inter-symbol interference. In the interfacing of the real world (analogue) with computers (digital), analogue filters are used to eliminate aliasing. Traditional approaches to designing analogue systems involves conventional analogue circuit design methods, followed by circuit simulation, prototyping and fabrication. Reconfigurability significantly eases that aspect of the design when an application requires filter characteristics to change in response to changes in system operating condition.

One issue holding the take up of this technology is that analogue signal processing is still a specialist design domain. It is conceivable that the FPAA will have a similar impact as the FPGA has had in industrial control engineering [17]. For many years, the programmable logic controller (PLC) has dominated industrial automation. The FPGA-based programmable automation controller (PAC) is challenging this stronghold. One important factor that has stimulated this is the availability of graphics-based rapid prototyping tools that eliminate the need for "esoteric" VHDL hand-coding. We hope that bringing in evolutionary search algorithms to the domain could automate design to a degree.

6. PROTOTYPE AND FUTURE WORK

We have implemented the key concepts of our architecture in a working prototype system, which automatically reconfigures the FPAA in response to environmental changes through the feedback expression process. A host PC genome application implements binding and genome expression, and downloads configurations to the FPAA. An environmental signal is fed into the FPAA; the FPAA's output is sent to the host PC, where it is transformed before being passed to the genome application to see if binding can occur.

Our prototype implements a simple behaviour that depends on the frequency of the input signal (here alternating between two different frequency sine waves). The single genome can be expressed in two ways, as a low band pass or as a high band pass filter. The output from the FPAA circuit is continuously Fourier transformed (in this simple case), resulting in one of two peaks. When the genome application detects a change in frequency, it triggers a different genome binding (analogous in this case to a change between an AT and a TA site, in the terms of Fig. 7). This results in a change in circuit expression, and the genome application downloads the appropriate newly expressed filter, thereby reconfiguring the FPAA. The cycle repeats when the next input signal frequency change occurs.

This prototype demonstrates both a proof of concept of many of the components of the architecture, and that the chosen FPAA is a suitable implementation platform. The FPAA configuration software was not designed to be used in

this way, with gross changes to unknown configurations, and Anadigm have been very helpful with our implementation.

Our next step is to replace the Fourier transform of the output signal with a wavelet transform, and integrate it into the binding algorithm. We will then embed the genome application in an evolutionary harness, and evolve (rather than handcode) suitable genomes and bindings. This will require the design of fitness functions evaluated over multiple contexts.

There is a practical constraint on genome length, due to the limited number of CAMs that can fit onto a single FPAA. This could be increased by daisy-chaining a series of FPAAs: for example, with 4 FPAAs, genomes could have lengths of 16–30 nodes. One issue that may arise with longer genomes is the large number of different possible expressions (subsets of CAMs) that they support. With so many subset expressions possible, care needs to be taken to ensure that the circuit's output signal after wavelet transform has sufficient complexity and richness to achieve the fine discrimination necessary to explore the different subsets.

7. CONCLUSIONS

Biological evolution has come up with a neat trick: DNA encodes for proteins, and those proteins can govern the production of other proteins. Thus it encodes for the rules that dictate how it explores a particular functional domain. Not only that, but a tiny fraction of what could be expressed is ever realised in a phenotype. A genome contains solutions for *sets* of contexts. Change the contexts and the genome still has room for developmental exploration. This flexibility and redundancy of solutions has an impact on the re-use and conservation of genes during developmental processes. We believe that without similar mechanisms of interaction and feedback, digital genomes cannot guide themselves across functional search spaces in a way that fully exploits a domain's resources, and this is particularly true where that domain includes the complexity provided by real-world physics.

Our feedback architecture is a first attempt to physically reconstruct the exploratory mechanism of genetic regulatory networks. Our future work will use this platform as a basis from which to evolve developmental control systems.

8. ACKNOWLEDGEMENTS

Special thanks to Dave Lovell at Anadigm for helping us with reconfiguration of the FPAA and host application. Equipment was granted from the University of York from start up funding for the York Centre for Complex Systems Analysis. The research is funded by the EPSRC and Microsoft Research UK.

9. REFERENCES

[1] W. Banzhaf, G. Beslon, S. Christensen, J. Foster, F. Kepes, V. Lefort, J. Miller, M. Radman, and J. Ramsden. Guidelines: From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7(9):729, September 2006.

[2] S. Carroll. *Endless Forms Most Beautiful: The New Science of Evo Devo and the Making of the Animal Kingdom*. Weidenfeld & Nicolson, 2006.

[3] S. B. Carroll, J. K. Grenier, and S. D. Weatherbee. *From DNA to Diversity*. Blackwell, 2001.

[4] I. Daubechies. *Ten Lectures on Wavelets*. Capital City Press, Montpelier, Vermont., 1992.

[5] T. G. Gordon. Book review of Hardware Evolution by Adrian Thompson. In *Genetic Programming and Evolvable Machines*, 2(4). Kluwer Academic Publishers, 2001.

[6] S. Harding and J. F. Miller. Evolution in materio : A real-time robot controller in liquid crystal. In J. Lohn, D. Gwaltney, G. Hornby, R. Zebulum, D. Keymeulen, and A. Stoica, editors, *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, pages 229–238. IEEE Press, 2005.

[7] B. Hubbard. *The World According to Wavelets — The Story of a Mathematical Technique in the Making*. A K Peters, Natick, MA, 1998.

[8] J. R. Koza, L. Jones, M. Keane, and M. Streeter. Towards industrial strength automated design of analog electrical circuits by means of genetic programming. In U.-M. O'Reilly, T. Yu, R. L. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, chapter 8. Kluwer, 2004.

[9] S. Kumar and P. Bentley, editors. *On Growth, Form and Computers*. Elsevier Academic Press, 2003.

[10] J. F. Miller and K. Downing. Evolution in materio: Looking beyond the silicon box. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *The 2002 NASA/DoD Conference on Evolvable Hardware*, pages 167–176. IEEE Computer Society, 2002.

[11] 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, *Genetic Programming, Proceedings of EuroGP'2000*, volume 1802 of *LNCS*, pages 121–132, Edinburgh, 15-16 Apr. 2000. Springer-Verlag.

[12] A. Poularikas. *The Transforms and Applications Handbook*. CRC Press, Boca Raton, FL, 1995.

[13] S. Soliman and M. Srinath. *Continuous and Discrete Signals and Systems*. Prentice Hall International, Upper Saddle River NJ, 1998.

[14] A. Thompson. Silicon evolution. In J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 444–452. MIT Press, 1996.

[15] A. Thompson. Notes on design through artificial evolution: Opportunities and algorithms. In I. C. Parmee, editor, *Adaptive computing in design and manufacture V*, pages 17–26. Springer-Verlag, 2002.

[16] G. Tufte. Cellular Development: A Search for Functionality. *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 2669–2676, 2006.

[17] T. Walter and I. Bell. Where others fear to tread. *Computing and Control Engineering Journal*, pages 26–29, April/May 2006.

[18] L. Wolpert. Relationships between development and evolution. In P. Bentley and S. Kumar, editors, *On Growth, Form and Computers*. Elsevier, 2003.

[19] L. Wolpert, R. Beddington, and T. Jessell. *Principles of Development*. Oxford University Press, 2002.