Exploiting Development to Enhance the Scalability of Hardware Evolution

Submitted to the University of London for the degree of
Doctor of Philosophy

Timothy Glennie Wilson Gordon

Department of Computer Science
University College London
July 2005

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Abstract

Evolutionary algorithms do not scale well to the large, complex circuit design problems typical of the real world. Although techniques based on traditional design decomposition have been proposed to enhance hardware evolution's scalability, they often rely on traditional domain knowledge that may not be appropriate for evolutionary search and might limit evolution's opportunity to innovate.

It has been proposed that reliance on such knowledge can be avoided by introducing a model of biological development to the evolutionary algorithm, but this approach has not yet achieved its potential. Prior demonstrations of how development can enhance scalability used toy problems that are not indicative of evolving hardware. Prior attempts to apply development to hardware evolution have rarely been successful and have never explored its effect on scalability in detail.

This thesis demonstrates that development can enhance scalability in hardware evolution, primarily through a statistical comparison of hardware evolution's performance with and without development using circuit design problems of various sizes. This is reinforced by proposing and demonstrating three key mechanisms that development uses to enhance scalability: the creation of modules, the reuse of modules, and the discovery of design abstractions.

The thesis includes several minor contributions: hardware is evolved using a common reconfigurable architecture at a lower level of abstraction than reported elsewhere. It is shown that this can allow evolution to exploit the architecture more efficiently and perhaps search more effectively.

Also the benefits of several features of developmental models are explored through the biases they impose on the evolutionary search. Features that are explored include the type of environmental context development uses and the constraints on symmetry and information transmission they impose, genetic operators that may improve the robustness of gene networks, and how development is mapped to hardware. Also performance is compared against contemporary developmental models.

Acknowledgements

I would like to thank the following people:

My supervisor, Peter Rounce for all his encouragement and advice, for proof reading, and for his help in funding both the hardware that my work relied on and my conference trips around the world.

My other supervisor, Peter Bentley for fostering my interest in development, adopting me as a student, showing me how to write, convincing me I had something to write about and pestering me into finishing this thesis. Peter's support has been invaluable.

Bill Langdon for our many constructive discussions about the field and my work, and Julian Miller for his useful comments.

Scott McMillan at Xilinx for his help debugging and patching the early versions of JBits.

Daniel Roggen for supplying the patterns for the Norwegian flag problem.

Mike Brent for providing access to and support for the VLSI consortium tools, and Denis Timm and the rest of the Systems Group for the use of their equipment, from crocodile clips to oscilloscopes.

Matt Trotter, David Corney, Rob Burbidge and the other members of nUCLEAR and Primal, for helping me understand Machine Learning and Evolutionary Computation and for their friendship. Dirk Weirich for showing me how a Ph.D. should be done.

My parents, without whose support and encouragement this work would never have been started, let alone finished.

Finally I would like to thank my girlfriend Lianne King for so much: listening to my reasoning, double checking my algebra, sitting through endless practice talks, helping format and proof read this document, and putting up with a lot of boring weekends. You are a star!

Contents

List of Figures

List of Tables

1 Introduction

In the hundred years since John Ambrose Fleming invented the diode at University College London and gave birth to the field, electronics has become a well established engineering discipline. The resulting reservoir of knowledge has allowed the commercial semiconductor industry to grow at a remarkable rate in the intervening years, both in volume and device complexity with the now-famous Moore's Law holding true for forty years (Moore, 1965). The advent of ever cheaper, more efficient and more powerful electronics has changed society to the point where reliance on electronics for many daily tasks is now common. However there are still many niches where even cheaper, more efficient and more powerful hardware would be useful and it is the desire to exploit these niches that drives industry growth. In recent years improvements in hardware have for the most part been achieved by technology scaling: improving hardware fabrication processes rather than moving to fundamentally new circuit medium technologies or circuit designs. Yet it is becoming increasingly difficult to sustain the current rate of scaling using today's technologies without resorting to expensive tricks (McFarland, 1997; Borkar, 2000; Kahng, 2004).

This means that in the medium term the requirement for innovative circuit designs is likely to grow as it becomes necessary to squeeze more and more out of current circuit medium technologies. The long-term solution is likely to lie in the development of new technologies. But even when they do eventually become feasible commercially, they are likely at best to require design features that current circuit design methodologies were not conceived to provide, such as fault tolerance, and might require a complete rewriting of the design rules. Hence in the long term there is also a significant requirement for innovative circuit designs and design methodologies, and the cost of developing these in man-hours of research and design is likely to be considerable.

Over the past decade a new field applying evolutionary techniques to hardware design and synthesis has emerged. These techniques may be able provide a new option: evolution can be used to design and synthesise innovative circuits automatically, or at least aid in their design and synthesis by revealing innovative design concepts that traditional designers can learn from.

1.1 Scalability

An interesting feature of hardware evolution that is potentially useful for tackling the issues outlined above is its ability to discover circuits without relying on all the constraints that human designers require to simplify the design process. It has been shown that by relaxing such constraints evolution can discover innovative circuits more efficient than any conventional designs (Thompson and Layzell, 1999). So it may seem surprising that there are few examples of evolutionary methods being applied to real-world circuit design problems. This can be attributed to a number of issues including the limited robustness of, and difficulty in interpreting evolved solutions. But the most conspicuous issue is scalability: larger, more complex problems typical of those faced by real-world designers tend to be difficult for evolution to solve. Schema theorem suggests that evolution inherently designs solutions by combining small, weakly interacting components incrementally (Goldberg, 1989). It is often unclear how the parameters of a design problem can be presented to evolution so that such components emerge, and it rapidly becomes more difficult to do so as problem size increases and the fraction of potential solutions sampled by the algorithm at any one time diminishes (Thierens, 1999). The issue of scalability is likely to be most prevalent when evolution juggles huge numbers of tiny components as it does when designing a large circuit when traditional constraints have been relaxed. It is here that evolution is most in need of a helping hand and it is here that there is the greatest opportunity to discover innovative circuit designs

Many researchers have explored solving large problems with evolution by designing more complex components for evolution to operate on. Such an approach is usually dubbed Function Level Evolution (Higuchi et al., 1997; Zhao-Shuguang and Yang-Wanhai, 2002). This does not solve the scalability problem, but merely moves it somewhere else: a human expert is now required to design modules suitable for evolution (a tricky task in itself), and crucially the opportunity for innovative design within and between the modules is lost.

The conventional engineering approach to a solving large problem is to break it up into modules that interact with each other in a limited manner. For instance the software designer's tenet is that code should be arranged into modules with "low coupling" and "high cohesion" (Sommerville, 1992). The abstractions used by conventional circuit designers serve the same purpose. Perhaps because of the parallel between the weakly interacting components required by schema theorem and the modules that result from traditional design decomposition, some hardware evolution researchers have suggested using similar methods of decomposition to create a series of problems each of which is small enough for evolution to tackle (Torresen, 2002a; Shanthi et al., 2004). However decomposing a problem into strict sub-problems using traditional methods removes any opportunity for evolution to find innovative ways to partition

the problem itself. Furthermore there is no guarantee that the sub-problems that traditional designers find useful will be useful to evolution.

This thesis will argue that a better solution is for evolution to design its own modules from the ground up, thereby removing the need for a human designer, providing opportunity for innovation at all levels of abstraction, and allowing evolution to discover modules that are genuinely useful for evolutionary design rather than those presumed to be useful by a human designer.

Attempts at automatic modularisation in evolutionary design are not new. Methods such as Automatically Defined Functions (Koza, 1994) and Adaptive Representation through Learning (Rosca, 1995) have been used for many years in genetic programming, a branch of Evolutionary Computation. Although these have proved useful in many respects, the most impressive examples of evolutionary scalability, design innovation and automatic modularisation are still confined to biological evolution: the size and complexity of biological organisms ranges over many orders of magnitude, and for hundreds of millions of years evolution has managed to consistently generate new, innovative designs. Most large biological organisms are clearly highly modular: the body plans of most animals are made up of distinct tissues and organs. They also exhibit hierarchical modularity: insects have compound eyes and vertebrates segmented spines. These subcomponents are in turn composed of many cells.

The process that allows evolution to generate intricate, innovative organisms across such a wide range of sizes is *development*, the process by which egg becomes adult. This thesis will show that by applying a model of development to hardware evolution, evolution is able to discover large circuits without relying on the traditional abstractions that might limit innovation but would be necessary using a conventional evolutionary algorithm, and that there is potential benefit in allowing evolution to do so.

This thesis does not solve the scalability problem. It merely demonstrates a step towards that goal, highlighting mechanisms inherent to development that can enhance scalability, in particular for hardware evolution without the aid of the usual complement of traditional design constraints. In doing so, several models of biological development that have been tailored for both implementation in hardware and for the design of hardware are presented.

1.2 Hardware Evolution

Evolutionary Computation is the field of solving problems using learning algorithms inspired by biological evolution. These algorithms are collectively known as *evolutionary algorithms*. They

model the principles of selection, variation and inheritance that are the basis of the theory of Darwinian evolution, and have been applied to a huge spectrum of problems from classic optimisation (Langdon, 1997) to the creation of original music (Biles, 1994). Typically they work on a population of prospective solutions in parallel. Each member of the population is evaluated according to a problem-specific *fitness function* that tests how well each solution performs a required task, and assigns it a *fitness* score. A *selection* operator then chooses solutions with higher fitness probabilistically from the population to form the basis of a new generation of solutions. These solutions are then varied. Most commonly this is achieved by randomly altering each solution to model *mutation*. Another common operation called *crossover* recombines two solutions in some way to model sexual reproduction. The process is then iterated for a number of generations until a stopping condition is met. Typical stopping conditions are the discovery of a solution with a given fitness, or the completion of a predefined number of generations.

This thesis concerns the application of evolutionary algorithms to the automatic design and synthesis of electronic circuits, a field that has been coined *evolutionary electronics*, *evolvable hardware* and *hardware evolution* amongst others. Throughout this thesis the field will be referred to as hardware evolution, and will be explored in more detail as the work presented here is set in context in Chapter 2. In order to familiarise the reader with how circuits might be evolved, an example of the hardware evolution is now presented using the evolutionary algorithm used throughout this thesis.

1.2.1 An Example of Hardware Evolution

The class of evolutionary algorithm used in this thesis is a *genetic algorithm* (Goldberg, 1989). Most commonly these operate on a fixed size population of fixed length binary strings called *chromosomes*, where each bit can referenced by a unique index called a *locus*. Each chromosome encodes a set of parameters that describe a collection of electronic components and their interconnections, thus each set of parameter values represents an electronic circuit. The set of all possible combinations of parameter values defines the *search space* of the algorithm, and the circuits that they represent define the *solution space* of the algorithm. Traditionally every parameter set in the search space encodes a unique circuit description in the solution space. When referring to a chromosome / circuit pair, biological terminology is often used: the chromosome is often called the *genotype* and the circuit it represents is called the *phenotype*.

An example of hardware evolution is shown in Figure 1.2.1. The algorithm begins by initialising the bits of each chromosome with random values. The chromosomes are then

evaluated in turn by creating a circuit based on the parameter values, either as a simulated model of the circuit or as a concrete circuit embodied in reconfigurable hardware, an example of which is introduced in the next section. The circuit's fitness for performing the target task is then measured by passing it a set of test inputs and evaluating the circuit's output. The selection operator then populates the next generation of chromosomes probabilistically such that chromosomes with high fitness are more likely to be selected.

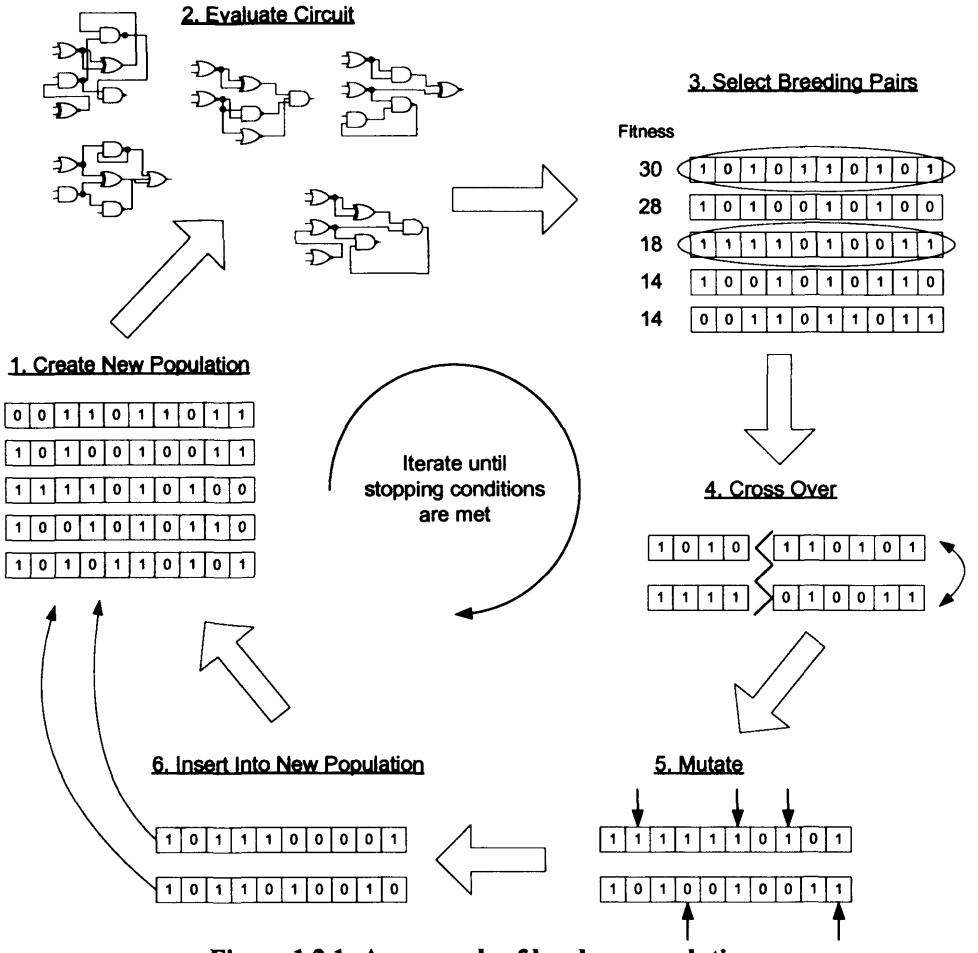


Figure 1.2.1: An example of hardware evolution.

There are many methods to achieve this, and the approach used in this thesis is called *two-member tournament selection* (Goldberg, 1989): the operator selects two individuals at random and compares their fitness. Only the individual with the highest fitness is inserted into the next generation. If they have equal fitness the individual to be inserted is chosen at random. Once the new population has been selected it is varied using *one-point crossover* and *point mutation* operators (Goldberg, 1989), again two common variation operators of many that have been reported. One point crossover recombines two chromosomes by choosing a locus at random along the chromosome and swapping every bit beyond this point between the strings. It is applied stochastically according to a fixed probability. Point mutation independently inverts each bit in the chromosome according to a fixed probability. Another crossover operator used occasionally in this thesis is *uniform* crossover. Here each bit in a chromosome is swapped

independently, according to a fixed probability. The crossover and mutation operators are applied to all members of the new population. Finally some copies of the best member of the original population are copied into the new population unchanged, a strategy called *elitism* (Goldberg, 1989). The new population is now complete and the algorithm then iterates the steps of evaluation, selection and variation until the stopping conditions are met.

1.2.2 Abstraction in Hardware Evolution

Details of the types of circuit that can be evolved and how these are represented by the parameters that make up the genotype have so far been omitted from this discussion. Typically each parameter describes the behaviour of a single component of the circuit or an interconnection between two components. Circuits have been evolved using an array of component and interconnection types that represent the entire spectrum of circuit design abstractions, from high level behavioural descriptions (Hemmi et al., 1996; Araujo et al., 2003) through networks of logic gates (Kajitani et al., 1998) (Miller et al., 1997) to networks of transistors (Stoica et al., 2002) or other analogue components (Zebulum et al., 1998).

One of the most interesting abstractions was used by Thompson (Thompson, 1996a), who evolved configurations for a Field Programmable Gate Array (FPGA). FPGAs are integrated circuits that consist of arrays of identical electronic components and wires interconnecting them. The behaviour of each component or interconnection can be altered by passing configuration data to the device. By passing the FPGA a configuration bitstream that describes several components and their interconnections, a circuit design can be realised in hardware. FPGAs are available commercially at low cost. They are one of several classes of reconfigurable devices that are useful for evolutionary work as they allow circuits to be created and evaluated in real hardware (termed *intrinsic* evolution). This avoids the need for a simulation of the circuit behaviour (termed *extrinsic* evolution), which can be computationally expensive.

Each bit in Thompson's chromosomes directly represented a bit in the configuration bitstream of the FPGA, with almost every bit for a small section of the chip included in the chromosome. This meant that evolution was free to explore almost all possible configurations hence almost all possible behaviours that the evolved area of the chip could manifest. It might be said that he evolved the section of the chip at the lowest possible level of configurability, or using the lowest possible design abstraction, for that FPGA. The majority of the solution space was made up of circuits that would never be considered by conventional digital circuit designers as they break some of the basic rules of circuit design. The best circuit found by evolution was one of these circuits, with much more efficient design than conventional techniques would yield. It could not have been discovered if evolution had explored the design space using a conventional design

abstraction, thus demonstrating that permitting evolution to work at low levels of abstraction increases the opportunity to discover innovative circuits.

The area of the chip that Thompson evolved was very small by conventional design standards, certainly too small to perform computations of the order required by most real-world problems. Even so the search space that resulted from allowing evolution to directly manipulate the FPGA at its lowest level of configurability was vast by evolutionary terms. It has already been discussed that generally evolution is not able to search very large spaces effectively because it cannot sample enough of the space to direct it to good solutions. This scalability problem has been recognised by many hardware evolution researchers as an issue that is critical to the field (Torresen, 2000b; Vassilev and Miller, 2000c; Haddow et al., 2001; Gordon and Bentley, 2002; Koza et al., 2003; Shanthi et al., 2004). The only reports of evolving circuit designs significantly larger than Thompson's intrinsic demonstration have been achieved by strictly limiting evolution's search to more conventional areas of solution space through abstraction, for example (Hemmi et al., 1996; Thomson and Arslan, 2003). However with this kind of approach, any solutions to large circuit design problems that rely on behaviours beyond the scope of conventional circuit design are also beyond the scope of evolutionary design. If evolution is to be allowed the opportunity to innovate at low design abstractions, another approach is needed.

1.3 Development

Nature does not appear to suffer from the scalability problem. It has managed to evolve large organisms without losing the ability to design complex, intricate features such as neural networks or immune systems. Many students of biology, for instance (Wagner and Altenberg, 1996; Kirschner and Gerhart, 1998; Wolpert et al., 2002; Schlosser, 2004), believe that the key to nature's ability to learn and manipulate complex adaptations is the process of development.

The term *development* describes the process that transforms a single-celled embryo into a complex adult organism (Wolpert et al., 2002). In this thesis it is used as a metaphor for the transformation of genotype to phenotype. The entire process is by no means completely understood. It encapsulates a huge array of interactions between genes, their products and the environment, from microscopic to macroscopic, some of seemingly minor importance, some ubiquitous to all stages of development. One mechanism that has a hand in all stages of development is differential gene activation: the genes that are active in a given cell at a given time can differ from those active in another.

1.3.1 Differential Gene Activation

Differential gene activation is on the whole self-regulated. The products of gene activation are proteins, and many of these play a role in gene activation itself. Such proteins are called *transcription factors* and control gene activation by interacting directly with the gene itself. The rate at which a particular protein is generated is dependent on the rate at which the gene that describes it is transcribed, hence the rate at which the gene's transcription factors are generated. As a transcription factor is simply a gene product, the rate at which it is generated relies on the rate at which other transcription factors are generated, and so on. The specificity of transcription factors varies, some affecting a single gene while others may be involved in the regulation of a number of genes.

1.3.2 Gene Regulatory Networks

The relationship between regulatory genes and their transcription factors can be modelled as a directed graph, where the nodes represent genes and the edges represent their products. If a gene product is a transcription factor, it is represented by edges connecting its parent gene to the genes it regulates. This type of network is known as a Gene Regulatory Network (GRN) (Kauffman, 1969), an example of which is shown in Figure 1.3.1.

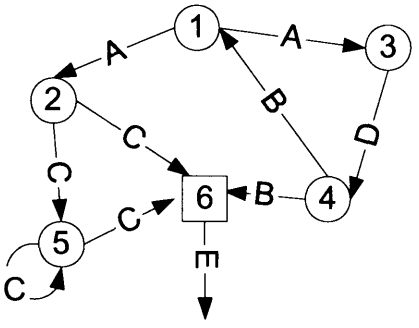


Figure 1.3.1: An example GRN with 5 regulatory genes (1-5) and their products (A-D). This network activates Gene 6, which generates a structural protein labelled E.

The GRN model has been used by developmental biologists to explain the observation of homeotic genes. A *homeotic gene* is one which when mutated transforms one complex phenotypic feature into another (Slack, 1991). Two examples are shown in Figure 1.3.2. The left-hand picture shows a normal, or "wild type" Drosophila head. The centre picture shows a head of a Drosophila with a single mutation that has resulted in the transformation of antennae into legs. The right-hand picture shows the head of a Drosophila with two mutations where in addition to the homeotic transformation of the antennae also exhibits legs in place of the proboscis. It has been shown that many homeotic genes are *master control genes* (Lewis, 1992) that when expressed activate a GRN, causing a cascade of activity in the network of genes that eventually leads to the formation of a complex structure.

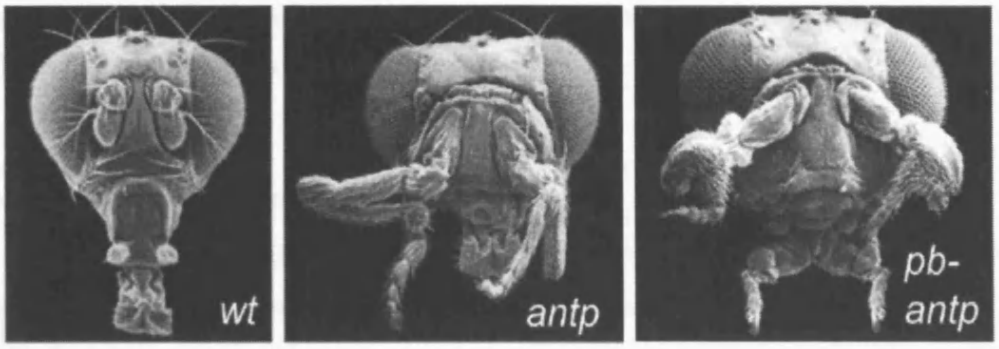


Figure 1.3.2: Three Drosophila heads, (a) wild type, (b) with the *antp* mutation and (c) with both *pb* and *antp* mutations. Micrographs courtesy of F. Rudolph Turner & Flybase (Drysdale et al., 2005).

The existence of homeotic genes in nature demonstrate that GRNs provide evolution a mechanism for encoding *modules* that perform complex functions through the activation of a single gene. Furthermore evolution is free to augment a GRN that encodes such a module with feedback loops, allowing the module to be *re-used* in an iterative or recursive manner.

Once evolution has discovered a useful module encoded as a GRN it can vary either the GRN or the expression of the master control gene to move directly between areas of search space that were previously separated by large distances, avoiding the potentially less fruitful space in between. Hence a GRN can be thought of as a mechanism for effectively transforming or *abstracting* the design space so that evolution is presented with a more tractable landscape, a smaller landscape, or perhaps both. Crucially evolution will have discovered any modules it uses in this way whilst working at a lower level of abstraction. There it will have been able to identify and exploit potentially useful innovations and incorporate them into the module.

There is nothing preventing evolution from incorporating the control genes for one GRN within another GRN, thus development is capable of building hierarchies of modules, unlike methods such as Function Level Evolution. Furthermore such modules are discovered in a manner quite different from the top-down design process common to traditional engineering. Instead they are created by combining components incrementally using evolution rather than domain knowledge to guide the search, unlike some methods that have been suggested for providing modularity in hardware evolution. In addition evolution is not constrained to use a design abstraction encapsulated by a module. Rather it is merely *biased* towards searching the space using the new design abstraction. It is still free to modify, or even disregard the module should it prove fruitful to do so.

The notion that the regulatory processes described by GRNs can be used to encapsulate useful design features within *modules* and provide a mechanism for their *reuse*, thus presenting a *design abstraction* is central to the argument made in this thesis as to how development can

enhance the scalability of hardware evolution, and central to the models of development used in this thesis to demonstrate scalability.

1.3.3 Intercellular Communication

The GRN model can explain how development can control and iterate the generation of a complex feature, but it cannot explain how development can ensure the positioning of a complex feature at a particular site in a large organism or iterate the generation of a feature over many specific sites within an organism. To do so development must communicate information that activates a GRN across space. The most common developmental communication mechanisms in large, multicellular organisms rely on the process of *induction*. Here a cell encodes regulatory information as a chemical signal, which is transmitted to nearby cells. A variety of inductive signal types have been identified (Slack, 1991), which pass information over various localities and at various rates. A thorough exploration of detailed biochemical models are of interest to developmental and molecular biologists, and artificial life researchers amongst others. The developmental models used in this thesis model intercellular communication and in particular the process of induction, in addition to the regulatory processes embodied by GRNs. However to maintain clarity the experiments and analysis in this thesis have been limited to focus on the engineering problem at hand. Hence the communication models abstract any biochemical details of both cells and induction. Only features that either aid the implementation of developmental models in hardware or aid the generation of circuit designs are modelled. The same is true of the models of regulation.

1.3.4 Problems Suited to Development

Not all circuit design problems might benefit from a developmental approach. An ideal problem would be one where an optimal circuit description is easily partitioned into a number of repeated modules, each contributing independently to the circuit's fitness. It should be possible to represent each unique module in a compact form similar to a GRN, and iterate the module across space using some form of intercellular communication, thus generate the final circuit design. Hence problems where solutions can take the form of large modular repetitive circuits that can be represented by a simple developmental process are the ideal target for development. Note that such circuits are not necessarily complex in the sense of information theory. In fact it is circuit designs that appear complex as they cannot be represented succinctly using conventional circuit components but can be described succinctly using some other representation that are of most interest.

1.4 Motivation and Hypothesis

The motivation for this work is two-fold. First it aims to show that the introduction of a developmental genotype-phenotype map can allow evolution to design larger electronic circuits automatically, leaving open the possibility that one day the size and complexity of evolved circuits might rival those generated by conventional design techniques. The second aim of this work is to show that circuits can be evolved using development without preventing evolution from exploring innovative and evolvable design features found only at the lowest design abstractions. In achieving these aims, several features of biological development that are of benefit to the evolution of such circuits are identified. This work does not attempt to model biological development accurately, but merely to model it at a level of detail necessary to achieve the engineering goal of generating the circuits described above.

From the above background the hypothesis of this work can now be plainly stated:

The scalability of hardware evolution can be enhanced by exploiting a model of biological development.

Where: scalability is defined as the ability to solve a series of problems of increasing size where the larger problems can be decomposed into modules of the smaller problems

And: performance is measured as the expectation that evolution will provide fully compliant solutions to the problems.

In section 1.3 it was explained that one mechanism though which development could enhance scalability relies on three properties: abstraction, modularity and reuse. This thesis aims to support the hypothesis by highlighting these properties through the following three objectives:

1. *Abstraction:* show that a developmental model for hardware evolution can discover and encode biases towards design abstractions that might benefit hardware evolution (Chapter 4 and Chapter 6),

2. *Modularity:* show that evolution can encapsulate such design abstractions in useful developmental modules (Chapter 4 and Chapter 6), and

3. *Reuse:* show that evolution can re-use such modules to solve larger hardware evolution problems (Chapter 5 and Chapter 6).

This thesis aims to demonstrate the hypothesis itself through a series of experiments presented in Chapter 6 that compare the evolution of useful electronic circuits using a developmental mapping and a traditional 1:1 gene to component mapping.

In the course of demonstrating the hypothesis, the thesis also aims to achieve the following minor objectives:

4. Demonstrate hardware evolution of the Virtex architecture (Xilinx Inc., 2001) at a level of abstraction that imposes the fewest design constraints that has been reported in the literature.

5. Show that such an approach allows more efficient use of the resources provided by the Virtex architecture than previously reported work, and may provide a more evolvable search space than those constrained to combinational circuits.

6. Show that the choice of context is important to the ability of induction-based models of development to evolve hardware.

7. Show that a dynamic representation can increase the evolvability of rule-based developmental systems.

8. Show that it is possible to characterise the learning biases introduced by a developmental process to the extent that classes of problem that might and might not be suitable targets for a developmental approach can be identified.

9. Show that the model of development used for the scalability experiments in this thesis performs well in comparison with similar models introduced by other researchers.

10. Show that providing a bias towards traditional problem decomposition, as proposed by other approaches to scalability can actually harm scalability.

During the course of the scalability experiments large evolved circuits will be presented, including adders that are larger than any that have previously been presented using evolution alone (i.e. without relying on additional traditional problem decomposition techniques) and parity generators of similar size to the largest found in the literature.

Even so these circuits are not of a size that can challenge traditional design methodologies. Furthermore for hardware evolution to supplant conventional designers additional questions

such as how to ensure that the behaviour of a large evolved circuit generalises to unseen conditions and can be easily verified must also be answered. Unfortunately there are no immediate answers to such questions. However there is still value in enhancing the scalability of hardware evolution as it might allow the reach of the technique to be extended, to a point where it can tackle a set of moderately sized yet difficult problems where traditional design fares poorly, and evolution's ability to innovate is most needed. At that point hardware evolution might find a niche alongside traditional design.

1.5 Outline

The rest of the thesis is structured as follows. Chapter 2 reviews the fields of hardware evolution and developmental evolution in order to set the rest of the work presented here in context. Chapter 3 introduces an intrinsic circuit evolution platform that was developed to explore the hypothesis stated in Section 1.4, and presents experiments used to validate the platform. The final validation experiment is also used to demonstrate objectives 4 and 5. Chapter 4 presents an exploratory developmental hardware evolution system, along with analysis of circuits evolved with it that demonstrates objectives 1 and 2. Chapter 5 introduces a revised developmental hardware evolution system that incorporates a number of enhancements, and demonstrates objective 6. It also explores the concept of evolvability and presents experiments that demonstrate objective 7. Chapter 6 focuses on the evolution of large phenotypes. It begins by exploring the inductive biases inherent in the revised model of development, leading to a set of guidelines that demonstrate objective 8. This is followed by a comparison with similar models used by other researchers across a number of benchmark problems, the results of which demonstrate objective 9. The crux of the thesis follows: the primary hypothesis of the thesis is clearly demonstrated through a series of experiments that show scalability is enhanced when a developmental model is used in place of a traditional 1:1 genotype-phenotype map. Analysis of some of the large evolved circuits confirms objective 3. Chapter 6 also demonstrates objective 10 by repeating the adder scalability experiments but with an additional bias towards traditional problem decomposition. Chapter 7 reviews the objectives laid out here, introduces ideas for further work and presents conclusions.

2 Literature Review

This chapter reviews the two fields of research directly related to the work of this thesis. In Section 2.1 the field of hardware evolution is reviewed. Particular emphasis is given to the scalability issue, the motivation for this thesis. Section 2.2 reviews the use of genotype-phenotype mappings that model development in Evolutionary Computation, focussing on their benefits to scalability and their use in hardware evolution.

2.1 Hardware Evolution

Hardware evolution is the application of evolutionary algorithms to the automatic design and synthesis of electronic circuits. The field draws inspiration from several other fields, as shown in Figure 2.1.1. For many years computer scientists have modelled their learning algorithms on self-organising processes observed in nature. Perhaps the most well known example is the artificial neural network (ANN) (Widrow and Hoff, 1960; Rumelhart et al., 1988). Others include the collective decision-making of ant colonies (Dorigo et al., 1991), the adaptive ability of immune systems (Dasgupta, 1996), the growth of self-similar structures in plants (Lindenmayer, 1968), and Darwinian evolution (Goldberg, 1989). Collectively, work on such algorithms is known as bio-inspired software, which is shown at the intersection of Computer Science and Biology in Figure 2.1.1.

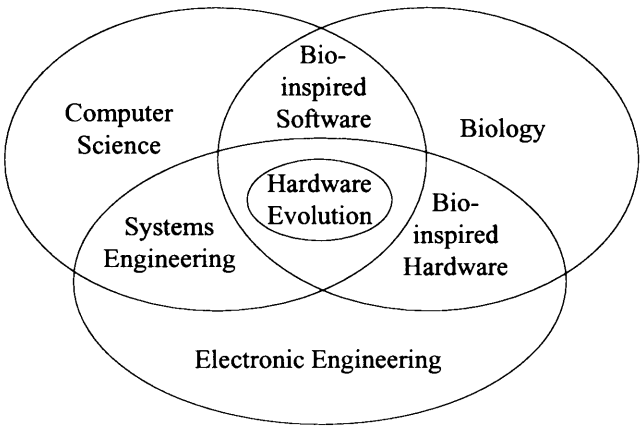


Figure 2.1.1: The elements of hardware evolution.

Nature has also inspired algorithms used in Electronic Engineering. For instance simulated annealing algorithms are used in many circuit partitioning, placement and routing algorithms (Sechen, 1988; Gerez, 1999). (These are inspired by the physical phenomenon of annealing as metals cool.) Interest in using ideas from nature has grown in recent years to the extent that the field of bio-inspired hardware is now firmly established in its own right, as shown at the intersection of Electronic Engineering and Biology in Figure 2.1.1. This field uses many of the

ideas adopted from nature by software developers, and some new ones, not only to improve features such as fault tolerance and reconfigurability but also for automatic circuit design. The field of hardware evolution is shown at the intersection of Computer Science, Biology and Electronic Engineering in Figure 2.1.1. The first half of this chapter reviews work in this area.

Hardware evolution encompasses the application of evolutionary algorithms to both analogue and digital circuit design and synthesis. However as digital methodologies are generally more structured than analogue methodologies it is possible to clarify the relationships between traditional digital hardware synthesis, Evolutionary Computation and hardware evolution in a little more detail, as shown in Figure 2.1.2.

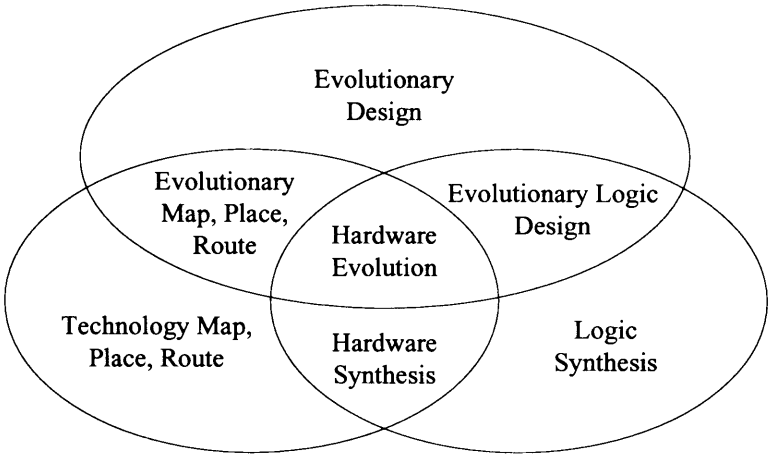


Figure 2.1.2: The interrelationships between hardware design and synthesis, and Evolutionary Computation.

Traditional digital hardware synthesis is a combination of two processes. First a human-designed circuit specification is mapped to a logical representation, normally a list of components and their interconnections or *netlist*, through the process of logic synthesis. This is represented as the lower right-hand set in Figure 2.1.2. The netlist then undergoes further combinatorially complex optimisation processes in order to place and route the circuit to the target technology. This area is represented as the lower left-hand set in Figure 2.1.2. Many modern EDA[1] tools apply intelligent techniques to these optimisation algorithms, and there is much research into use of evolution for these purposes (Göckel et al., 1997; Mazumder and Rudnick, 1999). Hence the set representing evolutionary design intersects with that of technology mapping, placement and routing in Figure 2.1.2 to yield evolutionary mapping, placement and routing. However circuit design, along with some optimisation decisions during the synthesis process is still in the domain of the human designer. It is only recently that significant interest has developed in implementing evolutionary techniques higher up the digital design flow for circuit design, a move that can allow evolution to generate creative designs that

[1] Electronic Design Automation

can rival or improve on human ones. Most commonly evolution has been used to design logic, as represented by the intersection of the areas of evolutionary design and logic synthesis in Figure 2.1.2. Some of the work in this intersection falls into the field of hardware evolution. However much work at the logical level is carried out in the spirit of evolving programs or other forms of logic, so is beyond the scope of this thesis.

The rest of this section is organised as follows. First Section 2.1.1 surveys applications that have made use of hardware evolution to date. Section 2.1.2 discusses features that can be used to classify work in the field. Then Section 2.1.3, 2.1.4 and 2.1.5 discuss key hardware evolution research. Section 2.1.3 deals with what in the author's view is the most important benefit of hardware evolution: *innovation*. The greatest and most actively researched problems that face the field and impede the use of hardware evolution as a real-world engineering tool are then discussed: *generalisation* (Section 2.1.4) and *evolvability*, which encompasses performance issues such as solution quality, scalability and algorithm speed (Section 2.1.5).

2.1.1 Applications of Hardware Evolution

Using evolution to design hardware brings a number of important benefits to electronics. Some of the more important areas where hardware evolution can be applied are:

- Automatic design of low cost hardware;
- Coping with poorly specified problems;
- Creation of adaptive systems;
- Creation of fault tolerant systems and
- Creation of innovative hardware

The remainder of this section will explore research in these areas in a little more detail.

Automatic Design of Low Cost Hardware

Various aspects of conventional circuit synthesis are commonly automated. Conventional digital synthesis involves mapping a circuit that has been designed in some abstract design space to a specific technology through the application of minimisation, placement and routing rules. As the technologies for synthesising more complex circuits have developed there has been an increasing need for better optimisation techniques, capable of handling the combinatorial explosion in the complexity of these mapping steps. For some time now intelligent techniques such as simulated annealing (Sechen, 1988) and ANNs (Yih and Mazumder, 1990) have been used to guide these mapping processes. More recently so has evolution (Göckel et al., 1997; Mazumder and Rudnick, 1999).

Such research is interesting and useful. However hardware evolution allows the automation of circuit production to be taken a step further, automating not only the synthesis stage but also the generation of the *circuit design* from a behavioural specification.

When a suitable behavioural specification is available hardware evolution might be able to replace the designer, or at least reduce the design time that is required, thus reduce production costs. This is particularly useful when design costs are a significant proportion of total costs, for instance when hardware is produced in low volumes using reconfigurable devices. Hardware evolution might even make one-off designs viable. One area where one-off designs might be useful is for medical applications. Already hardware evolution has been used to train a prosthetic hand controller to recognise the myoelectric signals of a specific individual that correspond to various motor actions (Kajitani et al., 1999). The solution was implemented entirely in hardware, hence was not only faster than a software implementation but also small and light, thus portable.

Evolution can also be used to reduce production costs on larger scales by optimising circuits that fail to meet their required specifications due to fabrication variations. For instance in (Murakawa et al., 1998) variations in the frequency of intermediate frequency (IF) filters were corrected using evolution to control transconductance amplifiers that regulate the supply current at various points within the circuit. IF filters tuned using this technique are already used in commercial products (Murakawa et al., 2002). This is particularly useful for analogue integrated circuit implementation, where component behaviours can vary quite widely, and tuning the components can allow the use of smaller, low power components. However evolution has also been used to counteract fabrication variations in digital circuits. Takahashi, Kasai et al. incorporated programmable delay elements in the registers of a memory test pattern generator (Takahashi et al., 2003), which allowed the evolved circuits to compensate for both clock skew and variations in data delays throughout the circuit. Simulation results demonstrated an almost 50% improvement in production yield.

Poorly Specified Problems

For some problems it is difficult to specify how the solution should function, but easy to specify a limited behavioural description of the problem, for instance noisy classification problems. Although evolution has been used to handle problems with poor specifications it is more common to use ANNs (Rumelhart et al., 1994). Yao and Higuchi (Yao and Higuchi, 1996) noted that evolved hardware has similarities with ANNs: both are often arranged as feed-forward networks of components, and both can learn non-linear functions. They also suggested evolved hardware has some advantages over ANNs. First they suggested that evolved hardware provides a performance advantage over ANN solutions as ANNs are most commonly implemented in software. However ANNs can be implemented in hardware as easily as an

evolutionary algorithm, so this comparison is somewhat unfair. They also made a more interesting point: when restricted to feed-forward networks (as is common), evolved digital hardware is usually more easily analysed and validated than an ANN. Therefore hardware evolution is suited to problems usually tackled with ANNs that require fast operation and reliable solutions.

Evolved hardware has been shown to rival ANNs for many pattern recognition problems. The prosthetic hand controller discussed above is a case in point. Other examples include character recognition (Iwata et al., 1996) and image recognition (Torresen, 2000b). Sekanina has evolved image noise filters that rival the best traditionally-designed circuits (Sekanina, 2003b). One of the advantages of evolutionary systems is the ease with which additional learning biases can be incorporated through the fitness function. For instance Higuchi et al. have evolved high-speed robust classifiers (Higuchi et al., 1995; Iwata et al., 1996) with good generalisation characteristics by specifying an additional bias based on machine learning theory. More recently do Amaral et al. evolved fuzzy functions that can be used as building blocks in the construction of fuzzy logic controllers (do Amaral et al., 2002).

Adaptive systems

Reconfigurable devices allow circuits to be synthesised quickly. Hence circuits implemented with reconfigurable devices can be adapted by evolution to changes in incoming data in real-time. There are many applications for such hardware, generally when real-time manual control over systems is not possible, for instance compression algorithms or real-time scheduling. Another exciting area of this kind is hardware for deep space exploration, where unexpected data that must be handled autonomously can be encountered.

Stoica et al. have noted that current lack of validation for on-line evolutionary systems mean that critical spacecraft control systems, and other mission-critical systems, cannot currently be placed under evolutionary control (Stoica et al., 1998). Greenwood and Song have proposed using evolutionary techniques in conjunction with formal verification techniques to circumvent this problem (Greenwood and Song, 2002), however to date only non-critical systems such as sensor processing systems have been explored, for example adaptive data compression systems (Fukunaga and Stechert, 1998).

Systems could also benefit from the ability to autonomously evolve to unexpected or unknown environmental conditions. For instance power management systems could be optimised depending on not only the nature of the data processed but also the power available. Similarly controllers for the deployment of booms, antennas and other moving parts could be evolved to avoid unexpected obstacles (Plante et al., 2003).

Hardware evolution-based adaptive data compression systems have also been developed for use closer to home. The image compression system developed by Sakanashi et al. used a standard prediction function to predict each pixel from a subset of surrounding pixels that are selected by a genetic algorithm (Sakanashi et al., 2001). It outperformed JBIG, the ISO standard for bi-level image compression by an average of around 50%. Since then the method has been proposed as a new ISO standard. Another approach is to break images into smaller sections and use evolution to model a function for each section (Salami et al., 1996). A similar technique has been used by Sekanina to evolve adaptive circuits that filter image noise in changing environments (Sekanina, 2002b).

Evolution has been used to evolve many other kinds of filter, including digital finite impulse response (FIR) filters, commonly used in audio applications such as noise and echo cancellation (Tufte and Haddow, 2000; Vinger and Torresen, 2003) and their more flexible but less reliable counterparts, infinite impulse response (IIR) filters (Sundaralingam and Sharman, 1998; White and Flockton, 2003). Analogue adaptive filters have also been evolved. For example Zebulum et al. presented signal extraction filters capable of adaptively amplifying the strongest component of the input signal whilst attenuating others, which could be used to improve signal/noise ratio (Zebulum et al., 2003). Furthermore through evolution these circuits could be adapted to new input profiles.

Other on-line hardware includes adaptive cell scheduling systems for ATM networks (Liu et al., 1997; Li and Lim, 2003). and on-line adaptive hashing systems that could be used to map cache blocks to tags dependent on access patterns (Damiani et al., 2000).

Fault Tolerant Systems

Ongoing advances in component miniaturisation have not been complemented by improvements in fabrication reliability. This means that many modern integrated circuit designs must be tolerant to fabrication faults. It is expected that this will become even more of an issue in future circuit technologies. Miniaturisation also exposes components to a greater risk of operational faults, which may arise from power fluctuations or ionising radiation. Reliability is of paramount importance for hardware such as medical equipment and transport control systems. Military and spacecraft systems are particularly susceptible to reliability problems as they are regularly subjected to harsh conditions. Most current techniques for fault tolerance rely on the presence of additional redundant components and thorough testing either at the point of manufacture or on-line, and add considerable cost and design complexity. Fortunately hardware evolution provides a number of mechanisms to introduce fault tolerance into circuits.

A class of adaptive system that was not mentioned in the previous section are circuits that can adapt to faults in their own hardware, thus providing a mechanism of fault recovery. An early

demonstration of such a system was given by Higuchi et al. (Higuchi et al., 1995), where an adaptive hardware system learned the behaviour of an expert robot controller by example using a genetic algorithm.

More recently Vigander demonstrated that a simple evolutionary system could restore most but not all functionality to a four bit multiplier that had been subjected to random faults (Vigander, 2001). Complete functionality could be restored by applying a voting system to select between several alternative circuits that had been repaired by evolution. Sinohara et al. used a multi-objective evolutionary algorithm that allowed essential functionality to be restored at the expense of secondary behaviour that was not deemed as important by the designer, such as power dissipation (Sinohara et al., 2001). This was demonstrated through the repair of NOR gates and inverters.

Hounsell and Arlsan have explored the repair of an evolved FIR filter after the injection of multiple faults (Hounsell and Arslan, 2001). They examined two different recovery methods. The first was to recall the final population of the evolutionary run that created the original filter design, and the second was to seed a new random population with a copy of the original design. Both mechanisms recovered functionality faster than re-running evolution with a completely random population, with population seeding outperforming population recall by a small margin.

Zebulum et al. demonstrated evolutionary recovery with a four bit digital-analogue converter (DAC) that had initially been evolved using traditionally-designed operational amplifiers and smaller DACs evolved in earlier experiments as building blocks. Faults were introduced into one of the operational amplifiers. The recovered circuit outperformed the circuit that had initially been evolved. Gwaltney and Ferguson investigated fault recovery in an evolved analogue motor controller using a similar method (Gwaltney and Ferguson, 2003), finding that evolution could recover from faults in some components better than others.

Most evolutionary fault recovery systems that have been demonstrated to date have only explored recovery from errors introduced into the components of the circuit. Lohn et al. demonstrated an evolutionary fault recovery system that can repair routing in addition to components (Lohn et al., 2003), which they suggest is important for modern routing-rich reconfigurable devices.

Another type of fault is the failure of a component at extreme temperatures. Stoica et al. have observed that multipliers, Gaussian curve generators and logic gates evolved under standard conditions degrade or fail at extreme temperatures (Stoica et al., 2001). However when re-evolved at those temperatures, the circuits regained functionality in all cases.

The discussion above has only dealt with fault *recovery*. Fault recovery is of little use without some means of fault *detection*. Fault detection is traditionally dealt with by incorporating additional hardware into a design to perform a built-in self test (BIST). Garvie and Thompson have demonstrated that evolution can be used to design small adders and multipliers that incorporate BIST at very low additional cost by sharing components between BIST and the circuit function (Garvie and Thompson, 2003). Such circuits are discussed in the context of innovative circuit designs in the following section.

A number of other bio-inspired on-line autonomous hardware fault tolerance mechanisms have been developed for both fault detection (Bradley and Tyrrell, 2001) and recovery (Macias and Durbeck, 2002; Tyrrell et al., 2003). Many of these have been proposed as a platform for evolutionary experiments, and some will be discussed later. However as they do not use evolution as an adaptive repair mechanism they will not be considered further at this point.

Fault *tolerance* is often used to describe systems that are inherently tolerant to faults rather than those that explicitly detect and/or recover from faults. Evolution has proved an ideal candidate for the exploration of fault tolerant systems. Work in this area is reviewed in the discussion of generalisation in Section 2.1.4.

Innovative Hardware

Traditional circuit designers tend to work on a problem from the top down, decomposing the problem in to smaller sub-problems that have limited interactions. They repeat the process until only a number of small problems remain, which are well understood and have known solutions. Each decomposition directs the process towards these solutions carefully using formal design rules. Evolution works differently. It works from the bottom up, combining components to make partial solutions to the design problem incrementally, until the solution meets the design criteria. This idea is discussed more fully in Section 2.1.3. For now the discussion shall be limited to when this feature might be useful.

The clearest cases for application are design spaces where domain knowledge about how components interact is limited, and so formal design rules have not yet been developed. New kinds of circuit technologies are beginning to emerge through advances in Electronic Engineering. Some radical technologies have design spaces that are very poorly understood. In these cases evolution might prove a useful technique in searching for innovative designs, as it can be guided purely by the behaviour of the evolving circuit (and the inductive biases of the evolutionary algorithm, which are discussed shortly) rather than relying on domain knowledge. An example of this is the field of nanoelectronics, where Thompson and Wasshuber have evolved innovative, although not particularly useful single electron NOR gates (Thompson and

Wasshuber, 2000).

There is a set of current digital technologies that traditional synthesis techniques are not ideally suited to, but are becoming increasingly important for circuit designers: programmable logic technologies. Many of these provide resources such as XOR gates and multiplexers, but popular logic minimisation techniques are best suited to generating solutions that do not map well to these elements. An evolutionary approach can work at a more suitable design abstraction and has the potential to search areas of space that a traditional designer would not explore thoroughly using conventional techniques. This can lead to the discovery of more parsimonious solutions, as was demonstrated by Miller et al. (Miller et al., 2000a).

There are other design spaces where the interactions between components are so complex that it has not been possible to develop formal methods to partition and decompose the design space. For instance, when compared to digital logic design, analogue design is much less well structured. Hence circuit design in this domain requires more expert knowledge. Evolutionary algorithms have proved very successful in discovering human-competitive (and better) analogue circuit designs (Koza et al., 2000; Aggarwal, 2003).

Perhaps the most successful application of evolution to complex design spaces has been the automatic design of antennas. Traditional antenna designs are based on a handful of well-known, regular topologies. The design space beyond these topologies is difficult to explore thoroughly because the interactions between the elements of the design become too complex to abstract. Linden has demonstrated that evolution is capable of discovering many highly unconventional, irregular antenna designs (Linden and Altshuler, 1999) and has shown that evolved antennas operate effectively in real-world settings using transmission of real data (Linden, 2002b), even where the signal path is obstructed (Linden, 2002a). As evolution has proved to perform well when applied to antenna design an evolved antenna has undergone flight qualification testing for NASA's Space Technology 5 mission and is due to be the first evolved hardware in space (Lohn et al., 2004).

Evolution can also be used to discover innovative solutions in design spaces that are generally considered well-understood. This is possible because evolution can search different areas of design space than traditional design techniques. If useful circuits lie in these areas, evolution might be able to discover them. This demands that evolution is allowed to work without the traditional constraints placed on circuit designs, and was first demonstrated by Thompson (Thompson, 1995a). Currently it has not yet yielded solutions to any significant real-world applications, but the concept has prompted a great deal of research and is discussed in Section 2.1.3.

Current evolutionary techniques only work well for small problems, as the search spaces can become vast for large circuits. However evolution can still be used to find small yet innovative designs evolved specifically to produce limited interactions and so can be used by traditional designers as building blocks for larger circuits. Such building blocks have been found for both analogue and digital designs (Miller et al., 1999; Aggarwal, 2003). This approach has also been advocated for use in more abstract design spaces (Sekanina, 2003a) where it was suggested that evolved or evolvable IP[2] cores could be provided for commercial use in reconfigurable devices. It has also been suggested that previously evolved building blocks may help evolution discover larger circuits (Zebulum et al., 2003).

Finally evolution has proved useful for the generation of circuits that incorporate several functions within one set of shared components, a task for which there is little domain knowledge. An example of this was described in the discussion of fault tolerant systems above, where adders and multipliers were evolved to incorporate a BIST function. Circuits designed to execute multiple functions, or achieve other multiple objectives efficiently are difficult to design using traditional techniques hence this area seems ripe for the application of evolution. A related idea is that of polymorphic electronics (Stoica et al., 2002) where a circuit is evolved to perform multiple functions using a shared set of components, with each function becoming apparent under different environmental conditions. For example a circuit might perform as an AND gate at one temperature and an OR gate at another. Such circuits might prove very useful for military and intelligence purposes. The technique has also recently been applied to the evolution of networks of gates (Sekanina, 2005).

Design innovation is in the author's view the most significant area of application for hardware evolution, and research in this area is discussed in detail in Section 2.1.3.

2.1.2 Concepts for Classifying Hardware Evolution

Having discussed the benefits of hardware evolution and some of the applications that these benefits allow, the remainder of this review of hardware evolution focuses on the main thrusts of research in the field, decomposed into three areas: innovation, generalisation and evolvability. Before this work is reviewed it is useful to introduce ideas and terms that have been useful for classifying work in hardware evolution, and that are used throughout the rest of this thesis.

---

[2] Intellectual Property

A number of researchers have developed simple classification schemes for hardware evolution (Hirst, 1996; Zebulum et al., 1996; Andersen, 1998; Torresen, 2000a). Two themes common to these schemes are to classify by *hardware abstraction* and *evaluation strategy*, although the class boundaries drawn by these reviews have varied. It will be seen in the following sections that the level of abstraction at which hardware evolution operates can have a profound effect on innovation, generalisation and evolvability. The choice of evaluation strategy can place significant constraints on the level of abstraction that evolution works at, but this choice must often be made for practical reasons. Both the practicalities of evolving hardware and the constraints that arise as a consequence of evaluation strategy are important, and are worth discussing in more detail.

Level of Abstraction

Traditional hardware design relies heavily on the concept of abstraction. An abstraction is usually selected by a designer to suit the problem, and constraints are imposed on the hardware implementation to ensure that its behaviour meets that of the abstraction. Typically greater levels of abstraction are required to simplify larger problems to manageable levels, often at the expense of the efficiency of the final design.

Analogously the level of abstraction used in hardware evolution plays an important role both in evolution's performance and the nature of the evolved hardware. Hirst was the first to classify work in the field using levels of abstraction (Hirst, 1996). He identified a number of stages within the traditional design and synthesis lifecycle of reconfigurable digital hardware where the problem representation could be mapped to the genotype of an evolutionary algorithm. Torresen has also partitioned work using levels of abstraction (Torresen, 2000b), this time into the digital and analogue paradigms of traditional hardware design. Anderson further divided Hirst's 'netlist' level into networks of logical gates and networks of units that perform more complex functional units (Andersen, 1998), which emphasised the focus of research at the time well. The models of abstraction used in these reviews suggest that the space of possible abstractions can be ordered in a single dimension, from the least abstract to the most abstract, according to traditional design methodologies. As it happens this is a useful way of visualising current work in the field and will be used later on in this discussion. However the space of abstractions that could be used for hardware evolution is in fact more complex. This can be seen by considering what constraints are imposed on hardware to ensure that traditional design abstractions hold.

Thompson et al. noted that these constraints can be grouped into spatial and temporal constraints (Thompson et al., 1995). Traditional design methodologies use temporal constraints to partly realise digital abstractions. The most common constraint is to restrict communication between units using a global clock to ensure that any transient non-digital activity is ignored. This approach is known as synchronous digital design. A less restrictive (and less common)

constraint used to achieve the same goal is to restrict communication to between units using point-to-point handshaking protocols. This is known as asynchronous digital design. Traditional analogue design does not impose explicit restrictions on the timing of communication between circuit elements. These three approaches can be seen as different points on a continuum of all possible temporal constraints that makes up one axis of hardware abstraction space, as shown in Figure 2.1.3.

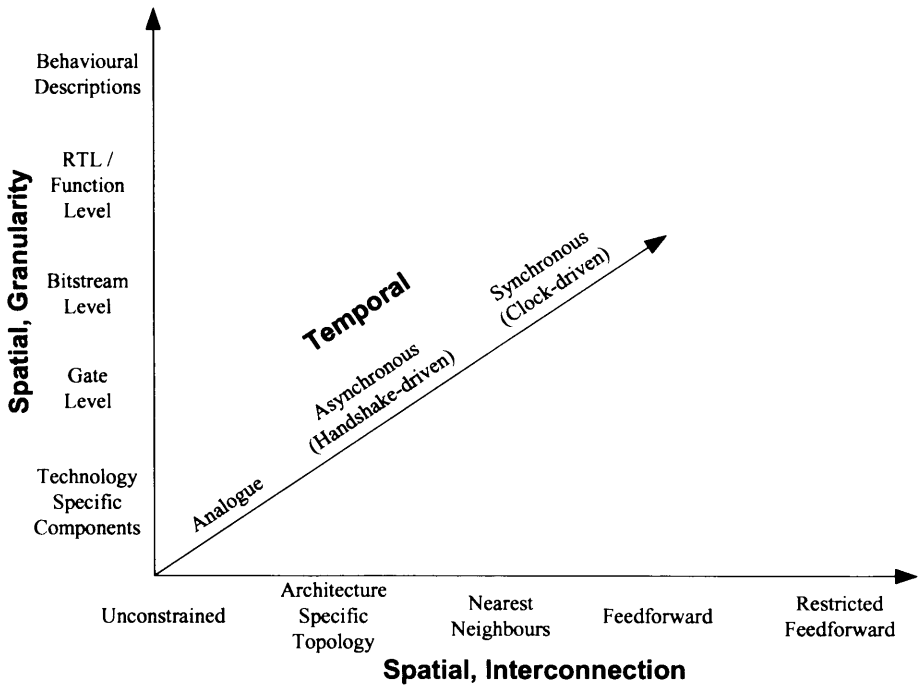


Figure 2.1.3: The hardware abstraction space defined by constraint.

Traditional design methodologies usually also impose spatial constraints, through both the granularity of the computational units and the interconnections between them, thus there are two additional dimensions to the hardware abstraction space of Figure 2.1.3. Most conventional design methodologies encapsulate collections of simple units into more complex units that can be reused without the need to understand their internal structure. For instance although digital hardware is usually realised using networks of transistors, digital designers rarely work directly with these components. Instead they limit their designs to use only a few different configurations of transistors, encapsulated and represented as logic gates. Likewise analogue designers often encapsulate basic components into amplifiers and their derivatives: integrators, differentiators and summing amplifiers. This level of granularity is represented as "Gate Level" in Figure 2.1.3. In turn specific configurations of digital gates are commonly encapsulated as units such as adders, multipliers or registers. This is represented as the "RT / Function level" (where RT stands for Register Transfer) in Figure 2.1.3. At the most abstract level a circuit can be described behaviourally using a hardware description language, thus abstracting all structure from the design problem. Hypothetically the axis of granularity could descend to level of fundamental particles. Given enough computational power and an accurate model of how such particles interact it would be possible to implement a system to evolve designs at this level, but

in reality there are no means to construct such circuits. (Furthermore other issues that are discussed later, scalability and generalisation, would likely preclude useful evolution.) For practical purposes spatial granularity is limited by the technology used to implement a circuit. For designs that are to be implemented as a conventional integrated circuit this limit might be at the level of transistors. Likewise analogue designers might construct circuits from components such as resistors, inductors, capacitors and transistors. This level is represented in Figure 2.1.3 as "Technology Specific Components". For reconfigurable logic this practical limit of granularity varies quite widely. For devices where the manufacturers have disclosed full details of the configuration bitstream format, such as Xilinx 6200 FPGAs (Xilinx Inc., 1997), it is limited only by the level of reconfigurability designed into the architecture. However (and particularly for SRAM-based devices) manufacturers rarely make public the full details of the architecture or the bitstream format, hence granularity is limited by the tools provided by the manufacturer to manipulate the bitstream safely. To highlight this issue, this level of granularity has been included in Figure 2.1.3 as "Bitstream Level".

In addition to constraints on granularity, traditional methodologies often impose spatial constraints by restricting circuit connectivity. The most common constraint of this type is for digital designers to restrict certain areas of logic to a feed-forward topology, which is shown in Figure 2.1.3. Hardware implementations of logic gates exhibit continuous-time dynamical behaviour. Enforcing a feedforward topology avoids continuous recurrent connections between logic gates, thus ensures that any dynamical behaviour the gates exhibit will subside in a finite length of time. For digital designs the feedforward constraint is often combined with a temporal constraint that ensures that any communication between elements of the circuit that do form feedback loops is restricted to occur only when this time has passed. (This restriction is often achieved by co-ordinating with a global clock.) Some abstractions such as those used in signal processing use an even stricter pipeline-like topology. Such constraints are included in Figure 2.1.3 as "Restricted Feedforward". For integrated circuits, especially reconfigurable devices, connectivity is never wholly unconstrained, as it limited either by the technology used to synthesise an integrated circuit or by the architecture of a reconfigurable device. Such constraints are classed as "Architecture-specific topology" in Figure 2.1.3. The most common constraint of this kind is to restrict communication to neighbouring units only. This is the primary method of routing in some reconfigurable devices (Xilinx Inc., 1997), and has found favour with some hardware evolution researchers as a simple abstraction that can be imposed on top of more versatile hardware (Fogarty et al., 1998; Hollingworth et al., 2000b; Haddow and Tufte, 2001). It is also used in some promising new technologies, such as amorphous computers (Haddow and van-Remortel, 2001). Hence it has been labelled separately in Figure 2.1.3.

The diagram reveals how little of the entire abstraction space is explored by traditional design techniques. Digital designers tend to work in the space towards the far end of each axis, and

points within this general region will later be referred to as *high abstractions*. Conversely analogue designers tend to work at or near where the axes meet. Points lying near this end of the axes will later be referred to as *low abstractions*. The space between these two extremes remains for the most part unexplored. The issues of whether it is either possible or beneficial for evolution to explore areas of design space that are not searched by traditional design methodologies are discussed later in this chapter. For now it need only be noted that with a few notable exceptions, current hardware evolution tends to mirror traditional design with respect to the abstractions used. This means that the vast majority of the work in the field can still be characterised using a single dimension of increasing abstraction, similar to Hirst's synthesis-based levels, rather than using the three axes of constraint shown in Figure 2.1.3. This simplifies visualisation of the work in the field and highlights the general levels of abstraction commonly used by researchers more clearly. Figure 2.1.4 shows a diagram of this kind, along with some examples of work at each level of abstraction.

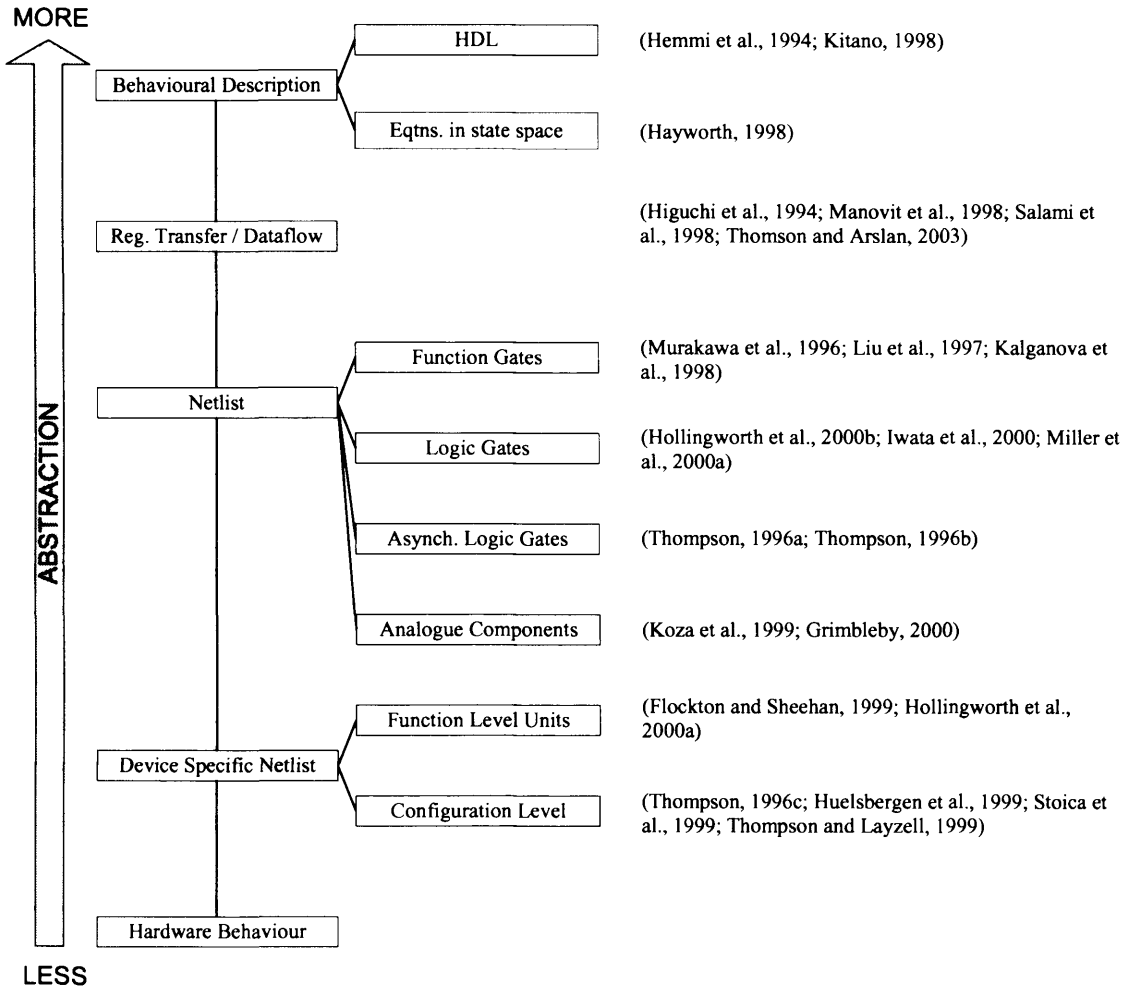


Figure 2.1.4: Common levels of abstraction applied to hardware evolution research.

Hardware Evaluation Strategy

Fitness values for the evaluation step of an evolutionary algorithm must be calculated for each member of the evolving population. Early hardware evolution research achieved this by

evaluating circuit simulations. At that time simulation was the only viable approach because physical hardware could not be constructed within the time-scale needed to use it for the evaluation stage of an evolutionary algorithm. Evaluation through simulation was dubbed *extrinsic* evaluation by de Garis (de Garis, 1994), who also noted that developments in reconfigurable hardware technology could allow solutions to be evaluated quickly using real hardware. This he called *intrinsic* evaluation. Reconfigurable devices suitable for intrinsic evaluation, for example field programmable gate arrays (FPGAs), are now widely available and are reviewed in Appendix A.

Each approach to evaluation has its advantages. Extrinsic evaluation is generally the most versatile way of evolving hardware. Any intrinsic platform can theoretically be modelled by simulation, given that a powerful enough model of the hardware exists. Furthermore the level of abstraction that a simulation represents can easily be altered to introduce or remove domain knowledge from the problem. For example with simulation, placement and routing constraints could be left to be optimised by other means later in the design process during an explicit synthesis step. Simulation also provides an easy, low cost mechanism to experiment with various target technologies. However extrinsic evaluation is often not the most efficient approach. The spatial and temporal constraints used by traditional designers abstract many physical phenomena. Hence low abstraction extrinsic evolution requires such phenomena to be closely modelled, and this is extremely computationally expensive. However useful results have been achieved using this approach, for instance (Koza et al., 1999; Hartmann et al., 2002).

Under different circumstances extrinsic evaluation can be efficient. For example Miller et al. constrained evolution to explore only feedforward combinational logic designs (Miller et al., 2000a). In this case the behaviour of the design could be predicted using an abstract logical model which is particularly suited to execution on microprocessors, and so could be carried out extremely efficiently. In fact in general when evolving circuits at higher design abstractions an extrinsic approach is usually the most sensible, as there is usually little loss of efficiency but much to gain in flexibility. Issues relating to evaluation strategy are considered further when discussing generalisation and innovation in the following sections.

Inductive Bias

The two sections above have discussed abstraction and hardware evaluation strategies, ideas that have been used by other researchers to classify hardware evolution. A more general concept that stems from the field of machine learning and is used throughout this thesis is the idea of *inductive bias*.

Inductive bias is a feature of all learning algorithms. Essentially the inductive bias of a learning algorithm is a set of assumptions that the learner uses to guide its search. Although the idea of

incorporating assumptions into a learning algorithm may sound unwise, it has been shown that it is vital for all learners to exhibit an inductive bias if they are to achieve anything better than learning by rote (Mitchell, 1997). Otherwise there is nothing to guide the search.

Bias can be separated into representational bias and procedural bias, according to a framework introduced by Rendell (Rendell, 1987) and clarified by Gordon and des Jardins (Gordon and des Jardins, 1995). The representational bias of an algorithm is the language that defines the search space within the entire space of possible solutions. For a standard genetic algorithm this is the bias imposed by the choice of genotype and how it maps to the solution space. Traditionally this bias strictly limits the algorithm to search particular areas of solution space hence in this thesis it is termed a *hard* bias, or *constraint*.

The procedural bias determines how the algorithm moves through the search space. For a genetic algorithm this bias is complex and dynamic, as it is defined not only by the choice of genetic operators but also by the information present in the population at any given point. However the stochastic nature of genetic operators means that the procedural bias of a standard genetic algorithm is a *soft* bias: one that is not always strictly observed. Hence a genetic algorithm is merely predisposed to exploring areas of space defined by the bias, but is capable of exploring beyond it.

A good evolutionary algorithm practitioner will select and modify the genetic representation, operators and fitness function so that the inductive bias of the algorithm suits the problem at hand. Such decisions can be shaped in many ways, for instance by knowledge from the problem domain or perhaps inspiration from some aspect of biological evolution. Whatever their basis, these decisions are usually central to achieving acceptable performance with an evolutionary algorithm.

Having introduced the concepts of abstraction, evaluation strategy and bias, the field of hardware evolution is now reviewed in three major threads: innovation, generalisation and evolvability.

2.1.3 Research on Innovation

There are many electronic circuit technologies and over time design methodologies have been developed to create and optimise circuit designs for different classes of technologies. However for some technologies there is little domain knowledge concerning how to conduct an efficient search of the design space. An example of this is the analogue domain, where as the size of the design space grows, the task of managing the interactions between components rapidly become intractable. For such technologies an evolutionary search of the design space might lead to the

discovery of circuits with useful characteristics not found in circuits that can be designed using traditional (and in this case limited) design methodologies. For the purposes of this thesis, such circuits are considered innovative circuits. Research into exploiting evolution to search such design spaces is discussed below under the heading "Complex Technologies".

Another class of technologies where there is a lack of domain knowledge are those that have been recently developed, but require a paradigm shift in the understanding of how a circuit should perform a computation. Examples in this class are radically new technologies such as those that facilitate quantum or amorphous computing, where there is still limited understanding of how individual components interact, hence robust and efficient design rules and the tools that use them have not yet been developed. As mentioned in Chapter 1 new technologies are likely to be crucial to the continued increase of circuit performance in the long-term, hence it is vital that new design rules and design tools are developed to exploit them. Evolution might not only prove an important addition to the designer's toolkit but also help reveal more general design principles, both of which would expedite the use of such technologies in industry. Research in this area is discussed below under the heading "New Technologies".

In Chapter 1 it was also noted that in the medium term there is likely to be an increased need for new and innovative designs based on current technologies as it becomes more difficult to achieve greater performance from them. Fortunately there is also a great deal of scope for the discovery of innovative designs for more familiar technologies where a large body of domain knowledge exists. In the section above it was stated that conventional design techniques do not search the entire space of all possible hardware designs specific to a given technology. For instance digital designers tend to search an abstract and fairly general design space, and implement their design using technology-specific mapping procedures. Conventional digital design spaces at all levels of abstraction, from behavioural to gate level, consist of collections of abstract units, where each unit has a strictly defined interface through which all interactions with other units must be processed. This strict definition of interactions between subunits of the problem ensures that the internal interactions of the design can be understood and managed successfully by the designer during the search, whatever the level of abstraction.

The search for a compliant design is usually dealt with in a top-down manner. At each step the problem is partitioned into a number of smaller sub-problems, each with their own strictly defined interfaces. At each step the problem is restated in terms of the interactions between these sub-problems. Thus each step is carried out at progressively lower levels of abstraction. The search is directed so that eventually a point is reached where the problem is described in terms of familiar sub-problems that can be mapped automatically (perhaps with a handful of additional decisions made by the designer) to a specific technology. In the digital domain the

sub-problems at this lowest level are often arithmetic and logical operators, and memories. The processes used to map to particular technologies also rely on heuristic searches which again often rely on top-down design.

The directed, top-down partitioning of the problem used by traditional design processes as described above means that an exhaustive search of the entire design space at the lowest levels of abstraction is not conducted. Hence there might be an opportunity to discover innovative designs within the space that is not searched by traditional design approaches. Interactions between sub-problems at higher levels of abstraction are fixed before the solutions to the sub-problems are considered hence there may be interesting, innovative solutions involving interactions between subunits that are not considered.

The inductive bias that evolution uses to direct its search through design space is quite different. Evolution begins its search at a low abstraction, and uses the biases inherent in both the genetic operators and the genetic representation to direct its search towards increasingly larger units that improve overall performance, which can be pictured as a bottom-up search. This leads to two important points. First, as the inductive bias employed by an evolutionary algorithm differs from the bias used in traditional design process, evolution searches the design space in a different order. Hence it can potentially search different areas of design space than traditional design methodologies, and this could lead to the discovery of innovative circuits.

One area within the bounds of digital circuit design where evolution has demonstrated its ability to innovate is in logic minimisation. Most logic minimisation processes are directed towards producing descriptions of logic in terms of either strictly Boolean or Reed-Muller operators, which must then be mapped to the target technology. Although this is ideal for technologies that have been designed to implement such operators efficiently, some modern technologies, and specifically some programmable logic architectures, have been designed according to additional criteria. For instance some programmable architectures are based on multiplexer units to improve the versatility of the reconfiguration process (Xilinx Inc., 1997). Other architectures incorporate combinations of Boolean and Reed-Muller gates to allow efficient implementation of both logic and arithmetic functions (Xilinx Inc., 2001). Hence designers that use traditional minimisation rules to generate logic and then implement their design using heuristics to map to a programmable architecture may not discover the most efficient circuits. Research that has demonstrated this issue is discussed in the "Programmable Architectures" section below.

A second important point is that not only is evolution likely to search design space in a different order than traditional approaches, but also evolution might be capable of searching the design space using the inductive biases inherent in its operators and the genetic representation alone:

the constraints that are used by traditional design to ensure that higher abstractions hold and direct the search to useful areas of design space, might not be necessary for or even useful to an evolutionary search. Although the majority of digital hardware evolution work preserves many constraints familiar to traditional designers, hardware evolution researchers are usually expert in traditional design techniques. Perhaps the regular use of traditional abstractions in hardware evolution owes more to the preconceptions researchers hold as to what electronic circuits should resemble than a calculated choice.

If the traditional design constraints imposed at each level of abstraction are reduced or removed, it might be possible to discover innovative circuits within the areas of space that such constraints exclude from the design space. The following section, entitled "Relaxing Constraints" discusses research that explores the evolution of circuits when the majority of these constraints are removed. However the constraints are present for a reason. When they are removed the interactions within the design space become too complex to manipulate effectively and the design space itself becomes too large for a designer to search thoroughly. Hence the following section focuses on two important issues: whether evolution can successfully manipulate the complex interactions between components that arise when such constraints are removed, and whether useful, innovative circuits that rely on such interactions actually exist within the expanded space.

Current approaches to scalability in hardware evolution rely on traditional methods of top-down problem decomposition applied through hard biases. One of the contentions of this thesis is that such approaches limit evolution's ability to innovate for the reasons given above. In Chapter 4 it will be shown that development can allow evolution to create its own abstractions, and in Chapter 6 it will be shown that this allows evolution to discover circuits with structural characteristics quite different from traditional designs. Furthermore it will be shown that the introduction of domain knowledge of how a circuit design problem can be decomposed can actually harm evolutionary scalability.

Relaxing Constraints

Seminal work on the relaxation of design abstractions was carried out by Thompson. The gates used in digital circuits are actually analogue components that saturate quickly. Both temporal and spatial constraints are traditionally applied that ensure the digital abstraction holds. Useful digital circuits tend to exhibit dynamical behaviour several orders of magnitude lower than the dynamics of their constituent components. Thompson set out to show that evolution could successfully manipulate the interactions between such high-speed components when temporal constraints that are traditionally needed to produce such low-speed behaviour from high speed components had been relaxed. To achieve this Thompson et al. evolved a recurrent network of high-speed digital gates at a netlist level of abstraction to behave as a low frequency oscillator

(Thompson et al., 1995). Fitness was measured as an average error based on the sum of the differences between desired and measured transition periods. Circuits were evaluated in simulation using an asynchronous digital model. Hence the search space contained only circuits that used behaviour modelled by the simulator, with the space strictly partitioned at the granularity of logic gates. This might suggest that the majority of traditional spatial and temporal constraints used by digital designers were still in place. However the simulator allowed the gates to interact asynchronously at timescales orders of magnitude faster than the target frequency of the oscillator, and no constraints were placed on the connectivity of the component network. Hence evolution's task was to manipulate the high speed asynchronous dynamics of the model to generate low-speed behaviour.

Circuits that exhibited the behaviour specified by the fitness function were successfully evolved, showing that it is possible for evolution to search without the full complement of temporal constraints (in this case synchronous constraints) usually needed by traditional designers. Analysis of the temporal behaviour of the circuit showed that it relied on phenomena that while known, would not have been used by traditional designers. Further, a graph-partitioning algorithm showed the circuit contained no significant structural modules as would be seen through the successive abstraction of a traditional top-down approach, suggesting that the circuit was both spatially and temporally innovative.

Thompson's experiment had demonstrated that evolution could successfully manipulate the dynamics of high speed components with no aid from traditional temporal constraints in simulation, but it was not clear whether this was possible with true physical components. This was shown in a later experiment (Thompson et al., 1995). The hardware used was a finite state machine (FSM) for a robot controller. As with a traditional FSM the timing of the state transitions could be controlled synchronously by a clock. However this was not a hard constraint, but was under genetic control for each individual register. (The spatial constraints associated with the traditional FSM architecture were not relaxed.) He dubbed this architecture a dynamic state machine (DSM). The evolved robot controller used a mixture of synchronous and asynchronous behaviour, and interacted with the environment in a complex dynamical manner to produce behaviour that would not have been possible using the entire FSM abstraction with such limited resources. Importantly he suggested that the ability of such a parsimonious controller to interact in such a complex manner with its environment was not directly attributable to any special feature of the DSM architecture, but from the ability of evolution to exploit its dynamical behaviour. By changing a traditional hard temporal constraint to a soft bias, Thompson had shown that evolution could find a circuit with the required functionality outside the scope of traditional design techniques, the constraint in this case being the synchrony imposed on a traditional FSM abstraction. In addition evolution had found a circuit

that made use of the rich dynamics that can arise by relaxing design constraints to perform a useful task, demonstrating that such dynamics can give rise to useful behaviour in the real world.

Thompson also carried out the first intrinsic evolution of a circuit evaluated on an FPGA. A 10x10 area of a Xilinx XC6126 bitstream was evolved. Almost all bits in the bitstream corresponding to this area were evolved, directly as the bits of the chromosome of a genetic algorithm (Thompson, 1996a). Thereby Thompson set about evolving a circuit very close to the bitstream level of Figure 2.1.3, the lowest level of abstraction possible with the device in question. This meant that all traditional temporal constraints that could be relaxed had been, as were all spatial constraints of connectivity and granularity, as far as the reconfigurability of architecture would allow. The task was to evolve a circuit to discriminate between 1 kHz and 10 kHz signals. Fitness was calculated by subjecting each circuit to five 500 ms bursts of each signal in a random order, and awarding high fitness to circuits with a large difference between the average voltages of the output during these bursts. The average voltages were measured with an analogue integrator. The only input to the circuit was the 1 kHz / 10 kHz signal; no clock was given, hence the task required that evolution find a continuous-time arrangement of components that discriminated between signals many orders of magnitude longer than the delay afforded by each individual component. The resulting circuit used a fraction of the resources that a traditional designer would need to achieve the same task. Following months of analysis Thompson and Layzell described the functionality of the circuit as 'bizarre' (Thompson and Layzell, 1999) and to date the nature of some of the mechanisms it used have not been determined, although they postulated that the circuit made use of the underlying physics of the substrate in some unknown way. Although the idea of evolution discovering and taking advantage of unknown physical phenomena is a powerful idea, Thompson and Layzell's analysis is the only example of its kind, hence it seems dangerous to draw such a strong conclusion. It is possible that their analysis is incomplete and that the behaviour of the circuit could be explained using more familiar phenomena.

Thompson and Layzell carried out a similar experiment this time providing the circuit with a 6 MHz oscillator signal, which could be used or ignored as evolution required (Thompson and Layzell, 2000). The prime motivation for the experiment was to investigate robustness, and so evaluation was carried out under a range of environmental conditions. This meant that the constraints to the system were the same as before except that a soft bias towards robust behaviour had been added through the fitness function (for reasons that will be discussed under the topic of generalisation later in this chapter). However an additional dynamical resource had been provided. The resulting circuit made use of the clock, and the design was simulated using the PSpice digital simulator. The simulated design behaved exactly as that of the real circuit,

showing that evolution had found a solution within the digital design abstraction of the simulator, even through the constraints did not explicitly require that. However, analysis of the simulation waveforms showed a large number of transient signals. This led them to the conclusion that potentially useful circuits lie within the digital abstraction that are undiscoverable using traditional digital design methodologies owing to their greedy, top-down nature, and that at least some of these circuits *can* be discovered using evolution.

Programmable Logic Architectures

It was noted above that common logic minimisation methodologies are geared towards generating logic descriptions in terms of either Boolean or Reed-Muller logic. However many programmable logic devices support additional components beyond these abstractions, such as multiplexers and lookup tables (LUTs), and often include combinations of Boolean and Reed-Muller algebras.

Miller et al. have conducted research into the discovery of innovative circuits, one of their main motivations being the derivation of new design principles that could be applied to logic abstractions such as those found in programmable logic devices. In (Miller et al., 1999) they noted that Boolean or other algebraic rules can map from a truth table of required circuit behaviour to an expression in terms of that algebra. They then suggested that a bottom-up evolutionary approach can search not just the class of expressions that the algebraic rules map to, but a larger space of logical representations, beyond commonly used algebras. To demonstrate this they successfully evolved one and two bit adders based on the ripple adder principle using a feed-forward netlist representation of AND, OR, NOT, XOR and MUX gates. This space is of interest as these gates are available as resources in many reconfigurable architectures. Many of the circuits reported in this and other work (Miller et al., 1997; Miller et al., 2000a) were unusual but interesting because of their efficiency in terms of gate count. They lay in the space of circuits making use of multiplexers and XOR gates, outside the space of traditional atomic Boolean logic units. They argued that these circuits were unlikely to be found using traditional algebraic methods, and so evolutionary "assemble-and-test" is a useful way that such a space can be explored. The work continued with the evolution of two bit and three bit multipliers. All work was carried out using gate-level logic simulation. Similar work has also been carried out using multiple valued logic (Kalganova et al., 1998).

Chapter 3 of this thesis presents the evolution of binary adder circuits using a reconfigurable architecture that incorporates components such as those described above. There it will be suggested that allowing evolution to freely explore a design space that contains such components might not only allow the generation of more efficient circuits, but that it may benefit the efficiency of the evolutionary search itself.

As mentioned above, the main aim of Miller's work was to show that *design principles* useful to traditional designers could be discovered by searching for patterns in evolved circuits. In particular Miller et al. suggested that by evolving a series of modules of increasing size, design principles that they have in common may be extracted from them. In (Miller et al., 1999) and (Miller et al., 1997) they evolved many one and two bit adders, and by inspection deduced the principle of the ripple adder. Although knowledge of this principle already exists in the domain, they went on to argue that evolution discovered and made use of it with no prior knowledge or explicit bias. As the design principle could be extracted from comparing one and two bit adders that had evolved to use the principle, they asserted that evolution could be used as a method of design principle discovery.

More recent work in this area has concentrated on developing an automatic method of principle detection. (Miller et al., 2000b) Having successfully evolved two and three bit multipliers that are much more compact than those of traditional design, they have integrated a data mining procedure to search for design principles (Job, Shankararaman et al. 1999). The learning algorithm used for the data mining process is an instance based learning technique called Case Based Reasoning (Mitchell 1997). In the discussion on scalability in Section 2.2 it will be argued that by modelling biological development it may be possible to allow evolution to encapsulate such design principles automatically without the need to resort to other learning techniques, and use evolution itself to select for design principles that are inherently evolvable.

Complex Technologies

It was noted above that there are complex design spaces for which it has not been possible to develop formal methods to partition and decompose the design space, and that evolutionary algorithms offer an alternative approach to the use of a human expert. An example of this kind of design space is that of analogue circuit design.

One traditional technique of simplifying an analogue design space is to fix the topology of the circuit to a design with well-known characteristics, and modify only parameters relating to the components within the design. A good deal of work that has used evolutionary algorithms in analogue circuit design takes this approach, which can be considered to have more in common with evolutionary optimisation than evolutionary circuit design (Arslan and Horrocks 1995; Murakawa, Yoshizawa et al. 1998). However as the field of hardware evolution has developed researchers have begun to use evolution to explore analogue circuit topologies. For instance Grimbleby developed a hybrid genetic algorithm / numerical search method that used the genetic algorithm to search topologies and a numerical design optimisation method to select parameter values for the evolved topologies (Grimbleby 2000). Koza et al. and Lohn and Columbano have both developed evolutionary circuit design methods that explore both topology and component parameters (Lohn and Colombano 1998; Koza, Bennett et al. 1999). These two

methods are of particular relevance to this thesis as they do not use a fixed mapping of genotype to phenotype. The benefits of using such an approach, and details of these two examples in particular, are discussed at length in Section 2.2.

With the advantages of evolutionary design in mind, Gallagher has recently advocated a return to the development of analogue computers (Gallagher, 2003), which today have been almost completely replaced by their digital counterparts. He distinguished two classes of analogue computers. The first are *direct* computers, which are designed to reproduce the behaviour of a physical system directly. The example he gave was of a serial RLC^3 circuit. This can be considered as directly modelling a damped harmonic oscillator where inductance is equivalent to mass, capacitance is equivalent to the inverse of spring elasticity, and resistance equivalent to frictional damping. Indirect analogue computers simply implement complex mathematical functions using building blocks that embody simple mathematical functions, such as adders and integrators. He suggested that the demise of the analogue computer was mostly down to a combination of the difficulty in discovering direct implementations of required computations and the difficulty in constructing accurate indirect models due to compound errors in component precision. He went on to point out that intrinsic evolution actually discovers direct implementations as the circuit is designed purely to replicate a specified behaviour rather than perform a mathematical function, and that for applications where size and power are vital, evolving direct analogue models should be considered as a serious alternative to digital models of analogue computations.

An impressive example of evolution's ability to manipulate interactions that are too complex for human designers to handle effectively is that of antenna design. It was already mentioned in Section 2.1.1 that evolution is capable of discovering an array of highly unconventional, irregular antenna designs. Early work used simulation but more recently antenna designs have also been evolved intrinsically. As with intrinsic circuit evolution this allows evolution to exploit intricate physical phenomena that are not easily modelled in simulations. In (Linden, 2001) Linden evolved an array of wires connected with reed switches, which are mechanical switches that are closed by an induced magnetic field, controllable from a computer. The antennas that he evolved made use of the complex electromechanical coupling between wire segments that resulted from the fields of the reed switches. Human designers would be unable to exploit such complex nonlinear physical interactions in a controlled manner.

---

[3] A circuit containing resistor, conductor and capacitor components.

New Technologies

At the beginning of this section the idea that evolutionary design is likely to be a useful tool for new circuit design technologies for which no domain knowledge exists was discussed briefly. In (Thompson, 2002) Thompson suggested that until an analytical model of a new technology is derived, the only means to design circuit implementations is to use blind search, such as an evolutionary approach. He used the phenomenon of quantum mechanical tunnelling to illustrate his point: he noted that as fabrication techniques move towards the use of nanoscale circuitry, quantum effects cannot continue to be suppressed to ensure that traditional macroscopic models fit. Instead design techniques must make use of them. However there is little domain knowledge available to suggest how they might be exploited. He then described a circuit design experiment that demonstrates that evolution is capable of exploiting such phenomena. The experiment consisted of an array of quantum dots between which electrons could only pass by quantum tunnelling. The task was to evolve a NOR gate by modifying effectively only the size, shape and position of the dots. Thus evolved circuits would rely on tunnelling effects to perform a logical function. (The task was carried out in simulation but the concept is unaffected.) The evolved circuit used a property called stochastic resonance where the thermal energy of the electrons allows stochastic transmission of a signal. This is an innovative property never before considered for the design of electronic circuits. That evolution discovered this property demonstrates its ability to blindly design in the absence of any useful design rules.

There are also hopes to exploit quantum effects in another way through quantum computing. Quantum computers do not process bits. Instead they process qubits, which exist in a superposition of states. This allows n coupled qubits to represent a superposition of 2^n states, and operators acting upon the qubits operate on the superposition of states in parallel (Benenti et al., 2004). This means that as the number of superposed bits they operate upon increases, the processing power of the device increases exponentially with respect to traditional computing devices. If quantum circuits are developed that can operate on superpositions of even tens of bits, they are likely to have enormous computing power. Theory has pointed to a number of rudimentary quantum gates that could be used to develop quantum circuits, although practice suggests that the number of interconnected gates is likely to become a limiting factor in their design. This has led a number of researchers to begin searching for innovative parsimonious sets of quantum gates using evolutionary algorithms (Surkan and Khuskivadze, 2002; Lukac et al., 2003).

Several researchers have recently suggested that the field should be designing new technologies to suit evolutionary algorithms rather than the reverse. Miller and Downing have noted that all of today's electronic components have been designed specifically for top-down design methodologies, and that researchers in hardware evolution have been 'abusing' these

components (Miller and Downing, 2002). In a similar vein to Thompson they argued that biological evolution is clearly capable of evolving extremely complex structure by *exploiting the physics of the surrounding environment.* Hence they suggested that researchers should be looking to substrates that exhibit rich, complex internal interactions, that are reconfigurable, ideally by small applied voltages, and that can be used to manipulate an incident signal through reconfiguration. They suggested that substances that exist in a state on the edge of disorder would be good candidates as they would exhibit rich range of interactions for evolution to exploit whilst able to quickly relax to a homogeneous quiescent state. They proposed a number of candidates, including liquid crystals, electroactive polymers and voltage controlled colloids. Recently Harding and Miller have attempted to evolve several functions including a transistor-like step function and a frequency discriminator similar in behaviour to that evolved by Thompson (Thompson, 1996a) using an evolutionary platform that incorporates a commercial liquid crystal display as a component (Harding and Miller, 2004). Although they were successful in evolving the required responses, and ruled out the possibility that the same behaviour could be evolved using only the control circuitry alone it is not clear that the liquid crystal itself is responsible for the observed behaviour rather than other elements of the display unit used. However this is a good indication that evolution might prove to be capable of manipulating similar substances to useful effect.

Amorphous computers have also been suggested as a substrate amenable to evolution (Haddow and van-Remortel, 2001). Amorphous computers are essentially large collection of simple, wireless units that perform computations. These units are unreliable, not geometrically aligned, and can only communicate locally, however these features mean they are likely to be relatively easy to synthesise in extremely large arrays compared to other technologies on the horizon. Unfortunately there is no current computational paradigm that can take advantage of their massively distributed function. Future nanoscale devices are also likely to have an amorphous structure, as Miller and Downing have pointed out, (Miller and Downing, 2002) hence this could be a major issue for upcoming computational devices. Haddow and van Remortel have suggested that by combining the principles of biological development and hardware evolution it may be possible to realise designs for amorphous computers (Haddow and van-Remortel, 2001).

2.1.4 Research on Generalisation

As was discussed in Chapter 1, the ability of evolution to generate innovative hardware designs may be vital to continued improvement in hardware performance in both the medium and long term. However there are a number of hurdles to be overcome before hardware evolution becomes a viable competitor to traditional design techniques for general real-world problems. Perhaps the greatest hurdle is the problem of scalability, which is discussed in Section 2.1.5. However another difficult problem is *generalisation*, which is the subject of this section.

Although this thesis is not directly concerned with generalisation it will be seen that it is an issue that is likely to be crucial to the viability of evolved innovative designs, hence it has been included in this review.

Inductive learners such as evolutionary algorithms infer hypotheses from observed training examples. In the case of hardware evolution, prospective circuits are tested by exposing them to different conditions, usually a range of input signals, and observing the circuit outputs, which are used to calculate fitness. When it is infeasible for the learner to observe all possible training examples it must generalise beyond the cases it has observed. Modern real-world circuits can process hundreds of input signals. It is infeasible to observe the effect of each possible combination of so many signals on a circuit, especially if the circuit has many internal states. Hence it is crucial for evolution to be able to generalise to unseen input cases well. As will be discussed later in this section, unseen inputs are but one (albeit important) example of unseen operating conditions that useful real-world circuits must generalise across. The following review of research into generalisation begins with work on input signal cases, followed by work on other environmental conditions. It concludes with work on fault tolerance, which again can be considered as a kind of generalisation.

Generalisation across Input Vectors

Several researchers have explored input generalisation under the framework of pattern recognition, a problem area where the issue of generalisation is well known. As was mentioned in Section 2.1.1 hardware evolution has been shown to generalise to unseen test cases for many real-world pattern recognition data sets, such as image and signal classification problems (Higuchi et al., 1995; Murakawa et al., 1999), and image and signal noise filtering problems (Sekanina, 2003b; Vinger and Torresen, 2003). For all of these examples evolution was restricted to exploring feed-forward networks of nonlinear processing units. Yao and Higuchi likened such networks to ANNs and have implied that the success of hardware evolution in problems like these is at least partially an outcome of this constraint (Yao and Higuchi, 1996). Iwata et al. successfully managed to improve upon the generalisation abilities of this kind of system by applying additional knowledge through the introduction of a heuristic based on the Minimum Description Length (MDL) principle for the discovery of maximum a posteriori hypotheses in Bayesian settings, that biases the search towards circuits that use few components circuits. For details of this interpretation of MDL see (Mitchell, 1997).

Generalisation has been studied in the context of more traditional computational circuits to a lesser extent, and with mixed results. Miller and Thomson investigated the generalisation of two and three bit multipliers with respect to the size of the input training sets (Miller and Thomson, 1998b). The task was to evolve a functional circuit from a subset of the truth table. They found that if evolution was presented with a subset of training cases throughout the entire evolutionary

run it was not able to produce general solutions. This suggests that unlike traditional pattern recognition problems there was no implicit bias towards generality, even though they constrained the design space to feed-forward networks. They also reported that even when evolution was provided with a new set of training cases randomly drawn from the truth table every generation, general solutions were still not found, suggesting that evolution had little memory in the context of this problem.

Miller and Thomson also investigated the evolution of square root functions (Miller and Thomson, 1998c). They discovered that general solutions were often generated when evolution was limited to an incomplete training set. This occurred when the missing training cases tested low-order bits, which contributed less to the fitness. This seems to answer the question as to why their earlier experiments failed to generalise, as is now explained with reference to another experiment.

Imamura, Foster and Krings also explored generalisation in multipliers (Imamura et al., 2000), and likewise found that evolving fully correct circuits was extremely difficult without access to a full training set. They pointed out that the problem was exacerbated in functions where each test vector contained equal amounts of information relevant to the problem, such as the case of the three bit multiplier studied by Miller and Thomson. However they suggested that in cases where the data contained a large amount of 'don't care' values, hardware evolution could be successful using a smaller test vector. Real word pattern classification data contain redundant information, which explains why they often generalise well when evolved multiplier circuits do not. It seems reasonable to assume that for any real-world problem some level of redundancy is likely to exist, although the problem of how to select test vectors remains. In such cases Immamura Foster and Krings suggested that an adaptive approach of allowing the evolving system to search for useful subsets of test vectors might be possible.

Generalising Across Operating Environments Though Representational Bias

Just as it is unrealistic for the algorithm to train using every conceivable circuit input, it is usually unrealistic to train under every conceivable operating environment. Operating environments might include a range of technologies or platforms on which a circuit must operate, and a range of environmental conditions that the hardware may be subjected to.

Traditional digital designers usually achieve generalisation across a range of technologies by imposing temporal and spatial constraints on the nature of the circuit so that a design abstraction is realised, as discussed in Section 2.1.2. By using traditional design abstractions the designer can assume that any circuit within the design space will behave as the abstraction predicts, but when the circuit is realised in a particular technology, technology-specific temporal constraints, specified by the manufacturer of the physical device with which the design is realised, must be

observed. The device manufacturer will also provide a set of environmental constraints, such as temperature, humidity and power requirements under which the abstraction is guaranteed to hold. To date hardware evolution usually avoids the issue of generalisation across a range of operating environments by constraining the genetic representation so that the search space contains circuits that adhere to the same constraints used by traditional designers. Of course this means the design is subject to traditional environmental constraints if the design is actually realised in hardware.

As mentioned in Section 2.1.3 the reason that hardware evolution practitioners constrain their representations in this way is perhaps more down to habit and familiarity than any explicit intention to achieve good generalisation characteristics. In fact there is only one case that the author is aware of where additional representational constraints have been imposed specifically to ensure good generalisation (Stoica et al., 2003). In this experiment a very high level of generalisation was required. Circuits were evolved at the transistor level, and the genetic representation was constrained to ensure that input signals connected to transistor gates rather than source or drain inputs, thus improving the loading characteristics of the evolved circuits. (The experiment is discussed in more detail in the following section.)

Generalisation across Operating Environments through Procedural Bias

Section 2.1.3 discussed a growing body of work where hardware evolution has been applied to design spaces that reach beyond traditional design abstractions in search of innovative designs. In these cases traditional constraints are relaxed, hence there is no guarantee that evolved circuits will exhibit good generalisation characteristics across a range of operating environments. Additionally there is little domain knowledge concerning how to achieve good generalisation without imposing traditional constraints. One solution is for evolution to infer this information from examples.

This problem was first acknowledged by Thompson when exploring design innovation through the relaxation of traditional constraints (Thompson, 1996a). He successfully evolved a circuit to distinguish between two frequencies using a Xilinx XC6200 FPGA but noted that it was not robust to changes in environmental conditions such as temperature, electronic surroundings, and power supply. He also noted that the design was not portable, not only to a different FPGA but even when realised on a different area of the same FPGA. Similar results have been reported by Masner et al. (Masner et al., 1999). Thompson went on to explore how solutions that generalised well across a range of operating environments could be evolved (Thompson, 1998). He started by specifying a number of parameters for an *operational envelope* that when varied affected the performance of this circuit. The parameters chosen were temperature, power supply, fabrication variations, packaging, electronic surroundings, output load and circuit position on the FPGA. The final population from the frequency discriminator experiment was

then allowed to evolve further. Each individual was evaluated randomly on one of five different FPGAs maintained at the limits of environmental conditions specified by the operational envelope parameters. Although there was no guarantee that the circuit would generalise to behave robustly to all environmental conditions within the envelope, Thompson found a level of robustness evolved in four out of five cases.

In a similar vein Stoica et al. explored the operation of circuits in extreme temperatures (Stoica et al., 2001). Their initial experiment involved subjecting both traditionally designed and circuits evolved under a single, standard operating environment (multipliers, Gaussian curve generators and logic gates) to extreme temperatures. This was primarily an experiment in evolutionary fault recovery, and they demonstrated that while all the designs initially failed under extreme conditions, they all regained functionality when re-evolved under the extreme conditions. However a population of 50 circuits re-evolved for 200 generations in this manner often exhibited degraded performance under standard conditions, where before they had functioned perfectly. This suggests that generalisation qualities can be lost easily if a consistent bias towards them is not asserted during evolution.

A problem closely related to Thompson's exploration of portability is the portability of extrinsically-evolved analogue circuits to physical devices. Analogue circuit simulators tend to simulate circuit behaviour very closely, and so it might be expected that analogue circuit designs evolved extrinsically generalise well to physical circuit technologies. However this does not happen in practice. One issue is that some phenomena that arise in simulation due to the physics model used by the simulator may not actually be feasible in the chosen implementation technology. A widespread example is that analogue simulators commonly allow the use of extremely high currents, and so evolution is free to take advantage of them in its design. Koza et al. have evolved many circuits extrinsically using a typical analogue abstraction provided by the Berkeley SPICE simulator (Koza et al., 1999), but found that they are practically infeasible because of their reliance on extremely high, and often infinite currents. Additionally analogue simulators use precise operating conditions. The circuits of Koza et al. are evolved to operate at 27°C. Analogue abstractions do not contain any bias within the representation towards generalisation across a range of temperatures, hence the circuits were often found to fail if the operating temperature was varied.

When evolving networks of transistors intrinsically Stoica et al. have come across the reverse problem: circuits evolved intrinsically may operate as expected but may not operate acceptably in simulation (Stoica et al., 1999). Their solution to the problem was to evaluate some circuits of each generation intrinsically, and some extrinsically. They dubbed this *mixtrinsic* evolution (Stoica et al., 2000). They suggested that mixtrinsic evolution could also be used to focus

evolutionary searches on innovative designs: circuits could be rewarded when found to behave differently in simulation than when realised in a physical circuit. This would encourage innovative behaviour not captured by simulation. However they have not explored this in practice. In (Guo et al., 2003) they developed the mixtrinsic method to include several different software models, based on various different processes, analysis tests and timing resolutions.

The discussion above has only dealt with portability between simulation and reconfigurable devices. An important issue is whether designs evolved intrinsically, extrinsically or indeed mixtrinsically are portable to custom ASICs[4], which cannot be used during evolution. Until recently this question had been left unanswered, but in (Stoica et al., 2003) Stoica et al. evolved transistor level circuits using a combination of comprehensive fitness testing on each individual and mixtrinsic testing across the population. Comprehensive tests included transient analyses at different frequencies, testing a number of loads. Mixtrinsic tests were SPICE analysis on several process models and a range of voltages and temperatures. Additionally an additional constraint was imposed through the representation to improve loading characteristics, as was mentioned earlier. Tests that were carried out mixtrinsically during evolution were carried out in full on the final evolved solutions, and revealed that some but not all of the circuits performed robustly across all tests. All circuits exposed to the full range of validation during evolution were successfully validated in silicon, showing that with careful validation procedures, portability of evolved designs to ASIC technologies is possible.

Fault Tolerance

The idea of evolving circuits that function under various environmental conditions can be extended to include the capacity of circuits to operate in the presence of faults. This was first investigated by Thompson (Thompson, 1996d). Here he evolved solutions to the DSM-based controller problem, introduced in Section 2.1.3 during the discussion on relaxing constraints, but this time in the presence of single-stuck-at (SSA) faults in the RAM used to hold a lookup table of state transitions for the state machine. Rather than test each candidate solution exhaustively across all sets of possible faults he aimed to test only the fault that caused the most degradation in each controller. He recognised that the population was likely to be made up of individuals of various designs hence the highest degradation of performance for member of the population was unlikely to be caused by the same fault in the RAM. To circumvent this problem, at each generation he averaged the DSM RAM bits of the entire population to give what he termed a 'consensus individual', and tested this against all possible faults, in the hope of identifying the fault most likely to reduce performance in any given individual. This procedure was only introduced once a good solution had been evolved, and then the population

[4] Application Specific Integrated Circuits

was tracked to see how it performed. Solutions that were tolerant to most SSA faults existed in the initial population of evolved solutions for reasons discussed in the section below. But as evolution proceeded in the presence of faults, tolerance was lost as the algorithm concentrated on tolerating the single worst fault, until eventually solutions tolerant to any single fault were discovered.

Canham and Tyrell extended this work to more complex faults that commonly develop in FPGA architectures (Canham and Tyrrell, 2002). They emulated the Xilinx 6200 FPGA series architecture (Xilinx Inc., 1997) on a Xilinx Virtex FPGA (Xilinx Inc., 2001), and introduced simulated SSA faults in the logic of the configurable logic blocks (CLBs) and short circuit faults between the inputs and outputs of the CLBs during evolution. They compared the circuits against a set of control circuits evolved in the absence of faults and found a large increase in fault tolerance that could not be explained purely by 'junk' faults occurring in unused areas of the FPGA.

Hartmann et al. have explored fault tolerance by inference from examples at a lower level of abstraction than those above. They successfully evolved fault tolerant circuits using non-perfect digital gates called messy gates (Hartmann et al., 2002). Various levels of noise were injected into digital gate models simulated using SPICE, and digital circuits were evolved. The circuits were manipulated by evolution at gate level granularity, but the evaluation of circuits was carried out using SPICE, hence other traditional digital constraints were not present. They discovered that adders and multipliers could be evolved under high levels of noise. They suggested that the noise might smooth the fitness landscape as highly fit circuits that depended on each gate to function were no longer present in the search space.

This is one of the few examples in the literature of hardware evolution at an unusual abstraction somewhere between the traditional digital and analogue spaces, which is realised by using an unusual combination of constraints from the axes shown in Figure 2.1.3.

Inherent Fault Tolerance

The biases of the evolutionary algorithm can exhibit an inherent tendency to generate solutions that generalise across certain conditions. Thereby evolved circuits might exhibit robustness to changes in those conditions although there is no explicitly defined bias to do so, providing another source of fault tolerance.

Thompson was the first to propose that evolved circuits may be inherently robust to some types of fault (Thompson, 1995b). He observed that an evolutionary algorithm will by nature be drawn to optima surrounded by broad areas of high fitness rather than sharp fitness peaks hence the fitness of an evolved solution that is then subjected to a mutation is likely to be higher than

expected. He demonstrated this through a series of experiments using artificial NK landscapes. For details of this type of landscape see (Kauffman and Levin, 1987).

He went on to suggest that if a circuit representation was designed so that a single mutation had the same effect on the circuit phenotype as a fault then evolved circuits would tend to be insensitive to that fault. He demonstrated this using the evolution of the DSM robot controller described in Section 2.1.3 as an example. Each bit of the RAM that encoded the controller's state machine was directly encoded in the chromosome, and so mutation of one of these bits had a similar effect to a 'single stuck at' (SSA) fault. Examination of the effect of SSA faults on a previously evolved state machine revealed that it was indeed robust to faults. However as state machines for this problem with similar fitness could not be easily generated by any means other than evolution, statistical tests of the evolved machine's resilience to faults were not carried out.

In (Thompson, 1996d) he pointed out that as the single-mutant neighbours of the circuit are likely to be also insensitive to faults for the same reason, evolved circuits are likely to degrade gracefully as faults are incrementally introduced, rather than fail suddenly. (Note that again such fault insensitivity is only likely if mutations alter the implementation of the circuit in the same way as a fault rather purely reproduce the behaviour.) He also used NK landscape models to demonstrate the single-bit effect discussed above for a larger set of landscapes than his earlier work, and showed that it was more pronounced for greater rates of mutation, until a maximum rate where the correlation between parent and offspring became too low for the genetic algorithm to follow a fitness gradient effectively.

Masner et al. found a similar effect during a study of how different representations alter the robustness of evolved sorting networks to a range of faults (Masner et al., 1999). They explored the relationship between the size and robustness of sorting networks using both a tree and linear representation. They noted that robustness tended to first increase and then decrease with the size of the network, and is therefore not due purely to the existence of redundant non-functional gates in the sorting networks.

Layzell and Thompson suggested that robustness can also arise from the incremental nature of the evolutionary process. In (Layzell and Thompson, 2000) they explored the ability of another member of the population to be robust to a fault that causes the fittest solution to fail. They called this populational fault tolerance (PFT). They showed that PFT is inherent in certain classes of evolved circuit, and tested various hypotheses that could explain its nature. As with Masner et al. they noted that fault tolerance did not seem to simply be a result of repeated, redundant sections of the design that failed. Instead descendants of an inherently different design that had at an earlier point in evolution been the best solution in the population were still

present in redundant genes within the members of the population that provided PFT. They demonstrated that this was not simply the result of population diversity by repeating the experiment using a single hillclimber to evolve solutions and then generating 50 single bit mutants of this single individual, thus a highly related population. These individuals presented similar levels of tolerance to faults as the traditionally evolved population, confirming that the fault tolerance was inherent to the *incremental* nature of evolutionary processes in general: the entire population contained remnants of inherently different solutions that evolution had explored earlier.

Fault tolerance can also arise from diversity in the population. Tyrrell et al. have explored such tolerance (Tyrrell et al., 2001). Unlike Layzell and Thompson's work population diversity was encouraged, by evolving oscillators using a population of 16 hillclimbers that did not interact with each other. This ensured that the evolved solutions did not share a common evolutionary history, so that any fault tolerance observed was not a result of the PFT effect proposed by Layzell and Thompson. When faults that caused the best member of the population to fail were introduced to the oscillators, another member of the population often retained relatively high fitness.

2.1.5 Research on Performance and Evolvability

Schema theorem implies that given an infinite number of evaluations a traditional genetic algorithm will discover the optimal solution in a search space (Goldberg, 1989). However evolution is only of practical use if it does so in a shorter time. A useful heuristic to gauge the performance of a genetic algorithm, and one used in this thesis, is that evolution should be more efficient than random (or exhaustive) search. In reality it is often the case that evolution performs poorly, either degrading to random search, or effectively converging to a local maximum in the search space.

A good deal of research in the field of hardware evolution is devoted to the following:

- improving the quality of solutions that evolution discovers for a given problem;
- improving the scalability of evolution to larger and/or more complex problems and
- improving the speed with which evolution finds acceptable solutions.

These ideas are highly interrelated as they all aim to improve the performance of the evolutionary search in order to achieve slightly different goals. Hence here they are dealt with together.

Representations

Selecting a good representation is crucial to the performance of an evolutionary algorithm. As discussed in Chapter 1, the representation of an evolutionary algorithm defines how solution space is mapped onto search space. In doing so it affects the performance of the algorithm as it delimits the solutions present in the search space, thereby fixing the density of acceptable solutions in the search space. Many researchers, particularly in the early days of hardware evolution, believed that performance could be improved by reducing the size of the search space and increasing the density of good solutions lying within it. This idea is discussed shortly. However representation has a second and usually more profound effect. Representation partly specifies the *order of traversal* of search space, as it sets the distance between any given points in space. Hence a change in representation can radically change the *nature* of the search space, not just its size. It is becoming increasingly recognised that it is not so important to have a small search space as one that allows evolution to discover incremental improvements in fitness that lead it to an acceptable solution (Dawkins, 1989; Altenberg, 1995; Thompson, 1996b). For the purposes of this thesis a search space that encourages this process is called an *evolvable* search space.

Miller and Thomson have explored how changes in circuit geometry affected the evolvability of a two bit multiplier (Miller and Thomson, 1998b), and how the functionality-to-routing ratio affected the evolvability of netlist representations of the SBOX problem space (Miller and Thomson, 1998a). They found that evolvability was affected profoundly but erratically by both factors, making it difficult to draw many general conclusions. However they noted that evolvability was improved by altering the representation to introduce cells dedicated to routing signals between functional cells. However it is dangerous to read too much into their results because they may be dependent on the SBOX problem, the biases imposed by the specific operators used within the algorithm, and the level of abstraction at which the experiments were conducted.

Function Level Evolution

The function level approach to improving the performance of hardware evolution was proposed by Murakawa et al. (Murakawa et al., 1996), and has since been adopted by many others (Torresen, 2000a; Sekanina, 2002b; Thomson and Arslan, 2003; Torresen, 2004). They pointed out that the size of the search space for a traditional binary genetic algorithm increases at a rate of 2^n for every additional n genes, and suggested that as evolution tackles larger problems the explosion in search space size prevents it from searching effectively. Hence they proposed function-level evolution: instead of using gate-level representations, traditional domain knowledge could be used to select high level computational units, such as adders, subtractors and sine generators that could be represented directly in the chromosome, thereby reducing the

size of the chromosome necessary to represent an acceptable solution. Although this approach has proved to be successful for specific problems, there are several issues which mean that it is not a good general solution. First is the problem of domain knowledge, which requires an experienced designer to select suitable function-level units for the problem at hand. If little or no domain knowledge exists for the problem it may not be suitable for a function level approach. Second the approach is not hierarchical: it is not scalable to problems of incrementally greater complexity without introducing more domain knowledge through the selection of increasingly more powerful functions. Third, the selection of function level units imposes a design abstraction as a hard bias: evolution is strictly limited to a search at one abstraction, and any innovative solutions that require modifications to the circuit at a lower level of abstraction will be unattainable. Finally, and perhaps most importantly, the functional units are selected using domain knowledge from traditional design processes. As has been discussed throughout this chapter, evolution performs an incremental, bottom-up search rather than a top-down design. It has already been pointed out that there is very little domain knowledge about the general *evolvability* of circuit design spaces, and so even functions selected by experienced designers may not be of value when attempting to solve a problem using an evolutionary algorithm.

Indeed Thompson argued that coarse-grained representations such as those employed by function-level evolution may actually reduce the evolvability of a hardware design space (Thompson, 1996b), as the addition or removal of a complex function from a circuit design is likely to have a more dramatic effect on the overall function of the circuit than a simple function. Thompson made a strong argument that traditional evolution has the capability to search larger spaces than are advocated by Murakawa et al. in (Murakawa et al., 1996). In particular he suggested that there may be features of many hardware design landscapes called *neutral networks* that allow search to continue beyond the point where the evolving population has converged and exhibits little genetic variation, the point where schema theorem suggests that evolution effectively halts.

Neutral Networks

A neutral network can be pictured as a collection of genotypes with phenotypes of identical fitness that are arranged in search space so as to make a pathway or network that can be navigated by evolution through the application of its genetic operators. It has been suggested that *genetic drift* along such networks can allow evolution to escape local optima that they would otherwise be anchored to (Huynen et al., 1996). The idea of neutral mutations has been recognised in the field of evolutionary biology for some time (Kimura, 1983) but has only been used as a model for search in Evolutionary Computation more recently. Harvey suggested that to make full advantage of neutral networks requires a redesign of evolutionary algorithms, and in light of this proposed the Species Adaptation Genetic Algorithm (SAGA) (Harvey, 1991). SAGA was designed to allow incremental changes in genotype length and place a much greater

emphasis on mutation than is common for genetic algorithms. He proved his point by using only a fixed length genetic algorithm with a SAGA-style mutation rate to search an incredibly large circuit design space (2^{1800}) for good solutions. When the algorithm was stopped owing to time constraints, fitness was still increasing even though the population had converged long before (Harvey and Thompson, 1997). Analysis of the evolutionary process did indeed reveal that a converged population had drifted along neutral networks to more fruitful areas of the search space. They suggested that the success experiment was due to increased mutation rate, and the full SAGA model might provide even greater evolvability. They also speculated that neutral networks might be a feature of many design spaces including hardware design spaces.

Since then neutrality has been explored for other hardware design problems. For instance Vassilev and Miller have investigated neutrality for a three bit multiplier problem (Vassilev and Miller, 2000a). They found that neutral changes at the start of an evolutionary run occur because of high redundancy in the genotype and as evolution proceeded and fitness increased, redundancy in the genotype reduced. However the number of neutral changes did not drop as quickly, suggesting that selection actually *promotes* neutral changes in order to search the design space. In line with Thompson they showed that when neutral mutations were forbidden, the evolvability of the landscape was reduced. They have also proposed that the search for innovation may be assisted by using current designs as a starting point for evolution, and a bridge composed of neutral solutions adjacent in space could be used to lead the search from conventional design space to areas beyond (Vassilev and Miller, 2000b).

Although both Thompson and Vassilev and Miller showed that neutrality can benefit evolvability, it seems unlikely to be a panacea for the explosion of search space size when evolution is applied to large problems. While large in evolutionary terms, the design space searched by Thompson's experiment is still extremely small when compared to most real-world hardware design problems. Furthermore as the neutral model is driven by mutation and genetic drift, evolution proceeds slowly and during drift phases equates to no more than a random walk. Thompson's experiment took several days to discover a satisfactory circuit. Vassilev and Miller found that successful evolutionary runs often required around a million evaluations to discover a three bit multiplier, and many runs failed to evolve fully adders (Vassilev and Miller, 2000c). (Although they found that by increasing the number of available gates the problem became easier.) This suggests that while evolvability can be somewhat improved by taking advantage of neutral networks, additional improvements in the evolutionary model are likely to be needed if large problems are to come within hardware evolution's grasp.

Incremental Learning

Following on from earlier work using the function-level approach, Torresen proposed another idea to improve scalability based on the evolution of increasingly complex components. He was

inspired by the apparent improvements in scalability that had been demonstrated by Koza through the introduction of Automatically Defined Functions (ADFs) to genetic programming (Koza). ADFs provide a mechanism for evolution to *re-use* innovations throughout a design once they have been discovered, and are discussed later. Torresen recognised that assisting evolution in its incremental, bottom-up design process rather than trying to impose knowledge derived from traditional top-down design might improve scalability. Hence he suggested that evolution could evolve and then re-use high level hardware functions (Torresen, 1998). The process could be repeated incrementally, so as to produce a complex solution based on a series of hierarchical modules that had been iteratively encapsulated into larger ones. He dubbed this *increased complexity evolution*. However it required a mechanism to modularise the problem into less complex, evolvable subtasks.

He suggested that this could be achieved either by simply subdividing the problem using a traditional functional decomposition, or even into sub-problems that each generates a single circuit output. He demonstrated this for a classification task where a number of character images were to be classified according to character. Each output of the circuit corresponded to one of the characters that could be input to the circuit. The aim was to evolve a circuit where at any time one output that corresponded to the character represented by the current input data was high. He decomposed the problem manually into a set of circuits where each would be evolved to detect only a single character. This led to a significant increase in evolutionary performance.

Unfortunately he stopped short of demonstrating the benefits that his approach could bring to scalability as he did not present a demonstration of *hierarchical* evolution: evolution at a higher level of abstraction using the evolved circuits as primitives. Furthermore this approach might actually curtail the opportunity for an incrementally evolved system to innovate (which in this thesis is regarded as the primary motivation for hardware evolution) as evolution is strictly limited in the way it solves the problem. Evolution is not permitted to explore optimisations that could be made through interaction between the subcomponents. Similar methods such as (Kalganova, 2000) and (Stomeo and Kalganova, 2004) suffer from similar problems or make additional assumptions, for instance that the circuit is combinational.

Some of these issues were partly addressed in (Torresen, 2001) when he applied the method to the evolution of a prosthetic hand controller. Here he used two evolutionary stages. First he evolved separate sub-circuits for each of the six hand motions. The second stage of evolution explored where output signals should be drawn from each of these circuits, thus providing a limited demonstration of the re-use of the evolved components as primitives for further evolution and providing a limited mechanism for evolution to explore interactions between the evolved subcircuits. Later work on the same problem went further (Torresen, 2002b): the first

stage of evolution was carried out several times, providing a pool of subcircuits for each movement. The second stage of evolution was then used to select and combine subcircuits from the pools generated for each movement. However the architecture of the system still strictly limited how many subcircuits were available for evolution to explore and how the evolved subcircuits could interact. Hence opportunities for evolution to discover innovative designs were extremely limited. Furthermore the system used two fixed stages of evolution, thus still only provided limited insight into the true potential of hierarchy in an incremental approach. In (Torresen, 2004) Torresen countered the first of these points by arguing that to date there are few examples where evolution has discovered truly innovative designs, hence the promotion of evolutionary innovation is not valuable. However there are many examples in the literature of innovative circuits discovered by evolution, some of which have been discussed earlier (Huelsbergen et al., 1999; Thompson and Layzell, 1999; Koza, 2003; Raichman et al., 2003). It seems likely that the issues preventing wider research into such circuits are not to do with the inability of evolution to innovate, or the value of allowing it to do so. Rather it is issues such as scalability, generalisation and circuit analysis that prevent their use in real-world settings.

Although increased complexity evolution is not a perfect solution to the scalability problem, it may be useful when there is strong evidence that a traditional decomposition is useful. For example Hounsell and Arslan (Hounsell and Arslan, 2000) used a similar method to evolve three bit multipliers, decomposing the problem by output pin. In this case they integrated the individual circuits, which were evolved extrinsically, automatically using standard logic minimisation techniques, thereby automating the technique and addressing to some extent the issue of circuit parsimony that Torresen had not touched upon. Shanthi et al. have also used a similar technique to evolve impressively large adders and multipliers (Shanthi et al., 2004), but without any real opportunity of innovation as the problem was again decomposed manually into extremely small, independently evolved sub-problems.

Kazadi et al. (Kazadi et al., 2001) removed the requirement to combine evolved circuits using traditional minimisation techniques, thereby increasing the opportunities for innovative circuit design. They achieved this by first evolving the correct behaviour for a single output, and then selecting a single parsimonious solution and encapsulating it as a module. The module was then used a primitive for another stage of evolution in which correct behaviour for an additional output was required. The process was iterated until correct behaviour for all outputs was observed. Although this method can automate the generation of a complete circuit, it again relies on functional decomposition by problem output producing evolvable sub-problems, which may not be the case.

Lohn et al. have compared a number of incremental-type systems. They compared three dynamic fitness functions against a static one (Lohn et al., 1999). The dynamic fitness functions increased in difficulty during an evolutionary run. One had a fixed increase in difficulty, based on domain knowledge, one had a simple adaptive increase based on the best fitness within the population, and one put the level of difficulty under genetic control by using a co-evolutionary approach. Co-evolutionary systems usually evolve two competing populations, where the fitness of each member of a population is determined by measuring its performance against members of the other population. In this case the first population remained the same as the earlier experiments and the second consisted of target vectors. The results showed that the co-evolutionary system performed best on an amplifier design problem, but the static system performed best across a range of problems. When discussing the reasons for this they noted that the discontinuity in the fitness landscapes resulting from the adaptive nature of the fitness functions might have reduced the evolvability of the incremental systems.

Dynamic Representations

Another proposal to improve evolutionary performance was to use a variable length representation, which might reduce the size of the search space necessary for a problem (Kajitani et al., 1996). In this example variation of the representation length was placed under evolutionary control. Performance for a pattern recognition problem was found to be improved over an algorithm that did not use variable-length representations, both in terms of solution parsimony and efficacy.

A similar approach was taken by Zebulum in an experiment to evolve Boolean functions using a chromosome of product terms that were summed by the fitness function (Zebulum et al., 1997). However the order in which the representation space was searched differed from the example above. Inspired by the observation that complex organisms have evolved from simpler ones, the population was seeded with short chromosomes. This assumes that there is a correlation between complex behaviour and complex structure. As was discussed earlier, Thompson has demonstrated that this is not necessarily true, as complexity in behaviour can arise from interactions of a simple system with a complex environment (Thompson et al., 1995). However the simplicity of the simulation used to evaluate circuit designs in this example may mean that in this case the assumption holds. A new operator was introduced to increase chromosome length, under the control of a fixed parameter rather than under evolutionary control as in the example above. Hence a simple pressure to move from short representations to long ones was set. It was found that a low rate of increase allowed fully functional, but more parsimonious solutions to be found over a larger rate.

In both the experiments described above each gene in the representation was mapped directly to a Boolean function, and the representation space was searched by adding and removing genes

guided by evolution in the first case, and by adding genes guided by a simple heuristic in the second case. In both cases the aim was merely to reduce the size of the search space rather than transform the fitness landscape to something more evolvable. Although such an approach might be a useful tool to improve scalability or performance to a small degree, it is only likely to provide a small improvement, as borne out by their results.

A More Appealing Solution

Although all the work into improving performance and scalability discussed above has limitations, it has identified many features that are likely to be useful and should be incorporated into future attempts to solve the problem. Some work has recognised that it might be useful to limit the explosion of search space as problem size increases, for instance (Murakawa et al., 1996; Sekanina, 2002b), although it is argued here that this should not come at the expense of an evolvable landscape. Some aim to transform the fitness landscape to improve its evolvability, for instance (Miller and Thomson, 1998a; Torresen, 1998) although it is argued here that this should not come at the expense of the opportunity to innovate, or rely entirely on existing domain knowledge. Re-use of previously discovered innovations has been identified as a useful tool (Torresen, 1998; Hounsell and Arslan, 2000; Kazadi et al., 2001; Shanthi et al., 2004), but it is argued here that the process by which innovations are encapsulated should allow the capture of units that are truly useful to evolutionary search, not unduly biased by the knowledge of traditional designers.

A solution that has the potential to incorporate all of these ideas already exists. Indeed examples of its ability to allow evolution to generate large, complex structures are ubiquitous. This solution is *development.* Nature essentially uses development to map the genotype of a multicellular organism to large and complex phenotypes. How development achieves this while addressing the issues above is dealt with in detail in Section 2.2 of this chapter.

2.2 Development

For three billion years Earth was inhabited by only single-celled organisms (Fortey, 1999). Five hundred million years ago multicellular organisms first evolved, and proved to be incredibly successful. In a geological flash a huge array of innovative body designs emerged as evolution rushed to fill newly exposed ecological niches. This unprecedented period of evolutionary exploration can be detected in the fossil record, and is known as the Cambrian Explosion (McMenamin and McMenamin, 1990). The descendents of these pioneers of multicellularity still make up a large proportion of modern phyla, and are known collectively as the Metazoan kingdom, or more commonly as animals.

An analogy can be drawn between the days of acellular dominance and the current state of Evolutionary Computation: evolution has been applied to a wide array of problems and in some cases provides the best engineering solutions (Koza, 2003). But to date evolution has been limited to small problems and would struggle to compete with traditional design techniques for most real-world design problems, for instance the design of a modern microprocessor, a passenger jet or a skyscraper. Perhaps what is needed for evolutionary design to make this leap is a second Cambrian Explosion that sees the advantages of multicellularity put to work in the field of Evolutionary Computation.

The process by which all multicellular organisms transform from a single-celled zygote to an adult composed of trillions of cells is called *development.* This section reviews the motivations behind, research into and application of development in Evolutionary Computation. Section 2.2.1 reviews the applications of development in Evolutionary Computation to date. Section 2.2.2 provides a background on developmental biology. Section 2.2.3 explains the benefits that development can bring to Evolutionary Computation and hardware evolution in particular. Section 2.2.4 reviews models of development that have been used by other researchers. Section 2.2.5 reviews the use of development in hardware evolution, from the earliest accounts to the state of the art, paying particular attention to scalability.

2.2.1 Applications of Developmental Models

Researchers have been putting models of development to use for many years, and have applied them to a wide array of problems. Developmental processes are not well understood by biologists. Therefore it is no surprise that majority of examples of computational models of development in the literature have been motivated by the hope that they might provide insight into their biological analogues. It is only in recent years that computing power has increased enough to allow complex models that cannot be easily interpreted by hand to be investigated. These models range from those that model complex biochemical pathways in single cells to

those that model the morphology and patterning of organisms, from trees to butterflies. Examples of work that models development to gain some insight into Biology are listed below:

- Biological morphogenesis (Savill and Hogeweg 1997) (Kumar and Bentley, 2003)
- Heterochrony (Dellaert 1995; Cangelosi 1999)
- Recapitulation (Nolfi and Parisi, 1993)
- Pattern formation (Turing 1952; Eggenberger and Dravid 1999; Murray 2002)
- Neural network development (Kitano, 1990; Eggenberger, 1997a)
- Regeneration (Eggenberger, 2003; Miller, 2004)
- Artificial Life (Sims, 1994; Bongard and Paul, 2000)
- Emergence of Gene Regulatory Networks (Kauffman, 1969; Bentley, 2004a)
- Plant development (Lindenmayer, 1968; Prusinkiewicz et al., 1993)
- Insect eye development (Dellaert, 1995)

A number of researchers have also realised that although the intimate details of biological development are still not fully understood it is possible to use current models to inspire design decisions for many engineering problems. Most commonly models of development have been used in conjunction with an evolutionary algorithm to design solutions automatically. A wide range of development-inspired evolutionary design applications are listed below:

- 3D structure design (Hornby and Pollack, 2001)
- GRNs for autonomous agent controllers (Quick, 2003; Bentley, 2004b)
- ANNs for autonomous agent controllers (Cangelosi and Elman, 1995; Gruau, 1995; Jakobi, 1995; Dellaert and Beer, 1996; Eggenberger, 1996)
- ANN design for classification (Gruau, 1992)
- Agent morphology (Dellaert, 1995; Hornby, 2003a)
- Circuit design (Koza et al., 1999; Miller and Thomson, 2003) (Hemmi et al., 1996; Tufte and Haddow, 2003b)
- Design of sorting networks (Belew and Kammeyer, 1993; Sekanina, 2004)
- Fault tolerant hardware (Mange et al., 2000; Edwards, 2002; Canham and Tyrrell, 2003)
- Program design (Koza, 1994)
- Telephone mast placement (Tateson, 1998)

2.2.2 Biological Background

Development has been studied for over a century, and is still by no means completely understood. However what biologists do know about development is fairly uniform across the Metazoan kingdom and relies on four fundamental developmental processes popularised by Wolpert et al. (Wolpert et al., 2002): *regional specification*, *cell differentiation*, *morphogenesis* and *growth*. These are now discussed.

The Fundamental Developmental Processes

Regional specification: Regional specification is a process of pattern formation. During this time a spatial and temporal pattern of cell activities is organised. It results in the cells of the germ layers acquiring different identities so that organised patterns of cell differentiation can later emerge. During early regional specification these identities may be defined by only subtle chemical differences. Regional specification begins during cleavage and continues throughout the early stages of development.

Cell differentiation: During cell differentiation cells become structurally and functionally different from each other, ending up as distinct cell types. Differentiation is a gradual process that occurs during most stages of the development of an organism, and is closely linked to pattern formation.

Morphogenesis: Morphogenesis is the movement of cells and tissues that alter the form of the embryo. It is achieved through a number of different strategies including cell division, alterations in cell adhesion and apoptosis, or programmed cell death.

Growth: An organism shows very little increase in size during development until the basic form of the embryo has been laid down. The majority of size increases result from growth. Morphogenesis can also be brought about by differences in growth rates of areas of cells.

The engine that drives these three processes is the activation of genes to produce proteins. However this engine is directed by the *differential* activation of genes: different genes can be activated in different cells and at different times, resulting in cells with differing chemical environments. To understand the mechanisms behind differential gene activation it is important to understand the process by which the information encoded by a gene is converted into a protein. The full process is a complex orchestra of biochemical pathways, but the part of the process responsible for the majority of differential gene activation is the stage where the instructions held in DNA are first decoded. This is *DNA transcription.*

DNA Transcription

The process of DNA transcription as described in (Raven and Johnson, 2001) is shown in Figure 2.2.1. It begins when protein called RNA polymerase binds to a gene. A gene consists of a series of molecular groups called base pairs. The RNA polymerase binds to a special sequence of base pairs that mark the start of the gene called a promoter. Once bound, the RNA polymerase travels along the DNA molecule, at the same time generating a molecule of RNA specific to the DNA from which it was generated. The rate at which genes are transcribed (hence expressed) is controlled by the presence of more proteins called transcription factors. These are called activators or repressors, depending on whether they increase or decrease the rate of transcription of a gene. They bind to the RNA polymerase and each other forming a complex of proteins, and often additionally require specific sequences of DNA upstream of the gene to stabilise the process. Typically many transcription factors are needed to stabilise the binding of RNA polymerase to the promoter.

Once the RNA has been generated it migrates to another area of the cell where it is used as a code to generate a specific protein. This model of transcription suggests a linear relationship between the concentration of the factors needed to transcribe a gene and the concentration of the protein that results from its transcription. However the situation is more complex. The bond between transcription factor and promoter is not binary. Instead the rate at which transcription factors bind to a promoter site also depend on the *affinity* between the factor and the site, which is a function of their steric interaction. The complexity of both protein stereochemistry and the reaction kinetics of the chemical processes involved throughout the process itself means that this is not an easy value to deduce.

The protein that eventually results from the transcription of a gene may be one of many classes of protein. The most common proteins are enzyme proteins. These catalyse biochemical reactions within the cell, and allow organisms to regulate the production of all non-protein substances. Another common class of proteins is that of structural proteins, which create or maintain the structure of the cell. Regulatory proteins, or transcription factors, make up a third class

There are a number of ways by which the capacity of a gene to be activated by transcription factors can be altered. These include chemical alteration of a gene (for instance by methylation), and interruption of the gene translation mechanism by enzymes (Slack, 1991). However these regulation strategies are sideshows to regulation by transcription factors as described above. Chapter 1 introduced the concept of a Gene Regulatory Network (GRN). These networks are useful models of the regulatory relationships between genes, transcription factors and other gene

products and will be used later in this chapter to explain some of the benefits of development in hardware evolution and later in this thesis to describe the regulation of hardware development.

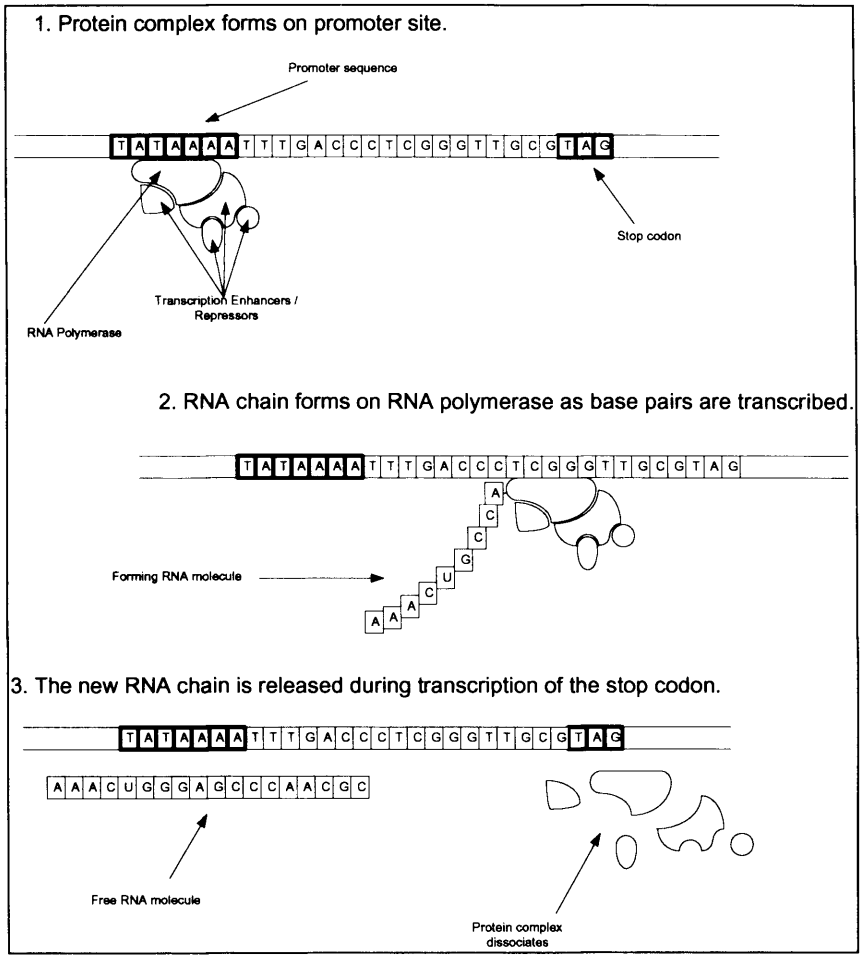


Figure 2.2.1: DNA transcription.

Intercellular Communication

Transcription provides a model of how a single cell can regulate itself and maintain a stable state. Acellular prokaryotes even use a simplified form of transcription to orchestrate their metabolism (Raven and Johnson, 2001), and it explains how an isolated cell can differentiate over time and maintain that differentiated state within a multicellular organism. However it does not explain the process of pattern formation used for regional specification in Metazoans, or what triggers the differentiation processes that lead to morphogenetic changes, including growth. To understand these fundamental processes introduced earlier requires an understanding of how information is transferred from one cell to another. Nature uses two mechanisms to achieve intercellular information transfer, *cell division* and *induction.*

Cell division occurs both during cleavage (the initial stage of cell division in an embryo) and growth. A cytoplasmic determinant is a substance which guarantees the assumption of a particular state by the cells which inherit it during cleavage (Slack, 1991). If inhomegeneities exist in the concentrations of determinants within the cytoplasm of the zygote, cleavage can

result in cells with different states. Developmental biologists believe that such determinants directly determine only the first few specified regions of the embryo and that further regionalisation involves inductive signalling (Slack, 1991). Cell division also occurs during growth. Most growth involves the division of cells with homogenous concentrations of determinants throughout their cytoplasm, yielding two daughter cells with the same state as the parent. However cell divisions that are either unequal in volume or where the plane of division is altered are both known to occur (Wolpert et al., 2002). Either of these processes can yield cells of the same lineage, but with different state.

Inductive signals are responsible for the majority of intercellular communication, not only within a developing embryo but also the communication required to regulate a developed organism. It is known that inductive signalling is achieved by the transmission of chemical substances between cells, and the direct gene products, proteins, might seem a likely candidate. However the size of a typical protein means that they are not generally capable of passing through a cell membrane. Slack (Slack, 1991) has identified four mechanisms by which chemical signals can pass into a cell. The most common mechanism is for an intercellular protein signal to bind to a receptor in the cell membrane which then activates a metabolic event within the cell that leads to the generation of a transcription factor. The second is that a signal is carried by a chemical such as a steroid that can pass through the cell membrane and directly activate an intracellular event. The third is that transcription factors interact directly before cleavage when there are no cell membranes to pass through. This strategy is used by Drosophila in early development but cannot be used once the membranes have formed. The fourth is to use a small molecule that can pass through a gap junction, a direct link between the cell walls of neighbouring cells. The existence of such varied and intricate mechanisms of communication further complicates how to model the relationship between the concentration of a transcription factor and rate of protein production discussed above. Eggenberger (Eggenberger and Dravid, 1999) noted that empirical data suggest that at least for some genes the concentration of transcription factors control the activity of a gene and the affinity between the transcription factors and the number of promoter sites controls the amplitude of the protein produced.

However the existence of several signalling strategies in Metazoans, and in particular the use of direct signalling before cleavage in Drosophila suggests that complex signalling strategies in themselves may not be an important feature of development. Instead they be used by nature simply to overcome the problem specific to the biological medium of transmission through cell membranes, meaning a detailed model of intercellular communication is perhaps not vital. Hence the initial models of inductive communication used in this thesis are extremely simple and computationally inexpensive. Chapter 5 explores in some detail how using slightly more complex modes of communication alters performance.

One aspect of the intercellular communication that has not yet been discussed is the mechanism by which a signalling molecule emitted from one cell traverses intercellular space to reach another. In nature this process is not under the direct control of the cell; rather it relies mostly on the physical process of diffusion. In the 1950s Turing considered how the patterns of animal coats might be formed (Turing, 1952). He suggested that by modelling only autocatalytic chemical reactions and diffusion it might be possible to generate stable patterns of reaction products that were similar to the markings of animal coats, with such patterns arising only from small inhomegeneities in the concentrations of otherwise uniform planes of the constituent chemicals. To demonstrate this he developed the reaction-diffusion model.

There are a number of limitations to his model. First, it essentially models gene regulation as a single chemical reaction and ignores the effects of cells, which is simplistic. Additionally as reaction-diffusion models are computationally expensive it is difficult to scale them up to use many chemicals without the model becoming intractable. However some researchers have continued to explore reaction diffusion in the context of morphogenesis (Gierer and Meinhardt, 1972; Slack, 1991; Murray, 2002). These models explain certain features of morphogenesis well, suggesting that reaction kinetics and diffusion might play an important role in intercellular communication and gene regulation.

2.2.3 Benefits of Development

There are a number of benefits that development can bring to Evolutionary Computation. These are now reviewed, paying particular attention to benefits that can be of use to hardware evolution and scalability. The GRN model that has already been introduced will be used to illustrate these benefits. However there are several other computational models of development, which are surveyed later, and most of the points made here are also applicable to other models.

Compact, Transforming Encoding

It has been discussed both above and in Chapter 1 the indirect interaction of the gene products of a GRN can be used to generate a phenotype. The most common observation made by researchers exploring development in Evolutionary Computation is that a GRN can be encoded as a fixed length genotype irrespective of the complexity of the problem at hand, thus it is possible for a short genotype to encode a solution to a problem that would require a large genotype if encoded using a traditional one-to-one mapping. (Dellaert and Beer, 1996; Kitano, 1998; Koza et al., 1999; Haddow et al., 2001). This is a valid point; however it is important to point out that the a developmental encoding is only useful in this respect if it can be guaranteed that there exists a GRN that can both be represented by the encoding and is capable of representing interactions of the complexity required to provide a solution to the problem: a complex problem will require a complex GRN to represent a solution. Additionally it was

discussed in Section 2.1 that as the search space of a traditional genetic algorithm explodes geometrically as the genotype length is increased, search space size is a valid concern. However it was also stressed that a more pressing concern for Evolutionary Computation is the evolvability of the search space. When considering both these points it is not clear what features of a problem and developmental encoding will result in a genotype that is both shorter and more evolvable that a traditional one. In some cases it may be that a larger, but evolvable genotype is of more use. As development transforms the search space to the solution space one possible benefit it could provide is to transform a space of low evolvability to one of high evolvability. How to achieve this is unfortunately still beyond the research community. However the kind of problem that development is suited to is discussed in the section on scalability later in this chapter and throughout this thesis. An example of a compact yet useful encoding is given in Chapter 4. The issue of evolvability is discussed in Chapters 4, 5 and 6.

Self-organising

Eggenberger (Eggenberger, 1997b) noted that the phenotype generated by a developmental process emerges from a self-organising system. Such systems can respond to complex dynamic changes in environment faster than traditionally designed systems (Holland, 1998).

Distributed

In nature the process of development is distributed across many cells. This can be a useful model for Evolutionary Computation as the processing required to implement a developmental algorithm can be distributed across a number of processors. It is of particular interest in hardware evolution as distributed processes translate very well to hardware. Thus the cost of employing development in addition to Evolutionary Computation can be reduced with respect to other classes of design problem.

Robust

Not only is the process of development distributed, but also *control* of development in nature is distributed, providing robustness. Eggenberger noted that developmental processes have an inherent stability (Eggenberger, 1997b). Redundancy and compensation in both GRNs and developmental signalling networks is common in many organisms (Kirschner and Gerhart, 1998; Morange, 2001). Most redundancy results because the molecules involved in these processes are fairly flexible in their stereospecificity. This allows interoperability, which can provide at least partial compensation for missing components. meaning that regulatory networks that might otherwise be separated by large areas of low fitness search space can be close together, thus potentially improving evolvability. The robustness of developmental processes in this respect has been explored by van Remortel et al. (van Remortel et al., 2002a). It has also been demonstrated that the developmental process can be used to repair damaged phenotypes (Miller, 2004), including circuits (Liu et al., 2005).

Fault Tolerant

Robustness is a feature that can be apparent not only in the developmental process itself, but also in the phenotypes that development generates. For instance functional redundancy is provided through the compartmentalisation of organisms into multiple cells. A multicellular structure can allow for neighbouring cells to provide redundancy and compensation for structural or signalling defects in a single cell, and the strategy of regulated pattern formation that development uses to orchestrate this process has been used as inspiration by a number of researchers in the development of fault-tolerant systems, in particular fault tolerant hardware (Mange et al., 2000; Canham and Tyrrell, 2003; Tyrrell et al., 2003).

Adaptive

Although development is used in this thesis as a metaphor for the map between genotype and phenotype, in nature development is a continual process that persists throughout adulthood. Kitano pointed out that developmental processes can be used to regulate certain phenotypic processes, such as metabolism, so that they are maintained between certain thresholds (Kitano, 1995). The ability of development to repair damage mentioned above is another example of how development can continue to adapt a developed phenotype.

Biologically Plausible

Dellaert (Dellaert, 1995) pointed out that because of the biological basis of developmental systems it may be possible to use the analysis techniques used by biologists to study computational models of development. Certainly models such as Random Boolean Networks (Kauffman, 1969) and reaction diffusion (Turing, 1952) have percolated into the field of computational development through biologists (Hogeweg, 2000). In addition analysis tools such as phase portraits that have been used by both biologists and in the more general area of dynamical systems theory have been used in computational development (Dellaert, 1995; van Remortel et al., 2002a).

Modular

It has already been discussed how regulated gene expression can be modelled by autocatalytic Gene Regulatory Networks (GRNs). This model is widely accepted by biologists and researchers of Evolutionary Computation alike (Kauffman, 1969; Dellaert and Beer, 1996; Morange, 2001; Bongard, 2002). Chapter 1 explained how such networks may be arranged as modules, controlled by a master control gene. When activated the master control gene causes a cascade of activity throughout a GRN module. At various stages some of the regulatory proteins that are generated during this cascade will act as transcription factors for structural genes that will directly contribute to the phenotype or enzyme genes that will catalyse reaction pathways

that generate structure. Thus a complex feature can be generated by a self-contained, modular GRN.

Iterative

If the master control gene discussed in the paragraph above is expressed multiple times (perhaps through a feedback loops in the GRN as mentioned in Chapter 1) the module can be reused, thereby generating multiple copies of a complex feature. When coupled with the spatial pattern formation strategies discussed in the Section 2.2.2 these modules can be iterated over both space and time.

Scalable

The two paragraphs above describe how modular features can be iterated across both space and time using developmental principles. If this idea was applied to the construction of a circuit rather than a biological organism, problem-specific circuit design abstractions could be learned in the form of modules generated by GRNs. As evolution maintains control over whether or not to make use of these modules, the algorithm contains only a soft bias towards using them. Thus the two problems of using top-down modules geared towards traditional circuit design, and applying such modules using traditional hard bias approaches like function level evolution are avoided. With this in mind this thesis contends that giving evolution the ability to control such a process can improve the scalability of evolution when applied to certain kinds of large problems.

Although the idea of tackling more *complex* problems may seem attractive in this context, the term *large* is used advisedly. Algorithmic complexity can be defined quantitatively by the notion of Kolmogorov Complexity (Mitchell, 1997): given a Turing machine the algorithmic complexity of a given string is the shortest program input to the Turing machine that would cause the machine to output the string in question and stop. If the output string was a circuit design, its Kolmogorov Complexity would measure how much information the design contains, hence its complexity. Unfortunately this value cannot be calculated for an arbitrary string as one cannot be sure that the program under trial is indeed the shortest possible, however it can be approximated by traditional compression algorithms such as LZ76 (Lempel and Ziv, 1976). This idea was introduced by Lehre and Haddow (Lehre and Haddow, 2003) where LZ76 was used as a measure of phenotype complexity.

The types of circuit design problem that are most likely to show an improvement in scalability through the use of development are not simply those where the circuit phenotypes have either high or low complexity but those that are *compressible*: those where the ratio of phenotype size to Kolmogorov Complexity is large. Such compressible circuit designs will be large but can be described succinctly, most likely using the concepts of modularity and reuse in the description;

hence they will be amenable to development. It is these kinds of circuits, be they ultimately complex or not, that are the direct target for this approach to scalability. For this reason, when the scalability of hardware evolution is explored in Chapter 6 the example circuit design problems are ones where traditionally-designed solutions are large, hence not easy to discover using conventional evolutionary methods, yet modular and compressible, hence not as complex as their phenotype size might at first glance suggest. Some of these ideas have recently been discussed in (Hartmann et al., 2005).

There are also likely to be circuits that are difficult to design by traditional techniques, hence might be intuitively considered complex, that are good targets for developmental hardware evolution. As was discussed in Section 2.1 Thompson has shown that there exist circuits that rely on the physics of the underlying device and the environment, and hence lie in an area of design space not accessible to traditional designers. Should it be possible to capture such features in developmental modules, evolution might be able to solve what at least on the outside seems to be a large and complex problem as it is not tractable using traditional techniques, but its theoretical compressibility is actually high. Such problems are not explored in this thesis; however they are likely to be good candidates for future research into scalability, and might be of real-world use.

2.2.4 Computational Models of Development

A number of different strategies for modelling development have been used by Evolutionary Computation researchers. Bentley divided these into two approaches, explicit and implicit (Bentley, 1999).

Explicit Development

Explicit systems evolve a genotype-phenotype mapping that explicitly specifies each step of a developmental process, much like a computer program. They must also explicitly specify any hierarchical modularity, iteration and recursion, usually by means of specially tailored programming constructs. This is the approach taken by Cellular Encoding, which was first developed by Gruau to develop ANN architectures (Gruau, 1992). More recently the same method has been used by Koza et al. to evolve analogue circuits (Koza et al., 1999). Cellular Encoding encodes each phenotype modification explicitly as a node in an acyclic directed graph. The phenotype is developed by interpreting the graph of rules one node at a time and applying the modification encoded in each node to an 'embryonic' circuit design. The graph edges define the flow of control within the development program. Gruau also used an explicit iteration construct to repeatedly activate a subtree of instructions, thus allowing iterative structures to be generated in the phenotype. Koza also made use of custom programming constructs to the same end. He introduced Automatically Defined Functions (Koza, 1994) to explicitly provide modularity, and Automatically Defined Loops and Automatically Defined

Copies (Koza et al., 1999) to provide iteration. Lohn and Columbano have used a similar method to evolve analogue circuits (Lohn and Colombano, 1998), but instead of encoding the development program in a tree they use a linear representation, which is applied to an embryonic circuit so that the phenotype 'unfolds' linearly. This means that its representational power is limited. Sims used a simple explicit system to develop 3D morphology and ANN controllers for autonomous agents (Sims, 1994). Although Koza et al. have demonstrated examples of modularisation and re-use of modules in their solutions (Koza et al., 1997; Koza et al., 2003), none of these groups has presented thorough analyses of the scalability of their models.

A problem with explicit approaches is that the mechanisms they use to develop phenotype depart significantly from biological reality where the developmental program *emerges* from the interaction of gene products. Biological evolution interacts with development by modifying gene products, thus altering the flow of control in an *implicit* manner. It is likely that this mechanism has merit in terms of evolvability: Dawkins suggested that over the course of time the developmental process has evolved to enhance its own evolvability (Dawkins, 1989), and in more recent years this idea has become widely accepted (Altenberg, 1994; Wagner and Altenberg, 1996; Marrow, 1999). Hence it likely that the successful interplay between evolution and development depends in some part on this implicit mechanism of flow of control. In particular gene interaction elegantly provides mechanisms of hierarchical modularity, iteration and recursion in way that has been honed by evolution over aeons to facilitate bottom-up evolutionary design. Unlike this, the explicit constructs discussed above have their origins in top-down programming design. As discussed in Section 2.1.2 there is no guarantee that such constructs are particularly evolvable, whether the problem is circuit design or the design of a biological organism. Unfortunately there is currently little empirical evidence to confirm the benefits of an implicit system over an explicit one. The only studies the author is aware of were carried out by Bentley and Kumar (Bentley and Kumar, 1999; Kumar and Bentley, 1999), who found that a simple implicit implementation often outperformed an explicit one when generating many simple shapes. However they also pointed out that the implicit system was biased towards generating particular classes of shapes, and did not perform well on a few small shapes. This area is certainly worthy of more research.

Implicit Development

Implicit developmental systems model the mechanisms that are used to alter the flow of control in biological development more closely. A common method of implementing implicit development is to use a production system. The idea behind this is that complex objects can be defined by successively rewriting a symbolic description of a simple object according to the set of production (or rewriting) rules. The precondition of each rule contains a symbol or a set of symbols that are matched against another set of symbols that make up the developing object.

Any matching symbols in the object are replaced with the symbols in the postcondition of the matching rule. Usually the rule set is iteratively applied to a fixed starting symbol (or set of symbols) until a stopping condition is met and the structure is considered to be fully developed. These rules can be considered as the grammar of a language, and it is the rules rather than the developing object that are evolved. Hence evolution does not directly alter either the phenotype or an explicit developmental program, thus modelling biological development more closely. Instead development is implicitly evolved as evolution modifies its grammar. The way the rules of a production system interact can be visualised by a production diagram. This is a directed graph where each node represents a symbol that is present in the rules. Edges between the nodes lead from the symbols present in the rule preconditions to the symbols in the rule postconditions. Cycles in the graph represent a form of iteration. Thus the interactions between production rules can be seen to parallel the interaction between gene products in a GRN, as discussed in Chapter 1. Production systems have been used to evolve solutions to a range of problems, from sorting networks (Belew and Kammeyer, 1993) to ANNs (Cangelosi, Parisi et al. 1994).

A class of production system designed specifically with biological plausibility in mind is the class of Lindenmayer systems (L-systems). These were proposed to model plant development (Lindenmayer, 1968), a branch of development that probably evolved independently after the evolution of the Metazoans. They have proved successful in both modelling and visualising many details of plant development and morphology (Lindenmayer, 1968; Prusinkiewicz et al., 1997). Traditional production rules systems apply rules serially. L-Systems differ from these by applying the complete set of rules in parallel at each rewriting timestep. This models the parallel division of cells in nature more closely. They have been applied to a number of evolutionary design problems, from the design of tables (Hornby and Pollack, 2001) to ANNs (Boers and Kuiper 1992) and have been proposed for evolving circuits (Kitano, 1998; Haddow et al., 2001). The canonical L-system is known as the D0L-system, or a deterministic context-free L-system. Determinism is traditionally maintained by ensuring that each symbol within the alphabet of a D0L-system has only one rewriting rule. D0L systems are context-free: the precondition of each rule always contains a single symbol. Hence each symbol is replaced deterministically regardless of the state of its neighbours. This allows D0L systems to roughly model process of cell division and generate iterative, recursive structures, such as those seen in plant morphology. However it fails to model an important aspect of Metazoan development that was discussed earlier: intercellular communication. Cellular automata (CA) models have demonstrated that simple local context can be used to perform complex global computations (Mitchell et al., 1993). Pattern formation in Metazoan development relies on local context, commonly through intercellular communication. Any developmental system that does not make use of local context is discarding the opportunity to make use of this additional computational layer, and

consequently reducing the range of developmental changes that can be affected at each iteration of the developmental process. Furthermore local context in nature is complex: it has been noted that developing biological cells often make use of intricate self-organising processes that result purely as a consequence of the physics and chemistry taking place in their local environment (or in other words their local context). Many of these processes can be considered as robust local learning mechanisms that might be of benefit to evolvability (Kirschner and Gerhart 1998). By using local context to control future development, nature incorporates these mechanisms into the developmental process to provide additional complexity with no additional biochemical cost. Thus it is unsurprising that there are few reports in the literature of context-free L-systems being used to solve engineering problems. On the other hand context-sensitive L-systems have been used, for instance for ANN design (Boers and Kuiper, 1992) and even for circuit design (Haddow et al., 2001). These use a rule structure that takes into account the state of directly neighbouring cells. One more recent and potentially useful technique is the use of parametric context-free L-systems (POL-systems) that allow external environmental parameters, one example being local context, to guide development (Hornby 2003). Although still highly abstracted from biological development, this type of L-system might well capture the features that are important for tackling the scalability problem in Evolutionary Computation.

A number of related developmental models have arisen from the study of biological systems. Many developmental systems, for instance (van Remortel et al., 2002b), use CA to model the interaction between cells. The CA rules specify how each cell should react to the states of the surrounding cells. This sort of approach has many similarities with context driven L-systems. However there are some differences. First, the product of the rule interaction is usually not a program to generate the solution, but the solution itself. This models biological systems more closely. Second, and perhaps most importantly, CA is inherently context driven. Several researchers have reported promising results using CA-based biological techniques. For instance, de Garis et al. used a CA model of biological development to develop ANN architectures (de Garis et al., 1997). However rather than evolving the CA rules they used hand-coded rules, and evolved the CA starting conditions. The use of CA to evolve hardware designs has also been advocated by Sipper et al. (Sipper et al., 1997a). However the work of this group has since taken another direction, instead for the most part exploring bio-inspired fault tolerance. Most of the more biologically plausible developmental systems that have sprung up in recent years use some form of CA to model the process, and are discussed in the following section.

Biologically Plausible Approaches

The models of development in the literature that explicitly model differential gene expression and intercellular communication are worth considering in a little more detail than the more abstract examples outlined above. They all model many developmental processes. Table 2.2.1 lists many of these models, showing which of the fundamental developmental processes,

regional specification, differentiation and morphogenesis are modelled. The table makes a distinction between three morphogenetic strategies: cell adhesion, cell division and apoptosis. Also included is a mechanism common to many of the models that concern ANN development, synapsis. This is the mechanism by which neurons become connected. This is of interest as it results in the formation of networks, much like electronic circuits. Table 2.2.1 also includes details of the regulation strategy used by each model. This is divided into two features: how a rule is activated, and the nature of its response once activated. The various types of strategy and their potential merits are explained in the following sections.

The fundamental developmental processes listed above can be considered as macroscopic processes that are each implemented by a number of microscopic strategies. It is likely that many of the microscopic strategies used by biological development result from the features of (and the restrictions imposed by) the biological medium in which they operate. This was briefly discussed earlier in the context of intercellular communication. Indeed it is also possible that some of the fundamental processes are intimately associated with the biological medium. In particular morphogenesis, the generation of form, by definition depends greatly on the physical properties of the medium. Because of this the details of biological morphogenesis are not likely to be directly relevant to hardware evolution problems, just as are any microscopic strategies that are specific to the biological medium. Hence modelling either morphogenesis or these kinds of microscopic strategies carefully might not provide a great performance benefit to hardware evolution.

However one morphogenetic process that is often overlooked and warrants further research is adhesion. In an elegant demonstration Hogeweg used the properties of adhesion to indirectly control cell growth, division and apoptosis, creating a variety of morphologies analogous with those found in nature (Hogeweg 2000).

Eggenberger also included adhesion in his model used to develop ANNs (Eggenberger 1996; Eggenberger 1997), and to explore the dynamics of regulatory systems (Eggenberger and Dravid 1999). Likewise he successfully generated a wide range of morphologies with rather complex interactions between the different types of structures, demonstrating that through his model evolution was capable of controlling the development of heterogeneous networks of interacting elements, a feature that might be useful for hardware evolution. However the results of Cangelosi et al. were not so promising, with only simple ANNs evolved successfully (Cangelosi, Parisi et al. 1994). This suggests that inclusion of adhesion in a developmental model may introduce problems as well as benefits. Unfortunately the three models differ in so many respects that further research is necessary to determine whether adhesion could benefit the development of hardware.

Reference	Regional Specification	Differentiation	Morphogenesis			Syn-apsis	Activation Strategy	Response Strategy
			Division	Adhesion	Apoptosis			
(Fleischer and Barr, 1993)	X	X	X	X	X		Boolean	Graded Nonlinear
(Kitano 1994)	X	X	X				Boolean	Graded Nonlinear
(Kitano 1995)	X	X	X		X	X	Boolean	Graded Nonlinear
(Cangelosi and Elman 1995)	X	X	X			X	Boolean	Graded Linear
(Cangelosi 1999)	X	X	X				Boolean	Graded Linear
(Jakobi 1995)	X	X	X	X			Boolean	Graded Nonlinear
(Dellaert and Beer 1996) (Model a)	X	X	X	X		X	Boolean	Boolean
(Dellaert and Beer 1996) (Model b)	X	X	X			X	Boolean	Graded Linear
(Eggenberger 1996)	X	X	X	X		X	Graded Linear	Graded Linear
(Eggenberger 1997)	X		X		X		Graded Linear	Graded Linear
(Eggenberger and Dravid 1999)	X	X					Boolean	Graded Nonlinear
(Eggenberger 1997)	X	X	X	X	X	X	Boolean	Graded Nonlinear
(Eggenberger 2000)		X				X	Boolean	Graded Nonlinear
(Savill and Hogeweg 1997)		X		X			Boolean	Boolean
(Hogeweg 2000)	X	X	X	X	X		Boolean	Boolean
(Bentley and Kumar 1999)	X		X				Boolean	None
Kumar and Bentley 1999	X		X				Boolean	None
(Kumar and Bentley 2000)	X		X				Boolean	None
(Bentley, 2003)							Boolean	Pseudo-Nonlinear
(Kumar and Bentley 2003)	X	X	X		X		Graded Linear	Graded Nonlinear
(Bentley 2004a)		X					Boolean	Graded Nonlinear
(Bentley 2004b)		X					Boolean	Graded Nonlinear
(Edwards, 2002)	X	X	X				None	Boolean
(Bongard 2002)	X	X	X				Boolean	Graded Nonlinear
(Miller and Thomson 2003)	X	X	X				Boolean	None
(Miller 2003)	X	X	X		X		Boolean	None
(Tufte and Haddow 2003)	X	X	X				Boolean	None
(Federici, 2004)	X	X	X				Graded Nonlinear	None
(Roggen et al., 2004)	(X)	X	X				None	None

Table 2.2.1: Processes used by some models of development.

Unlike morphogenesis, regional specification and differentiation, brought about by gene regulation and intercellular communication are both likely to be central to the generation of large, complex structures whatever the medium in which they are employed because they are vital to the mechanisms development uses to generate modularity and design reuse, as was discussed earlier. These are two features that are crucial to the argument made here as to how development can aid scalability.

It is also possible that some microscopic strategies have been employed by biological evolution not because they are the best solution to a particular problem, but because they provide an additional benefit by efficiently reusing some of the biochemical machinery that had already evolved before the advent of multicellular organisms. These strategies are again unlikely to be of direct use in the context of evolving hardware. However many microscopic strategies are likely to have evolved simply because they are the best solutions that evolution has discovered for the problem at hand. It is these strategies that are worthy of inclusion in models of development applied to hardware evolution.

The biologically plausible models of evolution that are under review here are now categorised in terms of how they model the central microscopic strategies of differential gene activation and intercellular communication. The gene regulation processes are further broken down into (a) the basic regulatory elements used in each model, (b) the strategy used to match the presence of these elements to rules, thus activate the rule, and (c) the strategy used to respond to rule activation. Analogously communication processes are broken down into (a) the basic elements of transmission, (b) the strategy used to transmit information between cells and (c) the strategy used to receive signals. Although this discussion reveals some hints as to what strategies are suitable for application to engineering problems and hardware evolution in particular, it will also show that few comparisons between developmental models have been presented in the literature, and that such comparisons would be beneficial for the progression of the field.

Regulatory Elements

Most of the models listed in Table 2.2.1 model proteins as their regulatory element, and these can be separated into regulatory proteins that interact with genes, and problem-dependent structural proteins that are used to alter the phenotype in some way. The exceptions are (Bentley and Kumar, 1999), (Kumar and Bentley, 1999; Kumar and Bentley, 2000), (Tufte and Haddow, 2003b), (Edwards, 2002) and (Miller and Thomson, 2003). The regulatory elements in these models are state variables derived from the phenotype, and these are used to alter the phenotype directly using this information. The state variables used by most of the models describe the functional behaviour of physically neighbouring cells. However the system used by Miller and Thomson in (Miller and Thomson, 2003) is the more complex. Miller's phenotype is a

feedforward logic design. Each cell in the phenotype performs a logical function, and is connected to two other cells. The state information provided to each cell consists of the positions in the phenotype of the two cells to which it is connected, the current function of the cell itself and the current position of the cell itself within the phenotype. Hence unlike the other models the information a cell uses to regulate itself is not necessarily all local in space. Instead it uses the connectivity, which can alter throughout development, to define its neighbourhood.

Although the proteins used in the remaining models of Table 2.2.1 can also be thought of as part of the phenotype they have been introduced purely to facilitate development, and their nature and the nature of their interactions are under control of the designer. (The bias imposed by the examples above is instead determined by the nature of the phenotype.) This could be beneficial if the environment that development experiences is sufficiently rich in useful behaviours, and of course sufficiently evolvable to allow evolution to exploit such behaviours through development. This is not unlikely as it might first seem. As mentioned earlier, biological development relies on the complexities of the environment to perform certain complex tasks. Examples are the formation of a cell membrane and chromosome orientation during mitosis (Kirschner and Gerhart, 1998). In the case of intrinsic hardware evolution, evolution has already been shown capable of exploiting complex physical processes when traditional constraints are relaxed, and this has resulted in rich yet evolvable solution spaces (Thompson and Layzell, 1999). However the systems above only allow very simple interactions between phenotypes and development, as the phenotypes themselves are strictly constrained to logical behaviour. While there might be some advantage to their use it seems unlikely that such interactions would yield enough complex yet useful behaviour to provide development a dramatic performance advantage should it make use of them.

It is not yet well understood how development can be harnessed for engineering purposes. Until this is the case, the study of systems based on simple virtual proteins is likely to be of most use to the field as not only is the bias they impose on the model less complex thus more easily analysed, they are independent of the nature of the phenotype hence of more general interest and more flexible if modifications are required during the course of research. This is likely to remain the case until rich behaviours, for instance those evident in low abstraction hardware evolution can be harnessed for this purpose successfully. Perhaps in view of this Miller's later work introduced virtual proteins to his model (Miller, 2003).

The remaining systems in Table 2.2.1 use only artificial proteins to define the environmental information used by developmental rules. Because of this they are likely to provide a better insight into what features in general are needed to model gene regulation in a way that can be exploited for engineering purposes.

These systems can be divided into those that predefine proteins as either structural or regulatory proteins, and those that allow all proteins to perform both structural and regulatory functions. Examples of systems where proteins can assume both regulatory and structural roles are (Kitano, 1995), (Dellaert and Beer, 1996), Hogeweg (Hogeweg, 2000) and (Bentley, 2004b). Examples of systems where a distinction between regulatory and structural proteins is made are (Eggenberger, 1996; Eggenberger, 1997a), (Cangelosi et al., 1994), (Bongard, 2002), (Kumar and Bentley, 2003), (Jakobi, 1995) and (Bentley, 2003; Bentley, 2004a). Many of these systems subdivide their regulatory and structural protein classes further into even more specific roles.

The benefits of using proteins that serve a dual structural and regulatory role are unclear as such a comparison has not been made in the literature. It might allow a reduction in the size of the search space as less proteins are needed to specify a particular developmental pathway, hence the length of the rules that encode protein descriptions can be reduced. However the additional constraint this imposes on the roles of particular proteins is likely to increase the ruggedness of the evolutionary fitness landscape, perhaps to an extent that the reduction in evolvability outweighs the benefits of a smaller search space.

Kitano's ANN protein model is more complex than most of those described above (Kitano, 1995). He models two classes of regulatory elements: regulatory proteins that activate genes and enzyme proteins that define reactions between the regulatory proteins, thus defining an artificial chemistry. This might be of great benefit: it was discussed earlier how development can harness complexity in the environment to gain expressiveness at little cost. A complex artificial chemistry is one way in which this could be achieved. Indeed Kitano managed to generate quite large collections of neurons (but did not evolve these to achieve any useful task). However any chemistry that is defined would need to be problem specific, embodying complex functions that are useful for the generation of a particular type of structure. If functions that are useful for the development of the phenotype are not inherent in the environment itself, this might result in an unnecessary computational expense. Other models that have made use of simpler artificial chemistries are Bentley's fractal protein models (Bentley, 2003; Bentley, 2004a) and Kumar's AES (Kumar, 2004).

Activation and Response Strategies

All the models in Table 2.2.1 use rules to perform gene regulation. Typically the environment of each cell, defined by the regulatory proteins present (or other local phenotypic state information) is compared in some way to the conditions specified in each rule. If the environment matches a rule precondition then the rule is activated. Once a rule is activated it

responds by altering the environment in some way, usually by altering the concentration of a protein, or some other state variable if the system does not model proteins, as discussed earlier.

The mechanism by which rules change the state of the cellular environment is extremely important to development. Successful evolution requires the ability to make small changes in genotype, and that they are reflected by correspondingly small changes in phenotype (Kauffman, 1993). If by altering a rule, the expression of a protein changes radically, the effect on global behaviour is also likely to be altered radically, leading to a solution space with poor evolutionary characteristics. Hence it may be useful to have a model where the overall effect that activating a rule has on its surroundings is graded rather than Boolean.

As can be seen in Table 2.2.1 many of the models under discussion provide a mechanism for evolution to make small changes to phenotypes either through a graded model of rule activation, a graded model of rule response, or both. It is interesting to note that many of the models that only provide a Boolean overall response have been reported to perform poorly in many respects. Bentley and Kumar reported problems with evolvability (Bentley and Kumar, 1999; Kumar and Bentley, 1999; Kumar and Bentley, 2000). Similarly Tufte and Haddow's system did not successfully evolve even quite simple circuits (Tufte and Haddow, 2003b). The model used by Miller and Thomson fared slightly better, but only managed to evolve simple circuits: one bit adders and small parity generators (Miller and Thomson, 2003).

However it should be noted these also happen to be the models that use information from the phenotype rather than artificial proteins to determine a cell's context. Bentley and Kumar's rule precondition specifies a conjunction of neighbouring cell states and/or don't care terms (Bentley and Kumar, 1999; Kumar and Bentley, 1999; Kumar and Bentley, 2000). Tufte and Haddow use a similar mechanism (Tufte and Haddow, 2003b). In (Kumar and Bentley, 1999) and (Bentley and Kumar, 1999) Bentley and Kumar also include information from two predefined protein gradients that are present across the cell array. As they still reported poor results, the lack of explicitly modelled proteins alone cannot be used to explain the poor performance of these systems, and other factors such as the use of Boolean regulation strategies are likely to be at least partly responsible. Miller and Thompson used a much more complex regulatory strategy, although it was still Boolean (Miller and Thomson, 2003). The genotype was effectively a single rule consisting of a logical function of the cell's state inputs, which included the position in the phenotype of the cells connected to it, represented using a Cartesian Genetic Program (CGP) (Miller and Thomson, 2000). It also doubled as the logic of the cell in the phenotype. Hence as evolution altered this rule it not only altered the regulatory function but also the function and inputs to the phenotypic cell. This means that the overall regulatory strategy is Boolean and dynamic, and the behaviour of the phenotype interacts intimately with the

genotype-phenotype map produced by development. The level of interplay means that both search and solution spaces are dynamic and nonlinear, and highly constrained by each other. Hence the limited success reported might owe more to evolution struggling to deal with the complexity of this model than the nature of the activation and response strategies.

Miller's later work on the evolution of pattern formation retained the use of a Cartesian Genetic Program acting as a single rule. However the inputs to this program consisted of not only the cell states of local neighbours but also a bitstring representing the concentration of a single protein at various points around the cell neighbourhood. This is unlike Miller's earlier model, which determined its neighbourhood in a dynamic, nonlinear manner through the current connectivity of a cell in the phenotype. The genetic program also had a number of outputs which determined the new chemical level of the cell, the new cell type of the cell and growth instructions. Thus like his earlier model the regulation strategy was Boolean. Miller showed that this new model performed well on pattern generation problems. However he has not presented any results using the problems he applied to his earlier model, hence it is difficult to determine whether the use of a dynamic neighbourhood determined by development was responsible for the failure of his earlier model to evolve all but the simplest circuits. Furthermore, Federici has presented a developmental model almost identical to Miller's using an ANN rather than a CGP for regulation, thus providing a graded nonlinear response. However he has not compared the two systems directly. This is unfortunate as should the Boolean regulatory model not have a particularly adverse effect, Miller's CGP-based model of regulation would be a promising candidate for application to hardware problems, as it is extremely compact and easy to implement in hardware. Such comparisons might also add further insight into whether explicit use of a protein model, the use of Boolean regulation strategies or both are detrimental to the performance of developmental systems.

In (Edwards, 2002) Edwards introduced an interesting model that used similar environmental context to that of Tufte and Haddow, but ensured that activation always occurs thus avoiding the need to select an activation model. The response was selected by measuring the Hamming distance between the context of the current cell and a set of predefined contexts that were mapped to future cell states using a lookup table. The response with the closest context in the lookup table was selected. This ensured the formation of autocatalytic sets of contexts, but again the benefits of such an approach are unclear as there has been no direct comparison between this and other models.

Several researchers have used a more complex graded model of gene regulation, where rules provide a linear relationship between changes they experience in their environment and the changes they generate. Examples are Eggenberger's early work (Eggenberger 1996;

Eggenberger 1997), the more complex of Dellaert and Beer's models in (Dellaert and Beer, 1996), and the work of Cangelosi and Elman (Cangelosi and Elman, 1995). Again it is difficult to compare these systems with each other, or against those that produce different regulation relationships because of the range of different problems with which they were demonstrated and the number of features by which all the models differ. However a few qualitative comments can be made. Unlike any of the Boolean regulatory models discussed above, Eggenberger managed to generate a wide range of morphologies that interact in complex ways. However both Cangelosi and Elman and Dellaert and Beer could only generate phenotypes that appear to be much simpler. Furthermore Dellaert and Beer found that the simpler of the models they explored, based on Kaufmann's Random Boolean Network (RBN) model (Kauffman, 1969) actually produced better results. As this provides Boolean regulation it might appear to throw earlier discussion into doubt. However when moving to the more complex system Dellaert and Beer had also altered their model of synapsis, and put the poorer performance of the more complex system down to this change. A similar argument could be made for Cangelosi and Elman's system.

One further general feature that might provide a performance benefit to gene regulation models can be identified. Kauffman suggested that many natural complex systems exist close to the boundary where their behaviour becomes chaotic (Kauffman, 1969). Along with a number of other researchers interested in complexity, for instance (Edwards, 2002; Wolfram, 2002), he suggested that these highly complex systems could be exploited for engineering purposes. Hence it is proposed here that in addition to the benefits that arise from using a graded activation and/or response strategy, the inclusion of some aspect of nonlinearity in the model may be beneficial as it might result in a more expressive system capable of generating more complex phenotypes.

A number of researchers have taken this approach, beginning with Eggenberger (Eggenberger and Dravid, 1999; Eggenberger, 2000). He used a simple Boolean activation strategy combined with a graded nonlinear response strategy: the response of a rule was determined by a continuous sigmoid function of the number of proteins that achieved a predefined activation threshold. Hence the affinity between environmental proteins and rules determined whether or not there was a response, but the number of activation sites determined the amplitude of the response. This model produced some of the most impressive results to date in the field, both modelling biological pattern formation closely and generating complex networks of neurons. Kumar's EDS system also used a sigmoid response function, but in conjunction with a linear activation strategy (Kumar and Bentley, 2003). With this he managed to control the generation of complex three dimensional arrangements of cells, once again suggesting that there may be benefits to using this type of regulation.

Bongard used a similar method to generate GRNs. However the response model was genetically determined (Bongard, 2002): each cell used a real-numbered variable to track the current concentration of each protein. Once a rule had been activated the rate of the single gene product that was encoded in the rule was increased by an amount specified in another real value encoded in the postcondition.

Bentley extended the concept of affinity and protein interaction used by Eggenberger in his fractal protein model (Bentley, 2003; Bentley, 2004a). Here proteins were represented by subsets of the two dimensional Mandelbrot fractal pattern (Mandelbrot, 1982), the motivation behind this being to allow an extremely rich environment for complex interactions between the proteins. Each rule contained a promoter site that defined a fractal protein, encoded by a real-valued triplet. The first two were x and y co-ordinates that defined the centre of the fractal subset that represents the protein. The third defined the length of a one side of a square that marks the size of the subset. The affinity between a rule and the local internal environment of the cell, called the cytoplasm in Bentley's model, was calculated by merging the proteins present in the cytoplasm: a series of points on both the merged protein pattern and the fractal defined by the promoter were sampled and the differences were summed. In early work (Bentley, 2003) this value was compared against a real-valued affinity threshold that was encoded within the rule. If the threshold had a positive value the rule was activated when the summed difference was equal to or lower than the threshold. If the threshold was negative the rule was activated if the difference was greater than the absolute value of the threshold. (This models rule repression.) Thus activation was Boolean. Later work used an original mechanism that approximately modelled a graded nonlinear response: the response was triggered with a probability that was a sigmoid function of the difference between the combined environmental fractal proteins and the affinity threshold defined in the rule. However experiments revealed that GRN formation was difficult and so a full model of protein concentrations was introduced in a manner similar to Eggenberger (Bentley 2004). The difference in performance between the pseudo-graded and truly graded models again suggests that graded regulatory behaviour can be beneficial.

A later fractal protein model introduced additional proteins introduced three additional sets of proteins, receptor proteins, environment proteins and behavioural proteins (Bentley, 2004b). These allowed the GRN to be influenced by and affect the external environment, thus control a wall-avoiding robot. Environment proteins were merged using the method described above. The merged protein was then masked by a cell receptor protein generated by a receptor gene before being merged into the cytoplasm. Behavioural genes generated behavioural proteins that were mapped to the robot's actuators.

Jakobi also used a Boolean activation strategy based on affinities, where a rule was activated by matching a section of an environmental protein to a section of the rule (Jakobi, 1995). However he used an extremely complex response strategy: a protein was represented as a string. To calculate a response the ends of the protein string were joined to make a loop and the three characters opposite the match site were matched against an arbitrary protein from a template class. The amplitude of the regulatory response was determined by this match. Jakobi could not evolve anything more than extremely simple ANNs. This is perhaps unsurprising as the model was extremely complex, using a wide variety of proteins that interacted in an effectively arbitrary manner.

Some researchers have looked to model biological development in even greater detail than those models above. For instance Kitano introduced an extremely sophisticated regulatory strategy that used both differential gene activation and an evolved artificial chemistry (Kitano, 1995). Each rule precondition specified a single regulatory element, an activation/inhibition bit, and a concentration threshold which was used to model a Boolean activation strategy. The rule postcondition defined both an explicit regulatory response and an artificial chemistry that generates an implicit response (Kitano, 1995). It specifies a single enzyme protein that is generated when the rule is activated, and two additional regulatory proteins. Should these two regulatory proteins be present in the environment, the enzyme catalyses a reaction between them. Thus the artificial chemistry is defined. The production of the enzyme is nonlinear, depending on a reaction-diffusion equation based on the Michaelis-Menten equation (Kaneko and Yomo, 1994). Similarly Fleischer's model implemented a continuous response strategy based on custom reaction diffusion functions, but did not involve the use of an artificial chemistry (Fleischer and Barr, 1993). Although these models might provide interesting insights into the benefits of continuous models, again there have been no comparisons presented in the literature that determine the effect of their introduction. As such models are computationally expensive the study of simpler models is likely to be more fruitful, particularly if the developmental model is to be implemented in hardware.

At many points in the discussion above it was noted that there is a lack of evidence in the literature that can be used to compare developmental models, highlighting the need for the community to develop a set of benchmark problems. Such problems could be used support any incremental changes made to a model or allow comparisons between similar models already used. Fortunately this has begun to be recognised by other researchers, with some proposing benchmark problems (Stanley and Miikulainen, 2003; Federici, 2004) and some presenting more careful comparative studies (Stanley and Miikulainen, 2003; Federici, 2004; Roggen and Federici, 2004). However more research in this area is vital.

Communication Elements

The mechanisms that developmental models use to pass information between cells in is likely to be important to their performance. Information must be passed freely enough to allow evolution to explore a wide range of phenotypes made up of many cells. However if information passes too freely it might be difficult for evolution to control development in a way that a number of stable environmental contexts can co-exist. Earlier two general methods of communication were introduced: cell division and cell induction. Almost all of the systems shown in Table 2.2.1 model cell division in some way. However the majority of explicit information transfer in these models is achieved through induction. Inductive transmission and reception strategies can be used to modulate how information moves through the phenotype thus provide a bias towards the generation of certain types of pattern, and it is these strategies that are discussed in this and the following two sections.

The systems listed in Table 2.2.1 that use local phenotype state rather than proteins to model regulation, (Bentley and Kumar, 1999), (Kumar and Bentley, 2000), (Tufte and Haddow, 2003b), (Edwards, 2002) and (Miller and Thomson, 2003), do not make use of any explicit communication strategies. Instead all state information used in the models is communicated directly to nearest neighbour cells. Hence they do not apply any additional communication bias and will not be discussed further.

The elements used for inductive communication by all the remaining systems listed in Table 2.2.1 are proteins that are generated by the activation of a rule. Of these, some researchers make no distinction between communication and regulatory proteins; all can be involved in both tasks. Examples in this class are Bongard (Bongard and Paul, 2000), Kitano (Kitano, 1995), Jakobi (Jakobi, 1995) Eggenberger (Eggenberger, 1996; Eggenberger and Dravid, 1999) and Miller (Miller, 2003).

An approach that models biological development more closely is to assign some proteins the task of regulation alone, with the rest either assigned to purely communication tasks or both communication and regulation. Both Hogeweg (Hogeweg, 2000) and Dellaert and Beer (Dellaert and Beer, 1996) predefine a number of the RBN nodes to act as both regulatory and communication proteins. The remainder of the nodes could only be involved in regulation. Similarly Fleischer and Barr (Fleischer and Barr, 1993), Dellaert and Beer (Dellaert and Beer, 1996) and Cangelosi et al. (Cangelosi and Elman, 1995; Cangelosi, 1999) assigned a subset of proteins to communication, but in these cases their models defined proteins explicitly rather than using an RBN.

There is little empirical evidence to suggest that this approach is advantageous, as models using both approaches have been found to perform both reasonably well and poorly. Until more thorough empirical evidence from the direct comparison of the two approaches is available it would seem sensible to use the most convenient method for the model at hand.

Transmission Strategies

Just as in nature, the details of the transmission strategies used by the models listed in Table 2.2.1 vary greatly. However they can be placed into one of three categories. The first class contain models that simply transmit information to their nearest neighbours. One can think of this class as modelling communication through gap junctions, or perhaps some form of short range diffusion. The second class are those that transmit information to a larger neighbourhood which is either defined by a range parameter or by a simple time-independent diffusion model. Finally there are those that model the process of diffusion carefully as a dynamic process.

Example of the first class include Hogeweg's (Hogeweg, 2000) and Dellaert and Beer's RBN-based systems (Dellaert and Beer, 1996), which both simply extend the RBN model to allow edges to pass between cells. These edges are logically ORed with the corresponding edges output from neighbouring nodes and the result is stored in an environmental state register. Edges then pass from the register to nodes within the cell. Dellaert and Beer's genome-cytoplasm model also used a nearest-neighbour transmission of proteins, but through an explicit protein model rather than an RBN. These models are computationally inexpensive and are the simplest to implement in hardware as the limited routing resources needed to transmit information to nearest neighbour cells are likely to be available. However they are likely to suffer if information needs to be transmitted over a large distance as the information must be explicitly processed and retransmitted by each cell it passes through.

Eggenberger's early work (Eggenberger, 1996; Eggenberger, 1997b) is a good example of the second class of transmission strategies. It used a range value encoded in each protein (and under genetic control) from which the transmission boundaries of a protein could be calculated. However these were used with a time-independent diffusion model to generate a graded concentration level. Other researchers such as Jakobi (Jakobi, 1995) have used a time-independent diffusion function alone to define concentrations at each developmental timestep. Such models have the capability of transmitting information over large distances without placing additional constraints on the developmental process that is to be evolved, but are slightly more computationally expensive, and if implemented in hardware would require substantial routing resources or some form of bus arbitration to connect all cells capable of communicating directly. However when routing resources are plentiful or when the

developmental model is implemented in software, such approaches might be a reasonable compromise in terms of computational expense and design complexity.

The majority of the models listed in Table 2.2.1 use a much more accurate but computationally expensive model of diffusion. For instance Kumar's EDS system, which was designed to be biologically plausible, uses a time-dependent diffusion model based on a Gaussian function (Kumar and Bentley, 2003). Although such models are likely to be of interest to biologists there are no immediately obvious benefits to their use for engineering problems, and there are no comprehensive comparative studies of different transmission models in the literature. The models used by Bongard (Bongard, 2002) and Miller (Miller, 2003) use a slightly simpler time-dependent linear diffusion model. Other researchers have used even more complex global differential equations to define the rate of diffusion with respect to time. Examples of this are Eggenberger's later work (Eggenberger and Dravid, 1999; Eggenberger, 2000), Kitano (Kitano, 1995) and Fleischer and Barr (Fleischer and Barr, 1993), the latter two examples including diffusion both into and out of cells. Due to the complexity involved with solving reaction diffusion equations, Kitano modelled the process at the scale of groups of cells, rather than single cells. Although models of this kind are unlikely to be suitable for future hardware implementation, such an approach might reduce computational expense enough for reaction diffusion models to be viable.

Reception Strategies

Two approaches to reception strategy can be found in the literature. In the first approach, a simple reception strategy is used where each cell interacts directly with any of the proteins that exist in the environment. For instance Hogeweg (Hogeweg, 2000), Dellaert and Beer (Dellaert and Beer, 1996), Bongard (Bongard, 2002), Jakobi (Jakobi, 1995), Miller (Miller, 2003) and Fleischer and Barr (Fleischer and Barr, 1993) all use simple reception strategies where any incoming protein can interact directly with the cell's developmental rules. As mentioned earlier, this approach is similar to that used by Drosophila in the early stages of development (Raven and Johnson, 2001).

Other models use a more complex strategy, modelled on other modes of communication found in biological development. These require communication proteins entering a cell to interact with another element that can be likened to a receptor protein in a cell membrane. Receptor proteins may be useful as they allow cells to ignore specific inductive signals, thereby acting as a filtering mechanism. This approach was used by Cangelosi et al., where rules could generate both receptor and signal proteins (Cangelosi et al., 1994; Cangelosi and Elman, 1995). Eggenberger took a similar approach (Eggenberger, 1996; Eggenberger, 1997a; Eggenberger, 1997b; Eggenberger and Dravid, 1999), as did Dellaert and Beer with their genome-cytoplasm

model (Dellaert and Beer, 1996) and Kumar with his EDS (Kumar, 2004). Kitano did not impose an explicit reception strategy but used an active transport term in the differential equation that models transmission to modulate reception (Kitano, 1995).

The filtering mechanism that can be provided by a reception strategy could be used to implement 'don't care' terms where the presence or absence of information from neighbouring cells is ignored. This could allow any implementation of 'don't care' terms encoded in the developmental rule to be avoided, thereby reducing the size of the rule hence the size of the evolutionary search space. However it was argued earlier that the use of receptors in nature might owe more to the biological medium than any advantage they have over simpler strategies. Unfortunately it is unclear from current research whether more complex reception strategies are of any benefit, owing to the lack of comparative investigations.

Just as with the review of regulation strategies, this review of communication strategies has highlighted this area as one ripe for more thorough comparative research. The current models used vary widely and comparative analyses of their models are few and far between. Such studies are necessary if the field is to move forward. As mentioned earlier, Chapter 5 of this thesis provides a step in this direction through comparative studies between various models of communication. Chapter 6 builds on this by comparing the most powerful model of development used in this thesis with those used by Roggen (Roggen and Federici, 2004), Federici (Federici, 2004) and Miller (Miller, 2003).

Synapsis

Although the majority of developmental processes and strategies that are potentially useful for the evolution of hardware have been discussed, one that has not been touched upon is synapsis. Synapsis, the formation of connections in neural networks, is modelled by many of the systems in Table 2.2.1, and is interesting from the point of view of this thesis as it may provide a biologically plausible (hence evolvable) method of evolving connections in circuits. For instance Dellaert and Beer used two different models of synapsis for the development of ANNs (Dellaert and Beer, 1996). In their simpler model, they defined two structural proteins, one that caused cells to differentiate into synapsis targets and one that caused cells to differentiate into neuron sources: at the end of development any cells that generated either of these during development were set either as a synapsis target or a neuron source respectively. At the end of development all synapsis neuron sources are connected to all synapsis targets within a predefined range. This mechanism provides an interesting approach to connecting networks of elements, which might be useful for hardware evolution. However when working with hardware that imposes a fixed routing architecture, such as most intrinsic evolutionary platforms, this kind of model would be difficult to implement: it would require some automatic routing algorithm

such as that incorporated into the POEtic architecture (Tyrrell et al., 2003; Roggen et al., 2004), and a mechanism to either avoid or deal with circuits that cannot be implemented. In Dellaert and Beer's more complex model called the genome-cytoplasm model (Dellaert and Beer, 1996) the presence of axon proteins causes an axon to begin or continue to grow over the neighbouring cell with the highest concentration of cell adhesion molecules. Synapsis occurs only when an axon reaches a cell that is producing a synapsis protein. This kind of mechanism is much more suited to hardware evolution as the model could take into account the routing resources available as development proceeds. Unfortunately Dellaert and Beer found that the genome-cytoplasm model was much less evolvable than their simpler model. Whether this was caused by the synapsis mechanism cannot be determined from their reported work, as the models are very different, and so individual strategies within the models cannot be compared. Exploring the benefits of synapsis using an architecture such as POEtic (Tyrrell et al., 2003) would be a worthwhile avenue for future research.

2.2.5 Development and Hardware

The previous sections have surveyed computational models of development. The following two sections place the work of this thesis in the context of other research using ideas from development for hardware design and synthesis. This section reviews general work on hardware inspired by developmental processes. The following section focuses specifically on the use of development in hardware evolution, paying particular attention to work that has explored the issue of scalability.

Although the field of computational development is relatively young, biological development has inspired hardware designers for over a decade. Perhaps the first device inspired by development was Firefly, a hardware implementation of a one dimensional non-uniform cellular automata (CA) machine used to evolve solutions to the synchronisation task (Sipper et al., 1997a) that requires intercellular communication analogous to gene regulation in development. Although this was an inspiring demonstration, the inflexibility and small size of the platform means that it is not suitable for studying the evolution of more useful circuits or more complex models of development.

Embryonics

The most ubiquitous example of hardware inspired by development is the embryonic array. The term 'embryonics' was coined by de Garis (de Garis, 1993) to denote electronic designs motivated by embryology, but it was the Embryonics project (Mange et al., 1995) that first set out to explore the concept thoroughly. This project centred on the creation of fault-tolerant hardware by modelling how nature achieves fault tolerance through self-replication and self-repair using developmental processes. The key analogies between embryonic circuits and biological organisms are that the design and implementation of a circuit can be separated much

like the genotype and phenotype of an organism, that circuit implementations can be partitioned into the configurable blocks of an FPGA mirroring cells in biological organisms, and that each block contains a copy of the entire circuit design, much as biological cells contain an entire copy of the genotype. Designs are mapped to a specialised FPGA using a highly abstracted analogy of differentiation in development, where the function of each FPGA block is determined by its context. This context takes the form of co-ordinates provided by neighbouring blocks. A block uses this information to determine its own co-ordinates, and configures itself using data from the genotype that describe the cell function at those co-ordinates. Each FPGA block usually also contains self-test logic. If a block detects that it is faulty it no longer calculates its own co-ordinates but becomes 'transparent', essentially passing incoming co-ordinates directly to its neighbours. This changes the neighbouring blocks' context, which will result in them essentially assuming the function of the original cell, although replacement strategies have been investigated to explore the tradeoff between logic complexity and reliability (Ortega and Tyrrell, 1999). Embryonic arrays have been implemented using reconfigurable devices (Ortega and Tyrrell, 2000; Tempesti et al., 2002), but only recently have the ideas that came out of this work been implemented in dedicated hardware, in the form of POEtic tissue (Tyrrell et al., 2003).

POEtic tissue embodies and extends the concepts of embryonic systems. It is a flexible reconfigurable architecture that incorporates features to allow the exploration of the interplay between evolution, development and lifetime learning, on-line and in hardware. Its organisation is based on the POE model introduced by Sipper et al. (Sipper et al., 1997b) that partitions bio-inspired hardware systems along three axes: phylogeny, or evolution (P), ontogeny, or development (O) and epigenesis, or lifetime learning (E). Hence it is a three-layered architecture, implemented on a specialised FPGA that has been developed for the purposes of the POEtic project (Tyrrell et al., 2003). The three layers of hierarchy are a genotype layer, a mapping layer and a phenotype layer. As these are software-configurable they allow exploration of different models for the layers, or even different combinations of the layers, directly in hardware. By basing all models on a common underlying architecture comparison of different models is facilitated.

Work involving the mapping layer where developmental models are implemented is of most relevance to this thesis. Two mapping layers have been explored, one a co-ordinate system that mirrors earlier embryonic work (Tempesti et al., 2003), and one where context is provided by the concentration of proteins that diffuse through the system (Roggen et al., 2003a). This mapping allows POEtic to explore much more than the fault tolerance applications of earlier embryonic work. Unlike the co-ordinate system a single context can occur in multiple positions within an array. This means that a set of rules that determine cell function from a particular

context can be evolved, in a fashion similar to the implicit developmental models described in Section 2.2.4. Evolved circuits can also make use of a novel automatic routing algorithm that is incorporated in the FPGA hardware itself (Thoma et al., 2003), but as all routing in the FPGA is multiplexer-based, contention cannot arise even if the algorithm is not used.

This mapping architecture has been used to evolve the development of adders, multiplexers and abstract patterns (Roggen and Federici, 2004; Roggen et al., 2004). However the model does not allow any interaction between the proteins that diffuse between cells. It was discussed above that interaction of regulatory elements is an important feature of development that provides a mechanism for computation, which can be used to increase the complexity of developed structures. So although the mapping has been used to evolve simple circuits, more complex designs might be difficult to achieve. Of course because of the versatility of the POEtic model, a comparative study of the affects of introducing interaction between regulatory elements should be possible by simply replacing the current mapping layer.

The POEtic project represents a major thread of current bio-inspired hardware research. The POEtic architecture is well suited for exploring developmental models for hardware evolution and the project has produced many interesting and valuable insights into the interplay between evolution, development and lifetime learning. However the majority of the project output has either explored the engineering issues associated with the hardware implementation of evolutionary (Thoma and Sanchez, 2004; Tyrrell et al., 2004), developmental (Tempesti et al., 2003; Thoma et al., 2003) and learning (Eriksson et al., 2003; Torres et al., 2004) mechanisms, demonstrated the how the architecture can be used to implement features such as fault tolerance (Tempesti et al., 2003) and fault detection (Canham and Tyrrell, 2003), or demonstrated applications such as autonomous agent design (Krohling et al., 2002; Roggen et al., 2003b) and audio modelling (Cooper et al., 2004). To date it has produced little work exploring the use of development for the evolution of circuits at the granularity of traditional electronic components, beyond the work of Roggen et al. outlined in the paragraph above and the work of Liu et al. discussed in the paragraphs below. However there is a growing body of work that has. This is surveyed in the following section.

Development and Hardware Evolution

A number of other researchers have also explored hardware evolution using development. A common theme throughout the literature is the lack of any evidence that development can actually generate large circuits with repetitive modular features. In fact the successful evolution of anything more than the most trivial circuit using development is rare. Perhaps this is the reason that there have been no comparative studies investigating the scalability of hardware evolution using both developmental and direct mappings.

One of the earliest (and more successful) efforts evolving developmental rules to generate hardware designs was conducted by Hemmi et al. where trees of hardware description language (HDL) circuit descriptions were evolved (Hemmi et al., 1995; Hemmi et al., 1996). The HDL described circuits at the register transfer level and provided arithmetic operators that were mapped automatically to units such as adders and accumulators during synthesis. The trees were generated by applying a subset of the production rules that defined the HDL to a simple starting axiom, then evolved using genetic programming. Crossover and mutation were constrained so that only legal programs were produced, and an explicit duplication operator was introduced to duplicate sub-branches, thus allow reuse of evolved substructures. They succeeded in evolving a synchronous adder (Hemmi, Mizoguchi et al. 1996), and moved on to solve an autonomous agent problem called the John Muir trail (Jefferson et al., 1990). No examples of evolved circuits were given, but for the trail problem they reported that finite state machines using as many as eight states were discovered by evolution. An eight state finite state machine is large in comparison to other hardware that has been evolved using developmental encodings. However no comparison was made with a more traditional encoding, so it is not clear just how hard the problem is using the primitive components they selected, and what benefit development had brought. They also noted that their production rule system was context-free. It was discussed above that this means the expressiveness of the developmental process may be limited. Hence this work may not be representative of developmental systems that model biology more closely. Additionally the level of abstraction at which evolution was applied was high. As was noted earlier, even though high levels of abstraction might allow the construction of large circuits, such abstractions waste opportunities for innovation. Additionally the components, designed for traditional methodologies, may not be particularly suitable for evolution, consequently limiting evolvability.

Haddow and Tufte have also used a production system approach for evolving hardware (Haddow et al., 2001), but at a lower level of abstraction. They used a variation of an L-System to evolve an array of five input LUTs. Their target was to maximise the number of LUTs that routed any one of four predetermined inputs directly to their output. Hence the contribution of each LUT to the phenotype's fitness is independent of the other LUTs in the array, and the maximum fitness for each LUT could be achieved by four of the 2^{32} possible LUT configurations. This problem is less complex than true circuit design problems. For evolution to proceed efficiently there must be high correlation between changes in the genotype and their effect on the fitness of the phenotype, so that small genetic changes can gently guide evolution to good solutions. If the correlation is too low, the search space becomes extremely rugged and evolution degrades to random search (Kauffman, 1993). Most circuit design problems use the circuit outputs in some way to determine fitness. Complex circuits are likely to have a nonlinear relationship between an alteration of the circuit structure and the behaviour of the circuit

outputs, hence fitness. This means that they are likely to have more rugged fitness landscapes than problems such as the pattern generation problem explored by Haddow et al, where there is high correlation between the size of changes in the circuit structure and fitness.

Even with this simplified problem, Haddow and Tufte's results were not promising. They evolved a set of L-System rules and a binary starting axiom, iteratively applying the rules to the axiom to generate a string of configuration bits for the LUT array. Thus the L-System could be envisaged as operating on an initial seed configuration of a single LUT that grew as the rules were applied. The preconditions of the rules used a ternary alphabet to represent true, false and don't care. At a given timestep the preconditions of all the rules were matched against the growing configuration bitstring and if a match was found the bits were replaced with the postcondition bits. Although Haddow and Tufte were inspired by context-free L-systems this model used a form of one-dimensional context as the rules contained more than one symbol in the preconditions. Pattern matching was not even limited to the bits of a single LUT. Instead a pattern could traverse the configuration bits for adjacent LUTs, thus the contents of one LUT could affect its neighbours. They did not introduce any two-dimensional context that might be useful when evolving a two-dimensional phenotype until later work.

It is somewhat surprising that they did not achieve good results from this problem, as it appears to be rather simple: a single rule with don't care terms as a precondition and a routing LUT configuration as a postcondition would solve the problem perfectly. One possible reason for the failure is that the rules were executed in order of specificity with the most specific first. Because of this rule ordering it would be unlikely that a general rule that solved the problem would be regularly executed, thus a more complex solution would be required. Another possible reason for the poor results is that fitness was measured so as to reward LUT configurations that used fewer inputs, with the intention that it would provide a bias towards one of the four the target configurations, as these use only a single input. However this bias actually generates a rather rugged landscape: if each of the target LUT configurations is altered by a single bit, it now requires all the LUT inputs to describe its full logical function. Because of the bias towards low number of inputs, these configurations have very low fitness. Additional single bit alterations can then reduce the number of inputs needed to describe the function, increasing fitness. Hence the local landscape for a single LUT is likely to be rugged and fairly deceptive.

Following this Tufte and Haddow's work moved away from a grammar-based model to a more biologically plausible model that was discussed in Section 2.2.4 (Tufte and Haddow, 2003b). Again this was specifically designed to evolve hardware. Each cell in the model was now restricted to one of a handful of logical states and the states of neighbouring cells was used as context. Again they only examined simple pattern-formation problems rather than circuit design

problems. They did not revisit the routing problem used earlier so it is difficult to make any comparisons between their two models, but they successfully evolved small arrays of predefined patterns.

In (Tufte and Haddow, 2003a) they presented a similar developmental model and recognised that the problems they explored were unlike traditional circuit design problems as they did not use circuit outputs to guide fitness. They suggested that the issue could be circumvented by treating the circuit as a non-uniform CA: by considering the combined states of the circuit's cells as the solution to a computational problem there would be no need for traditional inputs and outputs. This is an interesting idea worthy of further investigation. However to date they have not addressed the difficult question of how to represent a useful computational problem as a CA state, and how fitness could be measured during evolution. Most problems tackled by CAs to date, such as the density classification and synchronisation problems, (Mitchell, Crutchfield et al. 1996) are non-trivial computational tasks but they are still toy problems that have obvious CA representations. Many researchers have demonstrated that various two dimensional CAs that are capable of universal computation (von Neumann and Burks 1966; Codd 1968). Sipper has demonstrated this for non-uniform CAs (Sipper, 1997). However their solutions are rules that propagate patterns equivalent to logic gates, wires, clocks and memory. Specifying a problem as a combination of such patterns would require that the target circuit structure was known, thus there would be no design problem to be solved. It will be interesting to see how this representation problem is addressed in future work.

Miller and Thomson used a developmental model to evolve simple circuits (Miller and Thomson, 2003). The model was described in the section above, where it was pointed out that evolution struggles to deal with the complexity of the model as small genetic changes usually result in large changes in circuit structure. They compared the system's performance against a traditional encoding on a two bit adder problem. Although the developmental encoding found solutions to the problem it was outperformed by the traditional encoding. To explore scalability they also evolved even parity functions. The aim was to evolve a developmental program that solved successively larger parity problems (three, four and five parity) in three successive developmental timesteps. They succeeded in evolving many solutions to this problem. They also tested the circuits to see if they would solve larger parity problems, even though these problems had not been specified in the fitness function. One solved the six parity problem, and none solved the seven parity problem. This experiment provides a first insight into the scalability of developmental hardware evolution, and suggests that development might have the ability to be of use when solving large, regular problems by reusing developmental rules over successive timesteps. However by altering the fitness function to increase problem complexity at each timestep, they implicitly provided knowledge of how the problem should be partitioned in

exactly the same way as the work on incremental evolution discussed earlier. This leaves open the question of whether evolution is capable of making use of development to generate large modular circuits when presented with the entire problem at the outset, without a guide as to how the problem might be partitioned and modularised. Also although they compared the performance of a traditional encoding against a developmental encoding for the two bit adder problem they did not compare the *scalability* of the two encodings across a range of problem sizes, which is essential if developmental encodings are to be advocated to ease scalability problems. Such a comparison is provided here in Chapter 6 and forms the backbone of the evidence used to support the primary hypothesis of this thesis. Chapter 6 also compares an incremental fitness function similar to that used by Miller and Thompson with a non-incremental fitness function, revealing that manual decompositions can actually have a negative effect on evolutionary performance.

It was discussed earlier that Miller has since used a second model to generate a French flag (Miller, 2003; Miller, 2004), a problem introduced by Wolpert et al. (Wolpert et al., 2002). Recently this model has been modified by Liu et al. for implementation in an FPGA (Liu et al., 2004) (although evolution of the developmental activation function is still carried out offline using Cartesian Genetic Programming). The system has successfully evolved 6x6 patterns based on the French flag extrinsically. However fault tolerance, not scalability, appears to be the primary motivation for this work. One evolved solution was demonstrated to be capable of regenerating itself should the developed pattern be corrupted, simply by continuing the developmental program. This is an impressive result, and they have recently extended their work to the evolution of robust multipliers (Liu et al., 2005) but have not yet gone on to explore scalability. Van Remortel et al. have also explored the robustness of developed circuits (van Remortel, Lenaerts et al. 2002). They noted that biological development is not deterministic, and non-determinism might improve robustness in evolved circuits. They evolved an array of LUTs to maintain a fixed pattern of states, and injected noise into the phenotypes with different localities to model the effect of noise at different stages of non-deterministic development. The results were not promising as they found that circuits showed an exponential loss of robustness as noise increased. However they also found that the circuits were more robust to local noise (modelling non-deterministic events late in the development process) than distributed noise (modelling early non-deterministic events). This might suggest that non-deterministic events can have a useful role in the late stages of development. In later work van Remortel moved away from developmental circuit design, suggesting instead that simple models that do not involve any aspect of circuit design can be useful for exploring the interplay between development and evolution (van Remortel et al., 2003).

In (Koza, 1994) Koza stressed that scalability may be improved in genetic programming by reusing substructures that represent partial solutions to a large problem, and introduced the Automatically Defined Function (ADF) to the genetic programming paradigm to allow such reuse. He went on to use an explicit developmental system based on genetic programming to evolve a wide array of analogue circuits, including filters, amplifiers and computational circuits (Koza, Bennett et al. 1999), at the same time introducing the Automatically Defined Copy (ADC) to aid the iteration of substructures. In (Koza, Bennett et al. 1999) Koza et al. presented an evolved passive lowpass filter that reused a capacitative shunt six times by applying an ADC. This is an impressive level of reuse; just what will be required if evolutionary techniques are to scale to large circuit design problems. In (Koza, Keane et al. 2003) Koza et al. went on to present a Gedankenexperiment that demonstrated how the developmental system could outperform a traditional encoding on this problem. However they did not back this up with any empirical evidence. In the same article they also presented hierarchical reuse of ADFs in an evolved crossover filter. Hierarchical reuse is another feature that is vital if evolution is to incrementally build levels of abstraction in the way that has been described earlier, and so is of interest. Chapter 4 of this thesis demonstrates individual examples of such reuse in hardware evolved using a developmental system. However this is used only to support statistically valid empirical evidence that development can enhance scalability over a traditional encoding for circuit design problems, unlike Koza who has not to date provided such evidence for hardware evolution problems. One further point is that Koza's technique explicitly defines the developmental program, including these mechanisms of reuse, which for the reasons given in Section 2.2.4 might limit evolvability. Explicit techniques are likely to have considerably different evolutionary dynamics than implicit ones, so to generalise Koza's results to implicit systems might be misleading. The results presented in Chapters 4 and 6 are achieved using an implicit model, hence the results are of more direct relevance to those interested in such techniques. Lohn and Columbano have also used a mechanism similar to Koza's to evolve analogue circuits (Lohn and Colombano 1998).

Sekanina has also used an explicit model, to evolve sorting networks (Sekanina, 2004). The model used a linear genetic algorithm rather than a tree to encode instructions that described how a sorting network should be generated from a simple embryonic network. The results were impressive and suggested that the method was highly scalable: the developed networks were larger than those that could be evolved without development, and some evolved programs were general solutions for generating sorting networks. However the developmental model was explicit and highly specialised. It was constrained to evolve larger sorting networks incrementally from smaller networks using a handful of specialised operators. The incremental generation of sorting networks is a well-known technique (Knuth, 1998) hence it is unsurprising that Sekanina's model performed well and again leaves open the question of whether evolution

can use development to generate large circuits without a guide as to how the problem should be partitioned and modularised. Furthermore the model worked at a level of abstraction suitable for generating sorting networks rather than more general hardware designs.

Chapter 6 presents examples of large circuits (adders and parity generators) evolved using a developmental system. The largest of the evolved adders is a seven bit adder with carry. There is one other example in the literature of an evolved adder that is of comparable size (Shanthi et al., 2004). While this example was also evolved using a developmental system it relied heavily on the use of additional techniques, based on Torresen's increased complexity evolution that was discussed in Section 2.1.5 (Torresen, 2002a), to decompose the problem into several independently evolved sub-problems. As discussed earlier this might limit evolution's ability to innovate, which is one of the main motivations for using development over more traditional approaches. Furthermore unlike the circuits here connectivity was restricted to feedforward arrangements only, further constraining the problem space. The largest parity generator presented here is larger than any previously discovered using development and is of similar size to the larger evolved examples found in the literature (Koza, 1994; Rosca, 1995; Yu and Miller, 2002), which again were again restricted to feedforward connectivity unlike the circuit design space used here.

2.3 Summary

The work presented in this thesis lies at the intersection of two fields: hardware evolution and computational development. This chapter began by reviewing the field of hardware evolution. It surveyed the benefits brought about by hardware evolution and the applications to which it is particularly suited. Three ideas central to the discussion of current research trends were then introduced: the level of abstraction, which highlighted the wealth of circuit design space that remains unexplored to date, the hardware evaluation process, and the concept of inductive bias. State of the art research into three major issues was discussed in detail: innovation, generalisation and evolvability. This highlighted the benefits of allowing evolution to search for innovative designs and the need for the community to address the very real issues of scalability and generalisation if hardware evolution is to cross over to the real world. The discussion also revealed that to date the methods proposed to improve scalability (the focus of this thesis) other than development fail to provide a path to the evolution of large yet innovative circuit designs. It was suggested that development might have the potential to achieve this.

Following this the field of evolutionary development was reviewed. Applications to which developmental models have been applied were surveyed and the benefits that a developmental approach can lend to evolutionary design were discussed, focussing on design scalability. Models of development that have recently been used by researchers were classified as explicit

or implicit, and the more biologically plausible models were discussed in a greater level of detail, according to their regulatory and communication strategies. This highlighted the wide range of abstractions used by developmental models to date and revealed that there is a real need for comparative studies between models that implement different strategies. It also suggested some strategies that might be suited to hardware evolution using the research available to date. Finally work where development has been used in the context of hardware was reviewed, from the earliest accounts to the state of the art. First development-inspired architectures were discussed, followed by a review of the role of development in hardware evolution to date. The discussion noted that while many researchers believe that development can enhance the scalability of hardware evolution, its potential has not yet been demonstrated.

3 A Platform for Scalable, Innovative Hardware Evolution

This thesis explores ideas behind scalability and innovation in hardware evolution. This chapter presents a hardware evolution platform designed to allow the exploration of these ideas. In Chapter 4 it will be shown how the platform can be used in conjunction with a developmental genotype-phenotype mapping, but this chapter is concerned with more fundamental features of the platform. In Section 3.1 the criteria that were used to select elements of the platform are presented. In Section 3.2 the platform, designed to meet these criteria, is introduced. In Section 3.3 a series of experiments used to verify that the platform meets the criteria are presented. Section 3.4 summarises this chapter, including the contributions it makes to the field.

3.1 Design Criteria for the Hardware Evolution Platform

The four principle criteria that have been used as a basis of the hardware evolution platform on which the majority of the work in this thesis has been performed are now discussed. These are features required for innovation, scalability, evolvability and flexibility. Following this, criteria specific to the selection of an intrinsic hardware platform are also discussed.

3.1.1 Features for Innovation

In Section 2.1.3 it was noted that hardware evolution is likely to be of particular use when applied to complex design technologies where it has not been possible to develop formal methods to partition and decompose the design space, for instance for many analogue technologies (Flockton and Sheehan, 1999; Koza et al., 1999). It was also noted that evolution can discover innovative design features when working with common design spaces that are traditionally partitioned and decomposed from the top down, as evolution designs from the bottom up (Miller et al., 2000a). Additionally by relaxing the temporal and spatial constraints that allow for traditional top-down decomposition, evolution can explore a richer space beyond the scope of traditional designers (Thompson, 1996a), thereby offering further opportunity for innovation. It follows that the fewer constraints imposed on the evolutionary solution space, the more opportunity there may be to discover useful innovations.

Hence it seems probable that the best design spaces for innovation are those that impose as few design constraints, such as traditional digital design rules, as possible. Of these, complex analogue spaces such as those used by Koza and Flockton and Sheehan may provide the best

opportunity for innovation. However a space similar to that used by Thompson where components designed for digital circuits are used, but evolution is freed from as many of the spatial and temporal constraints associated with digital design as is practically possible, may still yield good opportunities for innovation.

3.1.2 Features for Evolvability

The issue of evolvability, introduced in Section 2.1.5, is highly interrelated with the issue of scalability. This thesis explores whether large hardware designs can be more easily evolved by allowing circuit modules to be evolved and then reused. If the underlying medium is incapable of directly supporting the evolution of small structures that can be used as modules in a larger circuit, then it will not be able to use development to combine modules into large circuits. Hence an evolutionary platform that is to be used to explore development should embody a basic level of evolvability so as this can be achieved.

For evolution to proceed efficiently there must be sufficient correlation between changes in the genotype and their consequent effect on phenotype fitness that small genetic changes can gently guide evolution to good solutions. If the correlation is too low, the search space becomes extremely rugged and evolution degrades to random search (Kauffman, 1993). Hence Section 2.1.5 suggested that coarse-grained representations may reduce the evolvability of a hardware design space, as the addition to or removal of a complex function from a circuit design is likely to have a more dramatic effect on the overall behaviour of the circuit than a simple function.

Allowing evolution fine-grained control over the evolutionary medium rather than imposing design abstractions derived from traditional top-down methodologies (as in function-level approaches) might provide better scope for bottom-up evolutionary design. Hence an ideal hardware evolution platform should allow evolution to make fine-grained changes to the phenotype.

In the same vein, components that radically change the behaviour of a circuit, be they fine-grained components or not, should also be avoided, as their use is likely to lead to similarly rugged landscapes. Examples of such components are logic gates, or similar high gain, threshold-like components. More suitable components might be ones that provide at least some level of graded response to an incoming signal. Even so the response of random networks of such devices may also be nonlinear, and although evolution may find them more amenable, they are not necessarily so.

3.1.3 Features for Scalability

In Section 2.1.5 it was noted that scalability, the ability of evolution to scale up to tackle large problems, is one barrier to the transfer of hardware evolution from the laboratory to the real world. It has also been stressed that scalability is the central issue of this thesis.

Hence a primary criterion of this evolutionary platform design is that it must be possible to instantiate large circuits and evaluate them efficiently. If hardware designs are evaluated intrinsically, the maximum circuit size is limited by the hardware resources provided by the reconfigurable device used. If designs are evaluated extrinsically, there is no hard limit to the size of the circuit that can be instantiated. Hence extrinsic evaluation using simulation may appear to be a sensible approach. However it is also necessary to evaluate the hardware designs efficiently. As discussed in the previous chapter, many simulations, for instance digital logic simulations, can be performed efficiently on modern microprocessors (Miller et al., 2000a) as they use an abstract logical model. If evolution was to be carried out at such levels of abstraction then an extrinsic evaluation might yet be a suitable choice.

However the discussion of innovation above explained that the best design spaces for innovation are likely to be those that relax these constraints. Any simulation where these constraints are relaxed would need to take into account the physical behaviour of the evolutionary medium in great detail, so would be computationally expensive, and probably prohibitively so. Hence if an evolutionary platform is to be used to explore scalability yet also provide good opportunities for innovation, an intrinsic platform is likely to be the best practical choice.

3.1.4 Features for Flexibility

It is a good idea to engineer as much flexibility as possible into a hardware evolution platform that is intended to be a research tool as it may need to meet unforeseen requirements as research unfolds. Because of this the ideal platform would be implemented entirely in software, as software development is almost always a more versatile process than hardware development. However Section 3.1.3 pointed out that in order to explore both scalability and innovation in any depth (without appropriating vast amounts of computing power) intrinsic evaluation is required. Although this will affect the flexibility of the platform in terms of what architecture must be explored, it could be possible to overlay virtual architectures on top of the native architecture if necessary, assuming the native device provides enough resources to do so.

Although evaluation should be carried out in hardware there is no requirement for the evolutionary algorithm that drives the system to be hardware-based. A hardware-only system,

especially a system based on a single die, might provide higher bandwidth and thereby faster evaluation times than a hybrid system. However using a software evolutionary algorithm allows different representations, different operators and different parameters to be easily explored. Such flexibility is a common requirement of research in Evolutionary Computation. Additionally the cost of synthesising an entire genetic algorithm in hardware is likely to be high, and so using a software evolutionary algorithm is likely to be a worthwhile tradeoff.

3.1.5 Device Class Selection

There are many classes of reconfigurable hardware devices, the most common of which are the programmable logic array (PLA), programmable array logic (PAL), the digital signal processor (DSP), the field programmable gate array (FPGA) and the field programmable analogue array (FPAA).

PLA and PAL devices only permit combinational configurations of logic elements, thus have limited opportunity for the realisation of highly innovative designs. DSPs have very limited freedom of topology: they are principally designed to apply a number of high-level mathematical functions to a signal in series. Thus again they are unlikely to offer good opportunities for innovation. Additionally the coarse-grained nature of their reconfigurability suggests that their evolvability might be low.

It has already been explained that unconstrained platforms are likely to have advantages of both evolvability and maximum opportunity for innovation. A device designed to enforce few if any temporal constraints such as an FPAA might therefore seem the ideal choice of evolutionary platform. Indeed devices such as the Motorola MPAA020 (Zebulum et al., 1998) and the Zetex TRAC (Flockton and Sheehan, 1999) have already been used for hardware evolution. However the difficulties in manufacturing analogue integrated circuits mean that devices available commercially provide coarse-grained reconfigurability and generally allow very limited reconfiguration of topology. Hence they actually present a fairly constrained design space that might not be particularly evolvable and have limited scope for innovation, impeding (but not preventing) their use for hardware evolution. Additionally the number of components that can be combined onto an integrated circuit is small. As this platform is to be used to explore scalability it is important to be able to instantiate relatively large circuits with many components under evolutionary control. Hence although FPAAs fit with the other criteria of this evolutionary platform, these issues rule out their use here.

FPGAs are unlikely to provide the level of evolvability and opportunity for innovation afforded by an ideal fine-grained, unconstrained topology FPAA. However many allow fine-grained

reconfiguration, and in general their interconnection topologies are extremely rich in comparison to other devices. If the traditional design rules that are usually imposed on such devices are relaxed, they are likely to provide a reasonably rich playground for evolution to innovate. Although the components they usually consist of are high-gain transistors, it has been shown that complex analogue behaviour can emerge from networks of these components, and that this behaviour can be controlled by evolution to generate innovative circuits (Thompson and Layzell, 1999). In addition FPGAs are large, potentially allowing search spaces of millions of components. This is far larger than state of the art evolutionary techniques can successfully manipulate. Hence although they do not fit the design criteria perfectly, FPGAs are expected to be the most suitable devices for exploring the scalability in innovative evolutionary design spaces.

3.1.6 Criteria for Intrinsic Hardware Evolution

This section considers criteria specific to the selection of a particular FPGA device. In (Thompson, 1996b) Thompson listed a number of criteria for intrinsic circuit evolution platforms. These were used to select a suitable device, and are discussed below.

Reconfigurable an unlimited number of times

Many reconfigurable devices are designed to be configured only once. Others are designed to be programmed a small number of times, but repeated configuration can eventually cause damage. Evolutionary experiments can require millions of evaluations, and so devices for intrinsic experiments should be able to be reconfigured indefinitely.

Fast and / or partial reconfiguration

As potentially millions of evaluations are needed, the evaluation process should be fast. Modern reconfigurable devices have millions of configurable transistors and consequently have large configuration bitstreams. Downloading these to a device can result in configuration being the bottleneck of the evolutionary process. The brute force solution to this problem is to use devices with high bandwidth configuration ports. Another solution is to evaluate a number of individuals at once on different areas of a single device, as proposed by de Garis and Higuchi et al. (de Garis, 1993) (Higuchi et al., 1994). This approach limits the population replacement strategy used by the evolutionary algorithm to strategies that replace individuals in multiples of the number of circuits evaluated at once. This means that a steady state genetic algorithm that replaces a single individual cannot be used. Additionally the minimum population size for generational algorithms is also limited. There is one more potential drawback. In Chapter 2 it was discussed that Thompson evolved circuits that relied on features unique to the particular transistors on which they were evolved: when the circuit was instantiated on another area of the same FPGA the circuit no longer functioned correctly. It may be useful to allow evolution to

explore design innovations that rely on such features. A useful innovation is likely to be evaluated many times during the course of evolution. If the innovation is evolved on different areas of the circuit, this potential source of innovation is lost. However it should be borne in mind that for practical circuits, this source of innovation is likely to be a hindrance, hence batch evaluation on a single device, or evaluation using different hardware may in many cases be a useful means of improving generalisation. Indeed this idea bears a close parallel with the concept of mixtrinsic evolution (Stoica et al., 2000) and has been proposed by Thompson himself as a means to improve the robustness of designs evolved at low design abstractions (Thompson, 1998)).

A more elegant solution to the problem of reconfiguration cost is to use partial reconfiguration. Here only the changes from the current configuration need to be uploaded to the device. This yields similar bandwidth to batch evaluation with no constraints on the learning algorithm or the opportunities for innovation.

Indestructibility or Validity Checking

In conventional CMOS technologies, a wire driven from two sources can cause a short circuit if one source drives the wire high and the other low. The high currents that result from such an event can damage the device. Hence it is sensible to prevent the possibility of contention by applying hard constraints rather than the softer biases that have been advocated so far. Some reconfigurable architectures such as the Xilinx 6200 are designed around an architecture where contention is impossible (Xilinx Inc., 1997). For those that are not, there are two main options: either constraints can be imposed directly through the representation (for instance by imposing an abstract architecture on top of the physical architecture) or circuits can be tested for contention before they are synthesized, then evaluated by an alternative means (for instance in simulation, or by allocating low fitness automatically) if such a condition is detected.

Fine Grain Reconfigurability (Fine Grain Architecture)

It has already been discussed how for evolution to have the most opportunity to innovate, it must be able to manipulate candidate circuits at a low level of abstraction. Hence a good platform needs fine-grain control over the evolving platform. Thompson also points out the distinction between fine grain architectures and fine grain reconfigurability, namely that although a device's architecture may be based on repeated large units, if these can be reconfigured at a finer level then this criterion will be met. However he did not consider the case of scalability through modularisation, as it is hypothesised is the case with a developmental approach. In this case, it would be useful to allow small (and large) modules to be created and reused across space. Hence a fine grained architecture is preferable to a coarse-grained one.

Flexible I/O

The method of supplying input and retrieving output from an evolved circuit can affect the feasibility of successful evolution, and so a platform that allows experimentation with this is useful.

Observability

In order to analyse how evolved circuits work, their internal signals need to be probed. When working with low design abstractions it may be impossible to remove the potential of signal probes to change the behaviour of the circuit and the probed signal, but architectures should be chosen with this as a consideration.

Low Cost (High Availability)

Cost is of particular importance when one of the ultimate motives behind using evolution is to lower costs through design automation. This thesis also recognises that low cost, and indeed wide availability are an important features for the selection of an FPGA used in evolutionary work, because the architecture applies many biases to the algorithm. In fact the discussion on partial reconfigurability mentioned a case where innovations discovered by evolution relied not only on a particular architecture, but on the particular transistors used in the experiment. To allow the best chance of repeatability, and to make any discoveries found of more general interest to the research community and beyond, a common, low-cost commercial architecture is a wise choice.

3.2 Platform Design

Section 3.1 has examined the criteria that the hardware evolution platform should meet. This section now presents the design of the platform, with reference to those criteria. First the features of the FPGA that was selected are explored, along with its drawbacks and how these can be circumvented. Following this is a description of the remainder of the platform.

3.2.1 The Virtex Architecture

Of current commercial FPGA architectures, the Xilinx Virtex architecture (Xilinx Inc., 2001) fits the criteria laid out above better than any other, hence it was selected as the basis of the evolutionary platform. It is an SRAM-based architecture, meaning that it can be reconfigured an indefinite number of times without failure, meeting one of the most vital criteria. Virtex operates in one of four configuration modes. The fastest is the SelectMAP mode, where data are transferred across an eight bit bus at a maximum frequency of 60 MHz, allowing burst transfers of up to 60 MBs^{-1}. In addition it allows partial configuration, hence both criteria of fast and partial configuration are met. It can operate across a wide range of I/O standards, and can

provide over three hundred user-definable pins, thereby meeting the criterion of flexible I/O. Additionally a wide range of fast, hierarchical internal routing allows good access to internal signals, therefore the criterion of observability is met. Finally it is widely available at reasonable cost and is currently used by many researchers in hardware evolution and related fields, thus meeting the criteria of low cost, widespread availability and relevance to the community. The Virtex architecture has been used for many more modern Xilinx products and is likely to be the basis of future Xilinx products, further ensuring the relevance of any architecture-specific findings for some years to come.

The two remaining criteria are not met so well. The most serious problem is that Virtex can be damaged by invalid configurations, as signal contention can arise. Care must be taken to prevent this from happening, and this issue is discussed in more detail below. The other criterion that is not perfectly met is the granularity of the architecture. Virtex, as with most modern FPGA architectures, is coarse-grained, and complete details of the reconfiguration bitstream, thus the finest details of the architecture, have not been disclosed by Xilinx. However Xilinx has provided a Java interface called JBits that allows a fine-grained level of reconfiguration (Guccione et al., 1999). The nature of the resources made available through the JBits application programming interface (API) means that evolution can work at a low level of abstraction where all traditional design abstractions have been relaxed, although not quite the finest level of reconfiguration possible on the device, the bitstream itself. This means that evolvability should not be unduly affected by the granularity issue. However, as was discussed in the section above, scalability may be. Even so, as so many of the other criteria are met well, Virtex remains the best choice of device for this work.

A simplified diagram of the Virtex architecture is shown in Figure 3.2.1. It is arranged as an array of configurable logic blocks (CLBs). There is a wide range of routing between these which can be grouped into three categories: single lines which connect CLBs to their neighbours, hex lines which connect a CLB to CLBs six blocks away, and long lines. The CLB array is surrounded by input / output blocks (IOBs) which contain logic and drivers for a range of I/O standards.

Each CLB consists of two slices, which each contain two SRAM-based four input function lookup tables (LUTs), some fast carry logic, and two flip-flops. There are thirteen inputs to each CLB slice: the inputs to both LUTs, the flip-flop clock, data, enable and set/reset lines. Outputs from each CLB can be drawn from a range of internal signals, including the lookup tables, the carry logic and the flip-flops. Configurable multiplexers select the connection between the routing and the CLB inputs and outputs. The JBits API allows independent reconfiguration of all of these elements. Each CLB is driven independently, so it is possible for contention to arise

between two drivers if two output multiplexers from two different CLBs drive the same line, as mentioned above.

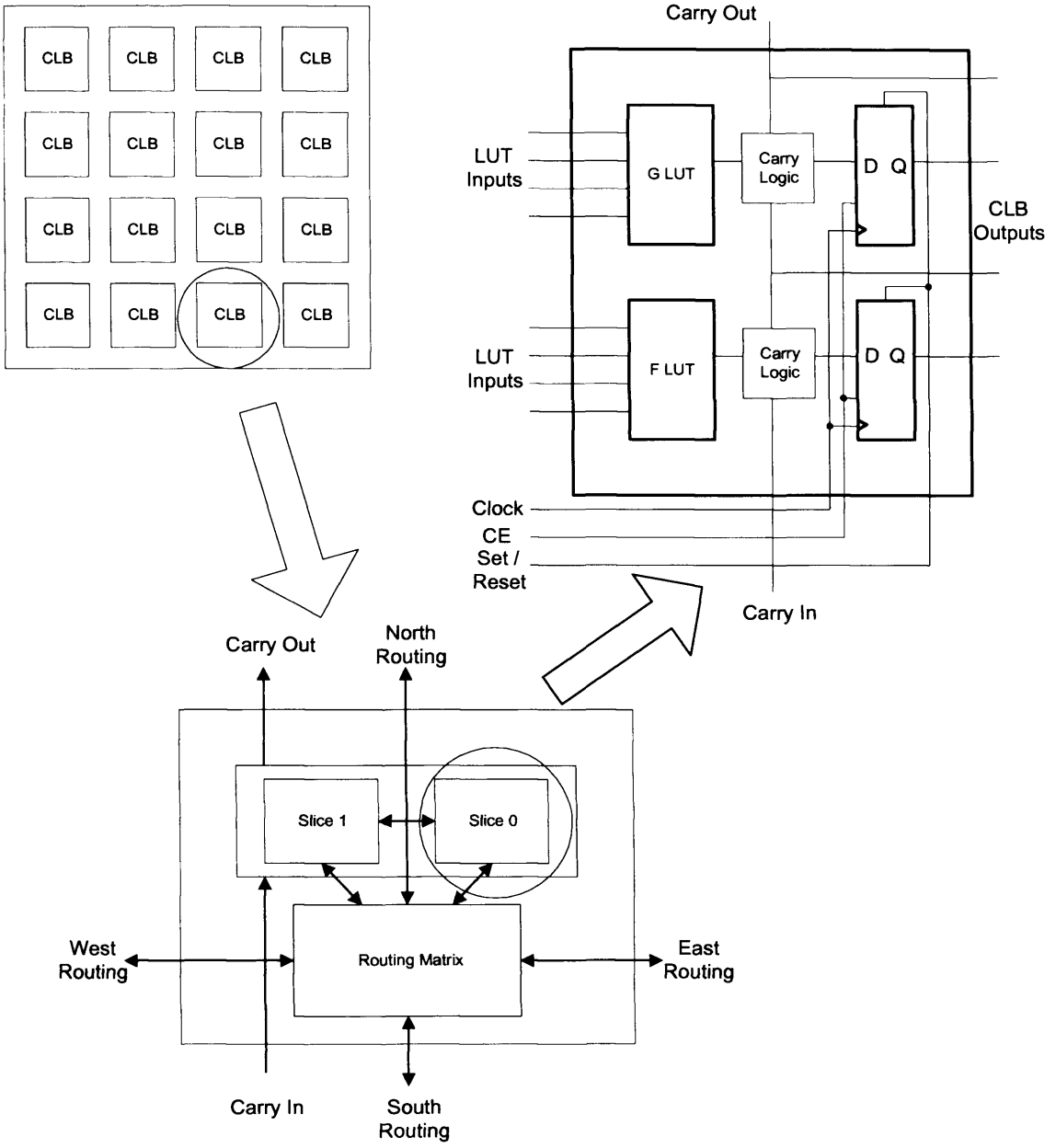


Figure 3.2.1: A simplified representation of the Virtex architecture.

3.2.2 Platform Description

In order to meet the criteria for flexibility it was decided that even though the hardware would be evaluated intrinsically, the evolutionary algorithm that would drive the system would be retained in software. Transfer of configuration data between the genetic algorithm and the Virtex FPGA was achieved using JBits API. JBits modifies a data file containing a pre-existing configuration bitstream, which is subsequently uploaded to the device. For the evolutionary platform this bitstream contained a template circuit generated using standard Xilinx design

tools. The bitstream provided a bus interface so that test data could be written to and read from the Virtex, and input and output buffers implemented in embedded RAM so that test and result data could be passed quickly to and from the predefined area of the device that was to be evolved.[5] When a circuit design was to be evaluated, JBits added the circuit to be evaluated to the template design, and the file was then written to the Virtex device using the Xilinx XHWIF interface that is provided with JBits. The platform design is shown in Figure 3.2.2.

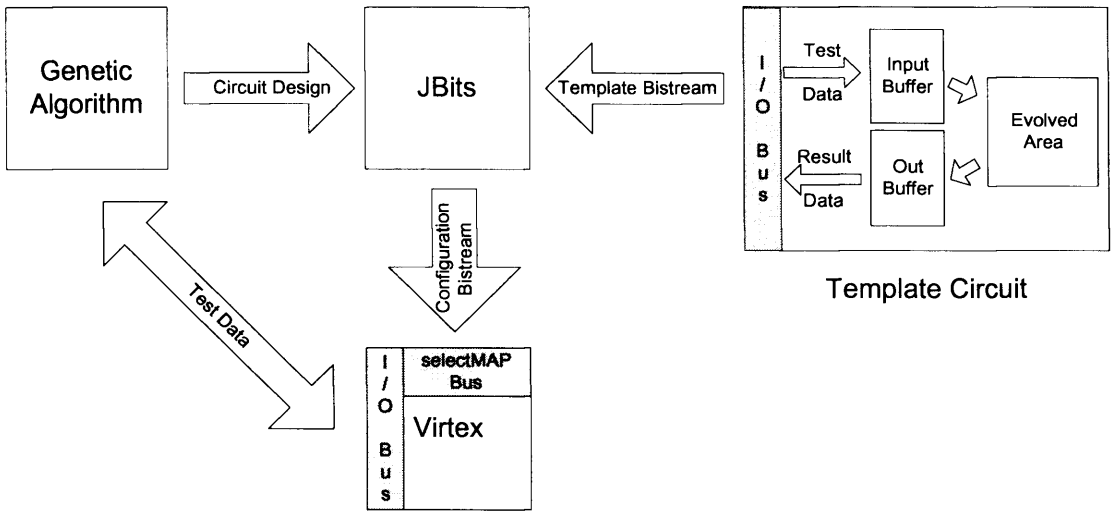


Figure 3.2.2: The hardware evolution platform.

An Alpha Data ADM-XRC development board (Alpha Data Parallel Systems Ltd., 1999) hosting a Xilinx XCV400 Virtex was used as the basis of the hardware platform. Communication between the host PC and the XRC board was achieved using a peripheral component interconnect (PCI) bus. The board includes a PCI-9080 PCI bridge manufactured by PLX Technology (PLX Technology Inc., 2000) that communicates with the FPGA across a 32 bit local bus that can operate at up to 40 MHz. Additional data lines are present to allow reconfiguration using the byte-wide SelectMAP interface. A limitation of the XRC is that the local configuration clock runs synchronously with the external PCI clock. Thus configuration rate is limited to 33MHz across the SelectMAP port, yielding a bandwidth of $33MBs^{-1}$. However the local bus clock operates independently so burst I/O transfers passing across the PCI bus operate at the full PCI33 bandwidth of $133MBs^{-1}$ and transfers across the local bus can burst at up to of $160MBs^{-1}$.

[5] The template circuit used for the experiments in this chapter did not include input and output buffers, although later work did. Instead data was routed directly to the evolved area using the PCI9080 direct slave transfer mode. Later incarnations introduced the buffers to improve transfer rates, with the transfers being carried out using Direct Memory Access. The template including buffers behaved identically to the original template on a number of test problems.

3.3 Platform Validation

A number of experiments were carried out to validate that the platform operated as expected. This began with validation of the genetic algorithm, followed by a validation of the entire platform. At the time the platform was designed there were no reports in the literature using Virtex for hardware evolution having relaxed the traditional digital design constraints, hence it was also important to validate that this was possible.

3.3.1 The Adder problem

The problem selected for the validation experiments in this chapter was the evolution of a two bit adder with carry. The evolution of adders has been well studied in the past, initially by Louis and Rawlins (Louis and Rawlins, 1991) and more recently by Miller et al. (Miller et al., 1997; Miller et al., 2000a; Miller and Thomson, 2003), Hollingworth et al. (Hollingworth et al., 2000a), Coello et al. (Coello et al., 2001), Kazadi et al. (Kazadi et al., 2001) and Shanthi et al. (Shanthi et al., 2004). The adder design space is very well understood, particularly in the digital domain. The aim is to evolve a circuit that transforms two n-bit binary numbers and one 1-bit binary carry signal to a n-bit binary sum and a 1-bit binary carry signal, for every possible combination of input signals. In all experiments conducted in this chapter fitness was measured simply by subjecting each candidate circuit to a test vector containing the complete two bit adder truthtable. One fitness point was awarded for each correct bit in the output sequence. In the case of a two bit adder, this yields a maximum fitness of 96. While the representation and the level of abstraction that was used to describe adders in each validation experiment were different, the required output behaviour, hence the fitness function, was identical in all cases.

Evolving binary adders might seem a strange choice of problem to demonstrate features of evolution beyond traditional digital abstractions. However it has two features that make it a good choice. First the problem has been well studied, both using traditional design and evolutionary design. Hence it is known that the solution space contains a rich array of potential solutions, and that addition can be achieved using a number of mechanisms. Second, the remaining chapters in this thesis are concerned with showing that evolution can use development to create and exploit modules that essentially embody useful design abstractions. The adder problem has been studied at various levels of abstraction, and a number of abstractions that are useful for traditional designers are well known and easy to implement in this architecture. For instance a digital abstraction can easily be enforced by using feedforward connections only. Another example of a useful traditional design abstraction is the encapsulation of full adders that can be chained together to form a larger adder. Although these abstractions may prove not to be of particular use for evolutionary circuit design, there are no abstractions that are known to be. Using the adder design space at least guarantees that potentially useful abstractions exist for development to explore, and exploit if possible. One

further benefit gained by exploring adders is that some work in this chapter concerns evolving adders on the Virtex architecture at a lower design abstraction than any previous work in the literature to date. By exploring the problem at a lower abstraction it will be demonstrated that the resources provided by the architecture can be exploited more efficiently by evolution than in previous work, lending further credence to Thompson's argument that there are benefits to low-abstraction evolution (Thompson, 1996b).

3.3.2 Validation of the Genetic Algorithm

Before conducting experiments at low design abstractions using intrinsic evaluation it was necessary to validate that the genetic algorithm performed as expected. Hence experiments originally performed by Miller et al. to investigate the extrinsic evolution of combinational adders (Miller et al., 1997) were repeated in simulation, using a custom logic simulator for evaluation. Mirroring the work of Miller et al., circuits were represented by a rectangular array of cells, indexed from the cell at the north-west corner, row first. Each cell performed one logic function on its two inputs, either a two input logic gate function or a multiplexer function. The sources for the cell inputs were either the outputs of other cells or a handful of global circuit inputs. These global inputs were the test input vector, the inverted test input vector, logic 0 or logic 1. To ensure that the evolved solutions were combinational circuits they were restricted to a feedforward topology: the source of each input was constrained to originate from a cell with a lower index than the cell itself. The circuit outputs were selected from the north and east sides of the cell array. This selection was under evolutionary control.

Problem Representation

The genetic algorithm used in this and other experiments was a standard population-based linear genetic algorithm with selection, crossover and mutation operators, after Goldberg (Goldberg, 1989), as discussed in Chapter 1. Later experiments used a binary representation, but in this case a circuit was represented as an integer string so that Miller et al's work was replicated closely. Each cell of the circuit was represented by a gene consisting of a triplet of integers. The first two integers of each gene represented the cell indexes of the two input sources, and the third integer represented the cell function. The locus of the gene represented the index of the current cell. The function of each cell was determined using a predefined lookup table that mapped functional alleles to logic functions. In the special cases when the function was a multiplexer, the allele was also used to determine the cell index of the multiplexer control signal source as shown in Table 3.3.1 where the entire lookup table of functions is presented. Following the genes that represented the cells, the chromosome contained three global cell outputs, each represented by an integer gene where the allele determined the output's cell index. In the following text these are labelled S0, S1, and COut, referring to the two sum bits and a carry out bit respectively.

The cardinality of each gene was set independently of the others in order to constrain the search space to feedforward circuits only, again mirroring the work of Miller et al. The mutation operator selected a random integer between zero and the cardinality, and was applied to all genes with the same probability. No bias was placed on selecting integers similar to the current value. (It was expected that this would not aid evolution as the ordering of the function lookup table was arbitrary.) Fitness was measured as discussed in Section 3.3.1.

Allele	Function	Allele	Function
0	A . B	7	!A
1	A . !B	8	!A +B
2	!A . B	9	!B
3	A ⊕ B	10	!A + B
4	A + B	11	!A + !B
5	!A . !B	12....	!C . A + C . B, where C = circuit input 0
6	!A ⊕ B	...(n)	!C . A + C . B, where C = input (n-12)

Table 3.3.1: The cell function lookup table.

The global inputs made available to the circuits were the four bits representing the two binary input values (labelled A0, A1, B0, and B1), the carry signal (labelled Cin), five signals representing the inverse of these signals (labelled !A0, !A1, !B0, !B1, !Cin), logic 0 and logic 1. Each input was given a cell index from 0 to 11, with the indexes of the true cells following on from this. Hence they were available to all cells in the circuit.

Genetic Parameters

The genetic parameters were mostly chosen by selecting those that Miller et al. had found useful in (Miller et al., 1997). Uniform crossover was used with breeding rate at 100%. The mutation rate was set to 5% of all genes in the population. The population size was set to 30. One difference from the experiments conducted by Miller et al. was the number of evaluations allowed before termination. Qualitative examination of early runs suggested that the 40,000 to 80,000 generations used by Miller were not necessary to achieve good results, with 20,000 generations sufficing. Hence a 20,000 generation termination limit was set for all further runs so that results could be gathered more quickly. Two member tournament selection was used. A tournament selection pressure, also used by Miller, was introduced and set to 0.8, again after Miller. This means that the winner of each tournament was selected only with 80% probability. The first generation of each run was randomly generated.

Results

Ten runs were each made for cell array sizes of 3 x 3, 3 wide by 4 high, and 4 x 4. The results of these experiments are shown in Table 3.3.2, and compare well to the results presented by Miller. Fitnesses have been normalised to a maximum of 100, and deviations normalised using the same ratio.

Array Size	Mean fitness of Best Solutions	Std. Dev. Of Best Solutions	% of Runs with Perfect Solutions
3x3	96.25	4.99	50%
4x3	96.88	3.71	50%
4x4	97.50	4.03	60%

Table 3.3.2: Results from ten runs of Miller's extrinsic adder problem across a range of array sizes

A series of ten random searches were also carried out on the 3x3 array search space, using the same number of evaluations as the evolutionary runs. These are shown in Figure 3.3.1, along with the best fitnesses of each generation of the 3x3 array evolutionary runs.

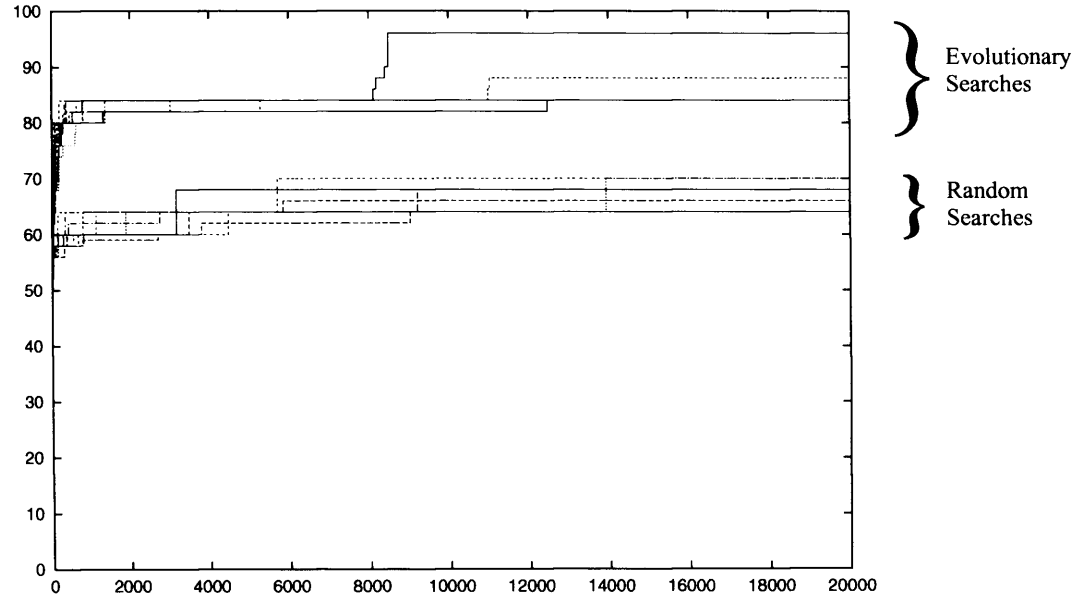


Figure 3.3.1: Best fitness against generations of an evolved two bit adder, on a 3x3 array along with ten random searches of the same length.

The experimental results broadly agree with Miller's results. For example, for a 3x3 array Miller reported a mean fitness of 96.14 with a standard deviation of 4.52, and 50% of cases perfect.

An example of a two bit adder evolved on the 3x3 array is shown in Figure 3.3.2. This circuit is a minor variation on a traditional two bit ripple-carry adder (Koren, 2001). Miller also reported the evolution of ripple-carry adders. The results display a trend towards higher mean fitnesses for larger arrays. This suggests that it is easier to find perfect solutions in larger arrays. However the solutions found within these arrays tended to involve more gates and connections than those discovered on smaller arrays, and so in this respect are less efficient. This is not surprising as no bias was used to search for parsimonious solutions. Miller et al. also noted this trend.

There were some minor differences between the algorithm and that reported by Miller. Miller used a "levels back" parameter to limit the length of the routing connections to a certain number of columns to the left of the current cell. As the arrays used in the experiments presented here were small with no more than four levels, the introduction of such a parameter was seen as

unnecessary, providing the results gathered were broadly similar to Miller's. This was the case. Miller also ran his algorithm for more generations, as discussed earlier.

Despite these two minor differences it can be concluded that the genetic algorithm used here performs similarly to that of Miller's as not only the statistical results gathered from both experiments are almost identical but the circuit designs discovered by evolution rely on similar innovations. This, combined with the fact that evolution clearly outperforms random search on this problem, leads to the conclusion that the genetic algorithm performs as expected.

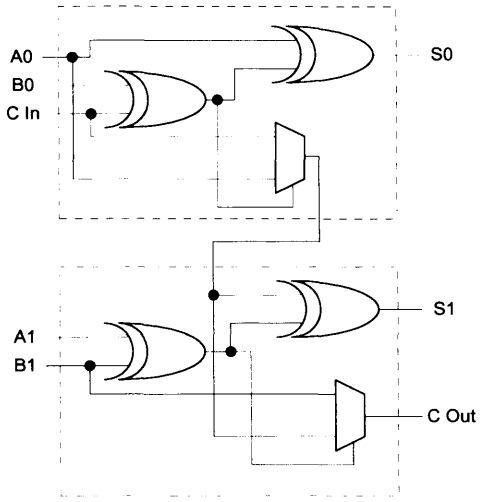


Figure 3.3.2: Example of an adder evolved on a 3x3 array.

3.3.3 A Demonstration of Low Abstraction Evolution using Virtex

Having validated the genetic algorithm by reproducing the work of Miller et al. using a simple logic simulator for evaluation, the intrinsic hardware platform presented in Section 3.2.2 was also validated with a series of experiments. First a hand designed optimal chromosome was passed to the evaluation function, and was confirmed to score maximum fitness. However it was also necessary to validate that circuits could be evolved using the platform, and that they could be evolved without the constraint of traditional digital design rules. Prior to this work the only work in the literature evolving circuits intrinsically on FPGAs with the digital design rules relaxed was Thompson's original demonstration using a frequency discriminator that was discussed in Chapter 2 (Thompson, 1996c), the evolution of oscillators by Huelsbergen et al. (Huelsbergen et al., 1999), and a single, unsuccessful experiment evolving patterns by Levi and Guccione (Levi and Guccione, 1999). Both Thompson and Huelsbergen et al. used Xilinx 6200 series FPGAs (Xilinx Inc., 1997), which are based around a quite different architecture to Virtex. The 6200 series uses a fine-grained sea-of-gates architecture where contention cannot arise. Hence both the components and the routing topology are quite different to that of Virtex. Levi's experiment used a Xilinx XC4000 series FPGA (Xilinx Inc., 1999). The XC4000 architecture is similar to that of Virtex: it is coarse-grained, with each CLB containing similar, but fewer resources. A major difference is that the local routing is structured so that contention

cannot occur. Levi made use of this in his experiments by constraining evolution only to explore local routing topologies. Unlike the XC4000 architecture, contention within the local routing of the Virtex architecture is possible, and additional constraints must be imposed on the architecture to avoid it. Additionally Levi did not succeed in evolving good solutions to his test problem. Hence whether circuits can be evolved at such low abstractions using Virtex, or even the similar XC4000 architecture, was still an open question.

The only reported evolution of circuits using Virtex prior to this work was that of Hollingworth et al (Hollingworth et al., 2000a), who evolved combinational adders. The representation was constrained by imposing a fixed feedforward routing topology so that only combinational circuits were explored, and only the lookup tables were evolved. Later work, carried out in parallel with the work presented in this chapter, imposed a sea-of-gates akin to the Xilinx XC6200 architecture on top of the native architecture, and an oscillator was evolved (Hollingworth et al., 2000b). This richer approach made better use of the resources of the FPGA, allowing feedback loops with no temporal constraints between function units. Hence circuits beyond the digital abstraction could be represented. However the virtual architecture placed considerable spatial constraints by restricting the routing made available to evolution. This meant that evolution was not able to explore the Virtex architecture free of as many design constraints as possible. In order to demonstrate that circuits can be evolved on the Virtex architecture free of almost all design constraints using the hardware evolution platform presented in this chapter, and to validate the entire evolutionary platform, the two bit adder problem was repeated using the evolutionary platform described above for intrinsic evaluation with as few hard constraints as possible.

Problem Representation

The approach used here attempted to allow evolution to manipulate as many of the components on the chip, including routing, as possible whilst avoiding the risk of contention. Hence almost all the resources available through JBits were placed under evolutionary control. An integer gene corresponding to each JBits resource was present in the genotype. The number of alleles for each of these genes matched the number of component states provided by JBits for that component in almost all cases. However it was so important to avoid damage of the device due to contention that the routing representation in this experiment (and all subsequent experiments) was partially restricted to avoid specific areas of space that might contain contentions, even though the restrictions also included a few circuits with no contention. This was achieved by three restrictions. First, only the single (nearest neighbour) wires were made available to evolution. Second the programmable interconnection points between routing wires were also not evolved. Finally, it was noted that although the Virtex architecture allows CLB input multiplexers to connect to a wide range of single lines, the connections between the output multiplexers and singles are sparse, and few can connect to any that their neighbours can. This

meant that in addition to the single wire and connection point restrictions, only eight of the possible forty-eight connections had to be prohibited to prevent any possible contention arising. The full chromosome structure for this experiment, and additional discussion of what resources were not evolved and why is presented in Appendix B. The resulting representation allows evolution to search the vast majority of the design space afforded by JBits, which is the lowest practical level of abstraction possible with Virtex.

The XCV400 Virtex used here contains a 60x40 array of CLBs surrounded by IOBs. Only a small area of the FPGA was evolved: in this experiment a 2x2 CLB array was used. As discussed in Section 3.2.2 this was achieved by manipulating a template bitstream with predefined I/O interfaces. For the two bit adder problem, the template contained five input registers (A0, B0, A1, B1 and Carry In) that were refreshed every clock cycle, and routed to the evolved area. The template also contained three output registers (Sum0, Sum1 and Carry Out) that were routed from the evolved area and refreshed after five clock cycles. This delay was introduced to ensure that any routing delay in prospective circuits was effectively ignored. The evaluation function for the problem queried these registers via the PCI bus.

One additional restriction was made to the cells on the edges of the evolved area. They were allowed to make routing connections to both the rest of the evolved area and the circuit input wires, but not to the other resources that lie on the edge of the evolved area. Inputs were routed directly into LUTs on the west edge. For instance all four LUTs on the south-west corner were restricted to using the A0 and B0 registers and the carry input register for inputs. The remaining input for each LUT was restricted to either an east or a north connection, and could not make a connection to the uncommitted logic to the west or south of the circuit. All non-LUT inputs to the CLB were also restricted to the east and north. The output of the evolved area was taken from hex lines on the east edge, which were not evolved. This led to a chromosome of 604 genes for a 2x2 evolved area. A diagram of the inputs to and outputs from the evolved area is shown in Figure 3.3.3.

Genetic Parameters

The genetic parameters for this experiment were unchanged from those used in the extrinsic experiment presented in Section 3.2.2, except the population size was increased from 30 to 50, and one-point crossover was used following work by Vassilev et al. (Vassilev et al., 1999) on the performance of crossover operators on three bit multiplier landscapes, an arithmetic problem highly related to the two bit adder problem. Also the number of generations before termination was reduced from 20,000 to 5000 owing to time constraints.

Unlike the extrinsic experiment described in section 3.2.2, the solutions were not constrained to combinational circuits, or indeed the digital abstraction as the representation allowed feedback

between units with no temporal constraints. When working at such low abstractions it is often necessary to specify generalisation that is normally hidden by higher abstractions, as discussed in Section 2.1.4. In order to ensure generalisation across any order of input sequences the order in which the test cases were presented to the evolved area was randomized for each evaluation. The genetic representation allows for circuits that may not generate the same fitness when evaluated twice as the outputs may exhibit dynamical variations unrelated to the input vector. Although such circuits are not useful final solutions they may contain valuable information about how to solve part of the problem, or how to traverse the fitness landscape to reach more fruitful areas of the search space. Because of this each individual was evaluated five times, and its worst fitness selected rather than discarding these solutions completely.

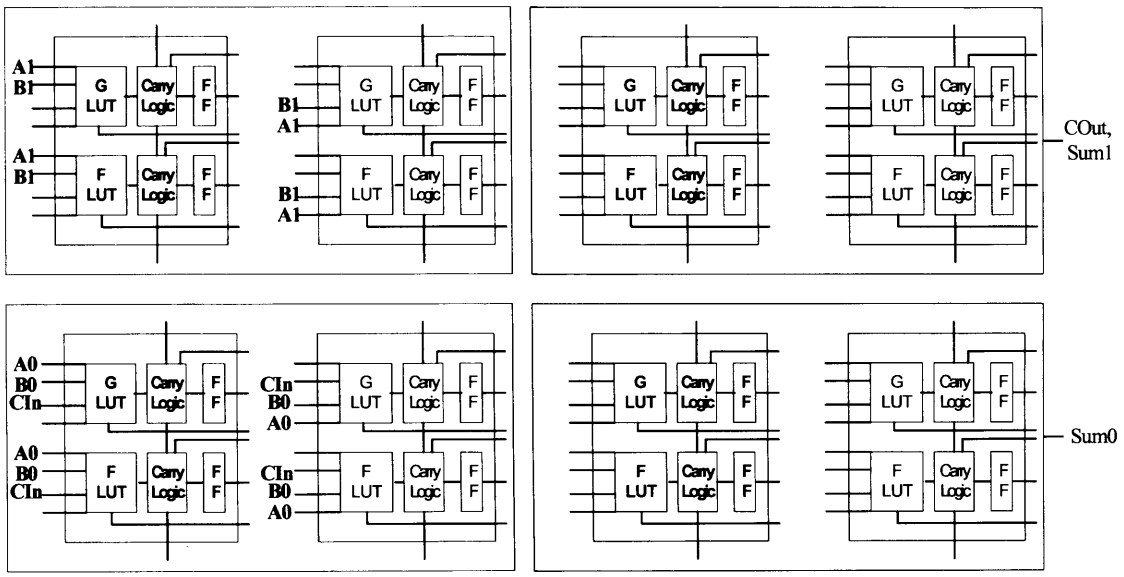


Figure 3.3.3: Inputs and outputs to the evolved area. Unlabelled LUT inputs were completely under evolutionary control.

Partial Reconfiguration

Initial experiments suffered from the speed it takes to reconfigure the FPGA. Configuration of each individual took around one second as a full configuration bitstream of over 300k was passed to the FPGA for each individual. Consequently few evaluations could be carried out in a reasonable timeframe. The extrinsic work presented earlier in this chapter suggested that large numbers of evaluations may be needed to evolve optimal solutions. This was confirmed by failure to evolve optimal solutions within 1000 generations of two test runs. As only a small area of the FPGA was used for evolution both options discussed in the section on reconfiguration criteria were available: either multiple solutions could be batched together in one configuration or partial reconfiguration could be used to reduce the bus traffic. As discussed earlier the most elegant solution to this problem is the use of partial reconfiguration. This was the approach taken, and an evaluation speedup of around an order of magnitude was achieved.

Results

Ten runs were carried out for a 2x2 cell. The results are shown in Table 3.3.3. Fitnesses and deviations have again been scaled to 100. The mean number of generations to find a perfect solution and the corresponding standard deviation ignore the single run that did not find a perfect solution. The average time to find a perfect solution was around 3 hours and 45 minutes using a 433MHz Celeron PC.

Mean Fitness of Best Solutions	Std. Dev. of Best Solutions	% Perfect Runs	Mean Generations to Find Perfect	Std. Dev. of Gens. to Find Perfect
99.58	1.26	90	2661	2169

Table 3.3.3: Results from ten runs of intrinsic two bit adder evolution

The results demonstrate that useful Virtex circuits can be found by evolution working at a very low design abstraction. It was expected that relaxing design constraints in this way would mean more work for the algorithm, as the search space increases so dramatically at lower abstractions. Comparison with Hollingworth et al.'s results from (Hollingworth et al., 2000a) supported this expectation, as they evolved functional adders with a 100% success rate, within a mean of 808 generations, with standard deviation of 259 generations. But surprisingly the huge increase in search space that resulted from allowing evolution to explore so much more of the Virtex architecture (around 2^{1000} as opposed to Hollingworth et al.'s space of 2^{256}) was not accompanied by a large decrease in the performance of the algorithm, rather only a small decrease. (The mean number of generations required to find a perfect solution increased from 808 generations to 2661 generations.) This suggests that not only was it possible to evolve useful circuits in this low-abstraction Virtex design space, but also for the adder problem at least (and it is tentatively suggested that they same may be true for similar arithmetic circuits) the low-level Virtex design space may be more evolvable than a more constrained space that limits all logic to LUTs and restricts routing. An explanation for this can be found in the features of the circuits evolved using the two different techniques. The Virtex CLB contains dedicated XOR gates that are useful for calculating arithmetic sums, linked together by logic that is designed to facilitate Manchester-like (Koren, 2001) carry chains. This allows a very efficient implementation of arithmetic circuits. Many of the adders evolved during the experiments presented here made use of this logic. The adders evolved by Hollingworth et al. (Hollingworth et al., 2000a) were evolved using a much more constrained search space that only explored feedforward networks of LUTs. This meant that their solutions could not make use of the implicit bias in the Virtex architecture towards the efficient design of arithmetic circuits. Relaxing the design constraints imposed by Hollingworth et al. and allowing evolution access to the additional logic within the Virtex CLB and routing between the CLBs meant that the evolved circuits not only used the Virtex architecture more efficiently, but also evolution could manipulate the resources under its control more easily to form adder circuits. This resulted in

only a slightly lower performance of the evolutionary algorithm over a much larger search space.

3.4 Summary

This chapter has presented the intrinsic hardware evolution platform that is used as the experimental platform throughout the rest of this thesis. The criteria used to design the platform have been introduced and the design choices made to ensure that platform meets these criteria have been discussed. Two sets of experiments have been presented. The first verified that the genetic algorithm used for the platform performs as expected. It also confirms the results obtained by Miller (Miller et al., 1997) using a similar system. The second set of experiments validated that the entire platform is capable of evolving hardware. It also demonstrated the evolution of hardware at a level of abstraction that imposes the fewest design constraints, hence the least designer-introduced bias, using the Virtex architecture that has been reported in the literature. The results of this set of experiments highlight that such an approach allows evolution to use the resources provided by the Virtex architecture more efficiently than when restricted to the search space used by Hollingworth et al. (Hollingworth et al., 2000a), and suggests that the search space provided by the lower abstraction approach might be more evolvable than a more constrained combinational space, in the case of the evolution of two bit adders.

4 Modularity and Abstraction in Developmental Hardware Evolution

The main aim of this thesis is to demonstrate that development can enhance the scalability of hardware evolution. Recent research applying developmental mappings to evolutionary algorithms was reviewed in Chapter 2, where it was established that development may allow evolution to solve a sub-problem and reuse the solution in other contexts. However the review also revealed that few engineering problems have been solved successfully using development.

Of those, most were problems that were solved with relatively simple ANNs. There are even fewer examples of development being used in the evolution of circuits. In fact prior to the research presented in this chapter, the only work in this area that succeeded in actually evolving circuits using an implicit developmental system was Hemmi et al's register transfer level evolution of grammar rules (Hemmi et al., 1994; Hemmi et al., 1995; Hemmi et al., 1996). (Tufte and Haddow also explored circuit development using a grammar-based system, but did not attempt to evolve useful circuits (Haddow et al., 2001; Tufte and Haddow, 2003b).) Hence prior to this work, hardware evolution using bio-inspired implicit development had not been demonstrated. This chapter presents the first attempt to do so. It also demonstrates the discovery by evolution of developmental modules that are potentially useful for scalability.

The rest of this chapter is structured as follows. Section 4.1 discusses the design criteria for a developmental hardware evolution system to enhance scalability. Section 4.2 introduces a developmental system designed according to these criteria and implemented using the evolutionary platform introduced in Chapter 3. Section 4.3 discusses the nature of modules that are useful for development. Section 4.4 presents experiments using the developmental system presented here to evolve circuits. This work predates all other attempts to evolve circuits using biologically plausible implicit development, and was first presented in (Gordon and Bentley, 2002). Section 4.5 presents analysis of the evolved circuits, showing a potentially useful design abstraction encoded in developmental modules, and explains why this is important for scalability.

4.1 Design Criteria for a Developmental Hardware Evolution System

The principle criteria used to design the developmental system presented in this chapter can be divided into those that benefit scalability in a general sense and those that are peculiar to

hardware evolution. The following section discusses general criteria for scalability and efficiency, and the subsequent section tackles the criteria specific to hardware evolution.

4.1.1 General criteria

The following features of development that were explained in detail in Chapter 2 are considered important for scalability and evolvability:

Generative

Developmental representations may enhance scalability by allowing the reuse of design elements. Rosenman and Gero dubbed representations with this feature as *generative* (Rosenman and Gero, 1999), and it is the most important criterion introduced here.

Implicit

It is likely that the implicit flow of control embodied by gene interaction facilitates incremental, bottom-up evolutionary design, unlike the explicit representations used by some researchers that have their origins in top-down programming design.

Context-Sensitive

Context sensitive developmental systems are more computationally powerful than context free systems, thus are able to realise a greater range of phenotypes. Additionally, through interaction with the local environment they can realise more complex and more robust phenotypes.

Flexible

As discussed in Chapter 3 in the context of the evolutionary platform, it is important that the developmental system be easy to modify so that it can meet new requirements as research proceeds. Thus the developmental system, like the evolutionary algorithm of the last chapter, should be implemented in software. Again this means that communication between the algorithm and the evolutionary platform incurs quite a large overhead. It is envisaged that an evolutionary system to be used in the real world could remove this communication overhead by implementing the entire system in hardware. It is beneficial to take this into account when designing such a developmental system by implementing developmental processes so that they are easily implemented in hardware when possible.

Evolvable

It has been stressed several times that efficient evolution requires good correlation between changes in the genotype and their effect on phenotype so that small genetic changes can gently guide evolution to good solutions. How this might be achieved using developmental systems was discussed in Chapter 2, where it was recognised that rule activation and response strategies, and communication strategies might play an important role in this regard. However activation

and response strategies geared towards good evolvability are generally more complex than their potentially less evolvable counterparts. This means that a tradeoff must be made between the evolvability of the developmental system and the cost that might be required to implement such a system in hardware.

4.1.2 Criteria for Hardware Evolution

Maps Gene Activity to Circuit Modification

A hardware evolution system must be able to manipulate circuit designs. To this end a developmental model for hardware evolution should model both regulatory gene products to allow implicit flow of control, and structural gene products that can modify a circuit in some way. In this case, a structural product should modify some feature of the evolutionary platform introduced in the previous chapter.

Explores Innovative Design Space

The map between structural gene products and circuit modifications should allow circuits beyond the space of traditional design to be explored. The Virtex FPGA at the heart of the evolutionary platform used here was designed for the digital design abstraction, so the system should be capable of representing circuits that do not adhere to digital design rules.

Incurs Low Computational Overhead

As potentially millions of evaluations are needed the entire circuit evaluation procedure, including development, must be fast. Therefore the mapping between development and configuration bitstream must be straightforward.

For the same reason, it is important to minimise the computational overhead of the developmental process itself. Chapter 2 noted that modelling both regional specification and differentiation is likely to be useful for hardware evolution, unlike morphogenesis, which is intimately tied to the biological medium. It was also noted that the same issue arises in the choice of the microscopic strategies used to implement regional specification and differentiation. As there is so little prior work on biologically plausible development for hardware evolution, the developmental strategies that are useful and transferable are not known, although the section above identified some features that might benefit evolvability at the expense of computational overhead. The aim of this thesis is to demonstrate scalability. In order to minimise computational overhead it was decided to only implement features necessary to demonstrate scalability successfully, by implementing a minimal set of features then expanding the model until the system was powerful enough to demonstrate scalability in hardware evolution.

4.2 An Exploratory Developmental System

This section presents a developmental system for hardware evolution that has been designed according to the criteria of Section 4.1. The general design features for a scalable developmental system will be presented first. These are the features related to gene regulation, context and communication. This is followed by a discussion of how structural gene products have been designed to map development to hardware.

4.2.1 Context, Regulation and Communication

The rule-based implicit systems surveyed in Chapter 2 satisfy the first two criteria of Section 4.1.1, namely that the process should be generative and implicit. It was decided that this system should also model genes as rules. In order to meet the third criterion of Section 4.1.1 the model had to include some form of context, and the form that this context should take had to be determined. This decision directly influenced the design of the gene regulation and intercellular communication strategies. The following three sections discuss the design decisions involved.

Context

Metazoan organisms are partitioned into cells, which can maintain different contexts. The idea of a cellular structure was used as the basis of the developmental abstraction used here. The architecture of the Virtex FPGA used in the evolutionary platform is also partitioned into regular units called Configurable Logic Blocks (CLBs). However whereas biological organisms are three dimensional, Virtex CLBs are arranged as a two dimensional array. In order to minimise the overhead of mapping development to a circuit it was decided that the developmental model should also be based on a two dimensional structure.

Again to minimise computation, development is carried out synchronously at discrete timesteps. To update the developing individual for a single timestep, the set of rules that make up the chromosome is tested against the environment in each of these cells, which consists of several virtual chemicals. If the environment differs between cells, it is possible for different rules to activate in each cell, which leads to their environments being altered in different ways. In this way, different chemical environments can be created and maintained in different cells. This leaves the question of what chemicals should be modelled to provide a cell's context.

Some models of development discussed in Chapter 2 used state variables derived from the phenotype as context. There it was argued that the bias imposed by such models might be complex and difficult to analyse, and as they are tied to the evaluation architecture might be difficult to generalise from and to modify. Chapter 2 also explained that process of DNA transcription is at the heart of biological development's generative ability. Transcription regulates the rate of gene expression through the presence of proteins called transcription

factors, which either increase (activators) or decrease (inhibitors) the transcription rate of a particular gene. All transcription factors are proteins that are generated by the expression of other genes. Thus a dynamic, autocatalytic network of gene products specifies which genes are expressed. By modelling a cell's chemical environment, or context, with only transcription factors, and ignoring all other chemistry present in biological cells, the model is kept as simple as possible to provide flexibility, yet encapsulates the key features that are needed to produce a generative, implicit, context-driven process. With this decision made, there remain only the questions of how to model and regulate these gene products, and how the interplay between these products should be exploited to produce a hardware design.

Gene Regulation Strategy

Chapter 2 categorised the gene regulation strategies previously used into three elements: regulatory elements, activation strategies and response strategies. It is worth stressing again that although the more complex models of gene regulation discussed in Chapter 2 may yield better evolvability than the simpler models, the philosophy used here is to initially use a simple model, and to introduce more complex features only if they are necessary to demonstrate scalability. This decision was taken partly to simplify analysis and minimise design time and computational overhead. However it was also taken because it might be useful to use the model designed here as a basis for a hardware implementation at a later date, and it is likely that fewer hardware resources would be required to implement a simpler model.

In keeping with this, the developmental system models the regulatory element (transcription factor proteins) as binary state variables, similar to the RBN models of Dellaert and Beer (Dellaert and Beer, 1996) and Hogeweg (Hogeweg, 2000). However the RBN model is not used for gene interaction here. Instead each gene is modelled as a rule, and gene regulation is modelled with simple Boolean activation and response strategies: the precondition of the rule specifies which proteins must be present (activators), and which must be absent (inhibitors) in order for that particular gene to activate. The postcondition of the rule defines the protein that is generated if the rule is activated. An example rule with five transcription factors is shown in Figure 4.2.1, and a lookup table of proteins and their postcondition codes is given in Appendix C.

For a gene like this to be activated, the proteins in the environment must match the pattern of proteins specified in the rule precondition. There are two bits in the rule precondition for each protein in the model (in this example proteins A-E) that specify whether it should be present, absent, or ignored for the gene to activate. This model allows evolution to generalise a rule more easily than the RBN model, as with a single mutation a protein previously required for a rule to activate can be set to a don't care state. With the RBN model this requires several mutations. The ability to generalise in this way may be useful for the evolution of developmental modules,

which may in turn aid evolvability. This will be discussed in more detail later in this chapter. A set of these rules make up the chromosome and defines how the proteins interact over time. At any given timestep in the developmental process, first the environment is inspected to determine which proteins are present, then each rule is inspected to determine whether the environment matches the rule, and if it does, the rule is activated. Once activated the postcondition is used as a key to a predefined lookup table of proteins. The protein matching the postcondition in this lookup table is then generated, and goes on to make up part of the protein environment of the following timestep. Any protein generated only exists for a single timestep, unless it is generated again through further rule activation. Protein generation is Boolean: if two or more rules generate a protein in a cell, the protein is generated just as if one rule had generated the protein. No details of concentration are modelled by rule response.

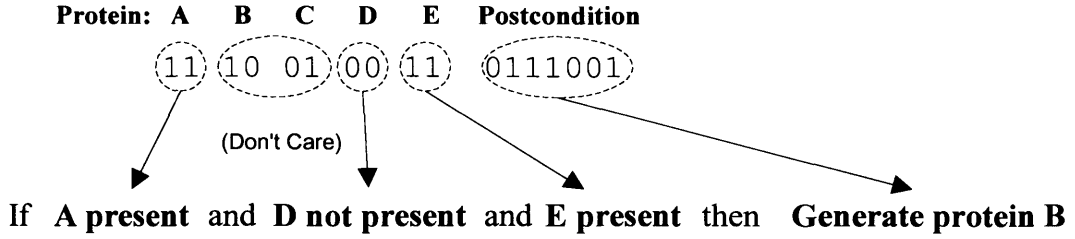


Figure 4.2.1: The Developmental Rule of the Exploratory System

Intercellular Communication Strategy

Context is a key feature of this model: cells must be able to affect their neighbour's environment. Biological development uses many forms of local intercellular communication to achieve this. As with regulatory models, Chapter 2 categorised the communication models used by other researchers by three features: communication element, transmission strategy and reception strategy. Chapter 2 also noted that the modes of communication in Biology vary widely in their physical mechanisms, suggesting that many strategies can be used successfully. As a simple model of development is required to meet the criterion of low computational overhead, it was decided to initially ignore all details of communication in biological development beyond the ideas that communication can be achieved through local cell-cell interactions and can involve the same gene products that are regulated by gene expression. Thus in the same way as Bongard (Bongard and Paul, 2000), Kitano (Kitano, 1995), Jakobi (Jakobi, 1995) and Miller (Miller, 2003), no distinction was made between regulatory and communication elements. A simple nearest-neighbour Boolean transmission strategy was used, similar to (Hogeweg, 2000) and Dellaert and Beer (Dellaert and Beer, 1996). A simple reception strategy was also used. It aimed to model protein concentration in the simplest manner possible: if half or more of the cell's neighbours generate a particular protein, then the cell considers the protein as present in its environment at the next timestep. Thus any incoming protein has the potential to interact directly with the cell's developmental rules much like the approach taken by

Bongard (Bongard, 2002), Jakobi (Jakobi, 1995), Miller (Miller, 2003) and Fleischer and Barr (Fleischer and Barr, 1993), but a potentially computationally expensive model of concentration that managed protein-rule interactions was not necessary.

To implement the communication process, each cell contains a protein detector and a protein generator, in order to record the proteins that are present and that are detected by each cell.

A developmental timestep for a cell proceeds thus:

(a) For each protein in the model the cell's protein detector sends a query to each of its Von Neumann neighbours (i.e. the four neighbours to the north, south, east and west on a two-dimensional grid) to determine if they are generating that protein. If half or more of the cell's neighbours are generating a protein then the protein is considered to have been detected and the cell's protein detector is updated to reflect this. (Step 1 in Figure 4.2.2)

(b) The rule set is tested against the pattern of proteins detected by the detector in step 1. (Step 2 in Figure 4.2.2) As each rule with a precondition matching the cell's current pattern of proteins is activated, the cell's protein generator is updated to represent the protein specified in the rule postcondition. (Step 3 in Figure 4.2.2)

This process is then repeated for a number of cycles, as shown in Figure 4.2.2, allowing the pattern of proteins formed across the global array of cells to change until a stable state or cycle of states is reached, or until development is halted after a predetermined number of timesteps.

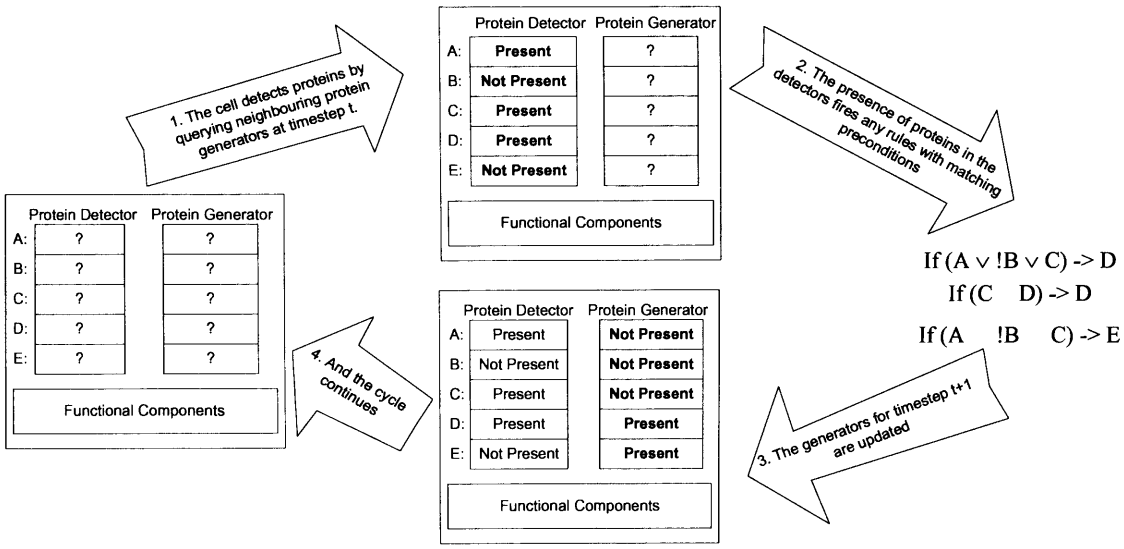


Figure 4.2.2: A developmental timestep highlighting the protein interaction model within a cell.

4.2.2 Mapping Development to Hardware

The developmental system described so far models the process of forming patterns of gene products. What remains to be introduced is a mechanism by which the patterns of gene products generate a circuit design.

First a method of mapping each cell in the model of development to components in the FPGA is required. It was noted earlier in this chapter that the developmental model is based on a two dimensional array of cells and the Virtex architecture of the evolutionary platform is essentially a two dimensional array of CLBs. One possibility that requires minimal computational effort would be to map each cell of the developmental system directly to a Virtex CLB. It was noted in Chapter 3 that to demonstrate scalability using development a fine-grained architecture is preferable, as it allows maximum opportunity for small modules to be discovered and reused by iteration across space. Unfortunately Virtex is a coarse-grained architecture. However mapping each cell directly to a CLB in this way is so straightforward that it was seen as an acceptable tradeoff, at least for this exploratory system.

This leaves the questions of what components in a CLB are controlled by development, and how. It has been stressed throughout this thesis that a key feature of hardware evolution is its ability to generate innovative designs, so it was important that the developmental system should allow this. Chapter 3 argued that the best opportunity for innovation using an architecture for digital design, such as Virtex, is likely to be when evolution is freed from as many of the spatial and temporal constraints associated with digital design as is practically possible (i.e. while still avoiding any possibility of contention), and evolution using such a system was demonstrated.

Although the same approach could have been taken here, the primary aim of the thesis is scalability, not innovation. Using a more constrained approach allows easier analysis of both development and the circuits generated by development. In addition as a more fine-grained architecture might benefit scalability by providing a stronger bias towards modular designs, it was decided to impose a virtual architecture with finer-grained CLBs on top of the Virtex architecture, similar to the approach taken by Hollingworth et al. (Hollingworth et al., 2000b) and Haddow and Tufte (Haddow and Tufte, 2001). However the virtual architecture was devised so that although restricted, it still allowed evolution to explore beyond the space of traditional digital design. Thus the system still has some potential to discover innovative designs and the results might still be of interest to researchers exploring the evolution of low-abstraction, non-digital design spaces. The virtual architecture is now presented.

The Virtual FPGA Architecture

The virtual architecture used for experiments in this chapter is shown in Figure 4.2.3. The section above explained that main aim of the virtual architecture is to provide a finer granularity than that of the Virtex architecture. An initial virtual architecture based on a single four input LUT was used in the earliest experiments, providing finer granularity than the Virtex CLB which incorporates four four input LUTs. However the results from these experiments were poor, and analysis of the virtual architecture revealed that it did not provide any means for two signal paths to cross (Gordon and Bentley, 2002). This was important for the test problem used (the same two bit adder with carry problem used in the previous chapter) as it is for most real-world circuit designs. To address this problem the virtual architecture was altered to incorporate two four input LUTs. These are labelled F and G in Figure 4.2.3.

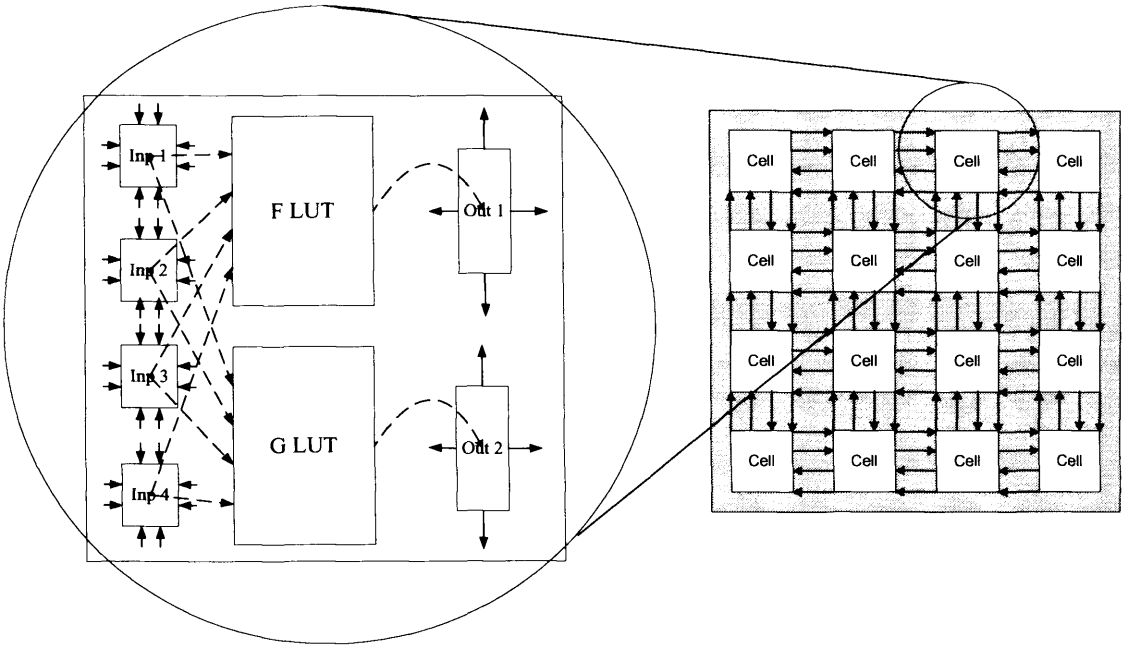


Figure 4.2.3: The virtual FPGA architecture. Each CLB contains 2 LUTs that share the 4 CLB inputs. A CLB contains 2 outputs that can drive the inputs of their Von Neumann neighbours. The solid lines, representing input and output connections, are configurable. The dashed lines are not.

Both LUTs in a virtual CLB share the same four inputs, and the CLB has only two outputs, the outputs of the two LUTs. The virtual architecture allows evolution control of three elements of the circuit:

(a) Logic: each minterm of the configuration truthtables for both 16 bit LUTs (labelled F and G in Figure 4.2.3) can be switched high or low independently.

(b) Outputs: Connections between the F and G-LUT outputs and wires leading to the north, south, east and west neighbours can be independently switched on or off via a series of virtual multiplexers as shown in Figure 4.2.4(a). (Off signals are pulled high.) Each LUT output has an independent route in each direction, so both outputs of a cell can be routed in any direction simultaneously.

(c) Inputs – For each of the four CLB inputs, evolution can select the input from the set of wires leading from the two LUT outputs of the north, south, east and west neighbouring CLBs to the current CLB, via a series of virtual multiplexers as shown in Figure 4.2.4(b). This allows evolution to select from any of the eight incoming routes, independently of whether these routes are actually being driven by a neighbouring LUT as described in (b) above.

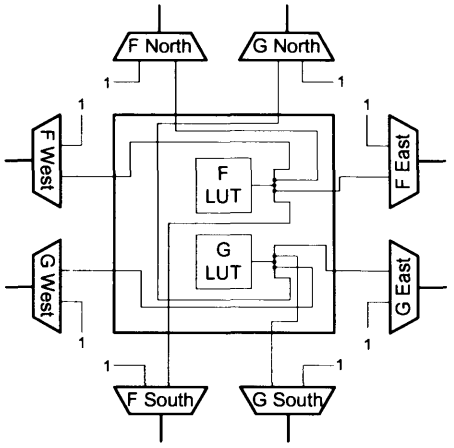


Figure 4.2.4(a): The outputs of the virtual CLB. Each LUT can drive a signal north, south east and west. When not driven, an output line is pulled high.

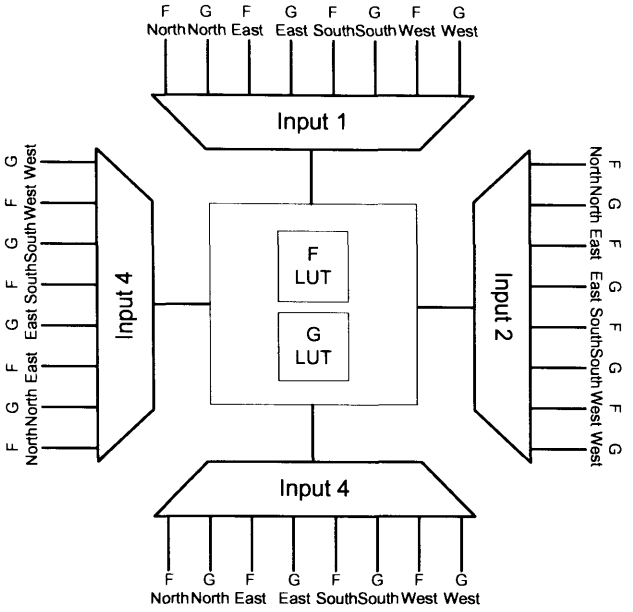


Figure 4.2.4(b): The inputs of the virtual CLB. Each input is selected from the virtual output multiplexers of its north, south east and west neighbours.

The virtual architecture described above applies spatial constraints to the Virtex architecture, in both the granularity and interconnection dimensions introduced in Section 2.1.2. Thus there are fewer opportunities for exploring innovative space than the almost completely unconstrained approach taken in the final experiment of Chapter 3. However the interconnection constraint used is at the nearest-neighbour level, thus if no additional temporal constraints are introduced, the solution space will still include circuits containing time-recurrent networks that lie beyond the scope of traditional digital design methodologies. Thus the solution space can still be considered to be operating at a low abstraction that provides much opportunity for the discovery of innovative circuits.

This architecture only makes use of a single slice of each Virtex CLB, but could be extended to include more elements and make use of the second slice. The input wires for each virtual CLB were mapped to one of eight hand-selected routes on the Virtex device: signals from the two LUTs of each CLB's four Von Neumann neighbours. Details of this mapping are presented in Appendix C.

A template circuit containing a bus interface, an input buffer, an output buffer and an array of CLBs to be evolved was synthesised using standard Xilinx design tools in the same manner as

the template circuit used in Chapter 3, but this time the CLBs were restricted so that they implemented the virtual architecture. Again at each evaluation the Xilinx JBits API was used to modify the template bitstream so that it encoded the circuit under evaluation, then the FPGA was configured with the bitstream.

Structural Proteins

In order to map the regulatory process to this architecture, the developmental cell model contains structural proteins that directly represent components of the virtual architecture. As there are a relatively large number of components in each cell, and the rule length is a fixed function of the number of proteins involved in regulation it was decided that structural proteins should not play a direct role in regulation. Thus they would not be included in the rule precondition, and the search space of possible rules would remain relatively small. Likewise structural proteins did not play a role in intercellular communication.

However it was necessary to have some means by which structural proteins could be generated, as they are responsible for modifying the hardware. Hence the rule postcondition was extended so that a unique code corresponding to each structural protein could be defined. If a rule with one of these codes as their postcondition was activated during any developmental timestep, the structural protein corresponding to the code was generated during that timestep. Over the course of development, the generation of each structural protein was recorded. Once development was complete the total amounts of each structural protein generated were used to determine what modifications should be made to the template circuit. The mechanisms by which circuit modifications are determined are detailed below.

Circuit modifications

Chapter 2 suggested that a graded regulation strategy might be beneficial for evolvability as it allows small genetic changes to be more easily correlated with small changes in phenotype. However in order to minimise computational complexity the activation and response strategies used in the model presented here are Boolean. In order to provide a mechanism by which evolution can alter the developmental process in a more graded manner a novel approach was taken to map structural proteins to phenotype. Development was allowed to proceed for a number of timesteps, allowing proteins to be produced and interact as described above. Other models of development usually map the proteins present at each developmental timestep to a phenotype. Thus the final phenotype is derived directly from the structural proteins present at the final timestep. In the model presented here, the total activity of the genes throughout development was used to derive the phenotype, by measuring the total amount of each structural protein that had accumulated over the course of development. It was envisaged that this might allow evolution more graded control over alterations to the phenotype, while still using Boolean activation and response strategies and regulatory units. The mechanism by which the phenotype

is derived from accumulated amounts of structural proteins is now described.

The components of the virtual architecture can be grouped into two different classes. The first class are those which can assume one of many states. The only components in the model of this class are the LUT inputs, which must be driven by a single wire. (Each input must be driven by only one of the eight surrounding outputs, the F and G outputs of its Von Neumann neighbours.) Hence each input must assume one of eight states. To control the selection process four groups of structural proteins were introduced, each group corresponding to one input. Within each group, eight unique structural proteins were defined, one to represent each possible connection between the input and the eight surrounding outputs. (From here on these proteins are referred to as input proteins.) At the end of development one of these eight connections had to be set for each input. To determine which one, the total amount of each input protein that was generated over the course of development was recorded, and the protein that was present in the greatest amount was considered to be the winner of a competition to drive the input. As this protein refers to a unique route between one input and one of the surrounding LUTs a connection between the input and that incoming route could be made within the FPGA. A lookup table was used to tie each input and each postcondition code to a specific structural protein. This table is reproduced in Appendix C.

The second class of components in the virtual architecture are those that assume only a Boolean state, rather than a range of possible states. An example of this is a minterm in the LUT truth table, which must be set either high or low. In case of the LUTs a unique structural protein was defined to represent each minterm. (From here on there will be referred to as LUT proteins. There are two four input LUTs, hence 32 minterms and 32 unique LUT proteins in total for each CLB.) The total amount of each LUT protein that was generated over the entire course of development was recorded, just as with the input proteins described earlier. If by the end of development the total amount of any of the LUT proteins had reached a predetermined threshold, then the corresponding minterm was set in the phenotype by setting the corresponding bit in the LUT truthtable of the FPGA. Likewise as outputs can only assume one of two states, on or off, structural proteins were used to represent each output, hence eight unique structural proteins were needed to define all possible outputs: the routes to the north, south, east and west for both the F and G-LUTs. If the total amount of any of the output proteins reached a predetermined threshold value by the end of development (described later), the route was set in the phenotype, meaning that the FPGA was set so that a route was present between the LUT to which the output refers and the surrounding CLBs, although whether this signal was actually used by any of the surrounding CLBs depended on the state of their inputs. Any floating inputs were pulled high, so that if a CLB selected an input that was not present because an output had not been set it would always receive a logic 1 signal. Appendix C shows lookup

tables that relate LUT minterms and LUT outputs to unique structural proteins and their postcondition codes.

The process described above explains how development can control the input wires, LUT logic and output wires that are present in a developed circuit. The entire mechanism by which evolution can manipulate the generation of circuits can now be summarised. Evolution is used to manipulate all the bits in a set of developmental rules as described above. This allows evolution to control the generation of regulatory proteins at different positions within the cellular array, as development proceeds over the course of many timesteps. In turn as the regulatory proteins produced by the rules interact, structural proteins are generated. The amount of each structural protein produced is recorded. This information can be thought of as a measure of activity of each structural protein, and is used to determine the state of the various components of the FPGA circuit in one of two ways: where a component can only assume a binary state a threshold value is used to define the boundary between the states. Where a component can assume a range of states, several proteins, each corresponding to one of these states compete and the most active is selected and mapped to the circuit.

Experiments carried out using this system and their results are presented in Section 4.4, with analysis in Section 4.5. However before this it useful to consider in a little more detail how this system might be able to provide scalability.

4.3 Scalability and Modules

Chapter 1 defined scalability in general terms as *the ability to solve a series of problems of increasing size where the larger problems can be decomposed into modules of the smaller problems*. In both Chapters 1 and 2 it has been argued that development could achieve this by encoding reusable features in developmental modules. In order to design a system to achieve this goal it is necessary to consider what these features are, what is the nature of a module, how might a module encode these features so that they can be reused, and how might a module be devised so that it is tractable to evolution.

The term 'module' is widely used in many fields, and has many different connotations. Schlosser and Wagner noted that there are two very general notions of how a module is defined in evolutionary developmental biology (Schlosser and Wagner, 2004):

"...either as a component of a system that operates largely independently of other components or as a component of a system that is repeatedly used."

The idea that a module is defined purely as a *reusable component* at first sight fits the work in this thesis well. Such a module does not necessarily have to encapsulate a concrete structure within a circuit. Instead it may encode some developmental process that can be reused in different contexts to give rise to different, but related structures or even behaviours. However *evolvable* modules require more than this. Evolvability requires correlation between small changes in genotype and their effects on the phenotype. Wagner and Altenberg suggested that one way to achieve this was from "independent genetic representation of functionally distinct character complexes" (Wagner and Altenberg, 1996): any small genetic change affects only a single structure or behaviour within the phenotype, rather than the entire organism. This would allow structures and behaviours to evolve largely independently, resulting in a more evolvable organism. Hence in the context of the work in this thesis, a developmental module is considered to be:

both a component of a system that is repeatedly used *and* a component of a system that operates largely independently of other components.

Now that a module has been more clearly defined, there remains the question of whether such developmental modules that are *useful for the generation of circuits* exist. Because a module need not relate directly to a structure in the circuit but to a behaviour, one potentially useful kind of module is one that encodes a design abstraction that biases evolutionary search towards useful areas of solution space. The genes involved in such a module would be reused several times to maintain the abstraction. Also the genes might exhibit high internal interdependencies, but crucially should be relatively unaffected by changes in genes not contained within the module.

The following sections present experiments using the developmental system detailed earlier in this chapter and analysis of the evolved developmental processes and their resultant circuits. The results of these experiments demonstrate that this developmental system can efficiently encode a module such as those described above, that the abstraction it represents is potentially useful to evolution, and such modules can be discovered by evolution.

4.4 Abstraction in Action

The developmental system described earlier in the chapter was applied to the two bit adder with carry problem introduced in Chapter 3. However to do this, values for a number of parameters associated with the developmental process had to be determined. Details of how these were selected are now outlined.

4.4.1 Parameter Selection

To aid parameter selection, the mechanism by which development might generate a two bit adder with carry circuit was considered. First an adder circuit that occupied a 2x5 array of virtual CLBs was hand-designed, and states were assigned to each CLB, with cells containing identical logic being assigned identical states. It was envisaged that one way (albeit not the most efficient) that development could generate this circuit would be to replicate these states in a pattern of regulatory proteins; the presence of each protein could then be used to activate structural genes that produced the logic needed in each cell. Five regulatory proteins were required to do this for the hand-designed circuit. Thus five regulatory proteins were included in the initial developmental model. Sensible initial states of the cells, i.e. the regulatory proteins present or absent at the first developmental timestep, also had to be determined. A method to introduce heterogeneity into the cell states is also required: if all cells are subject to the same states and rules, they cannot differentiate to different final states. This requirement is known as *symmetry breaking*. This is discussed in greater detail in Chapter 6, but one way this is achieved in nature is by maternal transcription factors present at the onset of development. To model maternal factors the initial states of each cell were evolved along with the developmental rules. The number of bits in the postcondition was set as the number required to encode a unique postcondition key for each protein, regulatory and structural, in the model. The 72 structural proteins and five regulatory proteins in the model could be encoded in seven bits, and each was assigned a code in the protein lookup table. If a rule that represented an unassigned code was activated, that rule had no effect on the model. The total number of rules in the chromosome was set so that if a random set of rules were created the probability that a rule was activated was near 0.5. This required 80 rules. The number of development timesteps used was determined through informal experiments: the model was run a number of times using random rules. It was noted that after around 30 timesteps the regulation process tended to reach a point attractor or a simple two or three state limit cycle, and could be considered to have reached a differentiated state. Thus the number of timesteps used was set to 30. Similarly the threshold amount of protein required for LUT minterms and LUT outputs to be set in the phenotype was selected through informal experimentation at 7.

4.4.2 Evolution of Adders

A 2x5 array of cells, the array required to encode the hand-designed adder in the section above, was evolved using the developmental system. 80 rules of 17 bits were used, yielding 1360 bits for the rule section of the chromosome. The initial state of each cell was also evolved, as discussed above. This was encoded at the start of the chromosome as an additional 50 bits making the final chromosome length 1410 bits. To provide a more conventional evolutionary model against which development could be compared, a naïve 1:1 genotype/phenotype mapping system was tested on the same problem using the same circuit components. The naïve 52 bit

representation for one cell is shown in Table 4.4.1. (The concatenation of ten cells yields a naïve chromosome length of 520 bits.)

For both systems, in order to prevent contention between the evolved area and the surrounding support circuitry, inputs and outputs on the edges of the evolved area other than those specified by the problem were forced off. The five cells on the west edge were provided with five input signals of the two bit adder with carry problem, A0, A1, B0, B1 and CIn, as their eastbound signals. The task was to evolve a circuit that mapped inputs to the three outputs Sum0, Sum1 and COut in accordance with the two bit adder with carry truth table, as in Chapter 3. Fitness was measured as the sum of the number of correct output bits across all input combinations, giving a maximum fitness of 96.

Locus	Component	Bits (Representation)
0-2	Input 1	3 (GN,GS,GE,GW, FN,FS,FE,FW)
3-5	Input 2	3(GN,GS,GE,GW, FN,FS,FE,FW)
6-8	Input 3	3 (GN,GS,GE,GW, FN,FS,FE,FW)
9-11	Input 4	3 (GN,GS,GE,GW, FN,FS,FE,FW)
12-27	GLUT	16 (16 minterms)
28-43	FLUT	16 (16 minterms)
44	Output GN on	1
45	Output GS on	1
46	Output GE on	1
47	Output GW on	1
48	Output FN on	1
49	Output FS on	1
50	Output FE on	1
51	Output FW on	1

Table 4.4.1: Equivalent naïve representation for the adder problem.

The genetic parameters, held constant for both representations, are shown in Table 4.4.2, and are identical to those used in the final experiment of Chapter 3, with two exceptions. Preliminary informal experiments with the developmental system suggested that increases in fitness beyond 2500 generations were rare, hence all runs were terminated at this point. This allowed the population size to be increased from 50 to 100 while maintaining the same total number of evaluations as earlier experiments.

Operator	Type	Rate
Selection	2 Member tournament	80%
Crossover	1-point	100%
Mutation	Point	5 per chrom.
Other parameters: Generational GA with elitism, 2500 generations, population size = 100, random initialisation.		

Table 4.4.2: Parameters for the evolutionary experiments.

To ensure generalisation across any order of input sequences, the order that they were presented to the circuit under evaluation was randomised, and to reduce the effects of potentially noisy evaluations each individual was evaluated five times, and its worst fitness selected.

Results gathered from five runs using development and five runs using the naïve representation are shown in Table 4.4.3. Five runs of a random search for the same number of evaluations are also presented for both representations. (This work was previously published in (Gordon and Bentley, 2002).)

System	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. of Best Solutions (σ_{best})	Best Solution
Developmental Evolution	80.20	3.30	84
Naïve Evolution	91.60	4.10	96
Developmental Random Search	71.90	0.74	73
Naïve Random Search	63.00	3.92	72

Table 4.4.3: Five runs of evolution and five random searches, both with and without development.

4.5 Analysis

The results show that solutions evolved with the developmental system generally have lower fitness than those evolved using the naïve mapping system. Unlike the naïve system, the developmental system did not succeed in discovering a fully functional adder. The results also show a clear difference between the improvement in mean fitness of the best solutions ($\overline{f}_{best}$) when moving from random search to the evolutionary search. When using the naïve system this increase is large. When using the developmental system, this is much smaller, and random search using the developmental system outperforms naïve random search. This suggests that although the density of moderate solutions in the developmental search space is higher than that of the naïve space, evolution is not able to manipulate development to find high fitness mappings. Student's t-test (Sheskin, 2003) was carried out to compare evolution against random search across the developmental search space and determine whether evolution performed significantly better than random search. The probability that the $\overline{f}_{best}$ values for both systems were significantly different was calculated as than 0.98, hence it is likely that evolution does significantly outperform random search. However the statistical population is very small and so this figure should be treated with caution. The poor performance of the initial model was not entirely unexpected, and was probably caused by one or more of the following issues.

First it should be noted that development relies on the interaction of rules. Any such system is inherently *epistatic*: alleles for two or more genes will interact so that their effects on fitness are nonlinear. Although this effect can be reduced through the evolution of non-interacting modules as discussed in the section above, some interactions will still exist between modules, and strong interactions are likely to exist within them, thus some level of epistasis will always exist in such a system. Because of this development is most likely to show performance benefits over a well-designed naïve system for large problems where the naïve search space becomes too large to

search effectively, and a developmental approach can harness its ability to reuse partial solutions to the problem. The search space used for this two bit adder with carry problem is relatively small, hence a one-to-one mapping is effective. Furthermore the two bit adder with carry problem has only limited scope for reuse. Scaling to larger adder problems is likely to be harder for the naïve system than the developmental system.

This issue might be exacerbated by the relatively coarse granularity of the virtual architecture. Although the granularity is finer than that of the underlying Virtex architecture, it is still quite coarse. Because of this it is necessary to model a large number of gene products (128 in total), most of which are structural. Although evolution may be capable of discovering useful configurations of cell environments, it may struggle to manipulate the large number of structural proteins within each cell that is necessary for it to discover design features that can be reused.

Even if this issue is offset in larger problems, the developmental system is not of much practical use if it cannot discover optimal solutions. Thus it is important to discover how the model of development might be improved. In the current model gene regulation relies on the interaction of rules through their Boolean activation and response strategies. Consider two rules of this type that interact to generate a feature in the phenotype at a particular position. Mutations made to either rule's postcondition might either generalise the rule by removing the requirement for a protein to be present or increase its specificity by introducing another factor. Increasing the specificity of a rule through a random mutation is likely to result in the rule no longer matching the surrounding environment, hence it will not longer activate. If the second rule requires the protein produced by the first rule in order to activate, it will also no longer activate, and so the entire feature in the phenotype will no longer be generated. Likewise a mutation in the postcondition might destroy the relationship between the two rules, and the feature will not be produced in the phenotype. Thus small genetic changes are likely to result in large changes in the phenotype, making evolutionary search degrade to performance close to random search. The results presented above fit with this scenario. If the activation and / or response strategy were graded rather than Boolean, although a mutation might still be detrimental to the strength of the relationship between the rules, the relationship might still exist in a weakened form, thus be more robust to genetic change.

Another cause for the poor performance of the developmental system might result from its inability to explore a wide range of cell state configurations, due to the extremely restrictive communication strategy. Consider a single cell surrounded by its four Von Neumann neighbours, and consider the state of a single protein within these cells. The protein could be either present or absent in each of the four neighbouring cells, hence there are 16 ways in which this protein could be distributed in these neighbours, ranging from absent in all through to

present in all. In the model that has been used in this chapter, the central cell can only detect the protein if it is present in half or more of the surrounding cells, otherwise it is considered as absent. Hence the model maps each of the 16 possible configurations to one of the two states detected by the central cell: eleven of the configurations map to 0 (protein absent) and the remaining five map to 1 (protein present). Thus many of the intricate configurations that could be generated by development are considered equivalent and any information embodied in the differences between these equivalent configurations is discarded. By altering the communication strategy so that these differences can be detected by the cell, some of this discarded information could be transferred between neighbouring cells at each timestep, and the computational power of the developmental system would be increased, perhaps enough to allow more useful configurations of cell environments to arise.

4.5.1 Modules in Development

Although the developmental system was unable to produce perfect solutions to the adder problem, there are interesting features in the solutions that were found, which were first published in (Gordon, 2003). Figure 4.5.1 shows the best circuit evolved along with protein concentrations in the cells at the start and end of development. The circuit diagram shows the 2x5 array of CLBs, each containing two LUTs, labelled F and G. Input signals are shown on the left of the diagram, and the evolved routing wires are shown outputting the right hand side of each CLB, with the inputs appearing on the left-hand side of each CLB. All but one of the routing wires for the circuit shown were generated by the seven rules also shown in Figure 4.5.1. The wires shown as thick black lines are generated by the three rules in Group 1, and transmit signals north. The wires shown as dotted lines are generated by the two rules in Group 2, and transmit signals east. It can be seen that almost all of these rules have fairly general preconditions, hence they are relatively unaffected by any changes in the surrounding context. Thus they meet the first criterion for an evolvable module introduced earlier: they operate largely independently of other components within the system, thus allowing evolution to determine the fitness contribution of each module largely independently of any other features of the circuit. When the rules of Group 1 are activated in vertically adjacent cells they interact with each other to produce northbound connections. Because they are general rules, by the end of development they are activated repeatedly in all cells, thus the northbound connections are iterated across the circuit. Repeated reuse is the second criterion of a module. Likewise the rules of Group 2 interact with each other to produce eastbound connections that are iterated across the circuit. The first group of rules can be considered as a module that biases the phenotype towards making northbound connections and the second group of rules a module that biases the phenotype towards eastbound connections. Furthermore these two modules interact with each other to impose a bias towards feedforward circuits. A feedforward constraint is a spatial

interconnection constraint commonly used by traditional designers to realise the digital abstraction, as noted in Section 2.1.2. Although it cannot not be stated without doubt that this abstraction is useful to evolution, it is very useful in the traditional approach to adder design. As the adder problem has been so well studied by traditional design, and only a small area of logic was made available to evolution, it is unlikely that many new and innovative solutions lie in the evolutionary design space in this case. Thus at least in this case, it is quite likely that evolution will benefit from focussing its search on the digital design space, as it has done here. Finally it should be noted that both of the modules discussed here are largely independent of the other. Because of this it would have been easier for evolution to discover them and evaluate their value independently of each other in an incremental manner. However as they interact with each other to impose the digital design abstraction they can also be considered together as a larger module. Thus not only has evolution discovered potentially useful modules, it has also combined them to form hierarchy of modules. Chapter 2 argued that a long-term solution to the scalability problem requires a mechanism to incrementally build hierarchical layers abstraction. The formation of hierarchical modules demonstrated here is such a mechanism.

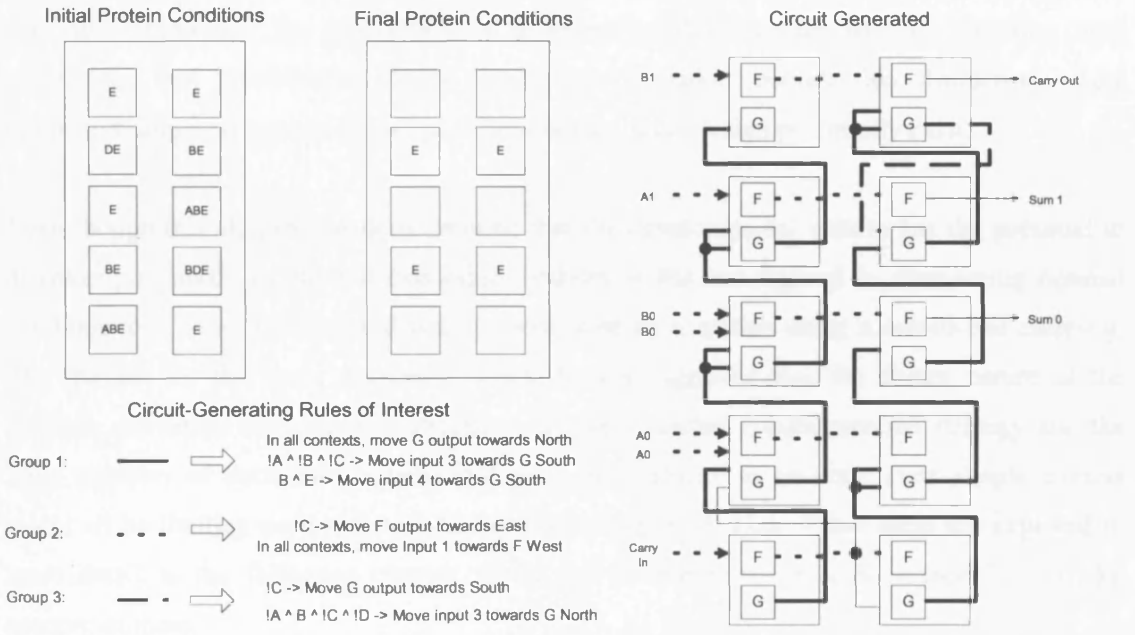


Figure 4.5.1: A solution to the two bit adder problem found by the developmental system.

There is another interesting feature that can be noted from the analysis of this circuit. One of only two routing wires not generated by these two groups of rules is the wire shown as a dashed line. This wire was generated by the two rules in group 3. Whilst the first of these rules is general, and is actually activated in all cells, the second is very specific, and is only generated in a single cell. Additionally it can be seen from the figure that this rule breaks the feedforward design abstraction, passing a signal southward, and in fact forms a time-recurrent loop that would not have been considered by traditional design techniques. This demonstrates that even though potentially useful design abstractions can be encapsulated succinctly by the

developmental rules, under specific conditions a general abstraction such as this can be overridden where evolution sees fit. This means that evolution is not prevented from finding a potentially useful innovation beyond the feedforward design abstraction that the first two groups of rules impose.

4.6 Summary

Chapter 2 argued that development can enhance the scalability of evolutionary search by discovering design abstractions that are potentially useful to evolution automatically and biasing the search towards areas of design space that lie within these abstractions. Chapter 2 also explained that evolution solves problems incrementally from the bottom up, thus it is likely to discover useful abstractions more easily if they can be imposed in a hierarchical manner. This chapter has presented an exploratory developmental hardware evolution system and used it to demonstrate the discovery of a potentially useful design abstraction by evolution. Furthermore it has demonstrated that this abstraction was encoded in modules that were general, thus evolvable, and that the modules formed a hierarchy that could be discovered incrementally. It was also noted that the abstraction is achieved by development without imposing hard constraints, thus providing evolution the ability to explore outside this abstraction where innovative solutions beyond the scope of traditional digital design potentially exist.

Even though this chapter has demonstrated that the developmental system has the potential to discover potentially useful and evolvable modules, it did not succeed in discovering optimal solutions to the test problem and was outperformed by a system using a one-to-one mapping. The reasons for this were discussed, where it was suggested that the abrupt nature of the Boolean activation and response strategies, the very limited communication strategy and the large numbers of structural proteins that must be produced to generate even simple circuits might all be limiting design factors that could be improved upon. These ideas are explored in more detail in the following chapter where the developmental system is modified to take account of them.

5 Development of Optimal Adder Designs

Chapter 4 presented an exploratory developmental system capable of learning and representing design abstractions that are potentially useful for hardware evolution. The primary measure of performance used in Chapter 4 and in the rest of this thesis is the likelihood that evolution will discover circuits that meet the functional criteria set out by the fitness function perfectly. This was chosen because in the real world it is necessary for circuits to perform perfectly and reliably, hence demonstrating that development outperforms a traditional 1:1 mapping using this measure is likely to be of more interest than another performance measure. Using this measure, the developmental mapping in Chapter 4 was outperformed by a 1:1 mapping on the two bit adder with carry problem. As noted in Chapter 4 this was not entirely unexpected, and it is likely that the development's performance is likely to compare more favourably to a 1:1 mapping as the problem size scales and evolution can take advantage of greater opportunities for reuse. Although it was anticipated that the *difference* in performance between the two kinds of system would drop and eventually reverse as problem size scaled, the absolute performance of both systems was expected to decrease when moving to larger problems. As the developmental system presented in Chapter 4 did not discover any perfect solutions for even the small two bit problem this performance measure would not be useful for demonstrating scalability with the system as it stood. Hence it was decided to improve the absolute performance of the developmental system.

Three factors that might hinder the performance of the system were identified in the analysis of experimental results in Chapter 4. These were the limited computational power of the communication strategy, the abrupt nature of the Boolean activation and response strategies and the large numbers of structural proteins necessary to generate useful phenotypes. This chapter identifies features that address these factors and introduces them to the developmental model used in the last chapter. It goes on to demonstrate that evolution can discover optimal solutions to the two bit adder with carry problem using the improved developmental model. The only other demonstration of hardware evolution using bio-inspired implicit development without resorting to the use of manual problem decomposition is the work of Miller and Thomson (Miller and Thomson 2003), who evolved a one bit adder with carry using a feedforward design abstraction that restricted the search to logical functions only, and gate-level granularity. In other words they tackled a potentially easier problem at a higher level of abstraction.

In this chapter the development process is decomposed so that the effect of the three design factors in question can be analysed more easily. The developmental system used in the previous chapter can be divided into two stages. The first stage is one of pattern generation, which is analogous to regional specification in biological development: the system must be capable of generating a number of different cellular contexts, where a context is defined by the regulatory proteins present in each cell. It must also be able to generate these contexts in a pattern that can be mapped to a functioning circuit using structural proteins. The second stage of development is that of mapping such a pattern of contexts to the circuit design using the structural proteins available. In nature this is achieved through differentiation and morphogenesis. (As morphogenesis is not modelled here, the process employed here is more akin to differentiation alone.) For a functioning circuit to be generated evolution must be able to manipulate *both* the pattern generation and mapping processes sufficiently well. By decomposing development into the two processes, it can be ascertained if either process is (or indeed both processes are) fundamentally limited in some way. Of the three potentially limiting design factors mentioned above, the communication strategy and the Boolean activation and response strategies directly affect the pattern generation process. Concerns about whether evolution can manipulate large numbers of structural proteins are related to the mapping stage.

Section 5.1 assesses the pattern generation process used in the previous chapter, and introduces a number of incremental improvements to the communication strategy, each followed by further assessment, until the process is powerful enough to generate patterns that are potentially useful for hardware evolution. Section 5.2 assesses changes made to the system designed to improve the evolvability of regulatory networks of genes based on Boolean activation and response strategies, again using the generation of useful patterns as a guide to performance. Section 5.3 investigates improvements that can be made to the mechanism by which development maps patterns of regulatory proteins to a circuit design, concluding by demonstrating that evolution can discover developmental rules that can generate fully functional designs for the two bit adder with carry problem, thus allowing the performance measure discussed above to be applied usefully. Furthermore it demonstrates that such circuits can be discovered by searching a space that lies beyond the scope of traditional design, which for many problems might provide greater opportunity for the discovery of innovative designs than searches of more conventional spaces. The model of development used in this successful demonstration is also used in Chapter 6 to demonstrate scalability and is reviewed in Section 5.4. This chapter is summarised in Section 5.5.

5.1 Pattern Formation

This section presents experiments that explore how the communication strategy used in the developmental system presented in Chapter 4 affects the system's ability to form useful patterns of proteins. It goes on to measure the affect on pattern formation of introducing improvements to the communication strategy.

5.1.1 Measuring Performance using Target Patterns

In order to assess the performance of pattern formation, it was necessary to consider what kind of patterns of proteins would generally be useful for hardware evolution. The no-free-lunch theorem suggests that it is not easy to answer such a question unequivocally for all problems (Wolpert and MacReady, 1997). However the adder design problem used in the previous two chapters is in many ways representative of the kinds of problem that it may be useful to apply development to: addition is central to most real-world computational circuits, and traditionally-designed adder circuits are usually modular and regular, hence potentially amenable to development. Consequently the patterns that are useful for the development of adders might serve as a guide to the kind of patterns that might be useful for hardware evolution. This leads to the question of what kinds of patterns might be useful for generating adders. To answer this a two bit ripple-carry adder was designed by hand within the 2x5 array of cells used in Chapter 4. The circuit was designed so that a number of the cells used identical logic. Each cell was then assigned a protein. Cells containing identical logic were assigned to the same protein. The circuit and the proteins assigned to each cell are shown in Figure 5.1.1. This procedure ensured that the map between the pattern of assigned proteins and the hand-designed circuit could be achieved using developmental rules of the type introduced in Chapter 4, using only the circuit-modifying structural proteins detailed in Appendix C. If evolution was capable of discovering rules that generated this pattern of proteins, it would be guaranteed that the developmental system was at least capable of encoding a solution to the adder problem.

To explore whether the pattern formation process was powerful enough for evolution to discover a rule set that encoded this pattern without involving the additional complexity of the mapping process a new fitness function was devised based on the Hamming distance of the candidate pattern of proteins from the target pattern of proteins. The longest distance possible was 50, which would result if every protein that was specified as on in the target pattern was off in the candidate solution, in every cell of the 2x5 array, and vice versa. Fitness was set as the measured distance between target and candidate subtracted from the longest distance of 50. This results in a maximum fitness of 50 and a minimum of 0.

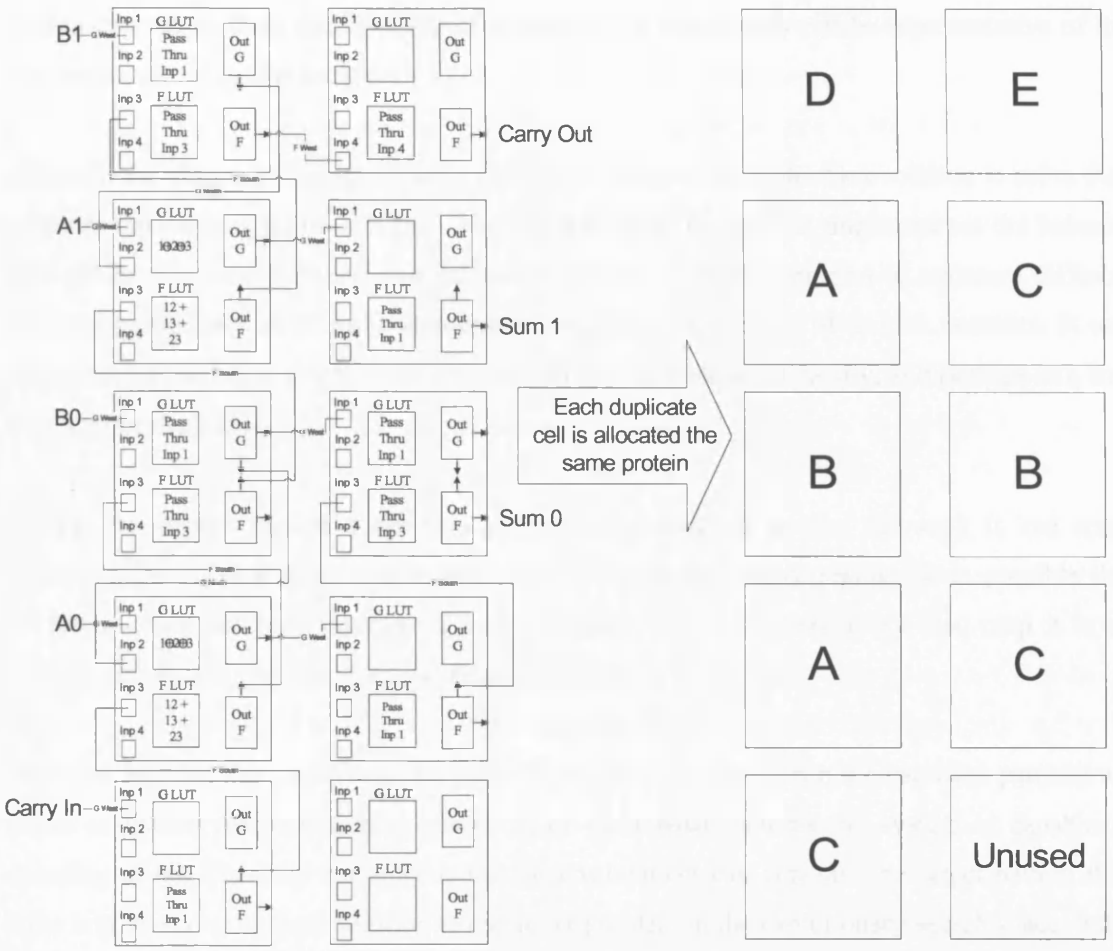


Figure 5.1.1: The hand-designed adder circuit and a pattern of proteins that distinguishes cells using different logic.

There are limitations to such an experiment. First the fitness function used in Chapter 4 is highly non-linear. A small change in a pattern brought about by mutation might lead to a large change in the circuit's behaviour. The fitness function based on Hamming distance is likely to have a more linear correlation between changes in the genotype and changes in fitness hence the problem is likely to be easier than the original circuit design problem. Even if this is not the case, a Hamming-based fitness function might bias evolution towards searching different areas of pattern space as the procedural biases of the algorithms differ. Hence the evolving patterns can only show that development is powerful enough to *encode* a potentially useful pattern.

Second the map between the pattern and the circuit was achieved by hand. The experiment does not reveal whether evolution could also discover rules to do this, but only that the developmental rules could *encode* an entire mapping between genome and circuit.

Finally, the pattern was arrived at by traditional top-down decomposition of the problem. There is no evidence to suggest that a solution partitioned in this way lies in an area of space that would be explored by the incremental, bottom-up evolutionary search, hence partitioning

pattern-generation from the mapping of a pattern to a circuit may not be representative of the way evolution solves the problem.

Although the chosen pattern generation problem is likely to be easier for evolution to solve than an entire circuit design problem there are two additional factors that might redress the balance. First the hand-designed developmental rule set does not use the proteins to represent different contexts particularly efficiently to represent the required number of unique contexts. It was expected that evolution might discover a way to use them more efficiently, and perhaps in a way that is more evolvable.

Second the hand-designed adder was purposefully designed so that although it had some regularity for development to take advance of, it was not highly regular. It is possible that evolution could not only discover a more regular pattern more easily but also map it to an optimal circuit using fewer structural rules.

However without first analysing the pattern formation process it is not clear what patterns are easier or harder for evolution to discover, or even what patterns the system is capable of encoding at all. The only certainty is that if development can generate the target pattern then there is at least one optimal solution to the adder problem in the evolutionary search space, but it may or may not lie in an area of space likely to be explored by evolution.

5.1.2 Assessing Pattern Formation

This section presents experiments to assess the ability of the developmental model to generate useful patterns. Most of this work was presented in (Gordon, 2003).

Experiment 1A: Evolving an Adder-Like Target Pattern

The first experiment conducted was as described above, to evolve the pattern of proteins deduced from the hand-designed adder. This pattern is shown in the 'Target Pattern' column of Table 5.1.1. The developmental system was altered so that the rule postconditions that had previously mapped to circuit alterations now had no effect. This allowed the use of exactly the same 17 bit rule structure as used in Chapter 4, with 80 rules used to make up the chromosome, again just as used in Chapter 4, but allowed the experiment to be carried out entirely in software, thus avoiding the costly circuit configuration and evaluation stages. One change was made to the genotype to simplify analysis. In the previous chapter the initial conditions (the regulatory proteins present or absent at the first developmental timestep) required to ensure that symmetry breaking was possible were encoded in an additional 50 bits within the chromosome and evolved. In the work presented in this chapter the initial conditions were instead pre-specified. Consequently the chromosome length was 1360 bits rather than 1410 bits. It was necessary to select initial conditions so that they still provided a means of symmetry breaking

without providing a trivial route to the formation of a target pattern. In (Wolpert et. at 2002) it was noted that pattern formation begins by defining the main axes of the embryo, and that pattern formation may then proceed by each cell using its current context to determine its position along each of these axes. The initial conditions used here were selected so that they differentiated the axes of the two dimensional cellular array: protein A was initially present in the cells along the base of the x-axis and protein B was present in the cells along the base of the Y axis. This pattern is shown in the 'Initial Conditions' column of Table 5.1.1 for Experiment 1A. The genetic and developmental parameters were identical to those used for the experiments in Chapter 4, except the number of generations was reduced from 2500 to 100 as the problem was expected to be less difficult. The experiment was repeated for 20 runs.

The results of the experiment are listed as 1A in Table 5.1.2. The target pattern consists of nine of the proteins set as present, as shown in the target protein pattern for Experiment 1A in Table 5.1.1, and all other proteins set as absent. In each of the 20 runs the fittest solution found actually generated no proteins at all. This meant that all nine proteins that were required to be present in the target pattern were not generated, giving a fitness of 41 out of a maximum of 50. It seemed that the system could not learn how to generate the pattern, or any part of the pattern that actually required some kind of computation.

Expt.	Initial Conditions	Target Pattern	Example Rule Set	Expt.	Initial Conditions	Target Pattern	Example Rule Set
1A	B B B B A,B A	D E A C B B A C C	N/A	1D	ABD ACD A D B C A D ABD ACD	ABD ACD A D B C A D ABD ACD	A!B!C->B !B!CD->C B->A C->D
1B	B B B B A,B A	D E A C B B A C C	N/A	1E	AB B BC A B C A B C A B C AB B BC	AB B BC A B C A B C A B C AB B BC	A!B->A B->B !BC->C
1C	D E A C B B A C C	D E A C B B A C C	N/A	1F	 A B C A B C	AB B BC A B C A B C A B C AB B BC	N/A

Table 5.1.1: Results of evolving target patterns of proteins A-E on 2x5 and 3x5 cell arrays with the exploratory system.

Experiment 1B: Addition of Current Cell State to Context

In Experiment 1A the context a cell used to determine its future state depended purely on the states of its neighbouring cells: if half or more of the cell's neighbours generated a particular protein, then the protein was considered present in the cell's environment. This means that there was no way in which the current state of a cell could directly affect the cell's next state. This does not model biology very closely, where any protein generated by the cell will immediately be present in its local environment. Furthermore from a computational standpoint, this creates

an extremely intimate relationship between the state of a cell and the states its neighbours at the previous timestep, constraining the patterns of proteins that can exist around a cell for it to assume a particular state at the following timestep. To determine if this affected the system's ability to generate the target pattern, the context of the model was altered to include the current state of each cell itself: if half or more of the cell's neighbours and the current cell generate a particular protein, it was considered present in the cell's environment at the next timestep. The experiment was then repeated, and is listed as Experiment 1B in Tables 5.1.1 and 5.1.2. The results were identical to Experiment 1A, suggesting that this change in context was not the only factor limiting evolution's performance. However as the change in context might be useful it was retained for later experiments.

Experiment 1C: Maintaining an Adder-Like Pattern

In addition to the need to generate patterns from simple initial conditions, once a useful pattern has been generated it might be useful if the system was able to maintain this pattern indefinitely. The target of the third experiment was purely to maintain the same pattern that experiments 1A and 1B attempted to generate, by setting the initial conditions to the same pattern as the target pattern, as shown in Table 5.1.1. The experimental results are shown in row 1C in Table 5.1.2. The solutions obtained were similar to those from Experiments 1A and 1B: 18 of the 20 runs generated protein B in the two cells of the central row cell and no other proteins in any other cell. This suggested that the system could only generate and maintain only the very simple patterns from the given initial conditions.

Expt.	Best / Max Fitness	% Perfect Runs	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. of Best Solutions (σ_{best})
1A	41/50	0	41	0.00
1B	41/50	0	41	0.00
1C	43/50	0	42.8	0.61
1D	50/50	10	45.1	2.43
1E	75/75	100	75.0	0.00
1F	61/75	0	61	0.00

Table 5.1.2: Results from the evolution of target patterns on 2x5 and 3x5 arrays with the initial model of development.

Experiment 1D: Maintaining a Simpler Pattern

A brief Gedankenexperiment revealed where some problems lay. An attempt was made to deduce a simple, yet non-trivial pattern that the system would be capable of maintaining for an arbitrary number of timesteps, if it were given as the initial condition on the 2x5 array. It was decided to design a pattern and corresponding set of rules that maintained the right and left hand columns as different environments. A requirement of this is that at least one protein appears in the left-hand column of the array that is not present in the right hand column, and vice versa. These proteins were dubbed the *side-determining proteins*. Manually designing a pattern to achieve this was not trivial, owing to the special conditions of intercellular communication

present in each corner. Each cell's protein environment consists of those proteins generated by half or more of the cells involved in its context, which was now the neighbouring cells and the cell itself. To minimise computational costs, this threshold had been calculated using an integer divide. Because the context of every corner only involves three cells, the threshold in these cases was set to one cell. Consequently any protein generated by a cell neighbouring a corner cell would always be present in the environment of the corner cell at the next timestep. The 2x5 array has adjacent corner cells, so it is impossible to create rules where these cells generate a protein that is not present in its neighbouring corner during development. As a result these cells cannot be the source of a side-determining protein. It was possible to design a rather complex pattern that avoided this problem by using non-corner cells to generate the side-determining proteins. This pattern is shown as Experiment 1D in Table 5.1.1 and can be maintained by the hand-designed rule set also shown in Table 5.1.1. To test whether evolution could discover a similar rule set the previous experiment was repeated using this pattern as both initial and target pattern, with results shown in Table 5.1.2. The experiment was then rerun allowing 1000 generations rather than 100, and the percentage of perfect runs rose to 80%. So it can be seen that in the case of the 2x5 array, selected patterns can be maintained, but the range of patterns that can be maintained is highly constrained, and finding such a pattern involves careful design. Furthermore a 2x5 array is a special case where corner cells are adjacent.

Experiment 1E: Maintaining a Pattern on a Larger Array

Experiment 1E in Table 5.1.2 shows the result of maintaining a simple pattern in a larger, 3x5 array that differentiates between the environments in all three columns, with all other parameters identical. (The pattern maintained is shown in Table 5.1.1) The results show an increase in performance over the 2x5 array case, with every run resulting in a perfect score. This improvement is easily explained: by moving to a 3x5 array, a buffer cell is introduced between the previously adjacent corner cells, and the problem of how to prevent excessive communication between these cells is removed. As arrays two cells wide are the only cases where this issue occurs, all further experiments were carried out using a 3x5 array, so more general conclusions could be made.

Experiment 1F: Generating a Simple Pattern

Having managed to show that a pattern that generates different environments in each column can be maintained by the system, an attempt was made to evolve a rule set that would not only maintain the pattern, but generate it from the simple initial conditions. The initial conditions chosen were that each column could initially be identified by a column-determining protein in the most southerly cell, as shown for Experiment 1F in Table 5.1.1. Results from the experiment are shown in Table 5.1.2. On examination of the best solutions found, every one generated protein B in every cell, protein A was only generated in the south-western corner cell and protein C was only generated in the south-eastern corner cell. From this it was inferred that the

system was able to make use of the initial conditions in the corner cells, and was able to generate general conditions across the entire array, much as seen in Chapter 4 where feedforward connections were generated across the entire array, with only a single corner cell breaking the abstraction. However evolution was not able to (easily) communicate information contained in the cells with special initial conditions (in this case the southerly row of cells) to the rest of the array, i.e. the cells could not communicate efficiently.

5.1.3 Improving Intercellular Communication

In order to improve the communication, a mechanism was devised by which information could be transmitted from a cell at the south of the array to a cell at the north. A simple scenario might be to generate a wavefront of information moving from the bottom to the top of the array, in effect 'growing' a pattern behind it. With the model of development as it was, this scenario would be very difficult to achieve. The major obstacle is that if development began with few proteins being generated in each cell, it was rare for a protein to actually be detected using a 3x5 array, as it was necessary for half or more of a cell's neighbours to be generating a protein for the cell to detect it. If the detection model was altered so that a cell detects every protein generated in its neighbourhood rather than only those generated by half or more of its neighbours, the task would become much simpler. A much richer set of states would be available in situations where protein generation is limited, as is likely to be the case when a small pattern is grown into a large one, as outlined above. An example of wavefront growth from simple initial conditions is shown in Table 5.1.3. The initial conditions in the first column of Table 5.1.3 show a new protein N being generated in the bottom row. Above this is a row of protein I acting as an interface, and above that, the protein to be replaced, protein O. The second column of Table 5.1.3 shows the proteins that would be detected by each cell using the old model and the third shows what would be detected with a model where every protein generated in a cell's Von Neumann neighbourhood was detected. The fourth column contains a rule set that if it were applied to a system using the new protein detection model, would generate a moving wavefront of I that travels up the array, replacing cells behind it with N. The final column describes the function that each rule in the set performs. This set of rules could be easily modified so that the pattern generated by the wavefront alternates at each timestep. However, there would still be difficulty in maintaining some patterns after the wavefront has moved on. This is the case if for instance the wavefront was required to leave behind horizontal stripes of cells that continue to generate one of two alternating proteins. Consider a pattern of protein generation with horizontal stripes of A and B. Should it be required to maintain this, the cells generating protein A must be able to experience a different environment than those generating protein B. But the symmetry of the pattern is such that each cell generating A has two neighbours generating B and vice versa. Were a cell able to detect every protein in its

neighbourhood, as suggested above, every cell would experience the same environment, containing both A and B.

One solution to this problem would be to introduce the concept of concentration to the detection model. For example consider a model where each cell generates each protein only at unit concentration but the concentration detected by a cell is the sum of the units generated by itself and its neighbours. Now consider a cell generating protein A, within an array that is generating a pattern of alternating horizontal stripes of proteins A and B. If the cell were to sum the concentrations generated by itself and its neighbours, it would experience A at a concentration of three (from the cell itself and its eastern and western neighbours). It would also experience B that is generated by its northern and southern neighbours, but only at a concentration of two. Hence if a rule was present to generate A that was activated when A and B were present in these concentrations, and another rule was present to generate B when A was at a concentration of two and B at three, the striped pattern could be maintained.

So by introducing a system where protein concentrations can be detected more accurately, it would be possible to open up a range of useful mechanisms for communicating information between cells. This approach is closely related to a two dimensional totalistic cellular automata, as described in (Wolfram, 2002).

States Generated:	States Detected (Old)	States Detected (New)	Rule Set	Rule Description
O O O O O O O O O I I I N N N	O O O O O O O O O I I I NI N NI	O O O O O O IO IO IO INO INO INO NI NI NI	N->N O->O O+I->I N+I+O->N N+I->N	Maintains cells already set to N Maintains cells still set to the O Creates a new row of I above the old row Grows a row of N, replacing the old I Maintains the row of N below I

Table 5.1.3: An example rule set that generates a travelling wavefront with the new model.

5.1.4 The Totalistic Developmental System

A new model of development that included these changes was implemented. To take advantage of these richer neighbourhood conditions it was necessary to alter the syntax of the rule precondition. In addition to the inequality operators used earlier, precedence operators were introduced. Each protein condition was now specified by five bits. The first two bits of the protein condition specify the operator, not equal to (00), less than or equal to (01) greater than or equal to (10) or equal to (11). Again the protein under test is determined by the locus of these bits. The final three bits define the protein concentration that the operator will act upon. This means that a rule can specify concentration values to range from zero to seven, even though the maximum concentration that could be detected is five, when a cell and all its neighbours are generating a protein. (This allows more scope for 'don't care' and 'ignore rule' terms to be formed using impossible and universal events.)

Because at this stage experiments were not concerned with the generation of a circuit but pattern generation, no postconditions that mapped to a circuit were included. This allowed the postcondition size to be reduced to two bits, reducing the search space considerably. Every postcondition in this new system mapped directly to one of four proteins, A-D. The number of proteins was reduced for two reasons. First, the target patterns specified for the experiments did not involve more than four proteins. Second this also reduced the size of the search space, as the precondition size was now 20 bits rather than 25 if five proteins were used. An example rule is shown in Figure 5.1.2.

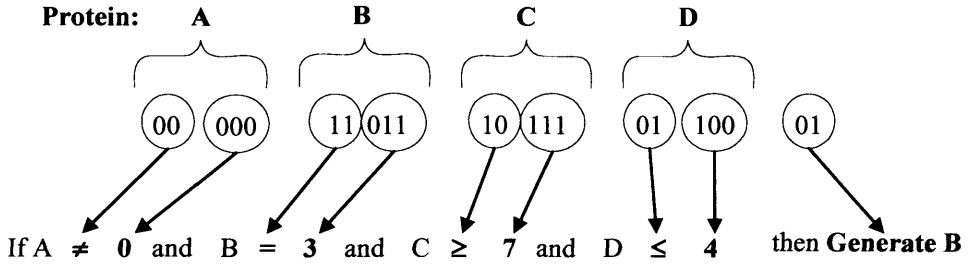


Figure 5.1.2: The totalistic developmental rule.

One other change was made to the system at this stage. By reducing the postcondition size to two bits, which always mapped to one of four proteins, the likelihood of two or more rules interacting with each other in an autocatalytic network was dramatically increased. Informal experiments suggested that because of this the number of rules in the chromosome could be reduced from 80 to 15 without adversely affecting the generation of such networks, but at the same time reducing the search space significantly. This resulted in a chromosome length of 330 bits.

Experiment 2A: Generating Columns of Proteins

Experiments were conducted with this system to verify that the patterns discussed above could be evolved from simple initial conditions. The first experiment was to evolve a target pattern where the concentration of protein A was high in the western column, but low in the others, B was high in the central column but low in the others, and C was high in the eastern column but low in the others. Initial conditions were set as if the southernmost three cells had generated A, B and C respectively at timestep *t-1*. This is analogous to Experiment 1F where the aim was to generate and maintain a different protein environment in each column of the array. The initial protein conditions, target protein pattern and the best pattern evolved are shown in Table 5.1.4 as Experiment 2A. For a given cell, the fitness contribution of each protein state was calculated as the difference between the final and target concentrations of that protein, subtracted from the maximum concentration that a rule could represent, which was currently seven. This gave a

maximum fitness of 420 and a minimum fitness of zero. Experimental results are shown in Table 5.1.5 as Experiment 2A. Initial experiments suggested that good results could be found within 1000 generations, so the experiment was run for ten runs of 1000 generations, with all other parameters as before. Of the ten runs, two evolved perfect solutions. The first perfect run used only three rules to generate the pattern:

(1) A!=0, B!=4, C==0, D<=1->C

(2) A<=2, B<=7, C==1, D<=3 ->B

(3) A!=1, B!=5, C!=0, D<=6 ->A

Expt.	Initial Conditions	Target Pattern	Best Evolved
2A	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00 00 10 11 11 01 00 10 10	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20
2B	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 40 04 00 00 00 40	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20	21 12 01 00 10 20 31 13 01 00 10 30 31 13 01 00 10 30 31 13 01 00 10 30 21 12 01 00 10 20
2C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00	23 23 23 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00 32 32 32 00 00 00 23 23 23 00 00 00	22 33 22 00 00 00 32 32 32 00 00 00 22 23 22 00 00 00 32 32 32 00 00 00 22 33 22 00 00 00
2D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 10 10 00 00 00	41 51 41 00 00 00 24 25 25 00 00 00 42 52 42 00 00 00 24 25 24 00 00 00 41 51 41 00 00 00	41 51 41 00 00 00 24 25 25 00 00 00 42 52 42 00 00 00 24 25 24 00 00 00 41 51 41 00 00 00
2E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 10 01 00 00 00 10	41 40 10 00 20 31 23 21 10 10 31 23 42 50 40 00 20 02 23 21 10 10 31 23 41 40 10 00 20 31	41 40 10 00 10 01 23 21 10 10 31 13 42 50 40 00 20 02 23 21 10 10 31 13 41 40 10 00 10 01
Where the concentration values shown are for the following proteins:	AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD AB CD		

Table 5.1.4: Initial protein concentrations, target protein patterns and best patterns evolved with the totalistic system. The lower right hand side of the table shows a key to protein concentrations.

Experiment	2A	2B	2C	2D	2E
Best Fitness	420	420	412	420	410
Max Fitness	420	420	420	420	420
% Perfect Runs	20	20	0	80	0
Mean (Best Fitnesses)	408.5	408.3	409.0	416.2	405
Std. Dev. (Best Fitnesses)	8.40	7.42	0.99	8.35	9.10

Table 5.1.5: Results from the evolution of patterns on a 3x5 cell array using the totalistic model.

These rules can be interpreted quite easily. Rule 1 and Rule 3 work in concert. In any given timestep, Rule 1 will generate C in all cells down one side of the array that currently contain A. Rule 3 will generate A in all cells of the opposite side of the array that contain C. So A and C generation swaps from side to side at each timestep. At the same time, the pattern grows one cell northwards at each timestep. This is because when a cell generates a protein, the cell above it will detect a low concentration of that protein, and so at the next timestep rules 1 and 3 will be activated in the cell lying to the north as well. Finally Rule 2 states that if A is present and C is present, then B should be generated. This only occurs where the output of the cells generating A and those generating C meet in the central column. So the generation of protein B, being dependent on A and C generation grows upward, following a cell behind the A and C generation front until the array is filled. At this point B generation is constant, and the only change between timesteps is that the presence of A and C alternates from side to side. The example has developed almost as envisaged in Section 5.1.3 with a wavefront travelling up the array until the array is filled. It is interesting to note that the second solution made use of eight rules, and no simple pattern of growth could be deduced in the complex interplay of rules. So not only can evolution make use of the growth mechanisms that the developmental system was designed to allow, it has found other mechanisms that are not easily interpretable.

Experiment 2B: More Realistic Initial Conditions

The experiment was repeated with different initial conditions. Previously the initial conditions had been set as the pattern that would occur if protein A had been generated in the western column at the last timestep, protein B in the central column and protein C in the eastern column. This has the effect of reducing the workload on the learning algorithm as all evolution had to do was to continue from the last step of protein generation. In the second experiment the initial conditions were simplified, putting more work on the algorithm. Each column was seeded with only the southernmost cell of each column set with the protein it should eventually fill with, at a concentration of four and are shown as Experiment 2B in Table 5.1.4. The experiment was then rerun ten runs of 1000 generations, and the results are shown in Table 5.1.5. The performance was almost identical to those of the first experiment. So at least in this case it was not necessary

to take great care over selecting initial conditions by emulating a hypothetical earlier generative step.

5.1.5 A Diffusion-like Model of Protein Concentrations

This section explores the generation of more complex patterns, and introduces changes to the developmental model to allow this.

Experiment 2C: Generating Stripes of Proteins

The next experiment set out to generate horizontal stripes of protein A alternating between low and high concentration from row to row, and horizontal stripes of protein B alternating between high and low concentrations, as shown in Table 5.1.4 as Experiment 2C. It was envisaged that this would be the result of horizontal stripes of A and B protein generation. The results for Experiment 2C are shown in Table 5.1.5. A perfect solution could not be found. However the vast majority of the ten runs achieved a high fitness of 410 or above, with the mean of the best fitnesses lying at 409.9. On closer examination of the patterns, it became clear that the system as it stood could not produce this pattern of protein concentrations from a striped pattern of protein A and B generation. Each cell on the east and west sides of the array has three neighbours. This means that each side-bound cell has an environment of two cells from a row generating one protein (itself and one cell horizontal to it) and two cells from rows generating the other protein (one above and one below). As both of these groups make an equal contribution to the cell's environment, the cell would have no means of determining which type of row it was located in. This results in a constant concentration of both proteins down either side, except in corner cell, which as was shown earlier behave differently again.

Experiment 2D: Generating Stripes with the Diffusion Model

It seemed that the level of communication between generating and non-generating cells again was not suitable for them to be able to organise themselves into the pattern required, because there is no mechanism for a cell to detect directly if it is generating a protein. Explicitly introducing such a case into the rules would increase the size of the search space considerably. An alternative, partial solution is to allow a cell that *generates* a protein to detect the protein at a greater concentration than a cell that only detects the protein generated by a neighbour. This can be considered as a highly abstracted model of the process of diffusion in the biological medium, where concentration reduces as molecules diffuse away from their source. Incorporating this change into the model would mean that if a cell detects a high concentration of a protein, the likelihood that the cell itself is generating the protein is higher than in the previous model, making the task of separating local signals from neighbouring signals easier, but not definitive. The model was changed so that when a protein was generated, the cell in which it was generated

would detect a contribution towards its final detected concentration of three for that protein, as opposed to one from each neighbour that generated the protein. The experiment was repeated having updated the initial conditions and target pattern to reflect these changes as shown in Table 5.1.4 as Experiment 2D, and the results of the experiment are shown in Table 5.1.5. The results show that eight of the ten runs succeeded in evolving the pattern perfectly.

Experiment 2E: Development of an Adder Pattern

Having introduced the totalistic model described in Section 5.1.4 and the diffusion model described in the paragraph above to improve the computational power of development, it was decided to attempt to evolve a pattern that evolution might be able to use to generate a two bit adder with carry.

To determine such a pattern, a two bit ripple-carry adder was again designed by hand, this time within the 3x5 array of cells used for the experiments detailed above. Then a pattern of proteins that could be mapped to the hand-designed circuit using the circuit-modifying structural proteins introduced in Chapter 4 was deduced, again by hand. Thus again it was ensured that if the pattern could be evolved there existed a fully functional adder within the search space of the evolutionary system. This pattern is shown as the target pattern for Experiment 2E in Table 5.1.4. Informal experiments revealed that evolution continued beyond 1000 generations, so the number of generations carried out by each run was increased to 5000, with all other genetic parameters unchanged from the previous experiments. The results for 20 runs of the experiment, shown in Table 5.1.5, showed no perfect solutions, the best solution found yielding a fitness of 410/420. This solution solved the problem correctly for three of the four proteins, but failed on perhaps the most complex protein pattern in the target pattern, protein C. As can be seen from Table 5.1.4, C was required to be generated in four cells. This solution succeeded in two cases in the central column (the cells where C is detected at a concentration of 3 in Table 5.1.4), but failed for the two cells at the corners of the eastern column.

In (Miller and Thomson, 2003) Miller and Thomson had noted that a hillclimbing algorithm had produced better results than a generational genetic algorithm when evolving circuits using a developmental mapping. To test whether such a change could improve the performance of the developmental model used here, the evolutionary algorithm was altered so that it behaved as a hillclimber, using only a point mutation operator set at five mutations per chromosome as had been used in the earlier evolutionary experiments. If a new prospective solution had equal fitness to the current solution, the new solution was selected with 50% probability. The hillclimber was run for 500,000 evaluations, in line with the number of evaluations carried out in the equivalent experiment using a generational genetic algorithm. Over the course of 20 runs, two perfect solutions were found, showing that the target pattern, hence the optimal hand-

designed adder, did indeed lie within the search space of the genetic algorithm, but the bias of the evolutionary algorithm prevented it from searching this space. This suggests that the developmental model is not very *evolvable* in its current representation using a traditional genetic algorithm. Although the move to a hillclimber had improved performance, the goal of this work was to demonstrate an enhancement in the scalability of hardware evolution by development, and to demonstrate this in a way that was of the widest relevance to the community. It is possible that the performance increase observed here was problem specific. Certainly the use of a hillclimber alters the inductive bias of the system in a subtle way that some researchers might consider a 'trick'. While this experiment suggested that it might be possible to evolve functional circuits and demonstrate scalability using the current model of development with a hillclimber, it was decided to explore how the model could be improved further so that a demonstration of scalability could be achieved using a fairly standard generational genetic algorithm. Hence the evolutionary algorithm used in the following experiments and throughout the rest of the thesis was identical to the one used in this chapter prior to the hillclimber experiment.

5.1.6 Outer Totalistic Development

In Section 5.1.5 it was noted some patterns cannot be generated without a mechanism for a cell to directly detect if it is generating a protein. In the previous section a partial solution was introduced where a protein generated within a cell was modelled with a higher concentration than the same protein generated by a neighbouring cell. This meant that if a cell detected a high concentration of a protein, there was a high likelihood that the cell itself was generating the protein, making the task of separating local signals from neighbouring signals easier, but not definitive as high concentrations could also be achieved if all neighbouring cells generated the protein. A definitive solution would be to alter the rule structure to include separate conditions for the state of a cell and the state of its neighbours but at the cost of increasing the search space. To test whether the ability to make this differentiation would outweigh the increase in search space, a new rule precondition was designed and experiments were conducted to compare the two approaches.

Experiment 3A: Outer Totalistic Development of an Adder Pattern

In the new model proteins are always generated at unit concentration, just as in all the models presented before the diffusion-like system was introduced in this chapter. Each protein condition in a rule is now specified by seven bits rather than five. The final five bits define the state required of the context supplied by the neighbouring cells, and is structured just as in the totalistic rule introduced in Section 5.1.4: the first two bits define either an equality, inequality or one of two precedence operators, and the final three bits define the protein concentration that the operator will act upon. Unlike the totalistic system used earlier this concentration is

measured by summing the amount of the protein generated in neighbouring cells but not the current cell. The rule can still specify concentration values to range from zero to seven, even though the maximum concentration that could be detected is four, when all neighbouring cells are generating a protein, thus continuing to allow 'don't care' and 'ignore rule' terms to be produced using universal and impossible events. As the protein can only be generated in unit concentration by the cell itself, only an additional two bit term is needed to specify whether in conjunction with the neighbours' concentration term each protein must also either have been generated (11), or not generated (00) by the cell itself to activate, with the other two bit combinations representing don't care terms. These terms are similar to those used by the original developmental rule introduced in Chapter 4, and the entire rule structure is closely related to the *outer* totalistic cellular automata described by Wolfram (Wolfram, 2002).

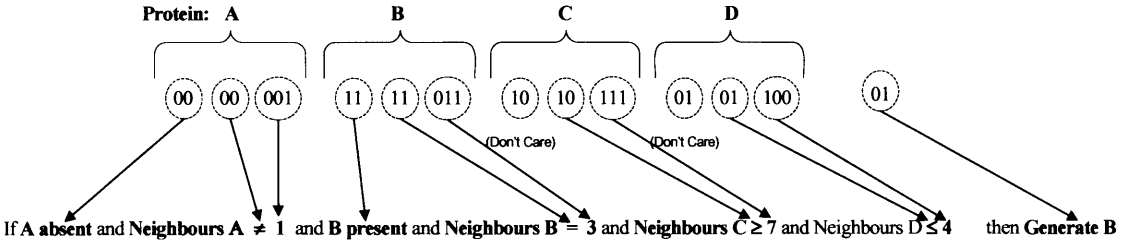


Figure 5.1.3: The Outer Totalistic developmental rule

Many of the target patterns used earlier were designed so that it was possible to design and encode a set of developmental rules manually that maps each pattern to an optimal adder design. Using the new outer totalistic rule structure, these patterns could not be easily mapped to an optimal adder design as the pattern-to-circuit mapping relied on each rule being able to detect the *collective* amount of protein generated by both a cell and its neighbours. However the new outer totalistic rule structure makes it possible to use a different set of mapping rules that ignore any proteins generated by neighbouring cells and instead depend purely on the proteins generated by the cells themselves. The target pattern was redesigned with such a mapping in mind, which meant that simplified fitness function could be used. As all the proteins generated by each cell were now done so at unit concentrations, the fitness function could once again be based on the Hamming distance between the candidate pattern of proteins and the target pattern of proteins, as used for the first experiments of this chapter. In the case of the 3x5 array used here the longest distance between two patterns was 60. Hence fitness was set as the measured distance between the target and candidate solution subtracted from 60, giving a maximum fitness of 60, and a minimum of zero. The initial conditions were set as in the previous experiments, except no protein contributions from neighbouring cells were set. This pattern is shown in the first column of Table 5.1.6. Just as for Experiments 2A-E, 20 runs of the experiment were carried out. The genetic and developmental parameters were also unchanged from the earlier experiments. The results of these experiments are shown in Table 5.1.7 as Experiment 3A.

Initial Conditions (Set in Current Cell)	Target Protein Contribution from Local Cell (3A & 3B)	Target Protein Contribution from Neighbouring Cells (3B)	Protein Key
00 00 00 00 00 00	10 10 00 00 00 10	11 10 10 00 20 01	Where the concentration values shown are for the following proteins: AB CD AB CD AB CD
00 00 00 00 00 00	01 00 00 00 10 01	20 21 10 10 01 20	AB CD AB CD AB CD
00 00 00 00 00 00	10 10 10 00 00 00	12 20 10 00 20 02	AB CD AB CD AB CD
00 00 00 00 00 00	01 00 00 00 10 01	20 21 10 10 01 20	AB CD AB CD AB CD
10 01 00 00 00 10	10 10 00 00 00 10	11 10 10 00 20 01	AB CD AB CD AB CD

Table 5.1.6: Initial conditions and target patterns for Experiments 3A and 3B.

Expt.	Best / Max Fitness	No. of Optimal Runs	% Optimal Runs	Mean (Best Fitnesses)	Std. Dev. (Best Fitnesses)	Normalised Mean (Best Fitnesses) /100	Normalised Std. Dev (Best Fitnesses)
3A	60/60	3	15	56.05	2.37	93.42	3.95
3B	420/420	2	6.67	393.0	11.90	93.57	2.83
2E	410/420	0	0	405	9.10	96.43	2.17

Table 5.1.7: Results of evolving target patterns on and 3x5 cell arrays with the totalistic system.

The results of Experiment 3A show that three runs of the 20 carried out resulted in the discovery of the target pattern by evolution, whereas when applied to the general totalistic model of Experiments 2E, evolution never succeeded in discovering an optimal solution. This suggests that either the evolved optimal solution, or a class of intermediate solutions that evolution is likely to discover, makes use of the ability to differentiate between a protein generated by a cell and one generated by its neighbours. It is unlikely that the increased performance of the system used in Experiment 3A over that of Experiment 2E has arisen purely by chance. However care must be taken when comparing the two results to determine the effect of the change in communication strategy as both the communication strategy and the fitness function have been altered between the experiments. The change in the fitness function may well result in a more evolvable fitness landscape, as the fitness contribution of each cell in the phenotype is now independent of other cells. This means that the evolution can evaluate the states of each cell independently of each other should it find a mechanism to modify them independently. Any comparison between the two models should take into account whether this potential change in evolvability is likely to remain in a system that maps patterns to a circuit design and calculates fitness from the behaviour of the circuit, rather than directly from a pattern of proteins. With the new rule structure, it is in fact possible to design rules that map a cell's individual protein contributions directly to structural modifications, thus take advantage of this property in the same way as the new fitness function. Indeed this was the approach taken when designing target pattern for Experiment 3A. However it is likely that evolution will at least explore mappings that make use of the protein contributions from neighbouring cells in the rules, as the structure

of the rules provides the ability to do so. Consequently it is possible that any benefit in evolvability provided by the new fitness function will not be transferred to a circuit design problem in full, as there is no hard bias towards using the (potentially) more evolvable rules.

Experiment 3B: Outer Totalistic vs. Inner Totalistic Development

To explore whether the new fitness function really did contribute to the enhanced evolvability of the outer totalistic system, a new fitness function was devised. The target pattern of local protein generation was kept as Experiment 3A. An additional target pattern was then derived from the local generation pattern by considering how each protein generated would be detected in neighbouring cells. This pattern can be seen in the third column of Table 5.1.5. Fitness was calculated in each cell and for each protein by measuring the difference between the local protein contribution in the candidate individual from the target local contribution, multiplying it by a factor of three and adding it to the difference between the neighbours' contributions and the target neighbours contribution. This modelled the fitness calculation carried out in Experiment 2E, hence the results of the current experiment are more comparable with Experiment 2E than those of Experiment 3A. The experiment was run with genetic and developmental parameters identical to those used in Experiment 3A. The results are shown in Table 5.1.7 as Experiment 3B. A comparison of the results from Experiment 3A and 3B show that optimal solutions are found in both cases, and at a similar rate (three optimal solutions were found in Experiment 3A and two in Experiment 3B). Additionally both experiments outperform Experiment 2E in terms of the number of optimal solutions found, implying that the increased performance of Experiment 3A over Experiment 2E was not entirely as a result of fitness relying on a contribution from neighbouring cells. This suggests that the improvement in performance results from an improvement in the communication strategy used by outer totalistic development rather than from any benefit provided by the change in fitness function, although as the sample sizes are low it cannot be said that the fitness function of 3A does not aid evolvability in some way.

Experiments 3A and 3B demonstrate that the outer totalistic communication strategy used in this developmental model is powerful enough to generate patterns that are potentially useful for hardware evolution. One further experiment was carried using this system. Most developmental parameters had been selected on rational grounds. However the number of rules used by the system had been selected in a fairly arbitrary manner using informal preliminary experiments. To ensure that this parameter was sensible, Experiment 3B was repeated using various numbers of developmental rules. The results obtained did not provide an obvious choice for the parameter value, so the experiment was repeated for a further ten runs for each rule length, making a total of 30 runs each. These results are shown in Table 5.1.8.

Analysis of the results still could not yield a definitive value for the parameter. As the number of runs that discover an optimal solution is very low in all the experiments presented in Table

5.1.8 it would not be wise to consider any measured difference as valid. Because of this the mean fitness of the best solutions ($\overline{f}_{best}$) found in each run provides a better source for analysis. Comparison of these values suggests that the optimal number of rules, at least for this problem, lies at around 20 rules. However even the variation between experiments in this measure is well within a single standard deviation of the results obtained for each experiment, and so this conclusion should again be treated with caution. T-tests were carried between each population of results, and are also tabulated in Table 5.1.8 where each value listed shows the test between the population on the current row and the population in the row below. Again the t-tests no not reveal significant differences in performance between adjacent experiments, except in the case moving from 20 to 25 rules, which may be a statistical aberration. Hence as the results suggest that changing the number of rules has little effect, it was decided that it was safe to set the parameter at the tentatively optimal value of 20 suggested by these experiments.

Number of Rules	No. of Runs Producing Optimal Solution	% Runs Producing Optimal Solution	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. Of Best Solutions (σ_{best})	T-Test (Current Row, Proceeding Row)
10	2	6.67	393	11.9	0.147
15	4	13.33	396.23	12.70	0.139
20	3	10	399.8	12.49	0.028
25	1	3.33	393.07	14.02	0.138
30	1	3.33	396.56	10.27	0.235
60	2	6.67	394.13	15.11	N/A

Table 5.1.8: Results from evolution of the adder target pattern with various number of rules using the outer totalistic system.

5.2 Exploring the Formation of Robust Regulatory Networks

Section 5.2 has dealt with improving the communication strategy used by development so that it is capable of generating patterns that may be useful for hardware evolution. Another factor that might be limiting the performance of the system which was identified at the start of this chapter is the activation and response strategies used. Boolean strategies might have difficulty forming regulatory networks as altering a rule through a genetic operation can abruptly remove any regulatory relationship it was with other rules, as was discussed in Section 4.5. Section 4.5 also noted that if the activation and / or response strategy were graded rather than Boolean, although a mutation might still be detrimental to the strength of the relationship between the rules, the relationship might still exist in a weakened form, thus be more robust to genetic change. However implementation of such graded strategies is likely to result in an algorithm with greater computational complexity than if Boolean strategies are used, hence they are less attractive for hardware evolution.

One way that a network of interacting Boolean rules might achieve some level of robustness is if it each rule was replicated, thus providing an identical redundant relationship between the

regulatory proteins involved. Evolution would then be free to modify one of these rules in order to explore related and potentially more useful regulatory processes without abruptly removing the original relationship and losing whatever contribution it made to the regulatory process. Once a more useful relationship had been found by evolution, the original rule could be discarded.

It might be possible to achieve such a process by duplicating rules. If this process was placed under evolutionary control by introducing a rule duplication operator, evolution could apply it when necessary. The following section explores this idea. As the focus of this section is how duplication affects gene regulation it is once again useful to decompose development into pattern formation and mapping processes, and explore duplication only in the context of pattern generation.

5.2.1 Experiment 4A: Rule Duplication

Ideally the model of development used for the rule duplication experiments would have been identical to the model developed at the end of Section 5.1. However the chromosome used in Section 5.1 was of fixed length. If a rule duplication operator were introduced to such a model, it would have to overwrite rules already present in the chromosome. If the overwritten rule implemented a useful regulatory relationship, the fitness of the resulting regulatory system would likely be reduced. Hence even if duplication increased the robustness of another useful relationship to future genetic change, the immediate effect would be negative, thus the more robust chromosome would be less likely to survive until its increased robustness could be of use. If evolution were allowed to duplicate a rule without immediately removing another, it might measure the effect of the improved robustness more easily.

Because of this, the system was modified to allow for variable length chromosomes. In addition, both rule duplication and rule removal operators were introduced. They were designed so that a rule would be selected at random for duplication or removal according to a fixed probability. In the case of duplication, the duplicated copy was inserted into the chromosome following the initial copy. The genetic operators were applied in the order: duplication, removal, crossover, mutation. Allowing variable length chromosomes also meant that four new parameters were introduced: the minimum and maximum length of a chromosome, and the duplication and removal rates. The minimum chromosome length was set at five rules to ensure that there was always a reasonable opportunity for the formation of regulatory networks, and the maximum length was set at 150 rules, to allow ample opportunity for duplication. Initial experiments set the duplication and removal rates to a rather arbitrary 0.75. The results of runs carried out using these parameters showed a much lower performance than when no duplication or removal was used. Examination of the best solutions from these runs revealed that the system was suffering

from bloat: rapid duplication occurred within the first few generations until the maximum chromosome size was reached. This is unsurprising, as most evolutionary algorithms that operate on variable length structures (for example genetic programming) suffer from this problem. The resulting search spaces were extremely large, thus evolution could not find reasonable solutions. To combat this problem the duplication rate was reduced to 0.001. This value was selected through informal experimentation so that the maximum chromosome size was reached after around 5000 generations, the maximum number of generations in this experiment. All other genetic parameters in this set of experiments were identical to those used in Experiment 3B, except the initial number of rules, which was set to 15. This was just below the 20 rules that the final experiment of Section 2.1 suggested, thus allowing some duplication to occur before the optimum was reached. The results of 50 runs evolving the target pattern of Experiment 3B with an identical fitness function are shown in Table 5.2.1 as Experiment 4A. The results suggest a clear improvement over the results obtained without duplication in Experiment 3B, with the number of runs discovering the target pattern increasing by over 100%. To confirm this, Experiment 3B was repeated until a total of 50 runs had been completed. These results are shown in Table 5.2.1 as Experiment 3B(2). Comparison with the results after the introduction of duplication shows that the number of optimal runs increased by some margin, but the $\overline{f}_{best}$ values were still quite similar. A t-test comparing $\overline{f}_{best}$ from the two sets of 50 results was calculated at 0.001, suggesting that although small, the difference in $\overline{f}_{best}$ was significant.

Expt.	Best / Max Fitness	No. of Optimal Runs	% Optimal Runs	Mean (Best Fitnesses) ($\overline{f}_{best}$)	Std. Dev. (Best Fitnesses) (σ_{best})
3B(2)	420/420	4	8	398.05	13.65
4A	420/420	9	18	401.82	13.87
4B	420/420	7	14	400.36	12.10

Table 5.2.1: Results of experiments involving rule duplication and removal.

5.2.2 Experiment 4B: Random Rule Insertion

To confirm that the improvement in performance brought about by duplication was due to an improvement in robustness through rule redundancy, the experiment was repeated, but this time when the duplication operator was applied, a randomly generated rule was inserted into the chromosome following the rule to be duplicated. As the inserted rule was randomly generated it would be unlikely to replicate a pathway already present, hence redundancy would not be introduced. The results are shown as Experiment 4B in Table 5.2.1. Surprisingly the results suggest that random insertion of rules produced a similar improvement to true rule duplication. A t-test comparing Experiment 4A and 4B gave a value of 0.29, suggesting that the difference in $\overline{f}_{best}$ was not significant. However a t-test comparing Experiment 3B(2) and 4B produced a value of 0.004. This suggests that a significant improvement in performance is gained not

through introducing rule duplication per se, but merely from some aspect of the new variable-length representation. One possible explanation for this was first offered by Harvey (Harvey, 1991) when introducing the Species Adaptation Genetic Algorithm (SAGA), which was briefly discussed in the context of neutral networks in Section 2.1.5. He suggested that large problem spaces might be better searched by evolving a converged species of solutions, and that the process would be facilitated if chromosome length increased slowly over the course of evolution, allowing evolution to incrementally improve the design of the species by exploring increasingly more complex solutions. A simple increase in chromosome length is exactly what is happening in Experiment 4B, and it is possible that evolution is using a mechanism such as that described by Harvey to discover more complex sets of interacting rules. Although it would be interesting to explore the theories that underlie SAGA and whether they model the evolution of developmental rules well, it is not the focus of this thesis. All that will be said at this point is that from the work presented here it cannot be concluded that duplication benefits the robustness of Boolean rule strategies by providing redundancy. Although it appears to provide a significant performance benefit over a fixed length system without duplication the improvement is fairly small. As the mechanism of this improvement is unclear, it was decided not to include a variable length representation or any duplication or removal operators in further experiments for fear that the use of an unexplained and non-general phenomenon might obfuscate the results, and reduce the interest of this thesis to the community. Hence the next set of experiments return to the regulatory system of Experiment 3B as a basis for further development.

5.3 Mapping Regulatory Networks to Circuit Designs

The work presented so far in this chapter has investigated whether the model of biological pattern formation using regulatory genes introduced in Chapter 4 was suited to hardware evolution problems. It was discovered that the model was biased towards patterns that were not particularly useful, and changes were introduced to bias it towards patterns that were potentially more useful. Likewise the mapping process is biased towards the generation of particular types of circuit. The results presented in the following experiments suggest that the current bias imposed by the mapping process is not useful for the generation of circuits. Hence this section proceeds by introducing changes to the system that apply a bias towards more useful circuits.

Just as analysis is simplified by partitioning development into pattern formation and mapping stages, analysis of the mapping stage can be simplified by decomposing it into processes that map proteins to alterations in the *logic* of the virtual architecture and processes that map proteins to alterations in the *routing* of the architecture, and exploring each of these in turn. Of course an argument could be made that evolution is unlikely to partition the problem in a top-down manner, as was noted when exploring pattern formation. However Chapter 4 presented

several evolved modules that solely encoded routing, so there is evidence to show that at least in come cases evolution does partition the problem in this way.

5.3.1 Exploring Protein to Logic Mappings

Exploration of the mapping process began with exploring the biases inherent in logic generation. To do this it was necessary to fix the routing in way that was both amenable to the design of a useful circuit, and likely to be discovered by evolution. Again the test problem is the two bit adder with carry problem. To ensure that the routing was fixed in a way that optimal adder circuits could be generated, another adder was designed manually. The previous design was not used as there was no evidence that the design of the routing could have been discovered by evolution. Chapter 4 presented a circuit that had been found by evolution, and had identified routing modules within the design. It was noted that they were very general, and iterated through every cell. It was decided to mirror this feature in the design of a new optimal adder, so the new circuit was designed so that the routing between every CLB was identical: each CLB received inputs from the F-LUT to the north, the G-LUT to the south and the F-LUT to the west, and all CLB outputs were switched on. By fixing the routing in this way a new virtual architecture was effectively imposed. This new virtual architecture also went some way towards addressing an issue identified in the analysis of Chapter 4 that may affect the ability of evolution to discover useful protein-circuit mappings: the number of structural proteins available to evolution in the exploratory system was large, because the granularity of the virtual CLB is still relatively coarse. It may be difficult for evolution to discover a set of useful circuit alterations within them. The fixed-routing virtual architecture reduces each CLB to two three input LUTs, which requires only 16 structural proteins to alter the state of minterms within the LUTs (eight for each LUT). The hand-designed optimal adder composed of the new fixed routing virtual architecture CLBs is shown in Figure 5.3.1(a).

One further change was made. The experiments in Chapter 4 and the adder designed in Section 5.1 all received input signals from the west edge of the circuit. The adder designed in this section assumed that the input carry signal was again received in the south-western cell, but this time from the south rather than from the west. This allowed the design to be more modular, hence it would likely provide an easier route to the reuse of elements of the design. A fixed pattern of proteins that could be used to map to the hand-designed circuit was also devised, in the same way as the patterns used in Section 5.1 were devised. This pattern is shown in Figure 5.3.1(b).

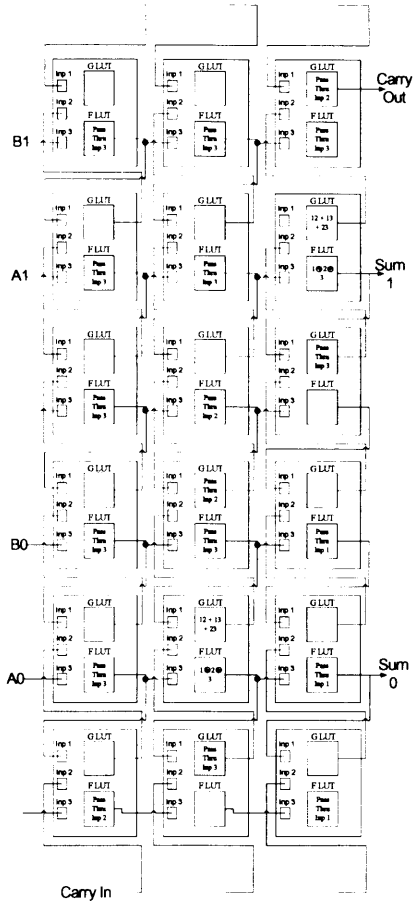


Figure 5.3.1(a): A hand-designed adder based on a new virtual architecture with fixed routing.

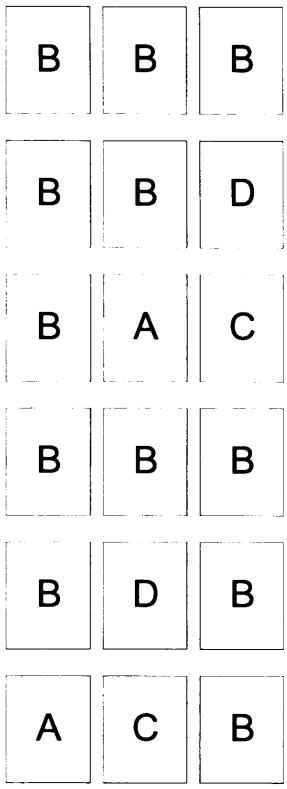


Figure 5.3.1(b): A pattern of proteins that distinguishes cells using different logic.

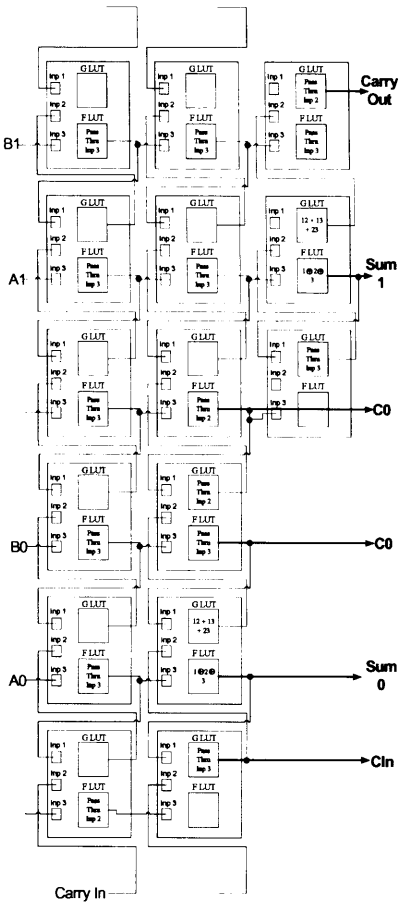


Figure 5.3.1(c): The adder of 5.3.1(a) with additional probe points used in Experiment 5E.

A question remained as to how the new, smaller set of structural proteins should be encoded within developmental rules. The exploratory system of Chapter 4 used a single class of rules that encoded both regulatory and structural proteins, depending on value of the seven bit postcondition. No structural proteins were used in the pattern generation experiments of Section 5.1. Instead one of four regulatory proteins was represented by a two bit postcondition. The reduction in postcondition size increased the likelihood that randomly generated rules would interact. As this may be important for successful generation of the test patterns, it was decided that rather than extend the size of the postcondition to allow structural proteins to be represented by the same class of rule, a new class of rule would be introduced that encoded structural proteins only, and a set of these rules would be evolved alongside a set of rules that encoded regulatory proteins only. The new class of rule had an identical structure to the outer totalistic rules used in Section 5.1, except it used a three bit postcondition that represented 16 unique structural proteins, each of which would alter the state of a minterm within one of the three input LUTs. Consequently the chromosome of the new design consisted of two sections: first a set of regulatory rules and second a set of structural rules. To determine how many structural rules might be required, the hand-designed circuit shown in Figure 5.3.1(a) was analysed to determine how many rules were needed to map the pattern shown in Figure 5.3.1(b) to the

circuit manually. 23 rules were required in total, so the number of structural rules in the chromosome was set at 30, to allow for mappings less optimal than the hand-designed one, and to allow for some neutrality in the model which may benefit evolvability.

Experiment 5A: Mapping a Pattern to a Circuit

An experiment was conducted to test whether the simplified system outlined above could achieve a successful mapping between the test pattern and an optimal circuit on a 3x5 array of three input LUTs. The chromosome consisted of 20 regulatory rules (suggested as the optimal number by the final experiments of Section 2.1) and 30 structural rules. All other parameters were set as the experiments conducted in Section 5.1. Each of the five runs was stopped after 2500 generations, identical to the number of generations used in Chapter 4. The results are shown in Table 5.3.1 as Experiment 5A.

Experiment	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. of Best Solutions (σ_{best})	Best Fitness / Max Fitness	% of Optimal Solutions
Exploratory	80.20	3.30	84 / 96	0
5A	77.20	5.26	83 / 96	0
5B	78.0	12.35	96 / 96	20

Table 5.3.1: Results of experiments exploring the original pattern to circuit mapping model.

The results do not suggest that the system is an improvement over the exploratory system of Chapter 4, which are provided in the same table for comparison. However few additional conclusions can be drawn through the comparison, as the systems differed greatly: the target problems, the search spaces, the architectures used and the size of the arrays are all different, in addition to the changes made to the design of the system. What can be said is that even with the improved pattern formation mechanism the system is not capable of evolving optimal solutions to the test problem. Hence further exploration of the mapping biases was warranted.

Experiment 5B: Mapping using Only Structural Rules

Another experiment was conducted to test the ability of evolution to evolve useful mappings. To limit the evolutionary search to mapping only, the chromosome evolved consisted only of 30 structural rules. No regulatory rules were evolved. Instead the initial conditions were set to the protein pattern shown in Figure 5.3.1(b). All other genetic and developmental parameters were identical to Experiment 5A. The results are shown in Table 5.3.1 as Experiment 5B. One optimal solution was discovered. It was expected that the problem of discovering a logic map should be easier for than if the pattern formation, routing and logic problems were attempted together. The results suggest that even this simple problem is actually quite difficult for evolution to solve, and that the system might benefit from changes to the mapping mechanism. To determine changes that may be useful the mapping bias was considered in more detail.

Analysing Bias In Logic Mapping

The hand-designed circuit uses a number of different LUT configurations. Some of these route incoming signals through the CLB and some of these perform logical operations to calculate sums and carrys. The design of the mapping process is biased towards the generation of particular LUT configurations. A useful, reusable module for the generation of a particular logical function could consist of a set of rules with identical preconditions, so that they would only be activated in identical contexts. Each rule would encode a different minterm, and the entire set would work together to generate the logical function required. For evolution to achieve this, each rule would have to be discovered individually. Hence LUT configurations with a greater number of minterms set are likely to be more difficult to evolve. An experiment was carried out to confirm this. A single cell (that mapped to a three input LUT) was evolved using the system described above. Fitness was measured by applying every possible input combination and summing the number of times the single output bit matched the target logical function. Hence the maximum fitness was eight. The experiment was repeated using every possible function of the three input LUT as the target function. Ten runs were conducted for each function. The problem was so simple that the stopping condition was set as the moment that the first optimal solution for each function was discovered, rather than at a fixed number of generations. In all cases the initial conditions for the cell were set so that protein A was present and all other proteins were absent. The mean number of generations taken to find the optimal solution was calculated for each logical function. A graph of these means was plotted against the number of minterms in each function is shown in Figure 5.3.2. Least-squares regressions were calculated for these data, using both a linear and second-order polynomial model. The coefficient of determination, r^2 is a statistic that measures how well the model fits the data and ranges from 1 for a perfect fit to 0 for random data. Using this measure the polynomial model fit the data well, yielding an r^2 value of 0.89. This line of best fit is also displayed in Figure 5.3.2. Based on this statistic, it can be said with reasonable confidence that there exists a bias against functions that require greater number of minterms to be set, and the more complex functions appear to be quadratically harder to discover.

The functions used in the hand-designed adder are of the most common type, all requiring four minterms to be set. Hence they are fairly difficult to discover. It is likely that this bias means that evolution will tend to explore circuits composed of the simpler functions hence discovery of the design shown in Figure 5.3.1 might be quite difficult.

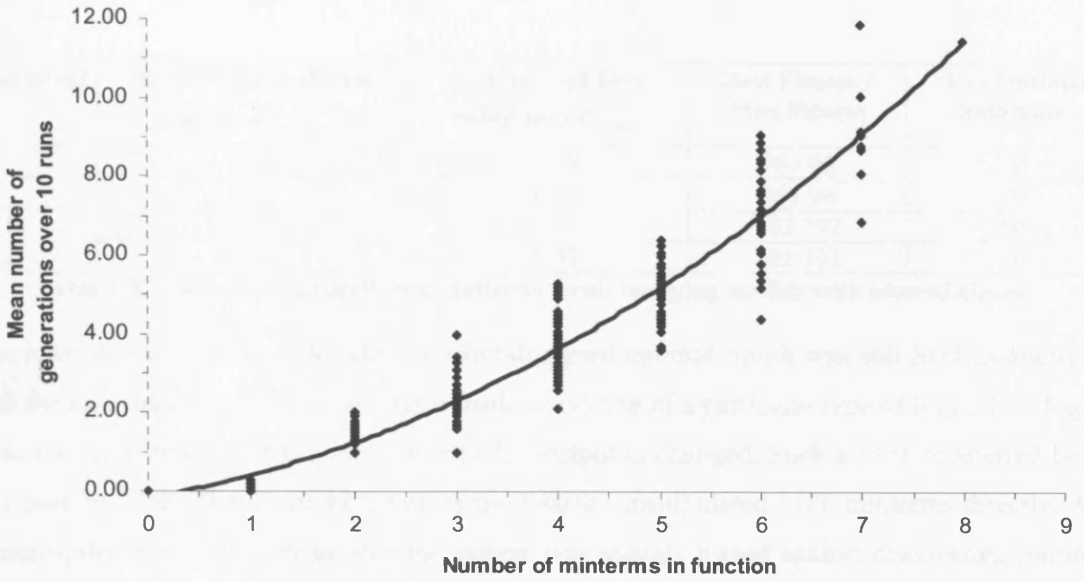


Figure 5.3.2: A graph of the number of generations required to evolve all functions of a three bit LUT.

Experiment 5C: Mapping with Bias-Free Structural Proteins

To address this issue a new set of structural proteins was devised that was not biased towards the generation of any particular logical function. Any bias towards particular logical functions would therefore be provided by the evolutionary algorithm, based on the fitness of the evolving circuits. Whereas previously a structural protein existed to alter each minterm in each LUT, each structural protein in the new set of proteins represented the entire state of a CLB, each represented by a unique 16 bit postcondition. Each postcondition was interpreted as a concatenation of the F and G-LUT configuration bitstrings. Although this represented a huge increase in the number of postconditions open to evolution, the new representation meant that there was no bias inherent towards any logical function in the development process. Bias would instead be imposed by the mutation, crossover and selection operators of the evolutionary algorithm. Furthermore only six of the new rules were needed to encode the target circuit, so even though the search space of all possible single rules was larger, the total evolutionary search space could in fact be reduced. The performance of the system using this new rule structure was tested by repeating Experiment 5B, but using the new structural rules. As the map between the protein pattern and an optimal circuit could be now achieved using only six rules the chromosome for this experiment included ten structural rules, so that evolution could explore solutions that used more than the minimum number of rules required. All other parameters were identical to those used in Experiment 5B. The results of the experiment are shown as Experiment 5C in Table 5.3.2. It can be seen that the results are fairly similar to that of

Experiment 5B (shown in Table 5.3.1), the difference in $\overline{f}_{best}$ being considerably less than the standard deviation of $\overline{f}_{best}$ in both experiments.

Experiment	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. of Best Solutions (σ_{best})	Best Fitness / Max Fitness	% of Optimal Solutions
5C	74.0	12.39	96 / 96	20
5D	80.8	9.96	96 / 96	20
5E (1)	178.40	19.92	192/192	60
5E (2)	176.15	17.77	192/192	50

Table 5.3.2: Results of experiments pattern-circuit mapping models with altered biases.

One possible explanation as to why the hand-designed optimal circuit was still hard to discover with the new mapping bias is that circuit made heavy use of a particular type of logic. This logic *routed* a signal from the LUT input to the LUT output unchanged. Such a LUT is referred here as a routing LUT. The structural proteins used earlier manipulated LUT minterms directly. As demonstrated earlier this meant that the system was slightly biased against discovering routing logic. The structural proteins introduced in Experiment 5C altered this bias so that there was no bias towards any particular logical function. However, with no bias, routing LUTs are still quite difficult to discover: only three of the 256 possible LUT configurations are routing LUTs, thus the probability of discovering one by chance is low. Of course a probability of 3/256 would be a worst-case scenario during evolution, as the LUT inputs are not always driven by logical functions. Instead some of the inputs may be constant, in which case some of the more complex logical functions would reduce to a routing LUT. However such LUTs are unlikely to be evolvable as they would not generalise well to other contexts, and even including such LUTs the probability of discovering any routing LUT is still low. Hence if the aim is to discover a circuit similar to the hand-designed circuit, it may be useful to provide a bias towards routing logic. This reasoning is supported by the work of Miller and Thomson discussed in Section 2.1.5 where it was noted that a bias towards routing logic can improve the evolvability of circuit design problems (Miller and Thomson, 1998a).

Experiment 5D: Applying a Bias Towards Routing

To explore whether this would benefit the evolution of optimal circuits, the structural proteins were altered to include such a bias. Two bits were inserted into each protein prior to the G-LUT configuration bitstring, and two bits were inserted prior to the F-LUT configuration bitstring. Hence the total length of each structural protein (and consequently the structural rule postconditions) was increased from 32 to 36 bits. Each of the new two bit codes were interpreted thus: if both bits were high (11) the proceeding 16 bits were interpreted as a LUT configuration bitstring, just as before. If the two bit code was 10, 01 or 00, the proceeding 16 bits were ignored and the LUT logic set so that it routed input 1, 2 or 3 from input to output respectively. The old and new structural proteins are shown in Figure 5.3.3 for comparison. An

experiment was conducted to test this alteration. All parameters were unchanged from Experiment 5C. The results are shown as Experiment 5D in Table 5.3.2. They suggest that the change might result in a slight improvement in performance, although again the sample size and deviations mean that this is not conclusive. Certainly the new protein structure did not provide evolution with a strong bias towards optimal circuits.

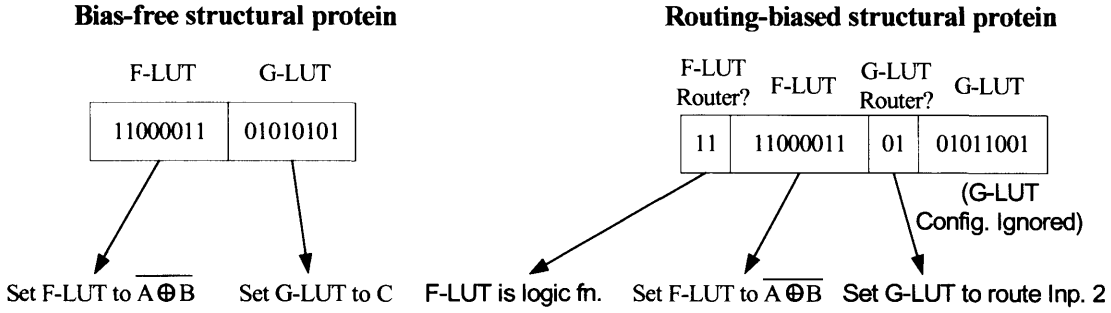


Figure 5.3.3: The structural proteins introduced to (a) remove bias towards any logic function and (b) to provide bias towards logic that routes signals.

Examination of the best solutions of each run in Experiment 5D revealed trends in the way evolution solved the problem. Most of these solutions generated signal S0 correctly, but only the optimal solution generated signals S1 and COut correctly. Of the four suboptimal runs, all failed to increase the fitness contribution of the S1 signal from that of its initial constant state. Three of them effectively routed input B0 to the COut probe point, thus realising a modest increase in fitness, with the fourth failing to increase the COut contribution from the initial constant state.

Credit Assignment

One way of explaining these apparent trends is from the perspective of credit assignment. The circuit is made up of a collection of logical functions which are generated by rules. For a behaviour to be realised a number of rules must interact with each other, ideally as a module. Consider the most common type of evolutionary run described above: one that has discovered a circuit that generates S0 correctly, but has failed to evolve any increase in fitness contribution from the S1 signal of the circuit or a perfect COut signal. Correct generation of S1 in the hand-designed circuit uses a carry signal that combines information about the Cin, A0 and B0 inputs. The generation and routing of this signal to the most significant addition site requires nine LUT configurations to be set. For any information to be routed from CIn to the S1 probe point requires at least seven LUTs to be set. If evolution is to discover that such information is important to the second sum it must set around this number of LUTs so that there is a signal path between input and output before it receives a fitness reward. This means that evolution would by chance have to discover a large number of rules that interact with each other in a useful way (ideally as a module) before the rules are assigned credit for their contribution to the

solution. Similarly, the discovery of a perfect COut signal would result in an even longer signal route, hence a greater credit assignment problem unless an internal carry from the least significant bits had already been discovered. However the solutions of Experiment 5D demonstrate that evolution can discover that B1 contributes to COut. Evolution only needs to discover two LUT configurations that route B1 to COut before it receives a fitness contribution which it can assign to the rules responsible for the increased fitness. This means that evolution can assign credit to the use of B1 in the generation of COut much more easily than the use of A0, B0 and CIn.

Experiment 5E: Biasing Circuit Structure using Intermediate Signals

One way of easing the credit assignment problem would be to identify useful intermediate signals, such as the internal carry used in the hand-designed adder, and provide a bias towards the exploration of solutions that use them. This bias could be applied through the fitness function. To test this idea the fitness function was altered and a new experiment was carried out. Previously the fitness function had been designed to test for the COut, S1 and S0 signals in the cells that generated these signals in the hand-designed optimal circuit. The new fitness function tested for two additional instances of the intermediate carry signal discussed above and one instance of the incoming carry signal (Cin) at the same positions in the array that were present in the hand-designed circuit. All the signals used in the new fitness function, and the positions within the array at which they were tested for are shown in Figure 5.3.1(c). This increased the maximum fitness to 192. All other parameters for this experiment were unchanged from Experiment 5D except the maximum number of generations was reduced to 1000, as experiments not reported here using a similar fitness function suggested that any improvement in fitness was extremely rare beyond this point. The results of this experiment are shown in Table 5.3.2 as Experiment 5E (1). Of the five runs, three discovered optimal solutions. To confirm that this apparent increase in performance was not a statistical aberration resulting from the small sample size, the experiment was run for an additional 15 runs. The results of the entire 20 runs are shown in Table 5.3.2 as Experiment 5E (2). The results show that biasing evolution to the discovery of circuits similar in structure to the hand-designed circuit clearly increased evolution's ability to discover a map between the predefined protein pattern and a functional adder.

Applying bias in this way is not necessarily a good idea. The bias that was added to the system effectively adds knowledge about traditional adder design to the algorithm. Introducing such knowledge might bias evolution away from innovative designs. Exploration of innovative designs was one of the motivations for introducing a developmental system. Additionally such knowledge is only available because a solution to the problem is already known. It would not be available for real-world problems. Furthermore the knowledge that was introduced required the

problem to be partitioned manually and information that might bias evolution towards this partitioning was selected manually (which in this case was the selection of signals to match a fairly arbitrary design). The resulting system is strongly biased towards a design that would otherwise be quite unevolvable. Another motivation for introducing development was that it might provide a mechanism for the automatic partitioning of a problem in a way that generates evolvable modules.

A New Target Adder

Rather than biasing evolution towards an arbitrary and perhaps not particularly evolvable target adder by introducing knowledge into the fitness function manually, it may be useful to consider what kinds of adders the natural biases of evolution and development are suited to discovering, and use such an adder as a target solution.

In the discussion above it was suggested that evolution struggles to solve problems that involves long signal paths as it cannot assign credit to long partial paths. If the inputs and outputs of the circuit were placed in a way that distributed inputs and outputs evenly across the circuit, and kept the path between inputs and outputs short, this might result in a more evolvable search space, as evolution would not have to infer large areas of a circuit design without any guide from fitness. Additionally the final pattern formation process used in this chapter was designed so that it is naturally biased towards particular target patterns. These were usually alternating stripes of two contexts, or columns of alternating contexts. If an adder could be devised so that it could be generated using such patterns, and presented inputs and outputs in the manner described above, it might prove much easier for evolution to discover than the rather arbitrary adders used earlier.

The design of such an adder is presented in Figure 5.3.4. It consists of a two columns of LUTs. The left hand column consists of alternating propagate and generate terms similar to those found in traditional adders. The second column also consists of alternating cells. The most southerly cell generates the Sum0 term by combining an incoming carry signal with the propagate term of the cell to the west. It also generates a partial carry signal. The cell to the north combines the partial carry signal with the generate signal from the west to generate a carry signal that can be used by the next cell in the chain. If evolution were to evolve this design it would only need to discover six LUT configurations to generate S1 before it received a fitness reward. Signals involving A0, B0 or C0 can be routed to the S1 probe point using four LUTs. However as the pattern of contexts is so regular, and the generation of S0 already relies on a propagate signal, it is possible that only four unique LUT configurations might have to be discovered before a fitness reward was received. One pattern of proteins that can be easily mapped to this circuit is shown in Figure 5.3.4. The pattern consists of two columns of alternating proteins. If evolution was able to discover a map from this pattern to an optimal circuit, it was expected that it would

be able to solve the entire development problem using the new input/output arrangement as it had already been demonstrated that such patterns could easily be found by evolution.

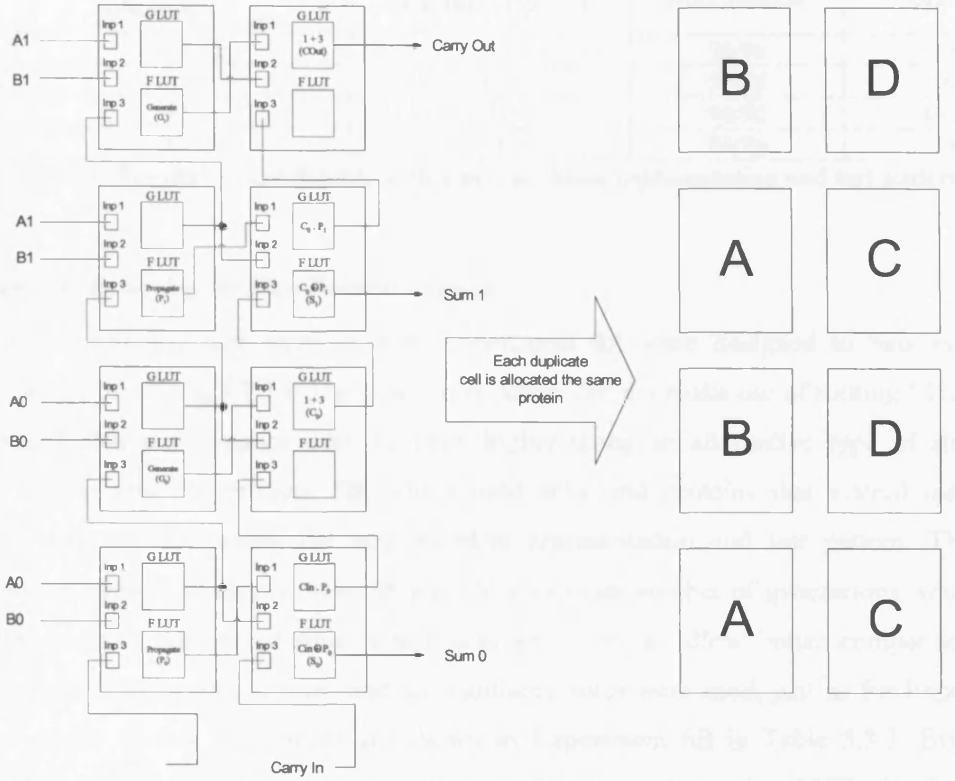


Figure 5.3.4: A hand-designed adder using a propagate-generate structure with fixed feedforward routing, and a pattern of proteins that distinguishes cells configured with different logic.

Experiment 6A: Mapping to the New Circuit Representation

An experiment was carried out to test whether evolution was able to discover such a map. A 2x4 array of virtual CLBs based on a fixed routing virtual architecture, suitable for implementing the design in Figure 5.3.4, was assigned for evolution. The inputs and outputs to the array were set as shown in Figure 5.3.4. This required a change in the fixed routing of the virtual architecture. Now input 1 was driven by the G-LUT to the west, input 2 was driven by the F-LUT to the west and input 3 was driven by the G-LUT to the south. Note that unlike earlier experiments this meant that evolution was exploring a feedforward architecture, so only logical functions could arise. The fitness function used was identical to that used in Experiment 5D, using only S0, S1 and C1 to measure fitness. It was anticipated that discovery of an optimal map would be easier for evolution than earlier experiments, so the maximum number of generations was reduced to 500 and 50 evolutionary runs were carried out. All other parameters were identical to Experiment 5D. The results of the experiment are shown in Table 5.3.3 as Experiment 6A. The results show a vast increase in performance over Experiment 5D, and suggest that performance has increased even over Experiment 5E which made use of prior knowledge of an optimal circuit design in the fitness function. (A further performance increase would be likely if the experiment was run for 1000 generations as with Experiment 5E.) Hence the new problem representation had produced a significant increase in performance over the experiments in set 5.

Experiment	Mean Fitness of Best Solutions ($\overline{f}_{best}$)	Std. Dev. of Best Solutions (σ_{best})	Best Fitness / Max Fitness	% of Optimal Solutions
6A	94.32	3.18	96/96	76
6B	96	0	96/96	100
6C	93.97	2.50	96/96	46.67
6D	85.69	12.19	96/96	26

Table 5.3.3: Results of experiments with a new problem representation and test pattern.

Experiment 6B: Removing the Bias Towards Routing

The structural proteins that were used in Experiment 6A were designed to bias evolution towards the use of routing LUTs. The new target design did not make use of routing LUTs, so it was expected that performance may be even higher using an alternative type of structural protein. To test this, Experiment 5B, which used structural proteins that altered individual minterms, was repeated using the new problem representation and test pattern. The only parameters changed from Experiment 5B was the maximum number of generations, which was set to 500 and the number of runs, which was set to 50, to allow better comparison with Experiment 6A. (30 structural rules and no regulatory rules were used, just as for Experiment 5B.) The results of this experiment are shown as Experiment 6B in Table 5.3.3. Every run discovered a perfect mapping, suggesting that the bias towards routing LUTs that had been imposed earlier was actually detrimental to performance with the new problem representation. As the sample size used in both experiments was relatively large, the measured performance increase is significant.

Experiment 6C: Evolving Developed Adders with Fixed Routing

Experiment 6B demonstrated that the biases inherent in the new representation combined with the mapping algorithm it used were suitable for mapping patterns of regulatory proteins evolved using the final regulatory model of Section 5.1 to optimal adder circuits. This suggested that evolution may be able to evolve such circuits from simple initial conditions. To test this the chromosome of Experiment 6B was extended to include 20 outer totalistic regulatory rules and a population of these chromosomes was evolved. The initial conditions were set as protein A present in the southwest cell and protein B in the southeast cell, but no neighbouring contributions were set in any cells. The fitness function used was identical to the other experiments of set 6, and the remaining developmental and evolutionary parameters were identical to Experiment 5A. The results of 30 runs of this experiment are shown in Table 5.3.3 as Experiment 6C. The results show that almost 50% of the runs found an optimal design, demonstrating that evolution was indeed capable of discovering optimal adder circuits from simple initial conditions by forming patterns of regulatory proteins and mapping these to a

circuit design using structural proteins. Analyses of some of these circuits that show how they are generated by development are presented in the next chapter in the context of scalability.

5.3.2 Exploring Logic and Routing

The experiments presented in the previous section have concentrated on evolving configurations for LUTs: the routing has been fixed to regular patterns similar to the routing generated by the modules presented in Chapter 4. Although the fixed-routing virtual architecture used in Experiment 6C could be used to demonstrate scalability, it constrained the types of circuit that could be evolved to combinational circuits. One of the aims for the planned scalability demonstration was to show that evolution can use development to search non-traditional design spaces and still find useful designs. Hence it would be useful if the developmental system was capable of manipulating routing in such a way that it could form time-recurrent loops of components, as it could with the virtual architecture presented in Chapter 4.

Experiment 6D: Evolving Developed Adders with Variable Routing

To allow evolution access to a richer solution space a new virtual architecture was imposed. Each CLB consisted of two three- input LUTs, with both LUTs in a CLB sharing the same three inputs, just as the fixed routing architectures used above. Again each CLB has two outputs, the outputs of the two LUTs. The routing available to each CLB was similar to the virtual architecture presented in Chapter 4. Connections were present between the F and G-LUT outputs and wires leading to the north, south, east and west neighbours. Both LUTs had their own independent routes in each direction, so both outputs of each CLB could be routed in any direction simultaneously. However unlike the outputs of Chapter 4, the connections between the outputs and the routing wires were permanent. Three CLB inputs were present as used earlier in this chapter. Each input was selected from the set of wires leading from the two LUT outputs of the north, south, east and west neighbouring CLBs to the current CLB, allowing the selection of any of the eight incoming routes. This architecture is shown in Figure 5.3.5(a) and (b).

New structural proteins had to be introduced into the model to allow evolution to manipulate the inputs. Just as in Chapter 4, eight of these proteins were introduced for each input, one for each possible input route, totalling 24 new structural proteins. Including the LUT proteins the model contained 40 distinct structural proteins. To encode these, the postcondition of the structural rules was extended to six bits, and a unique postcondition was assigned to each structural protein. Activation of rules encoding the remaining postcondition codes would have no effect on the phenotype. Other genetic and developmental parameters were identical to Experiment 6C, except 50 runs were carried out in total. The results of the experiment are shown in Table 5.3.3 as Experiment 6D. The results show that 26% of the runs evolved optimal adders, demonstrating that when using the model of development presented here, evolution can discover optimal solutions to the two bit adder with carry problem. Furthermore it achieves this in a search space

that contains circuits beyond the scope of traditional design. Detailed analyses of some of these circuits are presented in Chapter 6.

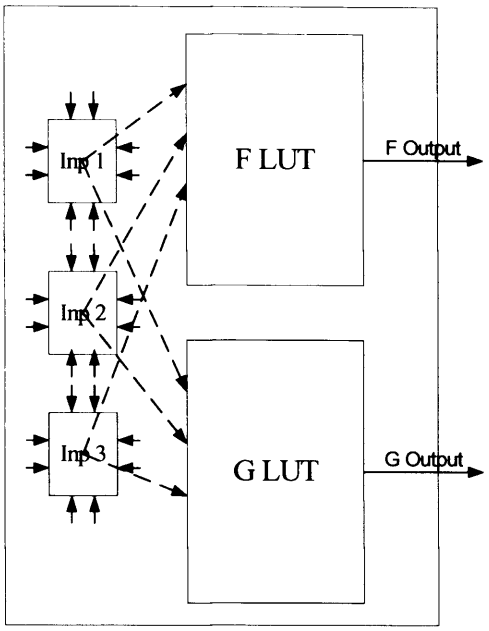


Figure 5.3.5(a): The three input virtual CLB.

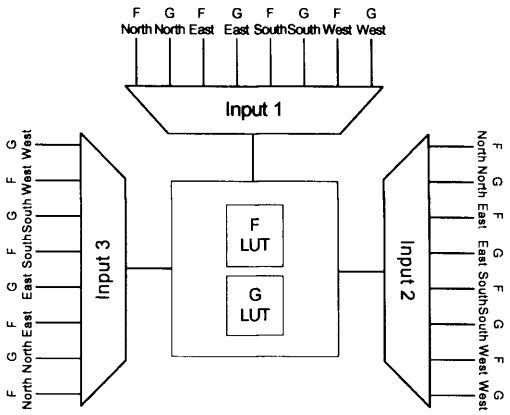


Figure 5.3.5(b): Inputs sources for the three input virtual CLB.

5.4 Summary of the Final Developmental Model

The model of development described in the experiment above was used to evolve fully functional two bit adders and will be used in the following chapter to demonstrate scalability. Many changes have been introduced to the model since it was first introduced in Chapter 4 hence the final model is now summarised.

It is called the Outer Totalistic model, and can be thought of as a two dimensional array of cells composed of two layers: a protein layer that models gene regulation and communication thus pattern formation, and an architecture layer that is used to map the patterns generated by the protein layer to a virtual FPGA configuration. Both layers and the virtual FPGA are now described.

5.4.1 The Protein Layer

The protein layer provides a means for evolution to *reuse* design innovations that it has discovered, hence is the primary means by which development is expected to enhance scalability. In this layer each cell contains a chemical environment that defines its state, or identity. A cell's environment is defined by a combination of four *virtual proteins*, labelled A to D. Each cell contains two sets of registers that are used to regulate the proteins: the *local environment* register and the *neighbouring environment* register, as shown in Figure 5.4.1. Their

function is described in the following paragraphs. The protein layer is implemented in software.

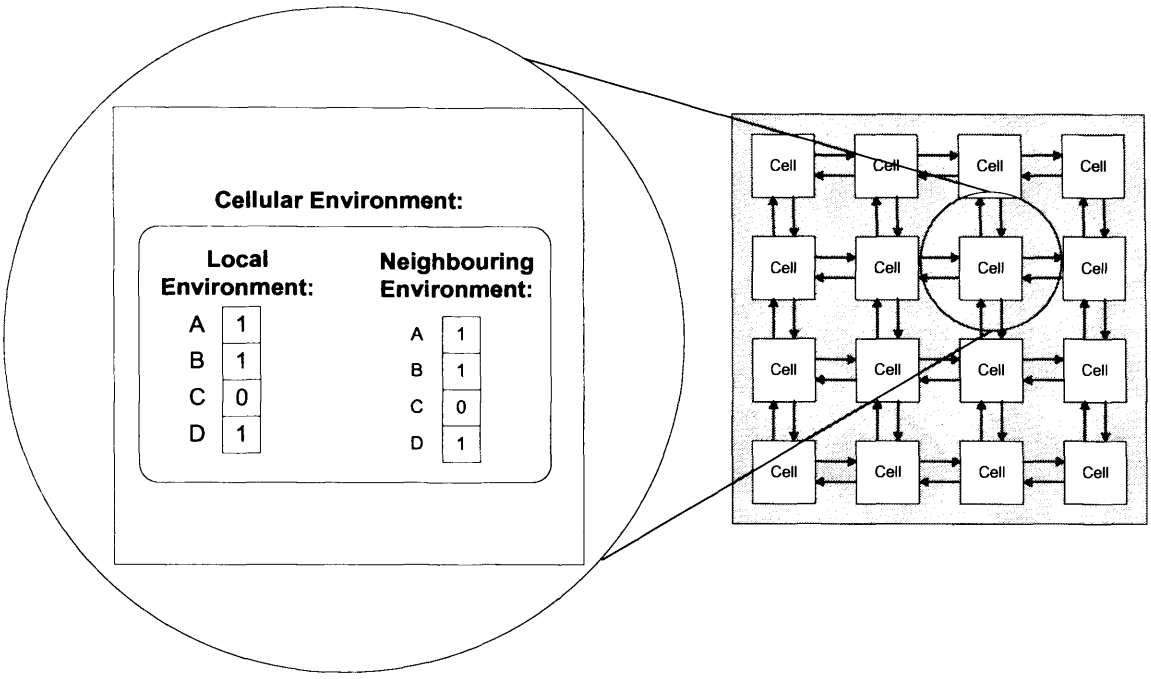


Figure 5.4.1: The contents of a cell in the protein layer.

In addition to the cellular array the protein layer contains a set of 20 rules called the *regulatory rule set.* This rule set describes how the environments of the cells change over the course of development. Development occurs over a series of discrete timesteps. At each timestep the environment of each cell is updated by determining the subset of regulatory rules that match the cell's current environment, and activating them simultaneously. When a rule is activated it generates a protein that makes up part of the environment of the cell at the next timestep.

Chemical Environment

A cell's chemical environment is divided into two areas: the *local environment* and the *neighbouring environment.* The local environment contains any proteins generated by a cell itself during the developmental timestep. Here each protein is represented by a Boolean variable: if no rules that generate the protein were activated in the previous timestep then it is absent, and if one or more rules that generate the protein were activated then it is present. The neighbouring environment is calculated from the proteins generated by the cell's Von Neumann neighbours at the previous timestep. Here each protein is represented by an integer and is calculated by summing the local environments of the cell's neighbours. Thus the maximum value for a protein is 4 (generated by all four Von Neumann neighbours) and the minimum is 0 (absent in all four Von Neumann Neighbours). This is called the *neighbouring concentration* of the protein.

Regulatory Rule Syntax

The precondition of each regulatory rule specifies what combination of the four virtual proteins must be present a cell's environment registers for that particular rule to activate. The rule structure is shown in Figure 5.4.2.

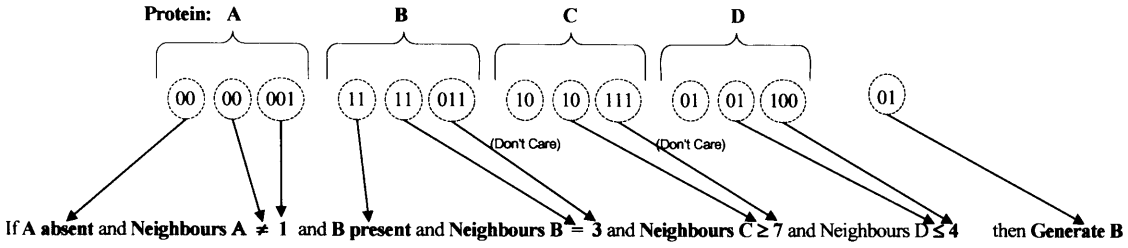


Figure 5.4.2: The Outer Totalistic rule structure.

The rule precondition is a conjunction of conditions that must be true for the rule to activate. There are two terms in each rule for each of the four proteins in the model. The first is a two bit condition that specifies what proteins must be present (11) or absent (00) in the local environment for the rule to activate, with the other two bit combinations representing don't care values. The second term is a five bit condition that defines how the neighbouring environment affects the activation of the rule: the first two bits define an operator, which is either an equality, inequality or one of two precedence operators, and the final three bits define the neighbouring concentration that the operator acts upon. As it is coded in three bits the rule can specify concentration values between 0 and 7 even though the maximum neighbouring concentration that can arise in the array is 4. This increases the number of "don't care" and "never activate" terms that can be produced using global and impossible events. The postcondition of a rule consists of two bits that define which protein is generated if the rule is activated.

Evolution acts on the rule set. The regulatory rule set makes up the first 600 bits of the chromosome, 20 rules each of 30 bits.

Protein Generation

The protein generation cycle is shown in Figure 5.4.3. At each timestep the developmental process cycles through every cell in the array, calculating the proteins that each cell will generate. First the subset of rules to be activated in each cell is determined by comparing the local and neighbouring environments of the cell against the entire rule set (Step 1 of Figure 5.4.3). If one or more of the rules in this subset specify a particular protein in their postcondition, this is recorded in the cell's local environment register (Step 2 of Figure 5.4.3). Once the local environment of all cells have been updated, the neighbouring environments are updated ready for the next developmental timestep (Step 3 of Figure 5.4.3): the values in the local environments of the cell's Von Neumann neighbours are summed giving neighbouring

concentrations for each of the four proteins, which are stored in the neighbouring environment register. The developmental timestep is then complete.

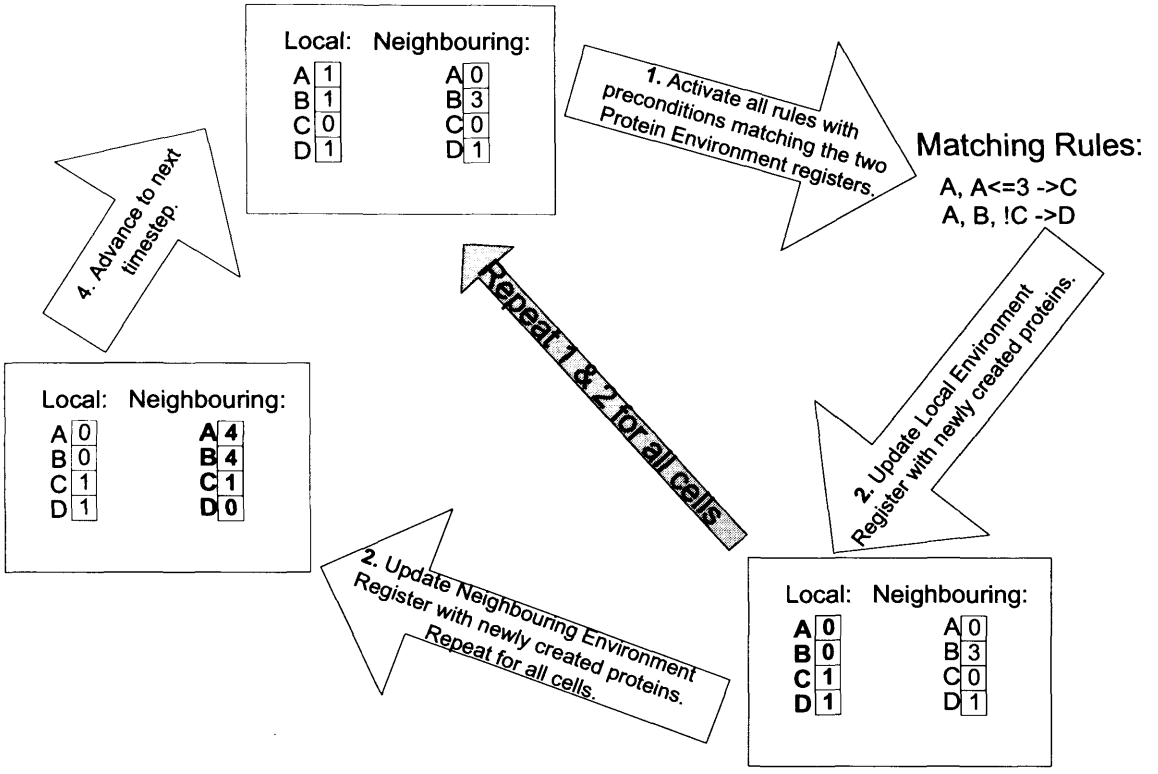


Figure 5.4.3: The protein generation process.

This process is repeated for a number of timesteps, allowing the pattern of proteins formed across the array of cells to change until a stable state or cycle of states is reached, or until development is halted after a predetermined number of timesteps.

For development to form useful patterns different subsets of rules must activate in different cells and/or at different times. For this to happen, it is necessary for a subset of cells to experience different contexts than others at the start of development. This is achieved by introducing a set of simple yet inhomogeneous initial conditions from which further inhomogeneity can develop. These are predefined, not evolved.

5.4.2 The Virtual FPGA

The virtual FPGA consists of a two dimensional array of virtual configurable logic blocks (CLBs). It is essentially a simplified model of the Virtex architecture and is shown in Figure 5.4.4. Each CLB contains three inputs. These drive two 3-input lookup tables (LUTs) labelled F and G. A LUT is configured by specifying the output for each possible input combination (or minterm), thus each LUT can be completely specified by 8 bits. Each input is also configurable and can be driven by any of the LUTs in the CLB's four Von Neumann neighbours. Hence each

input can assume one of eight sources. No other elements within the virtual FPGA are configurable.

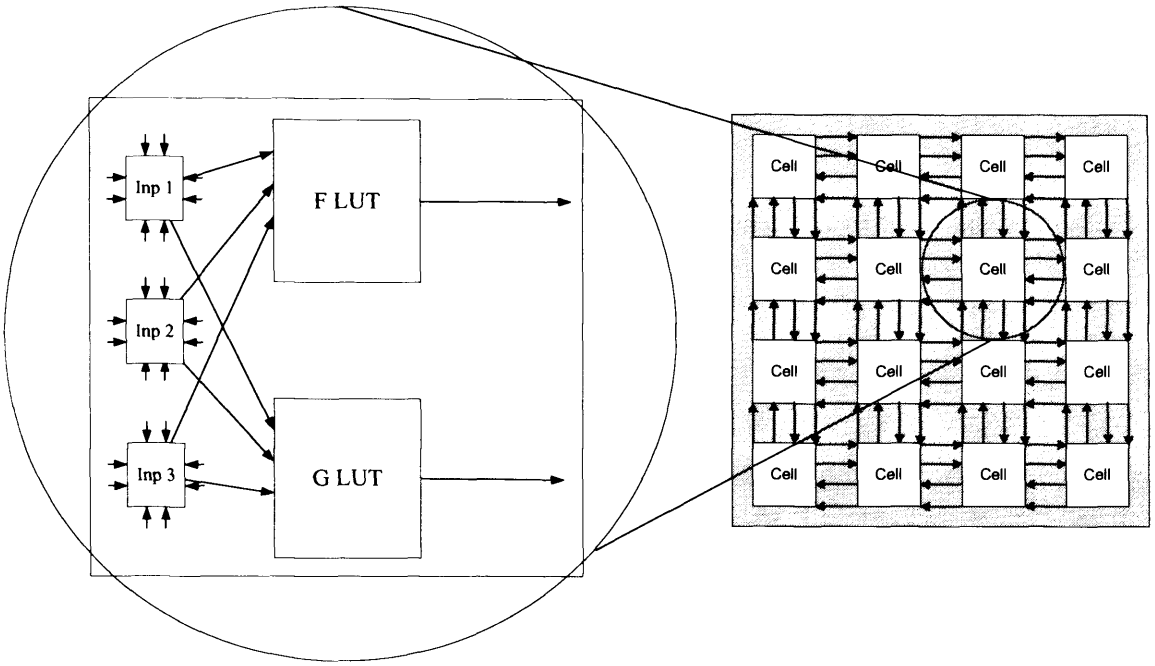


Figure 5.4.4: The Virtual Architecture. Only inputs and LUT logic are reconfigurable.

Each CLB in the virtual FPGA is associated with a unique cell in the protein layer and a unique cell in the architecture layer. The architecture layer is used to map activity in each cell of the protein layer to changes in the configuration of the CLB with which it is associated. This process is explained in Section 5.4.2.

Once the FPGA configuration has been determined through development it is mapped automatically to an area within the FPGA of the experimental platform introduced in Chapter 3 using the Xilinx JBits API (Guccione et al., 1999) and is ready for evaluation.

5.4.2 The Architecture Layer

The architecture layer is responsible for monitoring the state of the protein layer at each timestep and mapping this behaviour to a configuration for the virtual FPGA. Again it consists of a two dimensional array of cells where each cell is associated with a unique cell in the protein layer and a CLB in the virtual FPGA, and contains a set of *activation counters* that are used to store the data gathered as the protein layer is monitored throughout development. In addition the architecture layer contains a set of 30 rules called the *structural rule set.* These rules are responsible for gathering the data from the protein layer that is used to determine the circuit configuration.

At the end of development the data within each cell's activation counters are used to determine the configuration of its associated virtual CLB. The activation counters can be separated into two types: those that determine the configuration of the CLB's LUTs and those that determine the configuration of the CLB's inputs, as shown in Figure 5.4.5. Each cell contains one set of eight activation counters for each LUT. Each individual counter in this set corresponds to one minterm of the LUT, and is called a *LUT counter.* Each cell also contains one set of eight activation counters for each input. Each individual counter corresponds to one possible input source for the input, and is called an *input counter.* Hence each cell contains a total of 16 LUT counters and 24 input counters.

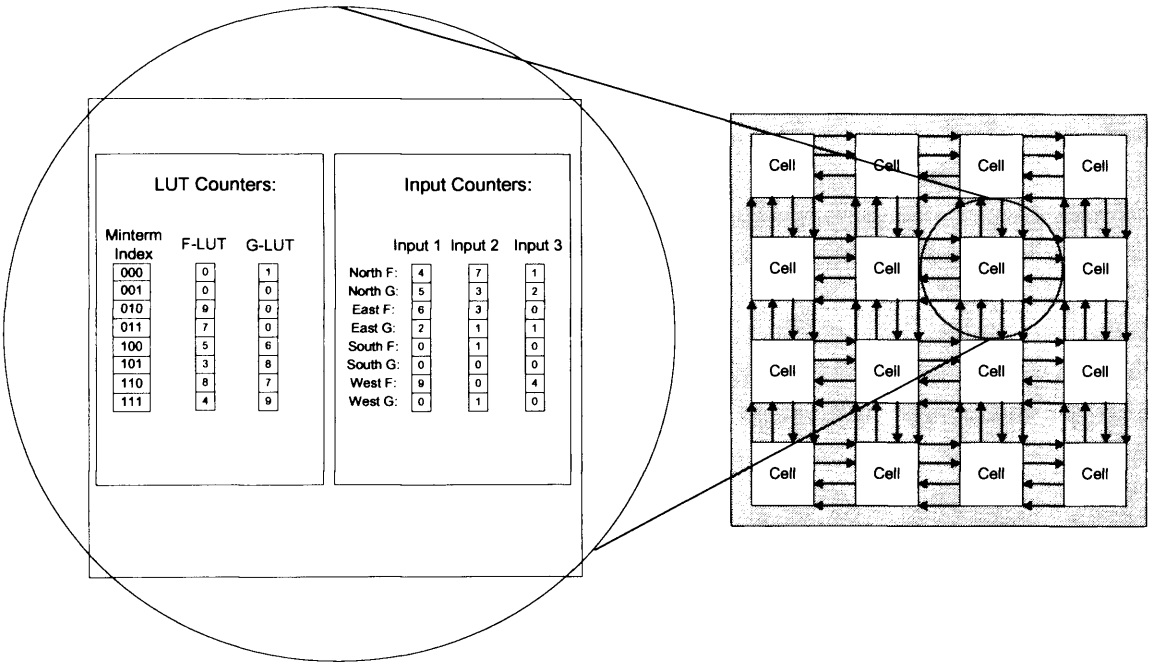


Figure 5.4.5: The contents of a cell in the architecture layer.

Structural Rule Syntax

The precondition of a structural rule is identical to the precondition of a regulatory rule and a structural rule is activated if the proteins present and/or absent in its corresponding protein layer cell match the rule's precondition. The postcondition of the rule consists of six bits that specifies a unique counter, hence a unique configurable element within the virtual CLB. This means that each structural rule is encoded in 34 bits.

The Architecture Mapping Process

The activation counters of each cell in the architecture layer are updated at each developmental timestep, prior to the timestep's protein generation cycle outlined in Section 5.4.1. The developmental process cycles through every cell in the array calculating the subset of structural rules to be activated. If one or more of the rules in this subset specify a particular activation

counter in their postcondition, this is that activation counter is incremented. An example cycle is shown in Figure 5.4.6(a).

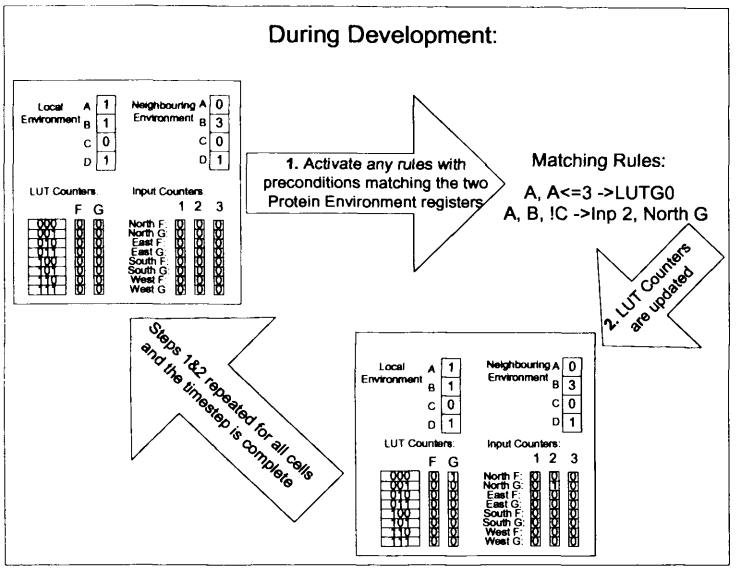


Figure 5.4.6(a): Counters are updated during an architecture mapping cycle.

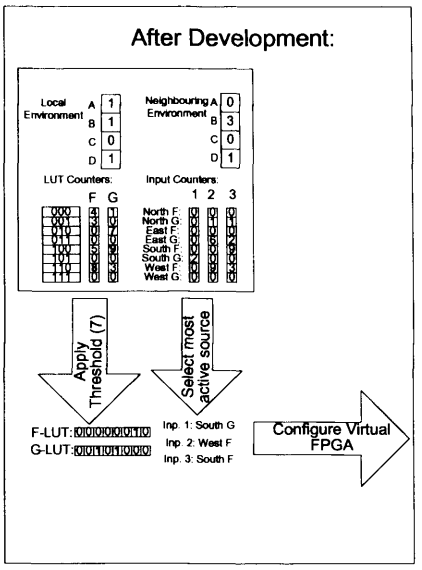


Figure 5.4.6(b): A CLB configuration is derived at the end of development.

At the end of development, the values held in the counters are used to configure the virtual FPGA. First development cycles through every cell in the architecture layer examining each LUT counter. If the counter value is greater than a predefined threshold value of 7 then the minterm configuration bit in the virtual architecture to which the counter corresponds is set high, otherwise it is set low. Development then cycles through every cell in the architecture layer, this time examining each *set* of input counters. (Recall a set of input counters contains a counter for each of the input's possible sources.) The counter with the highest value in the set is selected as the input source in the virtual FPGA. An example of how a virtual CLB configuration is derived from counters is shown in Figure 5.4.6(b). Thus the configurable elements of each CLB in the virtual FPGA are defined by the *accumulated activity* of structural rules throughout development.

In Section 5.4.1 it was stated that the regulatory rule set makes up the first 600 bits of the chromosome. The structural rule set, 30 rules each of 34 bits makes up the remaining 1020 bits of the chromosome. Hence the total chromosome length is 1620 bits.

5.5 Summary

This chapter introduced a number of changes to the developmental model used in Chapter 4 to improve evolution's ability to discover optimal solutions to the test adder problem. To allow

easier analysis of the effects of the changes development was decomposed into a pattern formation process and the process of mapping a pattern to a circuit design.

Analysis of the pattern formation process revealed that intercellular communication was highly limited. A number of changes were introduced that improved intercellular communication by increasing the number of local contexts that development was able to generate and maintain. In particular, rules that differentiated between the contribution made to a cell's context by itself and that made by neighbouring cells was found to be beneficial. The resulting pattern formation process was more expressive and was able to generate test patterns that were potentially useful for the development of adders. Intercellular communication was one of three factors identified in Chapter 4 as potentially performance-limiting, hence ripe for improvement.

Another of these factors was the abrupt nature of the Boolean activation and response strategies, which it was suggested were unlikely to be as evolvable as other the more graded alternatives used by some other researchers that were discussed in Chapter 2. However Boolean strategies are computationally inexpensive, and hence useful for modelling development in hardware evolution. A new genetic operator, rule duplication, was introduced to improve the ability of such rules to form robust regulatory networks, as this ability is likely to be important for evolvability. Although the introduction of this operator resulted in improved performance, analysis revealed that duplication was not improving performance as expected, by introducing redundancy in genetic networks. It was shown that the increased performance led from the introduction of a variable length chromosome needed to implement duplication, but the reason for this was not explored in detail, and as it remains unexplained duplication was not used in further experiments.

The mapping process was also analysed in this chapter. Initial analysis suggested that it may be useful to apply a bias towards logical functions that were useful in traditionally-designed circuits, but this did not prove to be useful to evolution. Further analysis revealed that the way the test problem had been represented meant that optimal solutions contained long signal paths. It was noted that evolving a set of rules that generates long signal paths can be difficult, as evolution cannot easily assign credit to partial solutions. Although it was demonstrated that this could be alleviated by explicitly introducing knowledge about the structure of a design, it was suggested that a better solution would be to reformulate the representation of the test problem. After reformulating the problem, it was demonstrated that patterns could be mapped to optimal circuit designs while avoiding the need to evolve long signal paths. This reformulation also introduced a finer-grained virtual architecture to take account of the third potentially performance-limiting factor identified in Chapter 4, that coarse granularity requires evolution to manipulate large numbers of structural proteins.

The final experiment of this chapter demonstrated that evolution could discover optimal solutions to the two bit adder with carry problem, and that it could do so when searching a design space richer than that considered by traditional design processes. Such spaces have greater potential to contain innovative designs than more restrictive spaces. The model of development used in this successful demonstration was then reviewed.

6 Scalability

The last chapter introduced a number of improvements to the initial model of development and culminated by demonstrating that evolution could discover an optimal two bit adder with carry circuit using an improved developmental model called the Outer Totalistic model. This chapter explores scalability, the central topic of this thesis, using the Outer Totalistic model.

Section 6.1 is concerned with the evolution of large patterns of proteins. The models of development used in this thesis generate circuits from patterns of proteins, and the study presented in this section demonstrates the evolution of high fitness solutions to pattern generation problems that might be useful for circuit generation. However it also serves another purpose. All the models of development used in this thesis are biased towards generating certain kinds of protein patterns, and this bias has a profound effect on the structure of the circuits that can be evolved. This is not a bad thing, as all inductive learners must exhibit some kind of bias if they are to perform meaningful learning (Mitchell, 1997). However characterisation of this bias might be useful for determining what kinds of hardware evolution problems are likely to be good candidates for a developmental approach, and the Outer Totalistic model in particular. Furthermore it might provide insight into how the Outer Totalistic model can be tuned to suit a more general set of hardware evolution problems, and reveal ideas that might be helpful in the study of similar models in the future. Hence Section 6.1 provides additional analysis in order to characterise developmental bias, leading to a set of guidelines that can help determine what kinds of problem the Outer Totalistic model is more suited to exploring. Section 6.1 concludes by evolving solutions to benchmark patterns that have been used by other researchers exploring development in hardware evolution, allowing the performance of the Outer Totalistic model to be compared to models they used. The majority of the results presented in Section 6.1 have been published in (Gordon and Bentley, 2005a).

Although it is interesting to explore scalability using patterns the primary hypothesis of this thesis is that the scalability of *hardware evolution* can be enhanced by exploiting a model of biological development. There are a few instances in the literature that demonstrate development can enhance scalability for pattern generation problems, such as (Bentley and Kumar, 1999; Roggen and Federici, 2004). In these cases and in almost all pattern generation problems fitness is based on the Cartesian distance of a candidate solution from the target pattern, hence there is a linear relationship between the distance of the candidate phenotype from the optimum and fitness. This is quite unlike for circuit design problems where phenotypes close to the optimum might exhibit low fitness, and small changes to a circuit

phenotype can perturb its observed behaviour greatly. (An example of an issue that arises from this phenomenon was discussed from the perspective of assigning credit to signals that require long routes in Section 5.3.1.) Hence it is not yet clear that the enhancement to scalability shown for some simple pattern generation problems will arise when development is applied to circuit design problems. This reasoning is supported by the results presented Chapter 5, where it was shown that patterns that can be mapped to circuit designs manually can be evolved when the circuit designs themselves cannot.

This means that there is currently very little evidence in the literature to suggest that development can improve scalability for traditional circuit design problems. One means of addressing this issue and clearly demonstrating the hypothesis of this thesis is to study scalability using true circuit design problems. Such a study is presented in Sections 6.2 and 6.3, using two common benchmark problems, the n-bit adder problem (Section 6.2) and the even n-bit parity problem (Section 6.3). In both cases comparisons are made with the same problems evolved using naïve 1:1 mapping. Results are presented that clearly demonstrate that the introduction of development enhances scalability, confirming the primary hypothesis of this thesis.

Several of the evolved circuits are also presented and analysed to demonstrate how development has improved scalability, thus reinforcing the empirical results. The largest of the evolved adders presented is a seven bit adder with carry. To date there is only one example in the literature of an evolved adder that is of comparable size (Shanthi et al., 2004). As discussed in Chapter 2 this example relied heavily on the use of additional techniques to decompose the problem into several independently evolved sub-problems, hence searched a much more constrained problem space. Furthermore in (Shanthi et al., 2004) connectivity was restricted to feedforward arrangements only, further constraining the problem space. The adders presented here are evolved with fewer design constraints, and allow evolution to explore circuits beyond the scope of the digital design abstraction, hence providing more opportunity for innovative designs to be discovered. The largest parity generator presented here is larger than any previously discovered using development and is of similar size to the larger evolved examples found in the literature (Koza, 1994; Rosca and Ballard, 1994; Yu and Miller, 2002), which again were again restricted to feedforward connectivity unlike the circuit design space used here. The majority of the work presented in Sections 6.2 and 6.3 has been presented in (Gordon and Bentley, 2005b).

6.1 Evolving Patterns on Large Arrays

Section 5.1 presented the evolution of various patterns using development across a 3x5 array of cells. This section presents similar experiments using a 20x20 array of cells to demonstrate that the evolution of patterns can scale to larger arrays. Analyses of the developmental model and the experimental results are also presented to provide insight into the kinds of patterns that the model is biased towards exploring.

The experiments involve the evolution of several sets of patterns. The first set of experiments, presented in Section 6.1.3, use target patterns that contain repeated motifs similar to those used in Chapter 5, but here the motifs are iterated over much larger cellular arrays. This is done to provide insight into how well evolution harnesses development's ability to reuse small modules that could map to simple units of circuitry, such as those that might be required to generate adders, multipliers or other similar highly regular structures. However traditionally designed circuits do not consist solely of small units that are repeated indefinitely. They also exhibit regularity of a higher order, for instance when signals must be routed over long distances between larger computational units, often over data buses. To determine whether the Outer Totalistic model might be able to exploit regularities of this scale, the evolution of patterns where the size of the repeated motif has been scaled up is presented in Section 6.1.4. Sections 6.1.5 and 6.1.6 explore the evolution of more complex patterns that have been used as benchmark patterns by other researchers exploring development in hardware evolution. This allows the performance of the Outer Totalistic model to be compared to models used by other researchers.

Before these experiments are presented, some general details of the experiments are given in Section 6.1.1 and some initial observations that are useful for characterising the bias of the developmental model are discussed in Section 6.1.2.

6.1.1 Experimental Setup

The model of development used here is the Outer Totalistic model used in Section 5.1.6: each rule defines two conditions for each protein, one is either a precedence or (in)equality condition describing how the sum amount of the protein generated by neighbouring cells affects the activation of the rule, and the other defines how the presence of a protein generated by the cell itself affects rule activation. Just as in Section 5.1.6, the developmental model does not contain any rules that map to circuit alterations, and a chromosome consists of 20 rules each of 30 bits, presenting a search space of 2^{600}. One change that was made to the model was to increase the number of timesteps for which the development process was iterated. In earlier chapters this was set to 30 timesteps. As the model only allows information to be transmitted one cell every

timestep, the minimum number of timesteps required for information to traverse the 20x20 arrays used here from one corner to the other would be 39 timesteps. Because of this the number of timesteps for the development process was increased to 50, to allow for some additional steps where information is not transmitted but computation occurs.

Ten runs of each experiment were carried out, either for 1000 generations or until an optimal solution was discovered. Fitness for the initial set of experiments was again based on the Hamming distance of the candidate pattern from the target pattern. The target pattern for the first set of experiments involved only protein A hence the longest distance between a candidate solution and the target pattern for the 20x20 array was 400. Fitness was set as the distance between the target pattern and the candidate solution for protein A subtracted from 400 yielding a maximum fitness of 400, and a minimum of zero.

6.1.2 Initial Considerations

Before the experiments are presented some features of the model that are likely to apply strong biases to the evolutionary search are considered.

Symmetry

With the Outer Totalistic model the information a cell receives from its neighbours (in the form of protein concentrations) is summed, hence a cell cannot distinguish information received from specific neighbouring cells. This means that the Outer Totalistic model is biased towards generating patterns with particular symmetries. For instance consider a hypothetical non-bounded Outer Totalistic array where the initial conditions are set so that a single cell contains one protein. As the developmental rules do not distinguish between the four symmetrically-arranged neighbouring cells, any pattern that develops from this source cell must exhibit the following symmetry elements that intersect at the source cell: a rotational four-fold point of symmetry and four lines of symmetry, one along the horizontal axis, one along the vertical axis and two bisecting these axes. An assemblage that exhibits such symmetry is said to have four-fold dihedral symmetry, or belong to the D_4 point group (Cotton, 1990).

Should good solutions to the problem under investigation exhibit D_4 symmetry this feature can be of great advantage, as it effectively reduces the problem to around an eighth of its original size, with the remainder of the solution arising automatically from the nature of the model. However there are likely to be many circuit design problems where good solutions exhibit more complex symmetry. For these problems the inherent symmetry of the model must be broken. There are two mechanisms by which development can do so. The first is if development is carried out on a bounded array, as in the experiments presented here. In this case the number of neighbours experienced by each cell is inhomogeneous across the array: the edges and corners of the array have fewer neighbours than cells that do not lie on the array boundary. Because of

this it is possible to construct rules that behave differently in these three types of cell, even under identical initial conditions. Hence the symmetry of the developing pattern could potentially be broken. Unfortunately in the 20x20 arrays used here the patterns that can arise directly from these inherent inhomegeneities are limited to those that form a D_4 point at the centre of the array, because each edge and each corner is identical, and they are arranged with D_4 symmetry around the central point. This symmetry could be eradicated by using an asymmetric array, but such an approach has not been used in these experiments.

The second mechanism by which symmetry can be broken is by interaction with a second pattern that has developed from another cell (or group of cells) elsewhere in the array. Of course such a pattern can only arise if an inhomogeneity is present in the array at the source of this pattern. Again this could only be brought about by either prudent selection of initial conditions at that position in the array or the development of a pattern from the inhomogeneity inherent in the edges and corners of a bounded array. Because of this constraint, the initial conditions for each experiment presented here were carefully chosen to ensure that the symmetry can be broken so that patterns with the target pattern symmetry can be generated. This issue is expanded upon later and specific initial conditions are derived for each experiment described in the following sections.

Information Transmission

It is now widely accepted that a common strategy used in biological development to specify different embryonic regions is the generation of some form of *positional information* which can then be interpreted by each cell to determine its fate (Wolpert et al., 2002). This information might be a global signal, for instance a concentration gradient that is formed along an axis of the developing embryo. This raises the question of how such a gradient might be formed. Some models based on ideas from dynamical systems theory can demonstrate the spontaneous formation of such gradients. They achieve this by amplifying small perturbations in the concentration of a nominally uniformly-distributed substance that arise from natural thermodynamics. (Turing, 1952; Gierer and Meinhardt, 1972; Slack, 1991). Such behaviour cannot be generated by the model of development used here. Instead, any positional information must arise from an initial inhomogeneity, just as with symmetry breaking process discussed above, and be transmitted across space. In the Outer Totalistic model each cell can only communicate directly with its Von Neumann neighbours, in order to keep computational complexity low. This means that positional information must be transmitted through local interactions with neighbours alone.

This is likely to have important consequences for scalability, for the following reason. In the Outer Totalistic model positional information must be represented as a combination of proteins: a protein context. The number of unique contexts is limited by the number of proteins present in

the system. The number of unique contexts is also constrained by interactions with neighbouring contexts: if a cell is to be labelled with a unique and stable protein context it must interact with the contexts of its neighbours in a benign manner. Each unique context must be governed by its relationship with neighbouring contexts, which is defined somewhere within the developmental rule set. In turn these neighbouring contexts must also be labelled by positional information that is defined by similar information in the rule set. The transmission of positional information over greater distances requires a greater amount of information within the rule set to specify the chain of relationships between neighbouring cells, from an initial point of inhomogeneity to the point where positional information is required. As evolution must discover these chains of rules, it is likely that any pattern requiring positional information to be transmitted over large distances will be difficult to evolve. In the following sections the kinds of patterns that do and do not require such long-distance transmission will be discussed.

The arguments laid out in earlier chapters suggest that development can enhance scalability by exploiting regularity during pattern formation. Regular patterns can be placed in one of two categories: those with translational symmetry, and those without. A pattern has translational symmetry if its motif can be consistently translated by some vector (perhaps in conjunction with additional symmetry operations) without altering the pattern. Examples of this class are cheques and stripes. These patterns represent a fairly trivial level of reuse that could easily be identified and exploited by a traditional designer. However they provide a simple means of exploring whether the model of development used here allows evolution to discover and reuse simple regularities that might be useful for evolving circuit elements such as arithmetic units and simple arrays of memory. Furthermore it was shown in Chapter 5 that similar patterns can be mapped to large circuits composed of repeated, relatively simple units of circuitry, such as adders. Examples of regular patterns with non-translational symmetry include the simplified French and Norwegian flag patterns that are investigated later in this chapter. Such patterns might be useful for the generation of more complex circuits, such as those requiring data buses or those containing several regular areas such as more complex memory caches.

Section 6.1.3 and 6.1.4 explore the development of patterns with translational symmetry. Section 6.1.3 focuses on patterns with small motifs, and Section 6.1.4 on larger ones. Section 6.1.5 explores the development of regular patterns with no translational symmetry.

6.1.3 Evolving Translational Patterns with Small Motifs

This section demonstrates the evolution of striped and chequed patterns similar to those evolved in Chapter 5, but scaled to the larger 20x20 array.

Experiment 7A: A Chequed Pattern

The target of the first experiment was to evolve a chequed pattern of proteins. This is presented in Table 6.1.1 as the target pattern for Experiment 7A. This target pattern is highly symmetrical: an infinite array of the pattern would contain a D_4 point at every cell. Hence the initial conditions were set simply as if a single cell at the south-west corner had generated protein A at timestep *t-1*. Both the initial conditions for the local environment and the initial conditions for the neighbouring environment were updated as shown in Table 6.1.1, where lowercase conditions refer to the neighbouring environment map and the uppercase condition refers to the local environment map. The results of the experiment are shown in Table 6.1.2 as Experiment 7A. Every run discovered an optimal solution. The mean number of generations to discover an optimal solution was 174.1.

One of the evolved rule sets is shown in Appendix D1, along with a series of diagrams depicting the states of the four proteins at each developmental timestep. This rule set was typical of the evolved solutions in that it develops the target pattern by generating a wavefront of information that moves diagonally across the array, one cell at each timestep, from the south-west corner where the initial conditions were set to the north-east corner. All the optimal solutions used this strategy to develop the final pattern, although the details of which proteins were involved in the process and how they were used to generate the final pattern varied. This is the same strategy that was envisaged for the evolution of smaller arrays of patterns in Chapter 5, and then observed in Experiment 2A of Chapter 5. The results of Experiment 2A in Chapter 5 and the experiments presented here cannot be compared directly as the Outer Totalistic model was not used in Experiment 2 and the target patterns used involved several proteins. However it is clear from the results of Experiment 7A that evolution is capable of regularly discovering developmental rule sets that can generate large arrays of simple chequed patterns from very simple initial conditions.

Experiment 7B: A Striped Pattern

The target of the second experiment was to evolve a striped pattern of proteins, which is shown in Table 6.1.1 as the target pattern for Experiment 7B. Striped patterns are less symmetrical than chequed patterns: an infinite array of striped cells does not exhibit D_4 symmetry around every cell. Each cell only contains two lines of reflection, along the horizontal and vertical axes and a two-fold point of rotation, or D_2 symmetry. Thus for this pattern to be attainable by the developmental model, the initial conditions must break the additional two lines of reflection and reduce the rotational symmetry from four-fold to two-fold. A set of initial conditions that achieve this are shown in Table 6.1.1, as the initial conditions for Experiment 7B. These conditions are as if three horizontally adjacent cells towards the centre of the array had

generated protein A and three cells vertically adjacent to this stripe of A had generated B at timestep *t-1*.

This pattern was considered harder for development to create than that of Experiment 7A for the following reason. Without the introduction of symmetry-breaking initial conditions, the developmental model used here is constrained to generating patterns with D_4 symmetry. This means that the maximum number of unique contexts between which it is possible for a set of rules to differentiate is limited to around one eighth of the array, as shown in Figure 6.1.1(a). The solution space explored by evolution is limited to patterns constructed from these contexts.

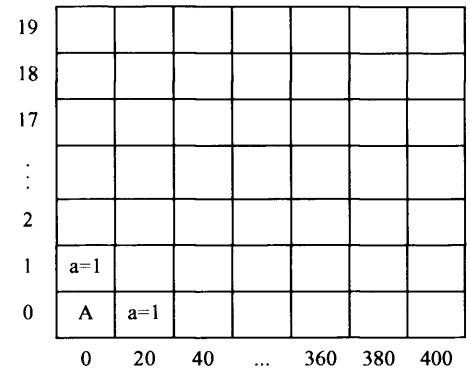
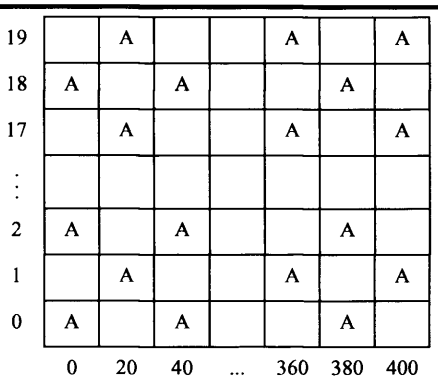
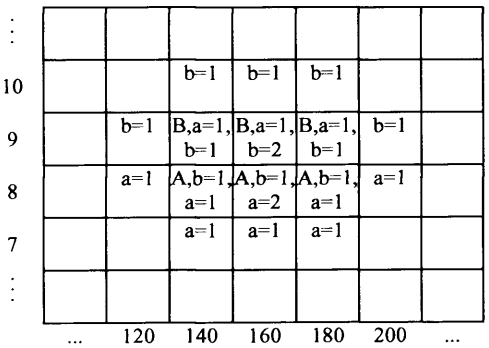
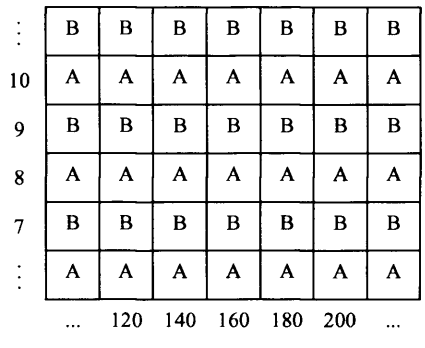
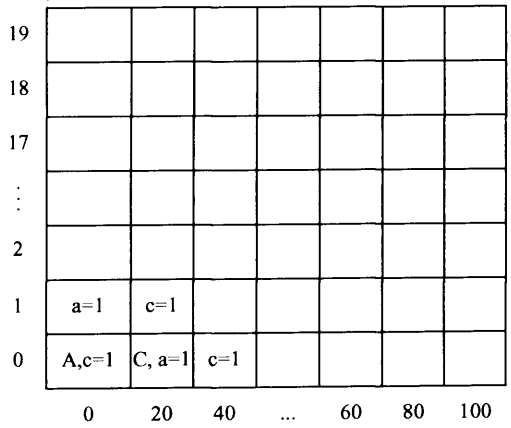
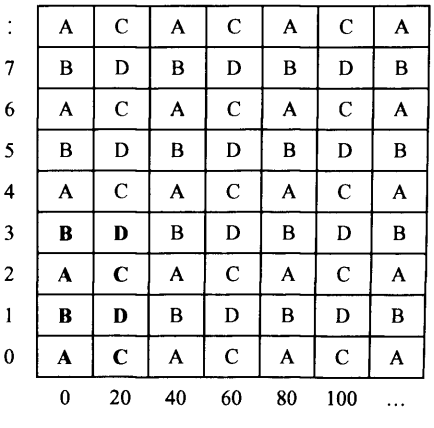
Expt.	Initial Conditions	Target Pattern
7A		
7B		
7C		

Table 6.1.1: Initial protein concentrations and target protein patterns for the evolution of chequed and striped patterns on a 20x20 array.

Experiment	Best / Max. Fitness	% Optimal Runs	Mean (Best Fitnesses)	Std. Dev. (Best Fitnesses)
7A	400/400	100	400.0	0.0
7B	400/400	40	362.5	43.92
7C	1529/1600	0	1398.4	49.82
3A	60/60	15	56.05	2.37

Table 6.1.2: Results from the evolution of patterns across large arrays.

Once symmetry is broken, the number of unique contexts exposed to evolution is increased (assuming a unique context can be assigned using the number of rules and proteins present in the system), as shown in Figure 6.1.1(b). However once this is done, evolution must discover rules to manipulate interactions between cells that was previously achieved as a consequence of the inherent symmetry of the model. For instance when using initial conditions that reduce D_4 symmetry to D_2 symmetry, as in Experiment 7B and as depicted in Figure 6.1.1, evolution must learn rules to control both horizontal and vertical growth rather than growth in one direction only. (A hand-designed solution for Experiment 7B required ten rules compared with two for Experiment 7A.) Not only must evolution now learn to control more interactions, it is also possible for the interactions governing growth in both directions to interact, leading to an increase in rule epistasis. Consequently the problem is likely to be more difficult for evolution to discover solutions.

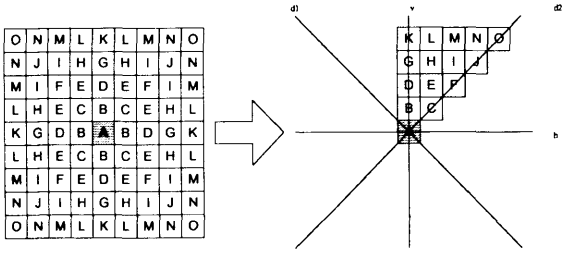


Figure 6.1.1(a): The maximum number of unique contexts available with simple initial conditions. This arises from the inherent D4 symmetry of the developmental model.

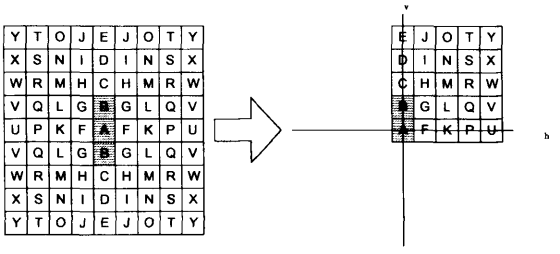


Figure 6.1.1(b): An increased number of unique contexts following the introduction of symmetry-breaking initial conditions.

The results of Experiment 7B support this reasoning: although evolution managed to discover optimal solutions, only 40% of the ten runs discovered an optimal solution. Nevertheless evolution did manage to discover optimal solutions and the most efficient solution used only four rules, many less than the hand-designed alternative. All of the optimal solutions discovered by evolution alternated between two different striped patterns at each timestep, with the pattern generated on the fiftieth timestep matching the target pattern. They all used a similar basic strategy, depicted in Figure 6.1.2.

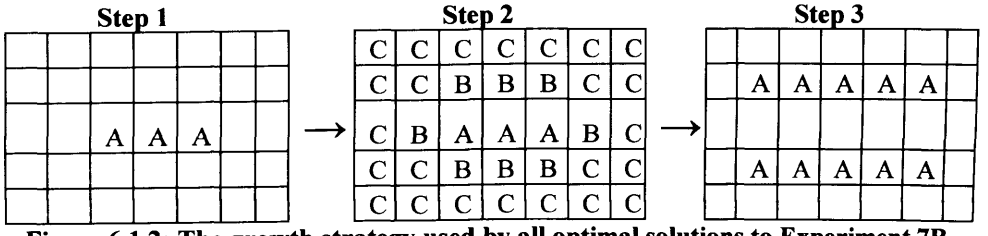


Figure 6.1.2: The growth strategy used by all optimal solutions to Experiment 7B.

At the first timestep a ring of another protein was created around the cells set with protein A as initial conditions (in the example of Figure 6.1.2 this is labelled B), and this was used as a mask to prevent the generation of a further protein that was otherwise homogenous across the array (labelled C in this example). At a further timestep cells generating B with fewer than three neighbouring cells generating C were set to A. A consequence of the geometry of the array is that cells meeting this criterion will be present both directly and diagonally above the current row of A, thus a new, larger row is created both above and below the original row. One of the evolved rule sets and diagrams showing the protein states during development are shown in Appendix D2.

Experiment 7C: A Pattern Derived From an Adder

While the chequed and striped patterns above provide some insight into how symmetry and motif size affects the performance of the Outer Totalistic model it does not provide direct evidence that the model is capable of evolving large patterns that are likely to be useful for hardware evolution. In Experiment 6 of Chapter 5 a hand-designed pattern was created and it was shown that evolution could successfully evolve a mapping between this pattern and a two bit adder. This pattern, extended to fill the 20x20 array by iterating it as a motif, was used as the target pattern for Experiment 7C. This target pattern is shown in Table 6.1.1 (with one example of the repeated motif in bold). As with similar experiments in Chapter 5 the initial conditions, also shown in Table 6.1.1, were set as if a single cell had generated A and a single cell had generated C at the timestep *t-1*. The fitness measure was again was based on the Hamming distance between the candidate pattern of proteins and the target pattern of proteins. On the 20x20 array the longest distance possible between a candidate and the target pattern using all four proteins was 1600. Fitness was the distance between the target and candidate solution subtracted from 1600, yielding a maximum fitness of 1600, and a minimum of zero. The results of the experiment are shown in Table 6.1.2. No optimal solutions were found. However the best solution came very close with a fitness of 1529. Examination of this solution revealed that the pattern was formed perfectly over an 18x18 subset of the array, with only the two rows and columns at the north and east edges failing to match the target. This solution is shown in Appendix D3. Unlike solutions to the earlier problems of Experiment 7 this solution made use of two wavefronts, one travelling east and one travelling north. It was noticed that after the completion of 50 developmental timesteps the wavefronts had not yet traversed the array. When the solution was run for an additional ten timesteps and its fitness rose to 1560, and only the extreme north row and east column failed to match the target. Had solutions been evolved using more timesteps, evolution may have had an opportunity to find rules to correct the error generated at these edges and yielded a perfect solution. This demonstrates that the developmental model performs well when evolving a large target pattern akin to what might actually be useful for hardware evolution.

The experimental results presented so far suggest that in most cases the evolution of patterns with small motifs scales to large arrays reasonably well hence evolution is exploiting a strategy that does not rely on global positional information. In both the experiments presented here and the experiments of Chapter 5, evolution tends to use wavefronts of activity that move across the cellular array, generating a stable pattern behind them. Rather than transmitting unique positional information over large distances, information is transmitted that defines position *relative* to a nearby newly-generated motif. As the motifs used in these experiments are small, only a handful of unique contexts need be defined by development and only a handful of rules are needed to govern them. Hence it seems reasonable to tentatively suggest that using this model of development, patterns with small motifs are generally scalable to large arrays, assuming that the symmetry requirements for the formation of the pattern are met by the initial conditions selected.

6.1.4 Evolving Patterns with Large Motifs

In Chapter 5 it was shown that using structural rules of a simple format it was possible to map from a pattern with a small four-cell motif to an optimal and general adder design. Section 6.2 will show that development can exploit the regularity of adder designs by generating patterns with similar small motifs. However circuits typical of real-world design problems are likely to be composed of larger, more complex structures, with regularity evident on a greater scale. If evolution is to exploit such regularity while still operating at a low level of abstraction, it is important for the developmental model to be able to generate regular patterns with larger motifs.

Evolution used relative positional information to generate the patterns in Section 6.1.3 to solve the difficulty associated with transmitting information large distances to unique sites using a nearest-neighbour model. This suggests that the generation of patterns based on larger motifs is likely to be more difficult as relative positional information must be transmitted larger distances hence more unique contexts that interact benignly with their neighbours must be defined by the rules.

Experiment 8A: The Effect of Distance on Information Transmission

A series of experiments were carried out to test whether evolution's performance did decrease as the distance over which positional information had to be transmitted was increased. The generation of an arbitrary motif is likely to be influenced by the relationship between the motif, the geometry of the cells and the geometry of the mechanism of information transmission employed by the developmental model. The initial pattern evolved in this set of experiments was selected to alleviate these effects by simplifying the motif to be generated as much as possible while requiring information to be transmitted over large distances. The pattern of the first experiment was a uniform pattern with D_4 symmetry based on a motif of a single cell with

protein A present. A series of patterns were used where this single cell was separated by an increasing number of cells where protein A was set low. These patterns are shown in Table 6.1.3 except the pattern with one gap, which is identical to the chequed pattern of Experiment 7A. Ten runs of each experiment were carried out using identical parameters, identical initial conditions and an identical fitness function to Experiment 7. The results are shown in Table 6.1.4, including the results for Experiment 7A, as they represent the same pattern with a gap of one cell.

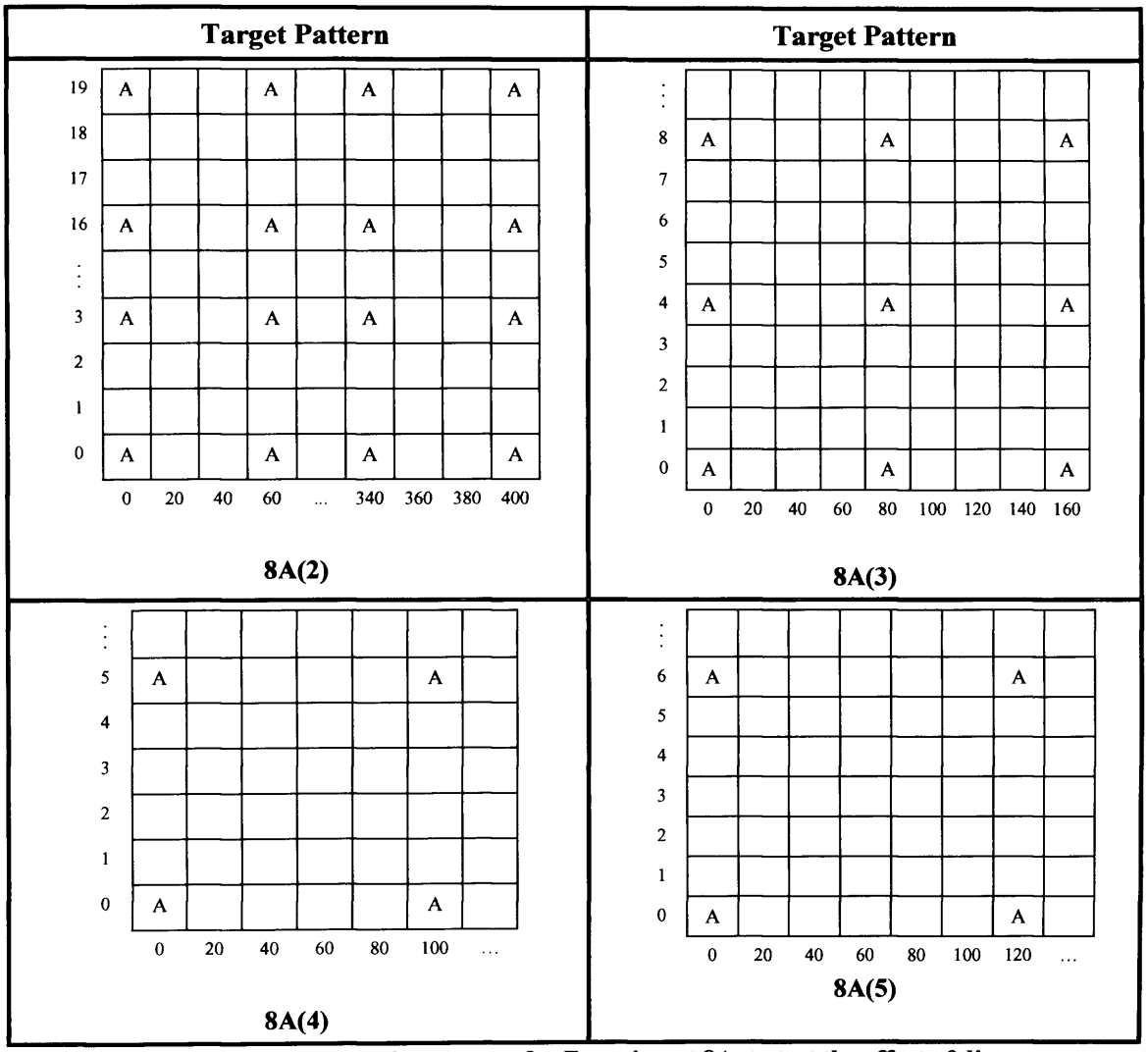


Table 6.1.3: The target protein patterns for Experiment 8A, to test the effect of distance on information transmission.

The results show that excepting Experiment 7A where optimal solutions were always discovered, there appears to be no trend towards decreasing fitness when moving to patterns with a greater distance between cells with A set in the target pattern. At first sight this suggests that the distance between motifs does not play a role in the difficulty of evolving a particular pattern. However examination of the evolved patterns reveals a different story. As the size of the array is constant at 20x20 cells, the number of cells that must be set with protein A present for each experiment drops as the size of the gap between the pattern increases. This means that

the fitness of a poor solution which does not generate protein A in any cell achieves a higher fitness as the gap between the cells that set A in the target pattern increases. Both Experiments 8A(2) and 8A(3) discover solutions that partially solve their respective problems, and the best solutions for these are shown in Appendix D4 and D5 respectively. Experiments 8A(4) and 8A(5) discovered trivial solutions where protein A was only set in the single cell where it had been provided as the initial condition. Hence it appears that the results from these experiments do not discount, and perhaps support the idea that there might be a general trend in the difficulty of discovering a strategy for transmitting information over large distances.

Experiment	Best Fitness	% Optimal Runs	Mean (Best Fitnesses)	Std. Dev. (Best Fitnesses)
7A	400	100	400.0	0.0
8A(2)	374	0	358.1	8.29
8A(3)	394	0	383.80	8.43
8A(4)	385	0	385.0	0
8A(5)	385	0	385.0	0

Table 6.1.4: Results of Experiment 8A, testing the effect of distance on information transmission.

Experiment 8B: Evolving Larger Chequed Patterns

One effect that is likely to contribute to the issue identified above is that as the size of this problem scales the ratio of cells that are tested for the presence of protein A to the cells that are tested for the absence of protein A by the fitness function drops. This means that as the problem scales the fitness function places more emphasis on cells that are not required to express protein A than those that are. To explore whether this had some effect on performance, experiments similar to Experiment 8A were carried out, but using motifs that required a consistent ratio of cells requiring the presence of A to cells requiring the absence of A. The pattern used in Experiment 8B was the chequed pattern of Experiment 7, but with the motif scaled to greater sizes, equally in both dimensions from a 1x1 motif to a 7x7 motif. The initial conditions were also changed. For experiments with each size of motif the initial conditions were set so that the array contained a single copy of that motif in the south-west corner. Again, the pattern with the smallest motif of this set is a chequed pattern of single cells, and so the results from Experiment 7 were reused as the 1x1 motif results for this experiment. A plot of the mean of the best fitnesses for 20 runs of each motif size is shown in Figure 6.1.3.

Figure 6.1.3 suggests that there is initially a trend towards lower fitness for larger motif size, but for the larger 6x6 and 7x7 cheques, fitness appears to increase. Additionally the 4x4 cheque has higher fitness than the 3x3 cheque and similar fitness to the 2x2 cheque. Although this might seem surprising it is easily explained. It was pointed out earlier that the corner cells can be used to generate inhomogeneity, but any pattern generated purely from these points will exhibit D_4 symmetry around the centre of the array. The target pattern for the 4x4 cheque exhibits such symmetry, and the target patterns for the 6x6 and 7x7 cheque are close to, but not quite

symmetric about this point and are close enough that many patterns that are symmetric about the centre of the array produce a high fitness score for these problems when using a 20x20 array. Examination of the evolved solutions for these problems revealed that evolution almost always made use of at least some information derived from the corner cells, although in many cases the pattern was perturbed by additional pattern formation around the motif provided as an initial condition. An example of such behaviour is provided in Appendix D6, where the best solution to the 7x7 cheque problem is presented. When development of this solution was rerun with the corner motif that was provided as an initial condition removed, the pattern formed was extremely similar to that formed when the initial conditions provided during evolution were present, and the solution only dropped in fitness by 1.5% from 368 to 362. This suggests that the majority of the information used by evolution to generate the pattern originated at the corner cells.

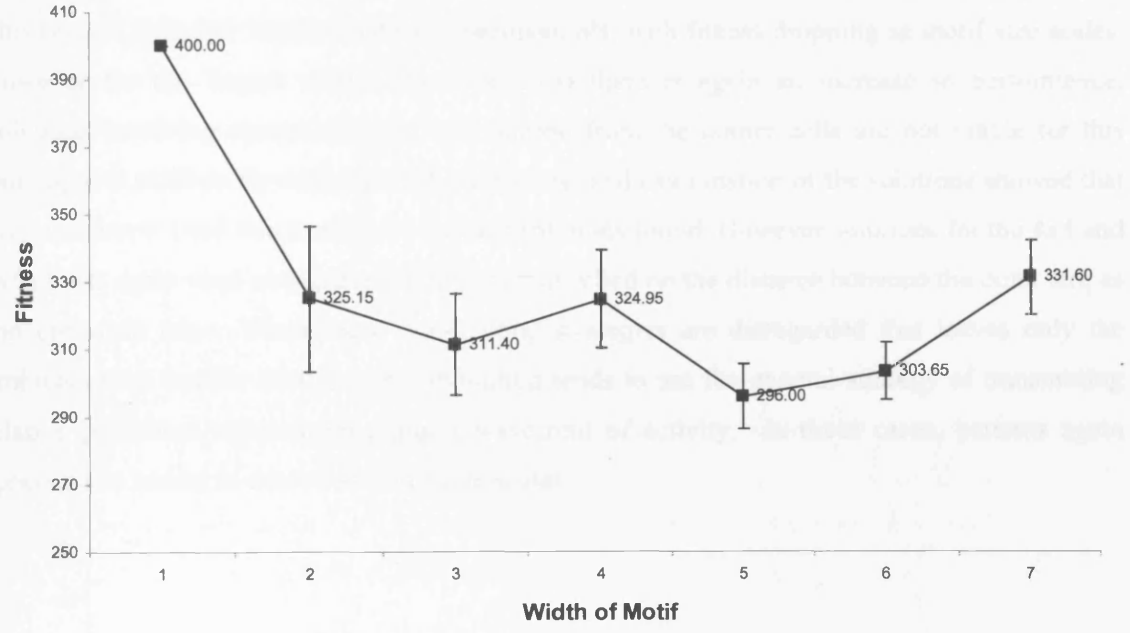


Figure 6.1.3: A plot of mean fitness against motif width for 20 runs of the evolution of a pattern with a chequed motif.

Patterns generated in this way can be considered as forming a single 10x10 motif that stretches from a corner to the centre of the array, which is replicated by the inherent symmetry of the developmental model. No transmission of relative positional information to allow the formation of multiple motifs is necessary and the problem is effectively simplified by the model's symmetry. The results suggest that evolution finds that this alternative strategy provides an easier route to moderately fit solutions than using the initial conditions alone. The same may be true for similar strategies that combine the use of both corner inhomogeneity and explicitly-provided initial conditions. However these strategies would not generate high fitness solutions for the same problem on larger arrays as the symmetry of the target pattern with respect to the array edges would change. Similar strategies are also unlikely to be as successful on larger

arrays for other problems that happen to exhibit D_4 symmetry around the centre of the array, as the complexity of the pattern fragment occupying a quarter of the array would increase with the array size. This would likely require an increase in the number of rules needed to specify the entire pattern correctly.

Setting aside non-general strategies that arise from unusual matches between the symmetry of the target pattern and the symmetry of the particular array used, the general trend appears to be again towards lower performance as motif size is scaled.

Experiment 8C: Evolving Larger Striped Patterns

Experiment 8C repeated Experiment 8B using the striped pattern of Experiment 7B as the target. A plot of the mean of the mean of the best fitnesses of each run is shown in Figure 6.1.4. This reveals a similar initial trend to Experiment 8B, with fitness dropping as motif size scales. However for the largest motifs (4x4 and 5x5) there is again an increase in performance. Solutions involving transmission of information from the corner cells are not viable for this pattern, as it exhibits D_2 rather than D_4 symmetry, and examination of the solutions showed that evolution never used this strategy in the best solutions found. However solutions for the 4x4 and 5x5 motifs again used non-general solutions that relied on the distance between the outer stripes and the array edge. When these non-general strategies are disregarded this leaves only the problems with smaller motifs where evolution tends to use the general strategy of transmitting relative positional information along a wavefront of activity. In these cases, patterns again appear to be harder to evolve as motif size scales.

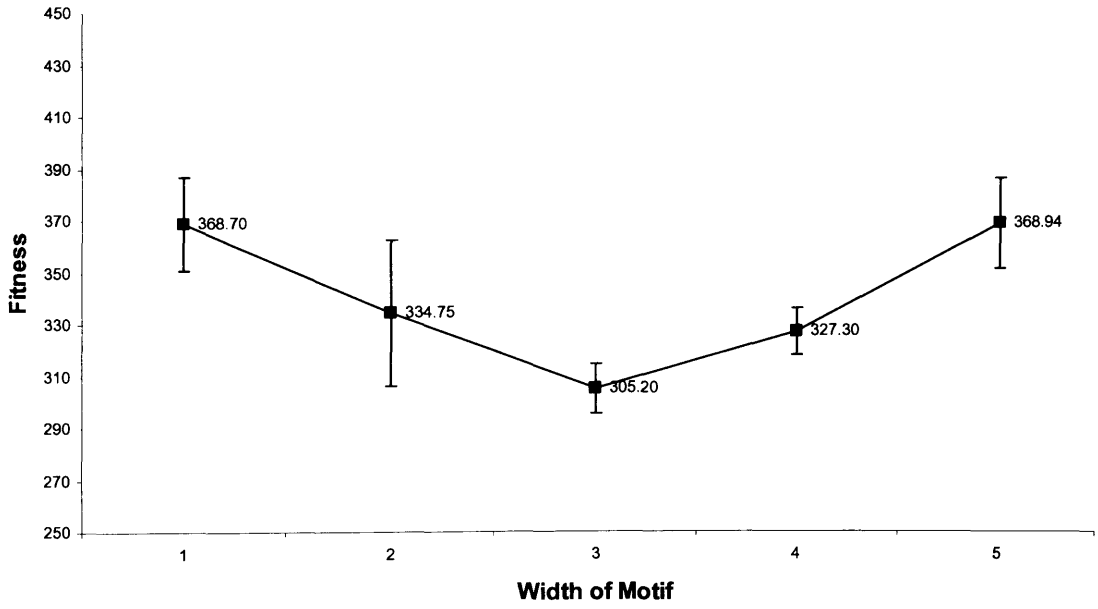


Figure 6.1.4: A plot of mean fitness against motif width for 20 runs of the evolution of a pattern with a striped motif.

6.1.5 Evolving Non-Uniform Patterns

The repeating patterns discussed above can all be formed by translating a motif uniformly across the array by some vector. However there are many other patterns that exhibit regularity, but not uniformly across the entire array. This section first discusses which types of non-uniform pattern can be exploited by development, and then demonstrates the evolution of non-uniform, yet scalable patterns on large arrays. The patterns that have been used in this section have also been evolved by other researchers interested in scalability, allowing a performance comparison between the Outer Totalistic model and other models of development used in the literature.

Regular non-uniform patterns can be separated into two classes. The first are patterns that possess point symmetry elements, and also exhibit translational symmetry within the primitive asymmetric area of the pattern. For patterns of this class every motif present in the primitive area can be mapped onto another using a simple translational operation, and the point symmetry of the pattern can then be used to reproduce the pattern in its entirety. Examples in this class are bow-tie shaped patterns, which could be useful in hardware evolution as they might allow the generation of matched pairs of multiplexers and demultiplexers for communication, and patterns based on crosses that might be useful for generating buses or other long-distance routing. Just as was discussed in the section above, development could provide a scalable solution to problems of this class by learning a set of rules that transmit information through the translation vector and another set that generate a new motif. The example patterns discussed above exhibit symmetry around a centre point, and hence present an opportunity for evolution to take advantage of the inherent symmetry in the developmental process. Less symmetrical structures of this class exist, for example triangles, which might be useful for the generation of multiplexer trees, and could be generated in the presence of symmetry-breaking initial conditions.

The second class of regular non-uniform patterns are those that do not exhibit strict translational symmetry, but can be decomposed into elements that do. Examples of such patterns are the simplified Norwegian and French flags shown in Figure 6.1.5(a) and 6.1.5(b). Patterns of this class are more likely to be useful for the generation of complex hardware that contain several of the circuit elements that have been discussed in conjunction with simpler patterns. One way development could solve such problems would be to learn a set of rules for each element of the pattern, with each set encoding both the translation vector and a motif-generating process for that particular element. For instance the Norwegian flag pattern in Figure 6.1.5(a) can be decomposed into intersecting blue stripes with surrounding white edges, a black edge surrounding the entire array, and all other cells red. One way to solve the problem would be to

use a set of rules to generate each stripe, a set to generate the white edges, a set to generate the black edge and a final set to set all remaining cells red.

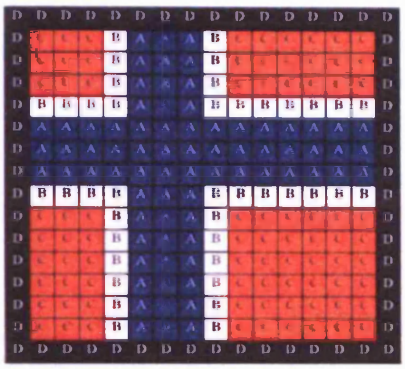


Figure 6.1.5(a): A pattern based on the Norwegian flag.

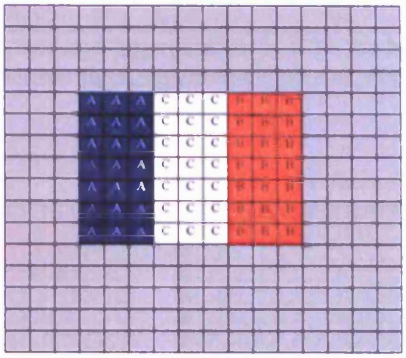


Figure 6.1.5(b): A pattern based on the French flag.

6.1.6 Comparison with Roggen and Federici

In (Roggen and Federici, 2004) Roggen and Federici evolved solutions to the Norwegian Flag problem. They compared two developmental systems, one that models diffusion but does not model any interaction between the proteins that diffuse across the array (called the Morphogenetic model), and the other based on Miller's Cartesian Genetic Programming (CGP) model (Miller and Thomson, 2000) that models both diffusion and protein interaction (called the Embryonic model). The major difference between Miller's and Roggen and Federici's implementation was that Miller's logical mapping between cell inputs and outputs was replaced with ANNs. Roggen and Federici carried out their comparison across a range of array sizes. An experiment was conducted to compare the performance of the Outer Totalistic model with those used by Roggen and Federici for array sizes of 16x16, 32x32 and 64x64. The 16x16 pattern is shown in Figure 6.1.5(a). The 32x32 pattern centres the cross on the 15x18th cell with the blue stripes five cells wide, the white stripes two cells wide and a surrounding border one cell wide. The 64x64 pattern is centred on the 29x35th cell with the blue stripes eleven cells wide, the white stripes five cells wide and a surrounding border two cells wide. All evolutionary and developmental parameters were as described for Experiment 8 except the fitness measure, the initial conditions and the number of developmental timesteps. The initial conditions for each pattern size were set as if the cell at the centre of the cross had generated protein A at timestep *t-1*. Four proteins were required to form the pattern and hence the fitness of each candidate solution ranged from a maximum of 4 x *n* to zero, where *n* is the total number of cells in the array. Ten runs of each pattern size were conducted, and the results are shown in Table 6.1.5, along with Roggen and Federici's results for comparison. (Note that the values tabulated for Roggen and Federici's results are approximate, as they were obtained visually from the plot presented in (Roggen and Federici, 2004)). The patterns of proteins for the best solutions evolved at each array size are shown in Figures 6.1.6, 6.1.7 and 6.1.8.

	Best			Mean			
Problem Size	Outer Totalistic	Morphogenetic	Embryonic	Outer Totalistic	Outer Tot. Std. Dev.	Morphogenetic	Embryonic
16x16	91.41	~76	~67	88.45	3.93	~70	~58
32x32	90.04	~73	~70	88.81	3.41	~70	~58
64x64	81.64	~72	~80	74.45	3.84	~68	~68

Table 6.1.5: Comparison between the current system and approximate results for both the Morphogenetic and Embryonic systems presented in (Roggen and Federici, 2004).

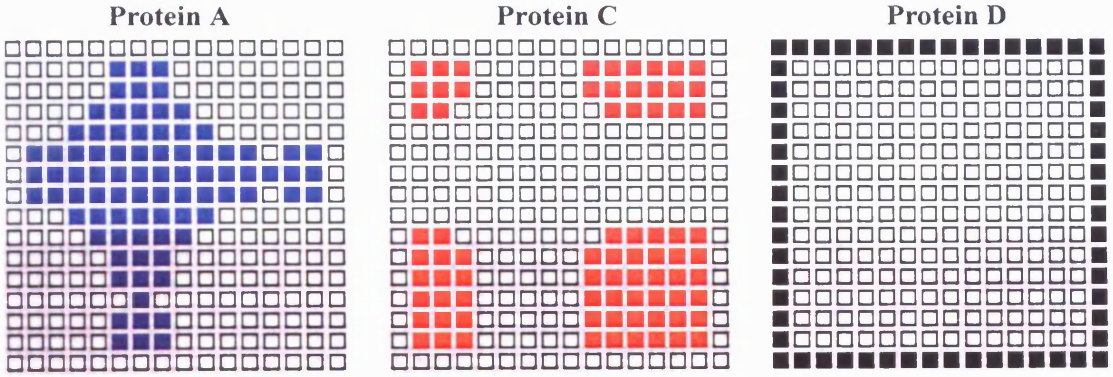


Figure 6.1.6: Protein map for the best evolved 16x16 Norwegian flag. In this solution protein B was not present at the end of development.

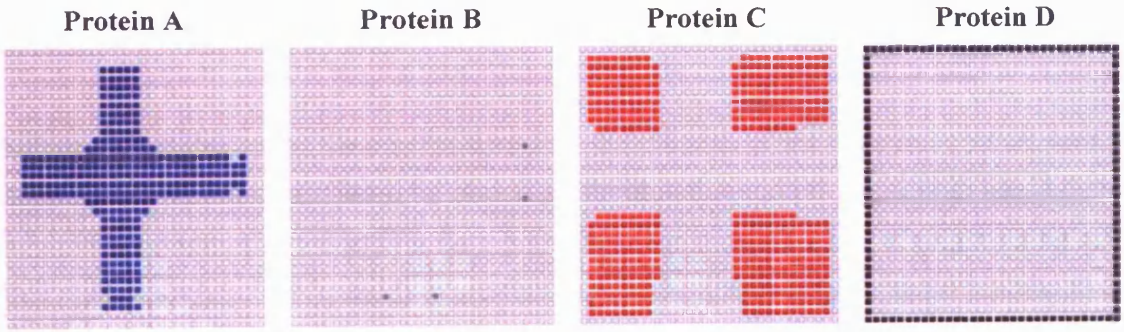


Figure 6.1.7: Protein map for the best evolved 32x32 Norwegian flag.

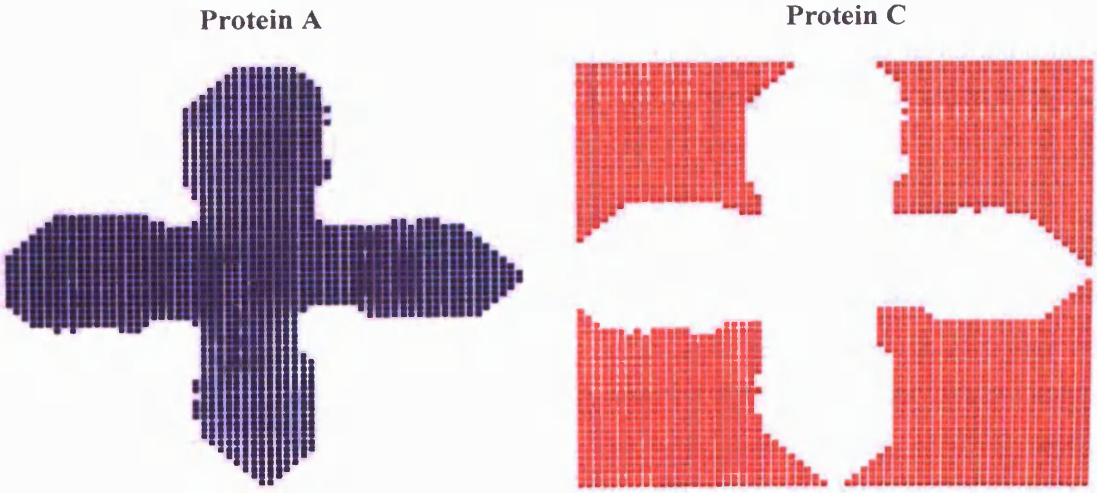


Figure 6.1.8: Protein map for the best evolved 64x64 Norwegian flag. In this solution proteins B and D were not present at the end of development.

The results in Table 6.1.5 suggest that the system presented here outperforms both the other systems across all three array sizes. Furthermore this was achieved in fewer evaluations than Roggen and Federici (100,000 rather than 800,000).

It is worth considering what features of the Outer Totalistic model may be leading to this improvement in performance. First the Morphogenetic model does not model any interaction between the proteins; protein generation is fixed at the start of development. This seems to constrain the patterns that can be generated to patterns with little fine detail: the patterns tend to consist of large areas where cells assume the same identity and boundaries between different. It is perhaps unsurprising that this model generates relatively simple patterns as the protein interaction of the other systems can be thought of as an additional layer of computation available to evolution.

Second, in both models used by Roggen and Federici each cell can only assume a single protein state. The Outer Totalistic model allows each cell to generate any combination of proteins, hence cells that contain the target protein along with other proteins can contribute towards fitness even though the state of the cell is not perfect. This may provide a smoother fitness landscape for this kind of problem.

A third point is that unlike the other two models, the cell state of the Embryonic model used by Roggen and Federici is calculated by evolving a function that accepts inputs from neighbouring cells independently. The model used here is *totalistic*, summing the states of neighbouring cells. This results in a hard bias towards generating symmetrical patterns (such as the solution to the Norwegian Flag problem shown in Figure 6.1.5) which the Embryonic model does not have. As stated earlier this can be of great benefit as it effectively reduces the problem size, and can be likened to processes in biological development which take advantage of the physics and chemistry inherent in the environment. In fact it was noted by Dawkins (Dawkins, 1989) that many biological organisms are highly symmetrical and that such symmetries are likely to arise as a natural consequence of their development, and might aid evolvability.

A fourth point relates to the use of carefully chosen initial conditions. The initial conditions that were required to break symmetry for the Outer Totalistic were positioned where the blue cross should be centred, using prior knowledge of the target pattern. Similarly the Morphogenetic model used initial conditions that were carefully positioned at the boundaries between areas of the pattern that should be set to different colours. The initial conditions used by the Embryonic model for this problem were not reported by Roggen and Federici, but should they have been set at random or at some position that did not use knowledge of the target pattern, the problem would have been harder for the model to solve.

Finally and perhaps most importantly the regulatory and communication strategies used differ in many ways, as does the syntax of the rules used to represent the model. In Chapter 2 it was stressed that detailed comparisons between such strategies would be useful. The Outer Totalistic model uses a Boolean regulation strategy whereas Roggen and Federici's Embryonic model uses a nonlinear graded strategy and their Morphogenetic model does not allow any interaction between proteins at all. The results of the experiments in Table 6.1.5 and Table 6.1.6 might at first glance suggest that a Boolean regulation strategy has no performance disadvantage, but as the models exhibit many additional differences it is not possible to make a meaningful comparison of the regulation and communication strategies used. However earlier in this chapter it was shown that if relative positional information can be exploited, as in this problem, a nearest neighbour strategy can perform reasonably well.

6.1.7 Comparison with Miller

Miller has also explored the generation of flag patterns including the French flag pattern shown in Figure 6.1.5(b) using his Cartesian Genetic Programming-based developmental system (Miller, 2004). In this study Miller was not interested only in scalability but pattern maintenance and regeneration. Hence the problem he tackled was not purely to generate the pattern at the final developmental timestep but to maintain it for a number of timesteps. This experiment was repeated using the Outer Totalistic model. The problem requires that a 9x7 French flag pattern be developed and maintained on a 16x16 array over ten developmental timesteps. Fitness was measured by summing the distance between the target and developed patterns as before, but the fitness was summed over the final four developmental timesteps, to produce a bias towards pattern maintenance. The developmental and genetic parameters for this experiment were identical to the previous experiments, except the number of developmental timesteps was reduced to ten in line with Miller's experiment, fitness was calculated across all four proteins again in line with Miller, and the initial conditions were set to allow the D4 symmetry of the system to be broken down to D2 symmetry. This was achieved by setting initial protein concentrations for a horizontal stripe of three cells at the centre of the area that should develop into the flag pattern. Each of the three cells were set as if they had generated one of proteins A, C or B at timestep *t-1*, mirroring the relationship between the three areas of protein in the target pattern shown in Figure 6.1.5(b). The results for 25 runs of the experiment are shown in the first row of Table 6.1.6. Results from (Miller, 2004) are provided for experiments conducted using zero and two proteins, representing the worst and best results achieved respectively.

The best performance for this problem was found by Miller's two protein model. However the mean of the best fitnesses ($\overline{f}_{best}$) achieved with the Outer Totalistic model is greater than both of Miller's models. Additionally the best solution found by the Outer Totalistic model had

greater fitness than the best found with Miller's zero protein system, and was close to the best performance of Miller's two protein model. Miller's results were obtained using 150,000 evaluations rather than 100,000 used here, suggesting that additional computational power is needed to achieve good results using Miller's models. Additionally the worst solutions found by Miller's models had significantly lower fitness than the worst result of the Outer Totalistic model, and a higher standard deviation. This suggests that the fitness landscapes generated by Miller's models are in some way more deceptive or present more local optima than the landscapes of the Outer Totalistic model, at least for this problem. From these results it seems reasonable to conclude that Miller's two protein model and the Outer Totalistic model provide comparable results.

Model	Best as % Max	$\overline{f}_{best}$ as % Max	Worst as % Max	Normalised Std. Dev.
Outer Tot. D2 Initial Conds.	97.17	95.84	94.92	0.46
Miller , 0 Prots	88.77	85.53	83.50	1.60
Miller , 2 Prots	98.83	91.67	87.30	3.03
Outer Tot. Univ. Init. Conds.	96.60	95.92	95.60	0.25

Table 6.1.6: Comparison between the Outer Totalistic model and Miller's CGP system used in (Miller, 2004) with zero and two proteins.

Possible explanations as to why Miller's models require more evaluations to achieve comparable results are similar to those given in Section 6.1.6. Just as with the Embryonic model, Miller's models are not biased towards symmetrical patterns, and so is likely to spend more time searching non-fruitful areas of solution space. Also as with Roggen and Federici's models, the cells of Miller's models can only assume a single distinct cell type, and so the fitness landscape is somewhat different to that searched by the model presented here. And again Miller's models do not require or use carefully positioned initial conditions that provide additional knowledge of the problem.

Miller's models also differ greatly from the Outer Totalistic model in the regulation and communication strategies used. Although his models use a Boolean regulation strategy the functions that it can represent and the way in which evolution can manipulate these functions differ from the Outer Totalistic model. Miller's communication strategy uses a time-dependent linear diffusion model for transmission of information between cells with a larger neighbourhood than that used by the Outer Totalistic model. This means that the transmission of information over larger distances is simplified as transmission of information between every cell does not have to be explicitly specified. Such a model might be expected to be able to generate patterns with large motifs that cannot be easily specified using relative positional information. Although the two models have produced comparable results on the test problems it would be

dangerous to use these data to make any firm conclusions about the merits of the regulatory and communication strategies used by the models as they differ in so many respects.

6.1.8 Universal Initial Conditions

It has been suggested here that the ability to constrain the developmental model so that it can only generate patterns with particular symmetries is an advantage: many conventional circuits are highly symmetrical and hence knowledge about the likely symmetries of solutions for a problem may exist before the problem is tackled. However there are likely to be many interesting problems where no such knowledge exists. Furthermore the inclusion of domain knowledge may guide evolution away from innovative designs.

Similarly the Outer Totalistic model was provided information about the French flag target pattern through prudent selection of symmetry-breaking initial conditions in the experiments presented in the Section 6.1.7. Although this was noted in Section 6.1.7, it means that the comparison between the Outer Totalistic model and Miller's CGP models is somewhat unfair as the experiments with translationally symmetric patterns suggested that the performance of the Outer Totalistic model is lower when evolution has more potentially unique contexts to manipulate.

These issues are easily addressed by the use of a set of initial conditions that break all symmetry. Such a set of conditions could be used to remove the symmetry constraints inherent in the model whenever there is no knowledge about the symmetries of good solutions. An example of such conditions is shown in Figure 6.1.9.

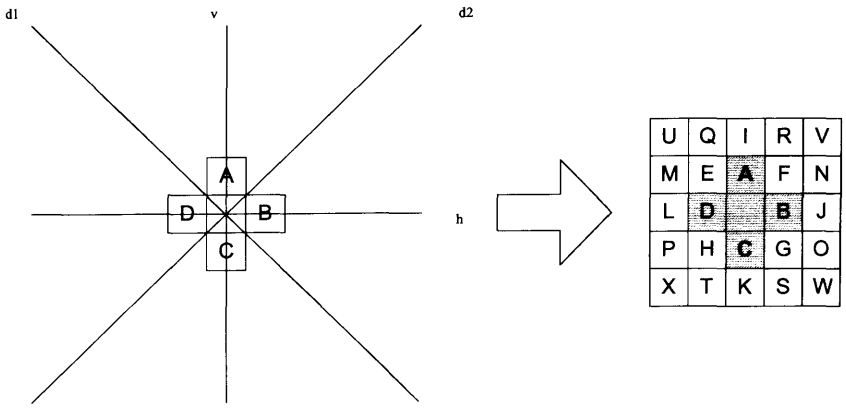


Figure 6.1.9: A set of universal initial conditions that break all symmetry and permit the maximum number of unique contexts.

They could also be used to provide a fairer comparison between the Outer Totalistic model and Miller's model. In light of this the French flag experiment was repeated using the set of initial conditions shown in Figure 6.1.9. The results are shown in the final row of Table 6.1.6. They

suggest that there may be a small performance decrease when using universal initial conditions, with the best fitness dropping by just over 0.5%, which is in line with the experiments presented earlier in this chapter. However given the size of the decrease and that the $\overline{f}_{best}$ and worst fitnesses actually increase slightly, any differences in performance may actually be masked by noise. It seems that the use of such initial conditions does not harm performance by any great margin, for this problem at least.

Another way of selecting initial conditions without using prior knowledge would be to evolve them. The conditions in a handful of cells could be included in the chromosome, much like in Chapter 4, and mirroring the evolution of maternal factors in biological development. However this approach has not been explored yet.

6.1.9 Summary and Guidelines for Suitable Problems

This section has shown through a number of experiments that the Outer Totalistic model is capable of generating large phenotypes, as will be needed to generate solutions to large circuit design problems. From the analysis of the experiments a number of guidelines can be summarized that suggest what kinds of problems this model of development is suited to:

(1) Positional information is more easily transmitted over short distances, thus problems where solutions are likely to be large and highly irregular are unsuitable for this model.

(2) If a large problem is to be tackled successfully it should exhibit regularity that allows relative rather than absolute positional information to be exploited.

(3) The model is likely to perform best when its inherent bias towards symmetrical patterns can be taken advantage of, and initial conditions should be chosen to promote this. If good solutions to a problem are likely to be of lower symmetry, the inherent symmetry can be broken by prudent selection of initial conditions. If little is known about the problem it is possible to introduce universal initial conditions that break all symmetry. Simple yet useful initial conditions could also be evolved.

(4) Care should be taken to avoid problems where many solutions are symmetrical around the initial conditions exhibit high, yet not optimal fitness: such solutions are likely to be deceptive.

The Outer Totalistic model has also been compared to similar models found in the literature and has been shown to provide comparable or better performance for the problems presented here. However this section has presented ideas that are perhaps of more general interest to researchers of computational development than the results in themselves. The idea of relative positional

information has been introduced. This, along with a simple analysis of how symmetry can both benefit and hinder development might provide the starting point for the creation of an analytical toolkit that could be used to design and compare developmental systems used in the future.

6.2 Scalability and the n-bit Adder with Carry Problem

This section demonstrates that the scalability of hardware evolution problems, which are likely to be less evolvable than the simple pattern generation problems explored above, can be enhanced by using a developmental genotype-phenotype mapping. It presents a series of experiments that evolve n-bit adder with carry circuits, ranging from a one bit adder with carry up to a seven bit adder with carry.

6.2.1 Experimental Setup

The experiments were carried out using the same model of development used by Experiment 6D in Chapter 5, and summarised at the end of Chapter 5: the chromosome consisted of a set of 20 Outer Totalistic regulatory rules and 30 structural rules that map to the same three-input virtual CLB used in Experiment 6D. Similarly the evolutionary algorithm, genetic and developmental parameters were identical to those used in Experiment 6D and evaluation was carried out using the same Virtex-based evolutionary platform, which was introduced in Chapter 3. The genetic parameters are shown in Table 6.2.1.

Operator	Type	Rate
Selection	2 Member tournament	80%
Crossover	1-point	100%
Mutation	Point	5 per chrom.
Other parameters: Generational GA with elitism, 2500 generations, population size = 100, random initialisation.		

Table 6.2.1: Parameters for the evolutionary experiments.

The task was to evolve an n-bit adder where n varied from one to seven. The area evolved for each experiment was a $2n$ x 2 array of virtual CLBs. The circuit inputs and outputs were set as shown in Figure 6.2.1. As the chromosome length depends on the number of rules provided it was identical for all problem sizes at 1620 bits. To prevent contention between the evolved area and the surrounding support circuitry the outputs on the edges of the evolved area other than those specified by the problem were forced off. For each experiment the task was to evolve a circuit that mapped the inputs to the outputs as shown in Figure 6.2.1 in accordance with an n-bit adder with carry truth table, where n ranged from one to seven. Fitness was measured as the total correct output bits across all input combinations, which gives a maximum fitness of $(n+1)$ x 2^{2n+1} and a minimum fitness of zero. 50 evolutionary runs were conducted for each problem size.

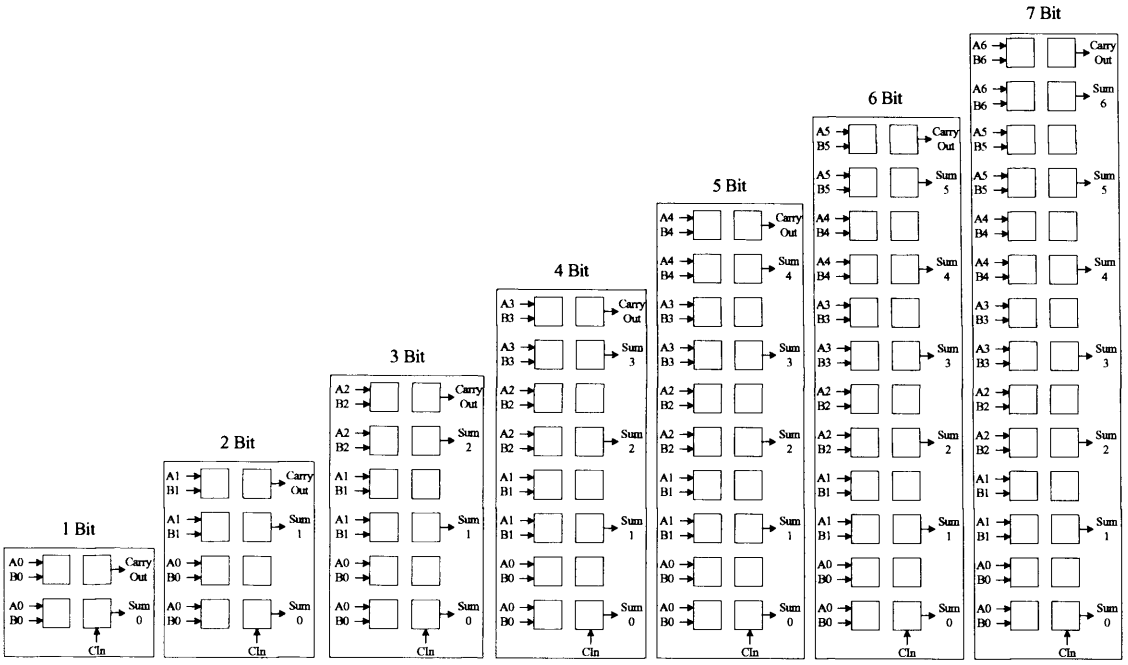


Figure 6.2.1: Diagram of evolved array areas, input points and output points for the adder scalability experiments.

The experiment was repeated using a naïve representation that mapped a gene to each component of the evolved array. The representation for one cell is shown in Table 6.2.2. Unlike the developmental system, the chromosome length of the naïve system varies with array size, from 100 bits for the 2x2 cell array used for the one bit problem up to 700 bits for the 14x2 cell array used for the seven bit problem.

Locus	Component	Bits (Representation)
0-2	Input 1	3 (GN,GS,GE,GW, FN,FS,FE,FW)
3-5	Input 2	3(GN,GS,GE,GW, FN,FS,FE,FW)
6-8	Input 3	3 (GN,GS,GE,GW, FN,FS,FE,FW)
9-16	GLUT	8 (8 minterms)
17-24	FLUT	8 (8 minterms)

Table 6.2.2: Equivalent naïve representation for the adder problem.

6.2.2 Results and Analysis

A summary of the experimental results are shown in Table 6.2.3. They suggest that for the smaller adder problems (one to three bit adders) the percentage of runs reaching optimal fitness is greater with the naïve system than the developmental system. However the percentage decays much more gently with increasing problem size for the developmental system than for the naïve system. This means that for larger adder problems (four bit and above) the developmental system outperforms the naïve system, and the performance differential increases with problem size. This can be seen more clearly in Figure 6.2.2, which shows plots of the percentage of runs reaching optimal fitness against problem size for both systems.

Adder Size / Bits	Max Fitness	% Runs with Max Fitness (Development)	% Runs with Max Fitness (Naïve)	Mean Best Fitnesses as % of Max for Development ($Dev\ \overline{f}_{best}$)	Mean Best Fitnesses as % of Max for Naïve ($Nv\ \overline{f}_{best}$)	Std. Dev. Best Develop. Solns. normalised to % ($Dev\ \sigma_{best}$)	Std. Dev. Best Naïve Solns. normalised to % ($Nv\ \sigma_{best}$)
1	16	32	100	87.88	100	11.81	0
2	96	26	48	85.69	95.44	12.70	4.95
3	512	18	28	85.80	92.48	13.50	6.19
4	2560	14	4	82.13	88.58	13.57	5.74
5	12288	12	0	82.61	82.99	12.30	5.39
6	57344	10	0	83.87	79.59	12.50	5.37
7	262144	8	0	82.09	76.61	12.37	4.81

Table 6.2.3: Results of the adder scalability experiments.

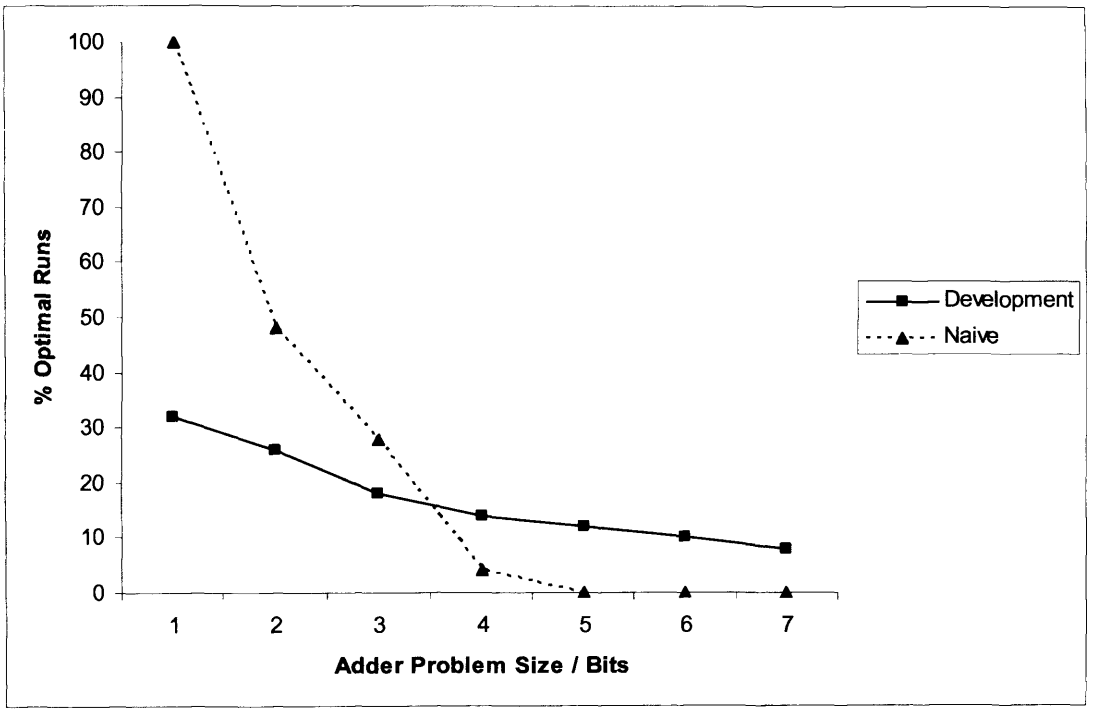


Figure 6.2.2: A plot of the percentage of runs reaching optimal fitness against problem size.

Figure 6.2.2 suggests that for both systems the relationship between fitness and problem size might best be modelled as exponential. Hence the percentage data was linearised by applying a logarithmic function, as shown in Figure 6.2.3. By the five bit experiment the percentage of optimal runs for the naïve system had already decayed to its minimum. Hence the data points gathered from these experiments were omitted from Figure 6.2.3 and the following statistical treatment as any error in their values would be amplified greatly by the linearisation function and would be likely to dominate any observed trend. Once linearised, the remaining data could be treated using statistics that assumed a linear model, yet still yield useful information.

The coefficient of determination, r^2, represents the fraction of variability in the response (the logarithm of the percentage of optimal runs) that can be explained by the variability in the independent variable (the problem size) assuming a linear model (Sheskin, 2003). A perfectly linear trend would return a value of 1, and random data 0. This was calculated for both sets of

data in shown Figure 6.2.3 to test the validity of the assumption that the relationship between the raw percentage of optimal runs and problem size was exponential, and yielded values of 0.985 for the developmental trend and 0.913 for the naïve trend. The r^2 values that resulted are reasonably high, agreeing with the initial observation that the trend might be exponential, but should be treated with caution owing to the limited data available, in particular for the naive system.

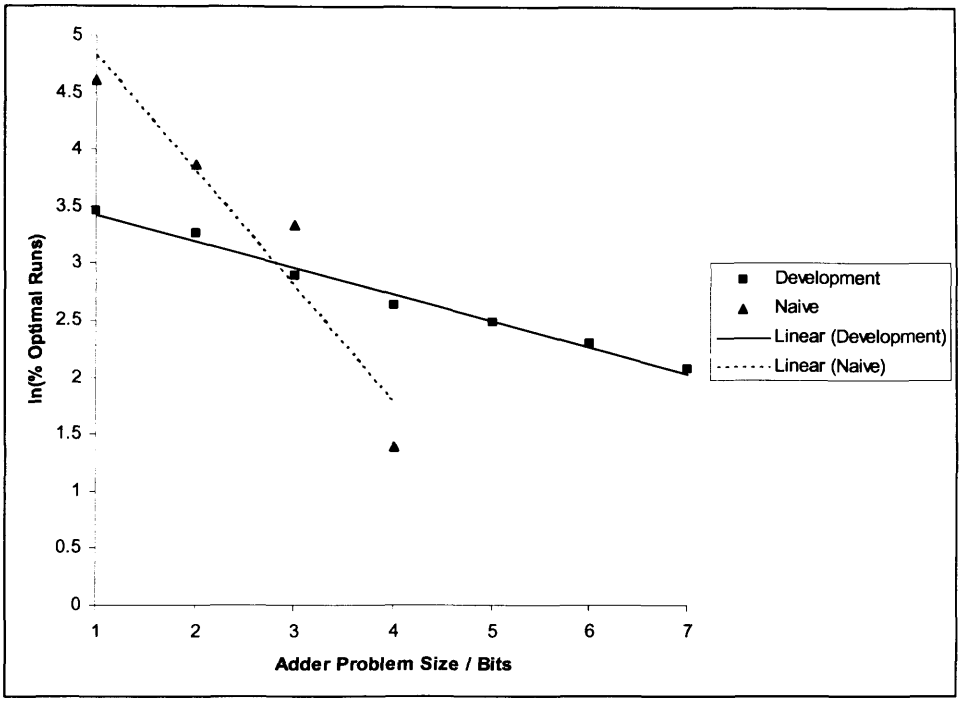


Figure 6.2.3: A plot of the percentage of runs reaching optimal fitness, linearised by applying a logarithmic function, against problem size.

From an engineering perspective the percentage of optimal runs (or some function of this value) is a useful measure, as it is vital for real-world circuits to operate perfectly. However as this measure decays to a minimum value over the studied problem sizes, other measures were used to provide further evidence of scalability.

One alternative measure is raw fitness, normalised to 100% to allow comparison between different problem sizes. If evolution scaled poorly with problem size, a strong negative correlation between problem size and performance would be expected. Hence one means of statistically detecting a difference in scalability between the naïve and developmental systems is to compare the correlations between performance and problem size of the two systems. Plots of fitness, normalised to 100 and linearised by applying a logarithmic function, against problem size are shown for the developmental system in Figure 6.2.4(a) and for the naïve system in Figure 6.2.4(b).

The Pearson Product correlation is a standard statistical test for correlation, and returns a coefficient, r, that can vary from a value of 1 for perfectly correlated data through 0 for

uncorrelated data to -1 for perfectly negatively correlated data and allows confidence intervals for the correlation to be calculated. Details of the method can be found in (Sheskin, 2003). The Pearson coefficient was calculated to test for a correlation between problem size and fitness using all 50 runs of each system. The r value was calculated to lie between -0.87 to -0.81 with 95% confidence for the naïve trend and -0.23 to -0.02 with 95% confidence for the developmental trend. The range calculated for the r-value of the naïve trend suggests that there is a high rate of decay in performance as problem size increases, whereas the r-value range for the developmental trend suggests a much lower rate of decay. The high confidence in the predicted ranges, taken with the large difference between the predicted ranges suggests that there is a highly significant difference in correlation. Hence the data suggests that the developmental system *scales* better with problem size than the naïve system.

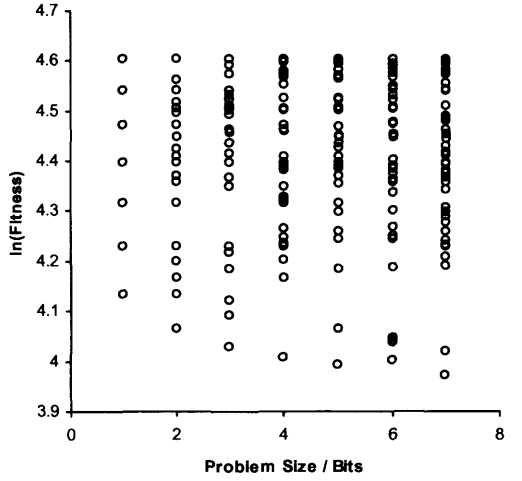


Figure 6.2.4(a): A plot of fitness normalised to 100 and linearised by applying a logarithmic function, against problem size for the developmental system, for all 50 runs.

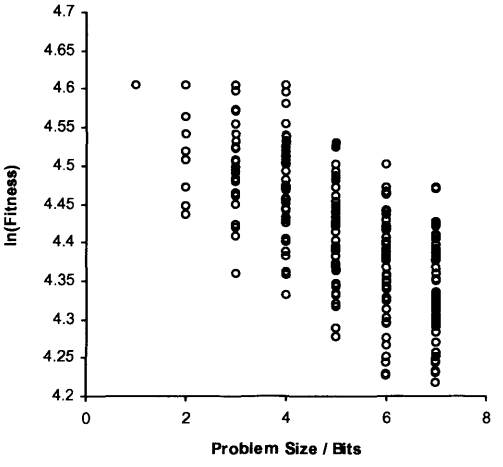


Figure 6.2.4(b): A plot of fitness normalised to 100 and linearised by applying a logarithmic function, against problem size for the naïve system, for all 50 runs.

Unfortunately analysing the data with the Pearson statistic has some flaws: it assumes that the data are linear and that both populations are normally distributed. Although the linear model is a reasonable assumption given that the data was linearised using the same function that yielded high coefficients of determination presented earlier, fitness did not appear to be distributed normally for each problem size, particularly for the developmental system (This is reflected in the high variances reported in Table 6.2.3.) Repeating the correlation calculation using the Spearman method (the non-parametric analogue of the Pearson method) using data that had been normalised to 100 but not linearised yielded similar predicted r ranges of between -0.28 and -0.07 for the developmental trend and -0.88 and -0.82 for the naïve trend. Again there is a large difference between the predicted ranges of r for both trends, and the trends have been calculated with high confidence. Thus just as with the Pearson statistics, the Spearman statistics strongly support the hypothesis that development can enhance the scalability of hardware evolution.

A further analysis was carried out to lend support to the evidence above. The aim was to produce paired data for each problem size to demonstrate that the performance *differential* increased monotonically with problem size. It was stated above that from an engineering perspective the percentage of runs that achieved optimal fitness was the performance measure of choice. However as for some problem sizes this had decayed to its minimum another measure was used in its place to simplify the analysis. Figure 6.2.5 shows plots of performance versus problem size using the mean of the best fitnesses achieved for each problem size for the naïve system ($Nv\overline{f}_{best}$) and the developmental system ($Dev\overline{f}_{best}$). When the r^2 coefficients of determination were calculated for all problem sizes using this measure they yielded values of 0.98 and 0.70 respectively, suggesting that a linear model fits the data well enough for linear analyses to provide useful and realistic information. Hence it was decided that the mean fitness of the best solutions might be less skewed hence more plausible to analyse statistically. It was noted above that the raw data did not appear to be distributed normally for each problem size hence using a mean value to measure performance is not a particularly good choice. However the benefit of including the large amount of data gathered for the larger problem sizes (where the naïve system could not generate any optimal solutions) in the analysis was considered to outweigh this disadvantage.

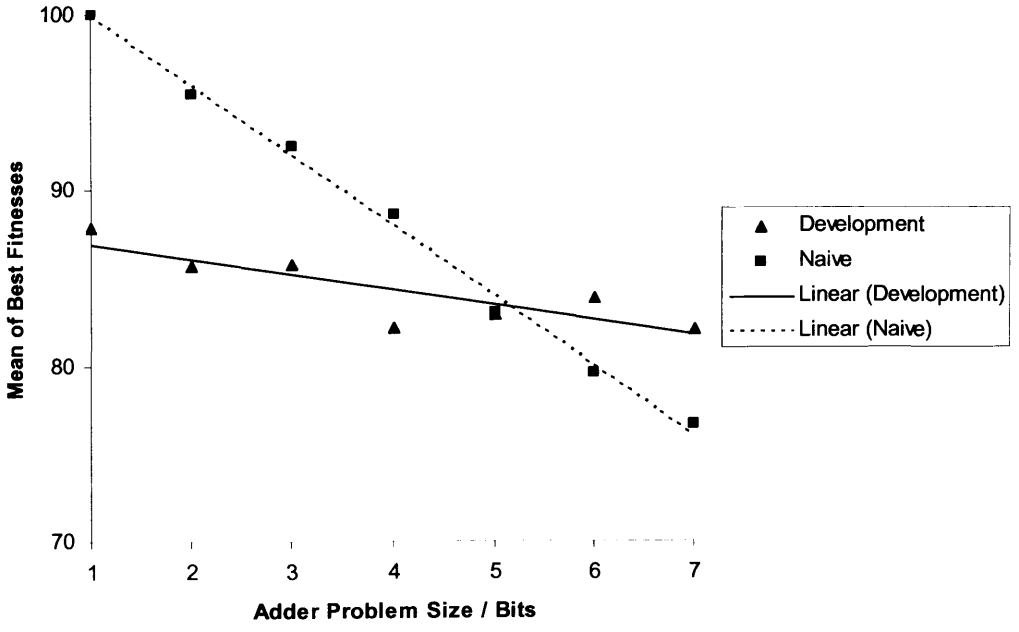


Figure 6.2.5: A plot of the mean of the best fitnesses for each problem size against problem size.

A plot of the difference between $Dev\overline{f}_{best}$ and $Nv\overline{f}_{best}$ for each problem size is shown in Figure 6.2.6, along with the least squares line of best fit. The Pearson correlation coefficient for this statistic was 0.979, suggesting a linear model is a reasonable assumption, and confirms statistically what is visually very clear: there is a strong positive correlation between problem size and enhanced performance of the developmental system over the naïve system. This clearly

shows that the performance of the developmental system *scales* better with problem size than the performance of the naïve system, confirming that for the combination of evaluation method and evolutionary algorithm used here, this developmental representation has enhanced the scalability of evolution when applied to the two bit adder with carry hardware evolution problem.

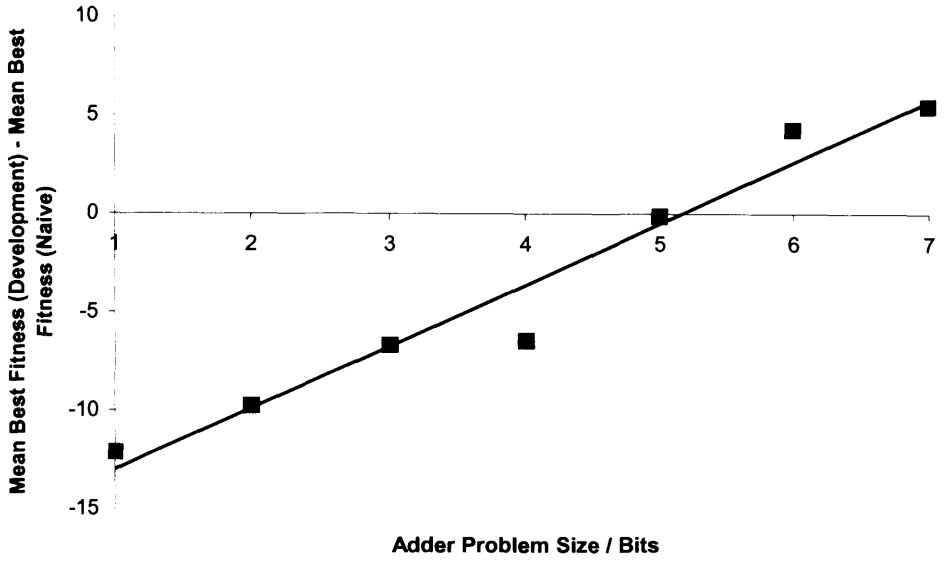


Figure 6.2.6: A plot of the difference between $Dev\,\overline{f}_{best}$ and $Nv\,\overline{f}_{best}$ against problem size.

6.2.3 Circuit Analysis

The evolved circuits exhibit a number of features that reinforce the argument that development can aid scalability, that it can do so without imposing hard constraints that are associated with traditional designer-imposed abstraction methods, and that this can lead to the discovery of innovative designs.

Modularity and Reuse without Hard Constraints

The most striking feature of the evolved designs is the high level of component reuse. It has been argued throughout this thesis that design reuse is the primary means by which development can enhance scalability. Figure 6.2.7 shows the first optimal circuit evolved for the two bit problem, which was typical of the optimal solutions evolved. Although the chromosome contained 30 structural rules, only nine of these were actually activated during the development of the circuit. Cells 1, 2, 5, 6 and 7 have assumed identical input configurations: inputs 1, 2 and 3 are driven by their west F-LUT, south G-LUT and west G-LUT respectively. These configurations were generated by the activation of the same two rules in each of these cells. As no rules relating to Input 1 were activated throughout development, this input took on a default configuration in all cells. Cells 0 and 3 share two inputs with the cells already mentioned, which were generated by the same rules as in the other cells.

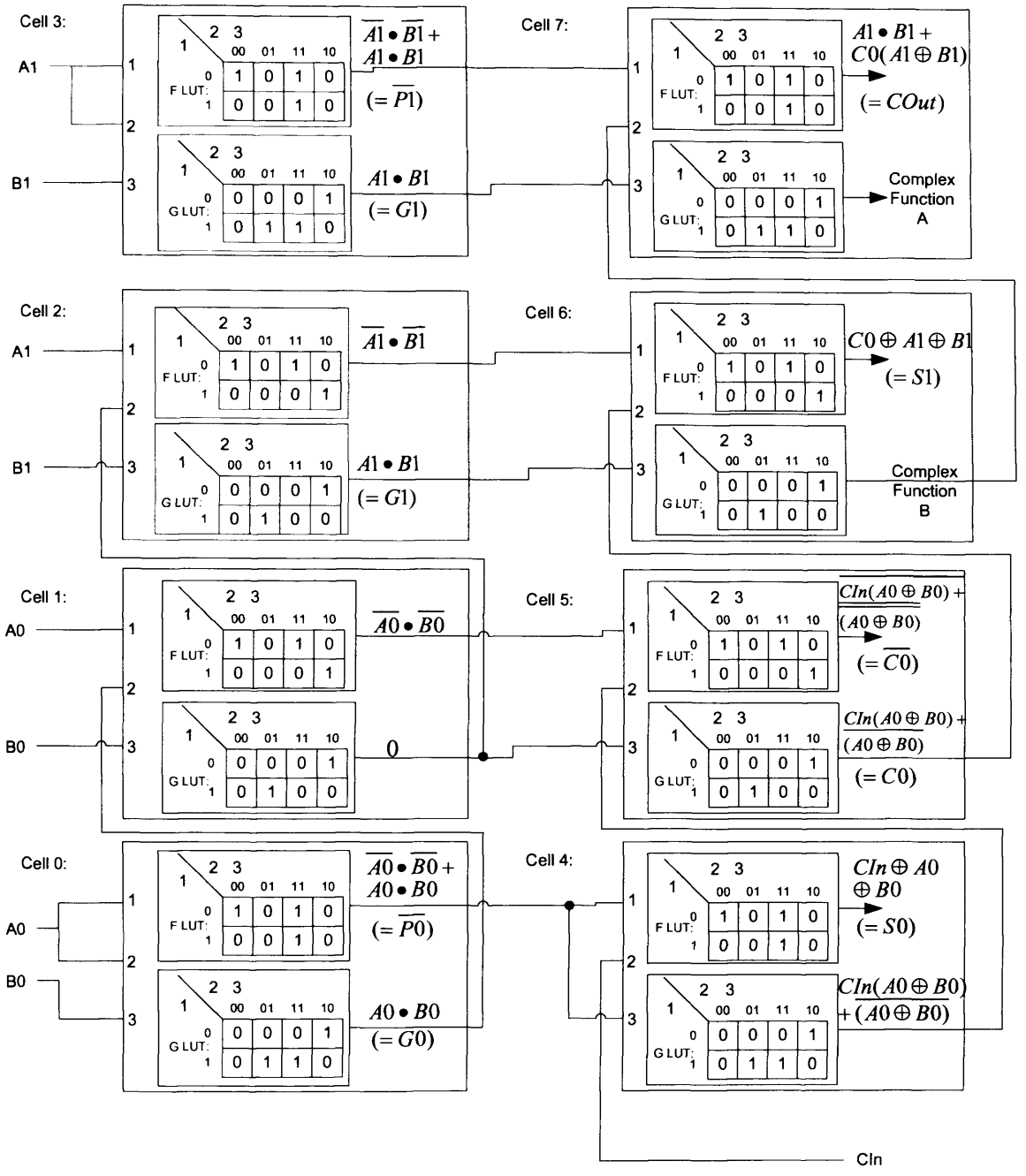


Figure 6.2.7: An evolved optimal two bit adder with carry. The figure shows the evolved CLB inputs, the evolved LUT configuration bits represented as K-Maps and the logical output of each LUT in the context of their inputs.

The rule that generated the remaining input in the other cells was inhibited in these cells due in part to the unusual cell neighbourhoods at the edges of the array and in part due to a chain of interactions that can be traced back to the asymmetric initial conditions provided. The only cell with a unique set of inputs, Cell 4, again shares two inputs with the cells discussed above, and again these were generated by the same rules. Generation of the remaining input was again inhibited by the initial conditions. Hence there are three cases of input reuse at the cellular level: a common set of inputs, generated by identical rules is present in most cells. Two cells share a set of identical inputs generated by identical rules, and a single cell has a unique set of inputs.

This can be interpreted, as in Chapter 4, as evolution applying a design abstraction that imposes a common set of inputs across the circuit, but breaking the abstraction where necessary. The evolved logic also shows an extremely high level of reuse. Cells 1, 2, 5 and 6 share identical logic and were generated by the same set of rules. Cells 0, 3, 4 and 7 also share identical logic generated by a common set of rules. Furthermore there is a great deal of similarity between the logic in each set, with several rules common to both sets of logic.

Design Innovation

The hand-designed 2x4 cell adder presented at the end of Section 5.3.1 is a ripple-carry adder, and of the designs familiar to the author is the only one which could be created within the geometry of the evolved array. Examination of the inputs to each cell of the evolved circuit shown in Figure 6.2.7 reveals that signals pass to the east and to the north, much like a ripple-carry adder. The behaviour of the circuit shown in Figure 6.2.7 can be simplified by considering the logical terms actually generated at each LUT output in the context of their supplied input signals, and are shown at the outputs of each LUT in Figure 6.2.7. Here it can be seen that although quite different from the hand-designed adder of Chapter 5, the circuit could be interpreted along the lines of a ripple-carry adder: although there was no explicit fitness bias towards doing so, the circuit actually generates and uses a traditional first stage carry out term (Marked C0 in Figure 6.2.7) that propagates up a carry chain in the right hand column of the circuit. The way the circuit uses the rest of the logic is not traditional for a ripple carry adder, although terms equating to the Generate functions and the inverse of the Propagate functions of a carry look-ahead adder (which are labelled as Px and Gx on Figure 6.2.7) are found throughout the circuit and directly used in the generation of the final sum and carry signals. It is not surprising that such similarities to traditional adders are found in the evolved circuits as the geometry of the array and the positioning of the inputs and outputs were selected to allow such a circuit to be generated, as discussed in Chapter 5. Furthermore the adder problem space is extremely well known to traditional designers using components that perform logical functions. Hence it is likely that evolution would discover designs similar in style to traditional adders, given the components provided. However it is important to realise that the evolution has managed to discover these circuits without the provision of explicit knowledge or bias towards good designs, and while operating at a low design abstraction that allows a large area of non-traditional search space to be considered.

A more pertinent point to consider is whether development might be of any engineering benefit for other problems that are not so well understood. A question that might provide some insight into this is whether evolution decomposes the problem in the same manner as would a traditional designer. If not, then it is likely to search areas of space that are not searched by traditional circuit design techniques, which might be of benefit when searching for innovative

designs. If this is the case, development would enjoy an advantage over other mechanisms that have been touted to improve scalability that were surveyed in Chapter 2, including function level evolution (Murakawa et al., 1996) and increased complexity evolution (Torresen, 2000b). A common theme of these mechanisms is that the problem must be decomposed manually by a designer (or arbitrarily) and that the decomposition that evolution is forced to respect might steer it away from areas of design space that contained potentially innovative circuits.

Traditional ripple-carry adders consist of modules of full adders and communication between them is limited to a single carry signal that passes between them. Larger adders can be constructed simply by adding full adder modules to the most significant end of a smaller adder. The hand-designed adder of Chapter 5 was composed of two full adder modules, one in the southernmost four cells and one in the northernmost four cells. A larger adder cannot be constructed from the evolved solution presented here by translating a copy of the southernmost four cells to a new 2x4 array to the north of the current circuit, as the south-easterly cell of the evolved circuit accepts a carry-in signal from the G-LUT of the cell that lies to the south of cell 4, but the carry out signal is generated in the F-LUT of the north-westerly cell, and a different function is generated in the G-LUT. Similarly, the circuit cannot be cleanly decomposed into the two stages of a traditional ripple-carry adder by considering the southern four cells as the first stage and the northern four cells as the second stage, as the carry signal is not the only signal used to generate the more significant outputs that passes between these sets of cells. None of the four optimal two bit solutions studied in detail could be considered a general solution in this way. One circuit that was evolved using the same developmental model but a fixed routing architecture in Experiment 6C of Chapter 5 could be decomposed in this way: the only signal to pass between the two stages was a traditional carry out signal, and the carry out from the second stage was generated in the correct position for either a single four cell adder unit or the entire circuit to be repeated to generate larger adders. However in general it seems that the bias towards structural design reuse that development applies to the search means that evolution favours generating circuits with high design reuse at the expense of the partitioning of communication between modules, as a traditional designer might do. This results in adder designs that are structurally highly uniform, but not general. (This might be expected as the problem did not specify that a general solution might be favourable.)

Even though the circuit phenotypes generated by evolution are occasionally decomposable into traditional full adder modules, it was found that in these cases the developmental processes that generate them are not: when the developmental process that generated the two bit adder shown in Figure 6.2.7 was applied to a larger 2x8 array and tested using the four bit adder fitness function, a low fitness solution resulted. This was because a number of protein interactions involved in the development of the circuit relied on the non-heterogeneous neighbourhoods of

the edge and corner cells. As the relationship between these edges and the rest of the circuit is altered by moving to a larger array, the pattern of contexts that the developmental process generates on a 2x8 array are different to when a 2x4 array was used. When the same procedure was carried out on the solution from Experiment 6C that was composed of full adder modules, it also failed to produce a high fitness solution. Hence it seems that in general neither the developmental processes used by evolution nor the evolved circuits are general solutions. Again it should be stressed that this might be expected as there was no explicit bias towards doing so.

Seven Bit Adders

Figures 6.2.8 and 6.2.9 show two optimal circuits evolved for the seven bit problem. Both have been verified using a logic simulator. The circuits presented in Figures 6.2.8 and 6.2.9 again display a high level of reuse: in the circuit of Figure 6.2.8 only three distinct LUT configurations are used and the inputs for each CLB are identical across the array. While the circuit in Figure 6.2.9 shows a similar level of reuse it has a quite different structure: the LUTs are identical for each cell and it is the inputs that vary between cells. The mechanism that the developmental model of this chapter uses to determine LUT configurations is quite different to the mechanism it uses to determine CLB inputs: a minterm is set in the final circuit only if its counter reaches a predetermined threshold value, and is independent of whether other minterms are set within the LUT. CLB inputs are selected by competition: the input that is most active throughout development is selected for the final circuit design. This suggests that evolution is capable of manipulating either developmental mechanism to search for useful circuits, and can be taken as a hint that evolution is not using some trick inherent to one or other of the mapping mechanisms to provide scalability.

However this is only a hint. There are many similarities between the development mechanisms for inputs and LUT configurations used here (for instance the underlying model of protein interaction is identical). Whether the scalability apparent here is truly the result of some general feature of all developmental models could be determined by repeating these experiments using a range of different models, but is beyond the scope of this thesis.

Innovation Revisited

The mechanisms of design reuse for the seven bit problem appear to be very similar to those used in the two bit problem. Again the designs cannot be easily decomposed into a series of full adders, suggesting that evolution does not merely apply development to decompose the problem in a traditional manner. However this evidence is rather tentative. To support it the scalability experiments using the developmental system were repeated, this time providing evolution with knowledge about how the problem would be decomposed traditionally.

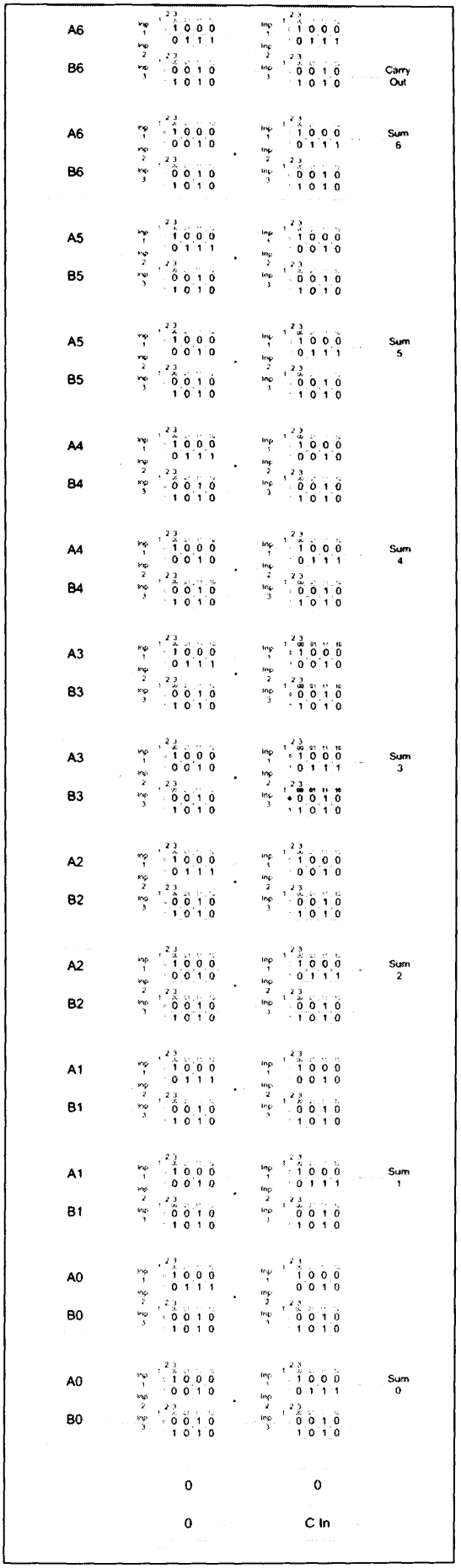


Figure 6.2.8(a): The first optimal seven bit adder with carry evolved.

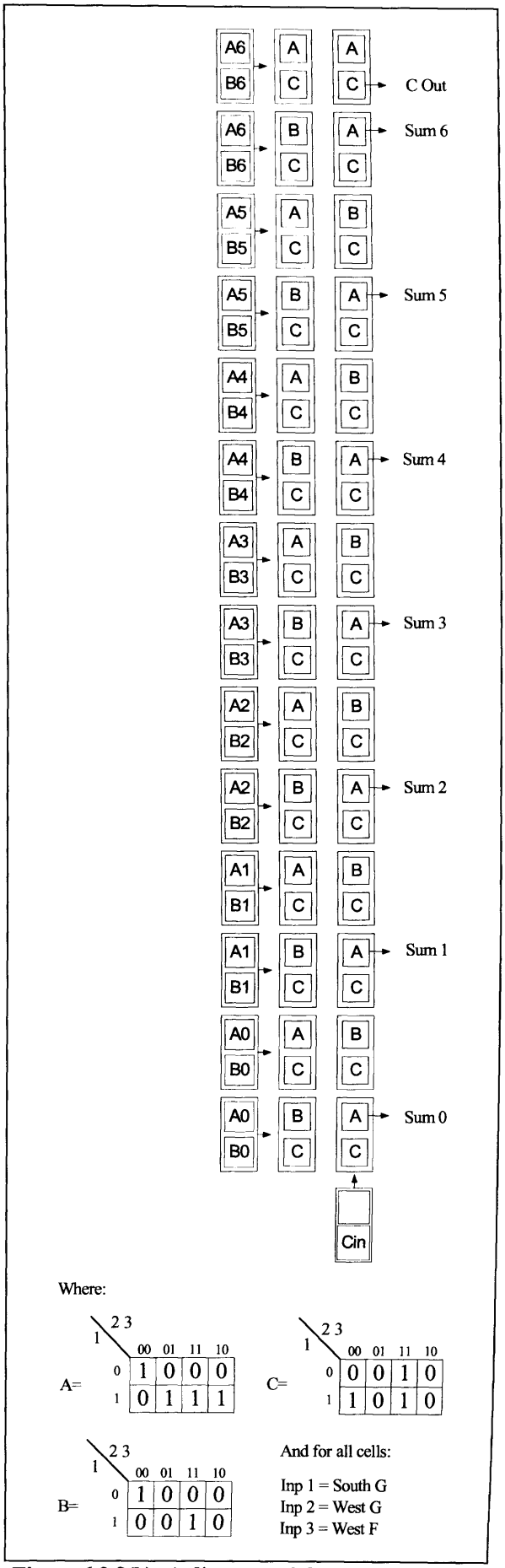


Figure 6.2.8(b): A diagram of the adder in Figure 6.2.8(a) highlighting the level of design reuse.

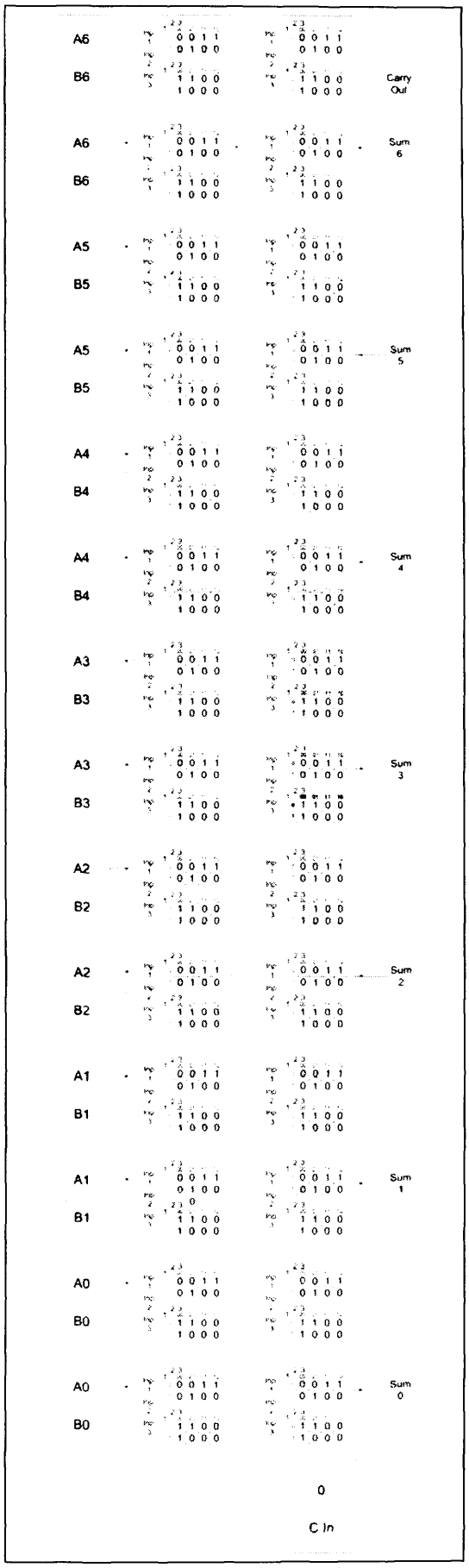


Figure 6.2.9(a): The second optimal seven bit adder with carry evolved.

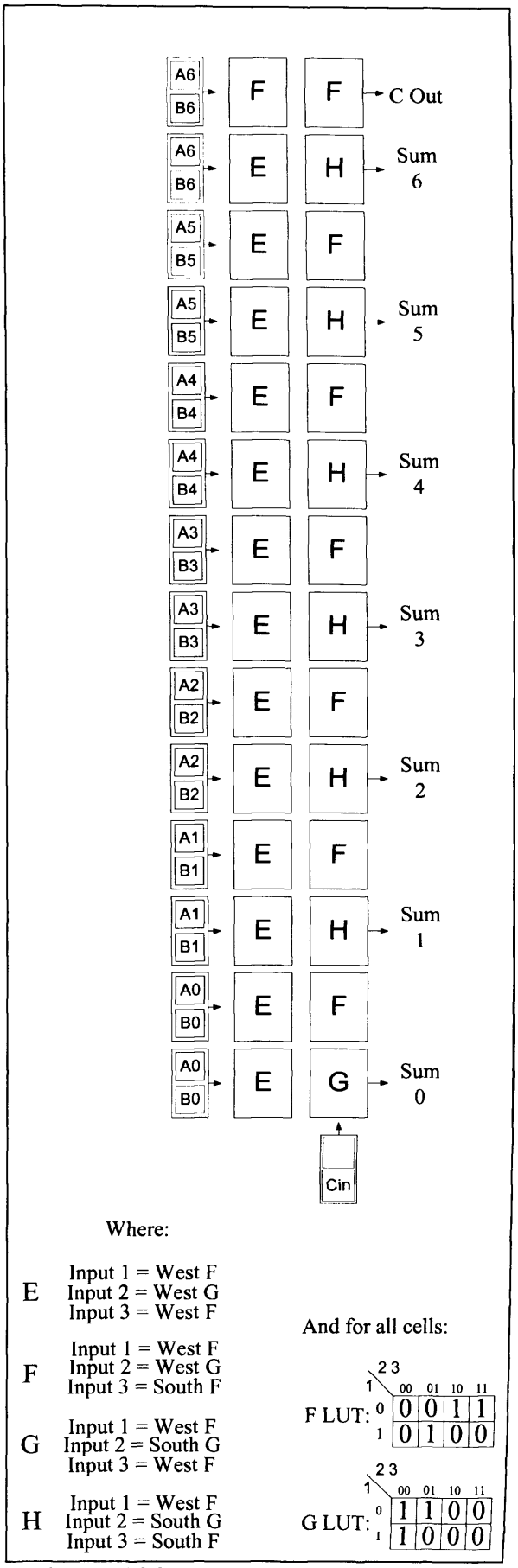


Figure 6.2.9(b): A diagram of the adder in Figure 6.2.9(a) highlighting the level of design reuse

To achieve this, an incremental fitness function that decomposed the problem according to output was used. Fitness was calculated as in the earlier set of experiments, but a fitness reward was only provided for a more significant output if the less significant outputs were generated perfectly. This means that evolutionary search is biased towards finding solutions that solve the less significant outputs first, much like a traditional designer might create a ripple-carry adder. A plot of the mean of the best fitnesses (normalised to 100) from 20 runs of incremental evolution against problem size is shown in Figure 6.2.10 along with results from non-incremental experiments presented earlier in this section for comparison.

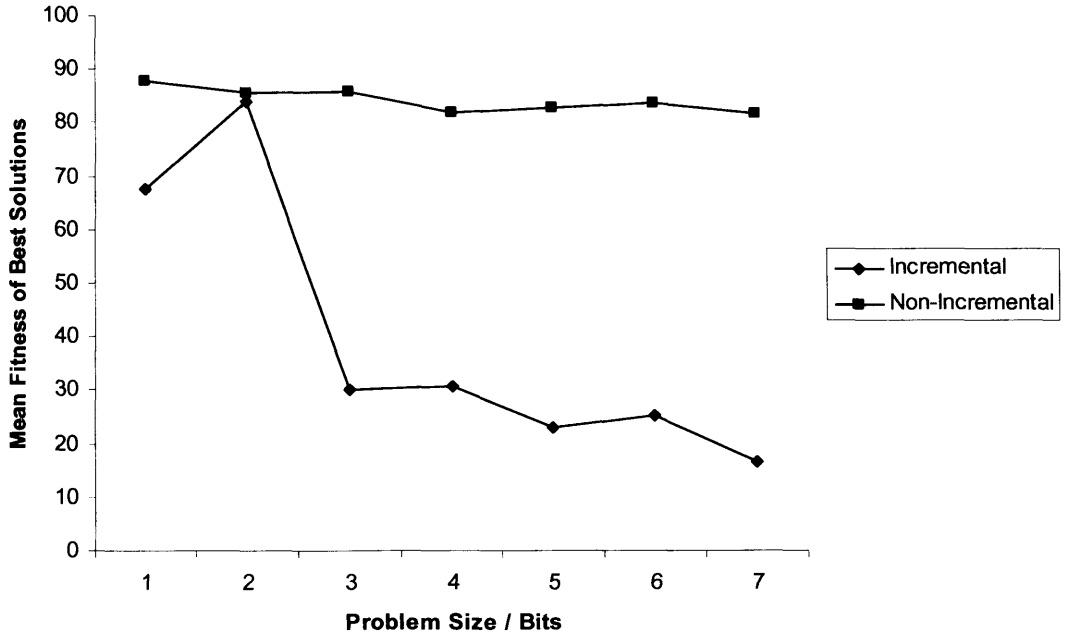


Figure 6.2.10: A plot of the mean of the best fitnesses against problem size for incremental and non-incremental evolution.

The results show that when evolution is biased towards a traditional composition scalability is not nearly as great as when evolution is free to decompose the problem as it sees fit. This suggests that when no bias towards a traditional decomposition is applied, evolution decomposes the problem in a non-traditional way, and that this has some benefit to evolvability. Hence not only does a developmental approach provide additional opportunities for innovation over techniques that rely on decomposition (Murakawa et al., 1996; Torresen, 2000b) or mechanisms that combine development and decomposition such as (Shanthi et al., 2004), but also the scalability of the design process can be greater.

Anecdotal evidence from the evolutionary history of the non-incremental two bit adder problem suggests why this might be true. For this problem it was found that the first fitness improvement over a circuit with a fixed output discovered by many runs was to make a pass-through connection between input B1 and output Sum1. As this innovation is both easily discovered and improves fitness there is likely to be a strong selection pressure towards its use, hence it is likely

to spread throughout the population quickly once it arises. The model of development used here is biased towards a high level of design reuse, hence is likely to attempt to reuse the innovation that led to an improvement in fitness in other cells. This means that evolution would be biased towards searching designs that connected many of the circuit inputs directly to the outputs, thus facilitating the discovery of the feedforward design abstraction, which appears to be useful for this problem. In the incremental experiment the fitness function prevented evolution access to this information until a solution was found that produced the S0 output perfectly. This may have impeded the discovery of the seemingly useful feed-forward abstraction.

6.3 Scalability and the Even n-bit Parity Problem

Section 6.2 presents strong evidence that scalability can enhance hardware evolution. However this was only demonstrated for a single problem: the n-bit adder with carry problem. Furthermore a crucial aspect of the developmental model, the method of communicating information across space, was designed with this problem in mind, as was the geometry of the evolved area. Hence although the evidence detailed above is strong, its scope and thereby its significance to the hardware evolution community is limited. If a similar enhancement in scalability could be demonstrated for another problem that the developmental system was not specifically designed to solve, the central claim of this thesis would be strengthened considerably and the work would bear more significance. With this in mind, experiments similar to those presented in Section 6.2 were carried out for the even n-bit parity problem.

6.3.1 Problem Description

The even parity problem is a benchmark problem that is popular with Evolutionary Computation researchers. The problem can be described thus: given an n bit word of data, the parity circuit must generate an additional parity bit. The value of the parity bit should be determined by the other data bits. For an even parity scheme, the parity bit should be set if there are an odd number of high bits in the data provided, otherwise it should be unset. Thus when the parity bit is considered with the raw data word, the entire word should always contain an even number of bits. In other words the parity bit can be calculated as the modulo 2 of the summed input bits.

Parity generators are used widely in digital communications applications and to improve memory integrity. Should a communication channel or an operational fault in a memory introduce a single bit error into a data word that contains a parity bit, the error can be detected by a parity detector circuit, and appropriate action can be taken to recover from the fault.

The problem has been explored by many machine learning researchers, particularly by those researching ANNs (Liu and Yao, 1997; Hohil et al., 1999; Hornby, 2003b) and genetic programming (Koza, 1994; Rosca and Ballard, 1994; Poli et al., 1999). The main reason for its popularity is that parity functions tend to present extremely discontinuous landscapes. Hence they are particularly hard to learn using a gradient-based search method such as ANN backpropagation or an evolutionary algorithm. In fact when evolution is forced to use an abstraction where the only primitive functions available are the logical functions XOR and EQ that would traditionally be used in the design of Boolean parity functions, the fitness of the entire search space assumes only one of two values; hence the problem resolves to a "needle in a haystack" search (Langdon and Poli, 1998). This is a perfect example of when restricting evolution to work with complex primitives that are helpful for the traditional design process actually harms evolvability. As discussed in Chapter 2 one competing approach to the scalability problem, Function Level Evolution (Murakawa et al., 1996) applies domain knowledge in this way to simplify circuit design problems, and unlike the developmental approach evolution is not free to alter the high level primitives provided. Hence for this problem at least, it is expected that using a developmental system that can craft high level functions that transform the evolutionary search space into something more tractable is likely to outperform Function Level Evolution and similar approaches that impose hard constraints based on traditional domain knowledge.

The parity problem has also been a popular target for those exploring scalability and design re-use (Koza, 1994; Rosca and Ballard, 1994; Yu and Miller, 2002; Hornby, 2003b; Walker and Miller, 2004). One traditional design for a parity generator decomposes the circuit into a series of modules arranged in a chain. Each module performs an XOR on the input from the previous module in the chain and an input bit, passing the output to the next module in the chain. Hence as with the adder problem it is known that the problem space contains modular solutions which development can exploit, although given the example above such modules might be harmful if discovered at the outset of evolution and used exclusively thereafter. The problem is also easily scalable, simply by increasing the number of bits in the word for which parity should be generated.

6.3.2 Experimental Setup

This section provides details of experiments that evolve even n-bit parity circuits, ranging from a two bit parity generator up to a twelve bit parity generator. The experiments were carried out using the same model of development used earlier in this chapter: the chromosome consisted of a set of 20 Outer Totalistic regulatory rules and 30 structural rules that map to the same three-input virtual CLB used in Experiment 6D. The evolutionary algorithm, genetic and

developmental parameters were identical to those used above and evaluation was carried out using the same Virtex-based evolutionary platform.

The task was to evolve an even n-bit parity generator where n varied from two to twelve. The area evolved for each experiment was an n x 2 array of virtual CLBs. The circuit inputs and outputs for the first six problem sizes are shown in Figure 6.3.1. The chromosome length was identical to the adder experiments at 1620 bits. As in the adder experiments described above the outputs on the edges of the evolved area other than those specified by the problem were forced off. For each experiment the task was to evolve a circuit that mapped the inputs to the outputs as shown in Figure 6.3.1 in accordance with an even n-bit parity truth table, where n ranged from two to twelve. Fitness was measured as for the non-incremental experiments of Section 6.2, by summing the total correct output bits across all input combinations, which gives a maximum fitness of 2^n and a minimum fitness of zero. 30 evolutionary runs were conducted for each problem size.

The experiment was then repeated using the same naïve representation as was used for the adder scalability experiments. The representation for one cell is shown in Table 6.3.1. The chromosome length of the naïve system varied with array size from 100 bits for the 2x2 cell array used for the one bit problem up to 400 bits for the 8x2 cell array used for the eight bit problem. The naïve system was not tested on any problem sizes larger than this as its performance had clearly decayed to a minimum.

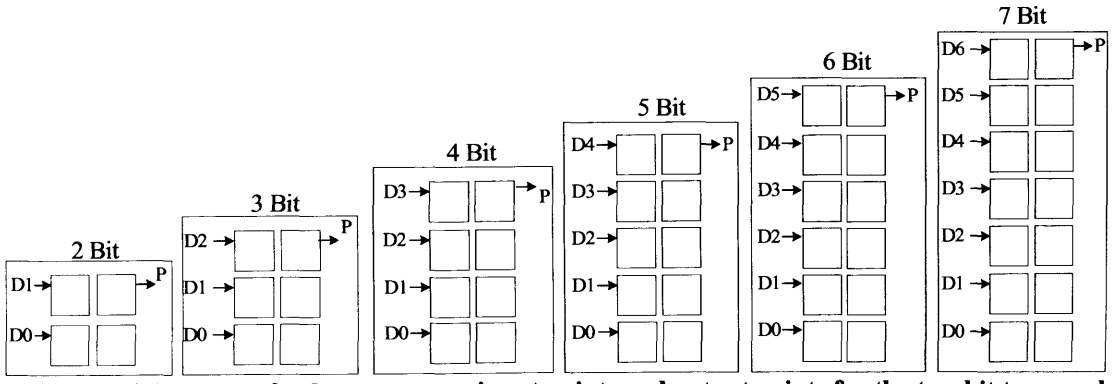


Figure 6.3.1: The evolved array areas, input points and output points for the two bit to seven bit parity scalability experiments.

Locus	Component	Bits (Representation)
0-2	Input 1	3 (GN,GS,GE,GW, FN,FS,FE,FW)
3-5	Input 2	3(GN,GS,GE,GW, FN,FS,FE,FW)
6-8	Input 3	3 (GN,GS,GE,GW, FN,FS,FE,FW)
9-16	GLUT	8 (8 minterms)
17-24	FLUT	8 (8 minterms)

Table 6.3.1: Equivalent naïve representation for the adder problem.

6.3.3 Results and Analysis

A summary of the experimental results are shown in Table 6.3.2. They suggest that just as with the adder problem, for the smaller parity problems (two and three bit generators) the percentage

of runs reaching optimal fitness with the naïve system is greater than or equal to the developmental system. However this percentage appears to decay rapidly for the naïve representation: for larger parity generator problems (three bit and above) the developmental system outperforms the naïve system by a considerable margin, and the performance differential again appears to increase with problem size.

Parity Size / Bits	Max Fitness	% Runs with Max Fitness (Development)	% Runs with Max Fitness (Naïve)	Mean Best Fitnesses for Development ($\overline{f}_{best}$)	Mean Best Fitnesses for Naïve ($\overline{f}_{best}$)	Std. Dev. Best Developed Solutions (σ_{best})	Std. Dev. Best Naïve Solns. (σ_{best})
2	4	76.67	100.00	94.17	100.00	10.75	0
3	8	90.00	90.00	96.25	95.00	11.44	15.26
4	16	83.33	20.00	93.96	60.00	14.26	20.34
5	32	73.33	3.33	92.92	51.67	12.14	9.13
6	64	86.67	0.00	95.89	50.10	11.32	0.57
7	128	83.33	0.00	94.92	50.03	12.20	0.14
8	256	70.00	0.00	90.90	50.07	15.93	0.29
9	512	80.00	N/A	94.59	N/A	12.12	N/A
10	1024	86.67	N/A	95.63	N/A	11.74	N/A
11	2048	80.00	N/A	95.41	N/A	11.50	N/A
12	4096	80.00	N/A	94.18	N/A	12.13	N/A

Table 6.3.2: Results of the even parity scalability experiments. Fitness values are normalised to 100.

Statistical analysis confirmed this apparent difference in scalability: the 95% confidence intervals for the Spearman correlation between problem size and raw fitness normalised to 100 were calculated for the naïve system as -0.8 to -0.68 suggesting a strong negative correlation. Conversely there is little evidence of a trend towards decaying percentages for the developmental system as the problem size increases: here the same statistic was calculated to lie between -0.12 to 0.10, suggesting a small negative, small positive or zero correlation might exist. (The correlation for development was also calculated using the percentage of runs reaching optimal fitness as -0.67 to 0.58, which does not suggest any correlation with confidence.) This difference in the trends can be seen clearly in Figure 6.3.2, which shows a plot of the percentage of runs reaching optimal fitness against problem size for both the naïve and developmental systems.

Hence just as with the adder problem the results suggest that performance with the developmental system scales better with problem size than performance with the naïve system, strengthening the argument that development can enhance scalability for hardware evolution problems. Whereas the developmental model had been designed specifically with the adder problem in mind, the developmental model used for the parity problem was unaltered from the adder experiments, and had not been chosen using any knowledge of the parity problem. Similarly no parameters had been altered or tuned to suit the problem. Hence although the evidence from the adder experiments confirms the hypothesis of this thesis, that the scalability of hardware evolution can be enhanced by exploiting a model of biological development, extending the scope of evidence to a second problem has enhanced its significance.

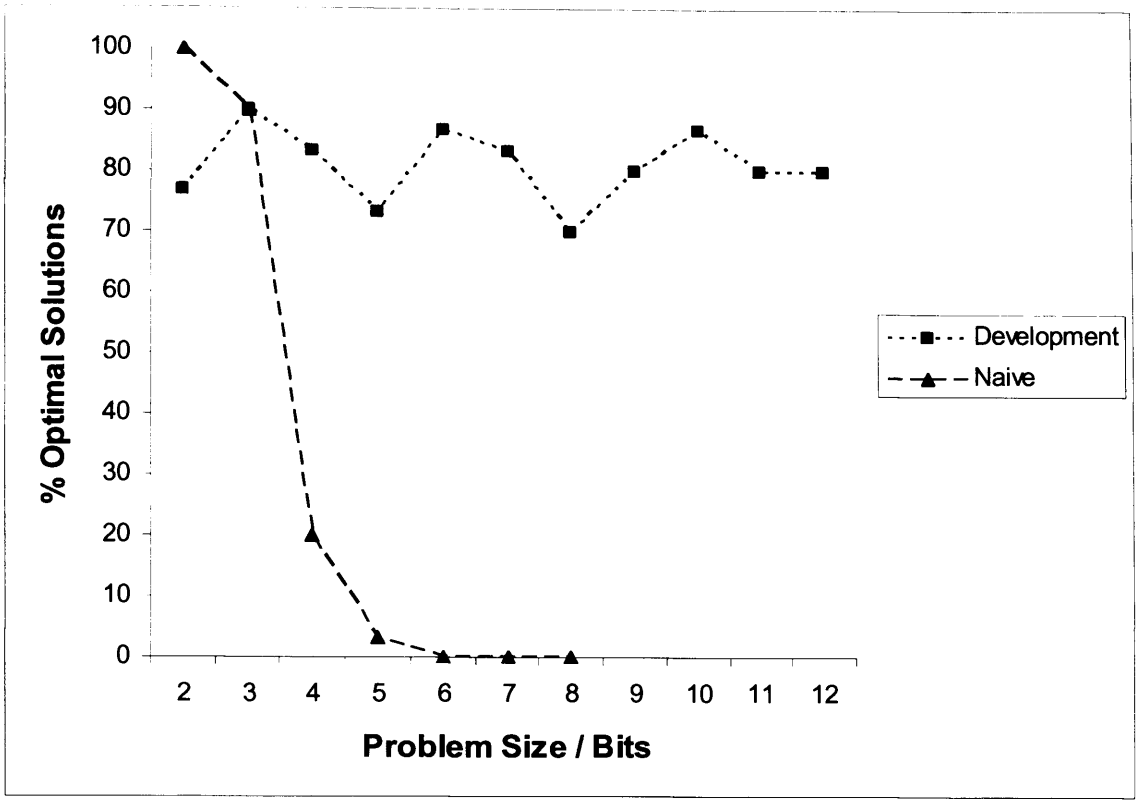


Figure 6.3.2: Plot of the percentage of optimal solutions found for the developmental and naïve system on the even n-bit parity problem.

6.3.4 Circuit Analysis

Analysis of the evolved circuits again suggests that the primary reason for the enhanced scalability of the developmental system is its ability to reuse components. Figures 6.3.3(a) and (b) show the first solution evolved to the eight bit parity problem, and is typical of the solutions evolved. Figure 6.3.3(a) shows the developed circuit inputs and configurations, and the logical output of each LUT given the context of its inputs. Figure 6.3.3(b) labels the different types of CLBs generated by evolution. The circuit is highly regular: it consists of only two distinct CLB types, labelled X and Y. As with the adder circuits presented earlier, all instances of each cell type shown are generated by the same set of structural rules, thus development provides reuse for the parity problem by the same mechanism as for the adder problem.

The solution in Figures 6.3.3(a) and (b), and in fact almost all the solutions to the parity problem that were analysed, exhibit even greater regularity than the analysed solutions to the adder problem. Just as in the circuit of Figures 6.3.3(a) and (b), most solutions tended to use a single type of CLB across the majority of the array and a second type at the edges and corners where development finds it easiest to generate inhomegeneities in protein concentrations.

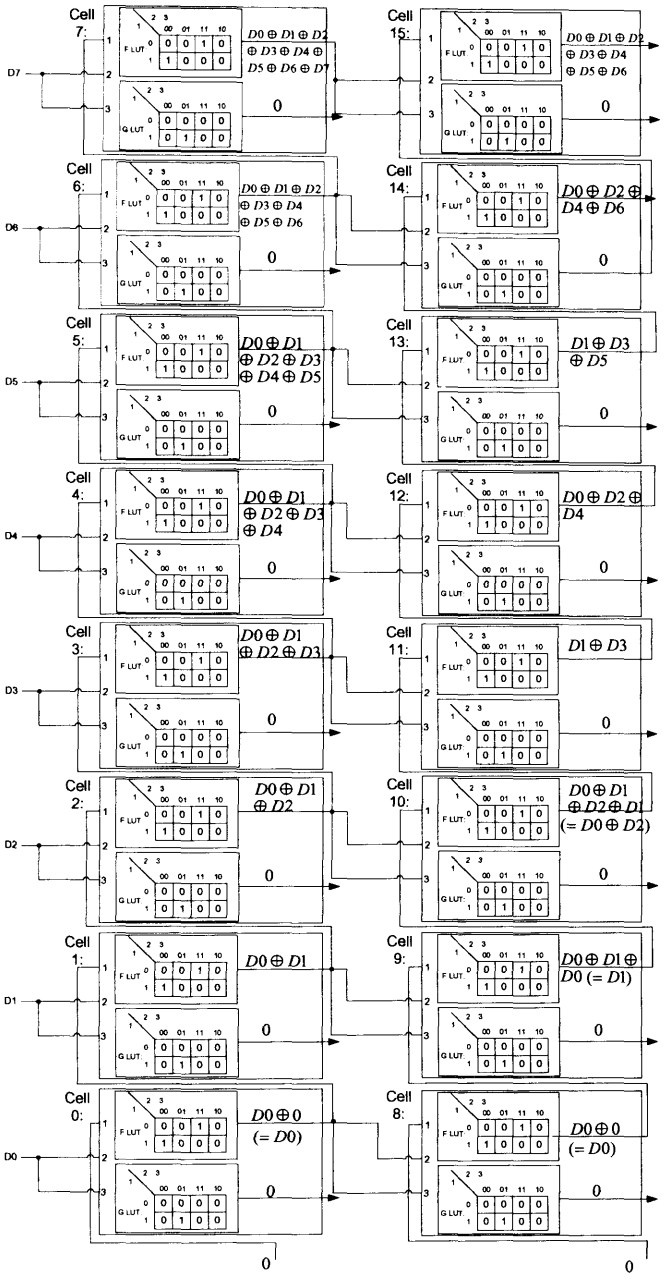


Figure 6.3.3(a): The first 8 bit parity circuit evolved with development, showing inputs, K-Maps of LUT configurations and the logical outputs of each LUT in the context of its inputs.

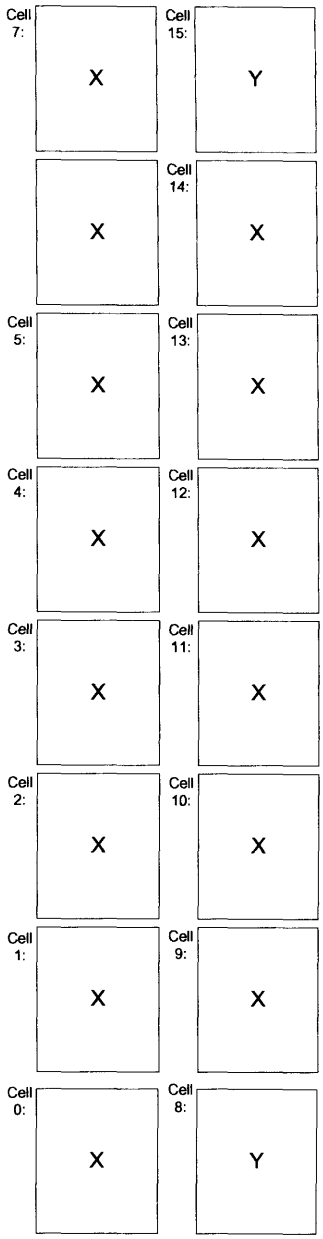


Figure 6.3.3(b): Distinct CLB types of the same circuit.

Problem Comparison

A comparison of Figures 6.3.2 and 6.2.2 suggests that there are differences in the behaviour of the developmental system between the adder problem and the parity problem. The first point to note is that the performance of the adder problem is in general lower than the parity problem. Second development's performance appears to decay much more rapidly with problem size for the adder problem than the parity problem. For these experiments problem size was measured in bits, which means different things for the two different problems. Hence a direct comparison between the graphs should not be made. To provide a direct comparison, what is needed is a measure of problem size shared by both problems. An example of such a measure is the number

of CLBs available to evolution at each problem size, as this directly represents the size of the phenotype for both problems. Figure 6.3.4 shows a plot of development's performance for both problems using this measure, with the performance data linearised using a logarithmic function as in Section 6.2, and includes linear least squares lines of best fit. Even with the correction for problem size, performance on the adder problem appears to decay more rapidly than with the parity problem.

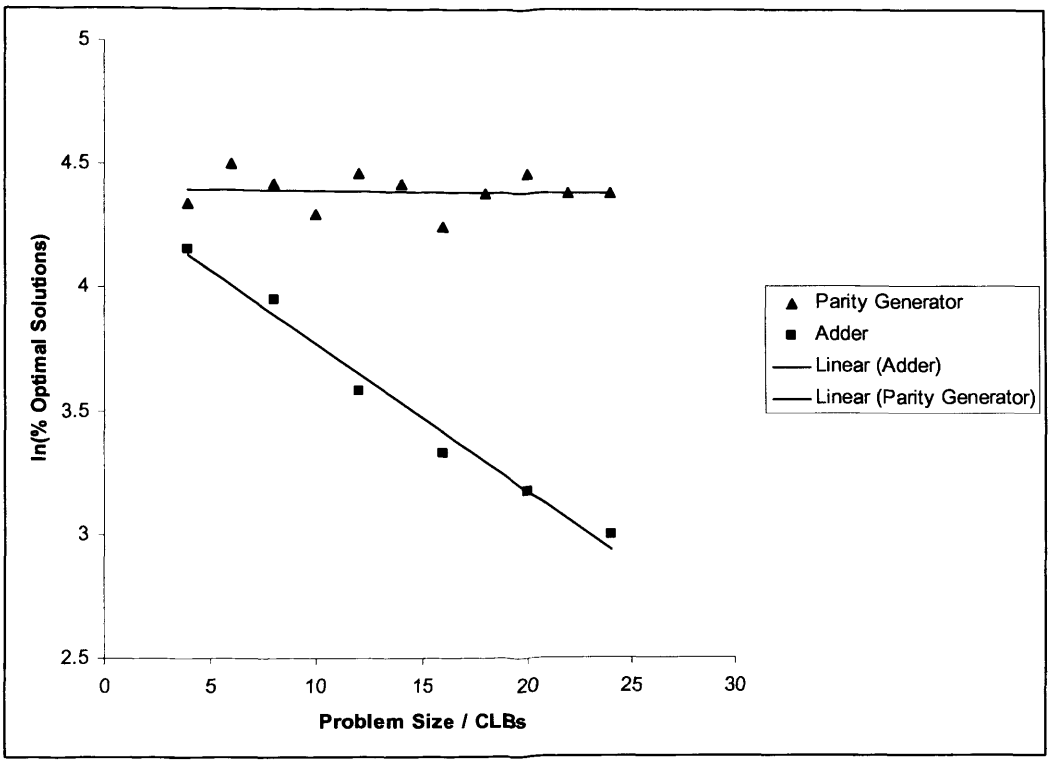


Figure 6.3.4: A plot of the percentage of optimal solutions using development for both problems, linearised by application of a logarithmic function, against problem size measured in CLBs.

It was noted earlier that the developmental system appears to be biased towards generating highly regular designs. A greater density of highly regular and optimal solutions in the parity design space might explain why the evolution tended to find solutions to the parity problem more easily than the adder problem. Such highly uniform designs are reminiscent of iterative array multiplier designs, and suggest that this technique might be useful for implementation on technologies where circuits with a high degree of uniformity are easier to synthesise than irregular designs.

In Section 6.2.3 it was noted that the optimal solutions to the adder problems that were evolved using the developmental system were not general: when the rules used to generate them were applied to larger problem sizes they no longer produced optimal solutions. It was also noted that this was unsurprising as there was no explicit bias towards generality.

Figure 6.3.5(a) shows another solution evolved for the eight bit parity problem. When the rules that generated this solution were applied to the twelve bit problem, the circuit shown in 6.3.5(b)

was generated, and this circuit is indeed an optimal solution to the larger problem. In fact the rule set that generated this circuit is general: when applied to an *n* x 2 array with the same initial conditions present in the southernmost two CLBs, an optimal solution to the parity problem is always generated.

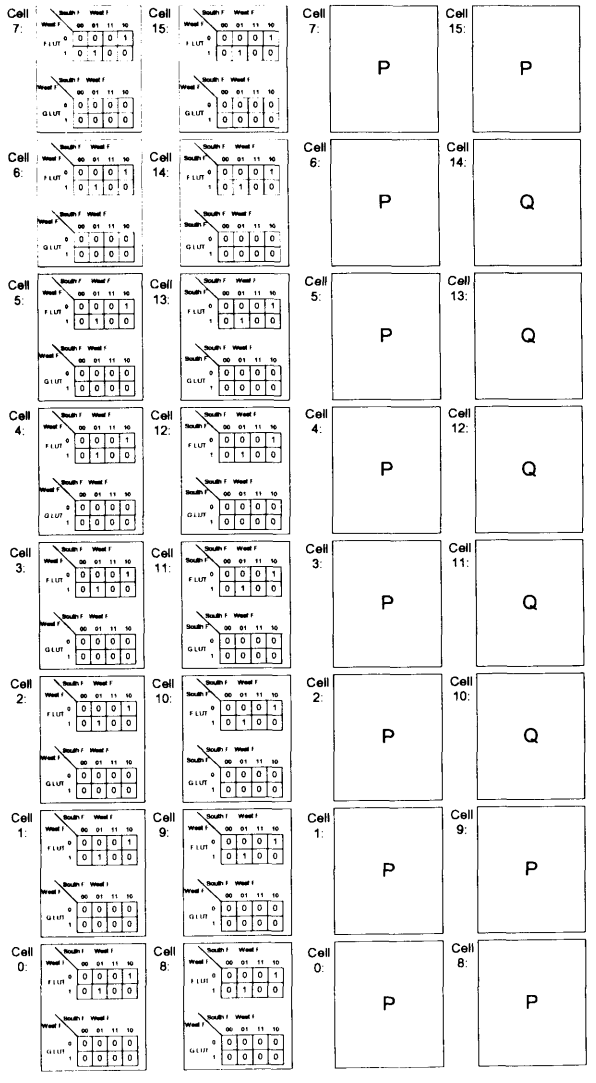


Figure 6.3.5(a): An optimal evolved solution to the eight bit parity problem, with two distinct CLB types P and Q.

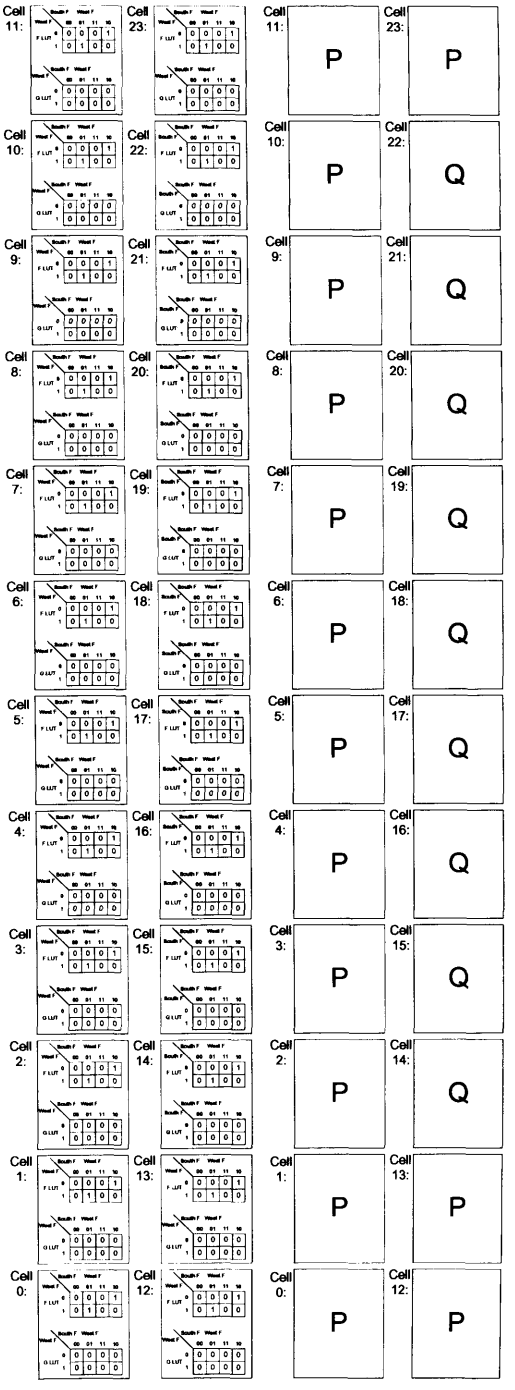


Figure 6.3.5(b): The optimal 12 bit solution generated when the same rule set is applied to the larger array. The CLB configurations P and Q are identical to those of Fig. 6.3.5(a).

The circuit shown in Figure 6.3.5(a) consists of two distinct CLB types, here marked P and Q. From south to north the array is made up of two rows of CLBs consisting only of type P, followed by five rows with one CLB of type P and one of type Q, and finally again one row of only type P. It can be seen by comparing the distinct CLB types shown in Figures 6.3.5(a) and (b) that to generate the larger parity generator development did not generate different CLB configurations. Instead it simply extended the central rows that consist of one CLB of type P and one CLB of type Q. Development achieves this using the regulatory rules to generate different contexts near the top and bottom of the array, as might be expected: as was noted above, development finds it easiest to generate inhomegeneities in protein concentrations at the edges and corners of the array. This phenomenon was apparent in many of the evolved solutions to the parity problem. Just as the presence of many highly regular and optimal solutions in the design space might explain the greater general level of performance of the developmental system when applied to the parity problem for small problem sizes, the ability of this particular developmental model to discover sets of rules that produce general solutions to the problem by extending a central uniform area of the circuit might explain why the scalability of the parity problem is greater than that of the adder problem.

6.4 Summary

This chapter has explored how the performance of the developmental model presented in Chapter 5, the Outer Totalistic model, scales to large problems. It began by demonstrating that the model is capable of generating large phenotypes using large patterns of proteins as a simple example. Following this the model was compared to similar models found in the literature and has been shown to equal or outperform these for a number of test problems, while using fewer evaluations. Additional analysis was also provided that allowed the biases imposed on evolution by the model to be characterised to some extent, and guidelines derived from the results of the experiments were provided to indicate what kinds of problems are suitable targets for the Outer Totalistic model, and how suitable initial conditions might be selected.

The model was then used to demonstrate the main hypothesis of this thesis, that the scalability of hardware evolution can be enhanced by exploiting a model of biological development. A series of experiments were carried out to compare evolution's performance evolving n-bit adder with carry circuits using both development and a naïve mapping, for a range of problem sizes. It was shown that while the performance of the naïve system was greater than the developmental system for small problem sizes (one to three bit adders) the developmental system outperformed the naïve system on all subsequent problem sizes. Several statistical analyses revealed that performance of the developmental system decayed more slowly than the performance of the

naïve system as problem size increased, i.e. that the *scalability* of the developmental system was greater than that of the naïve system, which is in line with the hypothesis of this thesis.

This thesis has suggested that development can enhance scalability by providing a mechanism for reusing design features and biasing the evolutionary search towards modular designs that take advantage of the mechanisms of reuse. Furthermore it has suggested that development provides this ability without curtailing evolution's ability to explore areas of design space that might contain innovative circuits. Analysis of the evolved adder circuits revealed a high degree of design reuse. It was also shown that while the circuits contained elements of traditional designs, they differed in a number of respects suggesting that development has not prevented evolution from exploring non-traditional areas of design space.

It was also demonstrated that biasing evolution towards a traditional decomposition of the problem led to reduced performance, suggesting that evolution benefits from the freedom to discover and apply its own design abstractions, a feature that is not present in other approaches to scalability for hardware evolution (Murakawa et al., 1996; Torresen, 2000b; Shanthi et al., 2004).

A second set of scalability experiments were then carried out for a different hardware evolution problem, the even n-bit parity problem. It was again observed both visually and through statistical analysis that scalability was enhanced when using development, and analysis of the evolved circuits revealed a high level of reuse. As the developmental model had not been designed with the parity problem in mind the results of these experiments increase the significance of the earlier scalability results, and suggest that the developmental technique might be applicable to a larger set of circuit design problems.

7 Conclusions and Further Work

This thesis has explored how the use of genotype-phenotype mappings inspired by biological development can benefit the evolution of electronic circuits. This chapter reviews the work of the thesis and its significance, and highlights areas that warrant further research. Following a brief précis of the thesis in Section 7.1, an outline of the major contribution of the thesis, the demonstration of the primary hypothesis and its supporting points, will given in Section 7.2. A number of minor contributions have also been made, and these are reviewed in Section 7.3. Plans for further work are presented in Section 7.4, including plans for providing further evidence to support the contributions of this thesis and plans to extend the work in new directions.

7.1 Thesis Précis

Chapter 1 introduced the subject area, outlined the scalability problem and how development might be of benefit, and introduced the hypothesis of this thesis. It also laid out three supporting points, and noted the minor contributions that were to be made throughout the thesis. Chapter 2 presented a comprehensive literature review covering the fields of hardware evolution and computational development. It highlighted work on scalability in hardware evolution, described in detail how development might be used to improve scalability, and made the case that a demonstration of this process applied to hardware evolution would be a useful contribution. Chapter 3 introduced the intrinsic hardware evolution platform that was used throughout the rest of this thesis, discussed the criteria used to design the platform and presented some validation experiments. Chapter 4 introduced a model of development that could be used to evolve hardware, and presented some exploratory experiments that represented the first attempts in the field to evolve electronic circuits using an implicit developmental process. Chapter 5 introduced and tested a number of changes to the developmental model designed to improve its performance. Chapter 6 focused on the main topic of this thesis, scalability. It explored the evolution of large phenotypes and presented two demonstrations of development enhancing scalability in hardware evolution.

7.2 Major Contributions

The major contribution of this thesis is the demonstration of the primary hypothesis that was introduced in Chapter 1:

The scalability of hardware evolution can be enhanced by exploiting a model of biological development.

In Chapter 6 a series of experiments were presented that compared evolution's performance evolving n-bit adder with carry circuits using both development and a naïve mapping, for a range of problem sizes. The results demonstrated clearly that while the performance of the naïve system was greater than the developmental system for small problem sizes (one to three bit adders) the developmental system outperformed the naïve system on all subsequent problem sizes. Several statistical analyses revealed that the performance of the developmental system decayed more slowly than the performance of the naïve system as problem size increased, i.e. that the *scalability* of the developmental system was greater than that of the naïve system, confirming the hypothesis.

Similar scalability experiments were also carried out for the even n-bit parity problem. It was again observed both visually and through statistical analysis that scalability was enhanced when using development. As the developmental model had not been designed with the parity problem in mind the results of these experiments increase the significance of the earlier scalability results, and suggest that the developmental technique might be applicable to a larger set of circuit design problems.

This thesis has argued that the mechanism by which development can enhance scalability relies primarily on three properties, *abstraction*, *modularity* and *reuse*: development can be used to present evolution with a more tractable abstract search space, and it can achieve this by encapsulating useful design concepts in modules that can be reused. With this mechanism in mind Chapter 1 introduced three contributions demonstrated by this thesis that highlight these properties in hardware evolution:

1. ***Abstraction***: *show that a developmental model for hardware evolution can discover and encode biases towards design abstractions that might benefit hardware evolution.*

Chapter 4 presented the first attempt to evolve circuits using biologically plausible implicit development. The problem tested was the evolution of a two bit adder with carry circuit with reduced traditional constraints. Although optimal adders were not evolved, analysis of an evolved rule set revealed that the rules responsible for generating the circuit could be separated into three groups. Two of these groups worked together to bias evolution towards exploring feedforward circuits. This constraint is widely used by traditional designers as an easy way to impose the digital design abstraction. Although it should be acknowledged that evolution might not find traditional design abstractions particularly useful, some are likely to be, as in this case

where the design problem at hand has been thoroughly studied using both traditional and evolutionary design techniques. The example used also showed that evolution was not forced to adhere to this abstraction: it was applied as a soft bias rather than a hard constraint that might prevent the exploration innovative designs. Therefore it was shown that the developmental system was capable of encoding biases towards design abstractions that are potentially very useful. Analysis of the circuits evolved in Chapter 6 supported this point further, by again identifying evolved rule sets that biased evolution towards using a feedforward design constraint when generating fully functional adder circuits.

2. ***Modularity***: *show that evolution can encapsulate such design abstractions in useful developmental modules.*

The nature of modules that are likely to be useful for evolution was discussed in Chapter 4, leading to a working definition of a useful developmental module: it should be a component of a system that is used repeatedly and a component of a system that operates largely independently of other components. It was then shown that the three sets of rules discussed in contribution 1 above had fairly general encodings thus did not interact with each other to any great extent. Hence it was shown that evolution had encapsulated the design abstraction discussed in contribution 1 in useful developmental modules. Furthermore it was noted that the two sets of modules that interacted to present a feedforward abstraction could be thought of as a single larger module, thus forming a hierarchy of modules. Both Chapter 2 and Chapter 4 argued that the creation of hierarchical layers of abstraction is likely to be a requirement of a general solution to the scalability problem, hence a useful feature of developmental modules.

3. ***Reuse***: *show that evolution can re-use such modules to solve larger hardware evolution problems.*

Chapter 5 presented the evolution of optimal two bit adders using a genotype-phenotype map modelled on biological development. Analysis of the evolved circuits presented in Chapter 6 revealed that the circuits were generated by the repeated re-use of sets of rules, thus forming a design abstraction. Similar re-use was shown in solutions to much larger problems, two seven bit adder with carry circuits and an even eight bit parity generator. While reuse has been previously been shown in developed hardware designs discovered by evolution (Koza et al., 2003), this example relied on hand-designed programming constructs to provide modularity and a thorough demonstration of how this might lead to scalability was not provided.

The clear demonstration in Chapter 6 of an enhancement in scalability when using a developmental genotype-phenotype map for two benchmark hardware evolution problems,

along with the evidence demonstrating the three supporting points laid out above clearly confirm the hypothesis of this thesis.

7.3 Minor Contributions

Chapter 1 also highlighted a number of minor contributions that were to be made throughout the thesis. These are now discussed.

4. *Demonstrate hardware evolution of the Virtex architecture at a level of abstraction that imposes the fewest design constraints that has been reported in the literature.*

5. *Show that such an approach allows more efficient use of the resources provided by the Virtex architecture than previously reported work, and may provide a more evolvable search space than those constrained to combinational circuits.*

FPGAs based on the Xilinx Virtex architecture are currently widely used as intrinsic platforms, hence results specific to the architecture are of interest to many researchers in the field. The value of low abstraction evolution has been shown using other architectures, most notably the Xilinx 6200 architecture (Thompson, 1996c; Huelsbergen et al., 1999; Raichman et al., 2003) and it has been argued throughout this thesis that allowing evolution to operate at low abstractions may allow it to explore a wider range of innovative designs. However many of the behaviours opened to evolution at low abstractions may be architecture-specific, consequently the benefits of low-abstraction evolution might not carry across to other platforms automatically. Hence a demonstration that evolution at very low abstractions using the Virtex architecture is not only possible but potentially useful would make a valuable contribution.

Chapter 3 presented two sets of experiments to validate the Virtex-based intrinsic evolutionary system that was used throughout this thesis. The second set demonstrated the intrinsic evolution of two bit adders at a level of abstraction that relaxed a wide range of design constraints commonly used in both traditional and evolutionary design: the genetic representation allowed unrestricted feedback between the elements of the evolving circuits and a clock that ensured temporal synchronisation was not imposed. Although some work by others carried out in parallel with this experiment also relaxed many traditional design constraints, it used a virtual architecture that placed considerable restrictions on the routing made available to evolution (Hollingworth et al., 2000b). Hence evolution was constrained to some extent as it was not allowed to search across the full range of behaviours possible with the Virtex architecture. Furthermore Hollingworth et al. did not make any comparisons between their abstraction and a more typical, digital abstraction.

Comparison of the results from the low-constraint evolution experiment presented in Chapter 3 with those achieved by Hollingworth et al. using a representation that restricted evolution to a search of combinational adders (Hollingworth et al., 2000a) revealed that although the low-constraint search space was much larger, performance only decreased slightly, indicating that evolution might find the design space more tractable. One potential explanation for this was that the low-constraint space of the experiment presented in Chapter 3 allowed evolution to explore circuits that used dedicated XOR gates that are useful for implementing arithmetic circuits efficiently using Manchester-like carry chains. The work of Hollingworth et al. using a more constrained search space only explored feedforward networks of LUTs, hence evolution could not make such good use of the hardware resources available. It was suggested that allowing evolution to manipulate such components might result in a more evolvable search space, at least in the case of the evolution of two bit adders and potentially other arithmetic problems.

6. *Show that the choice of context is important to the ability of induction-based models of development to evolve hardware.*

Development encapsulates a huge array of interactions between genes, their products and the environment. Of this myriad of processes it is not clear which are likely to be of the most benefit for hardware evolution and scalability in particular, and how they should be modelled. Unfortunately providing a complete answer to this question is well beyond the scope of this thesis. However identification of any features that affect evolutionary performance is likely to be of useful to the field. Processes and features modelled in developmental systems were reviewed in Chapter 2, and a number of these were used to define basic design criteria for the exploratory system introduced in Chapter 4. One of the features identified as important was the exploitation of environmental context, which allows the developmental mapping to be more expressive. Although the exploratory developmental system presented in Chapter 4 was context sensitive it used a highly simplified model of intercellular communication that limited the range of contexts that any given cell could differentiate between, in order to reduce computational complexity. However initial experiments found that optimal adders could not be evolved, and the model of intercellular communication was identified as a potentially performance-limiting factor, as little information could be transferred between cells at each timestep. Induction-based models, such as the models used throughout this thesis, rely on explicit intercellular communication to map changes in gene regulation to phenotype changes, and analysis of the pattern formation process in Experiments 1(a)-(f) of Chapter 5 confirmed that evolution struggled to generate solutions that communicated information between cells. A number of changes to the communication model were then introduced and tested. The improvements focused on increasing the number of different local contexts that development was able to generate and maintain: first the Totalistic model was introduced, followed by a model (inspired

by diffusion in physical systems) that allowed development to partly differentiate between the contribution made to a cell's context by itself and that made by neighbouring cells. This was extended to the Outer Totalistic model, which allowed development to make this differentiation concrete. Comparisons between the models were made using pattern formation problems that were simple but potentially useful for the development of circuits. The comparisons revealed that evolutionary performance was greatly affected by the changes that had been introduced. The Outer Totalistic model, which was then used throughout the rest of the thesis, was found to greatly enhance performance over the initial model used in Chapter 4. The only major difference between these systems was the way in which context was used.

7. *Show that a dynamic representation can increase the evolvability of rule-based developmental systems.*

Another feature of developmental models used in this thesis that was considered likely to limit evolvability was the Boolean nature of the rule activation and response strategies. It was noted in Chapter 4 that there was a tradeoff between the evolvability of the activation and response strategies and the cost that might be required to implement such a system in hardware in the future. It was suggested in Chapter 5 that the introduction of a rule duplication operator might bias evolution towards exploring networks of rules that contained redundant relationships, and that consequently evolvability might be enhanced, thus compensating for the use of Boolean activation and response strategies. Initial experiments suggested that the introduction of this operator improved evolutionary performance, slightly but significantly.

The experiment was repeated using an operator that inserted random rules rather than duplicating the rules already present. This is unlikely to duplicate existing relationships between proteins. Hence this tested whether the observed performance improvement was caused by duplicated rule interactions or by some other feature of the duplication model. Random insertion yielded similar results to duplication, suggesting that the improvement in performance was caused by the only other change made to the model when duplication was introduced. This was the introduction of a dynamic representation.

8. *Show that it is possible to characterise the learning biases introduced by a developmental process to the extent that classes of problem that might and might not be suitable targets for a developmental approach can be identified.*

Not only is there very little evidence in the literature as to what problems different developmental models are suited to, very little methodology has been developed to explore the issue. In Chapter 6 a number of guidelines were developed to suggest what kinds of problems

were suited to the Outer Totalistic model, through a combination of theoretical argument backed up by empirical evidence. The major conclusion was that the Outer Totalistic model of development is best suited to problems where high fitness phenotypes are symmetrical and exhibit a high degree of regularity so that relative rather than absolute positional information can be used to transmit pattern formation information across the phenotype. Although the guidelines developed are specific to the model used, the methodology used to derive them, based on the concepts of *relative positional information* and *symmetry*, could be applied to other models and could be used as the basis of an analytical toolkit for the evaluation of models of development.

9. *Show that the model of development used for the scalability experiments in this thesis performs well in comparison with similar models introduced by other researchers.*

Although the guidelines that were developed specific to the Outer Totalistic model were interesting in their own right, they say little about how the model might compare to others used in the literature. A great deal can be learned by comparing the performance of systems inspired by the same biological processes but implemented using slightly different abstractions. Such is the case for many developmental and evolutionary models in the literature. Hence it is surprising and unacceptable that comparative studies and benchmark problems are rare in both the field of hardware evolution and computational development.

Although careful comparative studies have been made throughout this thesis when introducing any modifications to the developmental model, Chapter 6 offered the opportunity to compare the Outer Totalistic model to those used by other researchers. Performance was measured using two pattern generation problems, the Norwegian Flag problem and the French Flag problem. The Outer Totalistic model was shown to exhibit comparable and in some instances better performance than the models it was compared against. Hence it was suggested that the Outer Totalistic model is a good choice for problems of this type (symmetrical problems that could be described using relative positional information) although it was acknowledged that some features of the other models might be useful for other classes of more complex problems and implementation in hardware.

10. *Show that providing a bias towards traditional problem decomposition, as proposed by other approaches to scalability, can actually harm scalability.*

Scalability is a fundamental problem for hardware evolution. Hence it is unsurprising that a number of approaches have been suggested to ease the problem. Although it is likely that alternative approaches will have merits particular to certain classes of problem, and that there

may be benefits to combining different approaches to scalability, it is important for the field to be aware of both the merits and disadvantages of all the alternatives. A common alternative to the developmental approach is to use some heuristic to partition the problem into sub-problems and then apply evolution to learn each sub-problem either independently or incrementally. It has been argued throughout this thesis that partitioning a problem in this way might reduce the opportunity for innovative design, as it limits the interactions between sub-problems.

Although it is the view of the author that innovation is the most valuable benefit of hardware evolution, it is recognised that other benefits such as automatic design, adaptability or evolution's ability to handle poorly specified problems might be central to many real-world applications. In such cases this argument against partitioning techniques (and other hard-bias approaches such as Function Level Evolution) might not seem relevant.

To demonstrate the importance of allowing evolution to learn its own biases, the adder scalability experiments of Chapter 6 were repeated using a partitioned, incremental approach. The problem was partitioned into sub-problems by circuit output, a common method of problem decomposition in traditional hardware design that has been used widely by proponents of partition-oriented approaches in hardware evolution. The fitness function was replaced with an incremental version that biased evolution towards initially exploring solutions for a single sub-problem (the least significant output) and incrementally exploring a larger subset of the problem. The incremental experiments clearly exhibited an increased loss of performance as problem size increased, showing that evolution benefits from the freedom to discover and apply its own design abstractions, and that traditional biases applied through a partition-oriented approach do not just limit innovation, but can actually *harm* scalability. Such an insight is likely to be of interest for any researchers concerned with using evolution to generate solutions to large, real-world problems, whether innovative designs are of interest or not.

7.4 Further Work

Research into scalability and developmental hardware evolution is in its infancy, and there is great opportunity for significant research. Many substantial questions remain and are likely to require long-term study. Such issues are discussed in Section 7.4.2. However there are also a number of areas where the contributions of this thesis could be strengthened in the short term. These issues are discussed in the following section.

7.4.1 Short-term Issues

The most important issue arising from the work presented in this thesis is that scalability in hardware evolution has only been demonstrated in very specific circumstances: only two

benchmark demonstrations, the evolution of adders and parity generators, were presented. Such circuits have a great deal in common hence the demonstrations only sample a small area of the space of problems for which development might be useful. Adders and parity generators are arithmetic circuits and their design spaces are extremely well known in the digital domain. Circuits of this class were chosen intentionally for this first demonstration of scalability, as it was known that optimal solutions existed in the design space and that they could easily be described in a modular fashion, hence there was a good opportunity for the benefits of development to be highlighted by their solution. However there is a wide range of real-world problems for which little domain knowledge exists, and to which development might be better suited. Demonstrating the advantages of scalability across a wider and more suitable range of problem classes is therefore a priority for future work.

Target Problems

A good starting point might be to explore problems for which it is known that non-traditional solutions exist. In (Thompson, 1996c) and (Thompson et al., 1999) Thompson explored the evolution of circuits that discriminated between two frequencies, and discovered highly innovative circuits that lay in areas of search space not searched by traditional design techniques. This problem has also been explored by Raichman et al. (Raichman et al., 2003) and Harding and Miller (Harding and Miller, 2004), and is particularly suitable for a demonstration of scalability not only due to the argument above but also because it can easily be scaled to a larger problem by introducing a greater range of frequencies to be discriminated.

One interesting feature of Thompson's experiment that might bias evolution towards exploring non-traditional circuits was the fitness function. Fitness was measured in a continuous manner by calculating the difference in the output voltage between periods when each frequency was applied, rather than the digital manner used in the experiments of this thesis. This approach could be applied to a range of more common problems, including the evolution of analogue arithmetic circuits, to bias evolution towards exploring circuits with greater reliance on the analogue behaviour of the components.

However there is a need for the community to begin exploring more useful, real-world problems to which hardware evolution might be suited, such as adaptive or poorly specified problems. Problems of this class that might also demonstrate the power of development are those that have been already used to explore other approaches to scalability. For instance the classification of myoelectric signals has been explored using both function level evolution and a partitioned approach (Torresen, 2001), as have various image recognition and filtering problems (Murakawa et al., 1996; Torresen, 2000b; Sekanina, 2002a). Applying development to these kinds of problems might not only demonstrate the benefits it can bring over more constrained

approaches to scalability but also begin to carve a niche for the use of development in the real world.

Other problems to which hardware evolution seems particularly suited are those with multiple functional objectives, for instance the design of circuits with built-in self test functions (Garvie and Thompson, 2003) or polymorphic circuits (Stoica et al., 2002). The traditional approach to such problems is to decompose them according to function, at the expense of efficiency. Evolution has already been shown to be capable of designing efficient solutions for small problems of this type, and by using development might be able to outperform traditional designers for similar real-world problems.

More General Environments

Not only were problems used in this thesis highly specific, the positions of the inputs and the outputs and the geometry of the array were very carefully chosen in the case of the adder problem, thus introducing a good deal of domain knowledge. Although this was not the case for the parity problem, no further investigation of other input and output positions and array geometries was conducted. This means that it is still not clear if evolution is capable of surmounting such issues. Furthermore Chapter 5 showed that performance could be greatly affected when such parameters were altered, and this is in line with previous work in hardware evolution (Miller and Thomson, 1998a; Miller and Thomson, 1998b). A more robust demonstration of scalability that explores a wide range of interesting problems should also perturb such parameters more effectively.

A related issue is that of the effect of environmental conditions on low abstraction evolution. Thompson and Layzell have shown that the performance of circuits evolved when traditional constraints have been relaxed can be affected by conditions such as the incident temperature, power supply, and levels of electromagnetic interference (Thompson and Layzell, 2000). Such conditions are unlikely to have affected the results of the hardware evolution experiments presented in this thesis as the solutions found adhered to the digital design paradigm. However in many of the experiments in this thesis evolution was observed exploring circuits beyond the digital design space, hence a more thorough study of scalability carried out at low abstractions using problems where evolution might make more use of its ability to search design space beyond the scope of traditional design should take these issues into consideration to ensure that any observed trends are not due to gradual changes in environmental conditions.

Improving Development

Before a study exploring the issues outlined above is carried out it would be necessary to improve the developmental model. Although this issue was examined in Chapter 5, it is clear that evolution using the Outer Totalistic model is still fairly limited: the probability of

discovering a seven bit adder in Chapter 6 was very low, suggesting that more complex circuits that rely less on domain knowledge are likely to be difficult to evolve. Furthermore although the model compared favourably to others on large pattern generation problems, optimal solutions could not always be evolved.

Further improvement of the model has two potentially opposing requirements: the model should exhibit good evolvability yet be computationally inexpensive. While this thesis has discussed the potential benefits of more complex models of regulation and communication, the models that have been used are extremely simple. One argument for the use of simple communication models was the observation that many communication methods exist in nature, and that some features of these might have arisen to solve problems unique to the biological medium. However recent work suggests that some of the potential advantages of more complex regulatory models discussed in Chapter 2 might be real (Kumar, 2004). Also modes of communication based on simple diffusion models might potentially ease the motif scaling problem explored in Chapter 6 by removing the reliance on relative positional information. Comparative studies of various models of communication and regulation are vital to understand what features are of most use to hardware evolution. Chapter 6 presented a simple comparison between the Outer Totalistic model used in this thesis and three other developmental models found in the literature, but few concrete conclusions could be drawn as the models differed in so many respects. A larger and more thorough investigation of communication and regulation strategies, and how they affect the evolution of circuits would be of great benefit, and would be a sensible direction for research in the short term. Such as study could also be extended to include the identification of a varied set of benchmark problems (including problems that might benefit from low abstraction evolution) that could be used for future comparisons.

Unfortunately the implementation of more complex models of regulation and communication is likely to require greater computational resources. However it might be possible to engineer solutions that do not require complex communication and rule strategies. An alternative that was briefly explored in this thesis is to retain Boolean strategies and use additional biases, such as rule duplication, to improve evolvability. While the introduction of a duplication operator appeared to improve performance it was shown in Chapter 5 that the performance benefit did not derive directly from the redundancies that duplication introduced to the networks of interacting rules. Furthermore the implementation of a duplication operator involved a wide range of changes to the evolutionary system, including the introduction of a variable length chromosome. A thorough study of the effects of each of these changes could not be conducted within the limited scope of the thesis, but this area is certainly worthy of further study.

In addition to rule duplication, biological evolution and development encapsulate a huge range of other processes that may benefit evolvability and may be significant for the hardware evolution. Studying such processes to discern whether they might be applied usefully to hardware evolution, or whether they are used by nature because of some feature of the biological medium would be valuable. An example of such a feature is the interaction between development and the environment in a way that allows development can take advantage of the natural complexity of physical processes without an additional computational cost. For instance cell division requires the formation of a membrane between the two compartments of the dividing cell. This process is not encoded genetically or even by development, but arises naturally as a consequence of the growing cell's surface tension. Similarly during mitosis chromosomes are organised before division by a process of segregation that relies on the growth of microtubules (Kirschner and Gerhart, 1998). Their growth is not directed by genetic or developmental means, but behaves purely as a random walk. If the specific connection is not made after a certain time the microtubule depolymerises and the process begins again. This can be viewed as a simple supervised learning mechanism that is hybridised with developmental learning through interaction with the environment, and might improve evolvability in a similar way to the Baldwin effect (Baldwin, 1896; Hinton and Nowlan, 1987). In fact Waddington proposed that a similar mechanism might exist by which interactions between a developmental system and the environment can become genetically fixed (Waddington, 1942).

However this leaves the question of what processes might be useful to incorporate and how they could be harnessed efficiently. An obvious method of incorporating such information to the developmental system is to introduce it as context: the developmental rules could be modified to include terms relating to environmental factors. Researchers have already begun to explore this idea. The developmental models used by Miller (Miller and Thomson, 2003; Miller, 2004) and Tufte and Haddow (Tufte and Haddow, 2003b) use information about the function of neighbouring cells as context, and Tyrrell et al. have suggested a framework for directly incorporating information from the external environment (Tyrrell et al., 2003). In these cases the choice of structures or signals to include as context are likely to be biased towards aspects of designs that are important to traditional design abstractions such as the logical function of neighbouring cells, and hardware implementation is likely to be costly. However another form of potentially useful environmental information is fitness. In the case of an intrinsically evaluated circuit, fitness can be viewed as a function of the true physical interaction between a circuit and its environment (Thompson, 1996c), hence by providing fitness as environmental context development may be able to interact with, and benefit from, the complexity arising from the environmental physics.

Should biologically-inspired mechanisms to improve evolvability prove fruitful, models of development that can be implemented efficiently in hardware, such as that presented by Miller (Miller, 2004) might yield evolvable developmental systems that can enhance scalability for a wide range of problems.

7.4.2 Long-Term Issues

The issues discussed in this section do not arise directly as a consequence of the work in this thesis, but are issues that stand in the way of routine use of evolutionary approaches in hardware design, hence might hamper the relevance of the work in the long term.

The first of these has been touched upon in this thesis but not explored in detail. It is the relationship between large and complex problems. It has been stressed that the problems under study are large, but not necessarily particularly complex, although the terms have not been defined particularly carefully. Achieving such a definition, and determining whether approaches to scalability can benefit the evolution of complex problems in addition to generating large phenotypes is an important but difficult issue.

Chapter 2 discussed work that has begun to explore the relationship between phenotype size and complexity, but is at an early stage. The issue of complexity is so difficult but potentially important that it has developed into a field in its own right, which might be a good place to start to look for answers to this problem.

It is likely that the evolution of truly complex circuits will not come easily. Although mechanisms to improve the evolvability of developmental systems that have been discussed above might help, there are other issues that are particular to the evolution of circuits. The most apparent is that current genetic circuit representations, developmental ones included, all suffer from the nonlinear relationship between changes in a circuit design and their effect on fitness. This gave rise to the credit allocation problem discussed in Chapter 5, which was circumvented rather than solved so that the goals of this thesis could be achieved. The successful evolution of large, complex circuits is likely to require a solution to this problem. One solution might be to look to mechanisms of credit allocation used elsewhere in the machine learning community, such as the Bucket Brigade algorithm used by Learning Classifier Systems (Goldberg, 1989). However as the credit allocation problem is a symptom of the larger problem of a poor representation, it may be more fruitful to consider entirely new ways to represent circuits.

Another major issue that is directly related to scalability is the problem of generalisation, in particular generalisation to unseen inputs. The greatest limitation to the size of the circuits presented in this thesis was the time required for their evaluation. In the case of the seven bit

adders each candidate circuit was evaluated using a truth table with 2^{15} entries. Additionally, each circuit was evaluated five times. With a maximum of 2500 generations, each run of the adder experiment required that up to $4x10^9$ input cases were tested. For each additional input this number increases by a factor of four. For a circuit with a large number of inputs, let alone a large number of internal states that each input case must be tested against, exhaustive testing is not a realistic prospect, even if the evolutionary process itself is capable of scaling to such large problems. This problem is further exacerbated when evolving circuits at low abstractions, where the traditional constraints used to guarantee good generalisation are not available, or when circuits are expected to operate in extreme conditions when such guarantees might not hold. In such cases additional test cases in a range of environmental conditions are likely to be required.

Preliminary work on generalisation to unseen input cases has suggested that evolved arithmetic circuits do not generalise well to unseen test cases (Miller and Thomson, 1998b; Imamura et al., 2000). However it has been shown that other problems with test sets that include more redundant data generalise reasonably well (Miller and Thomson, 1998c; Sekanina, 2002a; Torresen, 2002c). As the explosion of test set size might present a hard limit to scalability for at least some problems, it is vital that the field understands the issues behind, and the consequences of generalisation, so that classes of problems likely to be fruitful for hardware evolution can be determined.

7.5 Summary

As stated in Chapter 1 the motivation for this work was two-fold. Primarily it aimed to show that the introduction of a developmental genotype-phenotype map can allow evolution to design larger electronic circuits automatically. This has been shown clearly through the demonstration of the primary hypothesis and its three supporting contributions. Furthermore it aimed to show that development can achieve this while allowing evolution to explore innovative and evolvable design features found at the lowest design abstractions. This too has been demonstrated, along with a number of other minor contributions.

A number of ways to strengthen the argument laid out in this thesis have been discussed above, divided into short-tem issues that are directly related to the experiments conducted here and could form the basis of a follow-on study. A second set of more difficult and wide range issues was also discussed. Chapter 1 expressed the hope that one day the size and complexity of evolved circuits might rival those generated by conventional design techniques. If the issues outlined above can be tackled as successfully as the questions explored in this thesis, then such a possibility is indeed real.

References

1. Aggarwal, V. (2003) 'Evolving Sinusoidal Oscillators Using Genetic Algorithms', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 67-76.
2. Alpha Data Parallel Systems Ltd. (1999) *ADM-XRC User Manual.* Available from: http://www.alphadata.co.uk/pdf/adm-xrc_usermanual_v1_2.pdf.
3. Altenberg, L. (1994) 'The Evolution of Evolvability in Genetic Programming'. In Kinnear, K. E. (Ed.) *Advances in Genetic Programming,* MIT Press, Cambridge, MA, U.S.A., pp. 47-74.
4. Altenberg, L. (1995) 'The Schema Theorem and Price's Theorem'. In Whitley, D. and Vose, M. D. (Eds.), *Foundations of Genetic Algorithms 3,* Morgan Kaufmann, San Mateo, CA, U.S.A., pp. 23-49.
5. Andersen, P. (1998) *Evolvable Hardware: Artificial Evolution of Hardware Circuits in Simulation and Reality,* University of Aarhus, Aarhus, Denmark. M.Sc. Thesis.
6. Araujo, S. G., Mesquita, A. and Pedroze, A. C. P. (2003) 'Using Genetic Programming and High Level Synthesis to Design Optimized Datapath', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, March 17-20, 2003. Springer-Verlag, Berlin, Germany, pp. 434-445.
7. Baldwin, M. J. (1896) 'A New Factor in Evolution', *The American Naturalist,* (30), pp. 441-451, 536-553.
8. Belew, R. K. and Kammeyer, T. E. (1993) 'Evolving Aesthetic Sorting Networks Using Developmental Grammars', In: *Proceedings of the Fifth International Conference on Genetic Algorithms,* San Mateo, CA, U.S.A., July 17-21, 1993. Morgan Kaufmann, New York, NY, U.S.A., p. 629.
9. Benenti, G., Casati, G. and Strini, G. (2004) *Principles of Quantum Computation and Information: Basic Concepts,* World Scientific Publishing, Singapore.
10. Bentley, P. J. (Ed.) (1999) *Evolutionary Design by Computers,* Morgan Kaufmann, San Fransisco, CA, U.S.A.
11. Bentley, P. J. and Kumar, S. (1999) 'Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem.' In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference,* Orlando, FL, U.S.A., 13-17 July, 1999. Morgan Kaufmann, San Francisco, CA, U.S.A., pp. 35-43.
12. Bentley, P. J. (2003) 'Evolving Fractal Proteins', In: *Proceedings of the Fifth International Conference on Evolutionary Systems,* Trondheim, Norway, March 17-20, 2003. Springer-Verlag, Berlin, Germany, pp. 81-92.
13. Bentley, P. J. (2004a) 'Fractal Proteins', *Genetic Programming and Evolvable Machines,* 5 (1), pp. 71-101.
14. Bentley, P. J. (2004b) 'Adaptive Fractal Gene Regulatory Networks for Robot Control', In: *WORLDS Workshop, 2004 Genetic and Evolutionary Computation Conference,* Seattle, WA, U.S.A.2004. Springer-Verlag, Berlin, Germany.
15. Biles, J. A. (1994) 'Genjam: A Genetic Algorithm for Generating Jazz Solos', In: *Proceedings of the 1994 International Computer Music Conference,* San Francisco, CA, U.S.A., September, 1994. International Computer Music Association.

16. Boers, E. J. W. and Kuiper, H. (1992) *Biological Metaphors and the Design of Modular Artificial Neural Networks,* Leiden University, Leiden, Holland. Masters Thesis.

17. Bongard, J. C. and Paul, C. (2000) 'Investigating Morphological Symmetry and Locomotive Efficiency Using Virtual Embodied Evolution', In: *Proceedings of the Sixth International Conference on the Simulation of Adaptive Behaviour,* Paris, France, 11-16 September, 2000. MIT Press, Cambridge, MA, U.S.A., pp. 420-429.

18. Bongard, J. C. (2002) 'Evolving Modular Genetic Regulatory Networks', In: *Proceedings of the 2002 Congress on Evolutionary Computation,* Honolulu, HI, U.S.A., May 17-21, 2002. IEEE, Piscataway, NJ, U.S.A., pp. 1872-1877.

19. Borkar, S. (2000) 'Obeying Moore's Law Beyond 0.18 Micron', In: *Proceedings of the 13th Annual IEEE International ASIC/SOC Conference,* Washington DC, U.S.A., 13-16 September, 2000. IEEE Press, Piscataway, NJ, U.S.A., pp. 26-31.

20. Bradley, D. W. and Tyrrell, A. M. (2001) 'The Architecture for a Hardware Immune System', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 193-200.

21. Cangelosi, A., Parisi, D. and Nolfi, S. (1994) 'Cell Division and Migration in a 'Genotype' for Neural Networks', *Network: Computation in Neural Systems,* 5 (4), pp. 497-515.

22. Cangelosi, A. and Elman, J. L. (1995) *Gene Regulation and Biological Development in Neural Networks: An Exploratory Model,* Technical Report, Center for Research in Language, University of California at San Diego, San Diego.

23. Cangelosi, A. (1999) 'Heterochrony and Adaptation in Developing Neural Networks', In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference,* Orlando, FL, U.S.A., 13-17 July, 1999. Morgan Kaufmann, San Francisco, CA, U.S.A., pp. 1241-1248.

24. Canham, R. and Tyrrell, A. (2003) 'An Embryonic Array with Improved Efficiency and Fault Tolerance', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 265-272.

25. Canham, R. O. and Tyrrell, A. M. (2002) 'Evolved Fault Tolerance in Evolvable Hardware', In: *Proceedings of the 2002 World Congress on Computational Intelligence,* Honolulu, HI, U.S.A., 12-17 May, 2002. IEEE, Piscataway, NJ, U.S.A., pp. 1267-1271.

26. Coello, C. A. C., Christiansen, A. D. and Aguirre, A. H. (2001) 'Towards Automated Evolutionary Design of Combinational Circuits', *Computers & Electrical Engineering,* 27 (1), pp. 1-28.

27. Cooper, C., Howard, D. and Tyrrell, A. (2004) 'Using Gas to Create a Waveguide Model of the Oral Vocal Tract', In: *Proceedings of the Applications of Evolutionary Computing: EvoWorkshops 2004,* Coimbra, Portugal, April 5-7, 2004. Springer, Berlin, Germany, pp. 280 - 288.

28. Cotton, F. A. (1990) *Chemical Applications of Group Theory,* (2nd Edn), John Wiley & Sons Inc., New York.

29. Damiani, E., Liberali, V. and Tettamanzi, A. G. B. (2000) 'Dynamic Optimisation of Non-Linear Feed-Forward Circuits', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 41-50.

30. Dasgupta, D. (1996) 'Using Immunological Principles in Anomaly Detection', In: *Proceedings of the Artificial Neural Networks in Engineering Conference,* St. Louis, MO, U.S.A., 10-13 November, 1996. ASME Press, New York, NY, U.S.A.

31. Dawkins, R. (1989) 'The Evolution of Evolvability', In: *Artificial Life: Proceedings of an Interdisciplinary Workshop on the Synthsis and Simulation of Living Systems,* Los Alamos,

NM, U.S.A., 22 September, 1989. Addison-Wesley, Redwood City, CA, U.S.A., pp. 201-220.

32. de Garis, H. (1993) 'Evolvable Hardware: The Genetic Programming of Darwin Machines', In: *Proceedings of the First International Conference on Artificial Neural Networks and Genetic Algorithms,* Innsbruck, Austria, 14-16 April, 1993. Springer Verlag, Berlin, Germany, pp. 441-449.

33. de Garis, H. (1994) 'An Artificial Brain: ATR's CAM-Brain Project Aims to Build/Evolve an Artificial Brain with a Million Neural Net Modules inside a Trillion Cell Cellular Automata Machine', *New Generation Computing,* 12 (2), pp. 215-221.

34. de Garis, H., Kang, L. S., He, Q. M., Pan, Z. J., Ootani, M. and Ronald, E. (1997) 'Million Module Neural Systems Evolution - the Next Step in ATR's Billion Neuron Artificial Brain ("CAM-Brain") Project', In: *Proceedings of the Third European Conference on Artificial Evolution,* Nimes, France, 22-24 October, 1997. Springer-Verlag, Berlin, Germany, pp. 335-347.

35. Dellaert, F. (1995) *Toward a Biologically Defensible Model of Development,* M.Sc. Dissertation Case Western Reserve University, Cleveland, Ohio.

36. Dellaert, F. and Beer, R. D. (1996) 'A Developmental Model for the Evolution of Complete Autonomous Agents', In: *Proceedings of the Fourth International Conference on Simulation of Adaptive Behaviour,* Cambridge, MA, U.S.A., 9-13 September, 1996. MIT Press, Cambridge, MA, U.S.A., pp. 393-401.

37. do Amaral, J. F. M., do Amaral, J. L. M., Santini, C. C., Tanscheit, R., Vellasco, M. M. R. and Pacheco, M. A. C. (2002) 'Towards Evolvable Analog Fuzzy Logic Controllers', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 123-128.

38. Dorigo, M., Maniezzo, V. and Colorni, A. (1991) *The Ant System: An Autocatalytic Optimizing Process,* Technical Report 91-016 (Revised), Politecnico di Milano, Italy, Milan, Italy.

39. Drysdale, R. A., Crosby, M. A. and the Flybase Consortium (2005) 'Flybase: Genes and Gene Models.' *Nucleic Acids Research,* 33, pp. D390-D395. http://flybase.org.

40. Edwards, R. T. (2002) 'Circuit Morphologies and Ontogenies', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 251-260.

41. Eggenberger, P. (1996) 'Cell Interactions as a Control Tool of Developmental Processes for Evolutionary Robotics', In: *Fourth International Conference on Simulation of Adaptive Behaviour,* Cambridge, MA, U.S.A, 9-13 September, 1996. MIT Press, Cambridge, MA, U.S.A., pp. 440-448.

42. Eggenberger, P. (1997a) 'Creation of Neural Networks Based on Developmental and Evolutonary Principles', In: *Seventh International Conference on Artificial Neural Networks,* Lausanne, Switzerland, 8-10 October, 1997. Springer, Berlin, pp. 337-342.

43. Eggenberger, P. (1997b) 'Evolving Morphologies of Simulated 3d Organisms Based on Differential Gene Expression', In: *Fourth International Conference on Artificial Life,* Cambridge, MA, U.S.A., July 28-31, 1997. MIT Press, Cambridge, MA, U.S.A.

44. Eggenberger, P. and Dravid, R. (1999) 'An Evolutionary Approach to Pattern Formation Mechanisms on Lepidopteran Wings', In: *Proceedings of the 1999 Congress on Evolutionary Computation,* Washington DC, U.S.A., 6-9 July, 1999. IEEE, Piscataway, NJ, U.S.A., p. 473.

45. Eggenberger, P. (2000) 'Evolving Neural Network Structures Using Axonal Growth Mechanisms', In: *Proceedings of IEEE INNS ENNS International Joint Conference on*

Neural Networks, Como, Italy, 24-27 July, 2000. IEEE Computer Society , Los Alamitos, CA, U.S.A., pp. 6/591-595.

46. Eggenberger, P. (2003) 'Exploring Regenerative Mechanisms Found in Flatworms by Artificial Evolutionary Techniques Using Genetic Regulatory Networks', In: *Proceedings of the 2003 Congress on Evolutionary Computation,* Canberra, Australia2003. IEEE, Piscataway, NJ, U.S.A., pp. 2026-2033.

47. Eriksson, J., Torres, O., Mitchell, A., Tucker, G., Lindsay, K., Halliday, D., Rosenberg, J., Moreno, J. M. and Villa, A. (2003) 'Spiking Neural Networks for Reconfigurable Poetic Tissue', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 165-173.

48. Federici, D. (2004) 'Using Embryonic Stages to Increase the Evolvability of Development', In: *WORLDS Workshop, 2004 Genetic and Evolutionary Computation Conference,* Seattle, WA, U.S.A., 26-30 July, 2004. Springer-Verlag, Berlin, Germany.

49. Ferguson, I., Stoica, A., Keymeulen, D., Zebulum, R. and Duong, V. (2002) 'An Evolvable Hardware Platform Based on DSP and FPTA', In: *Proceedings of the 2002 Genetic and Evolutionary Computation Conference,* New York, NY, U.S.A., July 9-13, 2002. AAAI Press, Menlo Park, CA, U.S.A., pp. 145-152.

50. Fleischer, K. and Barr, A. H. (1993) 'A Simulation Testbed for the Study of Multicellular Development: Multiple Mechanisms of Morphogenesis'. In Langton, C. G. (Ed.) *Artificial Life Iii,* Addison-Wesley, Santa Fe, pp. 389-416.

51. Flockton, S. J. and Sheehan, K. (1999) 'A System for Intrinsic Evolution of Linear and Non-Linear Filters', In: *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 93-100.

52. Fogarty, T., Miller, J. F. and Thomson, P. (1998) 'Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs'. In Chawdhry, P. K., Roy, R. and Pant, R. K. (Eds.), *Soft Computing in Engineering Design and Manufacturing,* Springer-Verlag, Berlin, Germany, London, pp. 299-305.

53. Fortey, R. (1999) *Life: A Natural History of the First Four Billion Years of Life on Earth,* Vintage Books USA.

54. Fukunaga, A. and Stechert, A. (1998) 'Evolving Nonlinear Predictive Models for Lossless Image Compression with Genetic Programming', In: *Third Annual Genetic Programming Conference,* Madison, WI, U.S.A.1998. pp. 95-102.

55. Gallagher, J. C. (2003) 'The Once and Future Analog Alternative: Evolvable Hardware and Analog Computation', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 43-49.

56. Garvie, M. and Thompson, A. (2003) 'Evolution of Self-Diagnosing Hardware', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 238-248.

57. Gerez, S. H. (1999) *Algorithms for VLSI Design Automation,* John Wiley & Sons Ltd., Chichester.

58. Gierer, A. and Meinhardt, H. (1972) 'A Theory of Biological Pattern Formation', *Kybernetik,* 12, pp. 30-39.

59. Göckel, N., Drechsler, R. and Becker, B. (1997) 'A Multi-Layer Detailed Routing Approach Based on Evolutionary Algorithms', In: *Proceedings of the IEEE International Conference on Evolutionary Computation,* Indianapolis, IN, U.S.A1997. pp. 557-562.

60. Goldberg, D. E. (1989) *Genetic Algorithms in Search, Optimization, and Machine Learning,* Addison-Wesley, Reading, MA, U.S.A.

61. Gordon, D. F. and des Jardins, M. (1995) 'Evaluation and Selection of Biases in Machine Learning', *Machine Learning Journal,* (20), pp. 5-22.

62. Gordon, T. G. W. and Bentley, P. J. (2002) 'Towards Development in Evolvable Hardware', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 241-250.

63. Gordon, T. G. W. (2003) 'Exploring Models of Development for Evolutionary Circuit Design', In: *Proceedings of the 2003 Congress on Evolutionary Computation,* Canberra, Australia, 8-12 December, 2003. IEEE, Piscataway, NJ, U.S.A., pp. 2050-2057.

64. Gordon, T. G. W. and Bentley, P. J. (2005a) 'Bias and Scalability in Evolutionary Development', In: *Proceedings of the 2005 Genetic and Evolutionary Computation Conference,* Washington D.C., U.S.A., June 25-29, 2005. ACM Press, New York, NY, U.S.A., pp. 83-90.

65. Gordon, T. G. W. and Bentley, P. J. (2005b) 'Development Brings Scalability to Hardware Evolution', In: *Proceedings of the 2005 NASA / DoD Conference on Evolvable Hardware,* Washington D.C., U.S.A., 29 June - 1 July, 2005. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 272-279

66. Greenwood, G. W. and Song, X. (2002) 'How to Evolve Safe Control Strategies', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 129-130.

67. Grimbleby, J. B. (2000) 'Automatic Analogue Circuit Synthesis Using Genetic Algorithms', *IEE Proceedings on Circuits Devices and Systems,* 147 (6), pp. 319-323.

68. Gruau, F. (1992) *Cellular Encoding of Genetic Neural Network,* Technical Report 92.21, Laboratoire de lInformatique pour le Parallelisme, Ecole Normale Superieure de Lyon, Lyon.

69. Gruau, F. (1995) 'Artificial Cellular Development in Optimization and Compilation', In: *Proceedings of Towards Evolvable Hardware: An International Workshop on Evolvable Systems,* Lausanne, Switzerland, 2-3 October, 1995. Springer-Verlag, Berlin, Germany, pp. 48-75.

70. Guccione, S. A., Levi, D. and Sundararajan, P. (1999) 'Jbits: Java Based Interface for Reconfigurable Computing', In: *Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference,* Laurel, MD, U.S.A.1999. Digital Engineering Institute.

71. Guo, X., Stoica, A., Zebulum, R. and Keymeulen, D. (2003) 'Development of Consistent Equivalent Models by Mixed-Mode Search', In: *Proceedings of the IASTED International Conference on Modeling and Simulation,* Palm Springs, California, U.S.A., February 24-26, 2003. IASTED/ACTA Press, Anaheim, CA, U.S.A.

72. Gwaltney, D. A. and Ferguson, M. I. (2003) 'Intrinsic Hardware Evolution for the Design and Reconfiguration of Analog Speed Controller for a Dc Motor', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 81-90.

73. Haddow, P. C. and Tufte, G. (2001) 'Bridging the Genotype-Phenotype Mapping for Digital FPGAs', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 109 -115.

74. Haddow, P. C., Tufte, G. and van Remortel, P. (2001) 'Shrinking the Genotype: L-Systems for EHW?' In: *Proceedings of the Fourth International Conference on Evolvable Systems,* Tokyo, Japan, 3-5 October, 2001. Springer-Verlag, Berlin, Germany, pp. 128-129.

75. Haddow, P. C. and van-Remortel, P. (2001) 'From Here to There : Future Robust EHW Technologies for Large Digital Designs', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12 14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 232-239.

76. Harding, S. L. and Miller, J. F. (2004) 'A Tone Discriminator in Liquid Crystal', In: *Proceedings of the Congress on Evolutionary Computation,* Portland, OR, U.S.A.2004. IEEE, Piscataway, NJ, U.S.A., pp. 1800-1807.

77. Hartmann, M., Haddow, P. and Eskelund, F. (2002) 'Evolving Robust Digital Designs', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 36-45.

78. Hartmann, M., Lehre, P. K. and Haddow, P. (2005) 'Evolved Digital Circuits and Genome Complexity', In: *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware,* Washington D.C., U.S.A., 29 June-1 July, 2005. IEEE Computer Society, Los Alamitos, CA. U.S.A., pp. 79-86.

79. Harvey, I. (1991) 'Species Adaptation Genetic Algorithms: The Basis for a Continuing Saga', In: *Proceedings of the First European Conference on Artificial Life,* Paris, France, 11-13 December, 1991. MIT Press (Bradford Books). pp. 346-354.

80. Harvey, I. and Thompson, A. (1997) 'Through the Labyrinth, Evolution Finds a Way: A Silicon Ridge', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1997. Springer-Verlag, London, U.K., pp. 406-422.

81. Hayworth, K. (1998) 'The "Modeling Clay" Approach to Bio-Inspired Electronic Hardware', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 248-255.

82. Hemmi, H., Mizoguchi, J. and Shimohara, K. (1994) 'Development and Evolution of Hardware Behaviours', In: *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems,* Cambridge, MA, U.S.A, July 6-8, 1994. MIT Press, Cambridge, MA, U.S.A, pp. 371-376.

83. Hemmi, H., Mizoguchi, J. and Shimohara, K. (1995) 'Development and Evolution of Hardware Behaviors', In: *Proceedings of Towards Evolvable Hardware: An International Workshop on Evolvable Systems,* Lausanne, Switzerland, 2-3 October, 1995. Springer-Verlag, Berlin, Germany.

84. Hemmi, H., Mizoguchi, J. and Shimohara, K. (1996) 'Evolving Large Scale Digital Circuits', In: *Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems,* Nara, Japan, May 16-18, 1996. pp. 168-173.

85. Higuchi, T., Iba, H. and Manderick, B. (1994) 'Evolvable Hardware'. *Massively Parallel Artifical Intelligence,* MIT Press, Cambridge, MA, U.S.A.,, pp. 398-421.

86. Higuchi, T., Iwata, M., Kajitani, I., Iba, H., Hirao, Y., Furuya, T. and Manderick, B. (1995) 'Evolvable Hardware and Its Application to Pattern Recognition and Fault-Tolerant Systems', In: *Proceedings of Towards Evolvable Hardware: An International Workshop on Evolvable Systems,* Lausanne, Switzerland, 2-3 October, 1995. Springer-Verlag, Berlin, Germany, pp. 118-135.

87. Higuchi, T., Murakawa, M., Iwata, M., Kajitani, I., Weixin-Liu and Salami, M. (1997) 'Evolvable Hardware at Function Level', In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation,* Indianapolis, IN, U.S.A., 13-16 April, 1997. IEEE Press, New York, NY, U.S.A., pp. 187-192.

88. Hinton, G. E. and Nowlan, S. J. (1987) 'How Learning Can Guide Evolution.' *Complex Systems.,* (1), pp. 495-502.

89. Hirst, A. J. (1996) 'Notes on the Evolution of Adaptive Hardware', In: *Proceedings of the Second International Conference on Adaptive Computing in Engineering Design and*

Control, Plymouth, U.K., 26-28 March, 1996. University of Plymouth, Plymouth, U.K., pp. 212-219.

90. Hogeweg, P. (2000) 'Shapes in the Shadow: Evolutionary Dynamics of Morphogenesis', *Artificial Life,* 6 (1), pp. 85-101.

91. Hohil, M. E., Liu, D. and Smith, S. H. (1999) 'Solving the N-Bit Parity Problem Using Neural Networks', *Neural Networks,* 12 (9), pp. 1321-1323.

92. Holland, J. H. (1998) *Emergence: From Chaos to Order,* Addison-Wesley, Redwood City, CA, U.S.A.

93. Hollingworth, G., Smith, S. and Tyrrell, A. (2000a) 'The Intrinsic Evolution of Virtex Devices through Internet Reconfigurable Logic', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 72-79.

94. Hollingworth, G., Smith, S. and Tyrrell, A. (2000b) 'Safe Intrinsic Evolution of Virtex Devices', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 195-202.

95. Hornby, G. (2003a) *Generative Representations for Evolutionary Design Automation,* Brandeis University, Waltham, MA, U.S.A. Ph.D. Thesis.

96. Hornby, G. (2003b) 'Generative Representations for Evolving Families of Designs', In: *Proceedings of the 2003 Genetic and Evolutionary Computation Conference,* Chicago, IL, U.S.A., July 12-16, 2003. Springer-Verlag, Berlin, Germany, pp. 1678-1689.

97. Hornby, G. S. and Pollack, J. B. (2001) 'The Advantages of Generative Grammatical Encodings for Physical Design', In: *Proceedings of the 2001 Congress on Evolutionary Computation,* Seoul, South Korea, 27-30 May, 2001. IEEE, Piscataway, NJ, U.S.A.

98. Hounsell, B. I. and Arslan, T. (2000) 'A Novel Genetic Algorithm for the Automated Design of Performance Driven Digital Circuits', In: *Proceedings of the 2000 Congress on Evolutionary Computation,* La Jolla, CA, U.S.A., 16-19 July, 2000. IEEE, Piscataway, NJ, U.S.A., pp. 601-608.

99. Hounsell, B. L. and Arslan, T. (2001) 'Evolutionary Design and Adaptation of Digital Filters within an Rmbedded Fault Tolerant Hardware Platform', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 127-135.

100. Huelsbergen, L., Rietman, E. and Slous, R. (1999) 'Evolving Oscillators in Silico', *IEEE Transactions on Evolutionary Computation,* 3 (3), pp. 197-204.

101. Huynen, M. A., Stadler, P. F. and Fontana, W. (1996) 'Smoothness within Ruggedness: The Role of Neutrality in Adaptation', *Proceedings of the National Academy of Science,* 93.

102. Imamura, K., Foster, J. A. and Krings, A. W. (2000) 'The Test Vector Problem and Limitations to Evolving Digital Circuits', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 75-79.

103. Anadigm Inc., A. (2003) *AN120E04 FPAA Data Sheet,* http://www.anadigm.com.

104. Iwata, M., Kajitani, I., Yamada, H., Iba, H. and Higuchi, T. (1996) 'A Pattern Recognition System Using Evolvable Hardware', In: *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature,* Berlin, Germany, 22-26 September, 1996. Springer-Verlag, Berlin, Germany, pp. 761-770.

105. Iwata, M., Kajitani, I., Murakawa, M., Hirao, Y., Iba, H. and Higuchi, T. (2000) 'Pattern Recognition System Using Evolvable Hardware', *Systems and Computers in Japan,* 31 (4), pp. 1-11.

106. Jakobi, N. (1995) 'Harnessing Morphogenesis', In: *Proceedings of the Conference on Information Processing in Cells and Tissues,* Liverpool, U.K.1995. pp. 29-41.

107. Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C. and Wang, A. (1990) 'Evolution as a Theme in Artificial Life: The Genesys/Tracker System', In: *Proceedings of the Workshop on Artificial Life,* Santa Fe, New Mexico, February 5-9, 1990. Addison-Wesley, pp. 549-578.

108. Kahng, A. B. (2004) 'Design for Yield Needed for Further Scaling', *IEE Electronics Systems and Software,* April/May 2004, p. 48.

109. Kajitani, I., Hoshino, T., Iwata, M. and Higuchi, T. (1996) 'Variable Length Chromosome Ga for Evolvable Hardware', In: *Proceedings of the Third IEEE International Conference on Evolutionary Computation,* Nagoya, Japan, 20-22 May, 1996. IEEE Press, New York, NY, U.S.A., pp. 443-447.

110. Kajitani, I., Hoshino, T., Nishikawa, D., Yokoi, H., Nakaya, S., Yamauchi, T., Inuo, T., Kajihara, N., Iwata, M., Keymeulen, D. and Higuchi, T. (1998) 'A Gate-Level EHW Chip: Implementing Ga Operations and Reconfigurable Hardware on a Single Lsi', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 1-12.

111. Kajitani, I., Hoshino, T., Kajihara, N., Iwata, M. and Higuchi, T. (1999) 'An Evolvable Hardware Chip and Its Application as a Multi-Function Prosthetic Hand Controller', In: *Proceedings of the 16th National Conference on Artificial Intelligence,* Orlando, FL, U.S.A., July 18-22, 1999. AAAI Press, Menlo Park, CA, U.S.A, pp. 182-187.

112. Kalganova, T., Miller, J. F. and Fogarty, T. C. (1998) 'Some Aspects of an Evolvable Hardware Approach for Multiple-Valued Combinational Circuit Design', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 78-89.

113. Kalganova, T. (2000) 'Bidirectional Incremental Evolution in Extrinsic Evolvable Hardware', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 65-74.

114. Kaneko, K. and Yomo, T. (1994) 'Cell Division, Differentiation and Dynamic Clustering', *Physica D,* 75, pp. 89-102.

115. Kauffman, S. (1993) *The Origins of Order: Self-Organization and Selection in Evolution,* Oxford University Press, New York.

116. Kauffman, S. A. (1969) 'Metabolic Stability and Epigenesis in Randomly Constructed Genetic Nets', *Journal of Theoretical Biology,* 22, pp. 434-467.

117. Kauffman, S. A. and Levin, S. (1987) 'Towards a General Theory of Adaptive Walks on Rugged Landscapes.' *Journal of Theoretical Biology,* 128, pp. 11-45.

118. Kazadi, S., Qi, Y., Park, I., Huang, N., Hwu, P., Kwan, B., Lue, W. and Li, H. (2001) 'Insufficiency of Piecewise Evolution', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 223-231.

119. Kimura, M. (1983) *The Neutral Theory of Molecular Evolution,* Cambridge University Press, Cambridge, U.K.

120. Kirschner, M. and Gerhart, J. (1998) 'Evolvability', *Proceedings of the National Academy of Science,* 95 (8), pp. 8420-8427.

121. Kitano, H. (1990) 'Designing Neural Networks Using Genetic Algorithm with Graph Generation System', *Complex Systems,* 4, pp. 461-476.

122. Kitano, H. (1995) 'A Simple Model of Neurogenesis and Cell Differentiaton Based on Evolutionary Large-Scale Chaos', *Artificial Life,* 2 (1), pp. 79-99.

123. Kitano, H. (1998) 'Building Complex Systems Using Developmental Process: An Engineering Approach', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 218-229.

124. Knuth, D. E. (1998) *The Art of Computer Programming Volume Iii: Sorting and Searching,* (2nd Edn), Addison Wesley, Reading, MA, U.S.A.

125. Koren, I. (2001) *Computer Arithmetic Algorithms,* (2nd Edn), A.K. Peters, Natick, MA, U.S.A.

126. Koza, J. (1994) *Genetic Programming Ii: Automatic Discovery of Reusable Programs,* MIT Press, Cambridge, MA.

127. Koza, J., Bennett, F. H. I., Andre, D. and Keane, M. A. (1999) *Genetic Programming Iii,* Morgan-Kaufmann, San Francisco, California, U.S.A.

128. Koza, J. R., Bennett, F. H., Andre, D. and Keane, M. A. (1997) 'Reuse, Parameterized Reuse, and Hierarchical Reuse of Substructures in Evolving Electrical Circuits Using Genetic Programming', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1997. Springer-Verlag, London, U.K., pp. 312-326.

129. Koza, J. R., Keane, M. A., Jessen-Yu, Bennett, F. H., III and Mydlowec, W. (2000) 'Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming', *Genetic Programming and Evolvable Machines,* 1 (1-2), pp. 121-164.

130. Koza, J. R. (2003) *Genetic Programming Iv: Routine Human-Competitive Machine Intelligence,* Kluwer Academic Publishers, Norwell, MA, U.S.A.

131. Koza, J. R., Keane, M. A. and Streeter, M. J. (2003) 'The Importance of Reuse and Development in Evolvable Hardware', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 33-42.

132. Krohling, R. A., Zhou, Y. and Tyrrell, A. (2002) 'Evolving FPGA-Based Robot Controllers Using an Evolutionary Algorithm', In: *Proceedings of the First International Conference on Artificial Immune Systems,* Canterbury, U.K., 9-11 September, 2002. University of Kent Printing Unit, Canterbury, Kent, pp. 41-46.

133. Kumar, S. and Bentley, P. J. (1999) 'The Abcs of Evolutionary Design: Investigating the Evolvability of Embryogenies for Morphogenesis.' In: *Late-Breaking Paper at the Genetic and Evolutionary Computation Conference,* Orlando, FL, U.S.A,, 13-17 July, 1999. Morgan Kaufmann, San Francisco, CA, U.S.A., pp. 164-170.

134. Kumar, S. and Bentley, P. J. (2000) 'Implicit Evolvability: An Investigation into the Evolvability of an Embryogeny', In: *Late Breaking paper at the 2000 Genetic and Evolutionary Computation Conference,* Las Vegas, NV, U.S.A., 8-12 July, 2000. Morgan Kaufmann, San Francisco, CA, U.S.A.

135. Kumar, S. and Bentley, P. J. (2003) 'Biologically Inspired Evolutionary Development', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 57-68.

136. Kumar, S. (2004) *Investigating Computational Models of Development for the Construction of Shape and Form,* University College London, London, U.K. Ph.D. Thesis.

137. Langdon, W. B. (1997) 'Scheduling Maintenance of Electrical Power Transmission Networks Using Genetic Programming'. In Warwick, K. and Ekwue, A. O. (Eds.), *Artificial Intelligence Techniques in Power Systems,* IEE Press, London, pp. 220-237.

138. Langdon, W. B. and Poli, R. (1998) *Why "Building Blocks" Don't Work on Parity Problems,* CSRP-98-17, University of Birmingham, Birmingham, U.K.

139. Langeheine, J., Becker, J., Folling, S., Meier, K. and Schemmel, J. (2001) 'A CMOS FPTA Chip for Intrinsic Hardware Evolution of Analog Electronic Circuits', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 172-175.

140. Layzell, P. (1998) 'A New Research Tool for Intrinsic Hardware Evolution', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 47-56.

141. Layzell, P. and Thompson, A. (2000) 'Understanding Inherent Qualities of Evolved Circuits: Evolutionary History as a Predictor of Fault Tolerance', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 133-144.

142. Lehre, P. K. and Haddow, P. C. (2003) 'Developmental Mappings and Phenotypic Complexity', In: *Proceedings of the 2003 Congress on Evolutionary Computation,* Canberra, Australia, 8-12 December, 2003. IEEE, Piscataway, NJ, U.S.A., pp. 62-68.

143. Lempel, A. and Ziv, J. (1976) 'On the Complexity of Nite Sequences', *IEEE Transactions on Information Theory,* 22, pp. 75-81.

144. Levi, D. and Guccione, S. A. (1999) 'Geneticfpga: Evolving Stable Circuits on Mainstream FPGA Devices', In: *Proceedings of the NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 12-17.

145. Levi, D. (2000) 'Hereboy: A Fast Evolutionary Algorithm', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware.,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 17-24.

146. Lewis, E. B. (1992) 'Clusters of Master Control Genes Regulate the Development of Higher Organisms', *Journal of the American Medical Association,* 267, pp. 1524-1531.

147. Li, J. H. and Lim, M. H. (2003) 'Evolvable Fuzzy System for ATM Cell Scheduing', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 208-217.

148. Linden, D. S. and Altshuler, E. E. (1999) 'Evolving Wire Antennas Using Genetic Algorithms: A Review', In: *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 225-232.

149. Linden, D. S. (2001) 'A System for Evolving Antennas in-Situ', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 249-255.

150. Linden, D. S. (2002a) 'An Evolvable Antenna System for Optimizing Signal Strength in-Situ', In: *IEEE Antennas and Propagation Society International Symposium,* San Antonio, TX, U.S.A., 16-21 June, 2002. IEEE, Piscataway, NJ, U.S.A., pp. 346-349.

151. Linden, D. S. (2002b) 'Optimizing Signal Strength in-Situ Using an Evolvable Antenna System', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 147-151.

152. Lindenmayer, A. (1968) 'Mathematical Models for Cellular Interactions in Development I Filaments with One-Sided Inputs', *Journal of Theoretical Biology.,* 18, pp. 280-289.

153. Liu, H., Miller, J. F. and Tyrrell, A. (2004) 'An Intrinsic Robust Transient Fault-Tolerant Developmental Model for Digital Systems', In: *WORLDS Workshop, 2004 Genetic and Evolutionary Computation Conference,* Seattle, Washington, U.S.A., 26-30 July, 2004. Springer-Verlag, Berlin, Germany.

154. Liu, H., Miller, J. and Tyrrell, A. (2005) 'A Biological Development Model for the Design of Robust Multiplier', In: *Proceedings of Applications of Evolutionary Computing, EvoWorkshops,* Lausanne, Switzerland, 30 March - 1 April, 2005. Springer-Verlag, Berlin, Germany, pp. 195-204.

155. Liu, W., Murakawa, M. and Higuchi, T. (1997) 'ATM Cell Scheduling by Function Level Evolvable Hardware', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1997. Springer-Verlag, London, U.K., pp. 180-192.

156. Liu, Y. and Yao, X. (1997) 'Evolving Modular Neural Networks Which Generalise Well', In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation,* Indianapolis, IN, U.S.A., 13-16 April, 1997. IEEE Press, Piscataway, NJ, U.S.A., pp. 605--610.

157. Lohn, J., Larchev, G. and DeMara, R. (2003) 'A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 47-56.

158. Lohn, J., Linden, D. S., Hornby, G. D., Kraus, W. F., Rodriguez-Arroyo, A. and Seufert, S. (2004) 'Evolutionary Design of an X-Band Antenna for Nasa's Space Technology 5 Mission', In: *Proceedings of the 2004 IEEE Antenna and Propagation Society International Symposium and USNC/URSI National Radio Science Meeting,* Monterey, CA. U.S.A., 20-26 June, 2004. IEEE Press, Piscataway, NJ, U.S.A., pp. 2313-2316.

159. Lohn, J. D. and Colombano, S. P. (1998) 'Automated Analog Circuit Synthesis Using a Linear Representation', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 125-133.

160. Lohn, J. D., Haith, G. L., Colombano, S. P. and Stassinopoulos, D. (1999) 'A Comparison of Dynamic Fitness Cchedules for Evolutionary Design of Amplifiers', In: *Proceedings of the NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 87-92.

161. Louis, S. J. and Rawlins, G. J. E. (1991) 'Designer Genetic Algorithms: Genetic Algorithms in Structure Design', In: *Proceedings of the Fourth International Conference on Genetic Algorithms,* San Diego, CA, U.S.A1991. pp. 53-60.

162. Lukac, M., Perkowski, M. A., Goi, H., Pivtoraiko, M., Yu, C. H., Chung, K., Jeech, H., Kim, B. and Kim, Y. D. (2003) 'Evolutionary Approach to Quantum and Reversible Circuits Synthesis', *Artificial Intelligence Review,* 20 (3-4), pp. 361-417.

163. Macias, N. J. and Durbeck, L. J. K. (2002) 'Self-Assembling Circuits with Autonomous Fault Handling', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 46-55.

164. Mandelbrot, B. B. (1982) *The Fractal Geometry of Nature,* W. H. Freeman, New York, NY, U.S.A.

165. Mange, D., Goeke, M., Madon, D., Stauffer, A., Tempesti, G. and Durand, S. (1995) 'Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Array with Self-Repair and Self-Reproducing Properties', In: *Proceedings of Towards Evolvable Hardware: An International Workshop on Evolvable Systems,* Lausanne, Switzerland, 2-3 October, 1995. Springer-Verlag, Berlin, Germany.

166. Mange, D., Sipper, M., Stauffer, A. and Tempesti, G. (2000) 'Toward Self-Repairing and Self-Replicating Hardware: The Embryonics Approach', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 205-214.

167. Manovit, C., Aporntewan, C. and Chongstitvatana, P. (1998) 'Synthesis of Synchronous Sequential Logic Circuits from Partial Input/Output Sequences', In: *Proceedings of the*

Second International Conference on Evolvable Systems, Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 98-105.

168. Marrow, P. (1999) 'Evolvability: Evolution, Computation, Biology', In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program,* Orlando, FL, U.S.A., 13-17 July, 1999. Morgan Kaufmann, San Francisco, CA, U.S.A., pp. 30-33.

169. Masner, J., Cavalieri, J., Frenzel, J. and Foster, J. A. (1999) 'Representation and Robustness for Evolved Sorting Networks', In: *Proceedings of the NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 255-261.

170. Mazumder, P. and Rudnick, E. M. (1999) *Genetic Algorithms for VLSI Design, Layout and Test Automation,* Prentice-Hall, Upper Saddle River, NJ, U.S.A.

171. McFarland, G. (1997) *CMOS Technology Scaling and Its Impact on Cache Delay,* Stanford University, Stanford, CA, U.S.A. Ph.D. Thesis.

172. McMenamin, M. and McMenamin, D. (1990) *The Emergence of Animals: The Cambrian Breakthrough,* Columbia University Press, New York, NY, U.S.A.

173. Miller, J. F., Thomson, P. and Fogarty, T. C. (1997) 'Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study'. In Quagliarella, D., Periaux, J., Poloni, C. and Winter, G. (Eds.), *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications,* John Wiley and Sons Ltd, Chichester, U.K., London, U.K., pp. 105-131.

174. Miller, J. F. and Thomson, P. (1998a) 'Aspects of Digital Evolution: Evolvability and Architecture', In: *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature,* Amsterdam, The Netherlands, September 27-30, 1998. Springer-Verlag, Berlin, Germany, pp. 927-936.

175. Miller, J. F. and Thomson, P. (1998b) 'Aspects of Digital Evolution: Geometry and Learning', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 25-35.

176. Miller, J. F. and Thomson, P. (1998c) 'Evolving Digital Electronic Circuits for Real-Valued Function Generation Using a Genetic Algorithm', In: *Proceedings of the Third Annual Conference on Genetic Programming,* Madison, WI, U.S.A.1998. Morgan Kaufmann, San Francisco, CA, U.S.A,, pp. 863-868.

177. Miller, J. F., Kalganova, T., Lipnitskaya, N. and Job, D. (1999) 'The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles', In: *Proceedings of the AISB Symposium on Creative Evolutionary Systems,* Edinburgh, U.K., April 6-9th, 1999. AISB, Sussex, UK, pp. 65-74.

178. Miller, J. F., Job, D. and Vassilev, V. K. (2000a) 'Principles in the Evolutionary Design of Digital Circuits - Part I', *Genetic Programming and Evolvable Machines,* 1 (1/2), pp. 7-35.

179. Miller, J. F., Job, D. and Vassilev, V. K. (2000b) 'Principles in the Evolutionary Design of Digital Circuits - Part Ii', *Genetic Programming and Evolvable Machines,* 1 (3), pp. 259-288.

180. Miller, J. F. and Thomson, P. (2000) 'Cartesian Genetic Programming', In: *Proceedings of the 2000 European Genetic ProgrammingConference,* Edinburgh, U.K., 15-16 April, 2000. Springer-Verlag, Berlin, Germany, pp. 121-132.

181. Miller, J. F. and Downing, K. (2002) 'Evolution in Materio: Looking Beyond the Silicon Box', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 167-176.

182. Miller, J. F. (2003) 'Evolving Developmental Programs for Adaptation, Morphogenesis, and Self-Repair', In: *Proceedings of the Seventh European Conference on Artificial Life,*

Dortmund, Germany, September 14-17, 2003. Springer-Verlag, Berlin, Germany, pp. 256-265.

183. Miller, J. F. and Thomson, P. (2003) 'A Developmental Method for Growing Graphs and Circuits', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 93-104.

184. Miller, J. F. (2004) 'Evolving a Self-Repairing, Self-Regulating, French Flag Organism', In: *WORLDS Workshop, 2004 Genetic and Evolutionary Computation Conference,* Seattle, Washington, U.S.A., 26-30 July, 2004. Springer-Verlag, Berlin, Germany.

185. Mitchell, M., Hraber, P. T. and Crutchfield, J. P. (1993) 'Revisiting the Edge of Chaos: Evolving Cellular Automata to Perform Computations', *Complex Systems,* 7, pp. 89-130.

186. Mitchell, T. M. (1997) *Machine Learning,* McGraw-Hill, London.

187. Moore, G. E. (1965) 'Cramming More Components onto Integrated Circuits', *Electronics,* 38 (8), pp. 114-117.

188. Morange, M. (2001) *The Misunderstood Gene,* Harvard University Press, Cambridge, MA, U.S.A.

189. Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M. and Higuchi, T. (1996) 'Hardware Evolution at Function Level', In: *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature,* Berlin, Germany, 22-26 September, 1996. Springer-Verlag, Berlin, Germany, pp. 62-71.

190. Murakawa, M., Yoshizawa, S., Adachi, T., Suzuki, S., Takasuka, K., Iwata, M. and Higuchi, T. (1998) 'Analogue EHW Chip for Intermediate Frequency Filters', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 134-143.

191. Murakawa, M., Yoshizawa, S., Kajitani, I., Yao, X., Kajihara, N., Iwata, M. and Higuchi, T. (1999) 'The Grd Chip: Genetic Reconfiguration of DSPs for Neural Network Processing', *IEEE Transactions on Computers,* 48 (6), pp. 628-639.

192. Murakawa, M., Adachi, T., Nino, Y., Takahashi, E., Kasai, Y., Takasuka, K. and Higuchi, T. (2002) 'An AI-Calibrated If Filter: A Yield Enhancement Method with Area and Power Dissipation Reductions', In: *Proceedings of the 2002 IEEE Custom Integrated Circuits Conference,* Orlando, FL, U.S.A., May 12-15, 2002. IEEE Press, Piscataway, NJ, U.S.A., pp. 345-348.

193. Murray, J. D. (2002) *Mathematical Biology,* (3rd Edn), Springer-Verlag, Berlin, Germany.

194. Nolfi, S. and Parisi, D. (1993) *Phylogenetic Recapitulation in the Ontogeny of Artificial Neural Networks,* Technical Report, Institute of Psychology, Rome.

195. Ortega, C. and Tyrrell, A. (1999) 'Reliability Analysis in Self-Repairing Embryonic Systems', In: *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 120-128.

196. Ortega, C. and Tyrrell, A. (2000) 'A Hardware Implementation of an Embryonic Architecture Using Virtex FPGAs', In: *Proceedings of theThird International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer Verlag, Berlin, Germany, pp. 155-164.

197. Plante, J., Shaw, H., Mickens, L. and Johnson-Bey, C. (2003) 'Overview of Field Programmable Analog Arrays as Enabling Technology for Evolvable Hardware for High Reliability Systems', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., p. 77.

198. Plew, J., Arroyo, A. A. and Schwartz, E. M. (2002) 'Applying an Evolvable Hardware Approach to Autonomous Robot Design', In: *Proceedings of the Fifteenth Florida Conference on Recent Advances in Robotics,* Miami, FL., U.S.A., May 23-24, 2002.

199. PLX Technology Inc. (2000) *Pci-9080 Data Book,* (1.06 Edn), http://www.plxtech.com/download/PCI9000/9080/databook/9080db-106.pdf.

200. Poli, R., Page, J. and Langdon, W. B. (1999) 'Smooth Uniform Crossover, Sub-Machine Code GP and Demes: A Recipe for Solving High-Order Boolean Parity Problems', In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference,* Orlando, FL, U.S.A., 13-17 July, 1999. Morgan Kaufmann, San Francisco, CA, U.S.A., pp. 1162-1169.

201. Prusinkiewicz, P., Hammel, M. and Mjolsness, E. (1993) 'Animation of Plant Development', In: *Twentieth Annual Conference on Computer Graphics and Interactive Techniques,* Anaheim, California, U.S.A., August 2-6, 1993. pp. 351--360.

202. Prusinkiewicz, P., Hammel, M., Hanan, J. and Mech, R. (1997) 'Visual Models of Plant Development'. In Rozenberg, G. and Salomaa, A. (Eds.), *Handbook of Formal Languages,* Springer Verlag, Berlin.

203. Quick, T. (2003) 'Evolving Embodied Genetic Regulatory Network-Driven Control Systems', In: *Proceedings of the 2003 European Conference on Artificial Life,* Dortmund, Germany, 14-17 September, 2003. Springer, Berlin, Germany, pp. 266-277.

204. Raichman, N., Segev, R. and Ben-Jacob, E. (2003) 'Evolvable Hardware: Genetic Search in a Physical Realm', *Physica A,* 326 (1-2), pp. 265-285.

205. Ramsden, E. (2001) 'The ispPAC Family of Reconfigurable Analog Circuits', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 176-181.

206. Raven, P. and Johnson, G. (2001) *Biology,* (6th Edn), McGraw-Hill Higher Education.

207. Rendell, L. (1987) 'Similarity-Based Learning and Its Extensions.' *Computational Intelligence,* 3, pp. 241-266.

208. Roggen, D., Floreano, D. and Mattiussi, C. (2003a) 'A Morphogenetic Evolutionary System: Phylogenesis of the Poetic Circuit', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003a. Springer Verlag, Germany, pp. 153-164.

209. Roggen, D., Hofmann, S., Thoma, Y. and Floreano, D. (2003b) 'Hardware Spiking Neural Network with Run-Time Reconfigurable Connectivity in an Autonomous Robot', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 189-198.

210. Roggen, D. and Federici, D. (2004) 'Multi-Cellular Development: Is There Scalability and Robustness to Gain?' In: *Proceedings of the Eighth International Conference on Parallel Problem Solving in Nature,* Birmingham, U.K., 18-22 September, 2004. Springer-Verlag, Berlin, Germany, pp. 391-400.

211. Roggen, D., Thoma, Y. and Sanchez, E. (2004) 'An Evolving and Developing Cellular Electonic Circuit', In: *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems,* Boston, MA, U.S.A., 12-15 September, 2004. MIT Press, Cambridge, MA, U.S.A., pp. 33-38.

212. Rosca, J. P. and Ballard, D. H. (1994) 'Hierarchical Self-Organization in Genetic Programming', In: *Eleventh International Conference on Machine Learning,* New Brunswick, NJ, U.S.A.1994. Morgan Kaufmann, San Francisco, CA, U.S.A.

213. Rosca, J. P. (1995) 'Towards Automatic Discovery of Building Blocks in Genetic Programming', In: *Proceedings ofthe AAAI 1995 Fall Symposium Series.,* Cambridge, MA, U.S.A., 10-12 November, 1995. AAAI Press, Menlo, CA, U.S.A., pp. 78-85.

214. Rosenman, M. A. and Gero, J. S. (1999) 'Evolving Designs by Generating Useful Complex Gene Structures'. In Bentley, P. J. (Ed.) *Evolutionary Design by Computers,* Morgan Kaufmann, San Francisco, pp. 345-364.

215. Rumelhart, D. E., Hinton, G. E. and Williams, R. (1988) 'Learning Internal Representations by Error Propagation'. In Anderson, J. and Rosenfeld, E. (Eds.), *Neurocomputing,* MIT Press, Cambridge, MA, U.S.A., pp. 675-695.

216. Rumelhart, D. E., Widrow, B. and Lehr, M. (1994) 'The Basic Ideas in Neural Networks', *Communications of the ACM,* 37 (3), pp. 87-92.

217. Sakanashi, H., Iwata, M. and Higuchi, T. (2001) ' A Lossless Compression Method for Halftone Images Using Evolvable Hardware', In: *Proceedings of the Fourth International Conference on Evolvable Systems,* Tokyo, Japan, 3-5 October, 2001. Springer-Verlag, Berlin, Germany, pp. 314-326.

218. Salami, M., Murakawa, M. and Higuchi, T. (1996) 'Data Compression Based on Evolvable Hardware', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1996. Springer-Verlag, London, U.K., pp. 169-179.

219. Salami, M., Sakanashi, H., Tanaka, M., Iwata, M., Kurita, T. and Higuchi, T. (1998) 'On-Line Compression of High Precision Printer Images by Evolvable Hardware', In: *Proceedings of the 1998 Data Compression Conference,* Snowbird, UT, U.S.A., 30 March - 1 April, 1998. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 219-228.

220. Santini, C. C., Zebulum, R., Pacheco, M. A. C., Vellasco, M. M. R. and Szwarcman, M. H. (2001) 'PAMA - Programmable Analog Multiplexer Array', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 36-43.

221. Schlosser, G. (2004) 'The Role of Modules in Development and Evolution'. In Schlosser, G. and Wagner, G. (Eds.), *Modularity in Development and Evolution,* University of Chicago Press Ltd., London, U.K., pp. 519-582.

222. Schlosser, G. and Wagner, G. (2004) 'Introduction: The Modularity Concept in Developmental and Evolutionary Biology'. In Schlosser, G. and Wagner, G. (Eds.), *Modularity in Development and Evolution,* The University of Chicago Press Ltd., London, U.K., pp. 1-16.

223. Sechen (1988) *VLSI Placement and Global Routing Using Simulated Annealing,* Kluwer Academic Publishers, Boston, MA, U.S.A.

224. Sekanina, L. (2002a) 'Image Filter Design with Evolvable Hardware', In: *Proceedings of Applications of Evolutionary Computing: EvoWorkshops 2002 (Incorporating EvoCOP, EvoIASP, EvoSTIME/EvoPLAN),* Kinsale, Ireland, 3-4 April, 2002. Springer-Verlag, Berlin, Germany, pp. 255-266.

225. Sekanina, L. (2002b) 'Evolution of Digital Circuits Operating as Image Filters in Dynamically Changing Environment', In: *Proceedings of the Eigth International Conference on Soft Computing,* Brno, Czech Republic, 2002. pp. 33-38.

226. Sekanina, L. (2003a) 'Towards Evolvable IP Cores for FPGAs', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Systems,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 145-154.

227. Sekanina, L. (2003b) 'Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 135-144.

228. Sekanina, L. (2004) 'Evolving Constructors for Infinitely Growing Sorting Networks and Medians', In: *Proceedings of the 30th Conference on Current Trends in Theory and Practice of Computer Science,* Merin, Czech Republic., 24-30 January, 2004. Springer, Berlin, Germany, pp. 314-323.

229. Sekanina, L. (2005) 'Evolutionary Design of Gate-Level Polymorphic Digital Circuits', In: *Proceedings of Applications of Evolutionary Computing, EvoWorkshops,* Lausanne, Switzerland, 30 March - 1 April, 2005. Springer-Verlag, Berlin, Germany, pp. 185-194.

230. Shanthi, A. P., Muruhanandam, P. and Parthasarathi, R. (2004) 'Enhancing the Development Based Evolution of Digital Circuits', In: *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware,* Seattle, WA, U.S.A., June 24-26, 2004. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 91-96.

231. Sheskin, D. J. (2003) *The Handbook of Parametric and Nonparametric Statistical Procedures,* (3rd Edn), Chapman & Hall, Boca Raton, FL, U.S.A.

232. Sims, K. (1994) 'Evolving 3d Morphology and Behavior by Competition', In: *Fourth International Workshop on the Synthesis and Simulation of Living Systems,* Cambridge, MA, U.S.A., July 6-8, 1994. MIT Press, pp. 28-39.

233. Sinohara, H. T., Pacheco, M. A. C. and Vellasco, M. M. R. (2001) 'Repair of Analog Circuits: Extrinsic and Intrinsic Evolutionary Techniques', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 44-47.

234. Sipper, M. (1997) *Evolution of Parallel Cellular Machines: The Cellular Programming Approach,* Springer-Verlag, Heidelberg.

235. Sipper, M., Goeke, M., Mange, D., Stauffer, A., Sanchez, E. and Tomassini, M. (1997a) 'The Firefly Machine: Online Evolware', In: *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation,* Indianapolis, IN, U.S.A., April 13 - 16, 1997. pp. 181-186.

236. Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Perez-Uribe, A. and Stauffer, A. (1997b) 'A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems', *IEEE Transactions on Evolutionary Computation,* 1 (1), pp. 83-97.

237. Slack, J. M. W. (1991) *From Egg to Embryo,* (2nd Edn), Cambridge University Press, Cambridge.

238. Sommerville, I. (1992) *Software Engineering,* (4th Edn), Addison-Wesley, Reading, MA, U.S.A.

239. Stanley, K. and Miikulainen, R. (2003) 'A Taxonomy for Artificial Embryogeny.' *Artificial Life,* 9 (2), pp. 93-130.

240. Stoica, A., Fukunaga, A., Hayworth, K. and Salazar-Lazaro, C. (1998) 'Evolvable Hardware for Space Applications', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 166-173.

241. Stoica, A., Keymeulen, D., Tawel, R., Salazar-Lazaro, C. and Wei-Te-Li. (1999) 'Evolutionary Experiments with a Fine-Grained Reconfigurable Architecture for Analog and Digital CMOS Circuits', In: *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware,* Pasadena, CA, U.S.A., 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 76-84.

242. Stoica, A., Zebulum, R. and Keymeulen, D. (2000) 'Mixtrinsic Evolution', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 208-217.

243. Stoica, A., Keymeulen, D. and Zebulum, R. (2001) 'Evolvable Hardware Solutions for Extreme Temperature Electronics', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 93-97.

244. Stoica, A., Zebulum, R., Keymeulen, D. and Lohn, J. (2002) 'On Polymorphic Circuits and Their Design Using Evolutionary Algorithms', In: *20th IASTED International Multiconference on Applied Informatics,* nnsbruck, Austria, 18-21 February, 2002. ACTA Press, Anaheim, CA, USA, p. iv+490.

245. Stoica, A., Zebulum, R. S., Xin-Guo, Keymeulen, D., Ferguson, M. I. and Vu-Duong. (2003) 'Silicon Validation of Evolution-Designed Circuits', In: *Proceedings of the 2003*

NASA/DoD Conference on Evolvable Hardware, Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 21-25.

246. Stomeo, E. and Kalganova, T. (2004) 'Improving EHW Performance Introducing a New Decomposition Strategy', In: *Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems,* Singapore, 1-3 December, 2004. IEEE Inc., New York, NY, U.S.A., pp. 439-444.

247. Sundaralingam, S. and Sharman, K. C. (1998) 'Evolving Complex Adaptive Iir Structures', In: *9th European Signal Processing Conference,* Rhodes, Greece, 8 - 11 September, 1998. pp. 753-756.

248. Surkan, A. J. and Khuskivadze, A. (2002) 'Evolution of Quantum Computer Algorithms from Reversible Operators', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware.,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 186-187.

249. Takahashi, E., Kasai, Y., Murakawa, M. and Higuchi, T. (2003) 'A Post-Silicon Clock Timing Adjustment Using Genetic Algorithms', In: *Digest of Technical Papers from the 2003 Symposium on VLSI Circuits,* Kyoto, Japan, 12-14 June, 2003. IEEE Press, Piscataway, NJ, U.S.A., pp. 13-16.

250. Tateson, R. (1998) 'Self-Organising Pattern Formation: Fruit Flies and Cell Phones', In: *Parallel Problem Solving From Nature,* Amsterdam, The Netherlands, September, 1998. pp. 732-744.

251. Tempesti, G., Mange, D., Stauffer, A. and Teuscher, C. (2002) 'The Biowall: An Electronic Tissue for Prototyping Bio-Inspired Systems', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 221-230.

252. Tempesti, G., Roggen, D., Sanchez, E., Thoma, Y., Canham, R. and Tyrell, A. (2003) 'Ontogenetic Development and Fault Tolerance in the Poetic Tissue', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 141-152.

253. Thierens, D. (1999) 'Scalability Problems of Simple Genetic Algorithms', *Evolutionary Computation: Special Issue on Scalable Evolutionary Computation,* 7 (4), pp. 331-352.

254. Thoma, Y., Sanchez, E., Moreno Arostegui, J. M. and Tempesti, G. (2003) 'A Dynamic Routing Algorithm for a Bio-Inspired Reconfigurable Circuit', In: *Proceedings of the 13th International Conference on Field Programmable Logic and Applications,* Lisbon, Portugal, 1-3 September, 2003. Springer-Verlag, Berlin, Germany, pp. 681-690.

255. Thoma, Y. and Sanchez, E. (2004) 'A Reconfigurable Chip for Evolvable Hardware', In: *Proceedings of the 2004 Genetic and Evolutionary Computation Conference,* Seattle, Washington, 26-30 June, 2004. Springer-Verlag, pp. 816-827.

256. Thompson, A. (1995a) 'Evolving Electronic Robot Controllers That Exploit Hardware Resources', In: *Proceedings of the Third European Conference on Artificial Life,* Granada, Spain, 4-6 June, 1995. Springer-Verlag, Berlin, Germany, pp. 640-656.

257. Thompson, A. (1995b) 'Evolving Fault Tolerant Systems', In: *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications,* Sheffield, U.K., 12-14 September, 1995. IEE, London, U.K., pp. 524-529.

258. Thompson, A., Harvey, I. and Husbands, P. (1995) 'Unconstrained Evolution and Hard Consequences', In: *Proceedings of Towards Evolvable Hardware: An International Workshop on Evolvable Systems,* Lausanne, Switzerland, 2-3 October, 1995. Springer-Verlag, Berlin, Germany, pp. 136-165.

259. Thompson, A. (1996a) 'Silicon Evolution', In: *Proceedings of the 1st Annual Conference on Genetic Programming,* Stanford, CA, U.S.A.1996. pp. 444-452.

260. Thompson, A. (1996b) *Hardware Evolution,* University of Sussex, Brighton, U.K. D. Phil.

261. Thompson, A. (1996c) 'An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1996. Springer-Verlag, Berlin, Germany, pp. 390-405.

262. Thompson, A. (1996d) 'Evolutionary Techniques for Fault Tolerance', In: *Proceedings of the UKACC International Conference on Control,* Exeter, U.K., 2-5 September, 1996. IEE Press, London, U.K., pp. 693-698.

263. Thompson, A. (1998) 'On the Automatic Design of Robust Electronics through Artificial Evolution', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 13-24.

264. Thompson, A. and Layzell, P. (1999) 'Analysis of Unconventional Evolved Electronics', *Communications of the ACM,* 42 (4), pp. 71-79.

265. Thompson, A., Layzell, P. and Zebulum, R. S. (1999) 'Explorations in Design Space: Unconventional Electronics Design through Artificial Evolution', *IEEE Transactions on Evolutionary Computation,* 3 (3), pp. 167-196.

266. Thompson, A. and Layzell, P. (2000) 'Evolution of Robustness in an Electronics Design', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 218-228.

267. Thompson, A. and Wasshuber, C. (2000) 'Evolutionary Design of Single Electron Systems', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 109-116.

268. Thompson, A. (2002) 'Notes on Design through Artificial Evolution: Opportunities and Algorithms', *Adaptive Computing in Design and Manufacture,* 5 (1), pp. 17-26.

269. Thomson, R. and Arslan, T. (2003) 'The Evolutionary Design and Synthesis of Non-Linear Digital VLSI Systems', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 125-134.

270. Torres, O., Eriksson, J., Moreno, J. M. and Villa, A. (2004) 'Hardware Optimization and Serial Implementation of a Novel Spiking Neuron Model for the Poetic Tissue', *Biosystems,* 76.

271. Torresen, J. (1998) 'A Divide-and-Conquer Approach to Evolvable Hardware', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 57-65.

272. Torresen, J. (2000a) 'Possibilities and Limitations of Applying Evolvable Hardware to Real-World Applications', In: *Proceedings of the 10th International Conference on Field Programmable Logic and Applications,* Villach, Austria, 27-30 August, 2000. Springer-Verlag, Berlin, Germany, pp. 230-239.

273. Torresen, J. (2000b) 'Scalable Evolvable Hardware Applied to Road Image Recognition', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 245-252.

274. Torresen, J. (2001) 'Two-Step Incremental Evolution of a Prosthetic Hand Controller Based on Digital Logic Gates', In: *Proceedings of the Fourth International Conference on Evolvable Systems,* Tokyo, Japan, 3-5 October, 2001. Springer Verlag, Berlin, Germany, pp. 1-13.

275. Torresen, J. (2002a) 'A Scalable Approach to Evolvable Hardware', *Genetic Programming and Evolvable Machines,* 3 (3), pp. 259-282.

276. Torresen, J. (2002b) 'Evolving Both Hardware Subsystems and the Selection of Variants of Such into an Assembled System', In: *Proceeding of Modelling and Simulation 2002: The 16th European Simulation Multiconferece 2002,* Darmstadt, Germany, 3-5 June, 2002. SCS Europe, San Diego, CA, U.S.A., pp. 451-457.

277. Torresen, J. (2002c) 'A Dynamic Fitness Function Applied to Improve the Generalisation When Evolving a Signal Processing Hardware Architecture', In: *Proceedings of Applications of Evolutionary Computing: EvoWorkshops 2002 (Incorporating EvoCOP, EvoIASP, EvoSTIME/EvoPLAN),* Kinsale, Ireland, 3-4 April, 2002. Springer-Verlag, Berlin, Germany, pp. 267-279.

278. Torresen, J. (2004) 'Exploring Knowledge for Efficient Evolution of Hardware', In: *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware,* Seattle, WA, U.S.A., June 24-26, 2004. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 209-216.

279. Tufte, G. and Haddow, P. C. (2000) 'Evolving an Adaptive Digital Filter', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 143-150.

280. Tufte, G. and Haddow, P. (2003a) 'Identification of Functionality During Developmnet on a Virtual Sblock FPGA', In: *Proceedings of the 2003 Congress on Evolutionary Computation,* Canberra, Australia, 8-12 December, 2003. IEEE, Piscataway, NJ, U.S.A., pp. 731-738.

281. Tufte, G. and Haddow, P. C. (2003b) 'Building Knowledge into Developmental Rules for Circuit Design', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Berlin, Germany, pp. 69-81.

282. Turing, A. M. (1952) 'The Chemical Basis of Morphogenesis', *Philosophical Transactions of the Royal Society B,* 237, pp. 37-72.

283. Tyrrell, A., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J. M., Rosenberg, J. and Villa, A. (2003) 'Poetic Tissue: An Integrated Architecture for Bio-Inspired Hardware', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, March 17-20, 2003. Springer-Verlag, Berlin, Germany, pp. 129-140.

284. Tyrrell, A. M., Hollingworth, G. and Smith, S. L. (2001) 'Evolutionary Strategies and Intrinsic Fault Tolerance', In: *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware,* Long Beach, CA, U.S.A., 12-14 July, 2001. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 98-106.

285. Tyrrell, A. M., Krohling, R. A. and Zhou, Y. (2004) 'A New Evolutionary Algorithm for the Promotion of Evolvable Hardware', *IEE Proceedings of Computers and Digital Techniques,* 151 (4), pp. 267-275.

286. van Remortel, P., Lenaerts, T. and Manderick, B. (2002a) 'The Robustness of Small Developped Sblock Circuits Using Different Clocking Schemes', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 26-35.

287. van Remortel, P., Lenaerts, T. and Manderick, B. (2002b) 'Lineage and Induction in the Development of Evolved Genotypes for Non-Uniform 2d Cas', In: *Fifteenth Australian Joint Conference on Artificial Intelligence,* Canberra, Australia2002.

288. van Remortel, P., Ceuppens, J., Defaweux, A., Lenaerts, T. and Manderick, B. (2003) 'Developmental Effects on Tuneable Fitness Landscapes', In: *Proceedings of the Fifth International Conference on Evolvable Systems,* Trondheim, Norway, 17-20 March, 2003. Springer-Verlag, Heidelberg, Germany, pp. 117-128.

289. Vassilev, V., Miller, J. F. and Fogarty, T. C. (1999) 'On the Nature of Two-Bit Multiplier Landscapes', In: *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware,*

Pasadena, CA, U.S.A, 19-21 July, 1999. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 36-45.

290. Vassilev, V. and Miller, J. F. (2000a) 'The Advantages of Landscape Neutrality in Digital Circuit Evolution', In: *Proceedings of the Third International Conference on Evolvable Systems,* Edinburgh, U.K., 17-19 April, 2000. Springer-Verlag, Berlin, Germany, pp. 252-263.

291. Vassilev, V. and Miller, J. F. (2000b) 'Embedding Landscape Neutrality to Build a Bridge from the Conventional to a More Efficient Three-Bit Multiplier Circuit', In: *Proceedings of the 2000 Genetic and Evolutionary Computation Conference,* Las Vegas, NV, U.S.A., July 8-12, 2000. Morgan Kaufmann, San Francisco, CA, U.S.A., p. 539.

292. Vassilev, V. K. and Miller, J. F. (2000c) 'Scalability Problems of Digital Circuit Evolution', In: *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware.,* Palo Alto, CA, U.S.A., 13-15 July, 2000. IEEE Comput. Soc, Los Alamitos, CA, U.S.A., pp. 55-64.

293. Vigander, S. (2001) *Evolutionary Fault Repair of Electronics in Space Applications,* Norwegian University Sci. Tech, Trondheim, Norway. Master's Dissertation.

294. Vinger, K. A. and Torresen, J. (2003) 'Implementing Evolution of Fir-Filters Efficiently in an FPGA', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 26-29.

295. Waddington, C. H. (1942) 'Canalization of Development and the Inheritance of Acquired Characters', *Nature,* 150, pp. 563-565.

296. Wagner, G. and Altenberg, L. (1996) 'Perspective---Complex Adaptations and the Evolution of Evolvability', *Evolution,* 50 (3), pp. 967-976.

297. Walker, J. A. and Miller, J. F. (2004) 'Evolution and Acquisition of Modules in Cartesian Genetic Programming', In: *Proceedings of the Seventh European Conference on Genetic Programming,* Coimbra, Portugal, 5-7 April, 2004. MIT Press, Cambridge, MA, U.S.A., pp. 187-197.

298. White, M. S. and Flockton, S. J. (2003) 'A Comparison of Evolutionary Algorithms for Tracking Time-Varying Recursive Systems', *EURASIP Journal on Applied Signal Processing,* 2003 (8), pp. 834-840.

299. Widrow, B. and Hoff, M. E. J. (1960) 'Adaptive Switching Circuits', In: *1960 IRE WESCON Convention Record,* New York, NY, U.S.A.1960. IRE, New York, NY, U.S.A., pp. 96-104.

300. Wolfram, S. (2002) *A New Kind of Science,* Wolfram Media, Champaign, IL, U.S.A.

301. Wolpert, D. and MacReady, W. (1997) 'No Free Lunch Theorems for Optimization', *IEEE Transactions on Evolutionary Computation,* 1 (1), pp. 67-82.

302. Wolpert, L., Beddington, R., Jessel, T., Lawrence, P., Meyerowitz, E. and Smith, J. (2002) *Principles of Development,* (2nd Edn), Oxford University Press, Oxford, U.K.

303. Xilinx Inc. (1997) *Xilinx Xc6200 FPGA-Based Reconfigurable Co-Processor Data Sheet,* http://www.xilinx.com/partinfo/6200.pdf.

304. Xilinx Inc. (1999) *Xc4000/Xla/Xv Field Programmable Gate Arrays,* http://www.xilinx.com/bvdocs/publications/ds015.pdf.

305. Xilinx Inc. (2001) *Virtex 2.5 V Field Programmable Gate Arrays Data Sheet,* http://direct.xilinx.com/partinfo/ds003.pdf.

306. Yao, X. and Higuchi, T. (1996) 'Promises and Challenges of Evolvable Hardware', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1996. Springer-Verlag, London, U.K., pp. 55-78.

307. Yih, J. S. and Mazumder, P. (1990) 'A Neural Network Design for Circuit Partitioning', *IEEE Transactions on Computer Aided Design,* 9 (10), pp. 1265-1271.

308. Yu, T. and Miller, J. F. (2002) 'Finding Needles in Haystacks Is Not Hard with Neutrality', In: *Proceedings of the Fifth European Conference on Genetic Programming,* Kinsale, Ireland, 3-5 April, 2002. Springer-Verlag, Berlin, Germany, pp. 13-25.

309. Zebulum, R. S., Pacheco, M. A. and Vellasco, M. (1996) 'Evolvable Systems in Hardware Design: Taxonomy, Survey and Applications', In: *Proceedings of the First International Conference on Evolvable Systems,* Tsukuba, Japan, 7-8 October, 1996. Springer-Verlag, London, U.K., pp. 344-358.

310. Zebulum, R. S., Aurélio Pacheo, M. and Vellasco, M. (1997) 'Increasing Length Genotypes in Evolutionary Electronics.' In: *7th International Conference on Genetic Algorithms,* East Lansing, MI, U.S.A1997.

311. Zebulum, R. S., Pacheco, M. A. and Vellasco, M. (1998) 'Analog Circuits Evolution in Extrinsic and Intrinsic Modes', In: *Proceedings of the Second International Conference on Evolvable Systems,* Lausanne, Switzerland, 23-25 September, 1998. Springer-Verlag, Heidelberg, Germany, pp. 154-165.

312. Zebulum, R. S., Keymeulen, D., Vu-Duong, Xin-Guo, Ferguson, M. I. and Stoica, A. (2003) 'Experimental Results in Evolutionary Fault-Recovery for Field Programmable Analog Devices', In: *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware,* Chicago, IL, U.S.A., 9-11 July, 2003. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 182-186.

313. Zhao-Shuguang and Yang-Wanhai (2002) 'Design of Logic Circuits Based on Evolution Computation and Online Evaluation', *Journal of Computer Aided Design & Computer Graphics,* 14 (8), pp. 735-742.

Appendix A: Intrinsic Device Survey

Working at low levels of abstraction can increase the opportunity for evolution to realise innovative designs. This appendix deals with the availability of platforms for hardware evolution that operate at abstractions suitably low to ensure that innovative design spaces are being searched.

In the discussion on evaluation strategies in Section 2.1.2 it was pointed out that any intrinsic platform can be modelled extrinsically by simulation, and that this approach may be more versatile. It was also noted that for work at low levels of abstraction extrinsic evaluation is rarely the most efficient approach, requiring an extremely detailed and computationally expensive simulation. Thus generally the most practical solution for low abstraction evolution is to use a true intrinsic platform.

A brief survey of devices that have reported in the hardware evolution literature is tabulated on the following pages. Some of these devices have only been proposed for hardware evolution experiments, but the majority have actually been used. Some are commercially available, and some have been developed by researchers. Commercial devices have not been developed with hardware evolution as a primary goal and so most struggle to compete with dedicated hardware evolution on performance, versatility and ease of use for the purposes of the hardware evolution researcher. However they do have advantages of availability and cost (although some that were used for early research are now no longer available), and so many researchers have explored their use for hardware evolution. They have been classified into three groups: commercial digital devices (Table A1), commercial analogue devices (Table A2) and research-specific devices (Table A3).

The majority of the work in this thesis was carried out using an intrinsic platform based on a Xilinx Virtex device. Of the commercially available devices available at that time, the Virtex was in the author's opinion the most suitable device for intrinsic evolution. Chapter 3 of this thesis provides full details of the decisions behind this choice and the rest of the design of the evolutionary platform. Chapter 3 goes on to demonstrate hardware evolution of Virtex with fewer constraints than have been reported elsewhere in the literature, and following on from the discussions on innovation and evolvability above discusses how this might be of benefit.

Xilinx 6200 (Thompson, 1996a; Koza et al., 1999; Thompson and Layzell, 2000) Expensive, discontinued. Developed for dynamic reconfiguration applications. Fast and indefinite reconfigurability, fully or partial. Homogenous fine-grained architecture of MUX units. MUX-based routing & unidirectional wiring means all configurations are valid. Good I/O. Fits hardware evolution criteria well.
Xilinx XC4000 (Levi and Guccione 1999) Low cost, discontinued. Indefinite but slow reconfigurability. No partial reconfigurability. Coarse-grained SRAM LUT based architecture. Damaged by invalid configurations. Parts of bitstream proprietary and undisclosed but resources can be reconfigured using Xilinx JBits software. Slow I/O. Fits hardware evolution criteria reasonably.
Xilinx Virtex / Virtex II / Virtex4 (Hollingworth et al., 2000a; Levi, 2000) Medium cost, widely available. Coarse-grained SRAM LUT based architecture. Fast and indefinite reconfigurability, full and partial. Can be damaged by random configurations. Some of the bitstream is proprietary and undisclosed, but many Virtex devices allow most hardware resources to be reconfigured using the Xilinx JBits API. Virtex II provides embedded multipliers, Virtex II Pro and 4 FX provide embedded CPU cores. Good I/O. Fits hardware evolution criteria well.
Altera Flex 10k (Plew et al., 2002) Low cost, limited availability. Coarse-grained SRAM LUT architecture. Can be reconfigured indefinitely, but not partially. Damaged by random configurations. Undisclosed bitstream. Slow I/O. Does not fit criteria for hardware evolution well.
Altera APEX 20K (Roggen et al., 2003b) Medium cost, widely available. Coarse-grained SRAM LUT based architecture. Can be reconfigured indefinitely, but not partially. Can be damaged by random configurations. Proprietary undisclosed bitstream. Good I/O. Does not fit criteria for hardware evolution well.

Table A1: Commercial digital devices.

Anadigm FPAA (Anadigm Inc., 2003): Up to four reconfigurable blocks with programmable interconnect. CABs contain two op-amps, capacitor banks, serial approximation register. Extremely coarse granularity for reconfiguration limits suitability for hardware evolution.
Lattice ispPAC (Ramsden, 2001) Designed for filtering applications. Based on programmable amplifiers. Limited reconfigurability (~10,000x) limits suitability for hardware evolution.
Motorola MPAA020 (Zebulum et al., 1998) 20 cells containing an op. amp, comparator, transistors, capacitors and SRAM. Range of circuits have been evolved. Much of the bitstream is undisclosed. Constrained to op. amp-based circuits, thus of limited use for hardware evolution. Discontinued.
Zetex TRAC (Flockton and Sheehan, 1999) Based around two pipelines of ten op-amps + programmable capacitors, resistors. Linear and non-linear functions successfully evolved. Large grained reconfigurability and limited topology limit suitability for hardware evolution.

Table A2: Commercial analogue devices.

FPTAs (Stoica et al., 1999; Langeheine et al., 2001; Ferguson et al., 2002) Reconfigurable at transistor level, additionally supporting capacitors and multiple I/O points. Programmable voltages control resistances of connecting switches for use as additional transistors. Some versions allow variable channel height and width. FPTA2 provides 8x8 array of FPTA cells
Embryonic Arrays (Tempesti et al., 2002; Tyrrell et al., 2003) Bio-inspired fault tolerant FPGA architecture. Programmable cells based on MUXtrees or LUTs. New POEtic array designed to support hierarchical logical genotype, developmental and phenotype layers, and includes CPU to encode evolutionary algorithm. Interesting architecture for developmental hardware evolution. No embryonic ASICs exist, with the exception of small POEtic samples.
Palmo (Hamilton, Papathanasiou et al. 1998): Designed for ANN applications. Based around array of integrators. PWM-based signalling rather than true analogue. All configurations valid.
Evolvable Motherboard (Layzell, 1998)- Array of analogue switches, connected to six interchangeable evolvable units. Evolution of gates, amplifiers and oscillators demonstrated using bipolar transistors as evolvable unit. Good I/O. Board-based architecture is not suitable for real world problems due to size, cost and number of evolvable units.
FIPSOC (Moreno, Madrenas et al. 1998) Complete evolutionary system aimed at mixed signal environments. Analogue and digital units. CPU and memory to encode evolutionary algorithm. Analogue units based around amplifiers. Digital units based on LUTs and flipflops. Context-based dynamic reconfiguration suitable for real-time adaptive systems.
PAMA (Santini et al., 2001) Fixed analogue MUX array allowing interconnection of interchangeable evolvable units. Most recent version implements a 32 16:1 bidirectional low on-resistance MUX/deMUX allowing for random configurations.

Table A3: Research-specific devices.

Appendix B: Design Space for the Direct Intrinsic Evolution of Two Bit Adder with Carry Circuits

This appendix reports the design constraints and resulting chromosome structure used for the intrinsic evolution of two bit adder with carry circuits in Chapter 3.

B.1 Design Space

Evolution was carried out using a Virtex XCV-400 (Xilinx Inc., 2001) as detailed in Chapter 3. The chromosome was designed to allow evolution control of as many resources as possible found in a generic Virtex configurable logic block (CLB). The Xilinx JBits API, Version 2.8 was used to provide access to Virtex resources (Guccione et al., 1999). The lowest level of access to these resources provided by the API is specified in the *com.xilinx.JBits.Virtex.Bits* package. A number of resources specified by the package were not evolved as they were not specific to a generic CLB or were not likely to be of any use over the small area of the FPGA that we aimed to evolve. These were: I/O blocks, BRAMs, long lines, hex lines, readback logic, chip enable routing and global clock routing. Additionally the lookup tables (LUTs) can be set to operate in a number of RAM modes instead of their standard LUT function. If evolution was given free control over the JBits commands that controlled these it might be possible for undocumented behaviours that could damage the device to arise. Therefore these bits were not evolved.

The logic in each Virtex CLB is driven separately, so it is possible for two CLBs to attempt to drive the same wire to different logic states, thus generate a short circuit. As such configurations might potentially damage the device, the routing between CLBs was restricted so that contention could not arise: First, no tristates or single-to-single programmable interconnection points (PIPs) were evolved. This meant that the only routing resources evolved were the PIPs that route a CLB output bus to the single lines running between neighbouring CLBs. Additionally eight these were not evolved as in some configurations they could result in contention between neighbouring CLBs. All remaining resources in the JBits *Bits* package were made available to evolution. The resources evolved are shown in Tables B1, B2 and B3. Essentially this includes all slice input multiplexers including flipflop and LUT inputs, all slice internals, all slice output bus multiplexers, and most output-bus-to-single PIPs.

B.2 Chromosome Structure

The chromosome was designed to represent the resources detailed above for a 2x2 area of CLBs. Each resource in these CLBs that was to be evolved was represented by an integer gene. The 164 genes needed to represent one CLB are shown, separated into input, internal CLB and output resources, in Tables A1, A2 and A3 respectively. The allele range for each gene varied

depending on the number of values the resource could be set to using the JBits API (beyond restricted values such as outputs driving tristates, as discussed above). For example Input 1 of the F-LUT in slice 0 (S0F1) can take on 27 different values, 25 of which are single lines from the north, south east or west, one of which is a low-latency connection from the CLB to the west, and one of which is an internal connection from the CLB's own Slice 1 X output, S1X. Hence the gene representing the S0F1 input for one CLB has integer alleles ranging from 0 to 26. The number of alleles for each gene is also shown in Tables B1, B2 and B3.

The only features that were not encoded in this way were the LUTs. Instead the logic for each LUT was represented in the chromosome by 16 bits that were used to set the truthtable for the LUT logic function.

A final restriction discussed in Chapter 3 was that the inputs to the CLBs on the west of the 2x2 array were fixed to ensure that the input signals used for evaluation always passed into the evolved area. Hence the 26 input genes shown in Table B1 were not included for each of the two CLBs that lay on the western edge of the evolved array. The chromosome was created by concatenating one copy of the remaining genes shown in Tables A2 and A3 for each of the western CLBs and one copy of the genes listed in all three tables for each of the eastern CLBs. The total chromosome length was 604 genes.

I/O Type	Input Name	Alleles	I/O Type	Input Name	Alleles
Slice 0 LUT Input	S0F1	27	Slice 0 Flipflop Input	S0Clk	8
Slice 0 LUT Input	S0F2	27	Slice 1 Flipflop Input	S1Clk	8
Slice 0 LUT Input	S0F3	27	Slice 0 Flipflop Input	S0SR	11
Slice 0 LUT Input	S0F4	26	Slice 1 Flipflop Input	S1SR	11
Slice 0 LUT Input	S0G1	27	Slice 0 Flipflop Input	S0BX	17
Slice 0 LUT Input	S0G2	27	Slice 1 Flipflop Input	S1BX	17
Slice 0 LUT Input	S0G3	27	Slice 0 Flipflop Input	S0BY	17
Slice 0 LUT Input	S0G4	27	Slice 1 Flipflop Input	S1BY	17
Slice 1 LUT Input	S1F1	27	Slice 0 Flipflop Input	S0CE	11
Slice 1 LUT Input	S1F2	27	Slice 1 Flipflop Input	S1CE	11
Slice 1 LUT Input	S1F3	27			
Slice 1 LUT Input	S1F4	27			
Slice 1 LUT Input	S1G1	27			
Slice 1 LUT Input	S1G2	27			
Slice 1 LUT Input	S1G3	27			
Slice 1 LUT Input	S1G4	27			

Table B1: Details of the input genes for one CLB

Logic Type	Component Name	Alleles	Logic Type	Component Name	Alleles
LUT Truthtable	SLICE0_F (x16)	16x2	LUT Truthtable	SLICE1_F (x16)	16x2
LUT Truthtable	SLICE0_G (x16)	16x2	LUT Truthtable	SLICE1_G (x16)	16x2
CLB Internal	S0Control.YB.YB	2	CLB Internal	S1Control.YB.YB	2
CLB Internal	S0Control.YCarrySelect	2	CLB Internal	S1Control.YCarrySelect	2
CLB Internal	S0Control.Y.Y	3	CLB Internal	S1Control.Y.Y	3
CLB Internal	S0Control.AndMux	4	CLB Internal	S1Control.AndMux	4
CLB Internal	S0Control.ClockInvert	2	CLB Internal	S1Control.ClockInvert	2
CLB Internal	S0Control.CeInvert	2	CLB Internal	S1Control.CeInvert	2
CLB Internal	S0Control.YDin.YDin	2	CLB Internal	S1Control.YDin.YDin	2
CLB Internal	S0Control.YffSetResetSelect	2	CLB Internal	S1Control.YffSetResetSelect	2
CLB Internal	S0Control.XffSetResetSelect	2	CLB Internal	S1Control.XffSetResetSelect	2
CLB Internal	S0Control..XCarrySelect	2	CLB Internal	S1Control..XCarrySelect	2
CLB Internal	S0Control.X.X	3	CLB Internal	S1Control.X.X	3
CLB Internal	S0Control.XDin.XDin	2	CLB Internal	S1Control.XDin.XDin	2
CLB Internal	S0Control.Cin.Cin	2	CLB Internal	S1Control.Cin.Cin	2
CLB Internal	S0Control.BxInvert	2	CLB Internal	S1Control.BxInvert	2
CLB Internal	S0Control.ByInvert	2	CLB Internal	S1Control.ByInvert	2
CLB Internal	S0Control.LatchMode	2	CLB Internal	S1Control.LatchMode	2
CLB Internal	S0Control.Sync	2	CLB Internal	S1Control.Sync	2

Table B2: Details of the CLB internal genes for one CLB

Routing Type	PIP Details	Alleles	Routing Type	PIP Details	Alleles
Output 0 PIP	OUT0_TO_SINGLE_EAST2	2	Output 4 PIP	OUT4_TO_SINGLE_EAST14	2
Output 0 PIP	OUT0_TO_SINGLE_NORTH1	2	Output 4 PIP	OUT4_TO_SINGLE_NORTH13	2
Output 0 PIP	OUT0_TO_SINGLE_SOUTH3	2	Output 4 PIP	OUT4_TO_SINGLE_SOUTH15	2
Output 0 PIP	OUT0_TO_SINGLE_WEST7	2	Output 4 PIP	OUT4_TO_SINGLE_WEST19	2
Output 1 PIP	OUT1_TO_SINGLE_EAST3	2	Output 5 PIP	OUT5_TO_SINGLE_EAST15	2
Output 1 PIP	OUT1_TO_SINGLE_NORTH2	2	Output 5 PIP	OUT5_TO_SINGLE_NORTH14	2
Output 1 PIP	OUT1_TO_SINGLE_SOUTH0	2	Output 5 PIP	OUT5_TO_SINGLE_SOUTH12	2
Output 1 PIP	OUT1_TO_SINGLE_WEST4	2	Output 5 PIP	OUT5_TO_SINGLE_WEST16	2
Output 1 PIP	OUT1_TO_SINGLE_WEST5	2	Output 5 PIP	OUT5_TO_SINGLE_WEST17	2
Output 2 PIP	OUT2_TO_SINGLE_EAST6	2	Output 6 PIP	OUT6_TO_SINGLE_EAST18	2
Output 2 PIP	OUT2_TO_SINGLE_NORTH6	2	Output 6 PIP	OUT6_TO_SINGLE_NORTH18	2
Output 2 PIP	OUT2_TO_SINGLE_NORTH8	2	Output 6 PIP	OUT6_TO_SINGLE_NORTH20	2
Output 2 PIP	OUT2_TO_SINGLE_SOUTH5	2	Output 6 PIP	OUT6_TO_SINGLE_SOUTH17	2
Output 2 PIP	OUT2_TO_SINGLE_SOUTH7	2	Output 6 PIP	OUT6_TO_SINGLE_SOUTH19	2
Output 2 PIP	OUT2_TO_SINGLE_WEST9	2	Output 6 PIP	OUT6_TO_SINGLE_WEST21	2
Output 3 PIP	OUT3_TO_SINGLE_EAST11	2	Output 7 PIP	OUT7_TO_SINGLE_EAST20	2
Output 3 PIP	OUT3_TO_SINGLE_EAST8	2	Output 7 PIP	OUT7_TO_SINGLE_EAST23	2
Output 3 PIP	OUT3_TO_SINGLE_NORTH9	2	Output 7 PIP	OUT7_TO_SINGLE_NORTH21	2
Output 3 PIP	OUT3_TO_SINGLE_SOUTH10	2	Output 7 PIP	OUT7_TO_SINGLE_SOUTH22	2
Output 3 PIP	OUT3_TO_SINGLE_WEST10	2	Output 7 PIP	OUT7_TO_SINGLE_WEST22	2

Table B3: Details of the output-MUX-to-Single-PIP genes for one CLB

Appendix C: Architecture and Virtual Protein Mappings for the Exploratory Developmental System

This appendix presents details of the exploratory developmental system used in Chapter 4. Section 1 of this appendix provides details of how the virtual architecture used in Chapter 4 was mapped to the Virtex architecture. The mapping was selected manually. Section 2 provides the lookup table that was used to map developmental rule activation to virtual protein production

C.1 Virtual Architecture Mapping

Each virtual CLB was mapped onto an S0 slice in a separate Virtex CLB. The S1 slice was not used. The routes between CLBs were selected so that any combination of inputs and outputs could be set on neighbouring Virtex CLBs without the possibility of contention arising. This was achieved by first splitting the eight Virtex CLB outputs multiplexers into two groups, those that would be used to carry F-LUT signals and those that would carry the G-LUT signals. Those multiplexers selected to carry F-LUT signals were OUT0, OUT1, OUT2 and OUT6. G-LUT signals were carried by OUT3, OUT4, OUT5 and OUT7. The JBits API documentation was then used to determine the single lines each of these outputs could connect to (the connections are fairly sparse), and in which direction they allowed signals to pass. The single lines that were to be used to carry the CLB outputs are listed in the two tables below, Table C1 showing the lines that carry a signal from the F-LUT of a CLB to its neighbours, and Table C2 showing lines that carry signals from the G-LUT of a CLB to its neighbours. If a virtual output multiplexer was set by evolution so as to route a LUT signal in a particular direction, the entire set of Virtex output multiplexers from Tables C1 or C2 corresponding to that direction for that LUT were set ON using JBits. For example, if evolution set a virtual CLB to output the F-LUT signal north, OUT0_TO_SINGLE_NORTH0, OUT0_TO_SINGLE_NORTH1, OUT2_TO_SINGLE_NORTH6 and OUT6_TO_SINGLE_NORTH18 were all set ON using JBits.

	North Output Wires	East Output Wires	South Output Wires	West Output Wires
OUT0	OUT0_TO_SINGLE_NORTH0 OUT0_TO_SINGLE_NORTH1	OUT0_TO_SINGLE_EAST2	OUT2_TO_SINGLE_SOUTH5	OUT0_TO_SINGLE_WEST7
OUT1		OUT1_TO_SINGLE_EAST3		OUT1_TO_SINGLE_WEST4
OUT2	OUT2_TO_SINGLE_NORTH6	OUT2_TO_SINGLE_EAST6		OUT2_TO_SINGLE_WEST9
OUT6	OUT6_TO_SINGLE_NORTH18	OUT6_TO_SINGLE_EAST18	OUT6_TO_SINGLE_SOUTH17 OUT6_TO_SINGLE_SOUTH19	OUT6_TO_SINGLE_WEST21

Table C1: Output multiplexer-to-single lines used to carry signals from the F-LUTs.

	North Output Wires	East Output Wires	South Output Wires	West Output Wires
OUT3	OUT3_TO_SINGLE_NORTH9	OUT3_TO_SINGLE_EAST11	OUT3_TO_SINGLE_SOUTH10	OUT3_TO_SINGLE_WEST10
OUT4			OUT4_TO_SINGLE_SOUTH13	
OUT5	OUT5_TO_SINGLE_NORTH14	OUT5_TO_SINGLE_EAST15 OUT5_TO_SINGLE_EAST17	OUT5_TO_SINGLE_SOUTH12	
OUT7	OUT7_TO_SINGLE_NORTH21	OUT7_TO_SINGLE_EAST20	OUT7_TO_SINGLE_SOUTH22	OUT7_TO_SINGLE_WEST22 OUT7_TO_SINGLE_WEST23

Table C2: Output multiplexer-to-single lines used to carry signals from the G-LUTs.

Again the JBits API documentation was used to determine how to map the virtual CLB inputs to Virtex single-to-LUT input lines. If possible, each LUT input line was paired to at least one F and at least one G output line from the table above, so that the input to output route could be set using only one single line. For instance from Table C2 above it can be seen that can OUT7 can connect to a single line travelling south using OUT7_TO_SINGLE_SOUTH22. The JBits documentation showed that Input 1 of both the F and G-LUTs in Slice0 could be connected to this line by setting S0F1.SINGLE_NORTH22 and S0G1.SINGLE_NORTH22 on, so if OUT7_TO_SINGLE_SOUTH22 was set on in the CLB to the north, input 1 of the current CLB would be driven by the G-LUT from the CLB to the north. Direct routes such as this example were found for most inputs, but in some cases additional SingleToSingle PIPs were needed to make the connection. A table of all the routes used to connect the single lines from Tables C1 and C2 above are shown in Table C3.

LUT Input	Driven From	Route
1	North G	S0F(G)1.SINGLE_NORTH22
1	East G	S0F(G)1.SINGLE_NORTH14, SingleToSingle.SINGLE_EAST10_TO_SINGLE_NORTH14
1	South G	S0F(G)1.SINGLE_SOUTH21,
1	West G	S0F(G)1.SINGLE_WEST17
1	North F	S0F(G)1.SINGLE_NORTH19
1	East F	S0F(G)1.SINGLE_WEST13, SingleToSingle.SINGLE_SOUTH11_TO_SINGLE_WEST13, SingleToSingle.SINGLE_EAST9_TO_SINGLE_SOUTH11
1	South F	S0F(G)1.SINGLE_SOUTH6
1	West F	S0F(G)1.SINGLE_WEST18
2	North G	S0F(G)2.SINGLE_NORTH12
2	East G	S0F(G)2.SINGLE_EAST23
2	South G	S0F(G)2.SINGLE_WEST12
2	West G	S0F(G)2.SINGLE_WEST20
2	North F	S0F(G)2.SINGLE_NORTH17
2	East F	S0F(G)2.SINGLE_SOUTH23, SingleToSingle.SINGLE_SOUTH23_TO_SINGLE_EAST21
2	South F	S0F(G)2.SINGLE_SOUTH1
2	West F	S0F(G)2.SINGLE_WEST2
3	North G	S0F(G)3.SINGLE_NORTH13
3	East G	S0F(G)3.SINGLE_EAST10
3	South G	S0F(G)3.SINGLE_SOUTH14
3	West G	S0F(G)3.SINGLE_WEST15
3	North F	S0F(G)3.SINGLE_EAST5, SingleToSingle.SINGLE_NORTH5_TO_SINGLE_EAST5
3	East F	S0F(G)3.SINGLE_EAST7
3	South F	S0F(G)3.SINGLE_SOUTH0
3	West F	S0F(G)3.SINGLE_WEST6
4	North G	S0F(G)4.SINGLE_NORTH10
4	East G	S0F(G)4.SINGLE_EAST22
4	South G	S0F(G)4.SINGLE_SOUTH9
4	West G	S0F(G)4.SINGLE_WEST11
4	North F	S0F(G)4.SINGLE_NORTH5
4	East F	S0F(G)4.SINGLE_EAST4
4	South F	S0F(G)4.SINGLE_SOUTH18
4	West F	S0F(G)4.SINGLE_WEST3

Table C3: The routes used to map the Virtual Architecture inter-CLB routing to Virtex

C.2 Developmental Rule Postcondition Key

The developmental model used in Chapter 4 used a set of five regulatory proteins and three different classes of structural proteins, (a) those that increase the likelihood that a particular route is used as a LUT input, (b) those that increase the likelihood that a LUT minterm is set and (c) those that increase the likelihood that an output route is set. When a rule was activated the seven bit postcondition was used to determine which of these proteins should be produced. A key that shows the postcondition code for each regulatory and input protein used by the model is shown in Table C4. Table C5 shows a similar key for logic and output proteins.

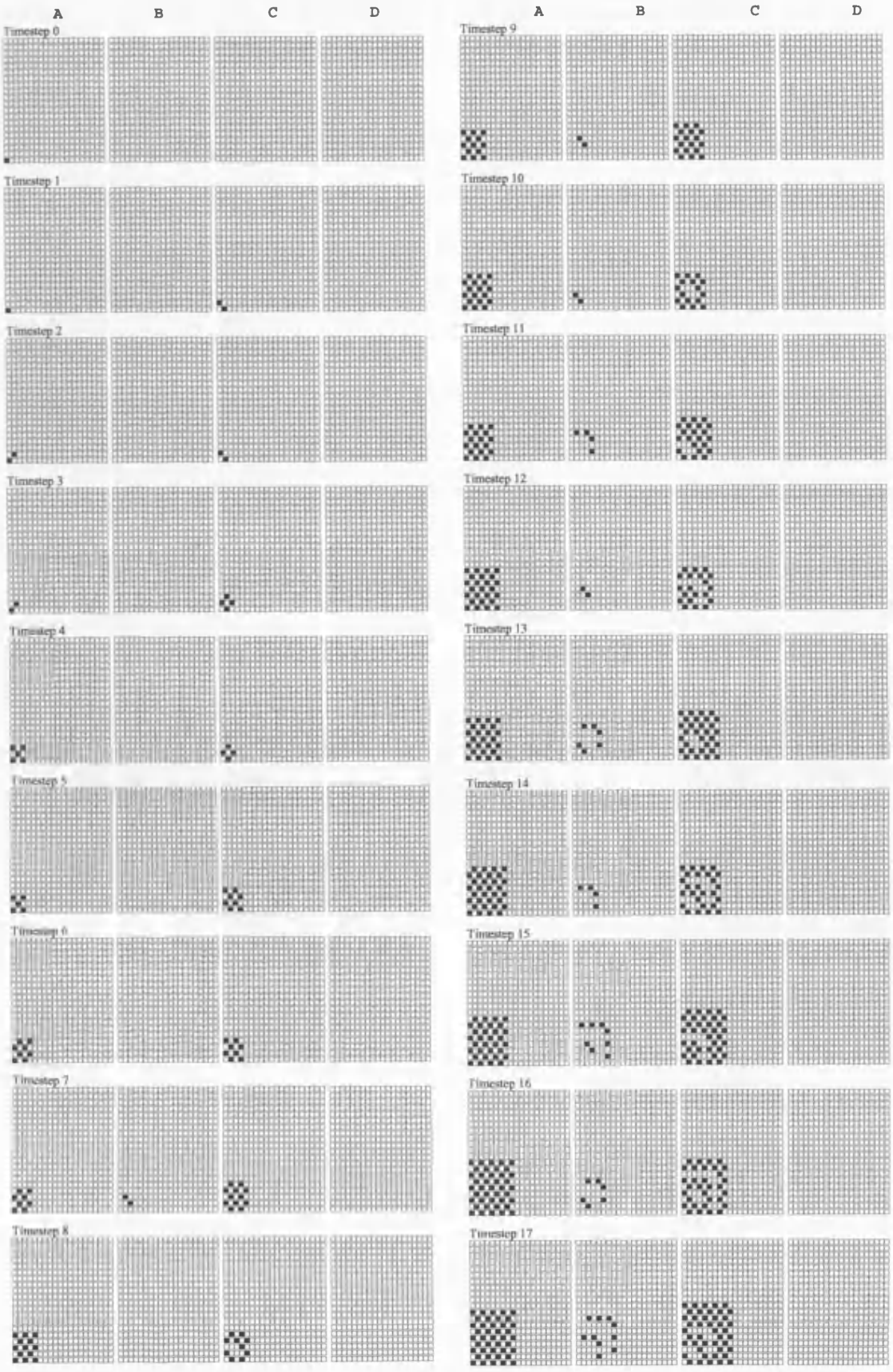
Post-condition	Postcondition Class	Virtual Protein Generated	Effect if activated
0000000	Protein Generator	Protein A	Protein is generated in timestep (t+1) if postcondition is activated at timestep t.
0000001	Protein Generator	Protein B	As above
0000010	Protein Generator	Protein C	As above
0000011	Protein Generator	Protein D	As above
0000100	Protein Generator	Protein E	As above
0000101	Input Selector	CLB Input 1	Selects route from North F-LUT as source of input if this postcondition was the most active of Input1 postconditions during development
0000110	Input Selector	CLB Input 1	As above, but selects North G-LUT
0000111	Input Selector	CLB Input 1	As above, but selects East F-LUT
0001000	Input Selector	CLB Input 1	As above, but selects East G-LUT
0001001	Input Selector	CLB Input 1	As above, but selects South F-LUT
0001010	Input Selector	CLB Input 1	As above, but selects South G-LUT
0001011	Input Selector	CLB Input 1	As above, but selects West F-LUT
0001100	Input Selector	CLB Input 1	As above, but selects West G-LUT
0001101	Input Selector	CLB Input 2	Selects route from North F-LUT as source of input if this postcondition was the most active of Input2 postconditions during development
0001110	Input Selector	CLB Input 2	As above, but selects North G-LUT
0001111	Input Selector	CLB Input 2	As above, but selects East F-LUT
0010000	Input Selector	CLB Input 2	As above, but selects East G-LUT
0010001	Input Selector	CLB Input 2	As above, but selects South F-LUT
0010010	Input Selector	CLB Input 2	As above, but selects South G-LUT
0010011	Input Selector	CLB Input 2	As above, but selects West F-LUT
0010100	Input Selector	CLB Input 2	As above, but selects West G-LUT
0010101	Input Selector	CLB Input 3	Selects route from North F-LUT as source of input if this postcondition was the most active of Input3 postconditions during development
0010110	Input Selector	CLB Input 3	As above, but selects North G-LUT
0010111	Input Selector	CLB Input 3	As above, but selects East F-LUT
0011000	Input Selector	CLB Input 3	As above, but selects East G-LUT
0011001	Input Selector	CLB Input 3	As above, but selects South F-LUT
0011010	Input Selector	CLB Input 3	As above, but selects South G-LUT
0011011	Input Selector	CLB Input 3	As above, but selects West F-LUT
0011100	Input Selector	CLB Input 3	As above, but selects West G-LUT
0011101	Input Selector	CLB Input 4	Selects route from North F-LUT as source of input if this postcondition was the most active of Input4 postconditions during development
0011110	Input Selector	CLB Input 4	As above, but selects North G-LUT
0011111	Input Selector	CLB Input 4	As above, but selects East F-LUT
0100000	Input Selector	CLB Input 4	As above, but selects East G-LUT
0100001	Input Selector	CLB Input 4	As above, but selects South F-LUT
0100010	Input Selector	CLB Input 4	As above, but selects South G-LUT
0100011	Input Selector	CLB Input 4	As above, but selects West F-LUT
0100100	Input Selector	CLB Input 4	As above, but selects West G-LUT

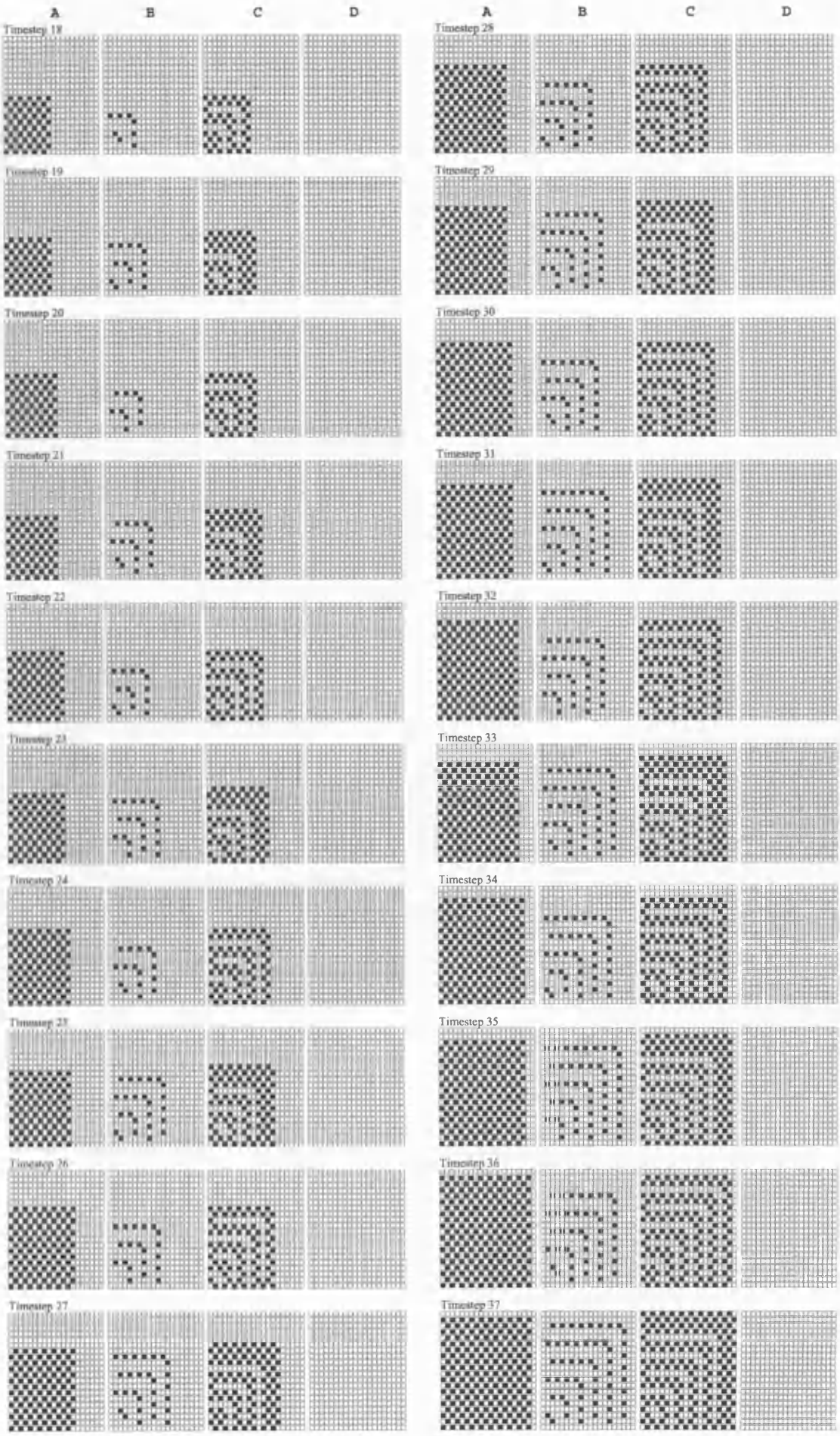
Table C4: Protein and Input rule postconditions, and their effect during development

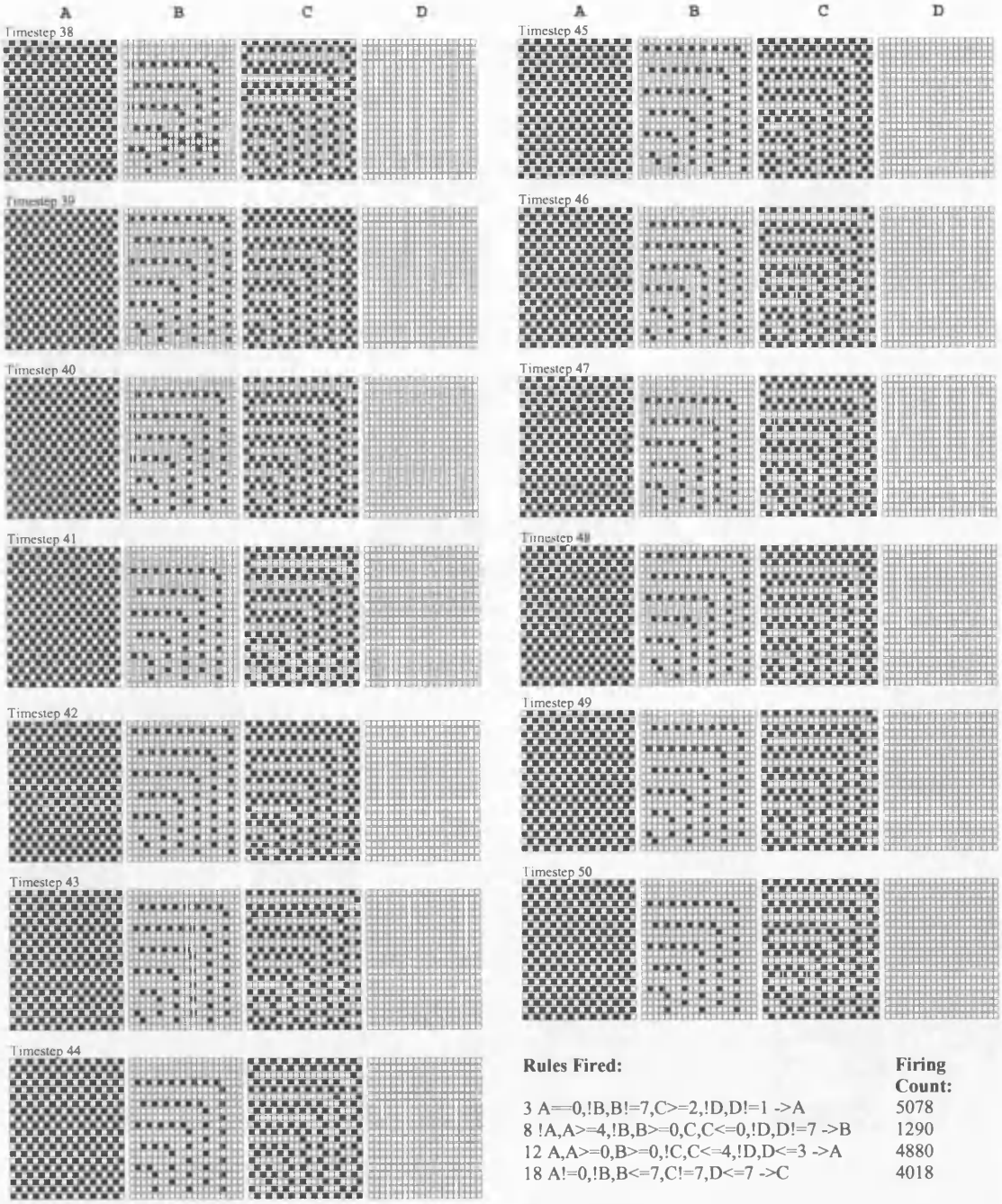
Post-condition	Postcondition Class	Virtual Protein Generated	Effect if activated
0100101	Logic Selector	F-LUT P-Term 0	Sets P-Term ON if this postcondition is activated 7 or more times during development
0100110	Logic Selector	F-LUT P-Term 1	As above
0100111	Logic Selector	F-LUT P-Term 2	As above
0101000	Logic Selector	F-LUT P-Term 3	As above
0101001	Logic Selector	F-LUT P-Term 4	As above
0101010	Logic Selector	F-LUT P-Term 5	As above
0101011	Logic Selector	F-LUT P-Term 6	As above
0101100	Logic Selector	F-LUT P-Term 7	As above
0101101	Logic Selector	F-LUT P-Term 8	As above
0101110	Logic Selector	F-LUT P-Term 9	As above
0101111	Logic Selector	F-LUT P-Term 10	As above
0110000	Logic Selector	F-LUT P-Term 11	As above
0110001	Logic Selector	F-LUT P-Term 12	As above
0110010	Logic Selector	F-LUT P-Term 13	As above
0110011	Logic Selector	F-LUT P-Term 14	As above
0110100	Logic Selector	F-LUT P-Term 15	As above
0110101	Logic Selector	G-LUT P-Term 0	As above
0110110	Logic Selector	G-LUT P-Term 1	As above
0110111	Logic Selector	G-LUT P-Term 2	As above
0111000	Logic Selector	G-LUT P-Term 3	As above
0111001	Logic Selector	G-LUT P-Term 4	As above
0111010	Logic Selector	G-LUT P-Term 5	As above
0111011	Logic Selector	G-LUT P-Term 6	As above
0111100	Logic Selector	G-LUT P-Term 7	As above
0111101	Logic Selector	G-LUT P-Term 8	As above
0111110	Logic Selector	G-LUT P-Term 9	As above
0111111	Logic Selector	G-LUT P-Term 10	As above
1000000	Logic Selector	G-LUT P-Term 11	As above
1000001	Logic Selector	G-LUT P-Term 12	As above
1000010	Logic Selector	G-LUT P-Term 13	As above
1000011	Logic Selector	G-LUT P-Term 14	As above
1000100	Logic Selector	G-LUT P-Term 15	As above
1000101	Output Selector	F-LUT North	If this postcondition is activated 7 or more times during development, routes the LUT output in the direction given in column 3.
1000110	Output Selector	F-LUT East	As above
1000111	Output Selector	F-LUT South	As above
1001000	Output Selector	F-LUT West	As above
1001001	Output Selector	G-LUT North	As above
1001010	Output Selector	G-LUT East	As above
1001011	Output Selector	G-LUT South	As above
1001100	Output Selector	G-LUT West	As above
1001101	N/A	N/A	No effect
.	.	.	.
.	.	.	.
.	.	.	.
1111111	N/A	N/A	No effect

Table C5: Logic and output postconditions, and their effect during development

Appendix D1: Protein map showing the development of a solution from Experiment 7A




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
Timestep 27
A
B
C
D
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35
Timestep 36
Timestep 37

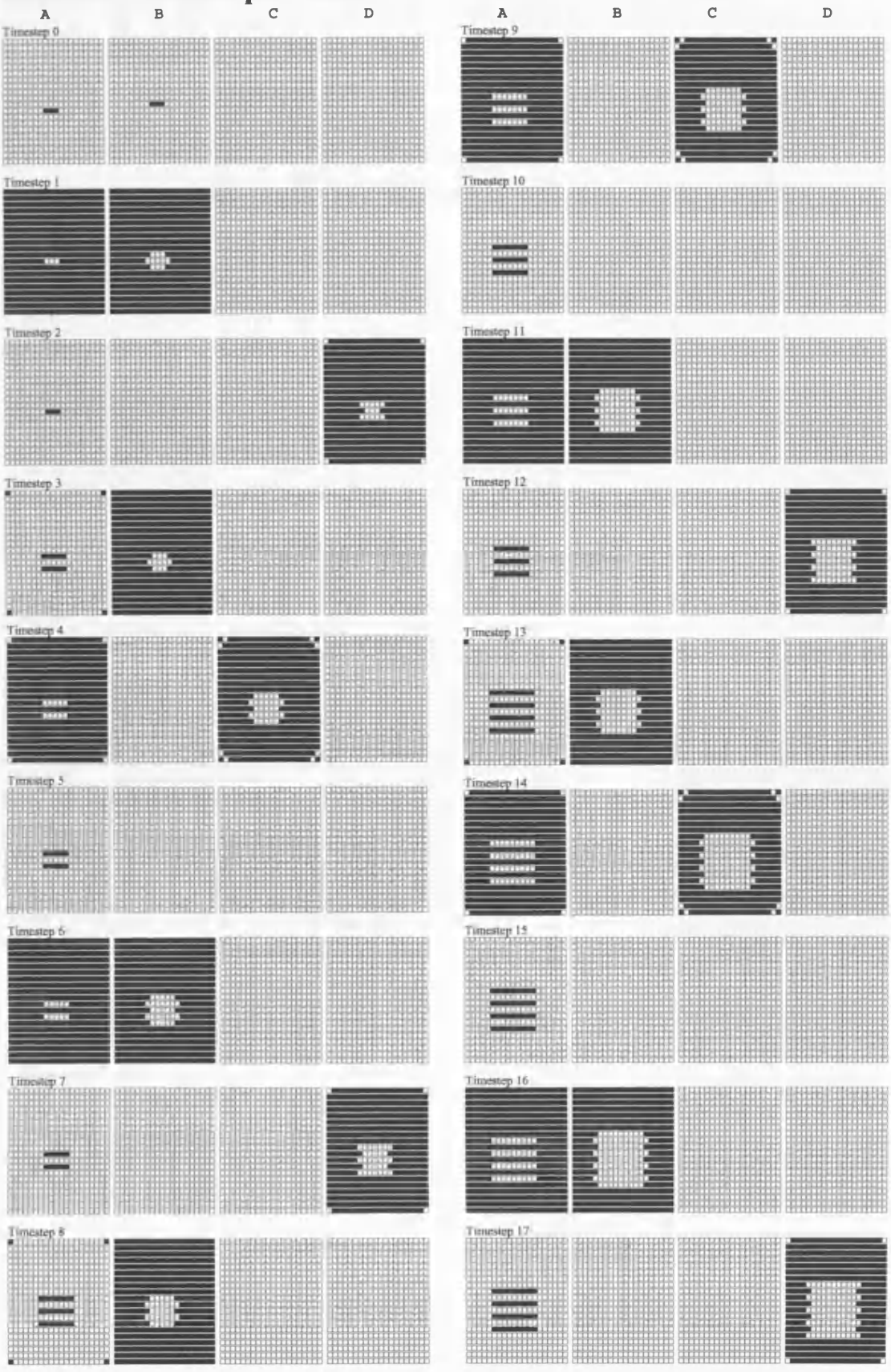


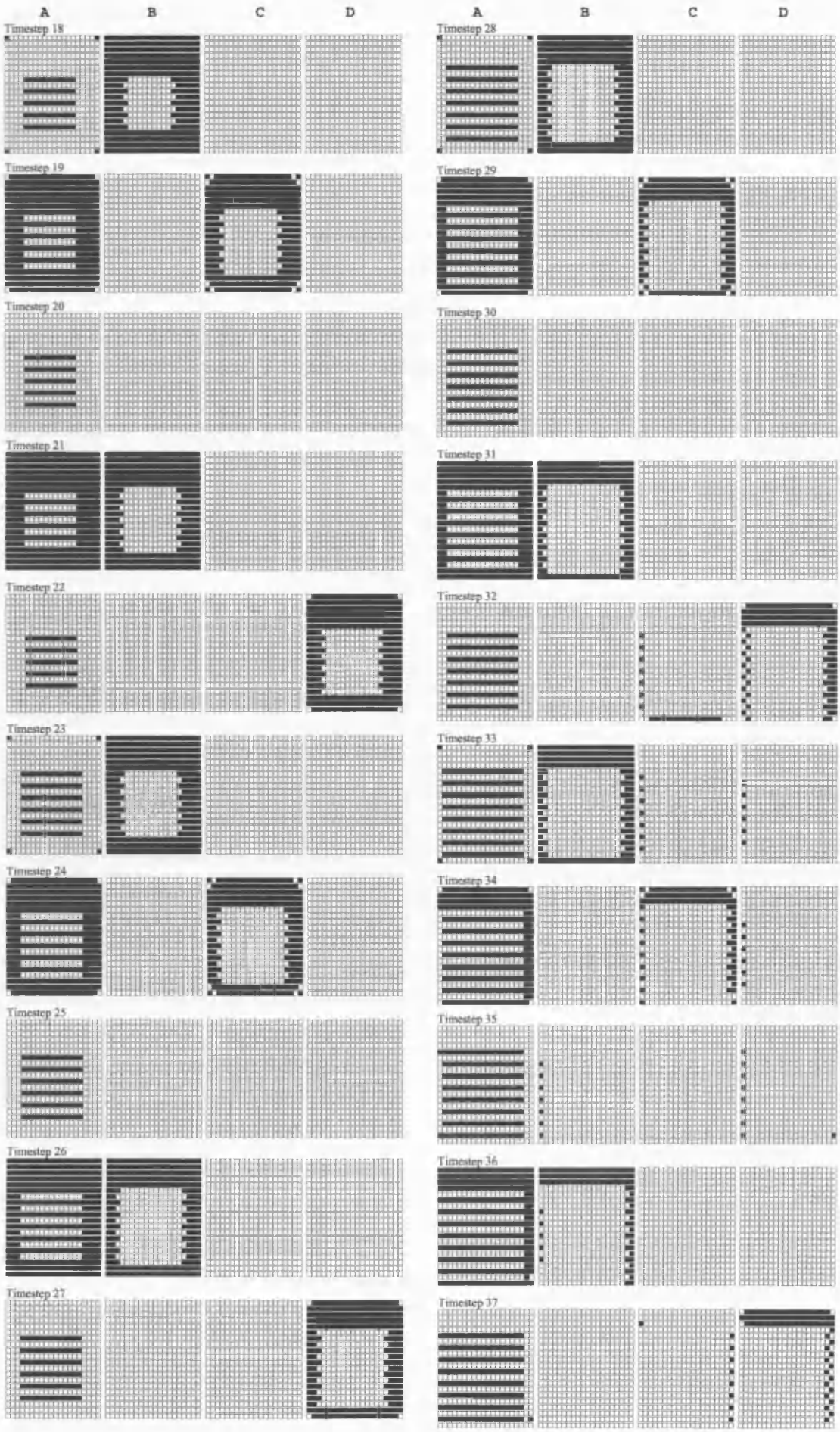
Rules Fired:	**Firing Count:**
3 A==0,!B,B!=7,C>=2,!D,D!=1 ->A	5078
8 !A,A>=4,!B,B>=0,C,C<=0,!D,D!=7 ->B	1290
12 A,A>=0,B>=0,!C,C<=4,!D,D<=3 ->A	4880
18 A!=0,!B,B<=7,C!=7,D<=7 ->C	4018

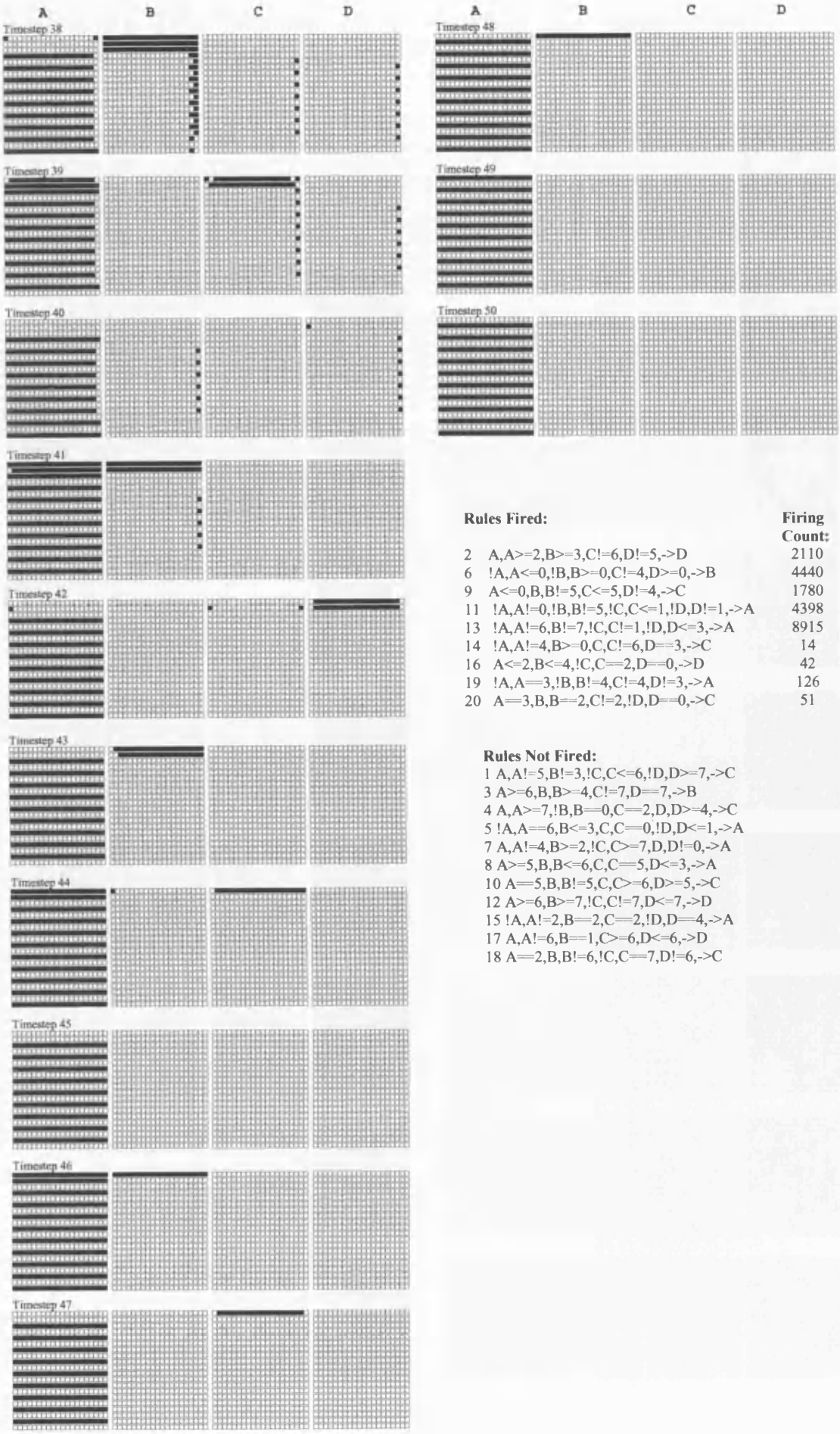
Rules Not Fired:

1 A,A==3,B<=5,C<=6,D>=5 ->A
2 A,A>=5,B>=2,C>=0,!D,D==4 ->D
4 A==1,B,B<=0,C!=3,D>=6 ->D
5 A,A>=6,!B,B==2,!C,C==7,D>=6 ->D
6 !A,A==7,B==7,!C,C>=1,!D,D>=6 ->D
7 A==4,B==5,!C,C<=2,!D,D<=5 ->C
9 A,A==1,B<=1,C<=5,!D,D<=0 ->A
10 A!=4,!B,B!=1,C,C==1,D<=1 ->D
11 A,A==3,B>=7,C<=7,D==3 ->C
13 A>=3,!B,B==5,C,C==0,D>=3 ->D
14 A==7,!B,B==0,C,C>=5,!D,D==2 ->B
15 A!=0,!B,B==4,C,C>=5,!D,D==6 ->D
16 !A,A>=2,B==1,C,C<=5,D,D==4 ->B
17 A<=6,!B,B==5,C,C==5,D>=7 ->D
19 !A,A!=7,B>=2,C!=6,D==5 ->C
20 A!=1,B==6,C==7,D>=6 ->B

Appendix D2: Protein map showing the development of a solution from Experiment 7B




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
Timestep 27
A
B
C
D
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35
Timestep 36
Timestep 37

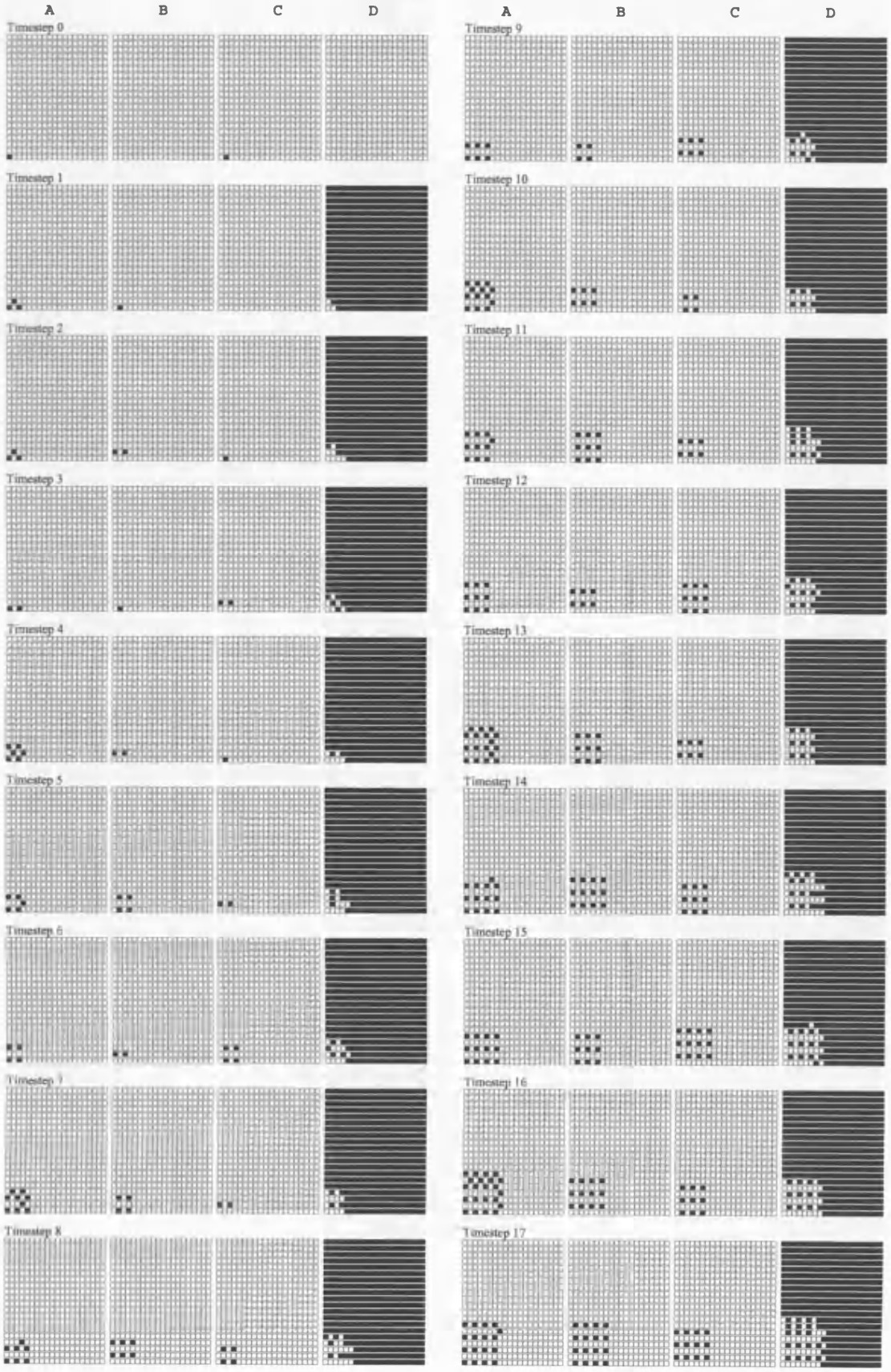


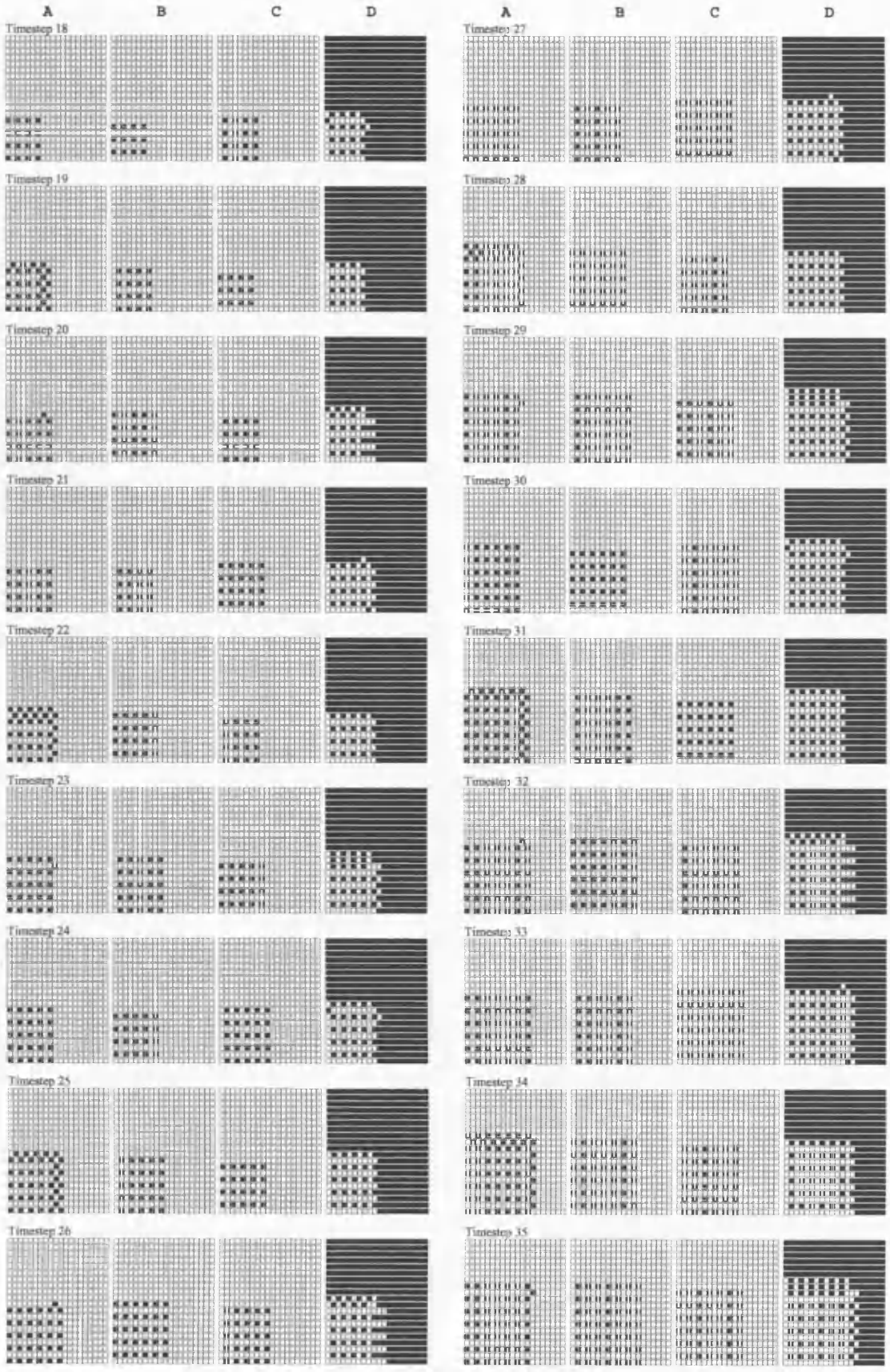
Rules Fired:		Firing Count:
2	A,A>=2,B>=3,C!=6,D!=5,->D	2110
6	!A,A<=0,!B,B>=0,C!=4,D>=0,->B	4440
9	A<=0,B,B!=5,C<=5,D!=4,->C	1780
11	!A,A!=0,!B,B!=5,!C,C<=1,!D,D!=1,->A	4398
13	!A,A!=6,B!=7,!C,C!=1,!D,D<=3,->A	8915
14	!A,A!=4,B>=0,C,C!=6,D==3,->C	14
16	A<=2,B<=4,!C,C==2,D==0,->D	42
19	!A,A==3,!B,B!=4,C!=4,D!=3,->A	126
20	A==3,B,B==2,C!=2,!D,D==0,->C	51

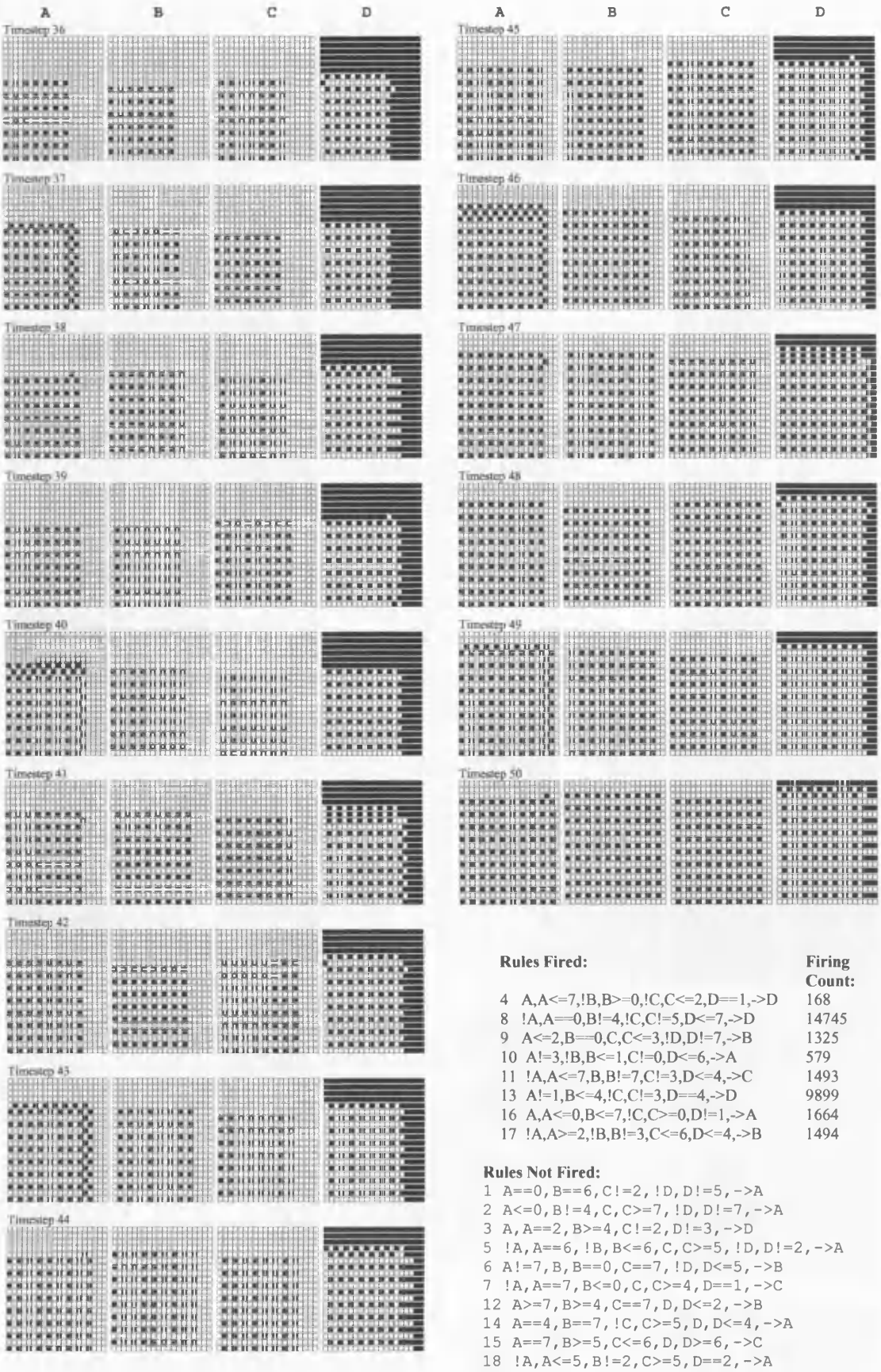
Rules Not Fired:

1 A,A!=5,B!=3,!C,C<=6,!D,D>=7,->C
3 A>=6,B,B>=4,C!=7,D==7,->B
4 A,A>=7,!B,B==0,C==2,D,D>=4,->C
5 !A,A==6,B<=3,C,C==0,!D,D<=1,->A
7 A,A!=4,B>=2,!C,C>=7,D,D!=0,->A
8 A>=5,B,B<=6,C,C==5,D<=3,->A
10 A==5,B,B!=5,C,C>=6,D>=5,->C
12 A>=6,B>=7,!C,C!=7,D<=7,->D
15 !A,A!=2,B==2,C==2,!D,D==4,->A
17 A,A!=6,B==1,C>=6,D<=6,->D
18 A==2,B,B!=6,!C,C==7,D!=6,->C

Appendix D3: Protein map showing the development of a solution from Experiment 7C




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
A
B
C
D
Timestep 27
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35

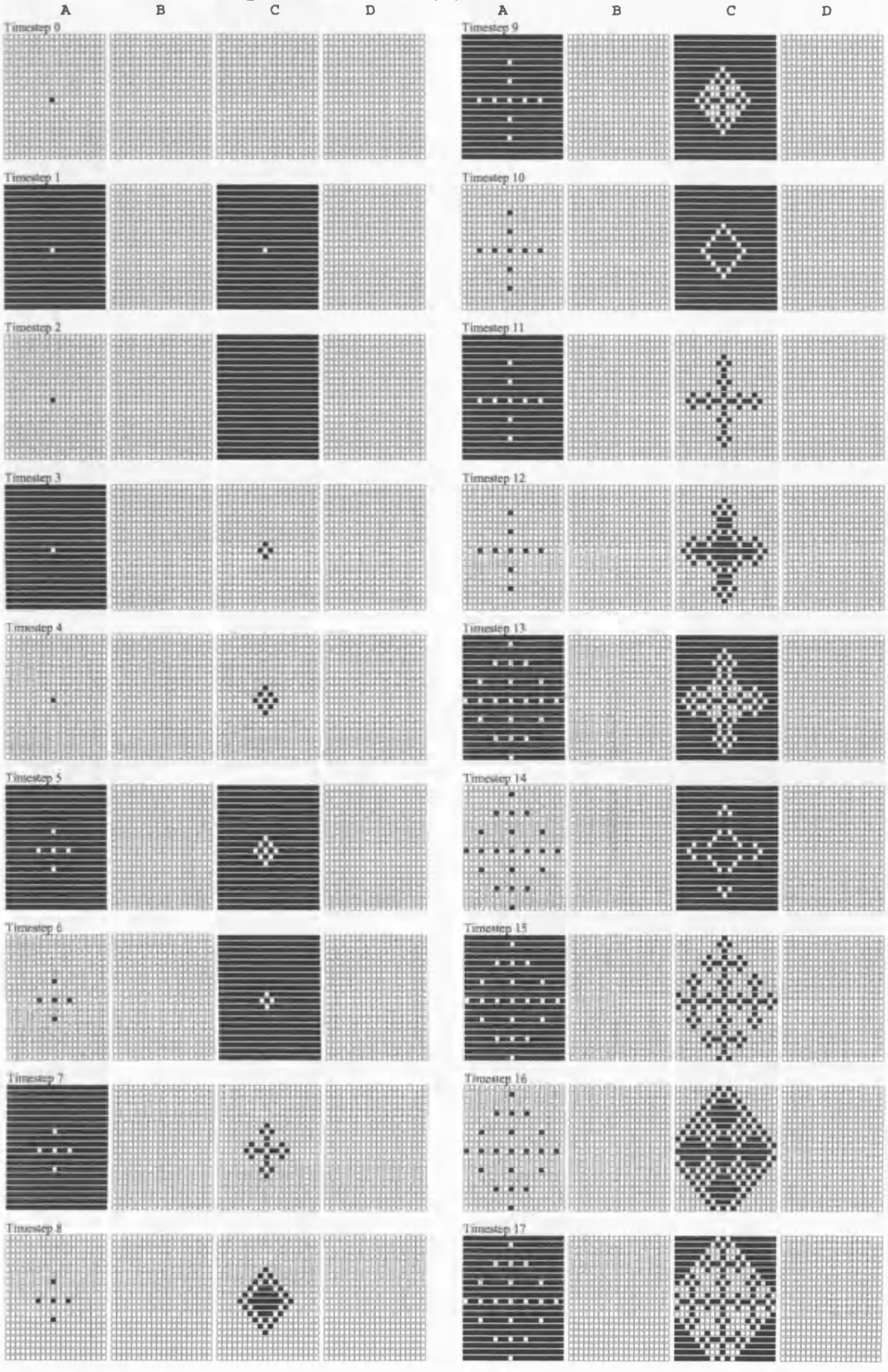


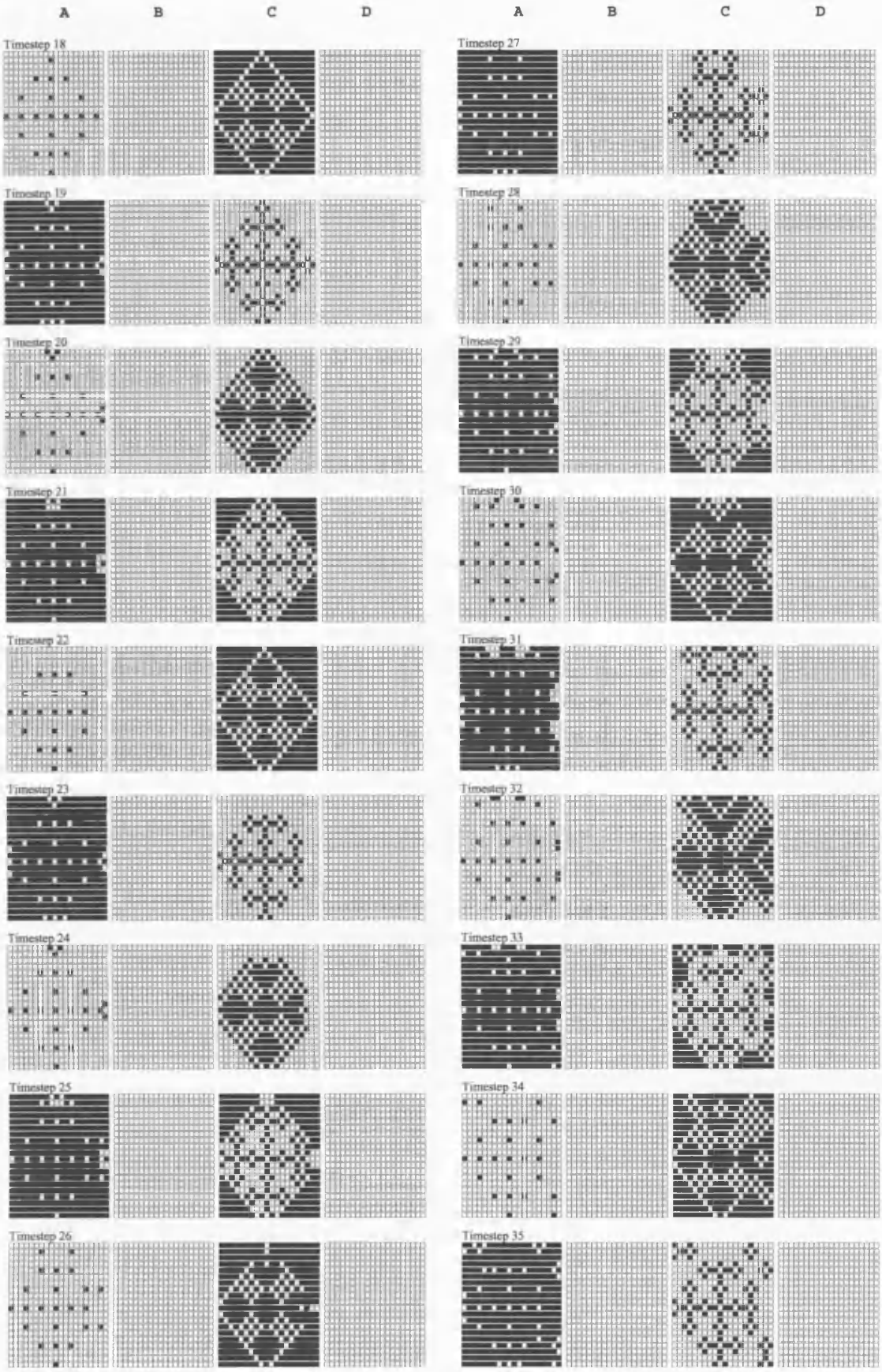
Rules Fired:	Firing Count:
4 A,A<=7,!B,B>=0,!C,C<=2,D==1,->D	168
8 !A,A==0,B!=4,!C,C!=5,D<=7,->D	14745
9 A<=2,B==0,C,C<=3,!D,D!=7,->B	1325
10 A!=3,!B,B<=1,C!=0,D<=6,->A	579
11 !A,A<=7,B,B!=7,C!=3,D<=4,->C	1493
13 A!=1,B<=4,!C,C!=3,D==4,->D	9899
16 A,A<=0,B<=7,!C,C>=0,D!=1,->A	1664
17 !A,A>=2,!B,B!=3,C<=6,D<=4,->B	1494

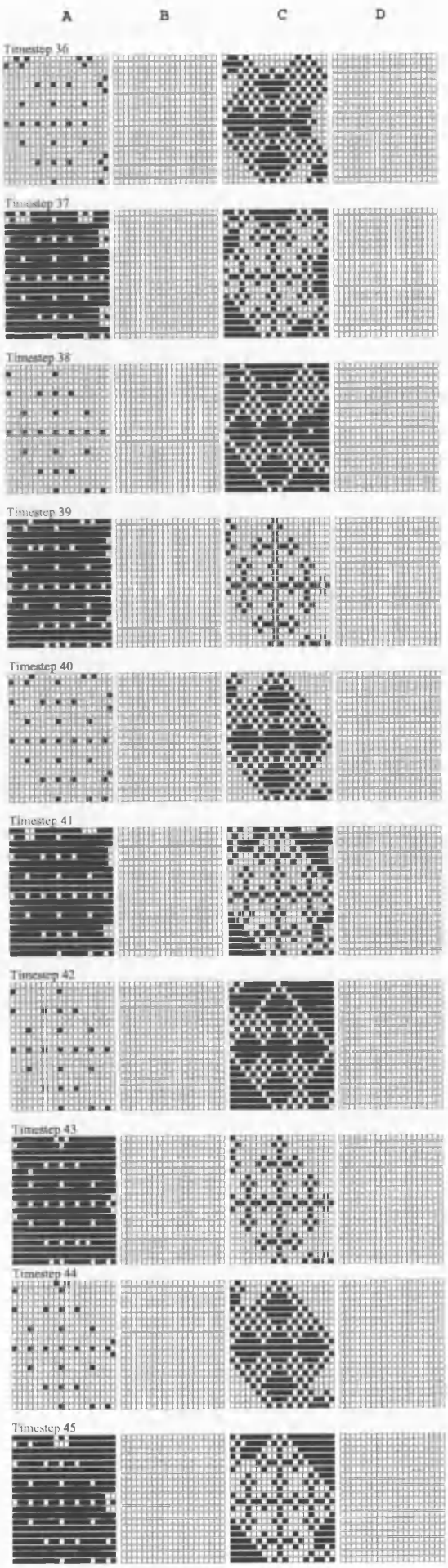
Rules Not Fired:

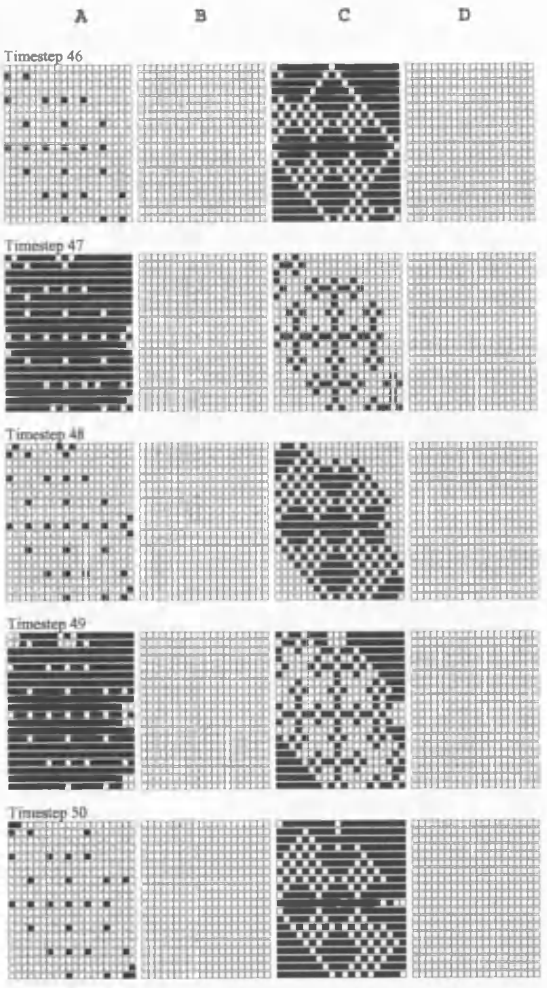
1 A==0,B==6,C!=2,!D,D!=5,->A
2 A<=0,B!=4,C,C>=7,!D,D!=7,->A
3 A,A==2,B>=4,C!=2,D!=3,->D
5 !A,A==6,!B,B<=6,C,C>=5,!D,D!=2,->A
6 A!=7,B,B==0,C==7,!D,D<=5,->B
7 !A,A==7,B<=0,C,C>=4,D==1,->C
12 A>=7,B>=4,C==7,D,D<=2,->B
14 A==4,B==7,!C,C>=5,D,D<=4,->A
15 A==7,B>=5,C<=6,D,D>=6,->C
18 !A,A<=5,B!=2,C>=5,D==2,->A
19 A,A==4,!B,B<=5,!C,C>=6,D,D>=4,->C
20 !A,A>=6,!B,B==3,!C,C<=7,!D,D>=5,->B

Appendix D4: Protein map showing the development of a solution from Experiment 8A(2)




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
A
B
C
D
Timestep 27
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35



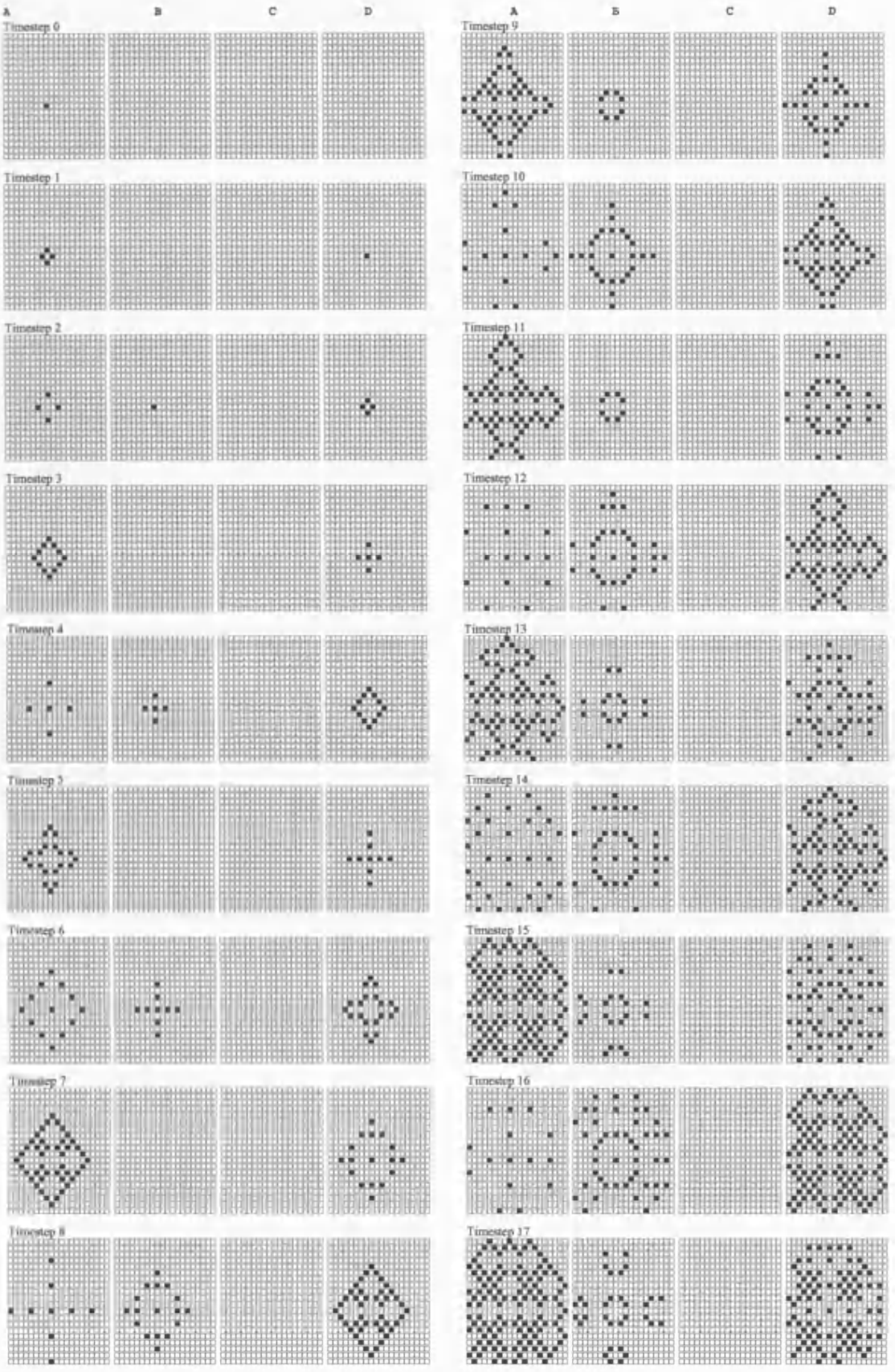


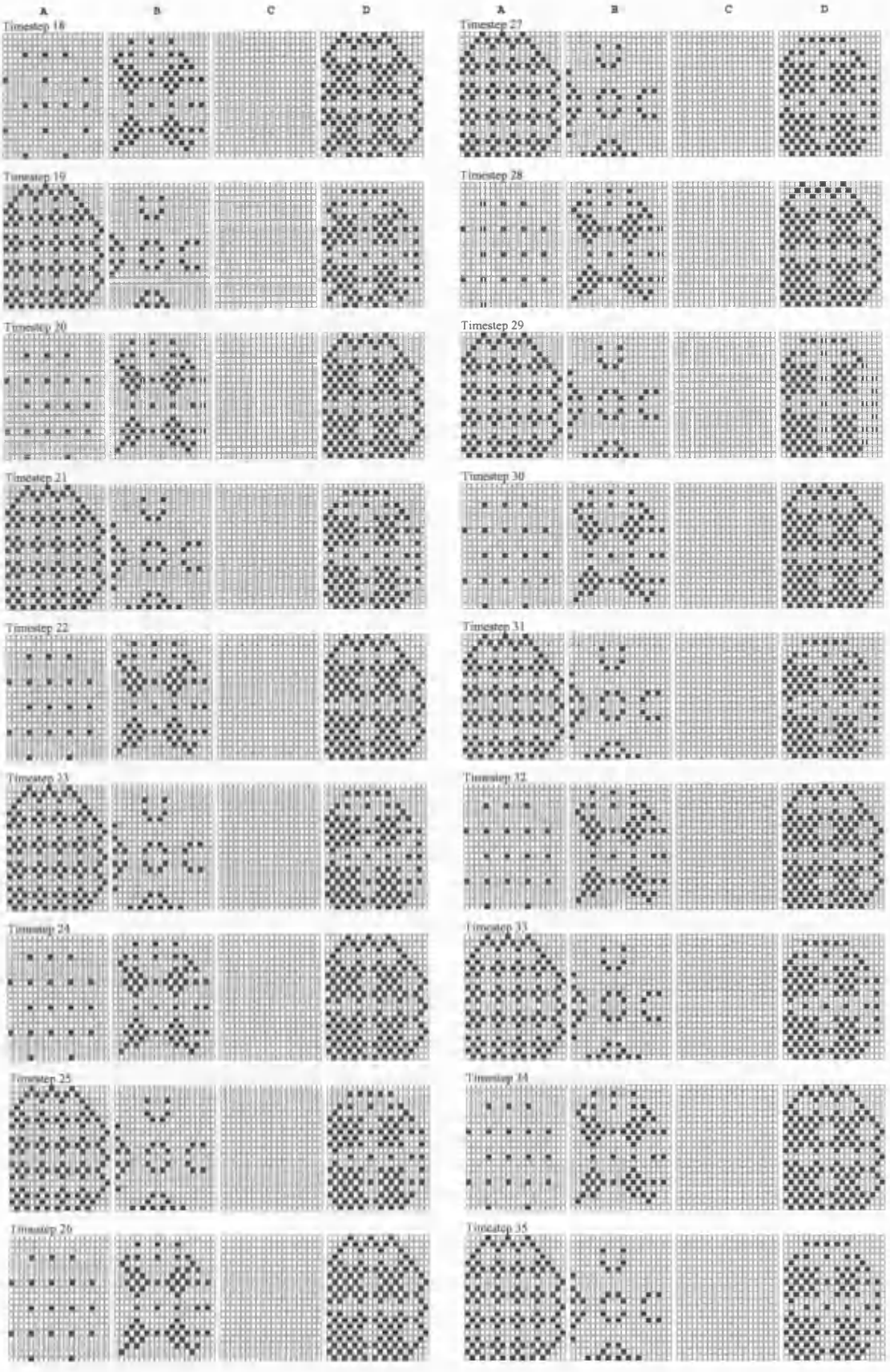
Rules Fired:	**Firing Count:**
8 !A,A!=2,B<=0,C!=1,!D,D<=5,->A	9864
13 !A,A<=1,B<=3,!C,C<=2,!D,D!=3,->C	2465
15 A>=1,B<=4,C>=1,D!=4,->C	8080

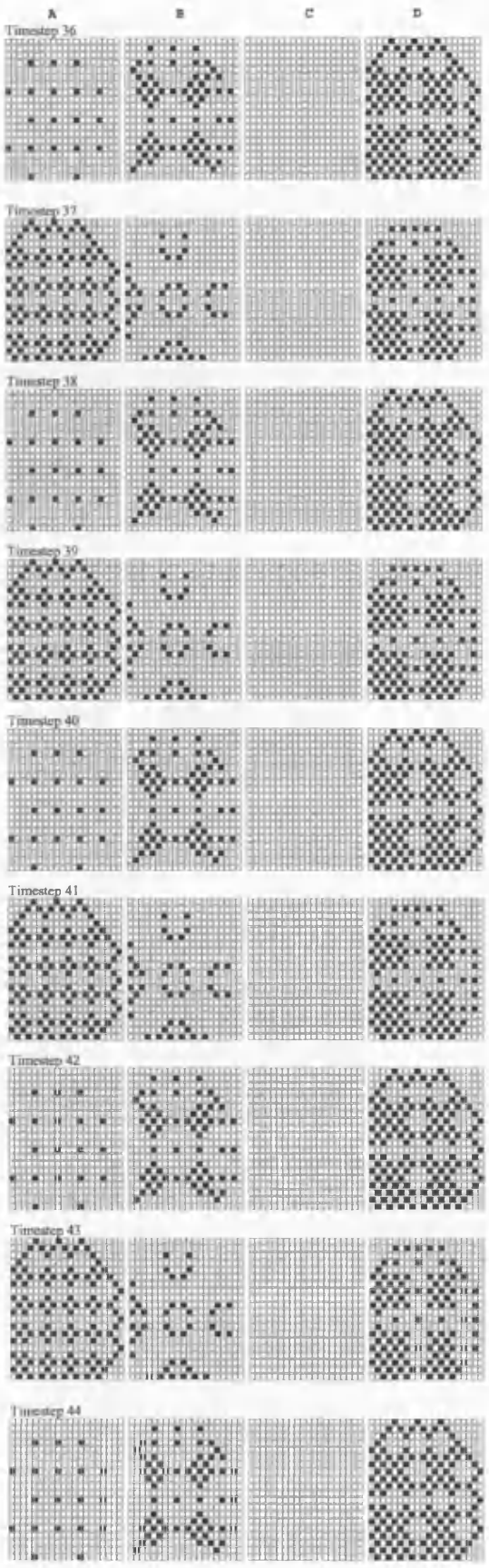
Rules Not Fired:

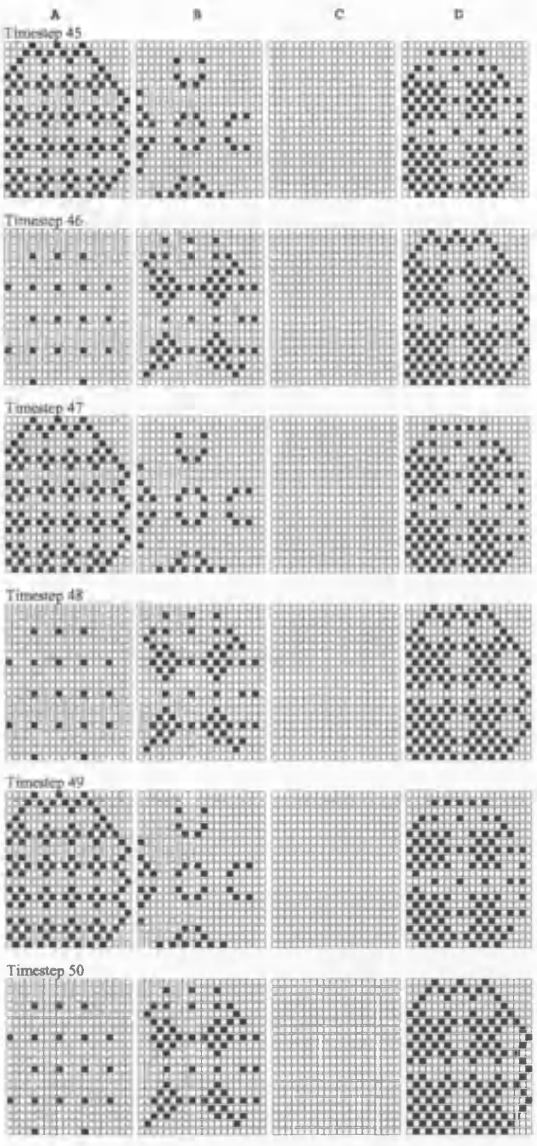
1 A,A>=6,B,B<=3,C,C==0,D,D<=7,->C
2 A,A!=0,B!=1,C!=2,D==6,->A
3 A,A<=3,B<=0,!C,C>=6,D!=2,->C
4 A!=1,!B,B>=6,C,C<=3,D,D==1,->C
5 A<=4,B>=5,!C,C>=4,D,D>=6,->C
6 A,A==2,B!=7,C>=7,!D,D<=5,->C
7 A,A!=6,B,B==4,C==4,D,D!=7,->A
9 A>=0,B,B<=3,!C,C<=0,D<=0,->C
10 !A,A>=0,B,B>=0,C<=0,!D,D==3,->A
11 A,A>=7,B>=2,!C,C>=6,D!=3,->B
12 A,A<=6,B,B<=2,C<=6,D<=0,->A
14 A!=6,B,B!=5,!C,C==5,!D,D==2,->D
16 A==4,B,B>=4,C==0,D,D!=1,->B
17 A<=1,B,B==5,C,C>=2,D>=7,->C
18 A<=2,B<=5,C<=0,!D,D>=5,->C
19 A==0,!B,B==5,C!=6,D,D>=7,->B
20 !A,A!=7,B>=3,C==2,D!=6,->D

Appendix D5: Protein map showing the development of a solution from Experiment 8A(3)




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
A
B
C
D
Timestep 27
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35



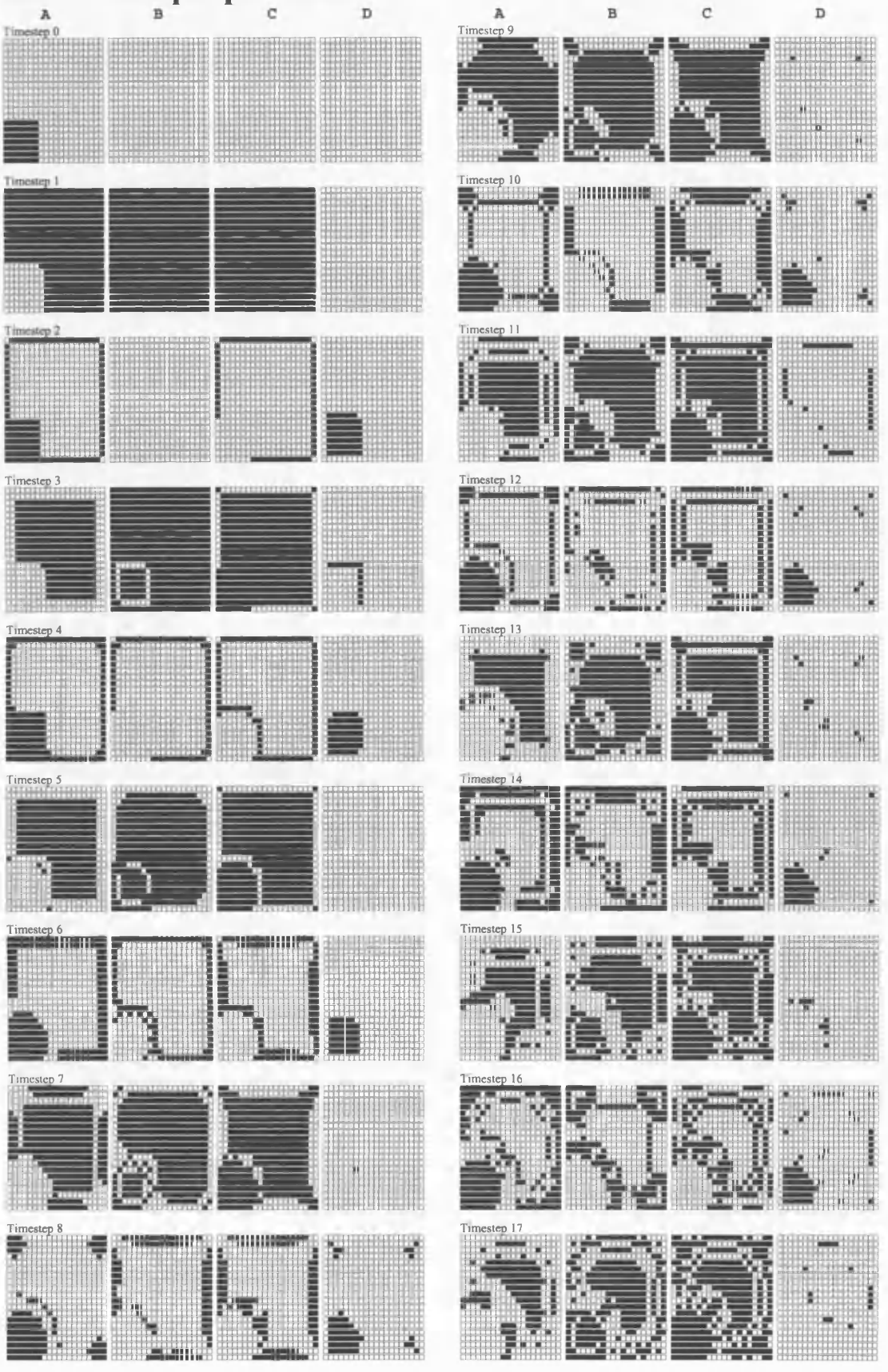


Rules Fired:	Firing Count:
2 A,A<=7,B<=1,!C,C<=4,D<=3,->D	2358
6 A>=0,B<=5,C!=1,D>=3,->D	2905
9 A==1,!B,B==0,C>=0,D==0,->A	394
12 A<=0,!B,B<=3,C==0,D,D<=1,->A	2238
16 A==2,B>=1,!C,C<=2,D,D!=4,->A	68
17 !A,A!=7,B==0,C!=5,D,D>=0,->B	1989

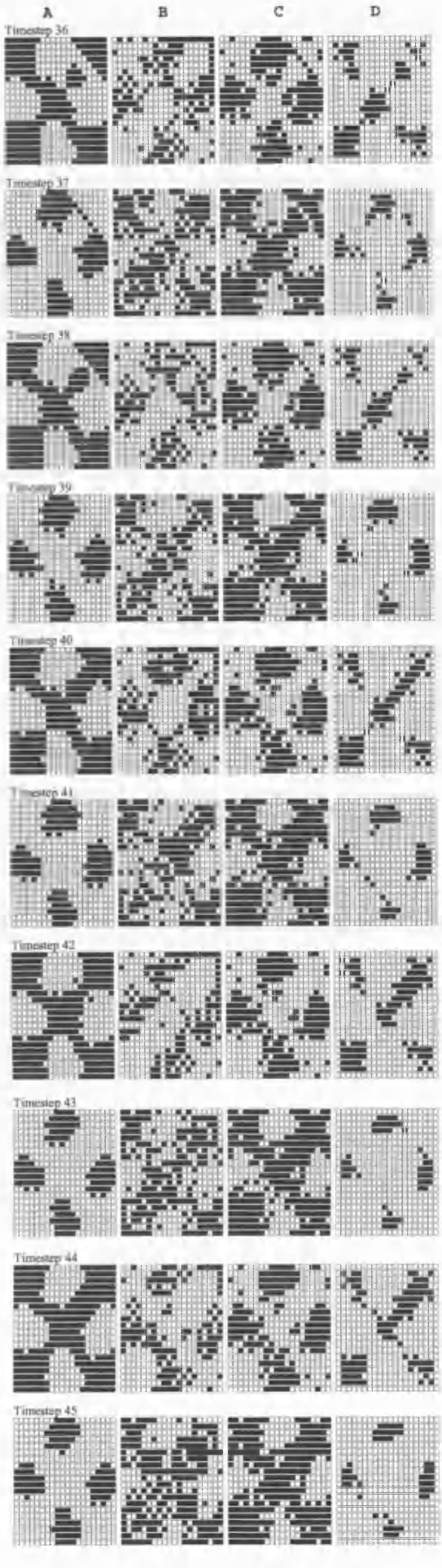
Rules Not Fired:

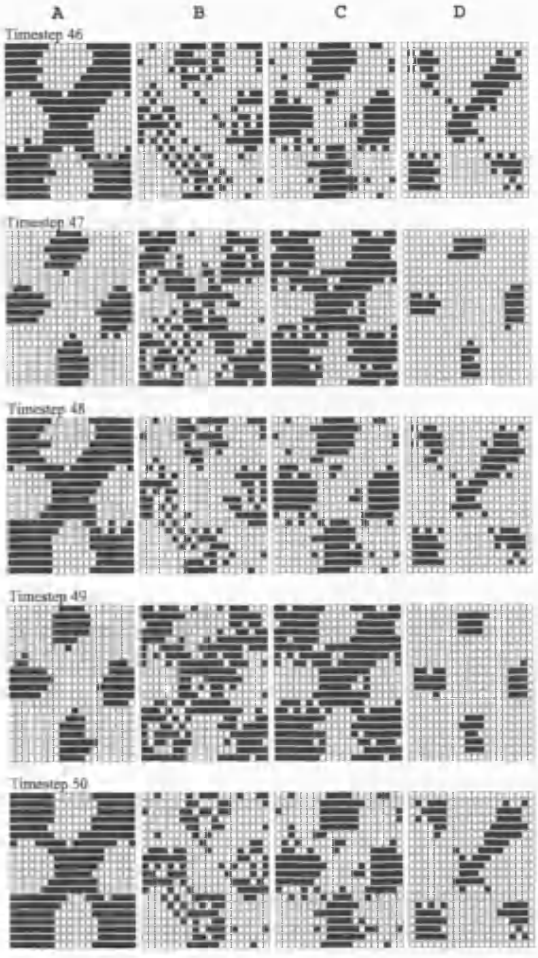
1 A,A!=7,B<=1,C>=3,!D,D==5,->B
3 !A,A!=0,B>=2,C==2,!D,D<=2,->D
4 A,A>=6,!B,B!=3,C,C==0,!D,D!=3,->C
5 !A,A==1,B<=6,C>=5,!D,D==5,->B
7 A==5,B,B==7,!C,C!=2,D<=4,->D
8 A!=5,!B,B==1,!C,C>=4,D!=4,->B
10 A<=5,!B,B>=4,!C,C==1,D>=5,->C
11 A!=4,!B,B<=5,C!=1,!D,D>=6,->A
13 A,A>=1,B==0,C==4,D,D>=3,->A
14 A,A<=5,!B,B!=5,C,C<=1,D==7,->D
15 !A,A==7,B>=0,!C,C==0,D==7,->B
18 !A,A<=5,B!=5,C,C==2,D!=5,->D
19 !A,A!=2,B!=4,!C,C>=1,D<=7,->D
20 A>=1,B<=4,C>=4,D>=4,->D

Appendix D6: Protein map showing the best solution found to the 7x7 cheque problem.




A
B
C
D
Timestep 18
Timestep 19
Timestep 20
Timestep 21
Timestep 22
Timestep 23
Timestep 24
Timestep 25
Timestep 26
A
B
C
D
Timestep 27
Timestep 28
Timestep 29
Timestep 30
Timestep 31
Timestep 32
Timestep 33
Timestep 34
Timestep 35





Rules Fired:

		Firing Count:
1	!A,A<=2,B!=6,C,C==4,D!=3,->D	2412
3	A,A>=0,!B,B==4,!C,C<=1,D<=5,->C	13
4	A!=0,B==3,C,C!=5,D<=4,->C	700
7	A<=3,B>=3,C<=2,D<=1,->B	1119
8	!A,A==0,B>=0,C<=4,D!=3,->A	7730
9	A,A<=4,B<=4,C==2,!D,D>=3,->B	14
10	A,A>=0,B!=6,C<=1,D,D!=3,->B	1551
11	A!=5,!B,B!=2,C!=4,!D,D<=5,->B	6445
12	A,A>=1,!B,B>=3,C>=4,D==1,->A	2
13	A!=6,B<=3,C,C==3,!D,D!=6,->A	2398
15	A!=4,B,B!=2,!C,C<=7,D<=2,->C	1096
18	A<=6,B<=7,C,C>=0,D,D!=3,->A	130
19	A<=5,B<=4,!C,C!=5,D<=4,->C	9728
20	!A,A!=3,B<=0,C<=5,D>=2,->D	67

Rules Not Fired:

2 A==5,B!=1,!C,C!=7,!D,D>=1,->C
5 A,A<=2,B>=1,C==6,D!=5,->D
6 A>=7,B!=0,C,C<=6,D==6,->D
14 !A,A>=6,B<=5,C==7,D,D!=2,->A
16 A<=5,B,B>=7,C!=2,!D,D>=7,->B
17 A==2,B<=0,C,C==7,D>=2,->A

Appendix E1: List of Publications

Much of the work presented in this thesis has appeared previously in the following publications:

Gordon, T. G. W. and Bentley, P. J. (2005) 'Development Brings Scalability to Hardware Evolution', In: *Proceedings of the 2005 NASA / DoD Conference on Evolvable Hardware,* Washington D.C., U.S.A., 29 June - 1 July, 2005. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 272-279.

Gordon, T. G. W. and Bentley, P. J. (2005) 'Bias and Scalability in Evolutionary Development', In: *Proceedings of the 2005 Genetic and Evolutionary Computation Conference,* Washington D.C., U.S.A., June 25-29, 2005. ACM Press, New York, NY, U.S.A., pp. 83-90.

Gordon, T. G. W. and Bentley, P. J. (2005) 'Evolving Hardware'. In Zomaya, A. (Ed.) *Handbook of Innovative Computing,* Springer-Verlag, Heidelberg, Germany, to appear.

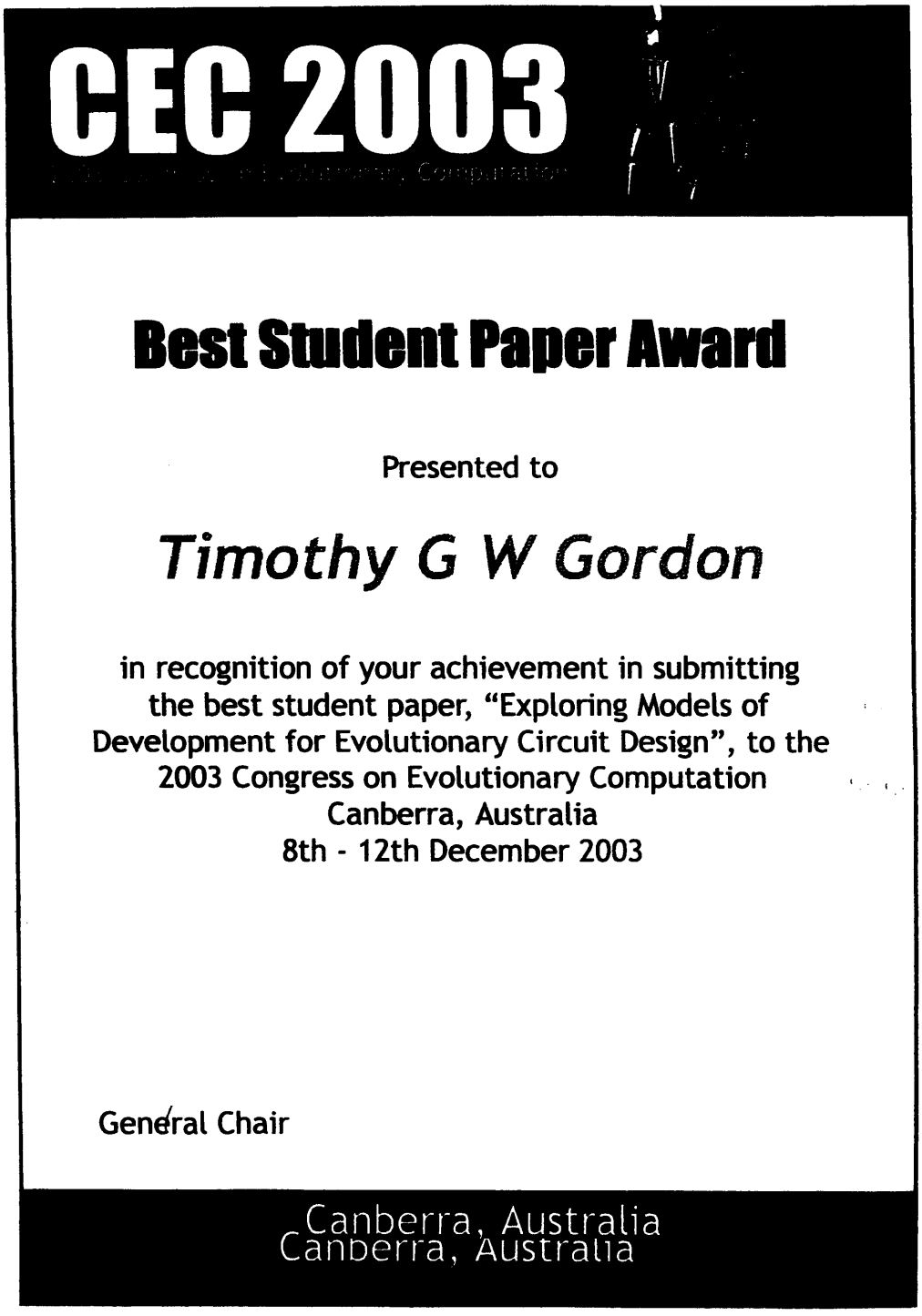
Gordon, T. G. W. (2003) 'Exploring Models of Development for Evolutionary Circuit Design', In: *Proceedings of the 2003 Congress on Evolutionary Computation,* Canberra, Australia, 8-12 December, 2003. IEEE, Piscataway, NJ, U.S.A., pp. 2050-2057.

Gordon, T. G. W. and Bentley, P. J. (2002) 'Towards Development in Evolvable Hardware', In: *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware,* Alexandria, VA, U.S.A., 15-18 July, 2002. IEEE Computer Society, Los Alamitos, CA, U.S.A., pp. 241-250.

Gordon, T. G. W. and Bentley, P. J. (2002) 'On Evolvable Hardware'. In Ovaska, S. and Sztandera, L. (Eds.), *Soft Computing in Industrial Electronics,* Physica-Verlag, Heidelberg, Germany, pp. 279-323.

Bentley, P. J., Gordon, T. G. W., Kim, J. and Kumar, S. (2001) 'New Trends in Evolutionary Computation', In: *Proceedings of the 2001 Congress on Evolutionary Computation,* Seoul, South Korea, 27-30 May, 2001. IEEE, Piscataway, NJ, U.S.A., pp. 162-169.

Appendix E2: CEC Best Student Paper Award



Appendix E2: CEC Best Student Paper Award