Development Brings Scalability to Hardware Evolution

Timothy G.W. Gordon and Peter J. Bentley
Department of Computer Science
University College London
Malet St, London WCIE 6BT, UK
{t.gordon,p.bentley}@cs.ucl.ac.uk

Abstract

The scalability problem is a major impediment to the use of hardware evolution for real-world circuit design problems. A potential solution is to model the map between genotype and phenotype on biological development. Although development has been shown to improve scalability for a few toy problems, it has not been demonstrated for any circuit design problems. This paper presents such a demonstration for two problems, the n-bit adder with carry and even n-bit parity problems, and shows that development imposes, and benefits from, fewer constraints on evolutionary innovation than other approaches to scalability.

1. Introduction

It is widely recognized that the inability of evolutionary algorithms to scale to large and complex circuit design problems is a major impediment to their application in the real world [1-4].

Recently researchers have begun to explore how modelling the genotype-phenotype map on biological development might improve the scalability of hardware evolution [4-6]. Although more traditional problem decomposition techniques have been used to improve scalability in hardware evolution to some extent [1, 3], unlike development they often require domain knowledge and decompose the problem using hard constraints on the interactions between sub-problems, thus limiting evolution's opportunity to explore innovative designs [4].

To date, however, development does not appear to have achieved its potential. There are a few instances in the literature that demonstrate development can enhance scalability for toy problems [7, 8]. These are pattern generation problems that show a high correlation between changes in phenotype and changes in fitness. This is quite unlike circuit evolution, where the relationship between changes in a circuit design and their effect on fitness can be highly non-linear.

In fact there is currently very little empirical evidence to suggest that development can improve scalability for traditional circuit design problems. For instance in [6] Miller used development to evolve one bit full adders and parity generators for up to 5 bits, but has not presented the evolutionary development of any larger circuits. Similarly in [4] Gordon and Bentley applied development to the evolution of two bit adders but failed to evolve fully functional solutions. In the analogue domain Koza et al. have shown instances where development allows design re-use [9] but have not studied scalability in detail.

This paper shows that an improved version of the developmental model of [4] can evolve fully functional two bit adders. It goes on to present a series of experiments that clearly demonstrate that development can enhance the scalability of hardware evolution using two benchmark circuit design problems, the n-bit adder problem and the even n-bit parity problem.

The largest of the evolved adders presented here 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 [10]. This example relied heavily on the use of additional techniques to decompose the problem into several independently evolved sub-problems, hence searched a more constrained problem space. 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 [11-13].

Section 2 of this paper presents a summary of the developmental model used here. Section 3 presents the evolution of a fully functional two bit adder. Section 4 presents experiments that demonstrate development

enhancing hardware evolution's scalability for the n-bit adder problem. Section 5 presents similar experiments for the even n-bit parity problem. Conclusions are drawn in Section 6.

2. Developmental Model

The developmental model used here consists of two layers: a protein layer that models biological development and an architecture layer that maps the product of development to a circuit, which is subsequently evaluated intrinsically using a Xilinx Virtex FPGA [14]. Each layer is now briefly described.

2.1 Protein Layer

The protein layer is responsible for providing a mechanism by which evolution can reuse design innovations that it has already discovered. This is the primary means by which development can enhance scalability [4, 9]. It is identical to the Outer Totalistic developmental model presented in [15], where full details of the model and the design decisions that lie behind it can be found. It consists of a set of rules called the *protein rule set*, and a two dimensional non-toroidal array of cells. The protein rule set describes how the contents of the cells in the cellular array alter during of development. Each cell in the array contains up to four proteins, A, B, C and D that define the cell's state. Development occurs over a series of discrete timesteps. At each timestep the proteins present in every cell at the next timestep are set by comparing the protein rule set to the proteins currently present in each cell, and activating the rules that match the cell's contents. When a rule is activated it generates a protein. The protein rule set models DNA transcription, the process at the heart of biological development's generative ability.

2.1.1 Protein Layer Rule Structure

An example of a single protein rule is shown in Figure 2.1. Each rule consists of 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 the cell itself must contain (11) or not contain (00) for the rule to activate, with the other two bit combinations representing don't care terms. The second term is a five bit condition that defines how the contents of the neighbouring cells 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 3 bits define a protein concentration that the operator acts upon. This concentration is measured by summing the amount of the protein generated by the cell's Von Neumann neighbours. The postcondition of the rule consists of two bits that define which protein is generated if the rule is activated.

For development to begin forming useful patterns it is necessary to begin with a set of simple yet inhomogeneous protein starting conditions from which further inhomogeneity can develop.

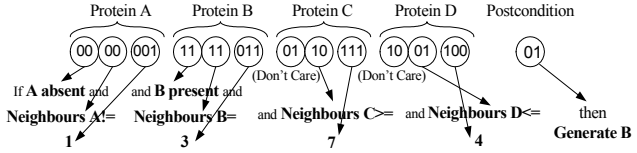

Figure 2.1: The Outer Totalistic Protein Rule

2.2 Architecture Layer

The architecture layer is similar to that used in [4], but it is applied to a simpler architecture. It consists of a virtual FPGA, a second set of rules called the *architecture rule set*, and a set of counters that are used to monitor the behaviour of the protein layer at each developmental timestep. At the end of development the counters determine how changes in the contents of the cells during development are mapped to the architecture layer. Once this mapping is complete the virtual architecture is automatically mapped to a Xilinx Virtex FPGA for evaluation.

The virtual architecture consists of a 2D array of virtual configurable logic blocks (CLBs). It is essentially a simplified model of the Virtex architecture [14]. This architecture is overlaid onto the array of cells in the protein layer. Thus each CLB in the architecture layer corresponds to a unique cell in the protein layer. Each CLB contains two 3-input lookup tables (LUTs) and 3 inputs that drive both LUTs. The source of each input is selected from the LUT outputs of the CLB's Von Neumann neighbours. The CLB also contains a set of counters, one for each configurable element in the CLB. They are used in conjunction with the architecture rules to map activity in the protein layer to changes in the CLB's configuration. This process is explained below. The protein rule set and the architecture rule set are simply concatenated to make up the chromosome.

2.2.1 Mapping Activity to Configuration Changes

An example of an architecture rule is shown in Figure 2.3. The precondition is identical to the precondition of a protein rule. Hence the activity of each architecture rule in each CLB is determined by the proteins present and/or absent in its corresponding protein layer cell. The postcondition of the rule consists of six bits that specify one of the counters

discussed above. Each counter (hence each unique architecture postcondition) corresponds to a single reconfigurable element in the virtual CLB. There is a counter for each of the 16 LUT minterms (eight for each of the two LUTs) and a counter for each of the 24 possible input sources (eight for each input).

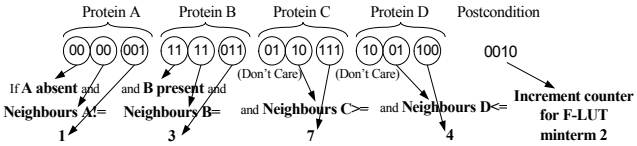

Figure 2.3: The Architecture Rule

Each time an architecture rule is activated during development the counter specified in the rule's postcondition is incremented. At the end of development, the counters are used to determine the structure of the circuit. First each counter that corresponds to a LUT minterm is queried. If the counter value is greater than a predefined threshold value then the minterm configuration bit to which it corresponds is set high. Should the counter value be below the threshold value, the minterm configuration bit is set low. Following LUT configuration the inputs are then configured, but in a slightly different way. For each input, all counters that correspond to one of its possible input sources are queried. The counter with the highest value, i.e. the input source that has been most active throughout development for that input, is selected as the input source. Once the circuit design is determined it is mapped to a Xilinx Virtex XCV400 FPGA for evaluation using the Xilinx JBits API [16].

3. Evolution of a Two Bit Adder

Before conducting extensive scalability experiments the model was tested to determine whether relatively small circuits could be evolved. The initial task was to evolve a two bit adder with carry: a circuit that mapped the five inputs A0, A1, B0, B1 and Cin to the three outputs Sum0, Sum1 and COut in accordance with the two bit adder with carry truth table [17]. Fitness was measured as the total correct output bits across all input combinations, which gives a maximum fitness of 96.

A 2x4 CLB array was evolved using the developmental model described. The problem inputs were provided in cells surrounding the evolved area as in Figure 3.1, which also shows output positions. The chromosome consisted of 20 protein rules and 30 architecture rules, yielding a length of 1620 bits. At the beginning of development some simple starting conditions were introduced: the south western cell was set as if protein A had been generated at developmental timestep *t-1*, and the south eastern cell was set as if protein B had been generated at timestep *t-1*. Development was carried out for 30 timesteps. Table 3.1 shows genetic parameters. Each individual was evaluated five times and its worst fitness selected.

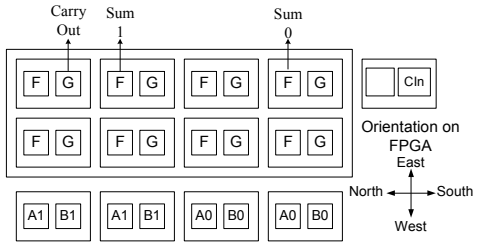

Figure 3.1: Diagram showing where inputs were provided and outputs were drawn on the evolved area

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

Table 3.1: Genetic parameters for all experiments

3.1 Results and analysis

50 evolutionary runs were carried out. The results are shown in Table 3.2, with finesses scaled to 100. 26% of the runs evolved optimal adders, It should be noted that the search space contains circuits with unclocked feedback between CLBs hence it is different from and perhaps richer than the traditional digital design space. Examples and analyses of the circuits evolved are presented in the following section.

Mean Fit. of Best Solns. ($\overline{f}_{best}$)	Std. Dev. of Best Solns. (σ_{best})	Best Fitness / Max Fitness	Optimal Solution %
85.69	12.19	96/96	26

Table 3.2 Results from 50 runs of 2 bit adder evolution

4. Scalability and the n-bit Adder Problem

To date there is little direct evidence that the scalability of hardware evolution can truly be enhanced by development. This section demonstrates such an enhancement clearly, for a real hardware evolution problem. It presents a series of experiments that evolve n-bit adder with carry circuits, ranging from a one bit adder up to a seven bit adder.

4.1 Experimental Setup

The experiments were carried out using the same developmental and evolutionary models and parameters as described above. The task was to evolve an n-bit adder where n varied from 1 to 7. The area evolved for each experiment was a *2n* x 2 array of virtual CLBs. Inputs and outputs for the four smallest problems are shown in Figure 4.1. Again the aim was to evolve a circuit that mapped the inputs to the outputs

in accordance with an n-bit adder with carry truth table [17]. Fitness was again measured as the total correct output bits across all input combinations, which gives a maximum fitness of $(n+1) \times 2^{2n+1}$. 50 runs were conducted for each problem size.

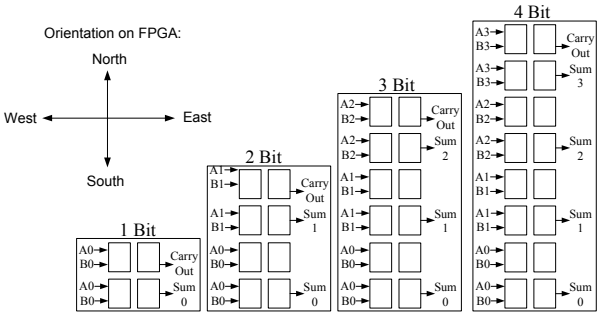

Figure 4.1: Layout of evolved areas

The experiment was repeated using a naïve 1:1 representation that mapped a gene to each component of the evolved array. The representation for one cell is shown in Table 4.1. Unlike the developmental system, the chromosome length of the naïve system varies with array size, from 100 bits for a 2x2 cell array (1 bit problem) to 700 bits for a 14x2 array (7 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 4.1: Naive representation for the adder problem.

4.2 Results and Analysis

The experimental results are shown in Table 4.2 and suggest that for small problems (1 to 3 bit) the proportion of runs reaching optimal fitness is greater with the naïve system than the developmental system. However the proportion decays much more gently with increasing problem size for the developmental system than for the naïve system. This means that for larger problems (4 bit and above) development outperforms the naïve system, and the performance differential increases with problem size. This can be seen more clearly in Figure 4.2, which shows a plot of the percentage of runs with optimal fitness against problem size for both the naïve and developmental systems along with a line of best fit.

Figure 4.2 suggests that for both systems the relationship between fitness and problem size might be linear. The coefficient of determination, r^2, represents the fraction of fitness variability that is explained by problem size variability assuming a linear model [18]. A perfect line has a value of 1, and random data 0. This was calculated for both sets of data in Figure 4.2 to test whether the data was linear and gave values of 0.918 for development and 0.898 for the naïve trend. When calculated for the naïve trend the data for the 6 and 7 bit problem were excluded, as the proportion of optimal runs had already decayed to its minimum. The measured r^2 values are reasonably high, suggesting that the trend might be linear.

Adder Size / Bits	Optimal Solution % Develop.	Optimal Solution % Naïve	Mean Best Fitness Develop. $Dev\ \overline{f}_{best}$	Mean Best Fitness Naïve $Nv\ \overline{f}_{best}$	Std. Dev. Best Develop. $Dev\ \sigma_{best}$	Std. Dev. Best Naïve $Nv\ \sigma_{best}$
1	32	100	87.88	100	11.81	0
2	26	48	85.69	95.44	12.70	4.95
3	18	28	85.80	92.48	13.50	6.19
4	14	4	82.13	88.58	13.57	5.74
5	12	0	82.61	82.99	12.30	5.39
6	10	0	83.87	79.59	12.50	5.37
7	8	0	82.09	76.61	12.37	4.81

Table 4.2: Results of the adder scalability experiments showing the mean of the best fitnesses scaled to 100%

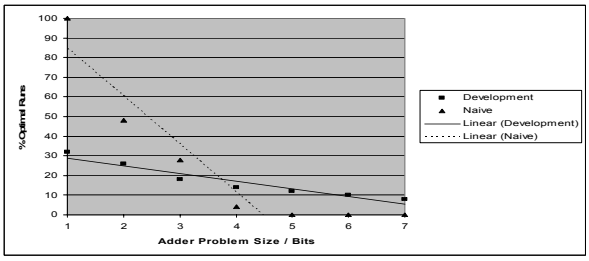

Figure 4.2: Plot of the percentage of runs reaching optimal fitness against problem size.

From an engineering perspective the percentage of optimal runs is a useful value to examine, as it is vital for real-world circuits to operate perfectly. However as this value decays to a minimum value over the studied problem sizes, raw fitness (scaled to 100 to allow comparison between the problem sizes) was used to provide further evidence of scalability.

If evolution scaled poorly with problem size, a strong negative correlation between problem size and performance would be expected. Hence one means of detecting a significant difference in scalability between the naïve and developmental systems is to compare the correlations of the two systems. Plots of fitness against problem size are shown for both developmental and naïve systems in Figures 4.3(a) and (b) respectively.

The Spearman correlation coefficient is a test for correlation that does not assume normally distributed data and can range from -1 to 1 [18]. It was calculated for problem size against fitness using the data from Figures 4.3(a) and (b). Using this method the correlation coefficients were calculated to lie between -0.28 and -0.07 for the developmental trend and between -0.88 and -0.82 for the naïve trend, with 95% confidence. The large difference in coefficients and high confidence suggests that there is a large difference between the rates of decay for both systems and that it

is highly significant.

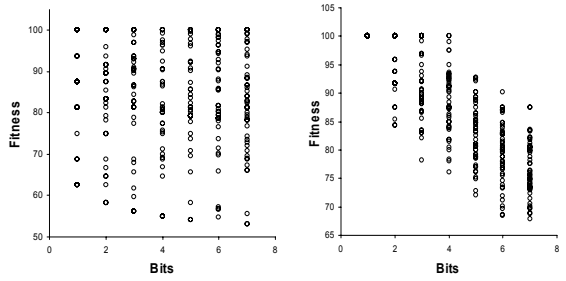

Figure 4.3(a) and (b): Plots of fitness normalised to 100 against problem size for the developmental system (left) and the naïve system (right) for all 50 runs

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. As the percentage of runs that achieved optimal fitness had decayed to its minimum for some problem sizes it was decided that the mean fitness of the best solutions might be more plausible to analyse statistically. The r^2 coefficients of determination calculated with this data were -0.98 and 0.70 respectively, suggesting that a linear model fits well enough for linear analyses to be realistic.

A plot of the difference between $Dev\,\overline{f}_{best}$ and $Nv\,\overline{f}_{best}$ for each problem size is shown in Figure 4.4, along with the least squares line of best fit.

The Pearson correlation coefficient [18] for this data was 0.979, again 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 when applied to the two bit adder with carry problem.

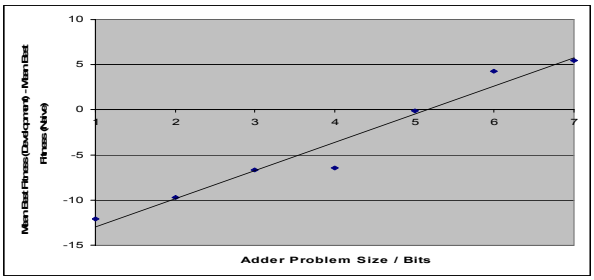

Figure 4.4: Plot of the difference between $Dev\,\overline{f}_{best}$ and $Nv\,\overline{f}_{best}$ against problem size.

4.3 Circuit Analysis

The evolved circuits exhibit several features that reinforce the argument that development can aid scalability, that it can do so without imposing hard constraints associated with traditional designer-imposed abstraction methods, and that this can lead to the discovery of innovative designs.

4.3.1 Modularity, Reuse without Hard Constraints

The most striking feature of the evolved designs is the high level of component reuse. Figure 4.5 shows the first optimal circuit evolved for the two bit problem, and is typical of the optimal solutions evolved. Cells 1, 2, and 5-7 assumed identical inputs, which were generated by the activation of the same two architecture rules in each of these cells. The only cell with a unique set of inputs, Cell 4, shares two inputs with the majority of cells, which generated by the same rules as for the other cells. Generation of the third input was inhibited by the initial conditions.

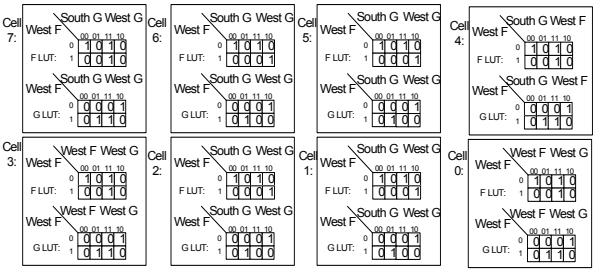

Figure 4.5: K-Maps of an evolved optimal two bit adder

Hence there are different levels of reuse at the cellular level: most cells use a common set of inputs generated by identical rules. A single cell has a unique set of inputs. This can be interpreted 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 rule set. There are also several rules common to both sets of logic.

4.3.2 Design Innovation

Of the traditional adder designs familiar to the authors the only one that would fit within the evolved array is a ripple-carry adder. Examination of the inputs to each cell of the circuit in Figure 4.5 reveals that signals pass east and north, much like a ripple-carry adder. Although there was no explicit fitness bias towards doing so, the circuit generates and uses a traditional first stage carry out term 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 typical of a ripple carry adder, although terms equating to the Generate functions and the inverse of the Propagate functions of a carry look-ahead adder 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 adder problem space is extremely well known to traditional designers. So it is likely that evolution would discover designs similar in style to traditional adders. However it is important to realise that the evolution has managed to discover these circuits without any 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 and rejected.

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 way as a traditional designer. If not, then it is likely to search areas of space that are not searched by traditional circuit design techniques, which could be of great benefit when searching for innovative designs, and present an advantage over other mechanisms have been proposed to improve scalability including function level evolution [1] and partitioned or incremental learning [3, 10]. A common theme of these mechanisms is that the problem must be decomposed by a designer (or arbitrarily) and that this strict decomposition might steer evolution 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. Larger adders can be constructed simply by adding full adder modules to the most significant end of the adder. A larger adder cannot be constructed from the evolved solution presented here by translating a copy of the circuit to a new 2x4 array to the north of the current circuit. Neither can the circuit be cleanly decomposed into the two stages of a traditional ripple-carry adder by considering the southern four cells as a the first stage and the northern four cells as the second stage, as a carry signal is not the only signal to pass between these sets of cells. None of the four optimal two bit solutions studied could be considered a general solution in this way. So it seems that development's bias towards design reuse comes at the expense of the partitioning of communication between modules as a traditional designer might do.

4.4 Seven Bit Adders

Figure 4.6 shows the first optimal circuit evolved for the seven bit problem, which was verified using a logic simulator. Again it shows a high level of reuse: only three distinct LUT configurations are used. Again the designs cannot be easily decomposed into a series of full adders, suggesting that evolution does not merely use development to decompose the problem in a traditional manner.

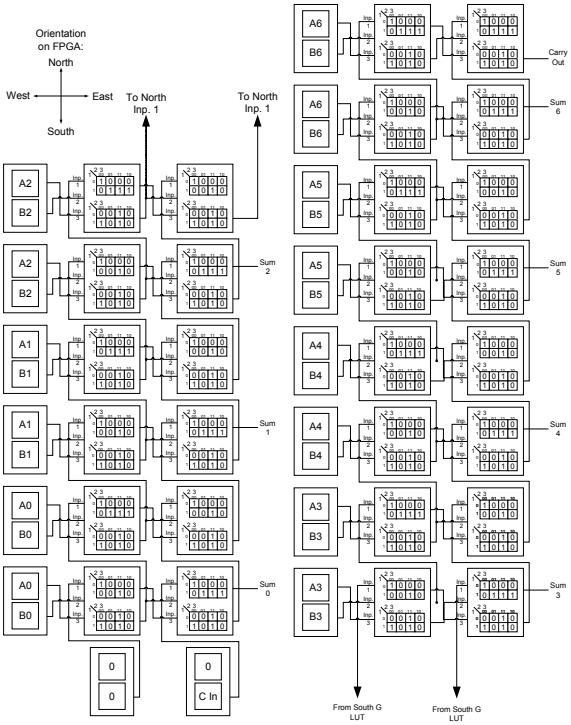

Figure 4.6: The first optimal seven bit adder evolved

However this evidence is 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. 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 was 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 (scaled to 100) from 20 runs of incremental evolution against problem size are shown in Figure 4.7 along with the non-incremental experimental results for comparison.

The results show that when evolution is biased towards decomposing the problem along traditional lines, scalability is not nearly as great as when evolution is free to decompose the problem as it sees fit. This suggests that when there is no bias towards a traditional decomposition, evolution decomposes the

problem in a non-traditional way, and that it has some benefit to evolvability. Hence not only does a developmental approach provide additional opportunities for innovation over techniques that rely on decomposition [1][3] or mechanisms that combine development and decomposition such as [10], but also the scalability of the design process can be greater.

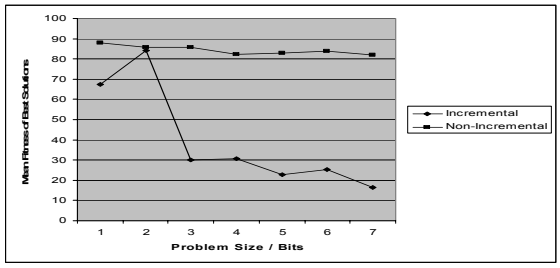

Figure 4.7: Mean (best fitnesses) against problem size for incremental and non-incremental evolution

Anecdotal evidence from the evolutionary history suggests why this might be. For many runs the first fitness improvement over a circuit with a fixed output was to make a pass-through connection between input B1 and output Sum1. As development is biased towards a high level of design reuse it is possible that this biased evolution towards searching designs that connected many of the circuit inputs to the outputs, and helped towards the discovery of the feedforward design abstraction, which is useful for this problem.

5. Scalability and Even n-bit Parity

Section 4 presents strong evidence that scalability can enhance hardware evolution. However this was only demonstrated for the n-bit adder with carry problem. Hence although the evidence detailed above is strong, its scope, thus its significance to the research community is limited. With this in mind, experiments similar to those presented in section 4 were carried out for the even n-bit parity problem.

This is a benchmark problem that is popular with evolutionary computation researchers [11, 12]. An even parity generator generates the modulo 2 of the summed input bits. It has also been a popular target for those exploring scalability [11,13] as it is easily scalable, simply by increasing the number of bits in the word for which parity is generated. Also just as with the adder problem traditional designs tend to be modular, so it is known that the problem space contains modular solutions for development to exploit.

5.1 Experimental Setup

This section provides details of experiments to evolve even n-bit parity circuits, ranging from a 2 bit generator to a 12 bit generator. The experiments were carried out using the same model of development, chromosome, evolutionary algorithm, genetic and developmental parameters as used for the adder experiments. An n x 2 array of virtual CLBs was evolved. Figure 5.1 shows the inputs and outputs for the first three problem sizes. Task was to evolve a circuit that mapped the inputs to outputs in accordance with an even n-bit parity truth table [17], where n ranged from 2 to 12. Fitness was measured as for the non-incremental experiments of section 4, by summing the total correct output bits across all input combinations, giving a maximum fitness of 2^n. 20 runs were conducted for each problem size. The experiment was then repeated using the same naïve representation used for the adder experiments.

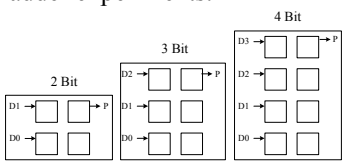

Figure 5.1: Layout of the evolved areas for the first six parity problem sizes

5.2 Results and Analysis

Table 5.1 shows the experimental results, and Figure 5.2 shows a plot of the percentage of runs reaching optimal fitness against problem size for both the naïve and developmental systems.

Parity Size / Bits	Max Fitness	% Optimal Runs (Development)	% Optimal Runs (Naïve)	Mean Best Fitnesses (Development) ($\overline{Dev\ f_{best}}$)	Mean Best Fitnesses (Naïve) ($\overline{Nv\ f_{best}}$)
2	4	75.00	100.00	93.75	100.00
3	8	90.00	95.00	96.25	97.50
4	16	85.00	20.00	94.38	60.00
5	32	70.00	0.00	91.88	50.00
6	64	90.00	0.00	96.33	50.15
7	128	90.00	0.00	96.29	50.05
8	256	65.00	0.00	88.22	50.02
9	512	85.00	N/A	96.25	N/A
10	1024	80.00	N/A	93.44	N/A
11	2048	80.00	N/A	95.49	N/A
12	4096	80.00	N/A	95.00	N/A

Table 5.1: Results of the even parity scalability experiments

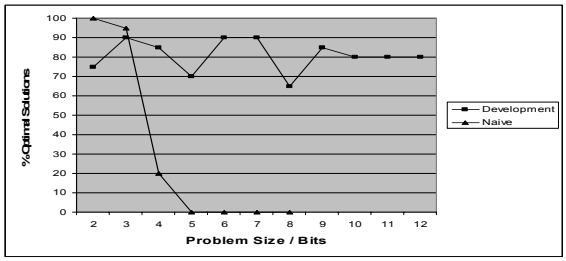

Figure 5.2: Percentage of optimal solutions found using development and naïve systems for the parity problems

They suggest that just as with the adder problem, for small parity problems (2 and 3 bit) the percentage of

runs reaching optimal fitness is greater with the naïve system than the developmental system. However this percentage decays rapidly for the naïve representation, whereas there is little evidence of a trend towards decaying percentages for the developmental system as the problem size increases. (The Spearman correlation coefficient calculated using data from all the developmental runs suggested a small negative correlation, but with only 58.2% confidence, meaning that that zero or positive correlation is within the margin of error.) For larger parity problems (four bit and above) development outperforms the naïve system, and the performance differential again appears to increase with problem size. Hence just as with the adder problem the results suggest that evolutionary performance using development scales better with problem size than performance with the naïve system.

6. Conclusions

It is vital that the scalability problem is tackled if hardware evolution is to rival traditional techniques, and development is one potential solution. However to date it has never been shown that development can actually enhance scalability for hardware evolution.

This paper has presented such a demonstration. A series of experiments compared evolution's performance evolving adder circuits using both development and a naïve mapping, for a range of problem sizes. Analysis showed the performance of the naive system decayed faster than the performance of the developmental system as problem size increased.

Analysis of the circuits revealed a high degree of design reuse. It was also shown that while they contained elements of traditional designs, they differed in many respects suggesting that development has not prevented evolution from exploring non-traditional areas of design space that might be rich in good solutions for other problems.

It was also demonstrated that biasing evolution towards a standard decomposition of the problem lowered performance, suggesting that evolution benefits from the freedom to discover and apply its own design abstractions, a feature not present in other approaches to scalability.

A second set of scalability experiments was conducted for the even n-bit parity problem. It was again observed that development enhanced scalability, suggesting that developmental techniques might be applicable to a larger set of circuit design problems.

7. References

1. Higuchi, T., et al. *Evolvable hardware at function level*. *1997 IEEE Int. Conf. on Evolutionary Computation*. 1997. Indianapolis, IN, USA: IEEE. pp. 187-192.
2. Vassilev, V.K. and J.F. Miller. *Scalability Problems of Digital Circuit Evolution*. *The 2nd NASA/DoD Workshop on Evolvable Hardware*. 2000. Palo Alto, CA: IEEE Comput. Soc. pp. 55-64.
3. Torresen, J., *A scalable approach to evolvable hardware.* Genetic Programming and Evolvable Machines, 2002. **3**(3): p. 259-282.
4. Gordon, T.G.W. and P.J. Bentley. *Towards Development in Evolvable Hardware*. *2002 NASA/DoD Conf. on Evolvable Hardware*. 2002. Washington DC. p. 241-250.
5. Tufte, G. and P.C. Haddow. *Building Knowledge into Developmental Rules for Circuit Design*. *5th Int. Conf. on Evolvable Systems*. 2003 Trondheim, Norway p. 69
6. Miller, J.F. and P. Thomson. *A Developmental Method for Growing Graphs and Circuits*. *5th Int. Conf. on Evolvable Systems*. 2003. Trondheim, Norway: Springer-Verlag. pp. 93-104.
7. Kumar, S. and P.J. Bentley. *Implicit Evolvability: An Investigation into the Evolvability of an Embryogeny*. Late breaker at *2nd Genetic and Evolutionary Comput. Conf*. 2000. Las Vegas, NV, USA: Morgan Kaufmann.
8. Roggen, D. and D. Federici. *Multi-cellular Development: Is There Scalability and Robustness to Gain? 8th Int. PPSN Conf.* 2004. Birmingham, U.K. pp. 391-400.
9. Koza, J.R., M.A. Keane, and M.J. Streeter. *The importance of reuse and development in evolvable hardware*. *2003 NASA/DoD Conf. on Evolvable Hardware*. 2003. Chicago, IL, USA: IEEE Comput. Soc, Los Alamitos, CA, USA. pp. 33-42.
10. Shanthi, A.P., P. Muruhanandam, and R. Parthasarathi. *Enhancing the Development Based Evolution of Digital Circuits*. *2004 NASA/DoD Conf. on Evolvable Hardware*. 2004. Seattle, USA: IEEE Press. pp. 91-96.
11. Yu, T. and J.F. Miller. *Finding Needles in Haystacks is Not Hard with Neutrality*. *5th European Conf. on Genetic Programming*. 2002. Kinsale, Eire. pp. 13-25.
12. Koza, J., *Genetic Programming II: Automatic Discovery of Reusable Programs* 1994, Cambridge MA: MIT Press.
13. Rosca, J.P. *Towards automatic discovery of building blocks in genetic programming*. *Proceedings of AAAI 1995. Fall Symp. Series. 10 12 Nov. 1995 Cambridge, MA, USA*. 1995: AAAI Press, Menlo, USA. pp. 78-85.
14. Xilinx_Inc., *Virtex 2.5 V FPGA Data Sheet*. 2001: http://direct.xilinx.com/partinfo/ds003.pdf.
15. Gordon, T.G.W. *Exploring Models of Development for Evolutionary Circuit Design*. *2003 Congress on Evolutionary Computation*. 2003. Canberra, Australia: IEEE Press, Piscataway, NJ, USA pp. 2050-2057.
16. Guccione, S.A., D. Levi, and P. Sundararajan. *JBits: Java Based Interface for Reconfigurable Computing*. *2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conf.*. 1999. Laurel, MD: Digital Eng. Institute.
17. Holdsworth, B. and C. Woods, *Digital Logic Design*. 2002, London: Newnes.
18. Sheskin, D.J., *The Handbook of Parametric and Nonparametric Statistical Procedures*. 3 ed. 2003: Chapman & Hall.