Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm

Julian F. Miller
Department of Computing, Napier University,
219 Colinton Road, Edinburgh, EH14 1DJ, UK.
Email: j.miller@dcs.napier.ac.uk,
Telephone: +44 (0)131 455 4305

Peter Thomson
Department of Computing, Napier University,
219 Colinton Road, Edinburgh, EH14 1DJ, UK.
Email: p.thomson@dcs.napier.ac.uk,
Telephone: +44 (0)131 455 4245

ABSTRACT

In this paper we describe experiments which attempt to evolve *digital* electronic circuits whose purpose is to implement real signals. As a convenience we chose to evolve mathematical functions i.e. the square-root and sine. Real numbers in the range 0.00-0.99 are encoded in binary using four bits per decimal place. The chromosome used is exactly modelled on the resources available on the Xilinx 6216 re-configurable Field Programmable Gate Array (FPGA), so that evolved circuit designs may be simply implemented on this target device. We investigated a number of ways of presenting examples to the circuit so that the target function might be learned, and also looked at two distinctly different fitness function definitions.

1. Introduction

There is, at the current time, a growing interest in the possibilities of designing electronic circuits using evolutionary techniques. A number of different approaches have been developed. Koza (Koza, 1992) showed how simple digital circuits could be evolved using Genetic Programming. This moved a step closer to actual hardware implementation when Iba et. al. (Iba, 1997) showed how it was possible to evolve digital circuits by evolving the functionality and connectivity of interconnected AND, OR, and NOT gates for intended use on a programmable logic array device (PLA). The Swiss group at EPFL developed a cellular automaton in hardware using Xilinx 6216 FPGAs which was able to carry out global synchronisation tasks despite the fact that the cells behaviour was determined locally (Goecke, 1997). Thompson (Thompson, 1997) showed how it was possible using a genetic algorithm to directly evolve configuring bit strings for the Xilinx 6216 chip to create a circuit which would carry out a frequency discrimination task. Other workers have evolved digital circuits with a variety of representations (Higuchi, 1996, Hemmi, 1996, Kitano, 1996, Zebulum, 1997).

Our own early work evolving digital electronic circuits using genetic algorithms has concentrated on arithmetic circuits (Fogarty, 1998, Miller, 1997). Although we showed that it was possible to evolve novel designs for small circuits (e.g. the two-bit multiplier) we recognised how much more difficult it became to evolve circuits with just a modest increase in function complexity, e.g. the three-bit multiplier. The approach we used was to evaluate every chromosome on the *complete* truth table associated with the desired function. Early work appeared to show that evaluating chromosomes on randomly chosen inputs gave much poorer results. We are still examining this issue in greater detail and will report our findings in due course. However, if one desires a perfect design, then it is clear that eventually one must subject the circuit to a test involving the complete truth table. Therein lies the difficulty in attempting to evolve much larger circuits, as the truth table grows exponentially with increasing numbers of input variables. In our previous work we employed a fitness function which calculated the percentage of correct output bits corresponding to appropriate input bits for the complete truth table. Such a low level definition of chromosome fitness might make the genetic algorithm too discriminatory and rule out certain circuits which fulfil the desired function in an approximate way but which at a bit level perform very poorly. Hence, we looked for a definition of fitness which could take into account the closeness of decimal numbers without suffering from the drawback which is encountered when the actual difference between two decimals might be small while the distance between binary strings (Hamming distance) might be very large. A natural place to do this was to attempt to evolve digitally encoded mathematical functions. Analogue circuits for carrying out mathematical functions have already been successfully evolved using Genetic Programming (Koza, 1997). We wanted to see if it was possible to use a reliable platform such as the Xilinx

device to create circuits which are able to handle *real* signals. In other words, could such a device be used to design circuit elements capable of signal processing applications such as robot control, filtering, equalisation etc. Hence the motivation for this paper. One of the advantages of the approach we use here is that we can obtain symbolic equations for any evolved design. This means that the solution may be implemented on *any* digital platform in a reliable way. Such a benefit would not be possible if we were simply training neural networks. In order for digital approximations to be useful, we must ensure that when the circuits are presented with input signals which they have not been trained with, they do not respond in a uncharacteristic manner. We show in this paper that it is indeed possible to evolve robust digital solutions that behave in this way.

In section 2 we describe our new chromosome representation which is essentially an accurate description of the resources available on the Xilinx 6200 series FPGAs, and differs from the previous structure which was modelled on less specific netlists. Although we have not yet employed the chip directly in the execution of our genetic algorithm, we are confident that the new representation will readily translate into the bit strings required to configure the device without violating the routing constraints. Hence, this work employs a simulation of the 6216 part, but in the near future we intend to employ the chip directly and so test the functionality in-circuit. In section 3 we present our results and give our conclusions in section 4.

2. Chromosome representation

Before we discuss the chromosome representation used, it is necessary to understand some key features and constraints which are imposed when we employ the circuit architecture of the 6216 FPGA. The 6216 chip contains a 64x64 array of cells. Inside each individual cell is the circuit shown in figure 1

It is important to note that the interpretation of the signals {N, S, E, W, N4, S4, E4, W4} is always looked at from the point of view of *outputs.* Thus the N input signal refers to the North *output* from the neighbour immediately below the cell in question (i.e. it is the *North going* signal). It is clear from this that the output of each cell may be any direct incoming signal from a neighbour except the input signal coming from the opposite direction to the cell output (e.g. the South-going signal cannot be routed to the North-going output). This effect may be achieved by configuring the Function Unit to act as a buffer (but this rather wastes the functionality of the Function Unit). The inputs N4, S4, E4, and W4 are special connections which allow connections to cells on 4x4 block boundaries. In our chromosome representation we did not incorporate this feature which may become important when evolving circuits which employ a number of 4x4 blocks. Further, we never allow the Function Unit to act as a buffer.

Since we are considering only combinational designs (non-sequential) it is vital to allow only *feed-forward* circuits. To achieve this we chose a cell connection scheme in which inputs are fed into a cell which are East-going or North going only. If the cell is a Multiplexer (mux) then we allow the control input of the mux to arrive from the North (indicated by 'S'). This scheme is shown in figure 3 below:

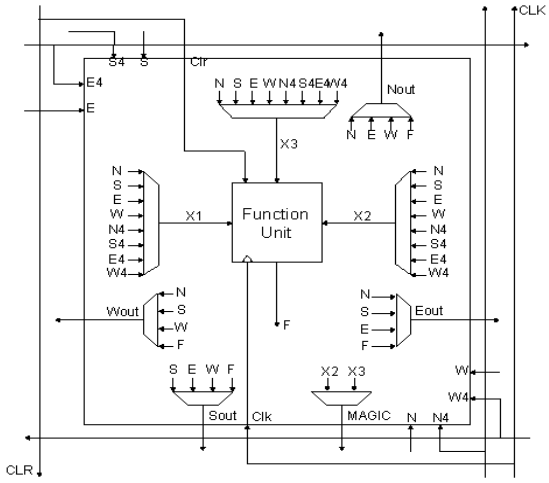

Figure 1 Xilinx 6216 part, internal cell layout

The Function Unit is shown in figure 2.

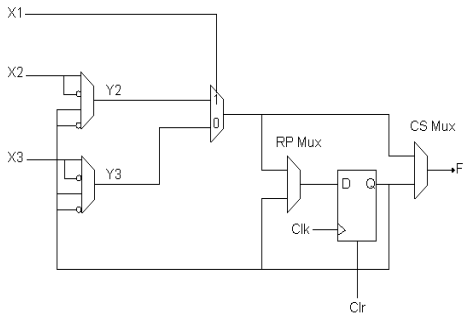

Figure 2 The functional sub-block of the 6216 logic cell

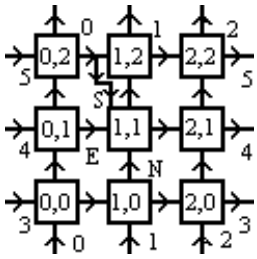

Figure 3 The feed-forward connection of cells

In figure 3 the cells are numbered according to their column and row position with the origin at the bottom left hand corner of the array. All arrows pointing towards (outwards from) a cell represent inputs (outputs). The primary inputs connect to cells on the leftmost column and lowest row. The primary outputs exit the cells which are

located on the topmost row and the rightmost column (in this case cells in column and row two). The cells are allowed to be one of the following types,

Table 1 Allowed functionalities of cells

Gate Type	Function	Gate Type	Function
-4	!A & !C + !B & C	6	A ^ B
-3	A & !C + !B & C	7	A | B
-2	!A & !C + B & C	8	!A & !B
-1	A & !C + B & C	9	!A ^ B
1	0	10	!B
2	1	11	A | !B
3	A & B	12	!A
4	A & !B	13	!A | B
5	!A & B	14	!A | !B

where A and B represent 'E' and 'N' inputs and C represents 'S' input. If a cell at position (col, row) is a mux then the 'S' input is assumed to the 'E' output of the cell located at position (col-1, row+1). If the mux cell is located at column zero then we take the control input of the mux from the primary input located at position (-1, row+1). In such a scheme cells in the top row of the cellular array are not allowed to be multiplexers.

The chromosome has four parts; the functional chromosome, the routing chromosome, the input chromosome, and the output chromosome. The functional chromosome is a set of integers representing the possible cell types as indicated by gate type in table 1. The routing chromosome consists of pairs of integers in the range 0-2, there being as many pairs as there are cells in the chosen cellular array. The pairs of integers represent the relationship between the cell outputs and the cell inputs according to the following table 2.If a routing component is 2 then the corresponding cell output is a function of the cell inputs. The particular function being determined by the cell functionality as indicated in the functional chromosome. Thus, the routing chromosome may ignore the corresponding gene in the functional chromosome if its value is set to 0 or 1. The input chromosome has #rows + #columns elements. Each element can take any integer from 0 to #primary_inputs + 1. If the element of the input chromosome is 0 or 1 then the corresponding input signal in the cellular array is assumed to be logical 0 or logical 1. The output chromosome has #primary_outputs elements which represent the places in the cellular array from which the primary outputs are to be taken. To illustrate the interpretation of the chromosome consider the following example, which represents a chromosome for a 3x3 array of cells as seen in table 3. For the sake of argument imagine that the target function is a 1-bit adder with carry. This has three inputs A, B, Cin and two outputs S and Cout.

Table 2 Relationship of routing chromosome components and cell output.

North	East	North output	East Output
routing component	routing component		
0	0	East input	North input
0	1	East input	East input
0	2	East input	Functional output
1	0	North input	North input
1	1	North input	East input
1	2	North input	Functional output
2	0	Functional output	North input
2	1	Functional output	East input
2	2	Functional output	Functional output

Table 3 Example chromosome for 3x3 array

Functional Part	2,9,11,12,-1,6,14,4,-3
Routing Part	0,1,2,1,1,2,2,2,2,2,0,0,1,1,1,0,2,2
Input part	2,3,0,4,1,3
Output part	5,1

We read the chromosome in the following order (0,0) - (2,0), (0,1) - (2,1) and (0,2) - (2,2) as in figure 3. The inputs and outputs are also read in as shown in figure 3. Thus the cell at (1,1) is a multiplexer of type - 1 (see Table 1). Its outputs as indicated by (2,2) are routed out to North and East. The inputs on the bottom row are A, B, '0', and along the left edge, Cin, '1', B. The Outputs S and Cout are connected to the top row middle cell and top right hand corner cell (east output) respectively.

In the genetic algorithm we used uniform crossover with tournament selection (size 2). Winners of tournaments are accepted with a probability of 0.7. The routing chromosomes are all initialised to 2; 0 and 1 are only introduced by mutation, this is found to be more effective than random initialisation. We use a fixed mutation rate which can be expressed as the percentage of all genes in the population to be randomly altered. The breeding rate represents the percentage of all chromosomes in the population which will be replaced by their offspring.

3. Results

We attempted to evolve circuits which would carry-out the square root function and the function *0.5(sin(x)+1),* in the range 0.0 - 0.99. We only attempted to evolve these functions to an accuracy of two decimal places. Each integer after the decimal point is represented by four bits. Thus in terms of a digital circuit there are 8 inputs and 8 outputs. To calculate the fitness of chromosomes we present a number of examples and then compare the circuit output as expressed as a real decimal with the correct result. For many of the experiments we used 20 examples from a total of 100. In some cases these examples were

randomly generated so that the same chromosome might encounter radically different example sets each time it was evaluated. To be more exact we considered two different fitness function definitions (a) and (b). In (a) we counted the percentage of correct output bits for the total correct output bits corresponding to 20 *fixed* examples. The examples being the numbers 0.00, 0.05, … ,0.95 expressed in binary. The correct values of the two mathematical functions were then calculated for these examples and expressed in binary. The second method of fitness assessment (b) was obtained by calculating the mean of the absolute differences between the correct real number values and the real number values as calculated by the circuit corresponding to the chromosome. The fitness was normalised so that a score of 100.0 represented no errors in circuit output (correct to two decimal places). In the case of fitness function (b) we examined a number of ways of generating the examples which would be presented to the circuit (i) 20 fixed examples 0.00 - 0.95, (ii) 20 randomly chosen examples, and (iii) 20 random examples with frequencies proportional to the average absolute gradient of the target function in the ranges 0.0 - 0.09, 0.1 - 0.19, …, 0.9 - 0.99. In all the figures below the numbers on the left-hand side of the caption indicate the fitness as calculated in the GA, either type (a) or (b), the figures on the right-hand side indicate the fitness calculated for all possible examples (100). Figure 4 shows three approximations to the square root function obtained with fitness function (a).

The x-axis denotes input numbers and the y-axis the circuit output. The smooth curve represents the actual square root function for comparison. The population size was 50 and the GA was run for 10,000 generations. The mutation rate was 1%, and the breeding rate 100%. The worst result of the ten runs is shown in figure 4 (i), an intermediate result is shown in (ii) and the best result shown in (iii). In figures 4 to 6 this pattern of (i) worst, (ii) intermediate and (iii) best is repeated. The results vary quite markedly. We expected that when the circuit was presented with inputs, between those of the example set, the output result would be almost random. After all none of the chromosomes had ever been tested on these numbers and the fitness is being assessed in terms of correct binary output bits *not* how close the decimal output is to the correct value. In figure 4(iii) the circuit is much more correct at positions 0.00, 0.10, …, 0.90 than at 0.05, 0.15, …, 0.95. In intermediate positions the output is fixed. There isn't an obvious explanation for this fact. Figures 4 (i) and (ii) are closer to our expectations as there are regions where the output fluctuates wildly. It is interesting to note how the real fitness can be quite different from the apparent fitness.

Figure 4(b) shows three final results from ten runs with parameters as before. Once again we see much fluctuation in the calculated square root values for figures (i) and (ii) due to the uncertainty in calculation at points between the fixed points at which the fitness was assessed. However the best result is very good though less so in the region 0.0 - 0.1. This was probably due to the rapidly changing gradient of the square root function within this interval.

Figure 5(a) shows results for ten runs with parameters as before but where the fitness function was type (b) described earlier and the fitness of chromosomes were assessed on their performance for 20 randomly generated examples. These being generated each time the chromosome was evaluated. The approximations were considerably more stable and similar in appearance. The best circuit performs just slightly worse than the best shown in figure 4(b)

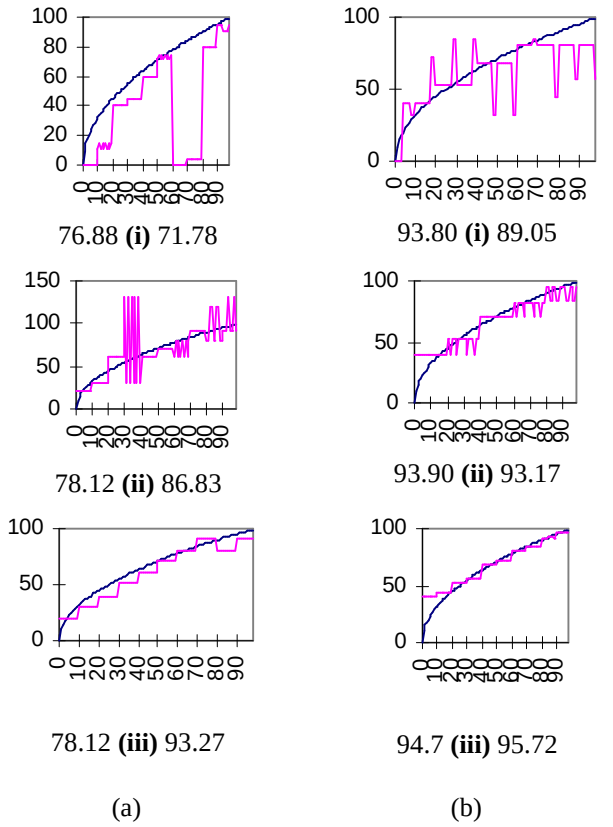

Figure 4 (a) Best circuit outputs for bit-wise fitness function and 20 fixed examples. (b) Best circuit output for mean absolute difference fitness function and 20 fixed examples.

In figure 5(b) we present the results of square root approximation from ten runs (parameters as before) with fitness function (b) and random examples chosen with a frequency proportional to the mean absolute gradient of the square root function. The results are comparable to those shown in figure 5(a), however the approximation to the square root is better in the small argument region.

In figure 6 we present some results for the function *0.5(sin(x)+1)*. The parameters were as before. Clearly this function is much more difficult to evolve within the geometry given (5x5). We intend to carry out further experiments to ascertain how the choice of geometry

affects the GA's ability to evolve the function. It is interesting to note the curious step-like behaviour of many of the graphs.

When we analyse the chromosome which produced the result shown in the graph of figure 5(a) part (iii), firstly we found that from 25 FPGA cells only 12 were required as functional logic gates, the remainder were used only as routing switches. Secondly, the resulting logic equations for the outputs were found to be surprisingly simple, except for the least-significant bit. The equations are as follows:-

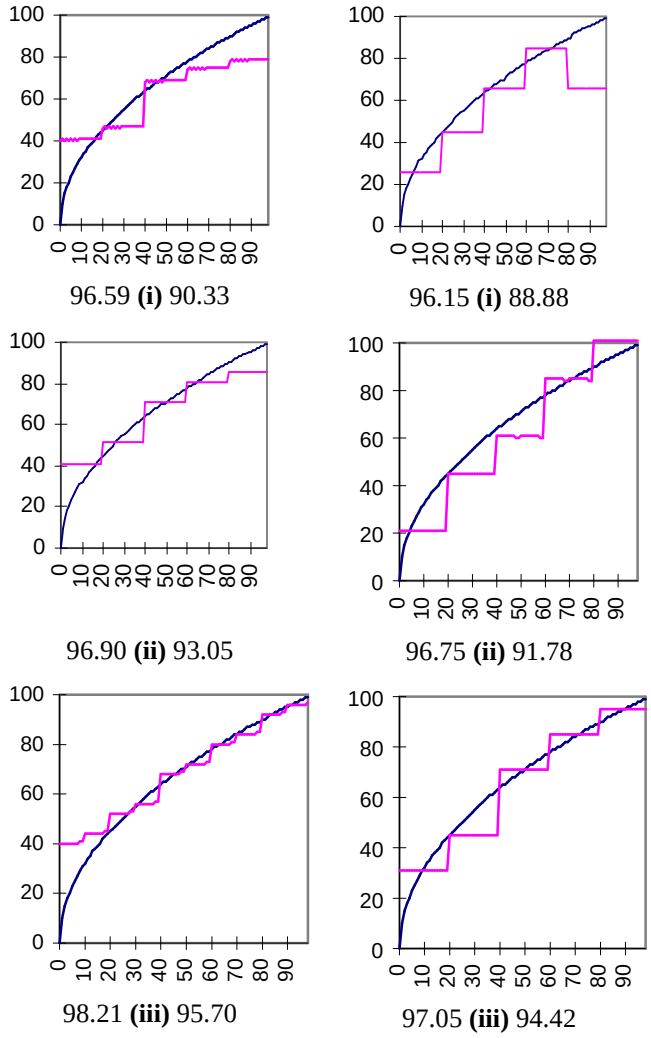

Figure 5 (a) Best circuit output for mean absolute difference fitness function and 20 randomly chosen examples. (b) Best circuit output for mean absolute difference fitness function and 20 randomly chosen examples with frequencies proportional to mean absolute gradient of target function.

$M_3 = m_3$; $M_2 = !m_3$; $M_1 = m_2$; $M_0 = m_1 \wedge m_3$; $L_3 = m_2$; $L_2 = m_0$; $L_1 = m_1 \wedge m_3$ (1)

where m_3 to m_0 are the input bits for the most significant decimal, and l_3 to l_0 are the input bits for the least significant decimal. M_3 to M_0 and L_3 to L_0 are the corresponding output bits. L_0 is not shown because this is an much more complex logical function, but in actual fact contributes very little to the final result. On analysing the chromosome which led to the results shown in figure 4(iii) we found that only 11 cells were required as functional logic gates. Again the resulting logic equations turned out to quite simple and make an interesting comparison with those above.

$M_3 = m_1 m_2 !m_3 \mid !m_2 m_3$; $M_2 = m_1 \wedge m_2$; $M_1 = !(m_1 \wedge m_3)$; $M_0 = m_0$; $L_3 = 0$; $L_2 = 0$; $L_1 = 0$; $L_0 = m_0$ (2)

When we analysed the circuit corresponding to figure 6(iii) we obtained the following logic equations.

$M_3 = 0$; $M_2 = M_1$; $M_1 = m_2 \wedge ((!m_2 \& !m_3) \mid (m_2 \& m_1))$; $M_0 = !m_2$; $L_3 = !m_2$; $L_2 = L_3$; $L_1 = m_1 \mid m_2$;

$L_0 = 1$ (3)

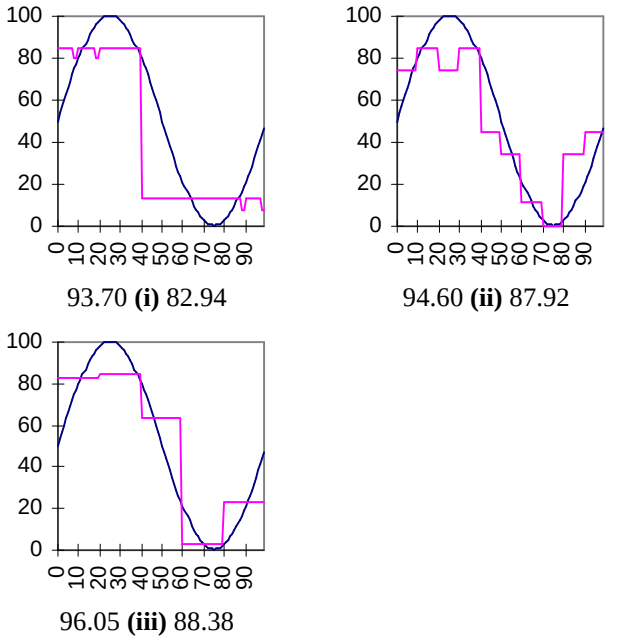

Figure 6 Best circuit output for mean absolute difference fitness function and 20 randomly chosen examples for sine function.

4. Conclusions

In this paper we have examined the evolution of real-valued functions using digital circuits. We have developed a chromosome representation which is an accurate model of the internal structure of the Xilinx 6216 FPGA. We have employed a number of different fitness functions in an attempt to develop good approximations of the target function. We found that an error-based fitness function with random examples produced better, and more stable, results than a direct bit-wise fitness function. However to our surprise we found that we obtained at least one reasonably good solution using the bit-wise representation, and that the circuit to some extent was able to interpolate between the chosen sample points.

One of the advantages of using digital circuits to evolve real numbered real-valued functions is that the evolved circuits can be analysed and equations obtained which describe in the logical terms the transfer function. These equations could of course be implemented on any convenient digital platform. We were greatly surprised by the extraordinary simplicity of the equations we obtained which approximated the square-root and sine functions. It is worth noting that it should be possible to obtain better approximations to the desired function by combining the best elements of individual solutions (e.g. using outputs of both (1) and (2) to produce a better solution), and we are currently investigating this.

Another factor which is very important in the effectiveness of the digital approximation is the chosen geometry (i.e. in terms of number and layout of the FPGA cells). Better approximations may be obtained by choosing alternative geometries. It is interesting to note that the circuits described in equations (3.1)-(3.3) do not make use of the binary inputs corresponding to the least significant decimal input digit (l_3 - l_0). The GA is clearly recognising that these inputs are not contributing significantly to the closeness of the global approximation. It may be that with a larger geometry that these inputs might be made use of.

The aim of this work has not been to accurately build mathematical function circuits but to merely explore firstly the degree to which evolved circuits could approximate the desired real signal response in a stable way. Additionally, we have obtained hitherto unknown logical equations which naturally approximate real numbered functions. We have also explored the effects of the way in which the input decimal quantities are digitally encoded (as this will clearly have a significant bearing upon the circuits that evolve), and we intend to examine this point further.

Finally, this work has convinced us that it is worthwhile, when considering evolved digital systems, to examine only a randomly selected subset of all possible outcomes, rather than the entire set of input combinations. In this spirit, we have returned to re-investigate our previous work using arithmetic circuits in order determine the effectiveness of this approach which greatly reduces the burden of the fitness calculation within the GA. In fact, a number of extremely interesting issues have emerged from these experiments, and we have been given a revitalised hope that we can eventually evolve larger systems using these techniques as a basis.

Bibliography

[A] *Lecture Notes in Computer Science - Towards Evolvable Hardware*, Vol. 1062, Springer-Verlag, 1996

[B] Higuchi T., Iwata M., and Liu W., (Editors), *Proceedings of The First International Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, now published *Lecture Notes in Computer Science*, Vol. 1259, Springer-Verlag, Heidelberg, 1997

Fogarty T. C., Miller J. F., and Thomson P., "Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs" in Soft Computing in Engineering Design and Manufacturing, P.K. Chawdhry,R. Roy and R.K.Pant (eds),Springer-Verlag, London, pages 299-305,1998.

Goeke M., Sipper M., Mange D., Stauffer A., Sanchez E., and Tomassini M., "Online Autonomous Evolware", in [B], pp. 96 -106.

Higuchi T., Iwata M., Kajitani I., Iba H., Hirao Y., Furuya T., and Manderick B., "Evolvable Hardware and Its Applications to Pattern Recognition and Fault-Tolerant Systems", in [A], pp. 118-135.

Hemmi H., Mizoguchi J., and Shimonara K., " Development and Evolution of Hardware Behaviours", in [A], pp. 250 - 265.

Iba H., Iwata M., and Higuchi T., Machine Learning Approach to Gate-Level Evolvable Hardware, in [B], pp. 327 - 343

Kitano H., "Morphogenesis of Evolvable Systems", in [A], pp. 99-107.

Koza J. R., *Genetic Programming*, The MIT Press, Cambridge, Massachusetts, 1992.

Koza J. R., Andre D., Bennett III F. H., and Keane M. A., "Design of a High-Gain Operational Amplifier and Other Circuits by Means of Genetic Programming", in Evolutionary Programming VI, Lecture Notes in Computer Science, Vol. 1213, pp. 125 - 135, Springer-Verlag 1997.

Miller J. F., Thomson P., and Fogarty T. C., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Eds: D. Quagliarella, J. Periaux, C. Poloni and G. Winter, Wiley, 1997

Thompson A., "An evolved circuit, intrinsic in silicon, entwined with physics", in [B], pp. 390 - 405.

Zebulum R. S., Pacheco M. A., and Vellasco M., "Evolvable Systems in Hardware Design: Taxonomy, Survey and Applications", in [B], pp. 344 - 358.