Discovering Novel Digital Circuits using Evolutionary Techniques

Julian F. Miller and Peter Thomson[1]
Department of Computer Studies, Napier University
219 Colinton Road, Edinburgh, EH14 1DJ

Abstract

Modern FPGAs provide a platform for implementation of uncommitted logic arrays which are also, in many cases, reconfigurable. Whilst this allows circuit functionality to be changed in time, it also provides a convenient environment in which to encourage the direct evolution (using genetic algorithms) of those circuit solutions themselves. In this paper we describe experiments which examine the possibility of evolving simple arithmetic and mathematical circuits. We show that it is possible to evolve both conventional cellular designs - such as ripple-carry adders - and also very novel solutions which are suitable for implementation on arrays such as Xilinx 6000 series FPGAs. We also discuss the evolutionary models that are used to achieve this: evolving network connection lists, and then refining to produce a closer simulation of the actual internal structure of the Xilinx architecture. We then go on to examine and discuss the possibility of evolving mathematical functions - such as the square-root directly in combinational logic.

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 [4] showed how simple digital circuits could be evolved using Genetic Programming. This moved a step closer to actual hardware implementation when Iba et. al. [3] 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 [2, 7]. Thompson [8] 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 [5, 9]

2. Designing Arithmetic Circuits using Evolution

In our early work [1, 6] we considered, for each potential design, a geometry (of a fixed size array) of *uncommitted* logic cells that exist between a set of desired inputs and outputs. The uncommitted logic cells are typical of the resource provided on the Xilinx XC6216 FPGA part. An uncommitted logic cell refers to a two-input, single-output logic module with no fixed functionality. The functionality may then be chosen, at the implementation time, to be any two-input logic function.

There are two aspects required to define any combinational logic network. The first is the cell-level functionality and the second is the inter-connectivity of the cells between circuit inputs and outputs. A *chromosome* is defined as a set of interconnections and gate level functionality for these cells from outputs back toward the inputs based upon a numbered rectangular grid of the cells themselves, as in Figure 2.1. This procedure was carried out in such a way that *all individual cell inputs could only be connected to cell outputs which were of a lower number* within this scheme. This is important because it eliminates any possibility of *feedback* connections which would give rise to sequential behaviour. The inputs that are made available are logic '0', logic '1', all primary inputs and primary inputs inverted. To illustrate this consider a 3 x 3 array of logic cells between two required primary inputs and two required outputs.

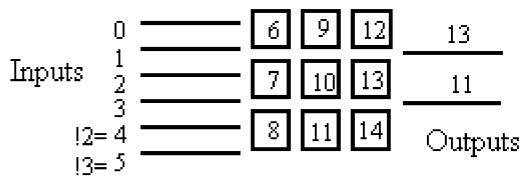


Figure 2.1 A 3 x 3 geometry of uncommitted logic cells with inputs, outputs and netlist numbering.

The inputs 0 and 1 are standard within the chromosome, and represent the fixed values, logic '0' and logic '1' respectively. The inputs (two in this case) are numbered 2 and 3, with 2 being the most significant. The lines 4 and 5 represent the inverted inputs 2 and 3 respectively. The logic cells which form the array are numbered column-wise from 6 to 14. The outputs are numbered 13 and 11, meaning that the most significant output is connected to the output of cell 13 and the least significant output is connected to the output of cell 11. These integer values, whilst denoting the physical location of each input, cell or output within the structure, now also represent connections or *routes* between the various points. Thus, a chromosome is merely a netlist of these integer values, where the *position* on the list tells us the cell or output which is being referred to, while the value tells us the connection (of cell or input) to which that point is connected, and the cell functionality.

[1] {j.miller, p.thomson}@dcs.napier.ac.uk

Each of the logic cells is capable of assuming the functionality of any two-input logic gate, or, alternatively a 2-1 *multiplexer* (MUX) with single control input. In the geometry shown in Figure 2.2 a sample chromosome (to be used in the GA) is shown below:

0 2 -1 1 3 -5 2 4 3 2 6 7 0 8 -10 7 8 -4 6 11 9 6 4 -9 2 11 7 13 11

Figure 2.2 A typical netlist chromosome for the 3 x 3 geometry of Figure 2.1

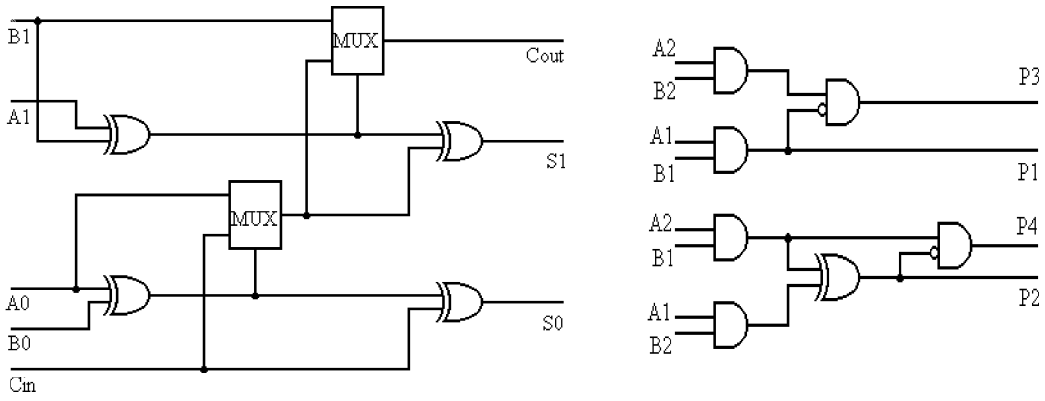
Notice, in this arrangement that the chromosome is split up into groups of three integers. The first two values relate to the connections of the two inputs to the gate or MUX. The third value may either be positive - in which case it is taken to represent the control input of a MUX - or negative - in which case it is taken to represent a two-input gate, where the modulus of the number indicates the function according to Table 2.1 (Gate Type 1) below. The first input to the cell is called A and the second input called B for convenience. For the logic operations, the C language symbols are used: (i) & for AND, (ii) | for OR, (iii) ^ for exclusive-OR, and (iv) ! for NOT.

Gate Type1	Gate Type2	Function	Gate Type1	Gate Type 2	Function
-	-4	!A & !C + !B & C	-4	6	A ^ B
-	-3	A & !C + !B & C	-5	7	A | B
-	-2	!A & !C + B & C	-6	8	!A & !B
-	-1	A & !C + B & C	-7	9	!A ^ B
-	1	0	-10	10	!B
-	2	1	-9	11	A | !B
-1	3	A & B	-8	12	!A
-2	4	A & !B	-11	13	!A | B
-3	5	!A & B	-12	14	!A | !B

Table 2.1 Cell gate functionality according to negative gene value in chromosome.

This means that the first cell with output number 6 and characterised by the triple {0, 2, -1} has its A input connected to '0', its B input connected to input 2, and since the third value is -1, the cell is an AND gate (thus in this case always produces a logical output of 0). Picking out the cell who's output is labelled 9, which is characterised by the triple {2, 6, 7}, it can be seen that its A input is connected to input 2 and its B input is connected to the output of cell 6, while since the third number is positive the cell is a MUX with control input connected to the output of cell 7.

Despite the fact that the method described above was capable of evolving both conventional (Figure 2.3(a)) and novel circuit designs (Figure 2.3(b)), they do not translate *directly* on to the structure of the Xilinx 6216 FPGA part.



(a) Evolved two-bit adder with carry (b) Evolved two-bit multiplier

Figure 2.3 Two evolved circuit designs.

To solve this problem, a new chromosome structure was developed which exactly models the resources provided on the chip. Figure 2.4 shows the structure of this new representation. In Figure 2.4 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).

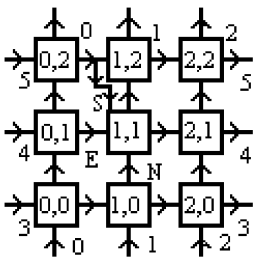


Figure 2.4. The modified cellular array structure for circuits on Xilinx 6216 FPGA.

The cells are allowed to be one of the following types, 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 new chromosome has four parts; the functional chromosome (shown in Table 2.1, Gate Type 2), the routing chromosome, the input chromosome, and the output chromosome. This can be seen in the example shown in Table 2.2 below. The circuit cell layout for this chromosome is seen in Figure 2.5. We are currently attempting to evolve arithmetic circuits using this new representation and our findings will be reported in due course. We have however already explored the use of this representation in the evolution of mathematical functions such as the square root function and the sine function 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.

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

Table 2.2 Example chromosome for 3x3 array

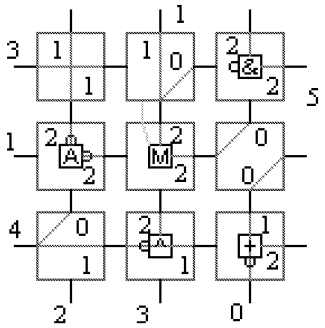


Figure 2.5 Cell layout for chromosome of Table 2.2

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. 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 results for the evolved square root function can be seen in Figure 2.6. 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 2.6 (i), an intermediate result is shown in (ii) and the best result shown in (iii). 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 2.6 (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 2.6 (i) and (ii) are closer to our expectations as there are regions where the output fluctuates wildly. Figure 2.7 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.

3. Conclusions

In this paper we have discussed two possible cellular representations for evolving digital circuits on the Xilinx 6216 FPGA chip. We have encoded these representations in the form of chromosomes and subjected them to a genetic algorithm. With the first representation we showed how it was possible to evolve functionally correct arithmetic circuits and recover known designs (i.e. 2-bit ripple-carry adder) as well as novel circuits (e.g. 2-bit multiplier). In fact we have been able to correctly evolve both the three-bit multiplier and the four-bit adder [6]. The second representation is an exact model of the resources available on the target chip. The chromosome comprised four distinct parts; the functional part which dictated the functions of the cells which would be used, the routing part which described how the signals would be routed between cells, the input part which described how the primary inputs would connect to the edges of the cellular array, and finally, the output part which determined which points on the edge of the array the primary outputs would connect to. We discussed how using the second representation it is possible to evolve digital circuits which approximate mathematical functions. We showed that on occasions some of these circuits were able to naturally interpolate.

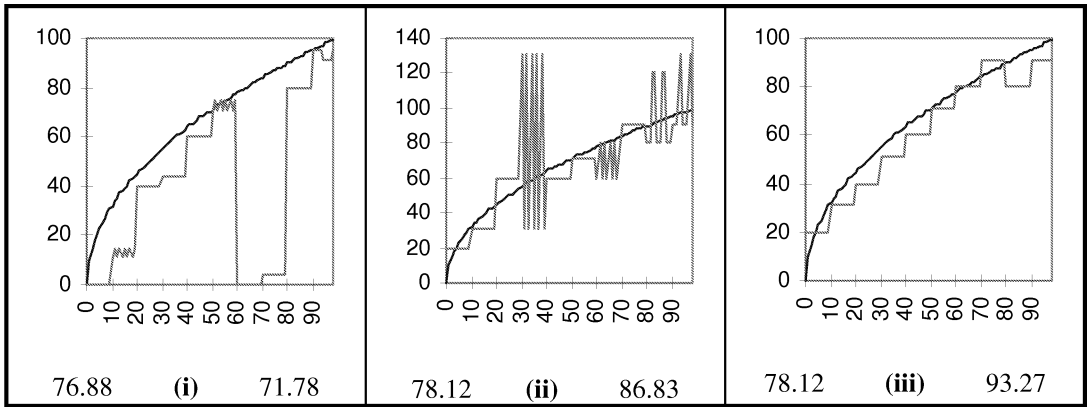


Figure 2.6 Best circuit outputs for bit-wise fitness function and 20 fixed examples.

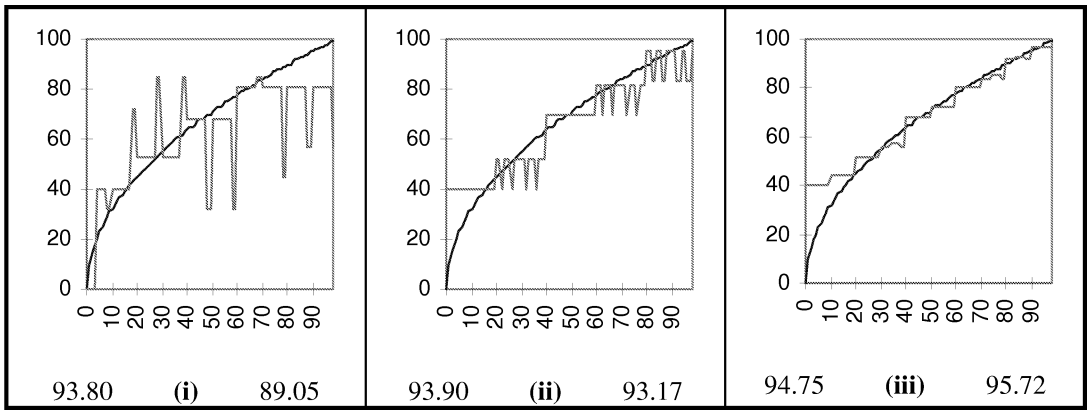


Figure 2.7 Best circuit output for mean absolute difference fitness function and 20 fixed examples (as before).

References

[A] 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.

[1] 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.

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

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

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

[5] 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.

[6] Miller J. F., Thomson P., and Fogarty T. C., "Designing Electronic Circuits Using Evolutionary Algorithms. Arithmetic Circuits: A Case Study", chapter 6, in Genetic Algorithms and Evolution Strategies in Engineering and Computer Science: Recent Advancements and Industrial Applications. Editors: D. Quagliarella, J. Periaux, C. Poloni and G. Winter, published by Wiley, 1997

[7] Sipper M., Sanchez E., Mange D., Tomassini M., Perez-Uribe A., and Stauffer A., "A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems", in *IEEE Transactions on Evolutionary Computation*, Vol. 1, No 1., pages 83-97.

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

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