Evolving Digital Logic Circuits on Xilinx 6000 Family FPGAs

T. C. Fogarty[1], J. F. Miller[1], P. Thomson[1]

[1]*Department of Computer Studies*
Napier University,
219 Colinton Road, Edinburgh

t.fogarty@dcs.napier.ac.uk
j.miller@dcs.napier.ac.uk
p.thomson@dcs.napier.ac.uk

Keywords: Evolutionary electronics, evolvable hardware

Abstract

This paper describes work which attempts to evolve circuit solutions for combinational logic systems directly onto Xilinx 6000 FPGA parts. The reason for attempting to evolve designs direct onto the device is twofold: (i) every circuit has a known functionality and (ii) every circuit must be able to be placed on the chip and then routed. Using evolutionary techniques allows us to consider these two important aspects of design and implementation as a single problem. The paper describes the basic method adopted, using a network list (netlist) chromosome and genes which represent circuit module function, and then discusses some of the results achieved, plus difficulties encountered, and some of the additional problems which still require to be solved in this new and exciting area of research.

1. Introduction

This paper is inspired by both the authors' own attempts to synthesise digital logic circuits by using genetic algorithms to resolve the variable-ordering problem of decision diagram representations [1][2], and by the work of Adrian Thompson who has previously evolved circuit solutions directly on the 6000 family of fine-grained field-programmable gate array (FPGA) devices produced by Xilinx [3].

Thompson's work describes the direct evolution of circuits on to existing hardware components which exploit the underlying analogue structure of a device that was designed for digital applications. This means that solutions which are evolved in this manner cannot be replicated easily from the existing design on to other identical physical parts. Where our approach differs, is that we are attempting to create only digital solutions, and, in this paper, those which concentrate upon finding combinational solutions to existing combinational design problems. However, this is just as important to the designer, as many of the problems that are encountered when attempting to place a synthesised circuit - by which we usually mean minimised in terms of numbers of gates used - on to an FPGA part are due to the automatic place and route (APR) software's inability to find routes on the actual chip for the design implementation. In this paper we have taken a radical new approach to circuit synthesis which addresses this difficulty. We evolve circuit functionality under the strict conditions of specific routing strategies that we know will be acceptable to the existing structure of the target FPGA part.

We also feel that this approach provide a benefit to engineers as we are effectively encouraging the design of novel circuit solutions which, when studied, could lead to a better understanding of design principles, or may even create new principles. In this paper we will demonstrate, in our results, via a small example of how hitherto unknown designs of existing circuits may emerge. The design discovered by the genetic algorithm for this example was surprising in several ways. Firstly, that it was possible to evolve this type of function in such a simple way, and, secondly, the way in which the actual solution deviated from the conventional design.

Evolvable Hardware is a very new topic for research [4][5][6][7] and at present is largely confined to the use of evolutionary algorithms in the synthesis of digital logic at the systems level [4]. There are very few examples where researchers are actually attempting to evolve the functionality of digital circuits, and even fewer where this is being done with the specific implementation platform in mind. In fact, as far as we are aware, Koza [5] and Higuchi [6]

head the only two other groups who have attempted to evolve actual digital circuits at gate level, and only Higuchi mentions implementation issues, and has a particular device in mind - a programmable logic device or PLD. Additionally, the most complex gate-level digital circuit that has been designed using genetic principles, we believe, is Koza's two-bit adder without carry [5].

In the results section of this paper, we will show our evolved designs with 100% functionality for the one-bit, two-bit and three-bit adders all with carry, and the two-bit unsigned multiplier. The latter three designs are by far the most complex digital circuits that have to date been created by purely evolutionary methods with no human design constraints. In the case of the two-bit multiplier, the circuit produced is radically simpler and more efficient than can be obtained by known design methods of minimisation. This would tend to suggest that it is easier for evolution to find simpler solutions than more complex ones.

2. The Evolvable Hardware Method

The method adopted in our technique is to consider, 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. A *chromosome*, represented as a list of integers, may then be created which forms a set of interconnections and gate level functionality for these cells from output back toward the inputs i.e. a network connection list, or *netlist*.

Each of the uncommitted logic cells is capable of assuming the functionality of any two-input logic gate, or, alternatively a 2-bit *universal logic module* (ULM) with single control line. Therefore, for a 4 by 4 geometry, a typical chromosome may look something like this:-

2 1 6 4 7 -13 0 3 -12 0 8 3 0 5 2 4 2 -1 7 1 9 8 1 -10 9 11 -7 4 8 -8 0 15 -4 11 17 -14
11 18 19 11 18 14 17 14 -1 0 15 -4 22 21

Figure 1. A typical netlist chromosome for the 4 by 4 geometry.

Notice, in this arrangement that we split the chromosome up into groups of three integers. This relates to the connection of the two inputs to the gate or ULM. The third value may either be positive - in which case it is the control connection of a ULM - or negative - in which case it is a two-input gate, and the modulus of the number indicates the function according to Table 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. There are only 12 entries on this table out of a possible 16 as 4 of the combinations: (i) all zeroes, (ii) all ones, (iii) input A passed straight through, and (iv) input B passed straight through are considered trivial - because these are already included amongst the possible input combinations, and they do not affect subsequent network connections in cascade.

Considering Figure 1 it can be seen that the first cell, i.e. number 8, indicated by being first in the chromosome list, has its A input connected to input 2, its B input connected to input 1, and its control input connected to input 6. This means that it is a ULM device, and the control is driven by input 6 because the third number on the list is positive. If we consider the second group of three digits, we see that the third integer is negative, -12 in fact. From Table 1, this indicates that this is an OR gate with both inputs inverted, and so the cell inverts inputs 4 and 7 and then ORs these together.

In this way, the entire circuit is defined. Note that, in this example, the two outputs are driven by the outputs of cells 22 and 21 respectively - these are the final two values on the chromosome, and because they drive individual outputs, they do not appear in a group of three integers like all other connections.

This, of course means, that for every circuit, there will inevitably be cells that are not actually connected to any of the outputs. These are redundant for the circuit they define, and are removed when the chromosome is analysed. This analysis is performed after the algorithm has evolved 100% functionality, and is not done as part of the basic procedure.

Table 1. Cell gate functionality according to negative integer value in chromosome.

Negative Integer	Gate Function
-1	A & B
-2	A & !B
-3	!A & B
-4	A ^ B
-5	A | B
-6	!A & !B
-7	!A ^ B
-8	!A
-9	A | !B
-10	!B
-11	!A | B
-12	!A | !B

It should be noted that the chosen netlist structure for the chromosome ensures that:-

1) all circuits are valid combinational logic circuits, and
2) chromosomes may be manipulated in such a way as to guarantee that all subsequent child chromosomes, after crossover and mutation by a genetic algorithm, will also represent valid combinational logic circuits.

It was considered important, in choosing the representation, that it should fill these two criteria as time would be wasted effecting repair to a less robust representation. It is also important in terms of translation of the evolved solutions directly on to the FPGA devices.

The fitness of each chromosome is defined as the percentage of the correct output bits for every input combination of the complete specification file. We do not attempt to evolve 100% functional circuits using an example set, but instead use the entire circuit specification. This is the only reliable method we have to discovered to date. In our work, we do not attempt to carry out an exercise in machine learning by trying to generalise from a limited example set since we only interested in evolving actual fully-functional, practical circuit solutions.

3. Results

We now present the results of some of the arithmetical circuits that we have thus far been able to evolve using the above method.

The circuits that we are looking at are binary adders which incorporate both input and output carry, and unsigned multipliers. Some of the results are extremely interesting when these are compared with the circuits that have been created by designers. Of course, many of these arithmetical circuits have standard forms, particularly the adders which tend to be comprised of cascaded *full-adders* with the option of incorporating additional circuitry to provide a *carry look-ahead* facility. This latter approach tends to make the circuits much more complex to implement in terms of the numbers of gates used. Evolution is capable of developing solutions for these applications where the carry circuitry is not constrained by considering its relationship to the sum, but is treated rather as an independent output of the entire combinational system. This appears to provide several advantages in that the resulting carry circuitry is greatly simplified when compared to that required by a carry look-ahead strategy. Treating the multiplier as the development of a purely combinational transform also appears to give beneficial results in terms of the resources required. In fact, when we evolved the 2-bit unsigned binary multiplier, and compared it with the circuit that would be achieved by conventional design techniques, the result was staggering.

Figure 2 below shows the evolved design for the 2-bit multiplier. This circuit contains only eight logic gates. Compare this with the design achieved by treating this as a straightforward combinational design and using conventional minimisation methods, which requires twenty gates, then this is a significant result!

When one considers that actual Boolean algebra representation of the respective solutions, then one is able to see where the evolutionary approach is scoring over the conventional methods.

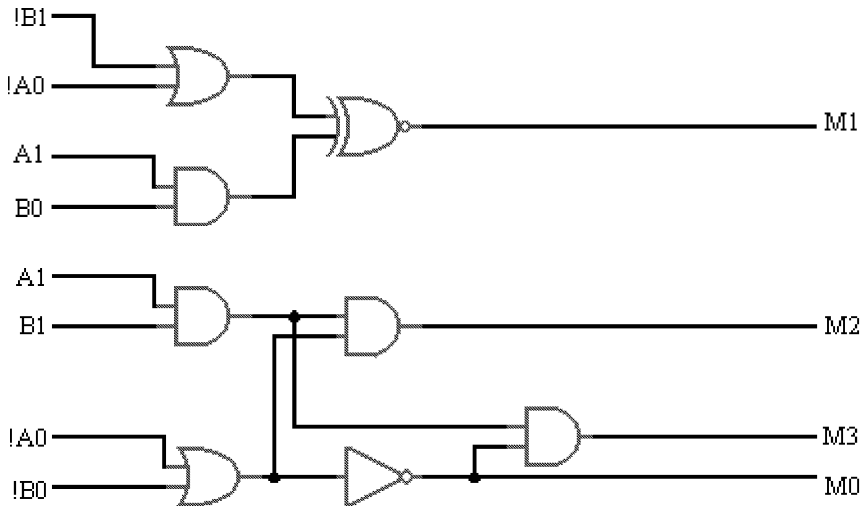

Figure 2. The evolved 2-bit unsigned binary multiplier

$$
\begin{aligned}
M0 &= \overline{\overline{A0} + \overline{B0}} \\
M1 &= \overline{(\overline{B1} + \overline{A0}) \oplus A1.B0} \\
M2 &= A1.B1.(\overline{A0} + \overline{B0}) \\
M3 &= A1.B1.\overline{(\overline{A0} + \overline{B0})}
\end{aligned}
\qquad (1)
$$

$$
\begin{aligned}
M0 &= A0.B0 \\
M1 &= A1.\overline{B1}.B0 + A1.\overline{A0}.B0 + \overline{A1}.A0.B1 + A0.B1.\overline{B0} \\
M2 &= A1.\overline{A0}.B1 + A1.B1.\overline{B0} \\
M3 &= A1.A0.B1.B0
\end{aligned}
\qquad (2)
$$

Equations (1) above show the evolved representation, whereas equations (2) show the solution minimised by *Karnaugh Map* [8]. The K-Map solution conveys a regular sum-of-products representation for the desired output functionality, whereas the evolved solution shows a mixture of both sum-of-product and product-of-sum forms that a designer would find extremely difficult to develop, as there are no rules that tell him/her which parts of the equations to expand and which parts to leave unaltered. This, then, is the difficulty for the conventional minimisation approach: *no such rules exist*. Additionally, in the evolved solution, there is great deal of re-use of the functional sub-blocks that are used to derive particular outputs. Consider, as an example, the derivation of outputs *M0* and *M2*. If *M0* had been merely represented in the AND form i.e. as a single AND gate (from equations 2), then it would not have been possible to re-use this in the derivation of *M2*. This is highly novel. In addition to making the circuit substantially simpler than the conventionally minmised representation, the maximum gate delay is the same i.e. a maximum of three gate delays, which is in itself an improvement upon the conventional cellular design approach to multipliers (using full adders) which incurs a thirteen gate delay.

It is important to note that these evolved designs are created within a fixed geometry of logic cells (where a cell is a two-input gate or 2-1 ULM), and these are pruned down to remove redundancy after the design is completed. In other

words, once the circuit has evolved, there are inevitably cells which contribute nothing to the solution - by either not being connected to any output, or by possessing a non-varying gate output. These are removed by a program which analyses for these conditions and performs the removal automatically. We often find that the final circuits require between 33% to 50% of the original gate resource within the applied geometry. This means that for an 8 by 8 geometry, for example, the solution would probably only require between 21 and 32 of the 64 original cells.

Figure 3 and Figure 4 show the evolved designs for the one-bit and two-bit full adders respectively.

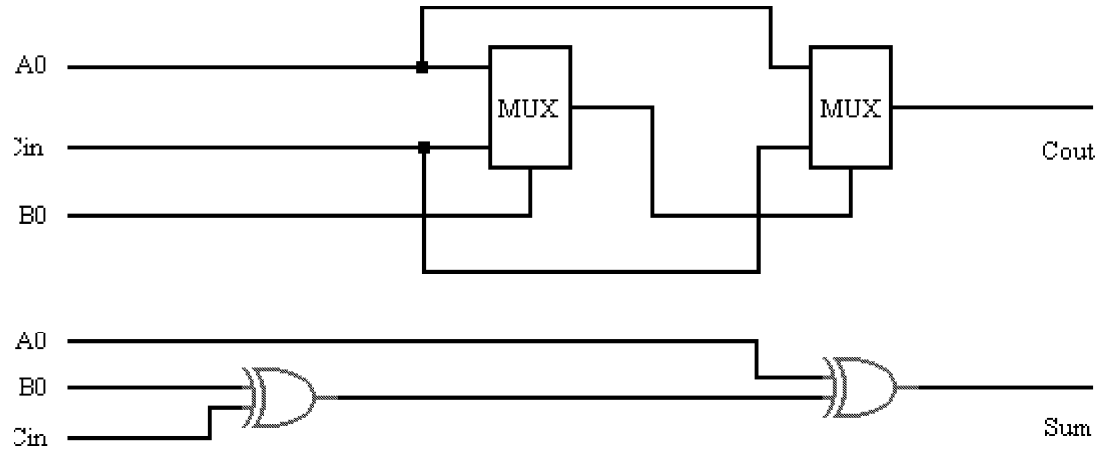

Figure 3. The evolved one-bit full adder with carry

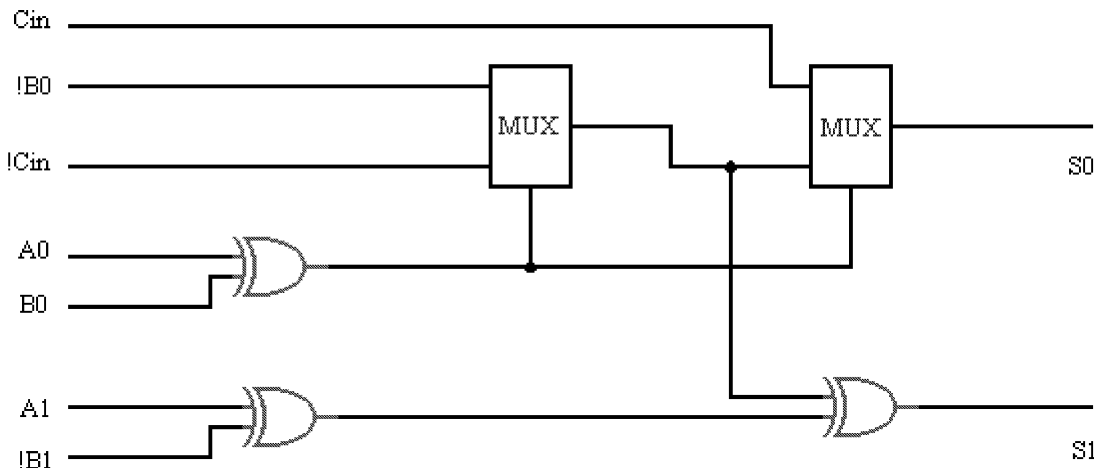

Figure 4(a) The evolved two-bit adder sum

It is interesting to note that the SUM output of the one-bit adder with carry circuit in Figure 3 is identical to that used in designed full adders. However, the carry is radically different. There is no attempt made by evolution to use aspects of the sum derivation - as is done in conventional designs - but rather the carry is developed in an entirely different way. This has the effect of producing a more economical and more elegant circuit solution, and again one which is not intuitively obvious to designers.

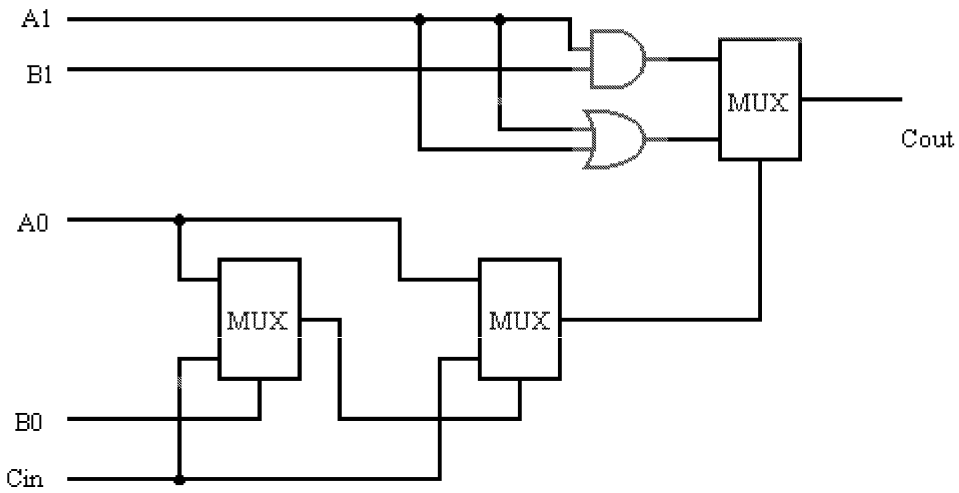

Figure 4(b). The evolved two-bit full adder carry

We were intrigued having evolved the one-bit adder carry to see whether evolution would use this sub-function when evolving a higher adder carry. In other words, we were asking the question as to whether evolution was in fact finding a generic way in which to construct carry circuits for any adder. We found the answer to be in the affirmative as can be seen in Figure 4 where the evolved solution for the two-bit adder with carry is presented. Interestingly, the control for the rightmost ULM is in the form of a one-bit carry circuit. This, then, controls whether the final output is to be the AND of the two most significant bits, or else the OR of these same two bits. This, once again, is a very elegant solution. Additionally, because the sum and carry are now both treated as combinational outputs in parallel, the carry does not suffer from the ripple effect of passing through both the full adders of a conventional design. That is, with the evolved design, the carry is available just as quickly as the sum. Using a carry look-ahead approach, even for this small adder, the conventional design would use 14 FPGA cells in the 6000 family, whereas the evolved design uses only 10. We envisage that this type of saving will probably scale-up significantly when we attempt to evolve larger adder designs.

It should be noted, that in evolving both the one-bit and two-bit adders, we have evolved both circuits as single multiple-output entities and, alternatively, as separate sum and carry circuits. We always found that it was much simpler to evolve solutions using the latter approach. We also found that this approach produced consistently more efficient solutions such as those shown the Figures 3 and 4.

4. Conclusions

In this paper we have presented an evolutionary approach for the design of combinational logic circuits. The method uses a genetic algorithm to evolve both a netlist structure and functionality for logic cells as represented on Xilinx's 6000 family of FPGA parts. The fitness of solutions is measure by their degree of correlation with a desired system functionality.

We feel that this approach to combinational circuit design has a distinct advantage over conventional design methods because: (i) we are not concerned about the application of complex rules to the minimisation of logic, and so there is no requirement for possibly abstract and lengthy implementation of these in computer code, (ii) we know for a fact that our solutions will fit on to the target implementation device, because we are designing with these specific architectural constraints built-in, (iii) our solutions are already routed for the target part, and so potential routing bottlenecks are automatically avoided, and (iv) extremely novel solutions may be discovered which may assist our understanding of design principles.

The points (ii) and (iii) above are particularly important as this removes the necessity for place and route software to be used. This can be both time-consuming and non-productive if the minimised design is unable to be routed due to limitations on the part that were previously unanticipated. We have had personal experience of this with designs

which were perfectly easily minimised by synthesis software, both proprietary and our own algorithms [9], but could not then be routed on to the Xilinx parts. This new approach takes a much more organic view, with the design effectively being grafted on to the part in question.

Clearly, there are difficulties with the evolutionary design of circuits in scaling these designs up to larger systems. Some of the questions that remain unanswered regarding scalability are:

(i) what is the largest circuit that can be evolved using current computer hardware?
(ii) what is the ideal geometry for any given circuit?
(iii) what is the ideal routing constraint?
(iv) can larger systems be realistically constructed from smaller sub-functions?

We are currently addressing some of these questions, and expect to report our findings in due course.

References

[1] P Thomson and J. F. Miller, 1996, "Symbolic Method for Simplifying AND-EXOR Representations of Boolean Functions using a Binary-Decision Technique and a Genetic Algorithm.", *IEE Proceedings on Computers and Digital Techniques*, Vol. 143, No. 2, pp. 151-155.

[2] J. F. Miller, P. V. G. Bradbeer and P. Thomson, 1996, "Experiences of using Evolutionary Techniques in Logic Minimisation", *1st Online Workshop on Soft Computing.*

[3] A. Thompson, 1996, "Silicon Evolution", in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel and R . L. Riolo Eds. Cambridge, MA, 1996, pp. 444-452, The MIT Press.

[4] E. Sanchez and M. Tomassini, Eds., 1996, *Towards Evolvable Hardware*, Vol. 1062 *of Lecture Notes in Computer Science, Springer-Verlag*, Heidelberg.

[5] J. R. Koza, "Genetic Programming", 1992, The MIT Press, Cambridge, MA.

[6] H. Iba, M. Iwata and T. Higuchi, 1996, "Machine Learning Approach to Gate-Level Evolvable Hardware.", Vol. 1062 *of Lecture Notes in Computer Science, Springer-Verlag*, Heidelberg.

[7] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe and A. Stauffer, 1997, "A Phylogenetic, Ontogenetic, and Epigenetic View of Bio-Inspired Hardware Systems*.", IEEE Transactions in Evolutionary Computation.*

[8] A. E. A. Almaini, 1994, "Electronic Logic Systems.", pp. 305-308*, 3rd Edition, Prentice-Hall International.*

[9] P. Thomson and J. F. Miller, 1997, "Comparison of AND-XOR Logic Synthesis using a Genetic Algorithm against MISII for Implementation on FPGAs.", submitted to *GALESIA '97.*