

A Novel Genetic Algorithms for the Automated Design of Performance Driven Digital Circuits

Ben. I Hounsell and Tughrul Arslan

Department of Electronics and Electrical Engineering

The University of Edinburgh

King's Buildings

Mayfield Rd

Edinburgh EH9 3JL

(Int +44) 131 650 5665 (enquiries)

Ben.Hounsell@ee.ed.ac.uk; Tughrul.Arslan@ec.ed.ac.uk

Abstract- The authors present a genetic algorithm for the design of high performance arithmetic circuits for evolvable hardware applications. A distinct feature of the algorithm is its ability to directly evolve and evaluate circuits in a *hardware description language* (HDL), within a novel environment termed the *Virtual Chip*. Because the Virtual Chip evolves circuit structures within a HDL, detailed simulation and analysis of each circuit is possible with any technology specific component library. This feature allows accurate analysis of performance issues such as timing and area.

The paper describes the genetic algorithm and the hardware evaluation environment, and provides results with a number of benchmark arithmetic circuits evolved under different performance driven timing and area constraints. Our results reveal that the genetic algorithm is able to exploit the flexibility provided by a novel chromosome architecture, and utilise a combination of primitive gates and macro components from a component library, in order to produce circuits which operate well within timing restrictions. The validity of our results are further supported by comparing the performance of functionally equivalent circuits generated using standard high-level design methodologies.

1 Introduction

Advances this decade in silicon technology have enabled design engineers to examine new methods of generating circuit designs. One such method has become known as evolvable hardware (EHW), which considers the automated design of digital systems using both software simulation and programmable hardware technologies.

Although a range of evolutionary algorithms have been applied to EHW applications including; *Genetic Programming*, *Evolutionary Strategies*, and *Evolutionary Programming*, the most dominant approach involves the use of *Genetic Algorithms* [1, 2].

Automated circuit design attempts to re-define the methodology by which electrical circuits are developed. Traditional techniques utilise a *top down* or *compartmentalised* design methodology by which complex systems are broken down into small sub-systems and assigned (usually) to a num-

ber of design groups. Evolvable hardware, inversely, approaches the design problem as a whole system generating a *'black box'* of the completed circuit. This technique is referred to as a *bottom up* design methodology. The only information an EHW framework has is that presented to it by the design engineer. For automated digital circuit design this is usually in the form of a boolean logic table.

Circuits generated via evolvable hardware are evaluated by one of two methods: *extrinsic* evaluation (software simulation), and direct *intrinsic* evaluation by which a circuit is transferred directly into silicon and then evaluated. Intrinsic evaluation has become feasible due to recent advances this decade in programmable hardware technology such as PLDs and FPGAs (*Programmable Logic Devices* and *Field Programmable Gate Arrays*). The successful generation of digital circuits using both extrinsic and intrinsic evaluation has demonstrated the potential of EHW for automated circuit design. It has also highlighted a number of inherent problems, particularly associated with intrinsic evaluation [3], and raised a number of questions as to how current EHW techniques can be further improved.

Evolvable hardware for automated digital design favours software based, or *extrinsic* evaluation, due to the simplicity of its implementation and the ease in which evolved circuits can be examined once a solution is found [4]. Using this approach only the final solution is downloaded onto a reconfigurable device. The majority of frameworks which employ extrinsic evaluation use a technology independent net-list to model a digital circuit undergoing evolution [4, 5], although representations which more closely model the characteristics of a particular hardware platform have also been presented [6].

Much attention in EHW research is given to the design of arithmetic circuits as they provide the foundation blocks for larger DSP (Digital Signal Processing) applications. With the advent of faster and larger FPGAs, resulting from advances in silicon technology, and the move towards deep sub-micron technologies, designers are under increasing pressure to provide high performance DSP circuits which take advantage of these new platforms. *The result are circuits which must operate under critical constraints imposed by high density, and the domination of interconnect capacitance* [7]. Innovative research into using EHW for DSP design has resulted in

the development of arithmetic circuits from sequential adder structures to more complex 3-bit parallel multiplier designs.

The automated design of arithmetic circuits is not trivial. Each possible circuit solution for a given task lies within a search space. The search space is defined by the number of different component building-blocks presented to the framework, the number of logic elements used to generate the circuit, and the application for which the circuit is being evolved. Evolutionary algorithms are employed within EHW as they provide a non-heuristic investigation of what is potentially a very large search space. Successful solutions are often made more difficult to find as the output response must be exact, for instance as part of a sequence of operations such as memory mapping. This differs from other types of circuit which instead approximate a specified analogue transfer function.

It is a combination of these factors which has resulted in the difficulties experienced by researchers to evolve circuits larger than those detailed. In addition, one draw-back of extrinsic evaluation is that little information is processed in terms of how accurately a system is modelled, as in most cases the additional detail required has not been integrated into the software. This inhibits the development of high performance DSP circuits where timing and area constraints are of great importance, and therefore *must* be accounted for in EHW applications. The authors are currently unaware of applications using genetic algorithms which incorporate timing and area performance considerations into the fitness of a circuit undergoing evaluation.

Perhaps the most important feature in evolvable hardware applications is the evolutionary algorithm. The algorithm needs to be able to generate quality circuit solutions, which in turn are implemented on a programmable device. The quality of the evolutionary algorithm is then characterised by its speed, flexibility, and ability to effectively search a solution space to produce circuits which are able to meet multiple performance criteria.

This paper presents both the genetic algorithm and the hardware evaluation environment, and provides results with a number of benchmark arithmetic circuits evolved under different performance driven timing and area constraints. Our results reveal that the genetic algorithm is able to exploit the flexibility provided by a novel chromosome architecture, and utilise a combination of primitive gates and macro components from a component library, in order to produce circuits which operate well within timing restrictions. Comparisons are then made between circuits generated using our genetic algorithm, and circuits generated using standard CAD based design methodologies.

2 Describing the Genetic Algorithm

2.1 Encoding a circuit within a chromosome

Genetic algorithms for evolvable hardware are used to develop chromosomes which then encode the functional de-

scription of a given circuit. As with many applications which utilise genetic algorithms, the resulting circuit is termed a *phenotype* as it comprises numerous smaller logic cells or *genotypes*. The terminologies used are designed to reflect the conceptual similarity between genetic algorithms, natural evolution, and genetics.

The genetic algorithm presented here uses a permutation-based encoding of fixed-length. As such only a specified number of logic elements are presented to the framework. From this, the desired circuit functionality must be generated. Using a fixed-length encoding is standard practice and is one of the main restrictions within which a genetic algorithm operates [1].

Specific sections of each chromosome are reserved for describing the inputs and outputs required for the desired circuit. Logic elements are referenced by position within the chromosome. Figure 1 displays the relative location of each encoded section. Circuit inputs are encoded in the first sec-

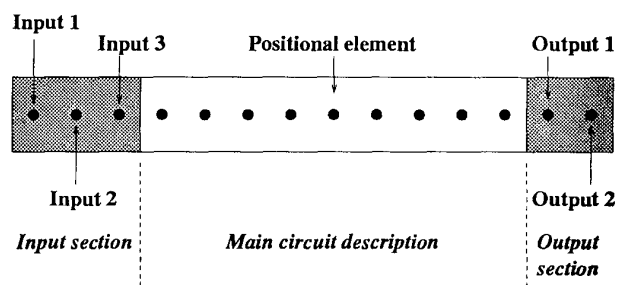


Figure 1: Chromosome structure defining sections for specific circuit description

tion of chromosome. If a circuit has I inputs, then the first I elements in the chromosome will describe these inputs. This description includes the input pin number in addition to which logic element the input pin is connected. Outputs are similarly defined at the end of the chromosome, where position relates to the identification of an output pin connected to a logic element. Total chromosome length, N , is then defined as the number of logic elements summed with the number of circuit inputs. Therefore, if a circuit has two outputs, whatever logic elements are at N and $N-1$ are connected to output pin one and output pin two respectively.

The encoding ensures that the number of inputs and outputs described by a chromosome remains consistent after operations such as crossover.

The genetic algorithm presented in this paper utilises a range of functional elements or *macro blocks*, along with simple gate primitives with which to generate various circuit structures. Macro blocks particularly suited to more complex arithmetic circuits were chosen such as a halfadder and fulladder. Other macro cells included small combinational logic blocks, in addition to simple through-connects. Figure 2 displays samples of the types of additional cells present in the component library.

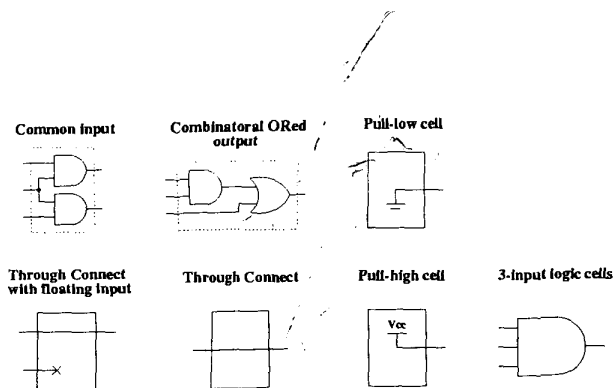


Figure 2: Generic style of macro and other logic elements provided to component library for the evolution of arithmetic circuits.

Each cell is connected within a flexible chromosome encoding which allows placement of any cell (macro or primitive) into any position within the string. This provides the EHW framework both the flexibility of standard gate level encodings, and the potential of building more complex systems afforded by less flexible functional architectures. Both circuits presented in Figure 3 are functionally

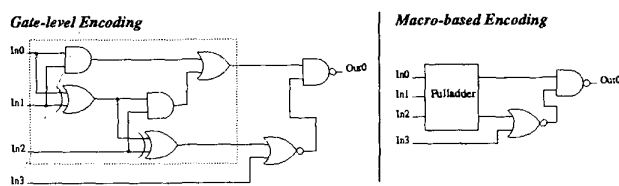


Figure 3: Comparison of a standard gate-level encoding with the novel macro-based encoding to describe a Fulladder with additional logic.

identical. However, the gate-level encoding would require a seven cell description to represent the circuit, while the macro-based encoding would require only three. Although more cell connectivity information is required to encode the fulladder cell described using the macro approach, the overall reduction in chromosome length justifies this.

2.2 Connecting Cells Within the Chromosome

Each genotype (logic element) in a circuit is allocated a specific position within the corresponding chromosome. The type of logic cell at any given position is initially determined randomly, however cells can be allocated different positions after initialisation through manipulation by *genetic operators* (see section 2.3). Figure 4 demonstrates the technique used to encode the connectivity of the fulladder cell depicted in Figure 3. It is important to note that a cells connectivity is not restricted to its nearest positional neighbour. Rather, cells are free to connect to any cell of higher position within the chromosome. This form of 'over-the-cell' connectivity provides a much wider range of possible circuit configurations. Feedback connections, however, are not permitted as their effects are not desirable for most DSP applications.

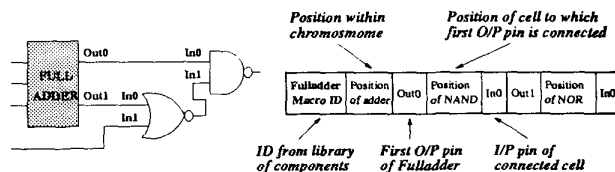


Figure 4: Example of macro-based encoding describing a macro element (fulladder) and its connectivity.

2.3 Parameters and Constraints

Several constraints are imposed during initialisation of the genetic algorithm. Some are designed to eliminate contentious circuit configurations, while others are a result of the evolutionary algorithm employed. Initial global parameters are entered by the designer and are as follows:

- Number of inputs and outputs required for the desired circuit;
- Definition of input and output vectors upon which evaluation takes place, and which describe circuit functionality;
- Number of logic elements within a chromosome used to create the circuit;
- The maximum number of possible fan-outs per cell output;
- Definition of global clock speed for timing constraints;
- Population size, defining the number of circuit solutions concurrently evolving within the search space.

So as to optimise cell connectivity within a fixed-length circuit encoding, each output pin on a logic element is randomly allocated a fan-out ranging between one, and a user defined maximum. Fan-out describes the number of logic elements that an individual element may connect with. The connectivity of any specific logic element is not restricted to its nearest positional neighbour.

2.4 The Genetic Operators

Crossover and mutation are the operators through which new circuit solutions are generated. Crossover swaps material between two individuals at randomly chosen points along the chromosome, producing *offspring*. The greater the number of crossover points, the higher the degree of intermixing between the parents. The resulting offspring encode two new circuit solutions, potentially better than the two parent solutions which generated them. The algorithms presented in this paper use single-point crossover. Due to the complex interaction of logic elements which comprise a circuit, the effects of both crossover and mutation can be highly disruptive to the search. This is a result of broken connections between logic

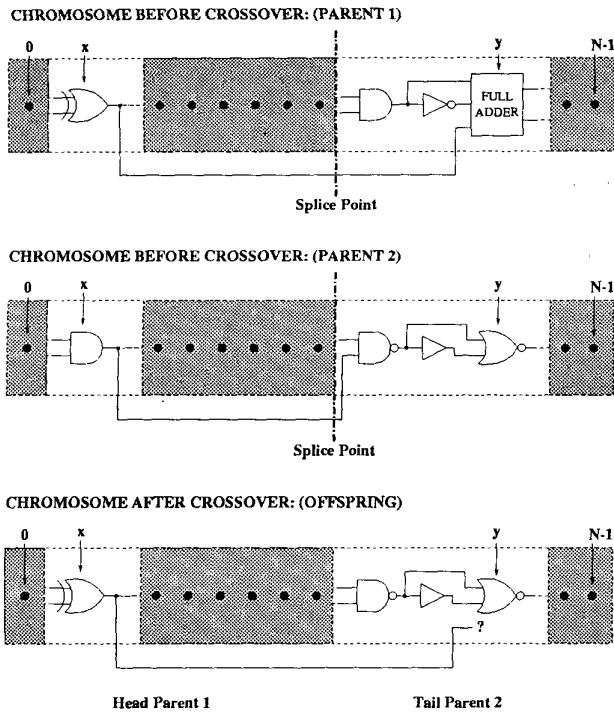


Figure 5: Example of broken element connectivity resulting from crossover.

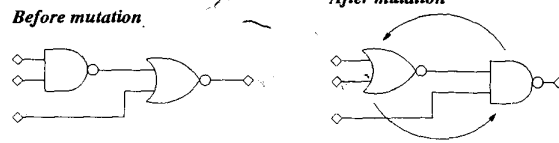
elements caused by recombination. Figure 5 illustrates this effect.

So as to minimise the negative effects of crossover, chromosome 'repair' is used to reconnect any element connections broken during the operation. This is achieved using a nearest neighbour connection rule. In the example offspring chromosome depicted in Figure 5, the logic element at position x is no longer able to connect to the new logic element now at position y . The repair algorithm instead attempts to connect element x to its nearest neighbour at position $x+1$. If this is unsuccessful then subsequent reconnection attempts are made from $x+2$ to $N-1$, where N denotes the total number of logic elements present in the chromosome. In the event that no logic elements are available for connection, the current element is assigned as floating and further attempts at reconnection made during later crossover operations. Each logic element in the resulting offspring chromosome is examined for broken connections after every crossover event.

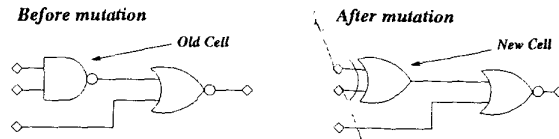
Mutation is used to maintain diversity within the population. It operates directly after crossover and is analogous to a copying infidelity as material is transferred from parent to offspring. Mutation is invoked with relatively low probability so as not to prove deleterious to the algorithm search. There are four circuit-specific mutation operators used within the genetic algorithm presented. Each is depicted in Figure 6.

Each operator was specifically designed to enhance the genetic algorithm by providing it with ability to introduce

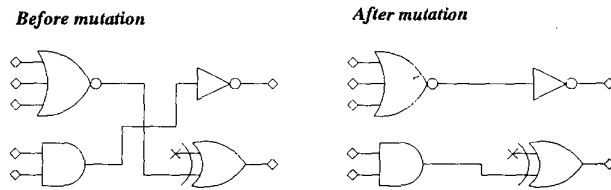
Inter-chromosome cell swapping:



Cell replacement:



Connection interchange:



Cell specific pin interchange

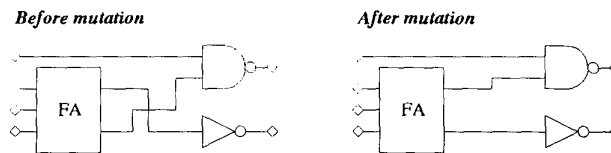


Figure 6: Four mutation operators used by the genetic algorithm.

both new logic elements and connections not obtainable using crossover. Of these operators only *cell replacement* needs explanation. The result of this mutation is to replace an existing logic element in the chromosome with one randomly selected from the component library. This ensures that new solutions can be obtained through diversification many generations after initialisation.

Our mutation rate is a derivation of the chromosome bit-length relationship originally proposed by Mühlenbein [2], and defined as:

$$P(m) = (1 / l)$$

Where the number of bits used to encode the chromosome, l , governs the mutation rate. Because the genetic algorithm presented here uses a permutation-based integer encoding, a direct translation of Mühlenbein's relationship is not possible. Instead the total number of logic elements encoded in the chromosome (N) is used to represent l . Mutation therefore operates on each *logic element* with the same probability rate given above. If an element becomes subject to mutation, one of the four mutation operators highlighted

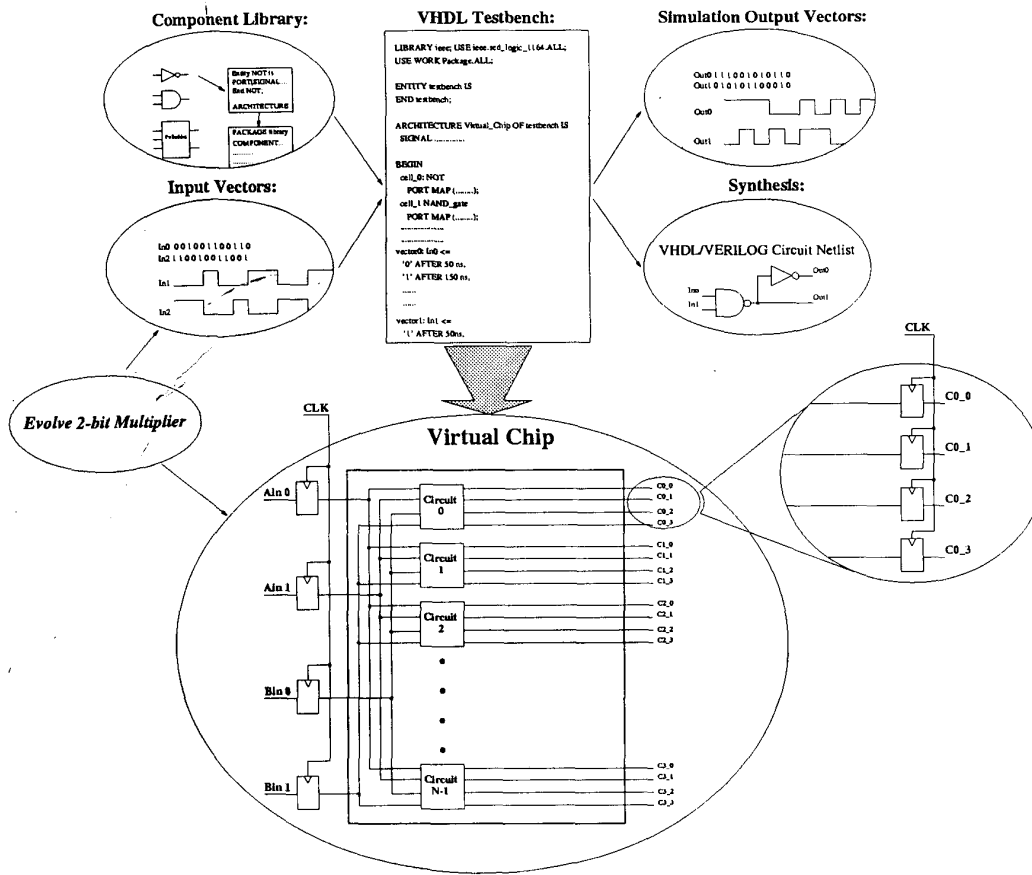


Figure 7: Graphical representation of the Virtual Chip environment, evolving a 2-bit multiplier within a population size of N .

in Figure 6 is applied, each with an equal probability of selection (0.25).

A summary of the parameters applied to our genetic algorithm are as follows:

- Generational genetic algorithm
- Two-way tournament selection implemented (Two chromosomes are selected randomly and the fittest becomes a member of the next generation)
- One-point crossover at 0.7; chromosome repair applied;
- Mutation using Mühlenbein derivation $P(m) = 1 / I [2]$; with application specific operators;
- Population size fifty.

Fitness is represented as a percentage of circuit functionality. Correctness is calculated by summing the total number of correct bits produced by the circuit solution under evaluation and comparing this to the desired output response. Fitness is expressed mathematically as follows:

$$R_b = 2^I * O$$

$$\text{Fitness} = E_b / R_b$$

Where R_b is the total number of bits comprising the desired output vectors, E_b is the actual number of bits matched with the desired output vectors during evaluation, 'O' are the output pins and 'I' the number of input pins. Evaluation is achieved through interaction with a HDL (Hardware Description Language), described in the following section.

2.5 The Virtual Chip Environment

VHDL (*Very High Speed Integrated Hardware Description Language*) is one of two dominant languages for describing digital electronic systems [8]. It is a technology independent environment and describes the structure of a digital system by describing subsystems (logic elements) and how they are interconnected. In addition, circuit descriptions can then be accurately simulated without the need for hardware prototyping. After successful testing a circuit can then be synthesised to provide a technology specific net-list, ready for transfer onto silicon. Almost all technology vendors provide models for logic elements within component libraries.

The Virtual Chip has been designed to provide an automated digital design procedure. Within this framework a novel genetic algorithm is used to evolve digital circuits. Its simulated environment evolves the structure of a circuit directly within the VHDL language. This is performed within a specially designed testbench. It is this testbench which instantiates and interconnects all logic elements within each chromosome, used to describe a specific circuit solution. Evaluation is performed by instantiating and simulating all circuits described within a population of chromosomes, as if they were being implemented within a single reconfigurable chip. Figure 7 illustrates a 2-bit multiplier evolving within the Virtual Chip environment.

As can be seen in Figure 7, each 2-bit multiplier has 4 inputs and 4 outputs. All inputs and outputs are synchronised with flip-flops to account for propagation delays and ensure that all output signals have reached a steady state. It is these flip-flops which, governed by a global clock, set the timing constraints within which the evolving circuit must operate. *A circuit with incorrect timing will produce output signals offset with those desired and will therefore incur low fitness.*

Each 4-bit output grouping represents an individual circuit evolving within the virtual environment. Each grouping is tagged according to the circuit's ID within the evolving population. Every circuit solution is therefore represented as a technology independent VHDL netlist. Netlists are direct interpretations of the circuit chromosome and define a circuit in terms of its logic elements and inter-connectivity. Standard CAD tools can then be used to both optimise the circuit and translate the generic netlist into a technology specific netlist suitable for implementation in hardware.

Due to the *implicit* parallelisation of the Virtual Chip environment, the entire population is compiled, and simulated as one entity. This differs from most standard approaches which evaluate each individual solution sequentially. As a result, within the Virtual Chip environment, an entire population of fifty individuals, evolving fifty 2-bit multiplier circuits, can be simulated and evaluated in approximately five seconds. In contrast, if each circuit were evaluated as an *individual* entity, it would take approximately two minutes to evaluate the same population. These results were obtained on a standard Sparc Ultra 10 workstation with 640Mb of memory.

The Virtual chip is a fusion of C code and VHDL. The genetic algorithm itself is executed in C and generates the VHDL required to instantiate each chromosome encoded circuit. After a circuit has been successfully evolved it is then passed through a CAD tool for optimisation. Figure 8 displays the execution flow and coding format of the Virtual Chip EHW framework.

3 Implementation and Results

The following section highlights the results obtained when using our genetic algorithm to autonomously generate arith-

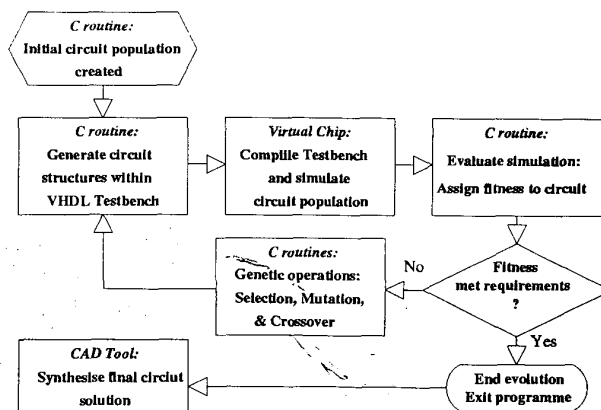


Figure 8: Execution flow and coding format of the genetic algorithm and Virtual Chip evaluation environment.

metic circuits evolved with performance constraints. Four circuit architectures are examined: 1-bit fulladder, 2-bit unsigned parallel multiplier, 7-bit pattern recogniser (one's voter) and 3-bit unsigned parallel multiplier. Each represents a benchmark circuit within the field of evolvable hardware research [6, 4, 9]. They also provide the foundation blocks for larger DSP applications. *Because evolvable hardware is still in its infancy, it should be noted that the circuits chosen represent some of the most complex arithmetic modules currently generated using an evolutionary algorithm.*

The performance of our genetic algorithm is compared with equivalent arithmetic circuits developed using standard HDL design methodologies. Using a standard HDL methodology, the four arithmetic circuits were developed in two stages. Each circuit was described at a behavioural level using a hardware description language. The description was then passed through an industrial synthesis tool in order to produce a technology specific netlist [10]. The same synthesis tool was used to optimise the technology independent netlists produced by our genetic algorithm.

For the purpose of future comparisons, the following terminologies will be used to describe the two different design approaches:

- *Genetic implementation*: a genetic algorithm for automated circuit design, presented in this paper.
- *Behavioural implementation*: conventional synthesis flow for digital circuit design from behavioural description written in HDL.

On average each circuit architecture was evolved ten times, and terminated after 10,000 generations if a fully correct solution (fitness of 1.0) had not been found. All four circuits were constrained by the genetic algorithm to operate no slower than 10 MHz. Chromosomes encoding fifteen logic elements (both gate primitives and macro blocks) were used to describe each of the four circuit architectures. The same

Method of Circuit Generation	Circuit Function	Circuit Area in equivalent NAND gates	Timing Slack at 10 MHz (ns)	Timing Slack at 100 MHz (ns)	Average Number of Generations
Genetic implementation	Fulladder	10.0	+INF	2.3617	674
Behavioural implementation	Fulladder	10.0	+INF	2.3617	NA
Genetic implementation	2bit multiplier	11.0	+INF	2.4907	3687
Behavioural implementation	2bit multiplier	14.0	+INF	2.0962	NA
Genetic implementation	3bit multiplier	60.0	+INF	1.8151	> 10,000
Behavioural implementation	3bit multiplier	59.0	+INF	1.7926	NA
Genetic implementation	7bit pattern recog	24.0	+INF	1.2697	2749
Behavioural implementation	7bit pattern recog	29.0	+INF	0.4949	NA

Table 1: Comparing different arithmetic circuit structures evolved using the genetic implementation, with that of functionally equivalent circuits generated using the behavioural implementation

number of logic elements were used to describe each architecture in order to demonstrate that combining both primitive and macro logic elements provides flexible, compact chromosome descriptions, with encoding lengths less influenced by circuit complexity.

Selected circuit netlists, generated using the genetic implementation, were optimised and re-synthesised for a specific silicon technology. The arithmetic circuits developed using the behavioural implementation were also synthesised using the same technology library. A technology library describes the physical characteristics (such as timing and area) of logic components associated with a specific fabrication process. In this experiment the *LSI Logic lca300K* technology library was used.

Using a technology library common to both genetic and behavioural implementations provides a common platform for comparison. As a result the performance of the arithmetic circuits generated using both implementations can then be effectively compared.

In addition to providing a technology specific circuit netlist, the synthesis procedure also provides circuit optimisation by removing redundant logic elements. This is particularly useful for circuits generated using the genetic implementation as many logic elements such as through connects (Figure 2) will also be removed. Table 1 displays both timing and area statistics for the four arithmetic circuits under investigation. Each circuit is identified as having been generated using either the genetic or behavioural implementation. Timing slack is defined to be the duration for which the slowest output of the circuit remained stable before the next data pulse arrives. It should be noted that *+INF* denotes that timing constraints are well within specified limits.

As detailed in section 2.2, our genetic implementation has been designed to incorporate performance constraints. Therefore, to further investigate the performance of the arithmetic circuits generated using the genetic implementation, selected netlists were re-optimised at an operational speed of 100 MHz. The selected netlists themselves were the same as those evolved to operate at 10 MHz found in Table 1, and were synthesised using the same technology library. Each circuit

was again compared with a functionally equivalent circuit designed using the behavioural implementation, and synthesised at 100 MHz. Both results are also displayed in Table 1.

Results show that circuits generated using the genetic implementation are of comparable or better performance than those produced using the behavioural implementation. Both fulladder circuits display identical timing and area characteristics. Although similar, the 'evolved' 2bit multiplier requires a smaller area than its equivalent, and also operates with less delay at 100 MHz. The 7bit pattern recogniser generated using the genetic implementation is noticeably more effective than its behavioural equivalent in both area and speed of operation.

Both 3bit multipliers are of comparable performance in terms of area and timing. However, the 3bit multiplier presented in this paper was evolved using a modular approach derived from our current algorithm. Modular approaches to circuit evolution have been investigated by a number of researchers, including our research group [11, 12]. Development of the modularised approach is on going, however, initial results for the evolution of the 3bit multiplier are shown here.

It is worth noting that although successfully optimised at 100 MHz, all circuits evolved by the genetic implementation were originally constrained to operate at only 10 MHz. The encouraging results therefore indicate *that robust timing performance can be achieved using performance related fitness criteria during evolution.*

Because our genetic algorithm directly manipulates a hardware description language, the *designer* requires little or no knowledge of digital design or hardware description languages. Only a boolean logic table, describing the circuits functionality, needs to be generated. Standard HDL tools do provide a means of synthesising circuits more efficiently than using behavioural-level circuit models. However, more detailed knowledge of circuit design and of *gate level* design methodologies is required. Such a low level design methodology also has the disadvantage of increasing the design time.

Analysis of the arithmetic circuits presented in this paper also provide insights into complexity issues associated with

automated circuit design. Table 1 details the average number of generations required by the genetic algorithm to produce a correct circuit solution. As would be expected, the number of generations taken to evolve a solution increases non-linearly with circuit complexity. Figure 9 displays the average performance of the genetic implementation when evolving each circuit architecture.

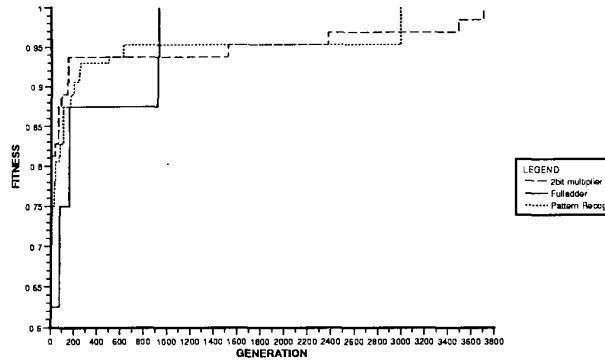


Figure 9: Typical number of generations taken to evolve a 1-bit fulladder, 2-bit multiplier and 7-bit pattern recogniser

The results presented in this section demonstrate that the flexible chromosome architecture detailed in this paper is able to provide circuits of varying complexity, whilst holding the chromosome length constant. The results indicate that by minimising the number of logic elements required to describe a chromosome, the complexity of the search space is also minimised. The trends shown in Figure 9 demonstrate the similarity in performance of evolving both the 2-bit multiplier and 7-bit pattern recogniser. Further research into the modularisation of circuit evolution hopes to produce larger circuits less influenced by the complexity of a chromosome encoding.

4 Conclusion

A novel genetic algorithm for the automated design of performance driven arithmetic circuits has been presented. The algorithm also describes a flexible chromosome encoding designed to fasciculate complex circuit structures with a minimal number of logic elements. The environment in which circuits are evaluated has also been presented. Four arithmetic architectures have been examined. Each architecture was autonomously generated using our genetic algorithm and compared with equivalent architectures developed using conventional high-level design methodologies. Results show that the arithmetic circuits generated using the genetic algorithm operate well within the timing restrictions imposed. For this reason the use of a genetic algorithm leads to a significant saving in design time whilst taking into consideration timing issues crucial to today's deep sub-micron technologies.

Bibliography

- [1] GOLDBERG, D. E., *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989
- [2] BÄCK, T., *Evolutionary Algorithms in theory and Practice. Evolutionary Strategies Evolutionary Programming Genetic Algorithms*. Oxford University Press, 1996
- [3] THOMPSON, A., LAYZELL, P., AND ZEBULUM, R. S., 'Explorations in design space: Unconventional electronics design through artificial evolution', *IEEE Transactions on Evolutionary Computation*, 3(3), 267–196, September 1999
- [4] ZEBULUM, R. S., PACHECO, M. A., AND VELLASCO, M., 'Evolvable systems in hardware design taxonomy, survey and applications', in *Evolvable Systems: From Biology to Hardware. (ICES 96)*, pp. 344–358, 1996
- [5] ARSLAN, T., HORROCKS, D. H., AND OZDEMIR, E., 'Structural cell-based vlsi circuit design using a genetic algorithm', in *IEEE International Symposium on Circuits and Systems*, pp. 308–311, Atlanta, USA, 1996
- [6] MILLER, J. F. AND THOMPSON, P., 'Aspects of digital evolution: Geometry and learning', in *Evolvable Systems: From Biology to Hardware. (ICES 98)*, pp. 25–35, 1998
- [7] PEDRAM, M., 'Power minimisation in ic design: Principles and applications', *ACM Transactions on Design Automation of Electronic Systems*, 1(1), 3–56, January 1996
- [8] ASHENDEN, P. J., *The Designer's Guide to VHDL*. Morgan Kaufmann Publishers, Inc, 1995
- [9] LEVI, D. AND GUCCIONE, S. A., 'Geneticfpga: Evolving stable circuits on mainstream fpgas', in A. Stoica, D. Keymeulen, and J. L. (Eds.), editors, *In Proceedings of the First NASA/DOD Workshop on Evolvable Hardware*, pp. 12–17. IEEE Computer Society Press, Los Alamitos, July 1999
- [10] Cadence Design Systems, Inc., *BuildGates User Guide*, release 2.3 edition, March 1999
- [11] TORRESEN, J., 'Increased complexity evolution applied to evolvable hardware', in B. Drs. Dagli, editor, *Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming and Complex Systems*, pp. X–Y, St. Louis, USA, November 1999. AN-NIE'99, ASME press
- [12] OZDEMIR, E., *Evolutionary methods for the design of digital circuits and systems*, Ph.D. thesis, The University of Wales, Cardiff, 1999