A Novel Evolvable Hardware Framework for the Evolution of High Performance Digital Circuits

Ben Hounsell
Dept. Electronics and Electrical Engineering
The University of Edinburgh
Edinburgh, Scotland EH9 3JL
Ben.Hounsell@ee.ed.ac.uk
(Int +44) 131 650 5665

Tughrul Arslan
Dept. Electronics and Electrical Engineering
The University of Edinburgh
Edinburgh, Scotland EH9 3JL
Tughrul.Arslan@ee.ed.ac.uk
(Int +44) 131 650 5592

Abstract

This paper presents a novel evolvable hardware framework for the automated design of digital circuits for high performance applications. The technique evolves circuits corresponding to each specific output, under restricted functionality and timing constraints. The individual circuits are then processed to generate the completed circuit. Applications have focused on the design of multi-output arithmetic circuits, such as a 3-bit multiplier. Circuit evolution is performed within a *Virtual Chip* environment, a fusion of C code, VHDL and CAD tool for synthesis. As a result of the tools used within the Virtual Chip, constraints such as timing are taken into account during evolution. Both primitive gates and functional macro blocks are available to the framework providing a flexible means for generating complex digital circuits. A 3-bit multiplier, evolved by our framework, is compared to an equivalent circuit generated through standard design techniques, and is found to be of comparable performance. An additional comparison is made with a multiplier evolved using a conventional evolvable hardware framework which does not use the phased approach to circuit evolution.

1 Introduction

Although evolvable hardware (EHW) has demonstrated the success of using evolutionary algorithms to generate digital circuits, it has also raised a number of questions as to how current EHW approaches to automated circuit design 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 [1]. 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 [1, 2], although representations which more closely model the characteristics of a particular hardware platform have also been presented [3].

Much attention in evolvable hardware 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 (DSM), 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 [4]. Inovative research into using EHW for DSP design has resulted in the development of arithmetic circuits from halfadder structures to more complex 2-bit parallel multiplier designs.

The automated design of digital 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 cells used to generate the circuit, and the application for which the circuit is being evolved. Evolutionary algorithms (EAs) 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 digital circuits where timing and area constraints are of great importance, and therefore *must* be accounted for by evolvable hardware applications.

The phased approach to EHW demonstrated in this paper, along with the use of both gate primitives and functional building blocks, implemented within the Virtual Chip environment, provide a means to both increasing the complexity of arithmetic circuits currently evolved, and the performance constraints under which these circuits must operate.

This paper therefore details the theory behind the phased approach to circuit evolution, and the Virtual Chip environment. The merit of the new phased technique is then demonstrated through comparison of an example 'evolved' circuit, with that of a functionally equivalent circuit developed through conventional design techniques. A final comparison is then made between the phased technique and our same algorithm but with phased evolution removed. This is to emphasise the difficulty of evolving complex arithmetic circuits using standard EHW frameworks.

The evolvable hardware framework presented in this paper therefore establishes a number of possible solutions to many of the problems associated with extrinsic circuit evaluation, and the generation of more complex circuits using EHW. A novel genetic algorithm is presented which approaches circuit design at an hierarchical level, using a flexible chromosome encoding.

2 Description of Evolvable Hardware Framework

The Virtual Chip has been designed to provide an automated digital design procedure which, after successful evolution of a digital system, is then able to synthesise a circuit solution to generate a technology specific net-list ready for transfer onto silicon. Within this framework a novel genetic algorithm is used to evolve non-sequential digital circuits.

2.1 Encoding a circuit within a chromosome

Evolvable hardware frameworks develop chromosomes which then encode the functional description of a given circuit. As with many applications which utilise GAs, 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 EHW, natural evolution, and genetics.

The evolvable hardware framework 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 [5].

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

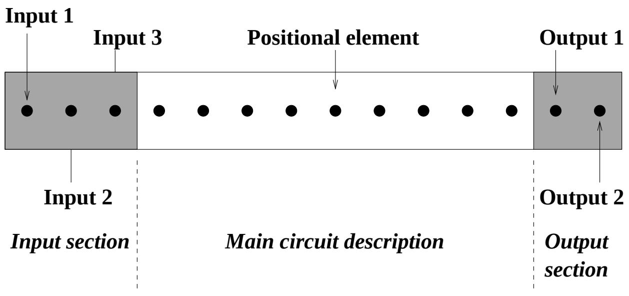


Figure 1: Chromosome Structure Defining Sections for Specific Circuit Description

first section 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, what ever 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 EHW framework 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 elements include small combinational logic blocks, in addition to simple through-connects. Figure 2 displays samples of the types of additional logic elements present in the component library. Each element is connected within a flexible chromosome

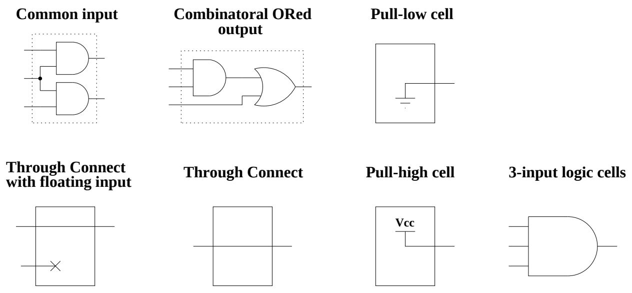


Figure 2: Generic Style of Macro and Other Logic Cells Provided to Library of Components For The Evolution of Complex Arithmetic Circuits.

encoding which allows placement of any logic element (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. For example, both circuits presented

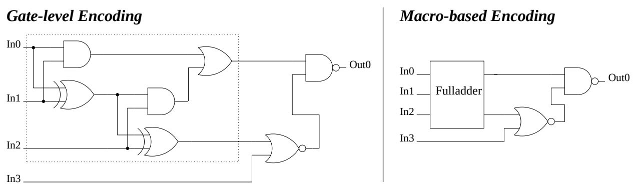


Figure 3: Comparison of a Standard Gate-level Encoding With The Novel Macro-based Encoding to Describe a Fulladder with Additional Logic.

in Figure 3 are functionally identical. However, the gate-level encoding would require a seven element 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.

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 element at any given position is initially determined randomly, however elements can be allocated different positions after initialisation through manipulation by *genetic operators*. Figure 4 demonstrates the technique used to encode the connectivity of the fulladder element depicted in Figure 3.

It is important to note that an elements connectivity is not restricted to its nearest positional neighbour. Rather, logic elements are free to connect to any element of higher position within the chromosome. This form of 'over-the-cell'

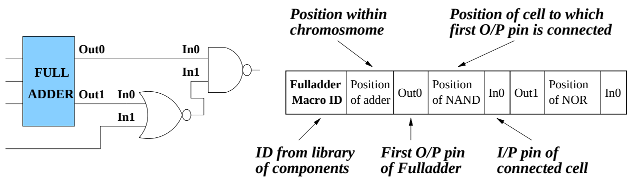

FULL ADDER
Out0
Out1
In0
In1
In0
In1
Position within chromosmome
Position of cell to which first O/P pin is connected
Fulladder Macro ID
Position of adder
Out0
Position of NAND
In0
Out1
Position of NOR
In0
ID from library of components
First O/P pin of Fulladder
I/P pin of connected cell

Figure 4: Example of Macro-based Encoding Used to Describe a Macro Cell (Fulladder) and its Connectivity.

connectivity provides a much wider range of possible circuit configurations. Feedback connections, however, are not permitted as there effects are not desirable for most DSP applications.

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

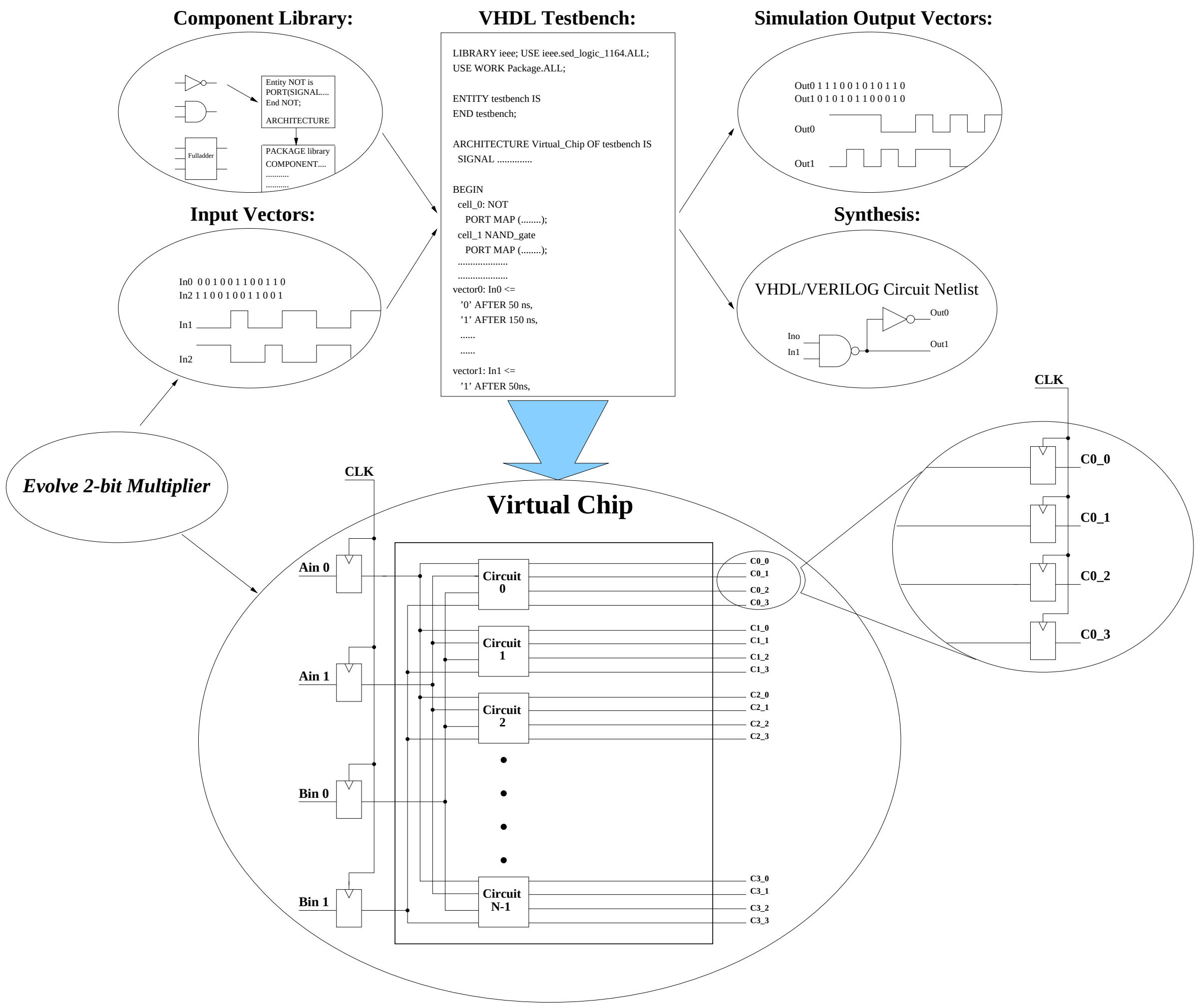
- Generational genetic algorithm
- Two-way tournament selection implemented (Two chromosomes are selected randomly and the fittest becomes a member of the next generation)
- One-point crossover at 0.7; chromosome repair applied;
- Mutation using Mulenbein derivation *P(m) = 1 / l* [6]; with application specific operators;
- Population size fifty.

A chromosomes repair algorithm is used to reconnect broken interconnects resulting from crossover and mutation. Once invoked, the repair algorithm attempts reconnection by focusing on the logic element whose output has been severed. The nearest positional neighbour to the severed logic element is then examined for connection. If no free connection is available, subsequent logic elements are examined, until the end of the chromosome is reached. In the event that no logic elements are available for connection, the current logic element is assigned as floating and further.

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. This is achieved through interaction with a HDL (Hardware Description Language), described in the following section.

2.2 The Virtual Chip Environment

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



```
LIBRARY ieee; USE ieee.sed_logic_1164.ALL;
USE WORK Package.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE Virtual_Chip OF testbench IS
  SIGNAL ..............

BEGIN
  cell_0: NOT
    PORT MAP (........);
  cell_1 NAND_gate
    PORT MAP (........);
  ....................
  ....................
vector0: In0 <=
   '0' AFTER 50 ns,
   '1' AFTER 150 ns,
   ......
   ......
vector1: In1 <=
   '1' AFTER 50ns,
```

Figure 5: Graphical Representation of The Virtual Chip Environment Within a VHDL Framework. Evolution of a 2-bit Multiplier With a Population Size of *N*.

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 5 illustrates a 2-bit multiplier evolving within the Virtual Chip environment.

As can be seen in Figure 5, 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 circuits ID within the evolving population.

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.

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 circuit encoded within a chromosome. After a circuit has been successfully evolved it is then passed through a CAD tool for synthesis. Figure 6 displays the execution flow and coding format of the

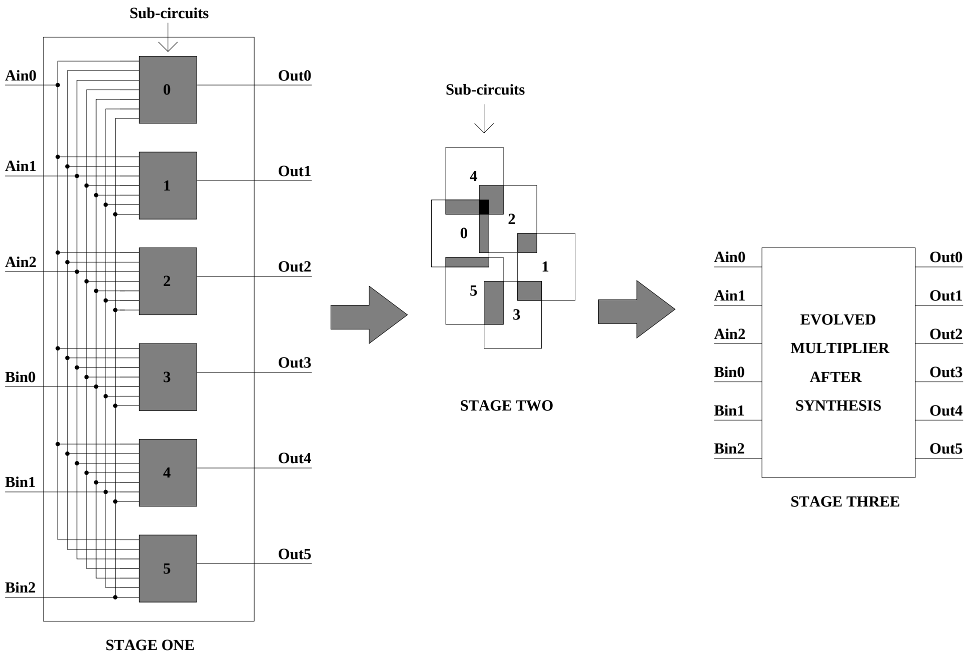


Figure 7: Example of Phased Evolution For The Automated Design of a 3-bit Multiplier.

Virtual Chip EHW framework.

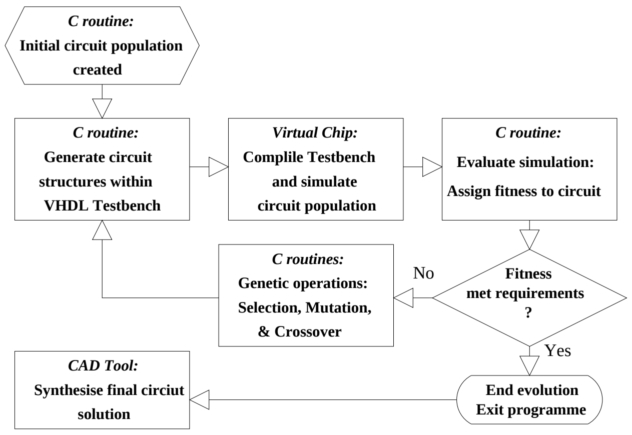

C routine:
Initial circuit population created
C routine:
Generate circuit structures within VHDL Testbench
Virtual Chip:
Complile Testbench and simulate circuit population
C routine:
Evaluate simulation:
Assign fitness to circuit
C routines:
Genetic operations:
Selection, Mutation, & Crossover
No
Fitness met requirements ?
Yes
CAD Tool:
Synthesise final circuit solution
End evolution
Exit programme

Figure 6: Execution Flow and Coding Format of Virtual Chip EHW Framework.

2.3 Phased Evolution

As highlighted in the introduction, the shear size of the search space involved in the automated design of digital circuits can often result in the failure of an evolvable hardware framework in finding a suitable solution. In addition to the number of logic elements required, and the desired circuit functionality, such complexity is well represented in the fitness evaluation of such circuits. In most cases evaluation consists of matching the output vectors of the circuit under analysis with actual output vectors, required by the desired functionality. By reducing the number of possible output vectors, and thus the required complexity of a circuit, it becomes possible to develop circuits of complexity that were previously difficult, or unattainable. Figure 7 visualises this approach for the example of a 3-bit parallel multiplier.

If the 3-bit multiplier were evolved as one unit the number of correct output bits required to correctly describe the entire circuit would be;

Output bits required = 2^I * O.

Where I is the number of inputs, and O the number of output bits required to encode the vector. However, stage one of the example circuit shown in Figure 7 demonstrates that, through phased evolution, the number of correct output bits required for each sub-circuit is;

Output bits required = 2^I

This represents a marked difference in circuit complexity.

Stage two in Figure 7 denotes the removal of redundant logic between the evolved sub-circuit structures as they are combined to generate the required circuit. A benefit of evolving partitioned circuits through phased evolution is to reduce the negative effects of the high degree of epistasis, inherent in design-based EHW applications. Epistasis describes the degree of inter dependency each element in the chromosome has on the other. It has been shown that a very high degree of epistasis, as can be found in high performance digital circuits, begins to favour random search over genetic algorithm techniques [8]. It might be assumed that simply combining each sub-circuits would result in an overall circuit much larger than that developed by either a design engineer, or an alternative EHW framework. Results in section 3 will show however that the high degree of common functionality between each of the sub-circuits generated, results in large amounts of cell reuse between circuits and thus impressive minimisation.

Stage three in Figure 7 represents circuit synthesis enabling the designer to investigate the evolved circuit for both different technologies, and confirm timing constraints are adhered to.

3 Implementation and Results

The following section details an example circuit evolved using our EHW framework and phased evolution. The example presented is that of an unsigned, 3-bit parallel multiplier. The multiplier is compared with a functionally equivalent design, generated using a standard behavioural level HDL-to-synthesis procedure.

So as to verify reproducibility, each of the phased outputs (six sub-circuits representing each of the six circuit outputs) were evolved ten times. After evolution, specific sub-circuit solutions were chosen at random, and combined to form the final multiplier. The circuit was constrained to run no slower than 10 MHz, and the area of each sub-circuit was restricted by a chromosome length of 15 logic elements. A total of 90 logic elements were therefore used to encode the multiplier. However, many of these elements will be simple through-connects and many will become redundant.

The completed circuit was then synthesised to remove redundancies and calculate cell area. Table 1 displays timing and area information about the 3bit multiplier evolved, along with the CAD-based equivalent. Timing slack is defined to be the duration for which the slowest output of the circuit remained stable before the next data pulse arrives.

To further test the performance of both multiplier circuits, each was synthesised to run at 100 MHz. The results are also displayed in Table 1. It should be noted that *+INF* denotes that timing constraints are well within specified limits. The results therefore indicate that, for a negligible increase in gate area, the evolved 3-bit multiplier operates equally as well at 100MHz as the hand designed, CAD based circuit. It should be noted that equivilant performance was obtained at this higher frequency, despite begin evolved to operate at only 10MHz.

The following compares the phased evolution technique with that of the same framework *without* phased evolution. Through this, the difficulty faced by single-step EHW techniques when evolving complex digital circuits becomes apparent. Figure 8 demonstrates the unsuccessful evolution of a 3-bit multiplier under the same constraints as previously detailed. In this case the total chromosome length was extended to 100 logic elements (both gate primitives and macro blocks), greater than the total number of elements used for the phased approach. Ten attempts were made to evolve a 3-bit multiplier in this way. In all cases the trend is typical of that shown in Figure 8, indicating that many more than 10,000 generations would be required to evolve a successful circuit. Although not substantiated, a figure of 30,000 to 40,000 generations is estimated at the current rate of progress observed.

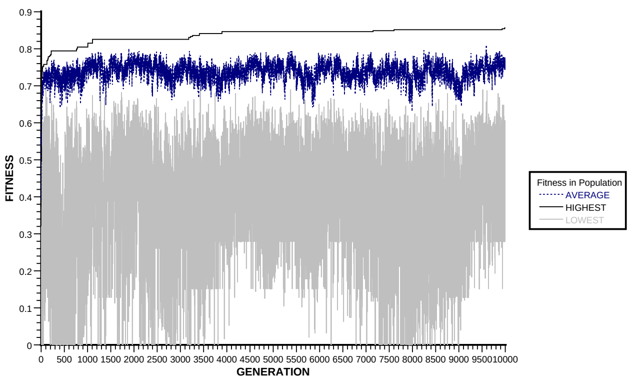


Figure 8: Example of Unsuccessful Evolution of 3-bit Multiplier Using Single-step EHW Techniques.

In stark contrast to the single-step approach, phased evolution provides the GA with numerous smaller complexity issues, and thus shorter evolution times. Table 2 displays the average number of generations taken to evolve each successful sub-circuit (a maximum of ten sub-circuits per output) for the 3-bit multiplier presented. It is clear that if each sub-circuit were evolved in serial approximately 20,000 generations would be required to generate the multiplier circuit. However, modern networking and efficient processors provide simple methods for executing the phased algorithm in parallel. In any case, by using phased evolution the number of generations required to evolve a 3-bit multiplier in either serial or parallel is considerably smaller than a standard one-step procedure.

Many of the sub-circuits evolved where compared to the average number of generations taken. Table 2 therefore reveals a good indicator of the circuit complexity required to produce the desired output. Figure 9 displays the synthesised 3-bit multiplier evolved through the phased technique,

Method of Circuit Generation	Circuit Area in NAND gates	Timing Slack at 10 MHz	Timing Slack at 100 MHz (ns)
Phased Evolution EHW framework	60.0	+INF	1.7266 - 1.8151
Standard CAD synthesis tool	59.0	+INF	1.7395 - 1.7926

Table 1: Comparing a 3bit Multiplier Evolved Using Our EHW Framework With That of a Functionally Equivalent Circuit Generated Using Standard CAD Techniques

Average Number of Generations to Evolve Sub-circuit					
Output0	Output1	Output2	Output3	Output4	Output5
3838	3930	7734	3508	827	55

Table 2: Average Number of Generations Taken by Phased Evolution to Evolve Sub-circuits For Each Output of 3-bit Multiplier

and analysed in this paper. Examination of the schematic demonstrates that the most complex logic path results in the output of pin two. Figure 10 presents the section of digital logic relating to the sub-circuit evolved for output two, after logic minimisation.

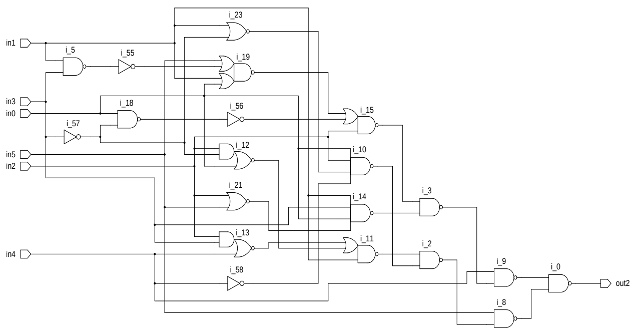


Figure 10: Schematic of Sub-circuit Relating to Functionality of Output 2 of 3-bit Multiplier.

Comparison with Figure 9 shows that sub-circuit 2 contains the most complex logic required to achieve correct functionality. This is confirmed by the number of generations taken on average to evolve the sub-circuit.

Figure 11 illustrates the simplification of the sub-circuit relating to output five. The resulting circuit demonstrates the

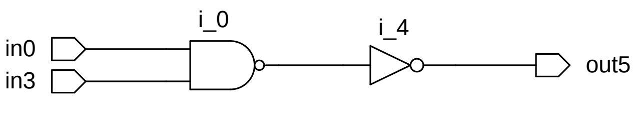


Figure 11: Schematic of Sub-circuit Relating to Functionality of Output 5 of 3-bit Multiplier.

effectiveness of both logic optimisation, and the availability of through-connect elements within the evolving component library (recall that each sub-circuit had a fixed length encoding of fifteen logic elements). Although not shown, the original sub-circuit representation of output five utilised a large number of through-connect and floating input elements (shown in Figure 2), before the removal of redundancies.

4 Conclusion

A phased approach to the evolution of high performance, multi-output, digital arithmetic circuits has been presented. The technique incorporates an EHW framework developed within a *Virtual Chip* environment for automated circuit design. The implementation of the Virtual chip environment is such that timing issues, critical to the development of high performance circuits, are taken into account during evolution. Analysis reveals that logic element reuse between sub-circuits is high, such that after the removal of redundant logic through synthesis, surface areas are comparable to functionally equivalent circuits generated through standard design techniques. This is attributed to a high degree of common functionality between each sub-circuit. Phased evolution partitions circuit complexity and in doing so reduces the associated degree of epistasis, making it possible to evolve complex circuits more effectively than standard EHW techniques. Results demonstrate the non-uniformity of the circuit complexity required by individual output paths. A 3-bit multiplier has been presented to exemplify the issues discussed. The evolved circuit was constrained to run at a minimum of 10 MHz, but also synthesised successfully at 100 MHz. In both cases, the circuit performed equally as well as a functionally equivalent multiplier generated through standard design techniques. Research is continuing using both phased evolution and the Virtual Chip, for the automated design of more complex DSP circuits evolved with inherent performance constraints.

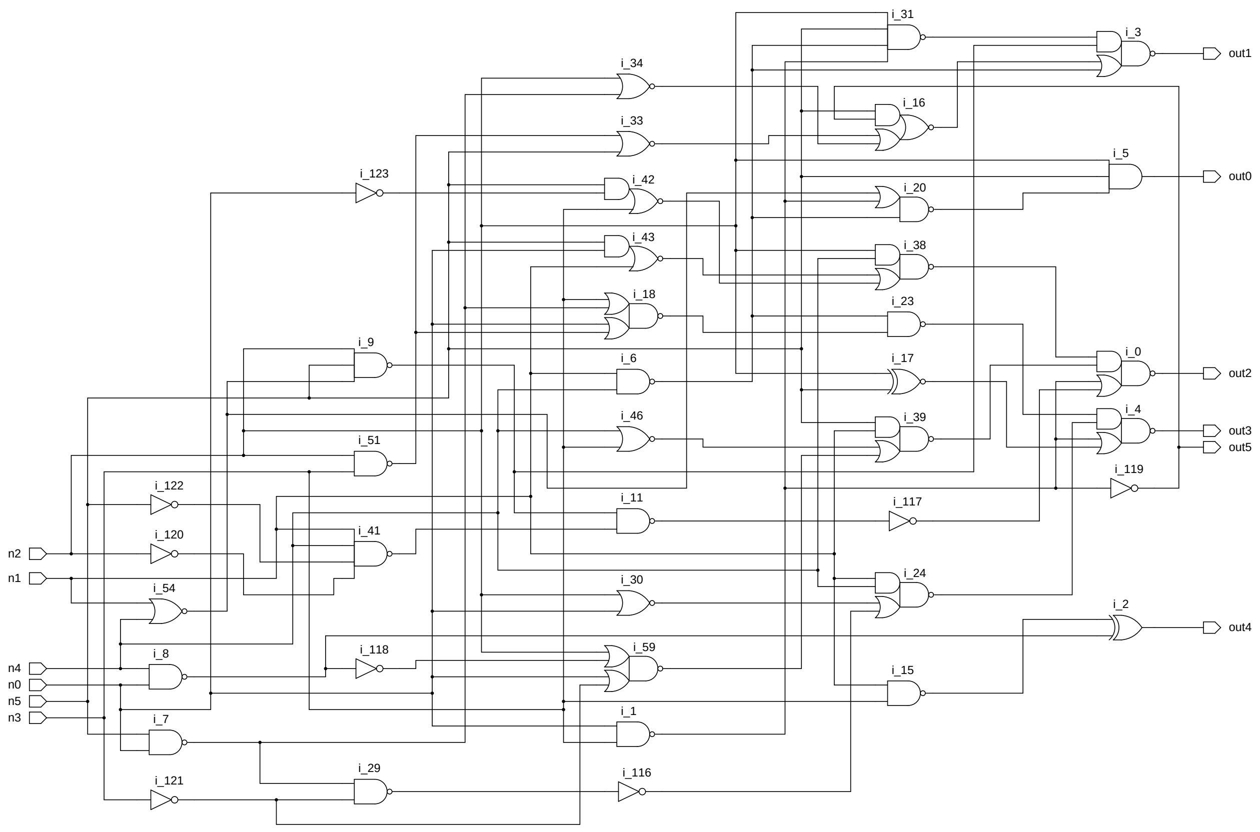


Figure 9: Example of Synthesised 3-bit Multiplier Generated Using Phased Evolution Technique Within The Virtual Chip Environment.

References

[1] 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

[2] 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

[3] 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

[4] PEDRAM, M., 'Power minimisation in ic design: Principles and applications', *ACM Transations on Design Automation of Electronic Systems*, **1(1)**, 3–56, January 1996

[5] GOLDBERG, D. E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, 1989

[6] BÄCK, T., Evolutionary Algorithms in theory and Practice. Evolutionary Strategies Evolutionary Programming Genetic Algorithms. Oxford University Press, 1996

[7] ASHENDEN, P. J., The Designer's Guide to VHDL. Morgan Kaufmann Publishers, Inc, 1995

[8] DAVIDOR, Y., 'Epistasis variance - suitability of a representation to genetic algorithms', Technical report, The Weizmann Institute of Science, Dept of Applied Mathematics and Computer Science, December 1989