

Structural Cell-based VLSI Circuit Design using a Genetic Algorithm

T. Arslan, D. H. Horrocks and E. Ozdemir

Cardiff School of Engineering,
University of Wales Cardiff, Cardiff CF2 1XH, UK.
arslan@uk.ac.cardiff,

ABSTRACT

A technique for the structural synthesis of VLSI circuits is presented. The technique uses a Genetic Algorithm which utilises a library of devices for the synthesis procedure which proved successful in searching a highly complex space of structures by producing designs which satisfy a multiple output circuit criteria which, in addition to satisfying the functionality constraint for each of the outputs, also satisfy hardware-specific criteria such as area and delay.

1. INTRODUCTION

Genetic Algorithms [1] (GAs) have been applied to various aspects of the digital VLSI design area. Examples include cell placement [2], channel routing [3], test pattern generation [4] and design for test [5], and VLSI-based signal processing [6]. However, the use of genetic algorithms for higher level *structural* VLSI design synthesis has been for severely restricted forms of structure [7]. The need for this restriction arises mainly from the complexity of the solution space which grows exponentially with the number of parameters to be optimised. Such a solution space is exemplified in the case of a GA which manipulates typical VLSI cell libraries which incorporate hundreds of cells with different attributes. The increase in the complexity of current VLSI circuits has led to a subsequent increase in the design effort required by the VLSI system designer who has to consider aspects at various levels throughout the design hierarchy. For this reason, it is particularly desirable if attributes at different levels in the design hierarchy are considered by the GA. Examples are considerations of attributes such as individual gate delays, their constituent transistor parameters, and the length of the interconnections in the circuit and their fan-in/fan-out.

This paper reports on a GA which overcomes the restrictions of the work carried out to date in the field of structural design synthesis (for example, [7]) and that carried out at a much higher level [8]. The developed GA can synthesise an unrestricted range of circuits using a multiple-constrained fitness function, which considers aspects at different levels over the design hierarchy, and can deal with multi-input/multi-output circuits.

2. CELL LIBRARIES

The design libraries used include primitive cells such as AND, OR, NAND, INV, XOR, XNOR. A specific library may include some of these primitives and other more advanced ones. A library, which could be viewed as lookup-table, includes information such as the following, about each cell:

1. Code: A number used in the cell-type parameter of a gene (see section 3).
2. Function : Logical function of the cell.
3. Delay: This includes delay parameters such as typical and maximum input-output delay in nano-seconds.
4. Area: The physical area of the cell.

In our implementation a typical library contains about one hundred cells which might include devices with duplicate functionality but different delay and area parameters. For the purpose of illustrating our technique and demonstrating our results, cell-libraries used for this work contain two-input/one-output primitives. However, the technique can easily be adapted to deal with cells with more i/o's.

3. THE GENETIC ALGORITHM

The GA is implemented to suit the structural design synthesis problem. For example, the chromosome representation in figure 1, is designed to incorporate individual library cells and their attributes (e.g., inputs and outputs) in a manner which can incorporate circuits of any size. The representation used is compatible with the *extracted* circuit representation used by the majority of CAD tools (such as Cadence®, Mentor®, and ES2®) which provides the added flexibility of integration with such tools and hence being a valuable tool to the VLSI circuit designer who is facing increasingly challenging tasks such as designing for low-power, area, speed, etc. In addition, the genetic operators are designed to take the circuit aspects of the structural design into consideration. The GA utilises a given design cell library in the structural design of a multi-input/multi-output logic function. This is done by allowing unrestricted choice within libraries and using a multi-constrained fitness function which allows the optimisation of hardware aspects such as

speed, area in addition to that of achieving the correct logic function(s). The GA continuously references the design cell library throughout its genetic evolution as the chromosomes are processed for fitness calculation. The use of the design library within a multi-constrained fitness framework is illustrated in figure 1.

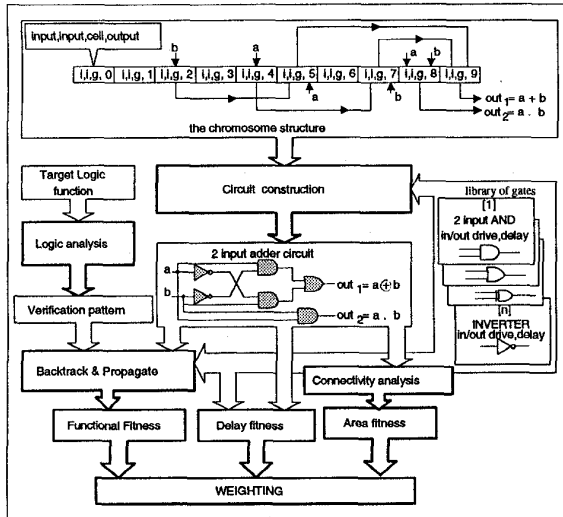


figure 1: Multi- Objective fitness evaluation

The genetic procedure commences by the random initialisation of each individual in a population of approximately fifty chromosomes. The roulette wheel method [1] is used in selecting pairs of individuals for crossover, see later, which will result in two new child individuals contributing to the next population. Next, two types of mutation are employed in order to add diversity into the population. The first is performed at the global chromosome level whereas the second is gene-specific. Both crossover and mutation will be discussed in greater detail later in this section.

Fitness is evaluated, see later, for each chromosome in the new population. If any of the individuals in this population satisfies the specifications for logic function and hardware-specific criteria then the genetic evolution terminates otherwise it is continued.

Representation

Each chromosome represents a possible design for the target circuit. A chromosome is a sequence of genes, each of which holds information about an individual cell in the design. These are as follows:

1. input 1: A number indicating the identity of a unique net (an internal connection or a primary one) which is input to the particular cell. This is

- the first input of the cell.
- input 2: The second input of cell .
- cell-type: Identifies the cell-type selected from the library.
- output: A number indicating the identify of a unique net which is output from the cell.

Crossover

The crossover function is defined to deal with the multi-output nature of the circuits synthesised. This is illustrated in figure 2, where chromosome1 and chromosome2 are the parents selected using the Roulette Wheel. The output fitnesses f_1 and f_2 , for outputs out_1 and out_2 respectively in chromosome1, are compared and the weakest is selected for a swap operation. This operation replaces the set of cells and interconnections contributing to the weak output, the output corresponding to the smallest of f_1 and f_2 , by the corresponding set in chromosome2.

Following a crossover operation two repair/reorganisation functions might be used. The first ensures that the circuitry transferred to chromosome1 are adequately accommodated. Although most of the transferred circuitry will occupy positions in chromosome1 which were originally held by the circuitry contributing to the replaced output, this function is required because some extra positions may be necessary for cells which fanout to more than one cell, contributing to more than one output, in chromosome2. The crossover operation duplicates such cells in chromosome2 and then transfers them into chromosome1. The additional positions are usually found by searching the chromosome for redundant circuitry, that is, those not contributing to any output functionality.

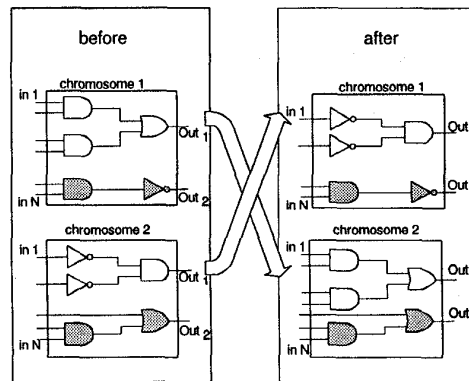


figure 2: Crossover operation

The wiring information in a gene consists of numeric codes which could refer to different wires in chromosome1 and chromosome2. The second repair operation ensures that codes associated with wires to be placed in chromosome1 are not duplicated in the

genes of this chromosome hence incorrectly altering the functionality of the circuit. If such duplication exists, then this repair operation associates new codes with the genes of the transferred circuitry. Crossover is performed at 50% rate.

Mutation

Two mutation operations are employed. These are as follows:

Global Mutation: This is performed at the global chromosome level, where a gene, at random, is selected and an arbitrary number of its parameters are selected, a single parameter or all parameters, and altered to a random value according to the position of each parameter in the gene. For example, if cell-type and *in1* are selected, then a random cell in the library is chosen for cell-type and a random net is selected for *in1*. This mutation is performed at a rate of 1%.

Circuit Mutation: This is a more *circuit-specific* operation in which sections of the chromosome are selected based on the fitnesses of each output in the circuit, see later. Circuit mutation targets cells and interconnections contributing to a weak output, as mentioned earlier in this section, and tries to alter the cell-types and/or the interconnections in the aim of improving the output. Circuit mutation also monitors delay and area parameters in the circuit and alters cells, based on their delay and area fitnesses, by replacing them with other cells in the library. This mutation is performed at a rate of 1-2%.

The fitness function

The multi-objective fitness function is a crucial part of the genetic system. The *target logic functions*, which are to be structurally synthesised, are processed by this part in order to obtain an optimum set of input/output patterns which can test the functionality of each prospective circuit structure, defined in a chromosome. A special circuit construction function is used to encode the chromosome and put it in a form where it could be analysed for aspects such as connectivity and redundancy. The patterns, above, are then used to evaluate the functionality of the constructed circuit through a sequence of operations which involve propagate logic values through the paths in the circuit and backtracking when necessary. This is especially true for the cases of multi-output circuits or where the circuit includes some unused redundancy.

Fitness is calculated as the weighted sum of three individual sub-fitness for each of the functionality, area, and delay parameters. The overall fitness is

defined as follows:

$$\text{Overall-fitness} = \alpha \cdot \text{fitness-output} + \beta \cdot \text{fitness-area} + \chi \cdot \text{fitness-delay},$$

where α , β , χ are the respective weights.

The functionality fitness, output-fitness, is the *normalised* fitness for all outputs in the circuit and is obtained as follows: The fitness of each output is calculated by the propagation of input vectors, from the target function truth-table, throughout the circuit and evaluating the output. The number of correct matches with the truth table is used as a measure of fitness for the particular output considered. This is repeated for each output. Hence, *output fitness* for an n output circuit is calculated as follows:

$$\text{output fitness} = \frac{\sum_{i=1}^n f_{oi}}{n},$$

where f_{oi} is the number of matches for the i the output.

The delay fitness

This is calculated as the sum of the delays in the critical path, the longest path in the circuit, of the circuit.

Area fitness:

This is calculated as the sum of the areas of the active cells in a chromosome. Active cells are those which contribute to the functionality of the circuit output since occasionally the GA produces some redundant circuitry.

$$\text{area fitness} = \sum_{i=1}^n (\text{cell area})_i,$$

where n = number of active cells.

Both Delay and Area fitnesses are calculated with continuous reference to the design library.

The above fitness frame-work can accommodate additional sub-fitness functions for considering other aspects in a given circuit.

4. RESULTS

In our investigations the values of the weight parameters α , β and χ were chosen to be 10, 1 and 1 respectively. The performance of the technique is illustrated using the n -input adder and parity checker example problems [7]. The performance of the GA, with a number of such circuits, is shown in

figure 3. In each case the GA performance demonstrated is the average of 6 runs. The results show a steady improvement in circuit design solutions evolved by the GA until a generation is reached in which at least one member satisfies all delay, area, and functionality specifications. The number of generations required is typically 60-65 which is relatively small considering the size of the solution space. The GA used manipulates a single cell library containing more than 100 cells each of which are different in terms of delay and area although, some of these cells may perform the same functionality.

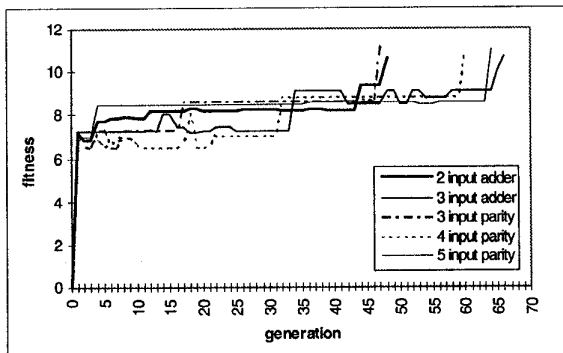


figure 3: GA Performance with different circuits

Our result indicate some discontinuity, (sharp rise and fall) in the evolved fitnesses. This is due to the nature of the structural synthesis problem in which a mere single incorrect interconnection or wrong type of cell may produce a completely incorrect logic function. This is also effected by the functionality fitness being the most heavily weighted in comparison to the others. Figure 4 illustrates the profile of the delay, outputs, area and, overall fitnesses during genetic evolution for a three-input two-output adder/subtractor circuit.

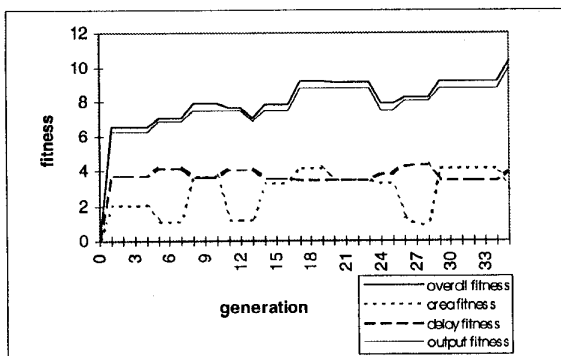


figure 4: Fitness evaluation for adder/subtractor circuit

5. CONCLUSIONS

The results obtained using the genetic algorithm indicate a great degree of flexibility in coping with special purpose arithmetic circuits, such as parity

check circuits and adders, in addition to those general purpose purely combinational circuits, such as decoders and multiplexers. In all of the cases the evolved populations of solutions provide a group of differing circuit designs, all of which satisfy the specified logic functions. This is an important feature since the GA offers the designer with alternative designs.

References

- [1] Goldberg D.E., "Genetic algorithms in search, optimisation and machine learning," Addison Wesley, Reading, 1989.
- [2] Hegde U. and Ashmore B., "A feasibility study of genetic placement", Texas Instrum. Technol. J., vol. 9, pp. 72-82, 1992.
- [3] Buttitta B., Orlando P., Sorbello F., and Vassallo G., "Monreale: A new genetic algorithm for the solution of the channel routing problem", Proc. IEEE, vol. 31, pp. 462-466, 1991.
- [4] O'Dare M. J. and Arslan T., "Generating test patterns for VLSI circuits using a genetic algorithm", IEE Electronics Letters, vol 30, No. 10, pp. 778-779, May 1994.
- [5] O'Dare M. J. and Arslan T., "System Design for test using a Genetically based ATPG system", IEE Colloquium on System Design for Testability, London, April 1995, pp. 9/1-9/5.
- [6] Arslan T. and Horrocks D. H., "A Genetic Algorithm for The Design of Finite Word Length Arbitrary Response Cascaded IIR Digital Filters", First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, Sheffield, September 1995, pp.276-281.
- [7] Louis S. J. and Rawlins G. J., "Designer Genetic Algorithms: Genetic Algorithms in Structure Design", Proceedings of the fourth international conference on Genetic Algorithms, San Diego, USA, July 1991, pp. 53-60.
- [8] Martin R. S. and Knight J. P., "Genetic Algorithms for Optimisation of Integrated Circuits Synthesis", Proceedings of the fifth international conference on Genetic Algorithms, Urbana-Champaign, USA, July 1993, pp. 432-438.