LAUTECH JOURNAL OF ENGINEERING AND TECHNOLOGY 4(1) 2007: 16 – 20

GENETIC ALGORITHM APPROACH TO SYNTHESIS OF 2, 2 FULL ADDER LOGIC NETWORK.

I.A. Mowete[a], F. A. Semire[b,*] and A. A. Yusuff [b]

[a]Department of Electrical and Electronics Engineering University of Lagos, Akoka, Lagos State, Nigeria.
[b]Department of Electronics and Electrical Engineering, Ladoke Akintola University of Technology, Ogbomoso, Oyo State, Nigeria.

*** Corresponding Author**

Abstract

The problem of synthesizing a minimum cost logic circuit is formulated via a genetic algorithm (GA). The synthesis proposes a scheme that "evolves" the minimized logic solution of a defined input function. Method employed combines the available logic gates in several of ways until a minimized and best result is achieved. The result obtained showed a considerable improvement over the existing methods.

Key words: genetic algorithm, logic synthesis, full adder network and sum of products form.

Introduction

In recent times, research has demonstrated the efficacy of employing genetic algorithm in system design: tasks previously thought impossible are being achieved with a little research and innovation. The advent of these technologies (genetically modeled circuits) has developed the ways to approach the research and development of both software and hardware through logic minimization theories (Chattopadhyay et al., 1996; Quine, 1952; McCluskey, 1965). The first published work on the application of a genetic algorithm was on a playing-game technique modeled after natural selection (Bayley, 1967). These findings later propelled a wide area of research in different fields; one among such is Sanghamnitra where he applied a genetic-like algorithm to the design of a set of detections for a pattern recognition and classification machines (Sanghamnitra et al, 1998; Sanghamnitra and Sankar, 2005). The concept of using genetic algorithm to evolve digital circuit designs was discussed in (Higuchi et al., 1993). Efforts were broadly divided into functional - level evolution and gate-level evolution. Louis and Gregory (1991) made use of genetic algorithms on design structures attempting to solve the combinational circuit design problem. His use of masked crossover genetic operator proved to be very advantageous, but the GA itself became more of a powerful search technique. Alan et al. (1999) extended the research done by Louis, and used GA to automate the design of combinational logic circuits. They designed logic circuits with matrices, with each element representing a gate and the inputs from previous elements. The gate was selected from a list of five fundamental gates. AND, NOT, XOR, OR and WIRE where the last representing a physical wire (i.e. absence of a gate). Thus, logic minimization was solved by a constraint on the matrix elements: Maximizing the wire element and hence minimizing the total number of gates necessary for the implementation of the circuit. Koza (1992) used genetic programming in the design of combinational circuit, however, from a different perspective. His objective was set on the successful generation of the circuits and not their minimization. The design of a hardware-based cost function that would accelerate the GA by several thousand times was described by Shackleford et al. (1997). An odd parity circuit required 24 basic cells (BCs) versus 28 BCs and a magnitude comparator require 20BCs versus 21 BCs produced by the commercial system was formulated for a genetic algorithm.

This research describes an approach in solving the problem of minimal cost logic circuit synthesis using genetic algorithms. The approach proposes a scheme that involves the introduction of an innovative logic minimization that employs the use of natural selection to map a design to a logic system while minimizing the resources utilized. This scheme is referred to as Genetic algorithm Synthesis. GAs, as the name precludes, is a synthesis scheme base on a standard genetic algorithm.

Genetic Algorithm Overview

Genetic Algorithms (Holland, 1975) are "search procedure" based on the mechanics of natural selection and nature genetics. The algorithm operates on a set of offered solutions called "populations". Each solution called "individual" can be any solution in the solution space represented by a string called "chromosome". Different solutions will be coded into

different chromosomes values .the solution space is searched in order to find the optimum for a "target function". The resulting value is referred as the "fitness value" of the solution, the current population is evolved creating a new generation with higher fitness. The evaluation is done using three operators.

Reproduction: this operator selects one individual from the current generation to the next generation. The probability of an individual to be selected is proportional to its fitness value i.e. the survival of the fittest solutions.

Crossover: two individuals are mated in order to exchange genetic information. The chromosomes, which represent the two solutions, are broken at same random space and combined together creating two new individual. With a randomly chosen crossover position 2, the two strings 01101 and 11000 yield the offspring 01000 and 11101 as a result of crossover.

Mutation: mutation is a random change in the chromosome. One bit in the chromosome string is toggled. The original and mutate individual represent different solutions. Thus, a string 100101 may, as a consequence of random mutation gets changed to 110101. The mutations operator gives the GA an opportunity to search in new corners of the solution space.

Methodology

In this section we present our application of genetic algorithms to the design of logic circuit. We will show our results on a 2,2 full adder logic network design. Each individual in the population represents a candidate solution to the logic design problem. The procedures involved are sectioned in to different modules as shown in Fig. 1 and they are as follow:

Random Number Generator: This module provides the input instantiation, selection, and crossover and mutation modules with pseudo - random numbers.

Input Instantiation: This module provides the input of each truth table combination, stored in external memory and its output is fed into the fitness module for comparison to the experimental output.

Selection: This module receives member and a random number as inputs. It employs roulette wheel and tournament type of selection to decide whether the number can be selected for the next generation.

Crossover/Mutation: These modules accept members as input, and use the input pseudo-random numbers to decide whether crossover/mutation are to be performed upon the individual or not. Simple point crossover and single - bit mutation are typically used.

Fitness: The module is where fitness value of each individual is evaluated. It accepts the output from the input instantiation module compares it to the experimental output and calculates the fitness of that individual. It stores the new individuals in memory and passes them on to the selection module.

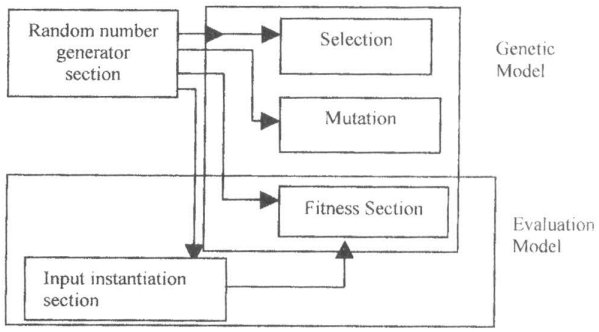

Figure 1: Overall block diagram

Experiments and Results

We implemented our methodology using the Jbuilder operating environment. Although it takes very much time to evaluate fitness values for each binary string, it was easy to implement such in JAVA

An experiment was performed on 2, 2 full adder logic function. The implementation technology was a library composed of OR, AND and XOR gates with the gate cost and size of the circuit measured in relation to number of logic gates and input pins.

For the logic function (2, 2, full adder) a genetically based algorithm was used to synthesis a 2, 2, full adder truth table shown in Table 1.

Table 1: Adder two - 2 bit truth table

Known Input				**Desired Output**			
a	b	C	D		x	y	z
0	0	0	0	0	0	0	0
0	0.	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	0	0	1
0	1	0	1	0	0	1	0
0	1	1	0	0	0	1	1
0	1	1	1	0	1	0	0
1	0	0	0	0	0	1	0
1	0	0	1	0	0	1	1
1	0	1	0	0	1	0	0
1	0	1	1	0	1	0	1
1	1	0	0	0	0	1	1
1	1	0	1	0	1	0	0
1	1	1	0	0	1	0	1
1	1	1	1	0	1	1	0

We ran our genetic algorithms code for 15 generations and we decided to move on the two best binary – strings from our evaluation to the next generation. Best fitness values from our generations are shown in Table 2.
The parameters for the genetic algorithms for our task are:
*Population size: 100
*Number of generation: 15
*Fitness evaluation: linear at 1.00
*Probability of crossover:0.6
*Probability of mutation: 0.001
*Type of mutation: single bit
*Type of crossover: single point
*Time of operation: 1.0 sec

Table 2: Table of result generated

bit 1 (z)		bit 2(y)	
logic gate combination	fitness value	logic gate combination	Fitness value
c	0.995	(((a^a)^(c&a))	0.9965
((d&b)^c)	0.997	b&d	0.997
((d&b)\b)	0.996	(a\c)	0.995
((b^((c\d)	0.994	b\(~a)	0.995
d^b	1.00	d	0.993
c/(c~)	0.995	a&b	0.994
b	0.995	(((a~))/c~a)	0.995
a~	0.994	((d&b)^b)	1.00
(a&a)/b	0.995	(b^a)	0.995
(a&a)7(a&a)/b	0.995	(c/(c~))	0.994

Bit 3 (x)	
((a^a)&((a^b)a	0.995
(a&((a^c)&(c/C)))	0.996
((d~)&a)	0.994
((d/(d~))	0.995
d	0.996
a	0.995
((b&c)a^d))c	1.00
(c&d)	0.995

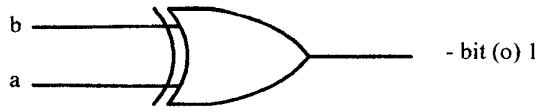

Fig. 2: Circuit for Output of bit 0

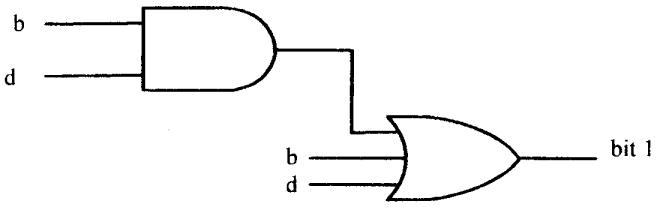

Fig..3: Circuit for Output of bit 1 of the Adder

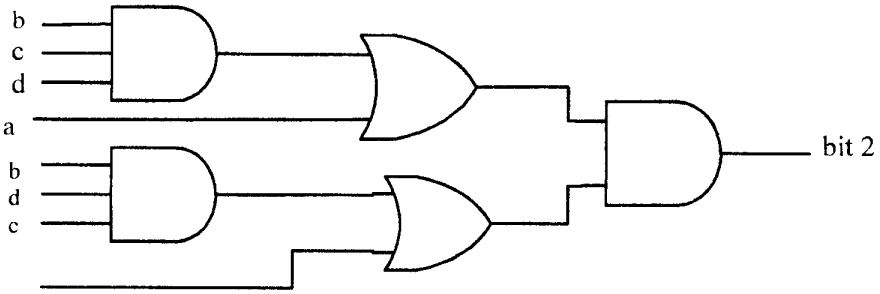

Fig 4: Circuit for output of bit 2 of the Adder

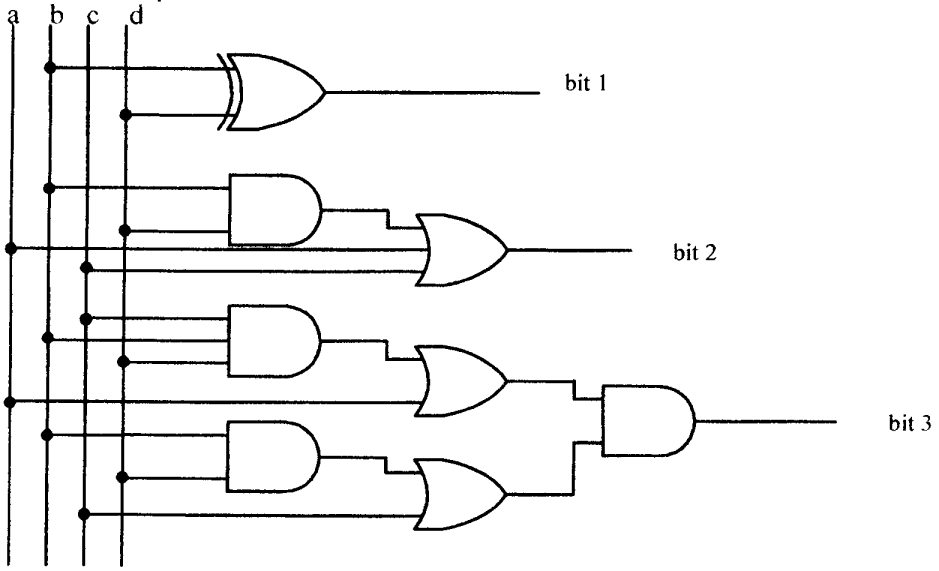

Fig 5: Synthesis Results for 2.2 Adder Circuit

Table 3: Experimental results

Circuit synthesis problem		Blocks	Input pins	Time (sec)	Generations
Full adder	Proposed method	6	13	1.0	15
	Existing method	7	18	2.0	20

As the results in Fig 1-4 show that a more compact circuits are realized by our GA approach. The proposed method took 1 second to scan through 15 generations as compared with existing method (Tatsuya et al., 1992) which took 2 seconds through 20 generations. The total input pins realized is 13 pins while that of existing method is 18 pins. The cost of circuit implementation, which is measured in terms of input pins, is reduced by a factor of 0.73 as depicted in Table 3.

Conclusion

The results presented in this paper indicate that genetic algorithms offer an attractive approach to logic circuit design. The performance is far better than the existing logic synthesis method in terms of cost compactability and speed. . Although this result is encouraging, much progress is needed for large-scale digital circuits.

References

Alan D., Christiansen Carlos A., Coello Coello and Arthro Hernandez Aquirre. (1994) "Towards automated evolutionary design of combinational circuits".

Bayley J.S (1967) "Genetic algorithm based on playing game technique modeled after natural selection," Bell System Technical Journal, .62 (3), 3220-3225.

Chattopadhyay, S., Roy, S. and Chaudhuri, P. P. (1996) "Synthesis of highly testable fixed - polarity AND – XOR Canenical network - A genetic algorithm - based approach," IEEE Transactions on computer. 45 (4), 487-490.

Higuchi T., Niwa T., Tanaka T., Iba H., De Garis H and Furuya T., (1993) "Evolving hardware with genetic learning: A first step towards building a Darwin Machine," Proceedings of the second international Conference on the simulation of Adaptive Behaviour (SAB 92), MIT press, 1993.

Holland J. H., (1975) "Adaptation in Natural and Artificial Systems" University of Michigan Press, 1975. (2nd edt: MIT press, 1992).

McCluskey, E. Jr., (1956) "Minimization of Boolean functions". Bell System Technical Journal, 35 (6), 1417 – 1444.

Quine, W.V. (1952). "A way to Simplify truth functions" Am. Math. Monthly, 59, 521-531.

Koza J. R., (1992) "Genetic programming on the programming of computers by means of Natural Selection", The MIT press, 1992.

Sanghamnitra B., Sankar K. P., and Murthy C.A, (1998) "Pattern classification using genetic algorithm", Pattern Recognition Letters, 19 (13), 1171-1181.

Sanghamnitra B and Sankar K. P., (2005),"Pattern recognition and machine Intelligence", First International conference , PREMI, Kolkata, India, December 20-22, 2005.

Shackleford, B., Okush, E., Yasuda, M. Koizumi, H., SCO, K. and Iwamoto T., (1997) "Hardware framework for accelerating the execution speed of a genetic algorithm", IEICE Transactions on Electronics, E80-C (7) ,962-969.

Sushil J. Louis and Gregory J.R., (1991) "Using genetic algorithms to design structures" Tech. Rep. 326, Computer Science Department, Indiana University, Bloomington, Indiana, USA, Feb. 1999.

Tatsuya M., Claud M.N.A and Roberto S., (1998) "Automatic circuit synthesis using Genetic Algorithm Based on the Coexistence of Heterogeneous Populations", Osaka institute of Technology