Genetic Algorithm based Logic Optimization for Multi-Output Majority Gate-Based Nano-electronic Circuits

Monireh Houshmand*, Saied Hosseini Khayat**, Razie Rezaei***
*Department of Electrical Engineering, Imamreza University of Mashhad, Iran
** Department of Electrical Engineering, Ferdowsi University of Mashhad, Iran
*** Department of Computer Engineering, Ferdowsi University of Mashhad, Iran

{monirehhoushmand, saied.hosseini, r.rezayee}@gmail.com

***Abstract-* The majority-gate and the inverter-gate together make a universal set of Boolean primitives in Quantum-dot Cellular Automata (QCA) circuits. An important step in designing QCA circuits is reducing the number of required primitives to implement a given Boolean function. This paper presents a method to reduce the number of primitive gates in a multi-output Boolean circuit. It extends the previous methodology based on genetic algorithm for converting sum of product expressions into a reduced number of QCA primitive gates in a single-output Boolean circuit. Simulation results show that the proposed method is able to reduce the number of primitive gates.**

Key words: Multi-output QCA circuits, Majority gate, Genetic algorithm, Hardware reduction.

I. INTRODUCTION

Quantum-dot Cellular Automata (QCA) circuits [1-6] are being investigated as a promising nanotechnology that attempts to create general computational functionality by controlling the position of single-electrons. The building blocks of the QCA circuit are the majority-gate and the inverter-gate.

Gate-level optimization is a necessary step in the design of logic circuits in general and QCA circuits in particular. Traditional logic reduction methods, such as Karnaugh maps (K-maps), always produce simplified expressions in the two standard forms: sum of products (SOP) or product of sums (POS). In the CMOS/silicon design, logic circuits are usually implemented using AND and OR gates based on SOP or POS formats. However, it is difficult to convert these two forms into majority expressions due to the complexity of multi-level majority gates. There has been some research in this area for QCA circuits [7-10].

In [10], an optimization method is proposed for single-output QCA circuits based on genetic algorithm (SO-GA). But, in this paper will be proved that for multi-output circuits (e.g., a full adder with sum and carry outputs) independently optimizing for each output does not produce an efficient solution. Consequently, in this paper, a genetic algorithm for multi-output circuits which is named MO-GA is presented. In this approach, first a SO-GA is run for one of the outputs and the desirable chromosomes are stored. Then other genetic algorithms are run for the other outputs, but not independent from the previous desirable chromosomes, in such a way that leads to a circuit having the maximum common gates with those chromosomes. In this way, this algorithm reduces the total number of gates in the circuit which implements all of the expected outputs. To demonstrate the effectiveness of the proposed algorithm, we choose some multi-output circuits and then in the first step, we optimize the circuits for each output independently using the SO-GA, and calculate the total number of required primitive gates. In the second step, we apply our proposed method (MO-GA), and calculate the total number of required gates again. Simulations show that our proposed method results in less or at most the same number of primitive gates in compare with the SO-GA method.

The remainder of this paper is organized as follows: in the next section some related background materials are presented. In section 3, an optimization for a single-output QCA circuit using the SO-GA is described. The proposed algorithm (MO-GA) is presented in Section 4. After that, simulation results are present and finally Section 6 concludes the paper.

II. BACKGROUND MATERIALS

A. QCA Basic

A QCA cell (Fig. 1) contains four quantum-dots positioned at the corners of a square and two electrons that can move to any quantum-dot within the cell through electron tunneling [1]. Due to Coulombic interactions, as shown in Fig. 2, only two stable configurations of an electron pair exist. These configurations are denoted as cell polarization P = +1 and P = -1, respectively. Using cell polarization P = +1 to represent logic '1'and P = -1 to represent logic'0', binary information can be encoded.

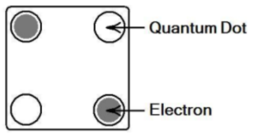

Figure1. A QCA cell

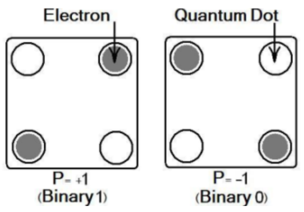

Figure 2. Two polarizations in a QCA cell

B. QCA Logic Devices

The fundamental QCA logic devices are the QCA wire, majority gate and inverter. In a QCA wire, a binary signal propagates from input to output because of the Coulombic interactions between cells.

The QCA majority gate is a device that implements a majority function. The reason why the device cell always assumes a majority polarization is that it is in this polarization state that the Coulombic repulsion between electrons in the input cells is minimized. Assuming A, B, and C as inputs, the logic function of the majority gate is:

$$M(A,B,C)=AB+AC+BC$$

By fixing the polarization of one input as logic '1' or '0' we can obtain an OR gate and an AND gate, respectively.

$$M(A,B,1)=A+B,\ M(A,B,0)=AB$$

Hence, more complex logic circuits can be constructed from OR and AND gates. In a QCA inverter, cells oriented at 45° to each other take on opposing polarization. Fig. 3 shows QCA wire, majority and inverter gate respectively.

III. QCA CIRCUIT OPTIMIZATION USING THE SO-GA

In [10] an algorithm is proposed for logic optimization of QCA circuits based on genetic algorithm (SO-GA). This optimization reduces the number of majority gates and inverters. Unlike other implementations that generally use matrix structure for digital circuit, in this method a tree structure is used for chromosome representation. The nodes of the tree are the majority-gates and inverter gates, leaves of the tree can be either logical '1's or Boolean variables. A majority function is denoted by M is defined as:

$$M(A,B,C) = AB+AC+BC$$

As an example, Fig. 4 shows $M(A,C',M(A,B,1)')$.

The fitness function is defined based on the similarity of the chromosome to the expected logical function. In addition, the chromosome that has fewer nodes is a better solution. Suppose n is the number of variables, F is the Boolean function, and C is the chromosome. The fitness function is defined as:

$$Fit(C) = \frac{N(F,C)}{2^n} \quad (1)$$

where $N(F,C)$ is the number of identical minterms between chromosome C and function F. If a chromosome has the same minterms as the function F, then the fitness function defined by (1) will become equal to 1. In that case, a different fitness function is used:

$$Fit(C) = 1 + \frac{1}{Nodes(C)} \quad (2)$$

where $Nodes(C)$ denotes the number of nodes in C. The above fitness functions (1) and (2) together can rank the chromosomes according to both their similarity to the Boolean function F and number of nodes in C. The process starts with an initial population created randomly. Evaluation is done by mutation and crossover. Mutation [11]gives another chance to a lost genome in the population to appear in next generation chromosomes. Genome loss happens when it is not produced in the initial population at all or due to crossover. To solve the problem of the genome loss, mutation in some chromosomes is done with some specific probability. In other words, mutation, changes one or more genomes in population with some other ones from problem space. But the job is too complicated in our structure. For example, suppose we want to change the inverter node with the majority node. Because there is a difference between the number of inputs in the structure of inverter and majority, this job becomes intricate. So a new method of mutation is used which works properly for this chromosome structure. In this method a random chromosome is produced and

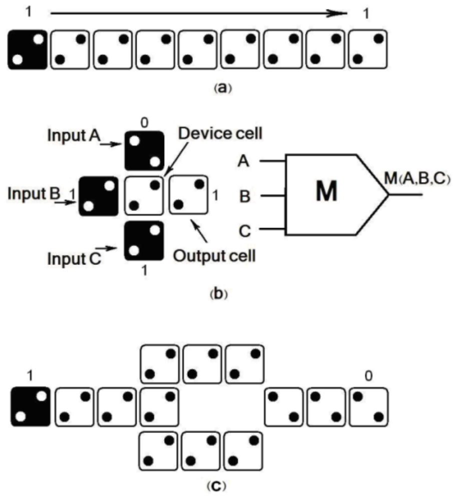

Figure 3. a) QCA wire b) QCA majority gate c) inverter gate

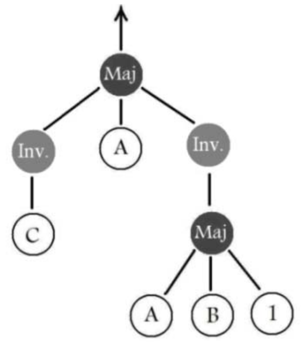

Figure 4. A chromosome for $M(A,C',M(A,B,1)')$

is added to the population instead of some chromosomes that has the worst fitness in population. This job will be done in some specific probability in generations. The new chromosomes can bring back the utilization chance of lost genomes and avoid local minima. For crossover, a random node and its sub-tree in one chromosome is exchanged by a random node and its sub-tree in another chromosome. Figs. 5 and 6 show a two point crossover.

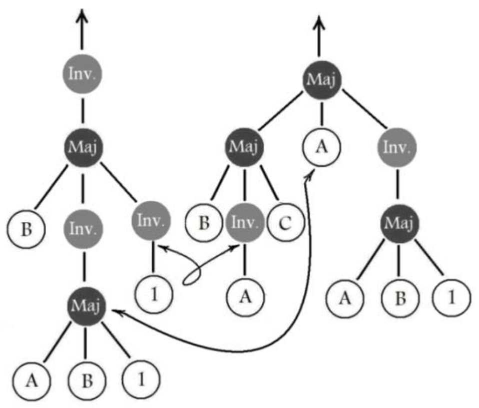

Figure 5. Before crossover

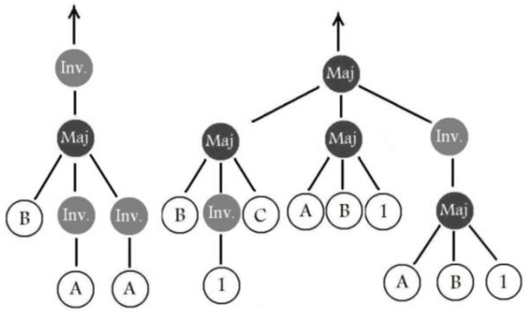

Figure 6. After crossover

IV. PROPOSED ALGORITHM

As mentioned before, SO-GA algorithm is efficient only for single-output circuits. But in applicable environment, most of circuits have more than one output. Optimizing multi-output circuits in such a way that each output is independently optimized, does not result in an optimum and efficient solution. So in this paper, we extend SO-GA for optimizing multi-output circuits.

Suppose we want to optimize the logic of multi-output circuit. First consider a circuit with two outputs, *out1* and *out2*. A tree structure with two outputs is used as chromosome representation. Fig. 7 illustrates such a chromosome. In this chromosome, outputs are as below:

$$Out1=M(M(A',B,C),A,M(A,B,1)')$$

$$Out2=M(M(A',B,C), M(A,B,1),0)'$$

Before explaining the algorithm it is necessary to define two functions: *Nodes(C)* and *Combine(ch1,ch2)*.

Nodes(C) returns the number of nodes in chromosome *C* as mentioned before. *Combine(ch1,ch2)* gets two chromosomes as input and returns a chromosome with two outputs that is a combination of *ch1* and *ch2*. The common nodes of *ch1* and *ch2* appear once in the combined chromosome. Fig. 8 is an example for explanation of this function.

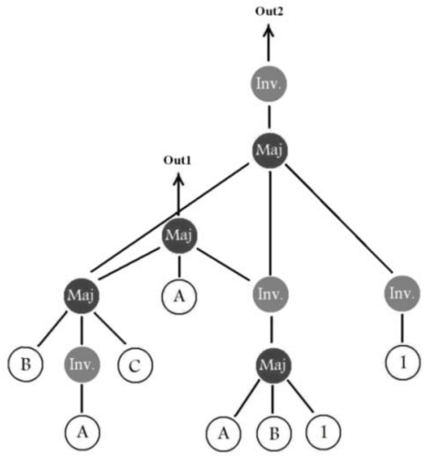

Figure 7. A two output chromosome

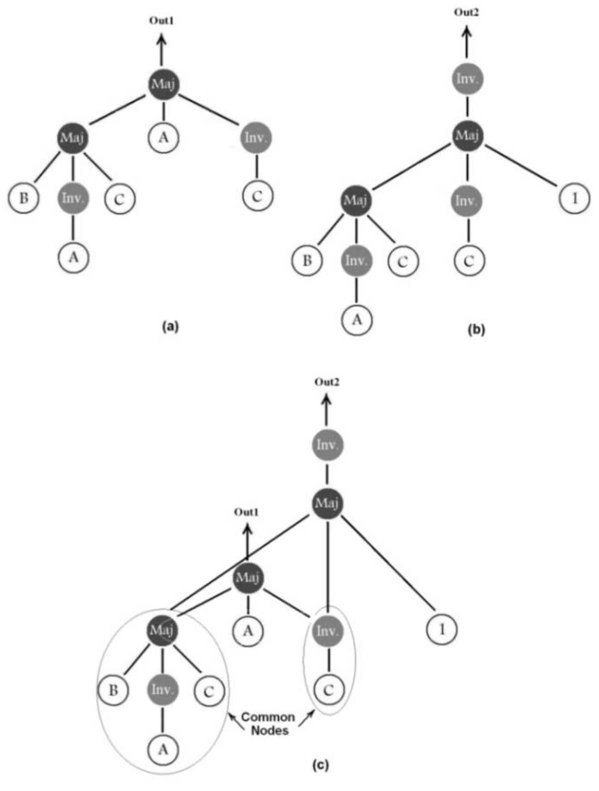

Figure 8. (a) *ch1,* (b) *ch2,* (c) combined chromosome

The algorithm is started by running SO-GA for the first output (*out1*). We run the SO-GA with sufficient number of generations to get some chromosomes with the fitness more than one, and then we store them. We call each of these chromosomes *CorrectFinalChromosome1*. Then for each *CorrectFinalChromosome1*, the proposed algorithm is run. Fig. 9 shows the pseudo code of the algorithm and Fig. 11 shows its flow chart.

In this algorithm the set of chromosomes for *out2* is called *Chromosome2*; and *Chromosome*[*j*,*i*] denotes the *i*-th chromosome in the *j*-th generation. A *FinalChromosome2*[*j*,*i*] is obtained by combining a *CorrectChromosome1* with a *Chromosome2*[*j*,*i*]. Then, *FinalChromosome2* with the best fitness is stored. We call each of these chromosomes *CorrectFinalChromosome2*.

After running the algorithm for each *CorrectFinalChromosome1* the best chromosome among the *CorrectFinalChromosomes*2 is selected as the final solution.

```
Generate a random first generation for Chromosome2
For j=1 to number of generations do
    For i=1 to number of chromosomes in generation do
        Generate a random number num_of_cuts between 1
            and Nodes(CorrectFinalChromosome1 )
        //merge operation(Fig 10)
        For k=1 to num_of_cuts do
            Generate a random number M between 0 and
                Nodes(Chromosome2[j,i])
            Generate a random number N between 0
                and Nodes(CorrectFinalChromosome1).
            If one of the variables M, N is zero,
                do nothing.
            Else replace the node number M and its sub-
                tree in Chromosme2[j,i] by the node
                number N and its sub-tree.
                in the CorrectFinalChromosome1
                //notice that CorrectChromosome1 does
                not change
        FinalChromosme2[j,i]=
        Combine(CorrectFinalChromosome1
        ,Chromosme2[j,i])
    Do the selection, crossover and mutation for
    Chromosome2 (in the same way as SO-GA in order to
    generate the next generation of Chromosome2). The
    number of nodes in the FinalChromosome2 is used in
    the calculation of fitness as follows:
        Nodes(FinalChromosome2) =
        Nodes(CorrectChromosome1) +
        Nodes(Chromosome2) – (number of common
        nodes)

Store CorrectFinalChromosome2. (the best chromosome in the
last generation).
```

Figure 9. The proposed algorithm

The algorithm is apriority extendable to circuits with *n*-outputs. To do this, in the *i-th* step, the algorithm in Fig. 9 is run for each of *CorrectFinalChromosome i-1* to obtain *CorrectFinalChromosomes i*. Finally, after *n* steps the best chromosome among the *CorrectFinalChromosome*s *n* is selected as the final solution.

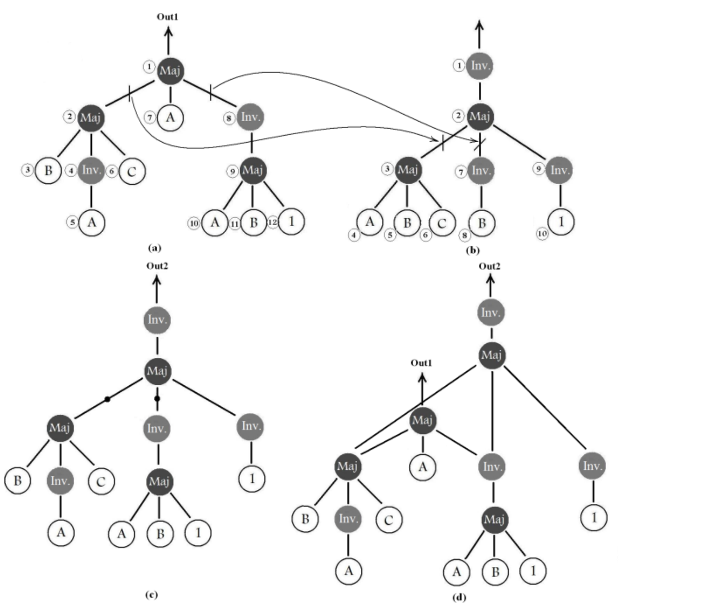

Figure 10. The merge operation for *num_of_cuts*=2 and *N*(1)=2 and *N*(2)=8, *M*(1)=3, *M*(2)=7. (a) correct final chromosome 1, (b) Chromosome2, (c) chromosome 2 after getting some common nodes from correct chromosome 1, (d) the final chromosome2 (with two output)

V. SIMULATION RESULTS

We applied our proposed algorithm on 1000 3-input/2-output and 1000 4-input/2-output Boolean functions that were randomly generated. For each function, first, we applied the SO-GA for both outputs, independently. Then, we applied our proposed MO-GA. In both cases, we counted the number of required gates. The Roulette Wheel algorithm is used for selection. The number of population and the number of generations are100 and 5000, respectively. In 42% of 3-input circuits, and in 38% of 4- input circuits, our algorithm achieves up to 24% reduction of gate counts. In the remaining cases, the two algorithms produced the same results. This happens when the logic circuits implementing each output have no common parts. Table. I shows a sample of comparisons between the results of the two algorithms.

TABLE I. Simulation results for a function out1=∑m(0,2,4,7), out2=∑m(0,2,3,4)

Out1(minterms)	∑m(0,2,4,7)
Out2(minterms)	∑m(0,2,3,4)
Result of both algorithms for out1	M(M(C,1,M(A,B,C))’,M(A’,C,M(1,B,A’)),A)
Result of SO_GA for out2	M(M(B,C,1),A,M(1,A,B)’)’
Result of MO_GA for out2	M(M(1,B,A),A’,M(1,C,M(A,B,C)’))

	required gates for out1	required gates for out2	Common gates	Total number of gates
SO_GA	7	5	0	7+5=12
MO_GA	7	6	3	7+6-3=10

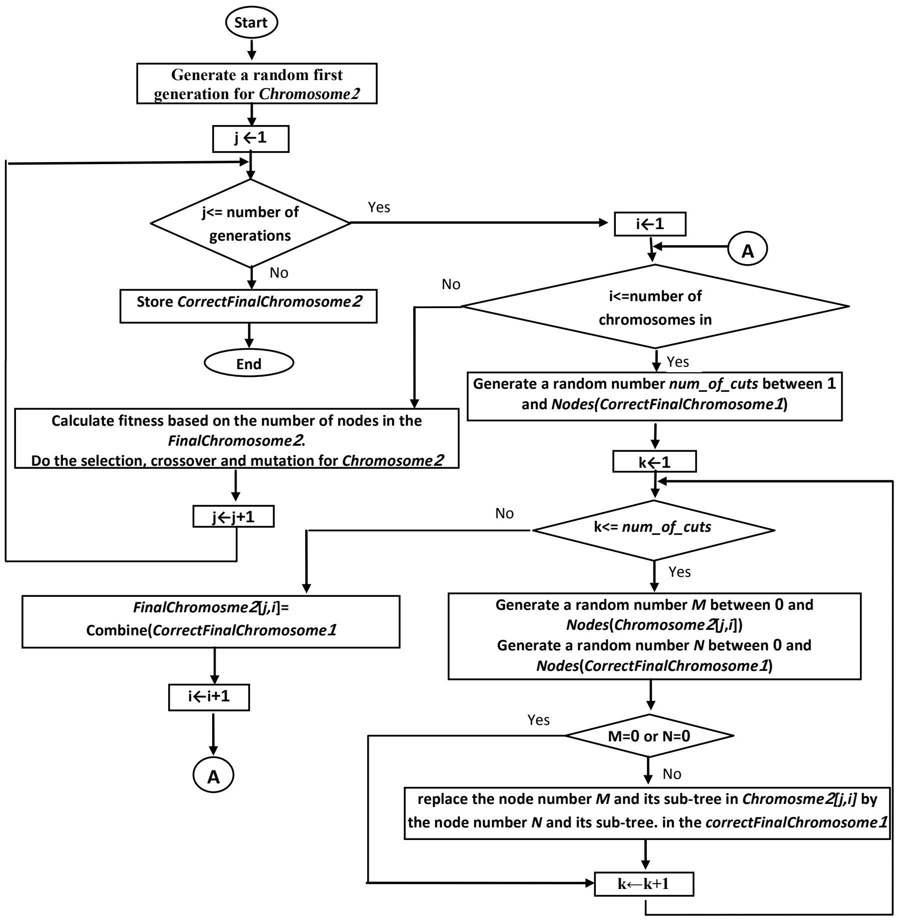
Start
Generate a random first generation for Chromosome2
j←1
j<= number of generations
Yes
No
Store CorrectFinalChromosome2
End
i←1
A
No
i<=number of chromosomes in
Yes
Generate a random number num_of_cuts between 1 and Nodes(CorrectFinalChromosome1)
Calculate fitness based on the number of nodes in the FinalChromosome2. Do the selection, crossover and mutation for Chromosome2
k←1
j←j+1
No
k<= num_of_cuts
Yes
FinalChromosme2[j,i]= Combine(CorrectFinalChromosome1
Generate a random number M between 0 and Nodes(Chromosome2[j,i]) Generate a random number N between 0 and Nodes(CorrectFinalChromosome1)
i←i+1
Yes
M=0 or N=0
No
A
replace the node number M and its sub-tree in Chromosme2[j,i] by the node number N and its sub-tree. in the correctFinalChromosome1
k←k+1

Figure 11. The flowchart of the proposed algorithm

VI. CONCLUSION

In this paper, a method is proposed for reducing the number of gates in multi-output QCA circuits based on genetic algorithm. Multi-output QCA circuits are common and reducing the number of required gates to implement them is of much interest. Simulation results for two output circuits show that our method produces fewer number of gates compared to independent runs of SO-GA. The algorithm is easily extendable to circuits with more than two outputs.

REFERENCES

[1] N. Gergel, S. Craft, and J. Lach, "Modeling QCA for area minimization in logic synthesis," Proc. ACM Great Lakes Symposium VLSI, April 2003.

[2] M. Wilson , nanotechnology: Basic Science and Emerging Technologies, London, U.K.: Chapman & Hall, 2002.

[3] C. S. Lent, "Quantum cellular automata, Nainventerechnology, " vol. 4, pp. 49–57, 1993.

[4] C. S. Lent and P. D. Tougaw, A device architecture for computing with quantum dots, Proc. IEEE, vol. 85, pp. 541–557, Apr. 1997.

[5] G. Toth and C. S. Lent, "Quasiadiabatic switching for metal-island quantum-dot cellular automata," J. Appl. Phys., vol. 85, no. 5, pp. 2977–2984, 1999.

[6] I. Amlani et al, "Experimental demonstration of a leadless quantum-dot cellular automata cell," Appl. Phys. Lett., vol. 77, no. 5, pp. 738–740,2000.

[7] Rui Zhang, P. Gupta, and N. K. Jha, "Synthesis of majority and minority networks and its applications to QCA, TPL and SET based nainventerechnologies," Int. Conference on VLSI Design, 2005.

[8] R. Zhang, K. Walnut, Wei Wang and G. Jullien, "A method of majority logic reduction for quantum cellular automata," IEEE Trans. Nainventerechnology, Vol. 3 , no. 4, Dec. 2004.

[9] Zhi Huo , Qishan Zhang , Zhi Huo and Qishan Zhang, "Logic optimization for Majority Gate-Based Nanoelectronic Circuits," ISCAS 2006.

[10] M.R. Bonyadi , S.M.R. Azghadi, N.M. Rad, K. Navi and E. Afjei, "Logic optimization for Majority Gate-Based Nanoelectronic Circuits Based on Genetic Algorithm," International Conference on Electrical Engineering, ICEE, 2007.

[11] Mitchell Melanie , An Introduction to Genetic Algorithms, A Bradford Book The MIT Press, Cambridge, Massachusetts, London, England, fifth printing 1999.