EXPO: AN EXPERT SYSTEM FOR OPTIMIZATION OF MULTI-LEVEL LOGIC.

Marek Perkowski, Portland State University, Portland , Oregon, USA.
Jyh-Ming F. Lin, National Chiao-Tung University, Republic of China.
Andrzej Wieclawski, Warsaw Technical Univarsity, Warsaw, Poland.

Abstract— This paper presents a method and a computer program for optimization of logic networks. The optimization of network is performed in a frame of search in solution trees, known from Artificial Intelligence. Special tree-search strategy is applied that permits user-defined selection and permutation of operators. Backtracking is implemented with use of operators which are inverse to the forward researching operators. Program EXPO is implemented in Franz Lisp on Vax 11/750 computer. The input to the program and the resultant network are multilevel and are realized in semiregular Weinberger-style gate matrix layout, which may be converted from or to standard circuits.

INDEX TERMS: Software engineering, VLSI design, design optimization, expert systems, computer aided design.

I. INTRODUCTION

The multilevel logic portion of a chip implementing a set of Boolean functions and sequential circuits occupies ordinarily a considerable part of chip area. Considerations include logic gates number, total area, gate oxide area, circuit redundancy, delay constraints, testability, wirability and power were criterias when performing the optimization. The algorithms to implement these consideration in optimizing multilevel network has long been the subjects of investigation.

During past years, many methods and automatic synthesis systems have been proposed for this purpose. The ALERT system concentrated on compiling a high-level described architecture into Boolean functions and then converting them into circuits [12]. The APDL system at Carnegie-Mellon University did the similar job and was extended to the experiment system CMUDA with implementation of data portion of a DEC PDP-8/E [1]. The LSS system took a description language at the register transfer level, transformed it into gates interconnection of the target technology [8, 7]. Total area, gate oxide area and delay time were considered to be summed as cost function of complex logic gate [2]. REDUND, a general constant propagation program in IBM, was used to detect all redundancies and then simply the logic [4]. After algebraic tools for designing multilevel logic, "weak division" and "strong division", were developed by Brayton and McMullen, Bartlett and Hatchtel used "weak division" to design gate array and standard cell in C [15].

There were some system proposed recently. SOCRATES system in GE used "weak division" and "optimize expert system", took timing constraints under consideration, can synthesize and optimize multilevel network [5]. MACDAS system using TVFG was proposed to design multi-level logic by transforming the two-level AND-OR circuits output from TVFG [13]. LSS system continuous to run and tried to consider all technology requirement when synthesizing logic, and proposed that the new technology adaptability is one of the marks of a design automation system's success [14].

Since the algorithmic methods to design optimal random-logic circuits are inefficient, we can see much effort were spent recently on expert system for these tasks [5]. The method presented in this paper was thought by the way of expert system, too. We also take expert system method as the key to adapt wide range criteria to the optimization.

The paper presents a method and a computer program for optimization of logic network with respect to their efficient implementation in ratioless Si-gate NMOS technology. It is assumed that the resultant network is multilevel, consists of complex negative gates, and is realized in semiregular Weinberger-style gate matrix layout.

The program transforms the network of simple gates to the form convenient for creating complex gates (negative gates) suitable for random logic implementation. By a negative gate we understand a negation of a positive function, where positive function is a unate function of affirmative arguments. Such functions can be efficiently realized in VLSI technologies.

The user can declare operators to optimize a network. Each operator transforms the selected segment of the logic network, that matches the left pattern of a transformation rule and fulfills certain conditions. The novelty of our approach is that operators describe not single logic transforms but whole classes of them. The effectiveness of the transformation depends on the proper selection of operators. The optimization of the network is performed in a frame of search in solution trees, known from Artificial Intelligence.

Special tree-search strategy is applied that permits for user-defined selection and permutation of operators. Backtracking is implemented with use of operators which are inverse to the forward searching operators.

Program is implemented on in Franz Lisp on Vax 11/750 computer. Results of running this program on benchmark networks will be presented.

II. OVERVIEW

The input circuit can include AND, NAND, OR, NOR and NOT gates, then it must be input to the system in the form of data base. The initial phase of optimization applies operators to transform the input logic network that matches the left pattern of a transformation rule and fulfills certain conditions. The pattern are general and are not restricted to gates with a specified number of inputs. The effectiveness of the transformation depends on the proper selection of operators. The optimization of the network is performed in a frame of search in solution trees.

At the first phase of the system, the depth-first strategy with one successor is applied in the program. It means that for the node created as the last in the solution tree, any applicable operator is applied, the same is done for the successor node. Next a backtrack in the solution tree is done and the procedure is iterated. After tree search is finished, the best solution is subject to next optimizations.

To limit the memory size, only the actual network and its transformed parts are stored. To creat the initial network, the "inverse operators" are applied in the backtracking mode. At this satge , most of the operators are of type that generally optimize area or cost considerations.

III. 2 TRANSFORMING OPERATORS

There exits in the system currently 19 operators, which can be added, removed and modified. The operators are writtern based on logic laws like deMorgan's Theorem, rules for shiffting inverters through the network, etc. The operators apply pattern matching in graphs, realized with an aid of Lisp data structures. The operators are generally of the type:

(condition, left pattern $\rightarrow$ right pattern)

Figure 1 to Figure 3 present examples of operators from TSS1 to TSS19. Operator TSS1 joins gate AND with the following NOT gate to creat a NAND gate. TSS2 is an identical operator for OR gate. Operator TSS3 removes OR gates which lead their outputs only to a NOR gate. Similar operators exist for NAND, OR, AND gates on second level, they are TSS4, TSS5 and TSS6. Operator TSS7 removes two serially connected inverters.

TSS8, TSS9 TSS10 and TSS11 are operators that apply deMorgan Theorem. The effectiveness of applying these operators requires that there exist gates n to m, o to p or xi to xr. Otherwise application of this operator is nonprofitable. Operator TSS12 decreases number of inputs in the gates of type NAME where NAME can be AND or OR gate. The necessary condition of its application is that the list of gates m inputs includes all gates denoted by u1 to ut. Operator TSS13 serves for joining the gates performing the same logical function while NAME can be AND, OR, NAND, NOR or NOT gate. Operators TSS14 to TSS19 serve purely for cooperation with another operators. They don't cause direct simplification of the logical network.

IV. INVERSE OPERATOR

Each operators TSS1 to TSS19 has its inverse operator. The behavior of the inverse operator is such, that subsequent application of opertor and its inverse operator doesn't change the network. The behavior of inverse operator is explained in Figure 4. As we see, direct application of operator TSS3 requires storing of data ((n1,...,nk),...,(m1,...ml)) to enable subsequent backtracking.

V. ORGANIZATION OF DATA BASE

The data base includes descriptions of gates in the form:

- type of the gate,
- its identifier in the form Lxxxx, where L-letter, x-digit,
- list of gates' inputs,
- list of gates' outputs.

The LISP property list of the identifier atom is used to store the respective gate description. The respective properties are stored under indicators TYPE, INPUT, OUTPUT, as shown in Figure 5.

Figure 6 lists the data base of the circuit structured with a four-input AND gate followed by a NOT gate .

VI. OPTIMIZATION ALGORITHM

The optimization process follows the given algorithm:

1. Read the data base.
2. Find and store the applicable operators for the initial operator selected by the user in the initial node of the search tree.
3. If the selected operator cannot be applied (attempt of its application fail) then go to 20.

4. Store this operator and arguments for inverse operator in the new node of the tree.
5. Store this node and all applicable operators, that can follow the selected operator.
6. Actual node := new node.
7. Select first operator among those applicable in the actual node of the solution tree.
8. If there is no such operator, then go to 11.
9. Delete the selected operator from the set of applicable operators for the actual node.
10. If selected operator can be applied (attempt succeeded) then go to 4, else go to 7.
11. Store operators whose application leads to the actual node of the tree.
12. If the last executed operator was an inverse one then go to 15.
13. Apply an algorithm, transforming the actual network into a network of complex gates.
14. Store the last solution as the minimum solution.
15. Apply the operator inverse to the last successfully applied forward operator.
16. If this is not the operator inverse to the initial operator, then go to 7.
17. Select the first operator among operators applicable in the initial node of the tree.
18. If there is such an operator, then go to 3, else end.

The example of application of the above algorithm is presented in Figure 8. Only SS2 and SS4 are the solutions, while none of the operators applicable in these nodes can be applied in them.

VII. TRANSFORMATION TO NEGATIVE GATES

After application of selected operators TSS1-TSS19, the network is optimized and ready for negative gates transformation. At this stage the complex gates are created and the circuit's segments obviously nonoptimal are simplified. The terminal multilevel network cannot include the positive gates, that is, those which do not terminate with the inverter. The gates are transformed in the sequence of their type: OR, AND, NAND, NOT, INPUT and OUTPUT. This order results from the fact that only positive gates AND and OR can be used to creat complex gates. If all positive gates have been then used there is no possibility of creating complex gates, which essentially simplifies further transformation of the network to the form realized in MOS technology.

The transformation of the given network's segment consists in checking the direct ancestors and successors of each gate and transfomation of this gate according to the information thus obtained. For instance the OR gate can be either transformed to the complex gate , for instance when such gate leads its output only to AND or NAND gates, or to the negative gate, for instance by joining with inverter. The most of the possible variants of transformation exists for gates OR and AND as well as for NOT gate. For the second group a big amount of transfomations result from necessity of removing serial or parallel connection of inverters. Such situations are possible, while for instance by creating negative gates generated are additional NOT gates.

VIII. PERFORMANCE EVALUATION AND DISCUSSION

Table 1 shows some benchmarks used to evaluate the performance of the program. They were translated from circuits or circuits' description to the data base described in the paper. Then the program was executed.

EXPO was run on each circuit fully automatically. THe user's only interaction with the system was fitting the applicable operators before running, as shown in Figure 7. Table 1 shows the experiment results. The "used time" are quoted in terms of seconds under the circumstance of VAX 11/750.

The example ALUPLA was first run out with 155 gates, after fixing the applicable operators, it can be run out with 151 gates. The same process was executed on runing example WIELA, the first result was 21 gates, later on we run out as 19 gates. The circuit 9SYM is a symmetric function which was proved very difficult to minimize, neither the EXPO failed to minimize it.

The performance of promgram execution depends on the selection of operators possible for application in the given node of tree. The number of operators should be minimal, but we also want the quasiminimal solution not to differ much from the optimal one. Application of operators, whose actions on the same element anihilate mutually should be also avoided. For instance, one of possible choices of the rules for applicable operators can be as the Figure 7 shows. The rules in Figure 7 can be interactively modified for certain nodes by the user depending on information of the circuit. They can be also automatically modified while searching the tree on the base of user parameters and search parameters.

IX. CONCLUSION

The system EXPO presented here can be applied for optimization of logic networks including 100 to 200 simple gates (AND, OR, NOR, NAND, NOT) which corresponds to 20 to 25 complex gates. Larger networks require high processing time, as well as memory usage.

To work with the benchmarks from some design conferences, the circuits' description must be translated to the data base presented in this paper. We hope to automata this translation in the future.

There are also many important problems related to the selection and implementation of the operators and optimizing programs, not discussed in the paper, like: design with inverterless positive gates which lead outputs to more than one complex gates, design with multiple-phase clocks, also design with exclusive-or operator, etc. These cases require certain restrictions for applying operators and modifying the programs, they appear as our future goals.

References

[1] A. Parker, D. Thomas, D. Siewiorek, M. Barbacci, L. Hafer, G. Leive, J. Kim, "The CMU Design Automation System - An Example of Automated Data Path Design," Proc. 16th Design Automation Conf., San Diego, CA, 1979, pp. 73-80.

[2] A. Wieclawski, "Optymalizacja ukladu logicznego przeznaczonego do realizacji w technologii MOS," Institute of Electron Technology, Technical University of Warsaw, 1982.

[3] C. Mead, L. Conway, "Introduction to VLSI Systems", Addison-Wesley Publishing Company, 1980.

[4] Daniel Brand, "Redundancy and Don't Cares in Logic Synthesis," IEEE TRANSACTIONS ON COMPUTERS. VOL. C-32. No.10, Oct. 1983, 947-952.

[5] David Gregory, Karen Bartlett, Aart de Geus, Gary Hachtel, "SOCRATES: A System for Automatically Synthesizing and Optimizing Combinational Logic," IEEE 23rd Design Automation Conf. 1986, 79-85.

[6] E. J. McCluskey, Jr., "Minimization of Boolean Functions," THE BELL SYSTEM TECH. J. Nov. 1956, pp. 1417-1443.

[7] John A. Darringer, Daniel Brand, John V Gerbi, William H. Joyner, Jr., Louise Trevillyan, "LSS: A System for Production Logic Synthesis," IBM J. Res. Develop. 28, No.5, Sep. 1984, 537-544.

[8] John A. Darringer, William H. Joyner, Jr., C. Leonard Berman, Louise Trevillyan, "Logic Synthesis Through Local Transformations," IBM J. Res. Develop. 25, No.4, July 1981, 272-280.

[9] John A. Darringer, William H. Joyner, Leonard Berman, Louise Trevillyan, "Experiments in Logic Synthesis," Proc. IEEE International Conf. on Circuits and Computers ICCC80, Port Chester, NY, 1980, pp. 234-237A.

[10] Karen Bartlett, William Cohen, AArt De Geus, Gary Hachtel, "Synthesis and Optimization of Multilevel Logic under Timing Constraints," IEEE TRANSACTIONS ON COMPUTER AIDED DESIGN. VOL. CAD-5, No.4, Oct. 1986, 582-595.

[11] Shirakawa, N. Okuda, T. Harada, S. Tani, H. Ozaki, "A Layout System for Random Logic Portion of MOS LSI", IEEE ACM 1980.

[12] Theodore D. Friedman, Sih-Chin Yang, "Methods Used in an Automatic Logic Design Generator (ALERT)," IEEE TRANSACTIONS ON COMPUTERS, VOL. C-18. No.7, July 1969, pp. 593-614.

[13] Tsutomu Sasao, "MACDAS: Multi-level AND-OR Circuit Synthesis using Two-variable Function Generators", IEEE 23rd Design Automation Conf. 1986, pp. 86-93.

[14] William H. Joyner, Tr., Louise H. Trevillyan, Daniel Brand, Theresa A. Nix, Steven C. Gundersen, "Technology Adaptation in Logic Synthesis," IEEE 23rd Design Automation Conf. 1986, pp. 94-100.

[15] Koren A. Bartlett, Gray D. Hachtel, "Library Specific Optimization of Multi-level Combinational Logic," Proc. IEEE Int. Conf. on Computer Design. Oct. 1985, pp 411-415.

```
(bdinit '((input b1 (nothing)(and b5))
          (input b2 (nothing)(and b5))
          (input b3 (nothing)(and b5))
          (input b4 (nothing)(and b5))
          (and b5 (input b1 input b2 input b3 input b4)(not b6))
          (not b6 (and b5)(output b0))
          (output b0 (not b6)(nothing)))
```

Figure 6. Example of data base.

```
after TSS1, (TSS2, TSS8, TSS9, TSS10, TSS11) are applicable
TSS2 -> (TSS1 TSS8 TSS10 TSS11)
TSS3 -> (TSS5 TSS13 TSS12 TSS8 TSS9 TSS10 TSS11)
TSS4 -> (TSS6 TSS13 TSS12 TSS8 TSS9 TSS10 TSS11)
TSS5 -> (TSS4 TSS13 TSS12 TSS8 TSS9 TSS10 TSS11)
TSS6 -> (TSS3 TSS13 TSS12 TSS8 TSS9 TSS10 TSS11)
TSS7 -> (TSS3 TSS4  TSS5  TSS6 TSS11 TSS10 TSS9 TSS8)
TSS8 -> (TSS1 TSS2 TSS3 TSS4 TSS5 TSS6 TSS9 TSS10 TSS16 TSS19)
TSS9 -> (TSS1 TSS2 TSS3 TSS4 TSS5 TSS6 TSS9 TSS11 TSS18 TSS19)
TSS10 -> (TSS1 TSS2 TSS3 TSS4 TSS5 TSS6 TSS9 TSS11 TSS16 TSS19)
TSS11 -> (TSS1 TSS2 TSS3 TSS4 TSS9 TSS10 TSS17 TSS18 TSS19)
TSS12 -> (TSS8 TSS9 TSS10 TSS11)
TSS13 -> (TSS1 TSS2 TSS8 TSS9 TSS10 TSS11)
TSS14 -> (TSS7 TSS11 TSS10 TSS9 TSS8)
TSS15 -> (TSS1 TSS2 TSS7 TSS11 TSS10 TSS9 TSS8 TSS16 TSS17)
TSS16 -> (TSS1 TSS2 TSS7 TSS14 TSS9 TSS11 TSS10 TSS8 TSS15)
TSS17 -> (TSS1 TSS2 TSS7 TSS14 TSS10 TSS11 TSS9 TSS8 TSS15)
TSS18 -> (TSS1 TSS2 TSS3 TSS4 TSS5 TSS6 TSS10 TSS11 TSS9 TSS8)
TSS19 -> (TSS1 TSS2 TSS3 TSS4 TSS5 TSS6 TSS11 TSS10 TSS9 TSS8)
```

Figure 7. Applicable operators.

```
Circuit name Initial desp. Final desp. operators executed time used
------------ ------------- ----------- ------------------ ---------
```

alupla	162 gates	151 gates	34	330 sec
wiela	24 gates	19 gates	19	22 sec
f51m	104 gates	103 gates	6	65 sec
z4ml	50 gates	50 gates	3	30 sec
9sym	162 gates	162 gates	6	90 sec
a66m	55 gates	52 gates	8	70 sec

Table 1. Experiment results.

Figure 1: