MINIMIZATION OF NETWORKS WITH COMPLEX GATES

Craig Files +, Andre Reis, Marek Perkowski +, Michel Robert, Daniel Auvergne
Laboratoire d'Informatique, de Robotique et de Microelectronique de Montpellier, France.
+ on sabbatical leave from Portland State University, Department of Electrical Engineering, USA

April 15, 1996

Abstract

This paper describes a top-end minimization algorithm that is used to optimize a complex gate network. The basis for this approach, is the use of a library-free layout generator, instead of a standard pre-characterized library, such as the traditional layout synthesis method of standard-cells. The library-free cells are made up of static CMOS complex gates, where the quality of the final layout is based on the initial network (input to the layout generator). That is, if the initial network is given in a form that cannot be easily implemented in complex gates, then the final layout will not be good. In this paper, we present a method that produces "high-quality" initial networks, using a new approach based on minimizing the complex gates.

Mathematically, a complex gate is one that realizes a negation of an arbitrary positive unate function. This paper describes a method of minimization of single-output, two-level networks from unlimited complex gates, implemented in the program COMPLEX. This program is used as a preprocessor to TROPIC, a automatic layout generator [2, 3]. The presented method is applied to completely specified functions to minimize the number of gates, variables, transistors and values of other parameters.

1 Introduction

The Library Free Technology Mapping (LFTM) problem is the extensive use of Static CMOS Complex Gates(SCCGs) instead of using a pre-characterized library, and three methodologies have already been proposed [6, 1, 5]. All of these methods intend to exploit the great number of different logic gates that can be implemented in CMOS technology for a given limitation in the number of serial transistors (87 different gates with three serial transistors and 3503 with four [6]). The method proposed in [6] is based on pattern matching and the methods proposed in [1, 5] use greedy structural approaches based on local limitation of serial transistors while constructing a SCCG structure.

The most important point of this approach is that the techniques are developed for a layout generator. Therefore, it can be used in high-performance applications that are usually optimized "by hand". The emphasis is to obtain solutions of a very "high quality", with the assumption that it may increase the design time, and/or limit the size of the inputed networks.

The method that is proposed in this paper is for single output, completely specified Boolean Functions. Note, this is a general method that can be used to design two-level and multi-level, multi-output circuits from complex gates for both completely and incompletely specified Boolean functions.

This paper is organized as follows: Section 2 is a short summary of previous research in this area. Section 3 gives a list of parameters used to optimize the results from the program COMPLEX. In Section 4, the algorithm for minimizing the complex gate network is given. Section 5 shows the experimental results of this method, and Section 6 concludes the paper.

Table 1: Number of SCCGs with limited serial transistors, from Detjens

		Number of Serial PMOS Transistors				
		1	2	3	4	5
Number of Serial NMOS Transistors	1	1	2	3	4	5
	2	2	7	18	42	90
	3	3	18	87	396	1677
	4	4	42	396	3503	28435
	5	5	90	1677	28435	125803

2 Previous Research

The LFTM problem is the task of finding a set of SCCGs that represents a circuit, obeying the following two conditions:

1. Each SCCG has less than a given number of serial transistors.
2. The overall number of transistors is as small as possible (i.e., the number of transistors is the main component of the cost function).

The number of different SCCGs for a given limitation in the number of serial NMOS and PMOS transistors was calculated in [6] and it is shown in Table 1. The set of SCCGs with a maximum of three serial NMOS transistors and four or less PMOS transistors is denoted by SCCG(3,4). From Table 1, the cardinality of SCCG(3,4) is $|\mathrm{SCCG}(3,4)| = 396$. Due to body effect, a maximum of four serial PMOS transistors and five NMOS serial transistors is allowed, resulting in 28,435 different SCCGs.

There have been several approaches developed to reduce the size of SCCGs. A more complete description of these methods can be found in [4]. The first, a pattern matching approach, uses pattern matching techniques to realize the SCCG network. The problem with this approach is that the number of patterns is greater than the number of SCCGs. A Boolean matching approach is done by comparing the logic functions of a sub-set of the subject description with the logic functions of library elements. The problem with this method is the number of libraries is limited to gates with 11 or less inputs. In practice, though, only gates with a maximum of 6 inputs are used. Because SCCG(3,3) contains SCCGs with 9 inputs and SCCG(4,4) can have 16 inputs, the Boolean matching approach cannot be fully utilized to perform LFTM. A greedy structural approach has also been proposed. In this method, the network is decomposed into a set of sub-networks by locally considering the number of serial transistors in use. Thus, a problem is that it is based only on local assessment, and not other global factors that may produce better results for LFTM. A process of negative logic network synthesis has also been proposed, where the methodology minimizes the number of negative cells as a primary objective and the number of interconnections as a secondary objective. This method, however, may produce complex cells that will not work properly with reasonable gate-delay time. This effect occurs when the number of serial transistors are excessive. Last, a technique used to synthesize three level NAND networks, also known as TANT networks. This approach produces a minimal unate function as an estimation of a minimal third level, but does not minimize the third level. Thus, a minimal solution is estimated and not found.

Concluding, the existing approaches do not produce sufficiently "high quality" results that fully utilize the properties of complex gates. This leads to the design methodology presented in this paper, that synthesizes with arbitrary negative logic gates and evaluates the parameters of the negative gate network (and if necessary, split it).

3 Parameters used in COMPLEX

The synthesis and optimization of logic functions realized with complex gates in TROPIC is done in COMPLEX by the minimization of a weighted cost function of several parameters. This means that our set-covering attempts to find a covering that minimizes a certain weighted cost function which is a sum of partial cost functions multiplied by user-assigned parameters. This way the following advantages are achieved:

1. A complex cost function is minimized, which better reflects the complexity of gates applied.
2. The user is able to tune the solutions by selecting the values of numerical weight coefficients.
3. Certain parameters can also be used as restricting conditions, thus further directing

the design and limiting the search space.

4. In the future, this process may be done automatically, where TROPIC would feedback information to COMPLEX.

In library-based design, it is common to minimize the depth of function's realization (the number of levels of logic is minimized for speed), the number and the size of the gates are minimized for area, or power, and the number of transistors is minimized for testability.

In order to evaluate the complex gate net list, a first order evaluation will be done using the following six parameters (we assume to be reasonable high level estimators).

The parameters to be calculated for a multi-input, multi-output module are the following:

P1 *The Total Number of Transistors.* It is assumed that the overall number of transistors in the network is the first order estimator for both area (we assume that the transistor area is constant by intervals) and power (the number of transistors is a relative estimator for the total layout capacitance) of the final circuit.

P2 *The Total Number of Static CMOS Complex Gates (SCCGs).* This parameter is also related to both area and power, because it is an estimator of the number of switching nodes.

P3 *Maximum Number of Serial Transistors in All Gates.* Each SCCG has its number of NMOS and PMOS serial transistors limited to a given number. Too many serial transistors may cause noise margin and speed problems due to increasing the threshold voltage due to body effect. For current technologies, 4 serial PMOS and 5 serial NMOS are within reasonable margins. (This parameter is used as a constraint rather than a component of the weighting cost function.)

P4 *Maximum Fanout of All Gates.* The user must have a means to control the maximum fanout of each SCCG.

P5 *Number of Serial SCCGs in the Module.* This parameter is related to the depth of the net list, and its speed. The logic depth inside each SCCG (and in all of the net list) is not significant.

P6 *Number of Transistors in the Longest Path in the Module.* In addition to the SCCG depth of the net list, the speed of the SCCGs is also important. It is assumed that the number of serial transistors inside each SCCG can be used as a first-order speed estimator.

P7 *The Total Number of Connection Points* with more than two adjacent transistors, other than VDD, Ground and output.

P8 *The Total Ratio Factor. The Normalized Ratio* of a single gate is the ratio of the number of serial to parallel transistors, normalized to a number not smaller than 1.

P9 *The Total Number of External Connections of Gates.*

Our goal is to control these parameters during the logic synthesis phase, and to verify, experimentally, how good they are in practice. Thus, evaluating how these parameters influence respective performance parameters extracted from the layout generated from TROPIC. In enumerative search procedures, such as set covering, many solutions have the same simple cost, and a random one may be selected. Our reasoning is to perform multi-criteria optimization at a slightly increased calculation cost.

4 Complex Gate Minimization

In this section, a SCCG minimization algorithm is presented for single output, completely specified functions. Note, that this method has been extended to multi-output, multi-level functions, realized in limited complex gates using a method called Dissected Pairs [7].

Complex Gate Minimization Algorithm.

1. Find the complement $\overline{f}$ of function f. This reduces the size of the function since an inverter is not needed to make the output of the SCCG positive.

2. Search through all possible minterms in $\overline{f}$. If a minterm has only one prime implicant covering it, the implicant is essential. When this occurs, the minterm and all minterms included in the essential prime are removed

from $\overline{f}$ by changing their values to don't cares. The essential implicant is placed in the set MI($\overline{f}$). This is implemented in COMPLEX by using a BDD to represent $\overline{f}$.

3. The remaining set of minterms is a set that has multiple primes covering them, and thus, the primes are non-essential. All possible primes for a minterm are placed in a *prime implicant covering table.* The prime covers found from this covering table, and from MI($\overline{f}$) produce a set of sufficient implicants that represent $\overline{f}$.

4. For each sufficient implicant, store the value in terms of all positive variable terms, or *kernels.* A kernel is defined as the set of positive literals in a given implicant. For example, given the implicant $x_1\overline{x_2}x_3\overline{x_4}$, the kernel is x_1x_3.

5. Combine all sufficient implicants that have the same kernel. For example, given the implicants with kernel x_1, $x_1\overline{x_2}\ \overline{x_3}$ and $x_1\overline{x_2}\ \overline{x_4}$, the implicants can be combined as $x_1(\overline{x_2}(\overline{x_3} + \overline{x_4}))$. Or it can be placed in the *initial complex implicant notation* of $(x_1)\overline{(x_2 + x_3x_4)}$.

6. Each sufficient implicant, in *initial complex implicant notation,* is placed in the set UF($\overline{f}$). The set UF($\overline{f}$) is then added to the *complex implicant covering table* which is used to find the SCCG with the "highest quality". This covering table is composed of positive kernels from the best cover as columns, and complex implicants that correspond to the positive kernels as rows.

7. Do an item by item *combine* of all the elements in the set UF to create *secondary complex implicants.* The combination of two implicants is done through a Boolean factorization method. In this method, a pair of implicants is combined by ORing the two kernels and ANDing the terms included in the OFF-terms of each implicant. For example, combining $x_4\overline{(x_1 + x_3)}$ and $x_1x_3\overline{(x_4)}$, ORing the kernels gives $(x_4+x_1x_3)$ and ANDing the terms in the OFF-terms gives $\overline{(x_4(x_1 + x_3))}$. Thus, the newly created *secondary complex implicant* is $(x_4 + x_1x_3)\overline{(x_4(x_1 + x_3))}$. There are cases, however, when this factorization method will not work. This occurs when there isn't a negative term that can cover all the OFF-terms and none of the ON-terms in both of the implicants. Thus, this factorization is only executed if the secondary complex term is equivalent to the union of the initial complex terms from which it was created.

   This procedure is recursive, and will complete when there are no new combinations possible. (This happens either when all possible kernels have been combined or when the elements that remain cannot be combined.) At each level of recursion place the new set of *secondary complex implicants* in the *complex implicant covering table.*

8. Solve the *complex implicant covering table* by evaluating the cost of each solution as a weighted function of the parameters given in Section 3. This means, select a subset of rows that covers all columns and minimizes the weighted cost function, of the form $\sum_{i=1}^{9} \alpha_i P_i$.

9. End. (Possibly repeat all steps using f instead of $\overline{f}$, because a reduced set of sufficient implicants will lead to a better solution.)

Example 1.
Given the function $\overline{f}(x_1, x_2, x_3, x_4) = \sum(1, 2, 3, 4, 5, 6, 10, 11, 12, 14)$, shown in Figure 1a. (2) Find all the essential implicants of $\overline{f}$. The minterms 1011 and 1100, produce the respective essential primes, $\overline{x_2}x_3$ and $x_2\overline{x_4}$, as shown in Figure 1b. (3) From the remaining two minterms the cover $\overline{x_1}\ \overline{x_3}x_4$ is selected. (4) The kernels for $\overline{f}$ are $\{x_3, x_2, x_4\}$. (5) There are no implicants with the same kernel, so nothing needs to be done here. (6) Initially, the set UF contains the elements: $(x_3)\overline{(x_2)}$, $(x_2)\overline{(x_4)}$, $(x_4)\overline{(x_1 + x_3)}$ which are placed in the *complex implicant covering table* shown in Figure 1d. (7) By combining the elements in UF, the *secondary complex implicants* are found as $(x_2 + x_3)\overline{(x_2x_4)}$ and $(x_2 + x_4)\overline{(x_4(x_1 + x_3))}$. Notice, that the kernels x_3 and x_4 could not be combined. In this case, if x_3 and x_4 are combined the resulting *secondary complex implicant* would be $(x_3 + x_4)\overline{(x_2(x_1 + x_3))}$ and is shown in Figure 1c. Thus, this does not cover

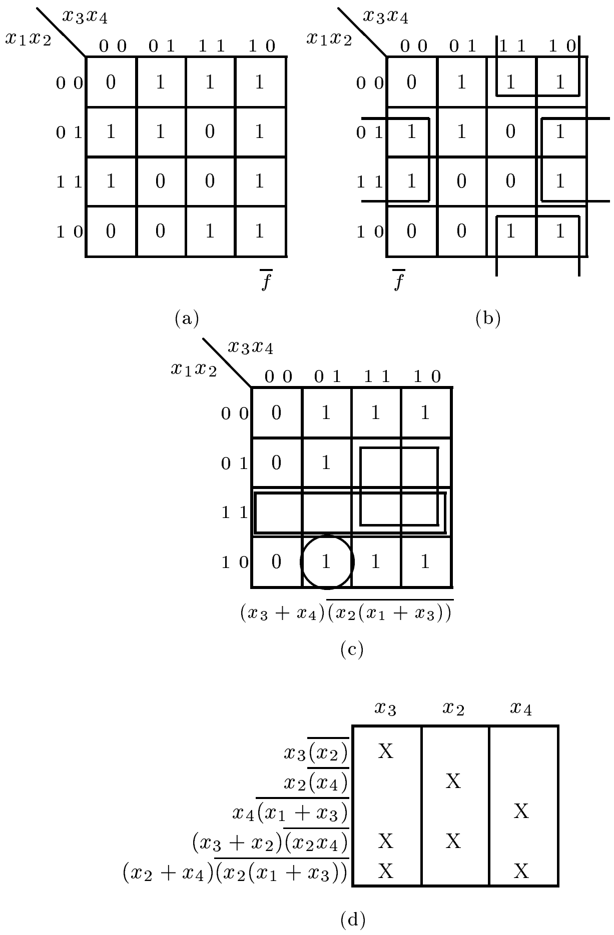

Figure 1: Solving the function in Example 1.

the same set of minterms as the *initial complex implicants.*

(8) The *complex implicant covering table* is shown in Figure 1d. The resulting minimized network is given by the function
$\overline{f} = (x_2 + x_3)\overline{(x_2x_4)} + (x_4)\overline{(x_1 + x_3)}$.

5 Results

The algorithm presented in the previous section has been incorporated in a program called COMPLEX. After running COMPLEX on a number of test functions, the following set was chosen because of their complexity. That is, functions that we feel are a fairly good test of COMPLEX. Most of the functions are eight variable functions, the only exception is the parity function of four variables. The reason for not using an eight variable representation is that COMPLEX generates a huge covering table, 128 kernels by 1000 rows. Even though many of the solutions are exactly the same, COMPLEX is not able to handle this size of covering table, yet. Note, the methodology of this algorithm is 2-level synthesis, and our goal is to only use this method on small functions. Thus, in the future, using it to evaluate functions that have been minimized by using other multi-level methods. This is the reason for evaluating the COMPLEX program on small functions, and doesn't cause us that much concern that COMPLEX is not able to minimize the eight variable parity function.

The results were generated by running COMPLEX on a test function and passing the best result of the function to the program TROPIC. For comparison, SIS, using a multi-level minimization method, was also used as an input to the TROPIC system. The results are shown in Table 2. In the table, the columns under each program are the parameters P1 (number of transistors), P2 (number of SCCGs), P5 (number of serial SCCG), which are described in Section 3. Delay is the estimated timing delay, in nanoseconds, of the resulting function after layout. The other parameters given in Section 3 are not shown because the information found within each of these parameters doesn't give values that can be used for comparison. The logic net-list generated by COMPLEX and SIS is converted to a transistor net-list with the number of serial transistors (P3) set to a maximum of four serial transistors [9, 8]. This transistor net-list is then used as the input to TROPIC.

Table 2: Comparison of results

	SIS multi-level				COMPLEX			
	P1	P2	P5	Delay	P1	P2	P5	Delay
ch22f0	40	9	5	1.88	40	3	3	2.14
ch47f0	94	17	9	4.58	78	6	4	2.65
or_and_chain	28	6	7	2.05	22	4	5	1.75
substr1	54	10	6	2.06	46	6	5	2.04
parity(4 var)	52	10	5	1.90	66	7	5	2.33
ch30f0	66	12	7	3.62	58	5	5	2.63
ch74f1	120	20	10	4.66	82	8	5	3.07
modulus2	96	18	8	3.10	76	10	5	2.70
rnd_m10	148	27	9	3.25	160	21	7	3.68
pal	160	28	7	2.84	320	36	10	6.06

The results show that COMPLEX performed very well compared to the multi-level minimization method SIS. The only function that COMPLEX didn't perform well on was the *pal* function which is a completely symmetric function, much like parity. This is expected since SIS should always beat COMPLEX on these type of functions. Also, for the function *rndm10*, COMPLEX returned a larger transistor count but the number of SCCGs was smaller.

6 Conclusion

A technique to minimize two-level networks from complex gates has been discussed.

A new factorization operation was introduced that finds good secondary complex implicants without creating all possibilities, and thus leads to "high quality" results. However, contrary to the method presented in [4, 7], we are not able to prove that the exact minimum of the first or second level is achieved. One method of improving this approach would be to repeat the algorithm for any set of prime kernels from a minimal prime cover. The comparison of the approximate and exact methods should be performed.

Moreover, the method assumes a two-level circuit of complex gates. Therefore, it is not competitive on large multilevel circuits with high EXOR components, such as parity. The method should be used for "high quality" minimization of functions in a window moved through a multi-level network created with EXOR synthesis methods. If no cost improvement is obtained for certain locations of the window, the original circuit remains and the window is moved to a new location.

We have extended the concept of a dissected pair [4] from a pair of a minterm and a false minterm to a pair of arbitrary one-cube and zero-cube. This permits applying the methods to larger functions, represented as arrays of arbitrary cubes. In particular, minimal implicants and minimal implicates [7].

References

[1] P. Abouzeid, R. Leveugle, G. Saucier, and R. Jamier, "Logic synthesis for automatic layout," *Proc. of EURO-ASIC*, pp. 146-151, 1992.

[2] F. Moraes, N. Azemard, M. Robert, and D. Auvergne, "Flexible Macrocell Layout Generator," *Proc. of 4th ACM/SIGDA Physical Design Workshop,* 1993, Los Angeles (USA), pp. 105-116.

[3] F. Moraes, M. Robert, D. Auvergne, and R. Reis, "An efficient layout synthesis tool for CMOS random logic circuits," *Proc. of IX SBMICRO*, 1994, Rio de Janeiro, Brazil, pp. 48-57.

[4] A. Reis, C.Files, M. Perkowski, M. Robert, and D. Auvergne, "Introduction to Exact Minimization of Networks with Complex Gates for High-Performance Layout Generator "TROPIC"" *Non published report*, LIRMM, Montpellier, France, April 1996.

[5] M. Berkelaar, and J. Jess, "Technology mapping for standard-cell generators," *Proc. ICCAD*, Santa Clara, pp. 470-473, 1988.

[6] E. Detjens, G. Gannot, R. Rudell, A. L. Sangiovanni-Vinccentelli, and A.Wang, "Technology mapping in MIS," *Proc. of ICCAD,* Santa Clara, pp. 116-119, 1987.

[7] C. Files, R. Tinder, A. Reis, and M. Perkowski, "Dissected Pairs, Minimal Implicants and Synthesis of Minimum Multi-Level Multi-Output Networks from Complex Gates," Continuation of this paper, in preparation. Includes multi-level, multi-output design and extensions to power minimization and testability.

[8] A. Reis, M. Robert, D. Auvergne, and R. Reis, "Associating CMOS transistors with BDD arcs for technology mapping," *IEE Electronics Letters*, Vol. 31, No.14, 6th July 1995, pp. 1118-20.

[9] A. Reis, M. Robert, D. Auvergne, and R. Reis, "Extensive using of CMOS complex gates with terminal suppressed BDDs" Journal of the Brazilian Computer Society, No. 2, Vol. 2, November 1995. pp.63-75.