AN ALGEBRAIC APPROACH FOR TRANSISTOR CIRCUIT SYNTHESIS

Hiroaki Yoshida† *Makoto Ikeda*‡ *Kunihiro Asada*‡

†Department of Electronic Engineering
‡VLSI Design and Education Center(VDEC)
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan

ABSTRACT

This paper presents an algebraic approach for transistor circuit synthesis. Our approach relies on a graph structure which encodes multiple circuit structural configurations. The proposed procedure implicitly enumerates possible circuit configurations via algebraic transformations in the graph structure, and efficiently finds a minimum solution among them. Experimental results on a benchmark suite targeting standard cell implementations show that the proposed procedure generated smaller transistor circuits than a technology mapping based approach in a comparable runtime.

1. INTRODUCTION

Transistor-level optimization is known as a powerful technique to improve a circuit performance beyond gate-level optimization. The recent significant progress in automated cell-layout generation [1][2] has made transistor-level optimization a feasible and practical solution. The latest researches have demonstrated that such optimization techniques could be successfully applied to real design projects and achieved significant performance improvements [3][4].

Although most of the recent studies have focused mainly on timing or power optimization, transistor-level optimization has a great potential to improve the area. Area optimization is one of the most classical problems in the field of logic synthesis. Conventionally, an area optimization is achieved by reducing the literal count in a multi-level logic network. Early works [5][6] provided exact multi-level logic minimization algorithms, however, the algorithms are very inefficient and apply to very small functions. An improved algorithm of [6] requires a couple of hours to synthesize small circuits with 12 2-input NAND gates[7]. In addition to their computational costs, these algorithms cannot be applied to the minimization of the number of transistors in a straightforward manner.

This paper presents a procedure which synthesizes a transistor circuit consisting of static CMOS compound gates. Figure 1 shows an example of a static CMOS compound gate. To reduce the computational complexity, circuit configurations are explored via algebraic transformations from prime and irredundant two-level circuits. Our approach relies on a graph structure which encodes multiple circuit structural configurations. The proposed procedure implicitly enumerates possible circuit configurations in the graph structure, and efficiently finds a minimum solution among them.

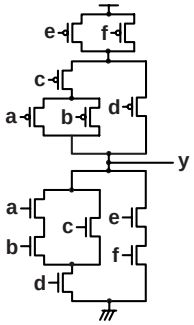


Figure 1. Static CMOS compound gate

2. DEFINITIONS

2.1. Equivalent AND2/INV Network

A *Boolean network* is defined as a directed acyclic graph in which every node has an associated Boolean function. An *AND2/INV network* is a Boolean network where the type of each node is limited to either a 2-input AND gate or an inverter. A *negative unate tree* is a sub-tree of an AND2/INV network with the following properties: (1) the root is an inverter, and (2) every path from the root to leaf has an odd number of inverters. In other words, a negative unate tree can be viewed as an AND-OR tree with an inverter at the root. A negative unate tree can be mapped into a static CMOS compound gate in a unique way by transforming the tree into the series-parallel nested network, and *vice versa*. An example of the mapping from a negative unate tree to a static CMOS compound gate is illustrated in Figure 2. An *equivalent AND2/INV network* is a network of disjoint negative unate trees.

The *cost* of an equivalent AND2/INV network is defined as the number of transistors in the corresponding static CMOS circuit. Figure 3 shows three primitive patterns which form a negative unate tree. Since each pattern has a cost of 2, the cost of a negative unate tree can be calculated as twice the number of patterns in the tree. This

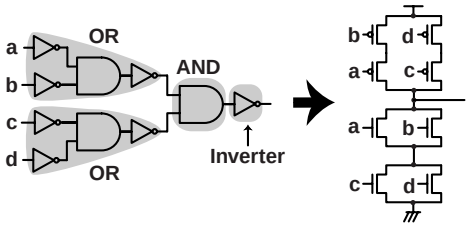
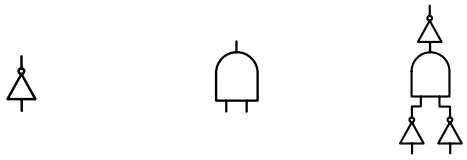


Figure 2. Negative unate tree (Cost=8)



(a) Inverter pattern (Cost=2) (b) AND pattern (Cost=2) (c) OR pattern (Cost=2)

Figure 3. Primitive patterns in equivalent AND2/INV network

is based on the observation that the number of transistors in a static CMOS gate is twice the number of inputs, and adding a pattern in a negative unate tree introduces a new input. Similarly, the cost of an equivalent AND2/INV network can be calculated as twice the number of patterns in the network.

2.2. Mapping Graph

A *mapping graph* proposed by Lehman *et al.* efficiently encodes multiple AND2/INV networks in a single graph structure [8]. A mapping graph is an AND2/INV network with a new type of node, called *choice node.* In a mapping graph, the output of a choice node represents a unique Boolean function. In other words, there cannot be two choice nodes which represent the same Boolean function. All inverters and 2-input AND gates with logically-equivalent outputs are connected to the corresponding choice node as its fanins. Figure 4 shows a mapping graph encoding different implementations of $f = abc$. In the figure, a mapping graph is partitioned into disjoint subgraphs, called *ugates*. The cycles introduced by inverters in a ugate are a mechanism to encode an inverter chain with an arbitrary number of stages. In a mapping graph, an AND2/INV network is decoded by selecting one or more fanins at each choice node.

3. SYNTHESIS PROCEDURE

3.1. Generating a Mapping Graph

Along with the mapping graph structure, Lehman *et al.* also provided the following procedure which encodes all possible algebraic decompositions of a Boolean network in a mapping graph. First, a Boolean network is decomposed into an arbitrary AND2/INV network and then every adjacent AND gates are collapsed into a bigger AND

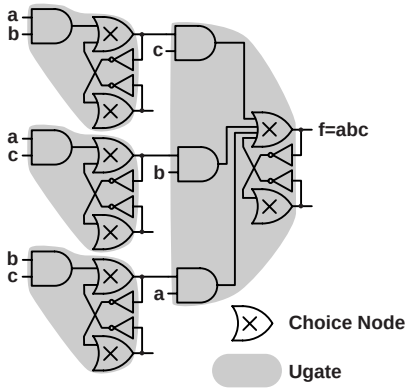


Figure 4. Mapping graph

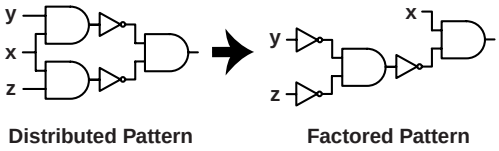


Figure 5. Distributed transformation

gate as much as possible. A mapping graph is constructed from this network by encoding all possible decompositions of each AND gate. Then, the distributed transformation shown in Figure 5 is exhaustively applied to the mapping graph. This procedure is referred to as *standard graph construction procedure* for a convenience of reference. The resulting mapping graph has the following property[8]:

Theorem 3.1 *A mapping graph generated by the standard graph construction procedure contains every AND2/INV decomposition of an arbitrary algebraic decomposition of a Boolean network.*

In the standard graph construction procedure, the set of AND2/INV networks encoded in a mapping graph depends upon the initial Boolean network. In our approach, we use a two-level network as an initial Boolean network. In the initial Boolean network, each output has a combinatorial node representing the *sum of all prime implicants* of the output function. A mapping graph is constructed from the Boolean network by the standard graph construction procedure. Although the sum of all prime implicants may be redundant, the mechanism of choice node resolves any redundancies by introducing cycles. Figure 6 shows an example of a redundant cycle. Finally, any redundant cycles are removed except the cycles in a ugate. This procedure is referred to as *extended graph construction procedure*. The resulting mapping graph has the following property:

Theorem 3.2 *A mapping graph generated by the extended graph construction procedure contains every AND2/INV decomposition of an arbitrary algebraic decomposition of an arbitrary prime-and-irredundant two-level Boolean network.*

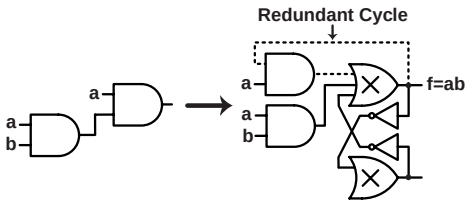


Figure 6. An example of a redundant cycle in a mapping graph

Note that the theorem does not always guarantee that a mapping graph generated by the extended graph construction procedure contains any network generated by Boolean decompositions, or any network generated from a two-level Boolean network which is neither prime nor irredundant.

3.2. Finding a Minimum Circuit

The basic idea for finding a minimum circuit is to exhaustively enumerate all possible equivalent AND2/INV networks encoded in a mapping graph. Choice nodes are visited in a topological order starting from the primary outputs. At each choice node, all possible matches are identified using the patterns shown in Figure 3. A choice node can be duplicated if there are two or more fanouts. If the choice node is a fanin of a primary output or has multiple fanouts after the duplication, only the inverter pattern is allowed to match at the choice node. This guarantees any resulting AND2/INV network to be an equivalent AND2/INV network. From the definition, either the AND or OR pattern must appear only in a tree whose root is an inverter.

Obviously, the above procedure is computationally too expensive. To reduce the computational cost, we introduce the notion of *frontier*. A frontier is a set of choice nodes whose output nets divide a mapping graph into two regions, a region which contains all primary outputs and a region which contains all primary inputs. The two regions are referred to as an *internal region* and an *external region*, respectively. An internal region has the following property:

Lemma 3.1 *Every minimum-cost equivalent AND2/INV network which can be divided by a frontier f contains a minimum-cost equivalent AND2/INV network of the internal region defined by the frontier f.*

This theorem implies that there is no need to record all possible solutions at each frontier, but it is sufficient to record a minimum-cost solution. From this observation, the proposed procedure described below is based on the dynamic-programming method.

Given a mapping graph which encodes multiple AND2/INV networks, a label $L(n)$ is assigned to each choice node n in the mapping graph, according to the topological order starting from the primary outputs. If there is only a directed path from m to n, then $L(m) > L(n)$. If there is also a directed path from n to m, then $L(m) = L(n)$.

```
Given: Mapping graph
Procedure Minimum Circuit
    Assign labels to choice nodes
    Record an initial frontier
    for L=1 to maximum label
        for each frontier F with label L
            Get a choice node C∈F whose label is L
            for each match M at C
                Expand the frontier by including M
                Record the expanded frontier with the minimum solution
            end for
        end for
    end for
    Obtain the minimum solution associated with the final frontier
    Transform the minimum network to a static CMOS circuit
end Procedure
```

Figure 7. Basic procedure for finding a minimum circuit

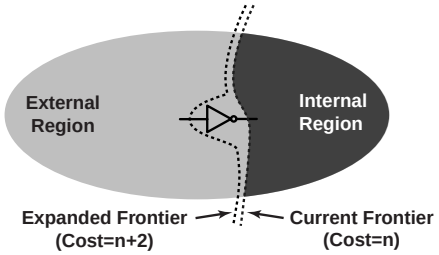


Figure 8. Frontier expansion

If there is no directed path in either direction between m and n, then $L(m) \neq L(n)$. Since a mapping graph does not have any cycle except the cycles in a ugate, no two choice nodes in different ugates can have the same label. A label of a frontier is defined as the smallest label in the set of the choice nodes in the frontier. A frontier f is said to be *interior* of a frontier g if the label of f is smaller than the label of g.

The basic procedure for finding a minimum circuit is shown in Figure 7. The procedure starts with a frontier consisting of the choice nodes whose fanouts are the primary outputs. The minimum-cost solution associated the frontier is an empty circuit. Suppose we are given a frontier with an associated minimum-cost solution of the internal region. First, a choice node with the smallest label is picked up from the set of choice nodes in the frontier. Then, the frontier is expanded by performing the matching procedure at the choice node as described above. Assuming that the cost of the minimum-cost solution at the current frontier is n, then the cost at the expanded frontier is $n + 2$. The expanded frontier is recorded with its associated minimum-cost solution. If the same frontier is already visited, the solution is updated only if the new solution is better. A frontier can be expanded only if there's no other interior frontier. Figure 8 illustrates how a frontier is expanded.

The solution associated with the frontier consisting of only the primary inputs corresponds to a minimum-cost solution. Finally, a static CMOS circuit with the minimum number of transistors is obtained by transforming each negative unate tree in the minimum-cost AND2/INV network into a static CMOS compound gate.

Table 1. Comparison between SIS and the proposed procedure

Circuit	*#inputs*	*#outputs*	*SIS*		*Proposed*		*Reduction [%]*
			#transistors	*CPU time [sec]*	*#transistors*	*CPU time [sec]*	
b1	3	4	26	0.1	26	0.0	0.0
C17	5	2	22	0.1	22	3.3	0.0
cm42a,e,f	4	2	22	0.1	20	0.1	9.1
cm42a,g,h	4	2	24	0.1	18	0.1	25.0
cm42a,i,j	4	2	22	0.1	18	0.1	18.2
decod,f,g	5	2	28	0.1	18	5.9	35.7
decod,h,i	5	2	28	0.1	20	6.0	28.6
decod,j,k	5	2	32	0.1	20	6.0	37.5
majority	5	1	32	0.1	20	1.1	37.5
t	5	2	22	0.1	22	0.0	0.0
x2,m,o	6	2	22	0.1	20	1.4	9.1
z4ml,27	3	1	20	0.1	20	0.1	0.0

4. EXPERIMENTAL RESULTS

The proposed procedure has been implemented on top of SIS [9]. We conducted experiments on a subset of MCNC91 benchmark circuits, which is similar to the one used in [7]. We would like to emphasize that the sizes of the benchmark circuits in Table 1 are reasonably big as standard cells. Besides, in a layout implementation, transistors in a cell may be divided into smaller transistors to fit the cell height. Since standard cells have a fixed height, a cell with large number of transistors results in a very long shape and will cause difficulties in placement.

For comparison, we synthesized static CMOS circuits on the same set of benchmark circuits using SIS technology mapper as follows. The cell library used consists of 23 logic types, ranging from inverter to AOI33, with the following modification. In the library, the area of each cell is substituted with the number of transistors in the cell. By using this trick, a total cell area corresponds to the number of transistors in a circuit. Theoretically, an area optimization with this library is supposed to generate a circuit with the minimum number of transistors. First, we performed an initial multi-level logic minimization using `script.rugged`. Then, a transistor circuit is synthesized by performing an area-optimal tree mapping (`map -m 0.0`).

Table 1 shows the comparison between SIS and the proposed procedure. In the table, the rightmost column shows the reduction rates of transistor counts. As can be seen from the table, the proposed procedure could reduce the number of transistors up to 37.5%, and the runtime is reasonably small.

5. CONCLUSIONS

This paper presented an algebraic approach for synthesizing static CMOS transistor circuits. The presented procedure implicitly enumerates possible circuit configurations in a graph structure which can efficiently encode multiple circuit structural configurations. Then, a dynamic programming based procedure finds a minimum solution among them. Experimental results on a benchmark suite targeting standard cell implementations demonstrated that the proposed procedure generates smaller circuits up to 37.5% than a technology mapping based approach. This fact reconfirms a great potential of transistor-level optimization for area minimization.

It is well known that other static CMOS logic styles, such as pass transistor logic, can reduce a transistor count further on some class of Boolean functions. An extension of the proposed procedure to such logic styles is currently being investigated.

6. REFERENCES

[1] M. Guruswamy *et al.*, "CELLERITY: A Fully Automatic Layout Synthesis System for Standard Cell Libraries," in *Proc. of ACM/IEEE Design Automation Conference*, pp. 327–332, Jun. 1997.

[2] *abraCAD Documentation*, Synopsys, Inc., 2003.

[3] R. Panda *et al.*, "Migration: A new technique to improve synthesized designs through incremental customization," in *Proc. of ACM/IEEE Design Automation Conference*, pp. 388–391, Nov. 1998.

[4] D. Bhattacharya and V. Boppana, "Design Optimization with Automated Flex-Cell Creation," in *Closing the Gap Between ASIC & Custom*, pp. 241–268, Kluwer Academic Publishers, 2002.

[5] E. L. Lawler, "An Approach to Multilevel Boolean Minimization," *Journal of ACM*, pp. 283–295, 1964.

[6] E. Davidson, "An Algorithm for NAND Decomposition under Network Constraints," *IEEE Trans. on Computers*, vol. 18, no. 12, pp. 1098–1109, 1969.

[7] R. Drechsler and W. Gunther, "Exact Circuit Synthesis," *IEEE/ACM Int. Workshop on Logic Synthesis*, 1998.

[8] E. Lehman *et al.*, "Logic Decomposition During Technology Mapping," *IEEE Trans. on Computer-Aided Design*, vol. 16, no. 8, pp. 813–834, Aug. 1997.

[9] R. K. Brayton *et al.*, "MIS: A Multiple-level Logic Optimization System," *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 6, pp. 1062–1081, Nov. 1987.