PAPER *Special Section on VLSI Design and CAD Algorithms*

A Structural Approach for Transistor Circuit Synthesis

Hiroaki YOSHIDA[†a)], ***Nonmember***, **Makoto IKEDA**[†], ***and*** **Kunihiro ASADA**[†], ***Members***

SUMMARY This paper presents a structural approach for synthesizing arbitrary *multi-output multi-stage* static CMOS circuits at the transistor level, targeting the reduction of transistor counts. To make the problem tractable, the solution space is restricted to the circuit structures which can be obtained by performing algebraic transformations on an arbitrary prime-and-irredundant two-level circuit. The proposed algorithm is guaranteed to find the optimal solution within the solution space. The circuit structures are implicitly enumerated via structural transformations on a single graph structure, then a dynamic-programming based algorithm efficiently finds the minimum solution among them. Experimental results on a benchmark suite targeting standard cell implementations demonstrate the feasibility and effectiveness of the proposed approach. We also demonstrated the efficiency of the proposed algorithm by a numerical analysis on randomly-generated problems.

key words: *transistor-level synthesis, static CMOS circuits, algebraic transformations, structural transformations, dynamic programming*

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 enabled transistor-level optimization to be used as a feasible and practical solution. The latest researches have demonstrated that such optimization techniques could be successfully applied to real design projects and have achieved significant performance improvements [3], [4]. In their approaches, transistor-level optimization is performed only at the intra-cell level to make the best use of the existing technologies such as static timing analysis, placement and routing, and cell-layout generation.

In general, transistor-level optimization includes two decision problems: *transistor sizing* and *circuit topology synthesis*. It is well known that transistor sizing is one of the effective techniques for timing optimization. Recent semi-custom design methodologies for high-performance ASICs have employed cell libraries with a rich variation of drive strengths [5], [6] to obtain a similar advantage of continuous transistor sizing. It is notable that their cell libraries has a small set of logic families. This fact implies that circuit topology optimization at the intra-cell level is not as effective as transistor sizing. In contrast, it is also known that the use of complex gates can reduce the design area. Although such complex gates are typically not available in cell libraries, circuit topology synthesis technique can generate such gates by combining several cells into a single cell [4]. Besides, the area reduction of non-critical regions improves the overall area utilization, which promotes a faster convergence of timing optimization. On this background, this paper focuses on area optimization at the transistor level.

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. An early work [7] provided an exact multi-level logic minimization algorithm, however, the algorithm applies only to a factored form of a Boolean function, i.e., a *single-output single-stage* static CMOS circuit. For synthesizing multi-output multi-stage static CMOS circuits, a number of heuristics have been developed [8], [9]. There was also an attempt to synthesize an arbitrary static CMOS circuit by technology mapping with a rich set of library cells [10]. It requires tens of thousands of cells to be prepared in advance, which is too infeasible. Due to their heuristic nature, these methods don't guarantee any optimality of the solution. Apart from static CMOS circuits, there are a large number of literatures on the synthesis of more general transistor circuits such as non-series-parallel circuits [11] and pass-transistor circuits [12], [13].

As mentioned above, transistor-level optimization targeting standard-cell based design flow is performed only at the intra-cell level. Hence, we believe that it is still worth developing a computationally-intensive algorithm even though it is applicable only to circuits as small as standard cells. This paper presents an algorithm which synthesizes arbitrary static CMOS circuits targeting the reduction of transistor counts. To make the problem tractable, the solution space is restricted to the circuit structures which can be obtained by performing algebraic transformations on an arbitrary prime-and-irredundant two-level circuit. The circuit structures are implicitly enumerated via structural transformations on a single graph structure, then a dynamic-programming based algorithm efficiently finds the minimum solution among them. The rest of the paper is organized as follows. In Sect. 2, we first show how static CMOS circuits are represented in our approach. After the problem formulation in Sect. 3, the proposed algorithm is described in Sects. 4 and 5. Section 6 presents experimental results on a benchmark suite targeting standard cell implementations and demonstrates the feasibility and effectiveness of the pro-

Manuscript received March 13, 2006.
Manuscript revised June 14, 2006.
Final manuscript received July 31, 2006.

[†]The authors are with the Faculty of Engineering, and VLSI Design and Education Center (VDEC), the University of Tokyo, Tokyo, 113-8656 Japan.
a) E-mail: hiroaki@silicon.u-tokyo.ac.jp

DOI: 10.1093/ietfec/e89–a.12.3529

posed approach. We also show the results of a numerical analysis on randomly-generated problems to demonstrate the efficiency of the proposed algorithm.

2. Representing a Static CMOS Circuit

A *static CMOS compound gate* is a channel connected component (CCC), which is a set of transistors connected at the sources and drains. It consists of two parts, a part of P-type transistors and a part of N-type transistors, which are structurally complementary. By regarding transistors as switches, it can be viewed as a series-parallel (*or* parallel-series) nested switch network which implements a Boolean function. Note that a static CMOS compound gate always implements a negative unate (i.e. monotonically decreasing) Boolean function. A *static CMOS circuit* is defined as a circuit of static CMOS compound gates. An example of a static CMOS circuit is shown in Fig. 1.

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 in which 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 *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 2 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. Similarly, the cost of an equivalent AND2/INV network can be calculated as twice the number of patterns in the network. Figure 3 shows the equivalent AND2/INV network corresponding to the static CMOS circuit in Fig. 1. The cost of the equivalent AND2/INV network is 14 since there are 7 patterns. As can be seen, this matches the number of the transistors in Fig. 1. Note that the notion of the cost of primitive cells is conceptual and it does not imply that an AND or OR gate can be implemented with 2 transistors.

The following lemmas show the relationship between the number of transistors and the cost.

Lemma 2.1. *For an arbitrary static CMOS compound gate γ with n transistors, there exists a negative unate tree with a cost of n.*

Proof. Let the function represented by γ be f. Since each input of γ is connected to a P-type transistor and an N-type transistor, the number of the inputs of γ is m where $2m = n$. By transforming the pull-up (pull-down) network of γ into a tree, an AND/OR tree with m leaves such that the tree represents the complement of f is obtained. By decomposing AND and OR nodes into 2-input nodes, we can obtain a binary AND/OR tree. The number of the nodes in a binary tree with m leaves is given as $m - 1$. By adding an inverter node at the root, the tree represents f and can be viewed as a negative unate tree by regarding the AND, OR and inverter nodes as primitive patterns. Since the number of the nodes in the tree is m, the cost is $n = 2m$ from the definition. ■

Lemma 2.2. *For an arbitrary static CMOS circuit χ with n transistors, there exists an equivalent AND2/INV network with a cost of n.*

Proof. Let Γ be the set of the static CMOS compound gates contained in χ and $T(\gamma)$ be the number of transistors in a

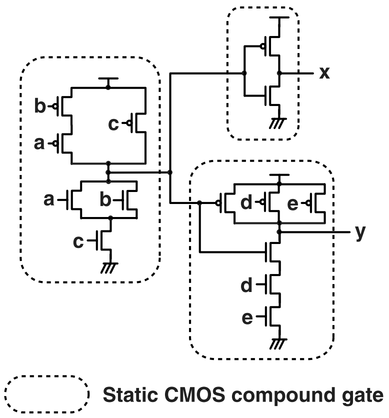


Fig. 1 Static CMOS circuit (14 transistors).

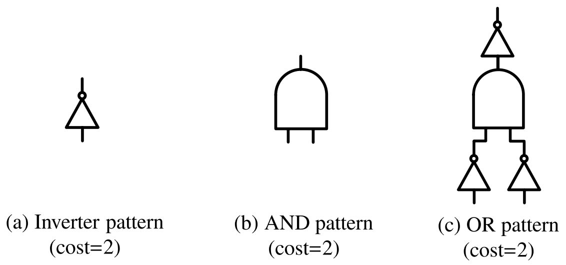


Fig. 2 Primitive patterns in equivalent AND2/INV network.

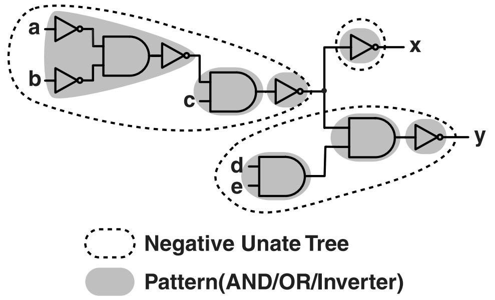


Fig. 3 Equivalent AND2/INV network corresponding to static CMOS circuit in Fig. 1. There are 7 patterns in the network and hence the cost is 14.

static CMOS compound gate $\gamma \in \Gamma$. Then, $\sum_{\gamma\in\Gamma} T(\gamma) = n$. For each $\gamma \in \Gamma$, there exists a negative unate tree with a cost of $T(\gamma)$ from Lemma 2.1. Therefore, the cost of the network is given as $\sum_{\gamma\in\Gamma} T(\gamma) = n$. ■

From these lemmas, we can prove the following theorem.

Theorem 2.1. *If an equivalent AND2/INV network v is minimum in terms of the cost, the corresponding static CMOS circuit χ is minimum in terms of the number of transistors.*

Proof. The proof is by contradiction. Let the cost of v be n. By performing the inverse transformation in the proof of Lemma 2.1, the corresponding static CMOS compound gate with n transistors is obtained from v. Assume that χ is not minimum, i.e., there exists a static CMOS circuit χ' with m transistors such that $m < n$. From Lemma 2.2, there exists an equivalent AND2/INV network v' with a cost of m which represents χ'. However, this contradicts the definition that v is minimum. ■

3. Problem Formulation

The problem addressed in this paper can be formulated as follows:

Problem 3.1. *Given a set of Boolean functions, find a static CMOS circuit which implements the Boolean functions with the minimum number of transistors.*

From Theorem 2.1, we can re-formulate the problem as follows:

Problem 3.1′. *Given a set of Boolean functions, find the minimum-cost equivalent AND2/INV network which implements the Boolean functions.*

The proposed algorithm is divided into two steps. The first step generates a mapping graph which implicitly enumerates possible AND2/INV networks via structural transformations. The second step produces the static CMOS circuit with the minimum number of transistors by finding the minimum-cost equivalent AND2/INV network encoded in the mapping graph.

4. Implicitly Enumerating AND2/INV Networks

4.1 Mapping Graph

A *mapping graph* proposed by Lehman et al. [14] efficiently encodes multiple AND2/INV networks in a single graph structure. 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. Given a mapping graph, an AND2/INV network is decoded by selecting one or more fanins at each choice node. 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.

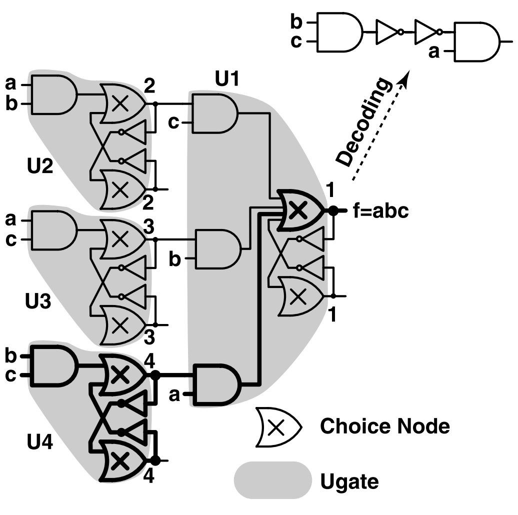


Fig. 4 A mapping graph (lower left diagram) encoding different implementations of $f = abc$. The highlighted portion in the mapping graph generates the AND2/INV network shown in the upper right diagram. The number shown next to each choice node is the label assigned to the choice node.

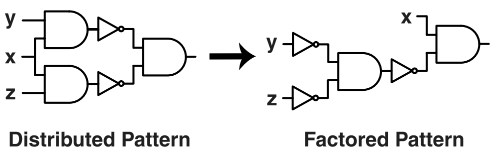


Fig. 5 Distributive transformation.

4.2 Constructing a Mapping Graph

Along with the mapping graph structure, Lehman et al. also provided the following procedure which encodes in a mapping graph all possible algebraic decompositions of a Boolean network. First, a Boolean network η is decomposed into an arbitrary AND2/INV network and then every adjacent AND gates are collapsed into a bigger AND gate as much as possible. A mapping graph is constructed from this network by encoding all possible decompositions of each AND gate. This step is referred to as Λ-construction step. We denote the set of all AND2/INV networks encoded in a mapping graph μ by $\Psi(\mu)$. Then, the resulting mapping graph μ_η^Λ has the following property (Theorem 4.1 in [14]):

Theorem 4.1. *Every AND2/INV decomposition of a Boolean network η is contained in $\Psi(\mu_\eta^\Lambda)$.*

Then, the distributive transformation shown in Fig. 5 is exhaustively applied to μ_η^Λ. This step is referred to as Δ-construction step and the resulting mapping graph μ_η^Δ has the following property (Theorem 4.2 in [14]):

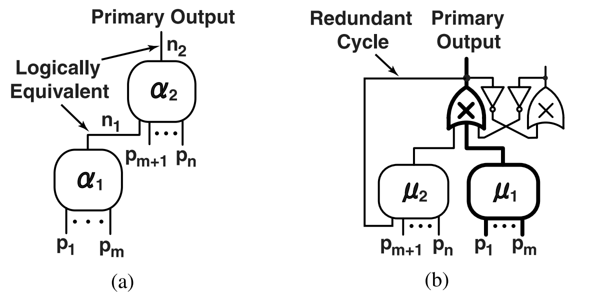


Fig. 6 An illustration of the proof of Theorem 4.2: (a) an AND2/INV decomposition of two-level Boolean network where α_1 and α_2 are the AND2/INV networks representing the logic-OR of the inputs and (b) a mapping graph constructed from (a).

Theorem 4.2. *Every AND2/INV decomposition of an arbitrary algebraic decomposition of a Boolean network η is contained in $\Psi(\mu_\eta^\Delta)$.*

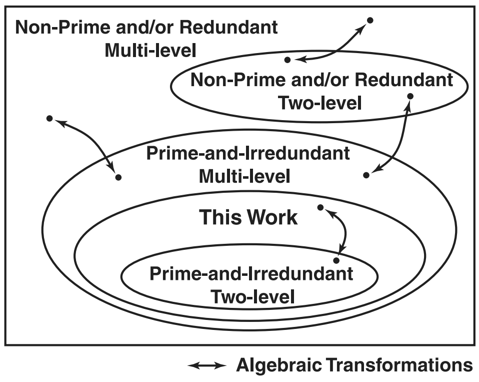
Obviously, the initial Boolean network η determines the set $\Psi(\mu_\eta^\Delta)$ of AND2/INV networks encoded in the final mapping graph μ_η^Δ. In our approach, we use a two-level network as an initial Boolean network. In the initial Boolean network, each output has a combinatorial node representing the *sum of all prime implicants* of the output function. A mapping graph is constructed from the Boolean network by the Λ-construction step. Similarly, another mapping graph is constructed from a Boolean network where each output has a combinatorial node representing the *sum of all prime implicants* of the *complementation* of the output function. The two mapping graphs are merged into a single mapping graph μ_P^Λ.

Lemma 4.1. *For an arbitrary prime-and-irredundant two-level Boolean network η, every AND2/INV decomposition of the Boolean network η is contained in $\Psi(\mu_P^\Lambda)$.*

Proof. Let f be the Boolean function of a primary output or its complement and let $p_1, ..., p_n$ be the all prime implicants of f. Consider an arbitrary prime-and-irredundant two-level expression $F = p_1+\cdots+p_m$ of the Boolean function f where $p_1, ...p_m$ are an irredundant set of the prime implicants. From Theorem 4.1, $\Psi(\mu_P^\Lambda)$ must contain an AND2/INV decomposition of η illustrated in Fig. 6 (a). Since the nodes n_1 and n_2 in the network are logically equivalent, they are the fanins of the same choice node in the resulting mapping graph. Figure 6(b) shows the mapping graph where μ_1 and μ_2 are the partial mapping graphs which are constructed from α_1 and α_2, respectively. Therefore, $\Psi(\mu_1) \subseteq \Psi(\mu_P^\Lambda)$. From Theorem 4.1, $\Psi(\mu_1)$ contains every AND2/INV decomposition of F. ■

Then, the Δ-construction step is performed on μ_P^Λ. The resulting mapping graph μ_P^Δ has the following property:

Theorem 4.3. *For an arbitrary prime-and-irredundant two-level Boolean network η, every AND2/INV decomposition of an arbitrary algebraic decomposition of the Boolean network η is contained in $\Psi(\mu_P^\Delta)$.*



Non-Prime and/or Redundant Multi-level
Non-Prime and/or Redundant Two-level
Prime-and-Irredundant Multi-level
This Work
Prime-and-Irredundant Two-level
Algebraic Transformations

Fig. 7 A Venn diagram which informally illustrates the relationships between the circuit structures encoded in μ_P^Δ and other circuit structures. Note that the sets of non-prime and/or redundant circuits are infinite sets.

Proof. Suppose an arbitrary prime-and-irredundant two-level Boolean network η. From Lemma 4.1, μ_P^Λ satisfies Theorem 4.1 for η. From this fact and Theorem 4.2, every AND2/INV decomposition of an arbitrary algebraic decomposition of η is contained in $\Psi(\mu_P^\Delta)$. ■

As mentioned in the proof, the proposed procedure introduces redundant cycles in a mapping graph. Every redundant cycle except the cycles in a ugate can be removed.

Figure 7 shows an informal illustration of the relationships between the circuit structures encoded in μ_P^Δ and other circuit structures. For the reduction of transistor counts, we are particularly interested in the set of prime-and-irredundant circuits. $\Psi(\mu_P^\Delta)$ contains every circuit structures which can be obtained by performing algebraic transformations on a prime-and-irredundant two-level circuit. However, there can exist prime-and-irredundant multi-level circuits which are not contained in $\Psi(\mu_P^\Delta)$. Those circuits can be obtained only by performing algebraic transformations on a non-prime and/or redundant circuit, or by performing non-algebraic (i.e. Boolean) transformations.

5. Finding the Minimum Circuit

5.1 Naive Approach

Given a mapping graph μ, the objective of this step is to find the minimum-cost equivalent AND2/INV network encoded in μ. One naive approach for finding the minimum solution is to exhaustively enumerate all possible equivalent AND2/INV networks encoded in μ and pick up the minimum-cost network. The 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 Fig. 2. 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. Both of the AND and OR patterns match only at single-fanout choice nodes. This guarantees any resulting AND2/INV network to be an equivalent AND2/INV

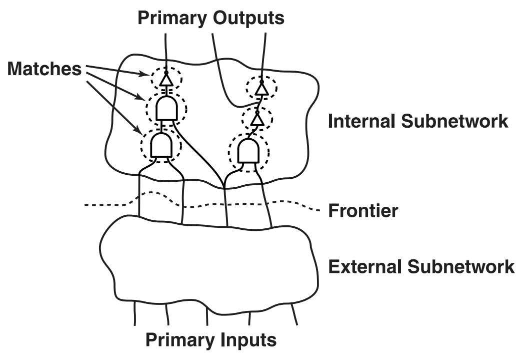


Fig. 8 A partially-covered mapping graph.

network.

Obviously, this naive approach is computationally too expensive. The runtime complexity of this approach is $O(s)$ where s is the number of structures explored during the search. For a mapping graph with n choice nodes where each choice node has k fanins, there are k^n AND2/INV networks in general. Based on the observation that the minimum solution for a subnetwork can be obtained independently of the solution for the remaining portion of the network, the proposed algorithm is based on *dynamic programming* [15]. Here, note that the proposed dynamic programming based algorithm is different from that of the tree covering [10] in the context of the technology mapping.

5.2 Dynamic Programming Based Algorithm

Suppose that a mapping graph μ is partially covered by the matched patterns from the primary outputs as illustrated in Fig. 8. We define a *(partial) cover* γ as a circuit consisting of the matches. A match ϕ is a network of 2-input AND gates and inverters, and corresponds to one of the patterns shown in Fig. 2. A match ϕ is an *input match* if and only if every input of ϕ is also an input net of γ, and we denote the set of input matches by $\Phi(\gamma)$. A frontier λ is a set of the nets in μ which correspond to the input nets of a partial cover. An internal subnetwork μ^I_λ is defined as a subnetwork of μ which consists of the transitive fanouts of λ. Similarly, an external subnetwork μ^E_λ is defined as a subnetwork of μ which consists of the transitive fanins of λ. A frontier λ is used to specify a partial cover and a subnetwork (e.g. γ_{λ_1}, $\mu^I_{\lambda_2}$, $\mu^E_{\lambda_3}$). The following lemma shows that the problem can be solved efficiently using dynamic programming.

Lemma 5.1. *Let γ_λ be the minimum-cost cover of μ^I_λ and let $\gamma_{\lambda-\phi}$ be the cover by removing $\phi \in \Phi(\gamma_\lambda)$ from γ_λ. Then, $\gamma_{\lambda-\phi}$ is the minimum-cost cover of $\mu_{\lambda-\phi}$.*

Proof. The proof is by contradiction. Let $\gamma^*_{\lambda-\phi}$ be the minimum-cost cover of $\mu^I_{\lambda-\phi}$ and γ^*_λ be the cover by adding ϕ to $\gamma^*_{\lambda-\phi}$. Assume that $\gamma_{\lambda-\phi}$ is not the minimum-cost cover of $\mu_{\lambda-\phi}$:

$$C(\gamma_{\lambda-\phi}) > C(\gamma^*_{\lambda-\phi}) \tag{1}$$

where $C(\gamma)$ is the cost of γ. Since every match has a cost of 2 by the definition, $C(\gamma_\lambda) = C(\gamma_{\lambda-\phi}) + 2$ and $C(\gamma^*_\lambda) = C(\gamma^*_{\lambda-\phi}) + 2$. Under the assumption (1), we can derive

$$C(\gamma_\lambda) > C(\gamma^*_\lambda). \tag{2}$$

However, (2) contradicts the definition that γ_λ is the minimum-cost cover of μ^I_λ. ■

```
Input: Mapping graph
Output: Minimum-cost equivalent AND2/INV network
Algorithm FindMinimumCostCover
  Assign labels ($1 \le l \le l_{max}$) to choice nodes
  Clear the sorted queues $Q[1], ..., Q[l_{max}]$
  Create an initial frontier $\lambda_1$ of the primary output nets
  $S[1] \leftarrow \emptyset$
  $Q[1] \leftarrow \lambda_1$
  for $l = 1$ to $l_{max}$
    while $Q[l] \neq \emptyset$
      $\lambda \leftarrow$ the first item in $Q[l]$
      $Q[l] \leftarrow Q[l] - \lambda$
      $n \leftarrow$ a net in $\lambda$ such that $L(n) = l$
      $c \leftarrow$ a choice node whose output is $n$
      for each match $\phi$ at $c$
        $\lambda' \leftarrow$ the expanded frontier by including $\phi$
        if $C(S[L(\lambda)]) + 2 < C(S[L(\lambda')])$
          $S[L(\lambda')] \leftarrow S[L(\lambda)] \cup \phi$
          $Q[L(\lambda')] \leftarrow Q[L(\lambda')] \cup \lambda'$
        end if
      end for
    end while
  end for
  $l_{final} \leftarrow$ the biggest $l$ such that $Q[l] \neq \emptyset$
  return $S[l_{final}]$
end Algorithm
```

Fig. 9 A pseudo-code for the dynamic programming based algorithm. The frontiers in a queue are sorted in ascending order of labels.

The lemma implies that it is sufficient to record only the minimum solution for each internal subnetwork μ^I. Based on this property, we developed the dynamic programming based algorithm. The pseudo-code for the algorithm is shown in Fig. 9.

In a mapping graph, a label $L(c)$ is assigned to each choice node c in the mapping graph, according to the topological order starting from the primary outputs. If there is only a directed path from c to d, then $L(c) > L(d)$. If there is also a directed path from d to c, then $L(c) = L(d)$. If there is no directed path in either direction between c and d, then $L(c) \neq L(d)$. 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 $L(n)$ of a net n is defined as $L(c_n)$ where c_n is the choice node whose output is connected to n. The label $L(\lambda)$ of a frontier λ is defined as the smallest label in the set of the nets in the frontier. A frontier λ is said to be *interior* of a frontier λ' if $L(\lambda) < L(\lambda')$. For instance, in Fig. 4, the number assigned to each choice node in the mapping graph shown is the label of the choice node. In the figure, two choice nodes in a ugate have the same label, and the choice nodes in the ugate U1 have a smaller label since U1 is the fanout of the ugates U2, U3 and U4.

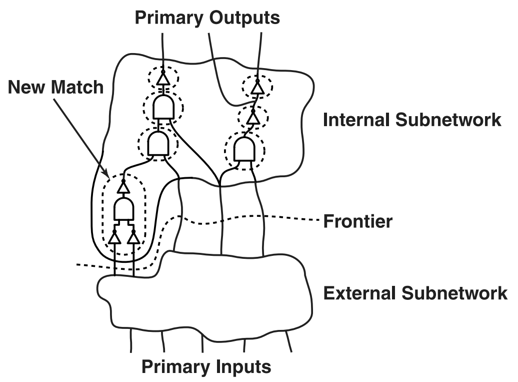


Fig. 10 A frontier expansion on the partially-covered mapping graph shown in Fig. 8.

The algorithm starts with an initial frontier λ_1 consisting of the primary output nets. The minimum-cost solution associated with λ_1 is an empty circuit. Suppose a frontier λ with an associated minimum-cost solution of the internal subnetwork μ^I_λ is given. First, a net n with the smallest label is picked up from λ. Then, the frontier is expanded by performing the matching procedure at c where c is the choice node whose output is connected to n. For each match ϕ, an expanded frontier λ' is generated by including ϕ in μ^I_λ. Assuming that the cost of the minimum-cost solution at the current frontier is S, then the cost at the expanded frontier is $S + 2$. 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 is no other interior frontier. Figure 10 illustrates a frontier expansion on the partially-covered mapping graph shown in Fig. 8.

The solution associated with the frontier consisting only of the primary input nets corresponds to the 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. The following theorem guarantees the optimality of the algorithm.

Theorem 5.1. *The algorithm finds the minimum-cost equivalent AND2/INV network encoded in a mapping graph.*

Proof. Regardless of whether at each frontier all possible covers are recoded or only the minimum cover is recorded, the set of frontiers explored by the procedure remains the same. If all possible covers are recorded at each frontier, the procedure is equivalent to the naive approach and hence finds the minimum solution. From Lemma 5.1, the optimality of the procedure is preserved even if only the minimum cover is recorded at each frontier. ■

The computational complexity of the procedure is approximated as follows. The runtime complexity is given as $O(m)$ where m is the total number of matches performed during the procedure. The space complexity is dominated by the size of the set of queues $\{Q[1], ..., Q[l_{max}]\}$ which contains all frontiers visited during the procedure. Therefore, the space complexity is approximated as $O(f)$ where f is the total number of frontiers visited during the procedure. Due to the nature of the problem, m and f are expected to be exponential to the problem size. Since it is difficult to analyze m and f theoretically, a numerical analysis using randomly-generated problems will be performed in the next section.

6. Experimental Results

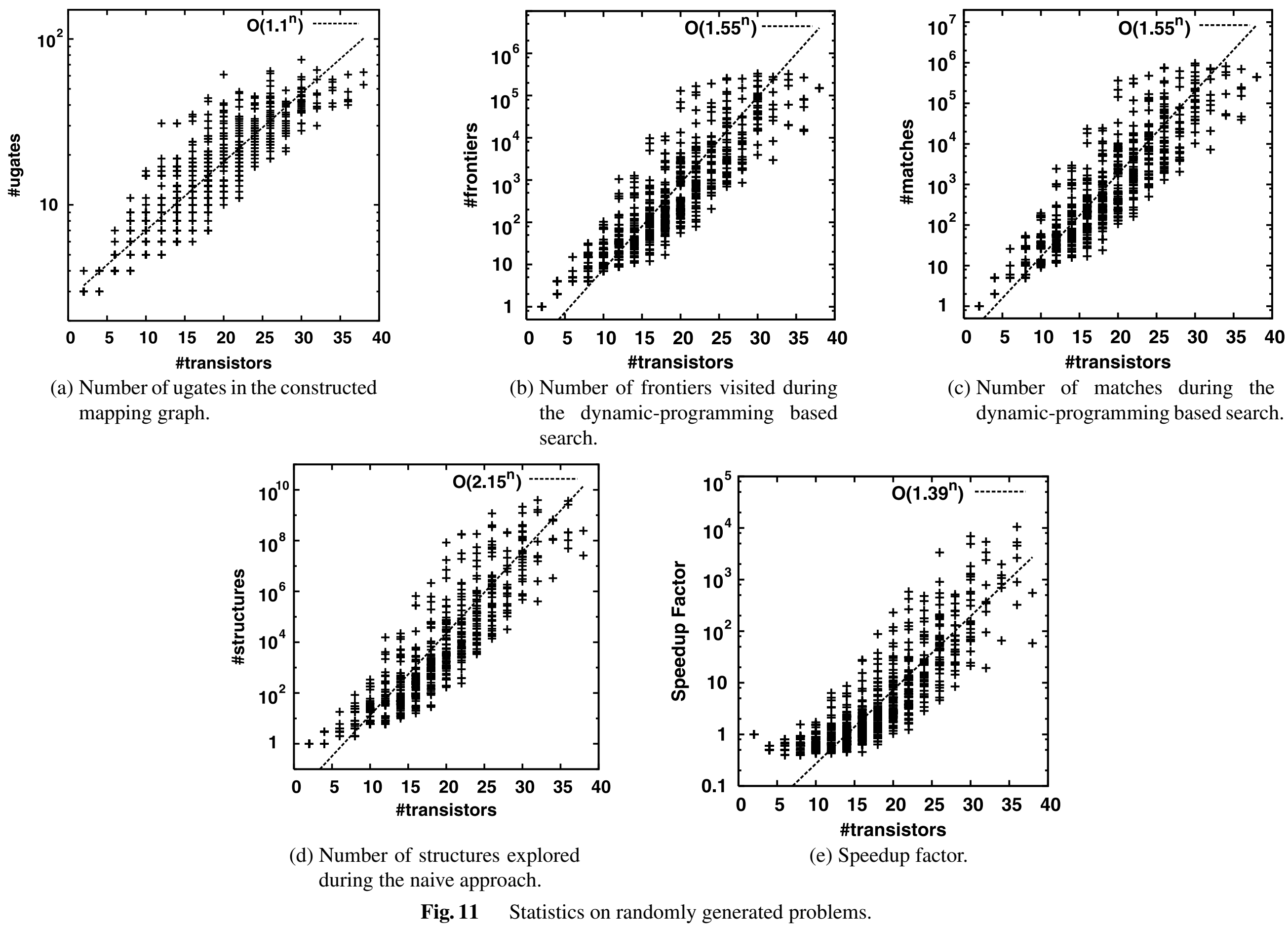
The proposed procedure has been implemented on top of SIS [16]. The platform was a Linux system on AMD Athlon 64 X2 4400 processor with 2 GB main memory. First, we conducted an experiment on a subset of MCNC91 benchmark circuits [17]. Some of them are subcircuits of the original circuits, consisting of a subset of the primary outputs and their transitive fanins. For instance, `cm42a,e,f` is a subcircuit of `cm42a` consisting of the primary outputs `e` and `f` and their transitive fanins. It is noticeable 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. A rich cell library was prepared in a similar way to that used in [10]. We generated all single-stage static CMOS cells such that the maximum number of inputs is limited to 6 and the maximum number of series-connected P-type (N-type) transistors to 4. The number of the logic functions is 461. 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. Ideally, 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.algebraic` and `script.rugged`. Then, a transistor circuit is synthesized by performing an area-optimal tree mapping (`map -m 0.0`). We also implemented one of the algorithms presented most recently [9]. Since the algorithm requires an optimized Boolean network as an input, we used the Boolean networks generated by the same logic minimization described above. Based on our experiments on the same set of problems, our implementation of the algorithm produced almost the same results as those of the SIS-based method and no better results were obtained.

Table 1 shows the comparison between the SIS-based method and the proposed algorithm. In the table, the rightmost column shows the reduction rates of transistor counts. The number of ugates in the constructed mapping graph and the numbers of frontiers and matches during the minimum solution search are also presented. The CPU time includes the time consumed by the mapping graph construction and the minimum solution search. As can be seen from the table,

Table 1 Comparison between SIS and the proposed algorithm.

Circuit	*#inputs*	*#outputs*	*SIS*		*Proposed*					*Reduction [%]*
			#transistors	*CPU time [sec]*	*#transistors*	*#ugates*	*#frontiers*	*#matches*	*CPU time [sec]*	
`b1`	3	4	24	0.1	24	15	292	610	0.0	0.0
`C17`	5	2	24	0.1	22	36	35557	83349	3.3	8.3
`cm42a,e,f`	4	2	16	0.1	16	22	1535	4064	0.1	0.0
`cm42a,g,h`	4	2	22	0.1	18	22	1535	4064	0.1	18.2
`cm42a,i,j`	4	2	22	0.1	18	22	1535	4064	0.1	18.2
`cm42a,k,l`	4	2	20	0.1	18	22	1535	4064	0.1	10.0
`cm42a,m,n`	4	2	22	0.1	18	22	1535	4064	0.1	18.2
`decod,f,g`	5	2	18	0.1	18	46	63806	199104	5.9	0.0
`decod,h,i`	5	2	20	0.1	20	46	63806	199104	6.0	0.0
`decod,j,k`	5	2	20	0.1	20	46	63806	199104	6.0	0.0
`decod,l,m`	5	2	22	0.1	20	46	63806	199104	5.9	9.1
`decod,n,o`	5	2	20	0.1	20	46	63806	199104	5.9	0.0
`decod,p,q`	5	2	22	0.1	20	46	63806	199104	5.9	9.1
`decod,r,s`	5	2	20	0.1	20	46	63806	199104	5.9	0.0
`majority`	5	1	26	0.1	20	63	12939	33772	1.1	23.1
`t`	5	2	24	0.1	22	36	35557	83349	3.2	8.3
`x2,k,l`	3	2	22	0.1	20	12	226	457	0.0	9.1
`x2,m,o`	6	2	22	0.1	20	66	16731	53126	1.4	9.1
`z4ml,27`	3	1	20	0.1	20	33	6407	18809	0.1	0.0



(a) Number of ugates in the constructed mapping graph.

(b) Number of frontiers visited during the dynamic-programming based search.

(c) Number of matches during the dynamic-programming based search.

(d) Number of structures explored during the naive approach.

(e) Speedup factor.

Fig. 11 Statistics on randomly generated problems.

the proposed procedure could reduce the number of transistors up to 23.1%, and the runtime is reasonably small. The proposed procedure failed to solve other bigger problems in the MCNC91 benchmark circuits.

Next, we conducted another experiment on randomly-generated problems in order to demonstrate the efficiency of the proposed algorithm. We generated 1000 problems randomly as follows. First, the numbers of inputs and outputs are randomly determined. Then, for each output, a sum-of-products expression is generated as the output function by randomly determining the number of products and generating the products randomly. In this experiment, the number of inputs is limited up to 6 and the number of outputs is limited up to 3. Next, the proposed algorithm is applied to the problems. If the total number of the matches exceeded 10^6 during the minimum solution search, the procedure is terminated and the problem is discarded. Figures 11(a)–(d) show the statistics of the problems. Since 146 problems were discarded in this experiment, the 854 points are plotted in the graphs. In all graphs, the X-axis correspond to the number of transistors in the resulting static CMOS circuit. Figure 11(a) shows the number of ugates in the constructed mapping graph, which is approximately $O(1.1^n)$ where n is the number of transistors. Figures 11(b) and (c) show f and m, where f and m are the total numbers of frontiers and matches during the dynamic-programming based algorithm, respectively. As explained in the previous section, f and m correspond to the space and runtime complexities of the dynamic-programming based algorithm respectively, and both are approximated by $O(1.55^n)$. To show the efficiency, we also applied the naive approach explained in Sect. 5.1 to the same set of the problems. Figure 11(d) show s, where s is the number of structures explored during the naive approach. The runtime complexity of the naive approach is approximated by s and hence $O(2.15^n)$. Figure 11(e) shows the speedup factor which is calculated as the ratio of s to m. The speedup factor quantifies the runtime efficiency of the dynamic-programming based algorithm against the naive approach, and is approximated by $O(1.39^n)$. Since the current implementation allocates $\sim$1000 bytes for each frontier, the memory is exhausted when solving problems with more than 10^6 frontiers. Based on this observation, the current implementation of the proposed algorithm can solve problems with 30–40 transistors. This also explains the reason why big problems could not be solved in the previous experiment on the MCNC91 benchmark circuits.

7. Conclusions

Transistor-level optimization is known as a powerful technique to improve the circuit area and performance beyond gate-level optimization. In this paper, we presented a structural approach for synthesizing an arbitrary static CMOS circuits targeting the reduction of transistor counts. The circuit structures are implicitly enumerated via structural transformations on a single graph structure, then a dynamic-programming based algorithm efficiently finds the minimum solution among them. We also showed that the solution space contains the circuit structures which can be obtained by performing algebraic transformations on an arbitrary prime-and-irredundant two-level circuit, and the proposed algorithm is guaranteed to find the optimal solution within it. The experimental results on a benchmark suite targeting standard cell implementations demonstrated the feasibility of the proposed procedure. We also demonstrated the efficiency of the proposed algorithm by a numerical analysis on randomly-generated problems. It is also shown that the proposed procedure sometimes generates significantly smaller circuits compared to SIS-based approach. This fact reconfirms a potential of transistor-level optimization for area minimization.

References

[1] M. Guruswamy, R.L. Maziasz, D. Dulitz, S. Raman, V. Chiluvuri, A. Fernandez, and L.G. Jones, "CELLERITY: A fully automatic layout synthesis system for standard cell libraries," Proc. ACM/IEEE Design Automation Conf., pp.327–332, June 1997.

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

[3] R. Panda, A. Dharchoudhury, T. Edwards, J. Norton, and D. Blaauw, "Migration: A new technique to improve synthesized designs through incremental customization," Proc. ACM/IEEE Design Automation Conf., pp.388–391, June 1998.

[4] D. Bhattacharya and V. Boppana, "Design optimization with automated flex-cell creation," in Closing the Gap between ASIC & Custom, pp.14–23, Kluwer Academic, 2002.

[5] G.A. Northrop and P.F. Lu, "A semi-custom design flow in high-performance microprocessor design," Proc. ACM/IEEE Design Automation Conf., pp.426–431, June 2001.

[6] N. Richardson, L.B. Huang, R. Hossain, J. Lewis, T. Zounes, and N. Soni, "The iCORE$^{\text{TM}}$ 520 MHz synthesizable CPU core," in Closing the Gap between ASIC & Custom, pp.225–240, Kluwer Academic, 2002.

[7] E.L. Lawler, "An approach to multilevel Boolean minimization," J. ACM, vol.11, no.3, pp.283–295, July 1964.

[8] S. Gavrilov, A. Glebov, S. Pullela, S. Moore, A. Dharchoudhury, R. Panda, G. Vijayan, and D. Blaauw, "Library-less synthesis for static CMOS combinational logic circuits," Proc. IEEE Int. Conf. on Computer-Aided Design, pp.658–662, Nov. 1997.

[9] C.P.L. Liu and J.A. Abraham, "Transistor level synthesis for static CMOS combinational circuits," Proc. Great Lakes Symposium on VLSI, pp.116–119, March 1999.

[10] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," Proc. IEEE Int. Conf. on Computer-Aided Design, pp.116–119, Nov. 1987.

[11] K. Asada and J. Mavor, "MOSYN: A MOS circuit synthesis program employing 3-way decomposition and reduction based on seven-valued logic," IEE Proc. Comput. Digit. Tech., vol.137, no.6, pp.451–461, Nov. 1990.

[12] K. Yano, Y. Sasaki, and K. Seki, "Top-down pass-transistor logic design," IEEE J. Solid-State Circuits, vol.31, no.6, pp.792–803, June 1996.

[13] P. Buch, A. Narayan, A.R. Newton, and A. Sangiovanni-Vincentelli, "Logic synthesis for large pass transistor circuits," Proc. IEEE Int. Conf. on Computer-Aided Design, pp.663–670, Nov. 1997.

[14] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness, "Logic decomposition during technology mapping," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.16, no.8, pp.813–834, Aug. 1997.

[15] B. Korte and J. Vygen, Combinatorial Optimization: Theory and Algorithms, Springer-Verlag, 2002.

[16] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A multiple-level logic optimization system," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol.6, no.6, pp.1062–1081, Nov. 1987.

[17] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Tech. Rep., Microelectronics Center of North Carolina, Jan. 1991.

Hiroaki Yoshida received the B.S. and M.S. degrees in electronic engineering from the University of Tokyo, Tokyo, Japan, in 2000 and 2002, respectively. From 2002 to 2004, he was a Software Engineer at Zenasis Technologies, Inc., in Campbell, CA, where he was working on the development of a leading-edge logic/physical/transistor-level timing optimization tool. Currently, he is pursuing the Ph.D. degree in electronic engineering at the University of Tokyo, Tokyo. His research interests include logic-level and transistor-level optimization of high-performance circuits. He is a student member of the Institute of Electrical and Electronics Engineers (IEEE) and the Association for Computing Machinery (ACM).

Makoto Ikeda received the B.S., M.S., and Ph.D. in electronics engineering from the University of Tokyo, Tokyo, Japan, in 1991, 1993, and 1996, respectively. He joined EE Department in the University of Tokyo as a faculty member in 1996, and is currently an associate professor of VLSI Design and Education Center (VDEC), the University of Tokyo. His research interests include reliability of VLSI design. He is a member of the Institute of Electrical and Electronics Engineers (IEEE) and the Information Processing Society of Japan (IPSJ).

Kunihiro Asada was born in Fukui, Japan, on June 16th, 1952. He received the B.S., M.S., and Ph.D. in electronic engineering from University of Tokyo in 1975, 1977, and 1980, respectively. In 1980 he joined the Faculty of Engineering, University of Tokyo, and became a lecturer, an associate professor and a professor in 1981, 1985 and 1995, respectively. From 1985 to 1986 he stayed in Edinburgh University as a visiting scholar supported by the British Council. From 1990 to 1992 he served as the first Editor of English version of IEICE (Institute of Electronics, Information and Communication Engineers of Japan) Transactions on Electronics. In 1996 he established VDEC (VLSI Design and Education Center) with his colleagues in University of Tokyo, which is a center to promote education and research of VLSI design in all the universities and colleges in Japan. He also served as the Chair of IEEE/SSCS Japan Chapter in 2001–2002. He is currently in charge of the director of VDEC. His research interest is design and evaluation of integrated systems and component devices. He has published more than 400 technical papers in journals and conference proceedings. He has received best paper awards from IEEJ (Institute of Electrical Engineers of Japan), IEICE and ICMTS1998/IEEE and so on. He is a member of the IEEE and IEEJ.