Resynthesis of Multi-level Circuits Under Tight Constraints Using Symbolic Optimization

Victor N. Kravets
IBM TJ Watson Research Center
Yorktown Heights, NY 10598
kravets@watson.ibm.com

Karem A. Sakallah
Department of Electrical Engineering and Computer Science
University of Michigan, Ann Arbor, MI 48109-2122
karem@umich.edu

ABSTRACT

We apply recently introduced constructive multi-level synthesis in the resynthesis loop targeting convergence of industrial designs. The incremental ability of the resynthesis approach allows more predictable circuit implementations while allowing their aggressive optimization. The approach is based on a very general symbolic decomposition template for logic synthesis that uses information-theoretical properties of a function to infer its decomposition patterns (rather than more conventional measures such as literal counts). Using this template the decomposition is done in a Boolean domain unrestricted by the representation of a function, enabling superior implementation choices driven by additional technological constraints. The symbolic optimization is applied in resynthesis of industrial circuits which have tight timing constraints yielding their much improved timing properties.

1 INTRODUCTION AND MOTIVATION

It is widely acknowledged that current electronic design automation methodologies must be evolved to handle the challenges and opportunities of finer-featured fabrication processes. These methodologies are fundamentally premised on the principle of *separation of concerns* in which a complex design flow is serialized into a sequence of manageable steps that are loosely coupled. In this scenario, decisions made in the early stages of design flow become binding constraints on later stages. Such serialization potentially yields less optimal designs than a methodology that simultaneously considers all design aspects. This is unavoidable, however, due to the practical infeasibility of concurrent optimization of all design parameters, and is deemed acceptable as long as the constraints that are fed forward can be met. The methodology breaks down, however, when these constraints become unsatisfiable; the typical action in such cases is an iteration that revisits earlier design stages to change suspected problematic decisions.

Such iteration has become particularly necessary between the logical and physical synthesis steps due to the inability of layout synthesis to satisfy timing requirements, i.e. achieve timing closure. Several solutions that address these issues were proposed recently. In [2, 13], various re-mapping techniques are applied as a post-processing step to reduce the power and delay of critical circuit sections. In [15] algebraic division is cleverly interleaved with technology mapping to permit a wider, albeit still local, exploration of the implementation space. Approaches based on SPFDs [23], which target the simplification of individual nodes based on the flexibility that arises in their neighborhood, were also studied to improve the topology of optimized circuits.

In this paper we advocate an alternative that lies between the two extremes of serialization and simultaneous optimization of these two steps. Specifically, we propose an aggressive optimization approach that interleaves the technology-independent and library binding stages of logic synthesis to produce implementations with more predictable timing properties that are easier to satisfy by the subsequent layout synthesis step. To relate these two stages more closely, we take a fresh look at functional decomposition and show how it can be used to advantageously link functional and implementation structures in logic synthesis.

Functional decomposition has been studied by many authors. The original concepts were due to Ashenhurst [1], Curtis [9], and Roth and Karp [17]. These early investigations were mostly concerned with the existence of certain types of decomposition rather than with the development of scalable synthesis algorithms. More recently, several authors have re-visited this early work for application in the domain of FPGA synthesis [14, 16, 19, 20, 26]. Practical general-purpose synthesis approaches, based on fast algebraic division algorithms, emerged in the early eighties [5]. The primary motivation for these approaches was efficient decomposition and realization of large random logic functions by a two-stage process: technology-independent restructuring followed by binding to a specified library of primitive gates. This methodology enjoyed a great deal of success and is incorporated in most commercial synthesis tools in use today.

Our contribution is a very general symbolic model of functional decomposition which can be "solved" to determine feasible decompositions. This computational model extends to accommodate the practical constraints of scalable synthesis without encumbered derivations. We use this model to decompose functions directly in terms of a small set of decomposition primitives that correspond to the functional content of cells from a typical library of semiconductor technology. The technology-specific decomposition steps enable optimization choices to be made in the context of accurate timing information while accounting for circuit area.

The predictable effects of such transformations open new improvement opportunities for designs that fail to meet their required timing or area constraints, allowing for aggressive correction of critical paths, or area recovery of a circuit without introducing new violations. The proposed symbolic decomposition model has been developed specifically for this purpose, and we therefore evaluate it in the resynthesis-based optimization of industrial circuits. During the optimization, circuit regions are selected iteratively and their functional representation is decomposed with the imposed constraints and objectives targeting improved circuit properties (either area or timing). We implement functional decomposition following the general notions introduced in [11, 12] but—using the theoretical foundation presented here—are able to significantly extend its optimization capability to handle real-world designs.

0-7803-7607-2/02/$17.00 ©2002 IEEE

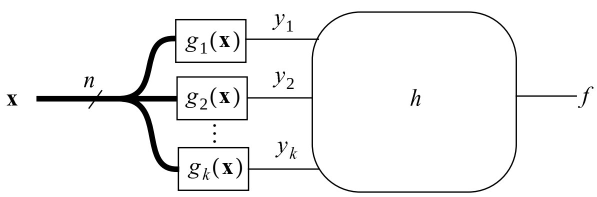

Figure 1: Generic decomposition step

The paper is organized as follows. Section 2 formally describes the symbolic model for functional decomposition. To insure its scalability, the model is modified in Section 3 to account for typical technological constraints that reflect existing semiconductor technologies. The overall synthesis flow that uses this model is described in Section 4. The approach is evaluated experimentally in Section 5 on publicly-available as well as proprietary industrial benchmarks. Section 6 concludes the paper with suggestions on further extensions of this methodology.

2 SYMBOLIC FORMULATION OF DECOMPOSITION

In this section we propose a symbolic model for functional decomposition that allows us to pose and answer several key questions related to scalable synthesis, including the existence of a decomposition, and the existence of universal primitives that allow the decomposition of certain classes of functions.

2.1 Generic Decomposition Template

Given an n-variable Boolean function $f(\mathbf{x})$, and k n-variable Boolean functions $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$, we say that f has an n-to-k decomposition with respect to $g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})$ if and only if there exists a k-variable function h such that

$$f(\mathbf{x}) = h(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x})) \quad (1)$$

A pictorial representation of this decomposition template is shown in Figure 1; h will be referred to as the *composition function*, whereas $g_1, \ldots, g_k$ will be called the *decomposition functions*. These functions introduce intermediate variables $y_1, \ldots, y_k$ into the network that serve as the support of the composition function. The decomposition is *support-reducing* if $k < n$. The k decomposition functions can be viewed as a single multi-output decomposition function $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$, and the intermediate variables can be represented by a k-vector $\mathbf{y} = (y_1, \ldots, y_k)$.

The decomposition template in (1) is sufficiently general to encompass all types of functional decomposition described in the literature, including simple and complex disjunctive and non-disjunctive decompositions [9]. As we show later, support-reducing decompositions in terms of fan-in bounded decomposition functions are particularly attractive from a practical perspective. Before such restrictions are imposed, though, we show in the remainder of this section the relations that must exist between the composition and decomposition functions for equation (1) to hold.

2.2 Computation of Composition Function

To determine if the decomposition in (1) exists, we can solve for h in terms of $\mathbf{g}$ and f. The solution, in general, is not unique and can be expressed as the following function interval [6]:

$$H = [\exists \mathbf{x}(c(\mathbf{x}, \mathbf{y}) \cdot f(\mathbf{x})), \forall \mathbf{x}(\overline{c(\mathbf{x}, \mathbf{y})} + f(\mathbf{x}))] \quad (2)$$

where $c(\mathbf{x}, \mathbf{y}) = (y_1 \equiv g_1(\mathbf{x}))\ldots(y_k \equiv g_k(\mathbf{x}))$ is a function that models the constraints introduced by the decomposition functions and represents a *care set*. Originally introduced by Cerny [8] as an *output characteristic function*, it evaluates to true for the set of input-output assignments that are consistent with the functional behavior of the circuit and to false otherwise. In recent years output characteristic functions have been used to describe the flexibility that arises in design optimization. Viewed as Boolean relations, Brayton and Somenzi [3] described how they can be used to compute the flexibility in optimizing hierarchical designs. Savoj [18] also used the output characteristic function to describe the maximum flexibility in the optimization of Boolean networks.

Note that when $c(\mathbf{x}, \mathbf{y}) = 1$, equation (2) simply reduces to $f(\mathbf{x})$; when $c(\mathbf{x}, \mathbf{y}) = 0$, on the other hand, the equation becomes the interval [0, 1] denoting an arbitrary function. The existential and universal quantification of $\mathbf{x}$ from the lower and upper interval bounds effectively "shrinks" the interval, and corresponds to the removal of these variables from the support of h making it a function of just the intermediate variables $\mathbf{y}$ (thus reflecting the structure shown in Figure 1).

In general, when $k \geq n$ it is always possible to find decomposition functions that will make the interval in (2) non-empty. Such is not the case in the presence of a fan-in bound on decomposition functions for $k < n$; it may not be possible to find a support-reducing decomposition in terms of fan-in bounded decomposition functions. In the examples below we illustrate how choices of the decomposition functions can effect the existence of decomposition.

Example 1 Let $f = x_1 \oplus x_2$, $g_1 = x_1$, $g_2 = \bar{x}_1 + \bar{x}_2$, and $g_3 = x_2$. The input-output characteristic function of these decomposition functions is:

$$c(x_1, x_2, x_3, y_1, y_2) = (y_1 \equiv x_1) \cdot (y_2 \equiv \bar{x}_1 + \bar{x}_2) \cdot (y_3 \equiv x_2)$$

Substituting in (2) we obtain the interval:

$$H = [y_1 y_2 \bar{y}_3 + \bar{y}_1 y_2 y_3, \bar{y}_1 \bar{y}_2 + y_1 y_2 + \bar{y}_2 \bar{y}_3 + y_2 y_3]$$

which represents sixteen possible decomposition solutions. For instance, $h_1 = \bar{y}_1 y_3 + y_1 y_2$ and $h_2 = y_1 y_2 + y_2 y_3$ are two functions from this interval that satisfy the decomposition template in (1) as can be readily verified by direct substitution. ■

Example 2 Let $f = x_1 \oplus x_2 \oplus x_3$, $g_1 = \bar{x}_1 + \bar{x}_2$, and $g_2 = x_3$. The H interval in this case is:

$$H = [y_1 + y_2, \bar{y}_1 y_2]$$

which is empty since its upper bound is less than its lower bound. Thus, f cannot be decomposed in terms of the given decomposition functions. If, however, the first decomposition function is replaced with $g_1 = x_1 \oplus x_2$, then (1) yields the following interval:

$$H = [y_1 \oplus y_2, y_1 \oplus y_2]$$

representing the unique decomposition $h = y_1 \oplus y_2$ with $y_1 = x_1 \oplus x_2$ and $y_2 = x_3$. ■

2.3 Computation of Decomposition Functions

Equation (2) can be used to compute sets of decomposition functions that will guarantee the existence of a decomposition according to the template in (1). To solve for the decomposition functions, we begin by noting that an arbitrary n-variable Boolean function can be expressed in terms of 2^n binary coefficients that denote the function value at each point in its variable space. Thus, we can express $g_j(\mathbf{x})$ as:

$$g_j(\mathbf{x}) = \sum_{i=0}^{2^n-1} \gamma_{ij} \cdot m_i(\mathbf{x}) \tag{3}$$

where $m_i(\mathbf{x})$ is the minterm on $\mathbf{x}$ whose decimal value is i and $\gamma_{ij} \in \{0, 1\}$ is the value of g_j corresponding to this minterm. Using $\Gamma = [\gamma_{ij}]$ to denote the $2^n \times k$ matrix of coefficients representing the k decomposition functions, the care set $c(\mathbf{x}, \mathbf{y})$ can be re-written as:

$$C(\mathbf{x}, \mathbf{y}, \Gamma) = \prod_{j=1}^{k} \left[y_j \equiv \sum_{i=0}^{2^n-1} \gamma_{ij} \cdot m_i(\mathbf{x}) \right] \tag{4}$$

A decomposition exists if and only if the interval (2) is non-empty, i.e., if its lower bound is less than or equal to its upper bound:

$$\begin{aligned} \exists\mathbf{x}(c(\mathbf{x}, \mathbf{y}) \cdot f(\mathbf{x})) &\le \forall\mathbf{x}(\overline{c(\mathbf{x}, \mathbf{y})} + f(\mathbf{x})) \\ &= \overline{\exists\mathbf{x}(c(\mathbf{x}, \mathbf{y}) \cdot f(\mathbf{x}))} + \forall\mathbf{x}(\overline{c(\mathbf{x}, \mathbf{y})} + f(\mathbf{x})) \end{aligned} \tag{5}$$

Substituting (4) in (5) and universally quantifying $\mathbf{y}$ yields:

$$G(\Gamma) = \forall\mathbf{y}(\overline{\exists\mathbf{x}(C(\mathbf{x}, \mathbf{y}, \Gamma) \cdot f(\mathbf{x}))} + \forall\mathbf{x}(\overline{C(\mathbf{x}, \mathbf{y}, \Gamma)} + f(\mathbf{x}))) \tag{6}$$

which is a Boolean function that encodes, via the Γ coefficients, all feasible decomposition functions $\mathbf{g}$.

Example 3 For $f = x_1 \oplus x_2 \oplus x_3$, and assuming invariance under complementation of the decomposition functions, application of (6) yields 99 3-to-2 non-trivial decomposition solutions. This number can be further reduced by discarding solutions whose decomposition functions have redundant signals. Some of the more interesting solutions are:

Solution A	**Solution B**	**Solution C**
$g_1 = x_1 \oplus x_2$	$g_1 = \bar{x}_1 x_2 + x_1 \bar{x}_3$	$g_1 = x_1 \bar{x}_2$
$g_2 = x_3$	$g_2 = \bar{x}_1 x_3 + x_1 x_2$	$g_2 = \bar{x}_1 x_2 \oplus x_3$

■

Equation (6) expresses all the decomposition solutions for a given function *f*. To find the decomposition solutions for an arbitrary n-variable function *f*, we introduce a vector of 2^n encoding coefficients $\Phi \equiv [\varphi_i]$ to express the universe of n-variable functions as:

$$F(\mathbf{x}, \Phi) = \sum_{i=0}^{2^n-1} \varphi_i \cdot m_i(\mathbf{x}) \tag{7}$$

Note that a complete assignment to Φ represents a particular completely specified function *f*; partial assignments to Φ denote families of functions. Re-writing (6) in terms of $F(\mathbf{x}, \Phi)$ we obtain:

$$\begin{aligned} G(\Phi, \Gamma) = \forall\mathbf{y}(&\overline{\exists\mathbf{x}(C(\mathbf{x}, \mathbf{y}, \Gamma) \cdot F(\mathbf{x}, \Phi))} + \\ &\forall\mathbf{x}(\overline{C(\mathbf{x}, \mathbf{y}, \Gamma)} + F(\mathbf{x}, \Phi))) \end{aligned} \tag{8}$$

which is a Boolean function that encodes all feasible decomposition functions $\mathbf{g}$ for any given function *f*. We show next how this encoding function can be used to derive a computational form suitable for large-scale synthesis.

3 PRACTICAL CONSTRAINTS

As stated, equation (8) is not useful for practical decomposition; it requires, in the worst case, the construction of a Boolean characteristic function of $2^n \cdot (k+1)$ encoding variables. The complexity of this function can, however, be dramatically reduced by imposing technological constraints on the decomposition functions $\mathbf{g}(\mathbf{x})$ as well as structural constraints on the decomposition template. Specifically, we require that the support of the decomposition functions be bounded by s, where s is the maximum allowable fan-in of the underlying implementation technology; in current CMOS processes, s is typically four. Additionally, we require the decomposition to be support-reducing, i.e. $k < n$. These two constraints can be viewed as a restriction on the class of functions that can be decomposed, namely those that admit support-reducing decompositions in terms of library primitives.

The fan-in restriction can be reflected in the decomposition template in several ways. We choose to enforce it by partitioning the input variables into two sets, $\mathbf{x}_g$ and $\mathbf{x}_h$, with $|\mathbf{x}_g| = s$. We will refer to $\mathbf{x}_g$ and $\mathbf{x}_h$, respectively, as the decomposition and composition variables. The decomposition template in (1) can now be expressed as:

$$f(\mathbf{x}_g, \mathbf{x}_h) = h(g_1(\mathbf{x}_g), \ldots, g_t(\mathbf{x}_g), \mathbf{x}_h) = \sum_{i=0}^{2^s-1} f_i(\mathbf{x}_h) \cdot m_i(\mathbf{x}_g) \tag{9}$$

where $g_1(\mathbf{x}_g), \ldots, g_t(\mathbf{x}_g)$ are decomposition primitives and $f_i(\mathbf{x}_h)$ are the cofactors [4] of $f(\mathbf{x}_g, \mathbf{x}_h)$ with respect to $\mathbf{x}_g$. This template defines an s-to-t *decomposition pattern* in which information about *f*'s dependence on the composition variables is now completely characterized by its 2^s cofactors with respect to the decomposition variables. The "structure" of the function can, thus, be captured by a set of up to 2^s independent coefficients ζ_i that act as proxies for these potentially complex cofactor functions. Specifically, let $r \le 2^s$ be the number of distinct cofactor functions in (9), and let $Z = [\zeta_0, \zeta_1, \ldots, \zeta_{r-1}]$ denote a vector of r independent coefficients (one per distinct cofactor). The s-to-t decomposition pattern in (9) can now be re-written as:

$$F(\mathbf{x}_g, \mathrm{Z}) = \sum_{i=0}^{r-1} M_i(\mathbf{x}_g) \cdot \zeta_i \tag{10}$$

where each $M_i(\mathbf{x}_g)$ denotes a sum-of-minterms with identical cofactors in (9). The set $\{M_0(\mathbf{x}_g), \ldots, M_{r-1}(\mathbf{x}_g)\}$, thus, is a partition of the $\mathbf{x}_g$ minterm space induced by the equivalence of the cofactor functions. We will refer to $F(\mathbf{x}_g, \mathrm{Z})$ as a *pattern* function; it represents all n-variable functions whose cofactors with respect to $\mathbf{x}_g$ partition the $\mathbf{x}_g$ minterm space as indicated.

It is instructive to compare equations (7) and (10): equation (7) represents the universe of all n-variable functions in terms of 2^n encoding coefficients; equation (10), on the other hand, effectively represents the same universe in terms of at most 2^s encoding coefficients no matter how large n might be! This is a direct consequence of the first constraint mentioned above, namely decomposition in terms of fan-in bounded decomposition functions. It also allows us to replace (8) with the following computationally tractable alternative:

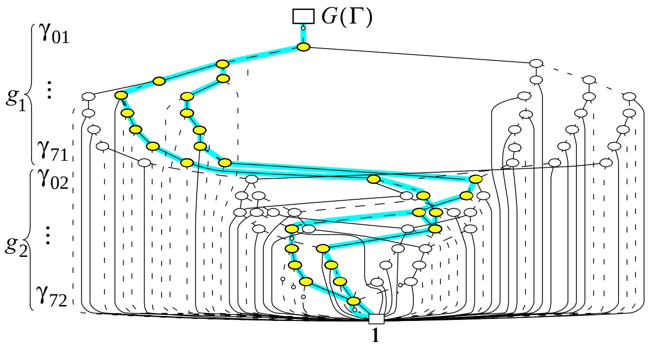

Figure 2: BDD for G(Γ) which encodes all 3-to-2 decomposition functions for $f = a\bar{b}d + a\bar{e} + adc + b\bar{e}d + \bar{b}c\bar{e}$ of Example 4

$$G(\Gamma) = \forall Z \forall \mathbf{y}(\overline{\exists \mathbf{x}_g C(\mathbf{x}_g, \mathbf{y}, \Gamma) \cdot F(\mathbf{x}_g, Z)} + \forall \mathbf{x}_g(\overline{C(\mathbf{x}_g, \mathbf{y}, \Gamma)} + F(\mathbf{x}_g, Z))) \tag{11}$$

This function encodes all feasible s-input decomposition functions **g** for the set of functions f whose structure (pattern) is characterized by the given Z coefficients, and forms the basis of the synthesis algorithms used in this work.

Example 4 Let $f = a\bar{b}d + a\bar{e} + adc + b\bar{e}d + \bar{b}c\bar{e}$. Decomposition with respect to $\{a, b, c\}$ yields:

$$f = (\bar{a}\bar{b}\bar{c}) \cdot (\mathbf{0}) + (a\bar{b}\bar{c} + a\bar{b}c + abc) \cdot (d + \bar{e}) + (\bar{a}b\bar{c} + \bar{a}bc) \cdot (d\bar{e}) + (ab\bar{c} + \bar{a}\bar{b}c) \cdot (\bar{e})$$

Thus, f can be represented by the pattern function:

$$F = \bar{a}\bar{b}\bar{c} \cdot \zeta_0 + (a\bar{b}\bar{c} + a\bar{b}c + abc) \cdot \zeta_1 + (\bar{a}b\bar{c} + \bar{a}bc) \cdot \zeta_2 + (ab\bar{c} + \bar{a}\bar{b}c) \cdot \zeta_3$$

where $\zeta_0, \zeta_1, \zeta_2$ and ζ_3 are independent coefficients. To find all feasible 3-to-2 decompositions of this function, we introduce sixteen encoding coefficients $\gamma_{01}, \ldots, \gamma_{71}$ and $\gamma_{02}, \ldots, \gamma_{72}$ to represent the two 3-input decomposition functions g_1 and g_2, respectively. Substitution in (11) finally yields the desired sixteen-coefficient function $G(\Gamma)$ that encodes all feasible decomposition functions. The BDD for this function is shown in Figure 2. This function has a total of 24 on-set minterms, each corresponding to a pair of possible decomposition functions. Assuming order-invariance of the decomposition functions, and invariance of their complements, the number of solutions reduces to just three:

Solution A	Solution B	Solution C
$g_1 = a\bar{b} + \bar{a}b + bc$	$g_1 = a\bar{b} + \bar{a}b + bc$	$g_1 = a + \bar{b}c$
$g_2 = \bar{a}b + b\bar{c} + \bar{a}c$	$g_2 = a + \bar{b}c$	$g_2 = \bar{a}b + b\bar{c} + \bar{a}c$

These three solutions are highlighted on the BDD of Figure 2. ■

The support reduction requirement implies that $t < s$. This, in turn, translates into a requirement on $f(\mathbf{x})$: for an s-to-t support-reducing pattern the number of f's distinct cofactors must be $\leq 2^t$. Functions that satisfy this support-reduction requirement include the class of symmetric functions, namely functions that remain invariant under certain permutations of their inputs. For any s they can have at most $s + 1$ distinct cofactors implying the existence of support-reducing decomposition of symmetric function for $s \geq 3$.

```
compute_pattern_function(f, x_g, x_h){
    U ← 1;                       // minterm space of x_g
    F ← 0;                       // pattern function
    i ← 0;                       // distinct cofactors count
    while U ≠ 0 do {             // while there are minterms
        select m ∈ U;            // pick a minterm
        u_i ← ∀x_h(f_m ≡ f);     // compute its equivalence class
        F ← F + u_i · ζ_i;       // extend F
        U ← U · ū_i;             // remove equivalence class
        i ← i + 1;               // increment cofactors count
    }
    return F;
}
```

Figure 3: Algorithm to create pattern function $F(\mathbf{x}_g, Z)$ for a given $f(\mathbf{x}_g, \mathbf{x}_h)$

4 CIRCUIT RESYNTHESIS

This section describes a resynthesis-based circuit optimization approach that was implemented in the M31 prototype tool. The tool was developed to evaluate the viability of the above decomposition concepts. We first describe the integration of symbolic decomposition model in a constructive decomposition flow. We then present a resynthesis driver based on such decomposition.

4.1 Constructive decomposition

We employ our symbolic formulation of decomposition in a constructive flow that successively decomposes the functions being synthesized directly in terms of appropriately-chosen decomposition primitives corresponding to cells in typical CMOS libraries. In this flow, the decomposition primitives are introduced incrementally causing the implementation circuit to expand forward from the primary inputs towards the primary outputs. The decomposition process terminates when all nodes become decomposed yielding a "final implementation" of the function. The constructive creation process facilitates management of the structure of the evolving implementation, and makes it possible to account for signal propagation times during the decomposition.

The M31 implementation is based on the repeated application of the following steps:

1. *Support Selection for Decomposition Functions*: Select an unimplemented node, and choose a suitable set of decomposition variables from its current support. Node are selected in topological order according to a complexity estimate of their functions—the node with the largest BDD whose fan-in nodes are all already implemented is selected first. The decomposition variables which enable support reduction and whose nodes have the earliest signal arrival times are then chosen from the node's support. To identify support-reducing decomposition variables the algorithm first looks for groups of symmetric variables. In the absence of symmetries the algorithm relies on a heuristic to identify support-reducing variable subsets. The heuristic requires checking whether a selected subset yields a feasible support-reducing decomposition. This check essentially amounts to building a pattern function. A fast algorithm for its creation is given in Figure 3. The number of ζ_i variables (i.e. distinct cofactors) used in its creation determines the parameter t.

2. *Computation of Decomposition Functions*: Select the decomposition primitives that yield a support-reducing decomposition. Feasible decomposition functions for the variables identified in step 1 are identified according to (11). The set of solutions is then further constrained by the encoded functional content of a given cell library. The M31 implementation tries to achieve decomposition with as few decomposition functions as possible, i.e. it favors small t. Such an objective tends to give fewer connections to the forward logic. Given a set of candidate decomposition functions, M31 ranks them based on how they affect signal propagation, area, and the number of connections they introduce (different priorities can be assigned to these metrics). The decomposition functions which lead to the smallest increase in these metrics are then selected. Since some of the gates corresponding to decomposition functions may already be in the network as a result of earlier iterations, the cost function also accounts for this sharing.
3. *Logic Re-Expression*: Select a composition function from the interval in (2). We currently select a composition function by efficiently remapping the interval's don't cares to the closest care points according to the *re-express* algorithm [12] developed for an earlier implementation of M31. Using this algorithm M31 attempts to re-express the functions of all unimplemented nodes in the network. Such a complete forward re-expression of logic maximizes node sharing in a multi-output circuit. It is worth noting that other criteria are also possible: for example, a maximally symmetric [21] composition function can be selected.

The above three steps are iterated in the algorithm until all nodes become implemented. Note that the first two steps in the algorithm may fail to yield a support-reducing decomposition for some functions. In our experimental analysis, described below, we were forcing M31 to exit the decomposition rendering their nodes as non-decomposable.

4.2 Resynthesis Driver

To handle large circuits with timing constraints M31 uses a simple driver that iteratively (one-at-a-time) re-synthesizes regions of a given depth (e.g., four levels of logic). For each of the regions, the signal arrival times at its inputs are extracted, and its structure is collapsed to two logic levels and re-implemented using the constructive timing-driven synthesis algorithm described above. The resynthesized region is accepted if it improves either the critical slack of the circuit, or its area without causing a reduction in slack. The resynthesis process terminates when all nodes in the network have been processed.

Constraining decomposition with a technology library allows us to better estimate the effect of the resulting circuit transformations. Operating in a technology-specific domain we are also able to drive decomposition with more accurate constraints and objectives. In particular, regions that are in timing-critical sections are resynthesized to minimize delay; other regions are resynthesized to minimize area.

5 EXPERIMENTAL EVALUATION

The M31 program implements the constructive synthesis approached and incorporates a resynthesis driver as described above. It utilizes the CUDD BDD package [24] to implement all symbolic manipulations. We evaluated M31 empirically, comparing its synthesis quality with SIS-1.2 [22]. The evaluation is performed on two sets of circuits: MCNC benchmarks [27] and macros from an industrial design.

Table 1: Characteristic of the M31 and SIS-1.2 circuits as estimated by SIS-1.2 after they were mapped into `mcnc` library

Circuit	Area		Delay		M31/SIS	
	SIS	M31	SIS	M31	Area	Delay
rd53	50	56	17.9	14.3	1.12	0.80
rd73	98	75	35.6	17.4	0.76	0.49
rd84	205	107	26.4	22.6	0.52	0.86
9sym	310	84	35.8	18.9	0.27	0.53
parity	75	75	12.4	15.5	1.00	1.25
my_adder	285	667	57.3	26.0	2.34	0.45
pm4	410	289	48.8	42.3	0.70	0.87
comp	168	240	24.0	25.3	1.43	1.01
z4ml	50	59	25.3	14.5	1.18	0.57
t481	53	56	11.8	11.9	1.06	1.01
pm1	87	72	11.1	10.8	0.83	0.97
c8	175	187	27.5	18.5	1.07	0.67
x4	522	624	38.3	22.2	1.19	0.58
count	216	272	58.6	23.3	1.26	0.40
pcler8	142	235	23.5	16.9	1.65	0.72
lal	146	166	22.5	15.6	1.14	0.70
sct	113	129	57.9	16.2	1.14	0.28
apex7	332	358	33.6	29.2	1.08	0.87
i2	296	295	21.7	17.9	0.99	0.82
Avg.	196.4	212.9	31.05	19.98	1.08	0.65

5.1 MCNC Benchmarks

We first use the MCNC benchmarks to asses M31's ability to decompose functions. To conduct the experiment we created the `mcnc+` library by extending the `mcnc` library with 3-input exclusive-or and 3-input majority cells. These primitives were added to create a "functionally complete" library that enables support-reducing decomposition for functions exhibiting simple symmetry [12]. Using this library the programs were tested on the same set of MCNC combinational benchmarks.

The benchmarks were provided either as two-level or multi-level specifications. M31 constructed symbolic representations from these specifications before commencing the synthesis process. SIS was run with `script.rugged`. The script was used to synthesize multi-level circuits from both the original specification and, when possible, from collapsed two-level forms; the best variant was then reported. To provide a fairer comparison, the netlists produced by both SIS and M31 were technology mapped within SIS to the `mcnc` library.

In the experiments circuits were synthesized with the restriction of $s \le 4$ and $t \le 3$ on the s-to-t support-reducing decomposition patterns. Although not all of the functions can be fully decomposed with these restrictions, our examination shows that for many MCNC benchmarks such decomposition is feasible. Table 1 lists a subset of the benchmarks that were fully decomposed with such restrictions. The table compares post-mapped circuits in terms of area and delay as reported by SIS.

Overall, these results show that, compared to those generated by SIS, the M31 circuits have a much better delay, while their area is only slightly increased. We should point out that the `parity`

Table 2: Results of optimizing circuits of an industrial design using SIS-1.2 and M31

Circuit	Original circuit						SIS-1.2						M31		
							script.rugged			script.delay					
	inp.	outp.	gates	wires	area	slack	wires	area	slack	wires	area	slack	wires	area	slack
macro1	487	255	1448	2652	3251	-10.19	timeout (10hrs)			3779	3937	-8.57	2386	3012	-8.14
macro2	378	437	2353	4485	5154	-10.67	4081	4476	-13.72	5532	5620	-9.33	3861	4574	-7.37
macro3	71	49	1833	3082	3116	-16.01	2522	2633	-8.30	2857	2867	-2.83	2418	2556	-0.93
macro4	52	110	1377	2403	2422	-12.07	out of memory			2396	2422	-7.35	2045	2093	-6.24
macro5	664	669	4362	8085	8191	-11.20	6809	7458	-13.70	8032	8345	-9.36	7083	7324	-7.89
macro6	358	87	1182	2196	2319	-13.30	out of memory			out of memory			1687	1856	-6.90
macro7	46	9	95	186	203	-6.68	188	209	-10.65	261	268	-7.42	179	199	-6.52
macro8	308	196	1558	2806	2893	-5.12	2190	2352	-6.28	2876	3065	-2.72	2380	2523	-3.42
macro9	258	227	1878	3404	3504	-20.34	2699	2972	-29.70	3406	3530	-12.30	2875	3035	-11.5
macro10	263	196	2007	3476	3535	-16.57	2881	3042	-30.17	4201	4259	-13.73	2790	2920	-14.2
macro11	822	816	11013	21079	21856	-14.86	timeout (10hrs)			timeout (10hrs)			18418	19645	-7.48
macro12	564	465	4991	9089	9432	-12.57	timeout (10hrs)			9465	9531	-9.63	7909	8390	-8.44
macro13	79	76	819	813	822	-13.38	732	761	-14.65	777	776	-10.74	679	715	-9.90
macro14	335	227	1863	3290	3313	-15.50	2616	2884	-10.20	3251	3308	-9.58	2749	2840	-9.96
macro15	473	404	1780	3057	3084	-14.00	2538	2692	-12.23	3300	3326	-9.42	2704	2802	-9.02
macro16	138	79	360	588	588	-12.96	452	474	-7.62	607	602	-7.22	486	509	-8.13
macro17	157	103	928	2172	2186	-3.82	out of memory			1892	1918	-4.32	1611	1750	-3.62
macro18	894	898	7789	14332	15063	-9.36	11678	12513	-14.58	15518	15644	-7.76	12247	13200	-7.01
macro19	348	323	2470	4893	5009	-3.50	out of memory			out of memory			4403	4579	0.77
macro20	294	102	659	1276	1279	-0.99	1074	1187	-4.56	1323	1499	-0.56	1112	1134	-0.39
Average ratios with respect to the original circuits:							0.84	0.88	1.42	1.07	1.05	0.72	0.84	0.87	0.62

and `comp` benchmarks, for which M31 has slightly larger delay than SIS, were synthesized from completely different starting points using these tools: while SIS used the original multi-level form as a starting point, M31 synthesized the netlist from a flat form. The run times of M31 were comparable to SIS.

5.2 Industrial Circuits

We show the applicability of the presented decomposition method to larger circuits by presenting re-synthesis results of a set of macros that constitute part of an industrial design. The majority of these circuits have over a thousand gates and large numbers of inputs and outputs. This makes use of their two-level forms as a starting point for decomposition virtually impossible. They also have skewed assertions for signal arrival times and different required times on their outputs. To accommodate these constraints, M31 was configured to use its re-synthesis driver described in Section 4. In the experiments the resynthesized regions were limited to a depth of 4. Signal propagation times were dynamically updated using the SIS load-based delay model.

We used `script.rugged` and `script.delay` to optimize these circuits in SIS (to improve area and delay, respectively [18, 25]). The `full_simplify` command was removed from these scripts as it was giving prohibitively large BDDs, forcing termination of the computation. Netlists optimized with `script.rugged` were technology-mapped using the `map -m 0 -AG` command; the `map -n 1 -AFG` command was used in `script.delay`. We found these commands to work best in meeting area and delay objectives of the scripts. To provide adequate comparison between M31 and SIS both tools used the same `mcnc` library in their optimization.

Table 2 lists the netlists that were used to evaluate the performance of M31. These netlists represent individual macros of a design whose original timing constraints were appropriately scaled according to the nominal delays of the `mcnc` library. Their area and slack were also computed according to the library. Columns 2-7 in the table characterize these circuits in terms of the number of inputs and outputs, gate count, number of wires (i.e. connections), area and slack.

The results in the table suggest the following observations:

- There is a pronounced improvement in area and slack in circuits optimized with M31. The improved quality is observed in all of the optimized macros. In contrast, SIS tends to improve circuits only in one of the parameters per script.
- With the exception of few cases, the slack of M31-optimized circuits is consistently better than the slack of those optimized with either `script.rugged` or `script.delay`. At the same time, the area of M31 circuits is comparable to circuits

Table 3: Further improvements to SIS-1.2 circuits using M31

Circuit	SIS circuit			M31	
	script	area	slack	area	slack
macro8	rugged	2352	-6.28	2314	-2.03
macro9	rugged	2972	-29.70	2889	-13.3
macro14	delay	3308	-9.58	3211	-8.28
macro15	rugged	2692	-12.23	2631	-9.34
macro16	delay	602	-7.22	528	-6.37
macro16	rugged	474	-7.62	474	-6.62
macro18	rugged	12513	-14.58	12323	-8.48

optimized with the area-oriented `script.rugged`.

- On a subset of larger benchmark SIS did not to complete its optimization either due to excessive memory requirements or because of significant run time (more than 10 hours). On the other hand, M31 exhibited more robust behavior: its run time increased gradually with circuit size (exceeding 1 hour only on `macro11`).
- The last experiment illustrates the incremental ability of M31 to improve already-optimized circuits. It was performed by running M31 on the SIS-optimized circuits whose area or delay in Table 2 is superior to the circuits produced by M31. These circuits are listed in Table 3. Column 2 of the table gives the name of the script used by SIS; its resulting area and slack are listed in columns 3 and 4. The results of applying M31 to these circuits are given in the last two columns of the table. The results show that in addition to improving SIS-optimized circuits, M31 is also able to improve its own results from Table 2.

6 CONCLUSIONS AND FUTURE WORK

In this paper we examined a resynthesis approach that is based on a new computational model for symbolic decomposition. The experimental results show that its optimization mechanism enables the creation of circuits with much improved properties. The incremental ability of the approach allows better assessment of circuit implementations while allowing for their aggressive optimization. More accurate physical-domain properties of deep-submicron technologies can be incorporated within the introduced symbolic formulation to further exploit its decomposition patterns.

Our experimental evaluation also suggests further study of better optimization drivers which can avoid “local optima.” An improved resynthesis engine would utilize information-theoretical circuit properties during region selection, in addition to its structure. Application of the introduced symbolic decomposition may also significantly benefit from utilizing decomposition flexibility arising from the region’s surrounding environment.

ACKNOWLEDGMENTS

This work was partially funded by the Semiconductor Research Corporation under contract number 99-TJ-690. Authors would also like to thank David Geiger and Nate Hieter for help with the experimental part of the work.

REFERENCES

[1] R. L. Ashenhurst. The decomposition of switching functions. *Ann. Computation Lab.*, Harvard University, vol. 29, pages 74-116, 1959.

[2] L. Benini, P. Vuillod, and G. De Micheli. Iterative re-mapping for logic circuits. *IEEE TCAD IC*, CAD-17(10):948–964, October 1998.

[3] R. K. Brayton and F. Somenzi. Boolean relations and the incomplete specification of logic networks. In *VLSI’89*, August 1989.

[4] R. K. Brayton, J. D. Cohen *et al.* Fast Recursive boolean function manipulation. In *Proc. IEEE Int. Symp. Circ. and Syst.*, pages 58-62, May 1982

[5] R. K. Brayton and C. McMullen. The decomposition and factorization of Boolean expressions. In *Proc. IEEE Int. Symp. Circ. and Syst.*, pages 29-54, May 1982.

[6] F. M. Brown. *Boolean Reasoning*. Kluwer Academic Publishers, Boston, 1990.

[7] R.E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE TC*, C-35(6):677–691, August 1986.

[8] E. Cerny. An approach to unified methodology of combinational switching circuits. *IEEE TC*, 27(8), 1977.

[9] H. A. Curtis. *New Approach to the Design of Switching Circuits.* Van Nostrand, Princeton, NJ, *1962.*

[10] J. Darringer, W. Joyner, L. Berman and L. Trevillyan. LSS: Logic synthesis through local transformations. *IBM J. Res. and Develop.*, 28(5):537–545, September 1984.

[11] V. N. Kravets and K. A. Sakallah. M32: A Constructive Multilevel Logic Synthesis System. In *Proc. 35th DAC*, pages 336-341, June 1998.

[12] V. N. Kravets and K. A. Sakallah. Constructive library-aware synthesis using symmetries. In *Proc. DATE Conference*, pages 208-216, March 2000.

[13] W. Kunz and D. Stoffel. *Reasoning in Boolean Networks*. Kluwer Academic Publishers, 1997.

[14] Y. T. Lai, M. Pedram, and Sarma B. K. Vrudhula. BDD based decomposition of logic functions with application to FPGA synthesis. In *Proc. 30th DAC*, pages 642–647, June 1993.

[15] E. Lehman, Y. Watanabe, J. Grodstein, and H. Harkness. Logic decomposition during technology mapping. In *Proc. ICCAD*, pages 264–271, November 1995.

[16] R. Murgai, R. K. Brayton, and A. Sangiovanni-Vincentelli. Optimum functional decompositions using encoding. In *Proc. 31st DAC*, pages 408–414, June 1994.

[17] J. P. Roth and R. Karp. Minimization over boolean graphs. *IBM J. Res. and Develop.*, 6(2):227–238, April 1962.

[18] H. Savoj. *Don’t Cares in Multi-Level Network Optimization*. Ph.D. thesis, University of California, Berkeley, 1992

[19] H. Sawada, T. Suyama, and A. Nagoya. Logic synthesis for look-up table based FPGAs using functional decomposition and support minimization. In *Proc. ICCAD*, pages 353–358, November 1995.

[20] C. Scholl, Multi-output functional decomposition with exploration of don’t cares. In *Proc. DATE Conference*, pp. 755-759, February 1998.

[21] C. Scholl, D. Moller, P. Molitor, and R. Drechsler. BDD minimization using symmetries. *IEEE TCAD IC*, 18(2):81–100, February 1999.

[22] E. M. Sentovich. SIS: A system for sequential circuit synthesis. Technical Report UCB/ERL M92/41, UC Berkeley, May 1992.

[23] S. Sinha and R. K. Brayton. Implementation and use of SPFDs in optimization of Boolean networks. In *Proc. ICCAD*, pages 103-110, November 1998.

[24] F. Somenzi. *CUDD: CU Decision Diagram Package*. University of Colorado, Boulder, 2.1.2 edition, April 1997.

[25] H. J. Touati, H. Savoj, and R. K. Brayton. Delay optimization of combinational logic circuits and partial collapsing. In *Proc. 28th DAC*, pages 188–191, June 1991.

[26] B. Wurth, K. Eckl, and K. Antreich. Functional multiple-output decomposition: theory and an implicit algorithm. In *Proc. 32nd DAC*, pages 54–59, June 1995.

[27] S. Yang. *Logic synthesis and optimization benchmarks user guide – ver. 3.0*. MCNC, Res. Triangle Park, NC, Jan. 1991.