Decomposition and Technology Mapping of Speed-Independent Circuits Using Boolean Relations*

Jordi Cortadella, Univ. Politècnica de Catalunya, Barcelona, Spain
Michael Kishinevsky, Alex Kondratyev, The University of Aizu, Japan
Luciano Lavagno, Politecnico di Torino, Italy
Enric Pastor, Univ. Politècnica de Catalunya, Barcelona, Spain
Alex Yakovlev, University of Newcastle upon Tyne, United Kingdom

Abstract

This paper presents a new technique for decomposition and technology mapping of speed-independent circuits. An initial circuit implementation is obtained in the form of a netlist of complex gates, *which may not be available in the design library. The proposed method iteratively performs* Boolean decomposition *of each such gate F into a two-input combinational or sequential gate G available in the library and two gates H_1 and H_2 simpler than F, while preserving the original behavior and speed-independence of the circuit. To extract functions for H_1 and H_2 the method uses* Boolean relations, *as opposed to the less powerful algebraic factorization approach used in previous methods. After logic decomposition, the overall library matching and optimization is carried out. Logic resynthesis, performed after speed-independent signal insertion for H_1 and H_2, allows for sharing of decomposed logic. Overall, this method is more general than the existing techniques based on restricted decomposition architectures, and thereby leads to better results in technology mapping.*

1 Introduction

Speed-independent circuits are *hazard-free under the unbounded gate delay model.* [4, 9, 6] provide general conditions for logic implementability of specifications into *complex gates*. The latter are allowed to have an arbitrary fanin.

To achieve greater practicality more recent work has been focused on the development of logic decomposition techniques. It falls into two categories. One of them attempts to achieve logic decomposition through the use of *standard architectures*. The other group comprises work targeting the decomposition of complex gates directly, by finding a behavior-preserving interconnection of simpler gates. In both cases, the major functional issue, in addition to logic simplification, is that the decomposed logic must not violate the original *speed-independent specification*.

Two examples of the first category are [1, 8]. The basic circuit architecture includes C elements (acting as latches) and combinational logic. This logic is assumed to consist of *AND* gates with potentially unbounded fain and unlimited input inversions and bounded fanin *OR* gates. *Monotonic Cover* (MC) requirements ensure implementability of a specification in this "standard-C" architecture and have an intuitive objective of making the first level (AND) gates work in a *one-hot* fashion with acknowledgment through one of the C-elements. Following this approach, methods for speed-independent decomposition into *implementable* libraries have been developed. E.g., the method of [13] decomposes (if possible) existing gates (e.g., a 3-input *AND* into two 2-input *AND*s), without any further search of the implementation space, and the method of [7] extends the decomposition to more complex (algebraic) divisors, but does not tackle the limitation of the initial MC architecture.

*This work has been funded by ESPRIT ACiD-WG Nr. 214949, CICYT TIC 95-0419, EPSRC grants GR/L24038 and GR/K70175, and MURST (project "VLSI Architectures").

The best representative of the second category appears to be the work of S. Burns [3]. It provides general conditions for speed-independent decomposition of complex (sequential) elements into two sequential elements (or a sequential and a combinational element). Notably, these conditions are analyzed using the original (unexpanded) behavioral model, thus improving the efficiency of the method. This work is, in our opinion, a big step in the right direction, but addresses mainly correctness issues. It does not describe how to use the efficient correctness checks in an optimization loop, and does not allow the sharing of a decomposed gate by different signal networks.

In [14, 12] methods for technology mapping of fundamental mode and speed-independent circuits using complex gates were presented. These methods however only identify when a set of simple logic gates can be implemented as a complex gate, but cannot perform a speed-independent decomposition of a signal function in case it does not fit into a single gate. A BDD-based implementation of [12] is used after decomposition as a post-optimization step in this work.

In our present work we are considering a more general framework which allows use of *arbitrary gates and latches* available in the library to decompose a complex gate function, as shown in Figure 1. In that respect, we are effectively making progress towards the more flexible second approach. The basic idea of this new method is as follows.

An initial complex gate is characterized by its function F. The result of decomposition is a library component designated by G and a set of (possibly still complex) gates labeled $H_1, H_2, \ldots H_n$. The latter are decomposed recursively until all elements are found in the library and optimized to achieve the lowest possible cost. We thus by and large put no restrictions on the implementation architecture in this work. However, as will be seen further, for the sake of practical efficiency, our implemented procedure deals only with the 2-input gates and/or latches to act as G-elements in the decomposition. The second important change of this work compared to [7] is that the new method is based on a full scale Boolean decomposition rather than just on algebraic factorization. This allows us to widen the scope of implementable solutions and improve on area cost (future work will tackle performance-oriented decomposition).

Our second goal in generalizing the C-element based decomposition has been to allow the designer to use more *conventional types of latches*, e.g. D-latches and SR-latches, instead of C-elements that may not exist in conventional standard-cell libraries. Furthermore, as our experimental results show (see Section 6), in many cases the use of standard latches instead of C-elements helps improving the circuit implementations considerably.

The power of this new method can be appreciated by looking at the example `hazard.g`.The original STG specification and its state graph are shown in Figure 2,a and b. The initial implementation using the "standard C-architecture" and its decomposition using two input gates by the method described in [7] are shown in Figure 2,c and d. Our new method produces a much cheeper

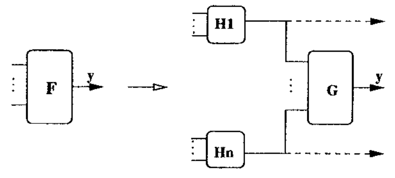


Figure 1: Framework for speed-independent decomposition

solution with just two D-latches, shown in Figure 2,e. Despite the apparent triviality (for an experienced human designer!) of this solution, none of the previously existing automated tools has been able to obtain it. Also note that the D-latches are used in a *speed-independent* fashion, and are thus free from meta-stability and hazard problems.

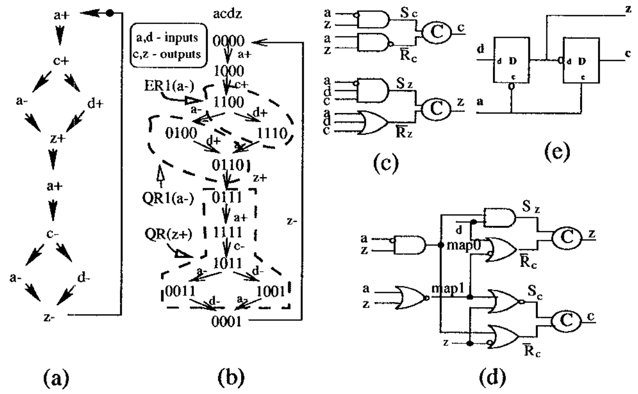


Figure 2: An example of Signal Transition Graph (a), State Graph (b) and their implementation (c)(d)(e) (benchmark `hazard.g`)

The paper is organized as follows. Section 2 introduces the main theoretical concepts. Section 3 presents an overview of the method. Section 4 describes the Boolean relation-based decomposition technique in more detail. Section 5 briefly describes its algorithmic implementation. Experimental results are presented in Section 6.

2 Background

2.1 State Graphs and Logic Implementability

A *State Graph* (SG) is a labeled directed graph whose nodes are called *states*. Each arc of an SG is labeled with an *event*, that is a rising ($a+$) or falling ($a-$) transition of a signal a in the specified circuit. We also allow notation $a*$ if we are not specific about the direction of the signal transition. Each state is labeled with a vector of signal values. An SG is *consistent* if its state labeling $v : S \rightarrow \{0,1\}^n$ is such that: in every transition sequence from the initial state, rising and falling transitions alternate for each signal. Figure 2,b shows the SG for the Signal Transition Graph in Figure 2,a, which is consistent. We write $s \xrightarrow{a}$ ($s \xrightarrow{a} s'$) if there is an arc from state s (to state s') labeled with a.

The set of all signals whose transitions label SG arcs are partitioned into a (possibly empty) set of inputs, which come from the environment, and a set of outputs or state signals that must be implemented. In addition to consistency, the following two properties of an SG are needed for their implementability in a speed-independent logic circuit.

The first property is *speed-independence*. It consists of three parts: determinism, commutativity and output-persistence. An SG is called *deterministic* if for each state s and each label a there can be at most one state s' such that $s \xrightarrow{a} s'$. An SG is called *commutative* if whenever two transitions can be executed from some state in any order, then their execution always leads to the same state, regardless of the order. An event a^* is called persistent in state s if it is enabled at s and remains enabled in any other state reachable from s by firing another event b^*. An SG is called *output-persistent* if its output signal events are persistent in all states and no output signal event can disable input events. Any transformation (e.g., insertion of new signals for decomposition), if performed at the SG level, may affect all three properties.

The second requirement, *Complete State Coding* (CSC), becomes necessary and sufficient for the existence of a logic circuit implementation. A consistent SG satisfies the CSC property if for every pair of states s, s' such that $v(s) = v(s')$, the set of output events enabled in both states is the same. (The SG in Figure 2,b is *output-persistent* and has CSC.) CSC does not however restrict the type of logic function implementing each signal. It requires that each signal is cast into a *single atomic* gate. The complexity of such a gate can however go beyond that provided in a concrete library or technology.

The concepts of excitation regions and quiescent regions are essential for transformation of SGs. A set of states is called an *excitation region* (ER) for event a^* (denoted by $ER(a^*)$) iff $s \in ER(a^*) \Leftrightarrow s \xrightarrow{a^*}$. The *quiescent region* (QR) (denoted by $QR(a^*)$) of a transition a^*, with excitation region $ER(a^*)$, is the set of states in which a is stable and keeps the same value. Examples of ER and QR are shown in Figure 2,b.

2.2 Property-preserving event insertion

Event insertion is an operation on an SG which selects a subset of states, splits each of them into two states and creates, on the basis of these new states, an excitation region for a new event. Figure 3 shows the chosen insertion scheme, analogous to that used by most authors in the area [15]. We say that an inserted signal a *is acknowledged* by a signal b, if b is one of the signals delayed by the insertion of a, (the same terminology will be used for the corresponding transitions). For example, d acknowledges x in Figure 3.

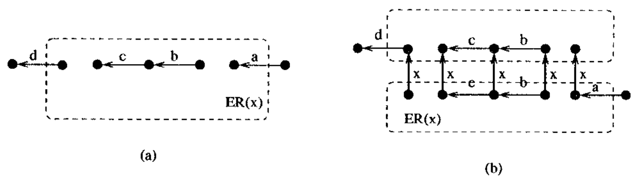


Figure 3: Event insertion: (a) before, (b) after insertion

State signal insertion must preserve the *speed-independence* of the original specification. The events corresponding to an inserted signal x are denoted $x*$, $x+$, $x-$, or, if no confusion occurs, simply by x. Let A be a deterministic, commutative SG and let A' be the SG obtained from A by inserting event x. We say that an insertion state set $ER(x)$ in A is a *speed-independence preserving set (SIP-set)* iff: (1) for each event a in A, if a is persistent in A, then it remains persistent in A', and (2) A' is deterministic and commutative. The formal conditions for the set of states r to be a SIP-set can be given in terms of intersections of r with the so-called state diamonds of SG [5].

A new signal is inserted using an *I-partition* of a set of states S into four blocks: $\{ER(x+), QR(x+), ER(x-), QR(x-)\}$ (similar to [15]). $QR(x-)(QR(x+))$ defines the states in which x will have the stable value 0 (1). $ER(x+)$ ($ER(x-)$) defines the excitation region of x in the new SG A'. To distinguish between the sets of states for the excitation (quiescent) regions of the inserted signal x in an original SG A and the new SG A' we will refer to them as $ER_A(x*)$ and $ER_{A'}(x*)$ ($QR_A(x*)$ and $QR_{A'}(x*)$), respectively. If the insertion of x preserves consistency and persistency, then the only transitions crossing boundaries of the blocks are the following: $QR(x-) \to ER_A(x+) \to QR(x+) \to ER_A(x-) \to QR(x-)$.

Example 2.1 *Figure 4 shows three different cases of the insertion of a new signal x into the SG for the* `hazard.g` *example. The insertion using $ER_A(x+)$ and $ER_A(x-)$ of Figure 4,a does not preserve speed-independence as the SIP set conditions are violated for $ER_A(x+)$ since transition $d+$, enabled in state* 1100, *will be delayed in state* 0100 *after inserting x.*

When signal x is inserted by the excitation regions in Figure 4,b then its positive switching is acknowledged by transitions $a-$, $d+$, while its negative switching by transition $z-$. The corresponding excitation regions satisfy the SIP conditions and the new SG A', obtained after insertion of signal x, is shown in Figure 4,b. Note that the acknowledgment of $x+$ by transitions $a-$, $d+$ results in delaying some input signal *transitions in A' until $x+$ fires. This changes the original I/O interface for SG A, because it requires the environment to look at the new signal before it can change a and d. This is generally undesirable and hence this insertion is rejected.*

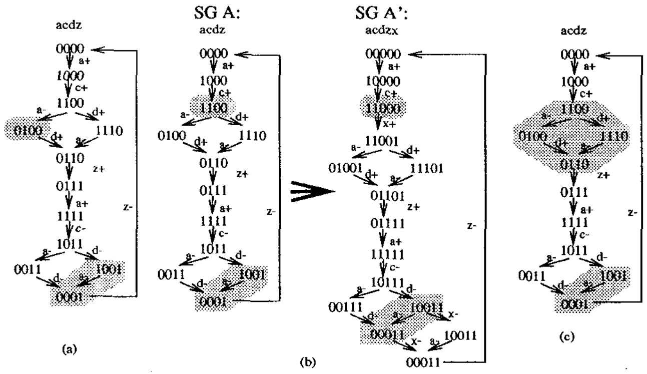


Figure 4: Different cases of signal insertion for benchmark `hazard.g`: violating the SIP-condition (a), changing the I/O interface (b), correct insertion (c)

The excitation regions $ER_A(x+)$ and $ER_A(x-)$ shown in Figure 4,c are SIP sets. They comply with the original I/O interface because positive and negative transitions of signal x are acknowledged only by output signal z. This insertion scheme is valid.

2.3 Basic definitions about Boolean Functions and Relations

An incompletely specified (scalar) Boolean function is a mapping $F : B^n \to \{0, 1, -\}$, where $B = \{0, 1\}$ and '$-$' is a *don't care* value. The subsets of domain B^n in which F holds the 0, 1 and don't care value are respectively called the *OFF-set*, *ON-set* and *DC-set*. F is *completely specified* if its DC-set is empty. We shall further always assume that F is a completely specified Boolean function unless said otherwise specifically.

A variable $x \in X$ is *essential* for function F (or F is dependent on x) if there exist at least two minterms $v1, v2$ different only in the value of x, such that $F(v1) \neq F(v2)$. The set of essential variables for a Boolean function F is called the *true support* of F and is denoted by $sup(F)$. It is clear that for an arbitrary Boolean function its support may not be the same as the true support. E.g., for a support $X = \{a, b, c\}$ and a function $F(X) = \bar{b} + c$ the true support of $F(X)$ is $sup(F) = \{b, c\}$, i.e. only a subset of X.

The *cofactor* of $F(X)$ with respect to x_i $(\overline{x_i})$ is defined as $F_{x_i} = F(x_1, \ldots, x_i = 1, \ldots, x_n)$ $(F_{\overline{x_i}} = F(x_1, \ldots, x_i = 0, \ldots, x_n)$, respectively). The Shannon expansion of a Boolean function $F(X)$ is based on its cofactors: $F(X) = x_i F_{x_i} + \overline{x_i} F_{\overline{x_i}}$. The *Boolean difference*, or *Boolean derivative*, of $F(X)$ with respect to $x_i \in X$ is defined as $\delta F/\delta x = F_{x_i} \oplus F_{\overline{x_i}}$

```
    do
1:      foreach non-input signal $x$ do
            $solutions(x) := \emptyset$;
2:          foreach gate $G \in$ {latches, and2, or2} do
                $solutions(x) := solutions(x) \cup decompositions(x,G)$;
            endfor
3:          $best\_H(x)$ := Best SIP candidate from $solutions(x)$;
        endfor
4:      if foreach $x$, $best\_H(x)$ is implementable
        or foreach $x$, $best\_H(x)$ is empty then exit loop;
5:      Let $H$ be the most complex $best\_H(x)$;
6:      Insert signal $z$ implementing $H$, derive new SG;
    forever
7:  Library matching;
```

Figure 5: Algorithm for logic decomposition and technology mapping.

A function $F(x_1, \ldots, x_i, \ldots, x_n)$ is *unate in variable x_i* if either $F_{\overline{x_i}} \leq F_{x_i}$ or $F_{x_i} \leq F_{\overline{x_i}}$ under ordering $0 \leq - \leq 1$. In the former case $(F_{\overline{x_i}} \leq F_{x_i})$ it is called *positive unate in x_i*, in the latter case *negative unate in x_i*. A function that is not unate in x_i is called *binate* in x_i. A function is (positive/negative) *unate* if it is (positive/negative) *unate* in all support variables. Otherwise it is binate. For example, the function $F = a + b + \bar{c}$ is positive unate in variable a because $F_a = 1 \geq F_{\bar{a}} = b + \bar{c}$.

For an incompletely specified function $F(X)$ with a DC-set, let us define the DC function $F_{DC} : B^n \to B$ such that $ON(F_{DC}) = DC(F)$. We will say that a function $\tilde{F}$ is an *implementation of* F if $F \cdot \overline{F_{DC}} \leq \tilde{F} \leq F + F_{DC}$.

A *Boolean relation*, R, is a generalization of a Boolean function, where a point in the domain B^n can be associated with several points in the codomain, i.e. $R \subseteq B^n \times \{0, 1\}^m$ [2, 11]. Sometimes, we use the "$-$" symbol as a shorthand in denoting elements in the codomain vector, e.g. -0 for 10 and 00. Boolean relations play an important role in multi-level logic synthesis [11], and we shall use them in our decomposition method.

Consider a set of Boolean functions $\mathcal{H} = \{H_1, H_2, \ldots, H_m\}$ with the same domain. Let $R \subseteq B^n \times \{0, 1\}^m$ be a Boolean relation with the same domain as functions from $\mathcal{H}$. We will say that $\mathcal{H}$ is *compatible* with R if for every point v in the domain of R the vector of values $(v, H_1(v), H_2(v), \ldots, H_m(v))$ is an element of R.

3 Overview of the method

Our proposed method for sequential decomposition of speed-independent circuits aimed at technology mapping consists of three main steps:

1. Synthesis via decomposition based on Boolean relations;
2. Signal insertion and generation of a new SG;
3. Library matching

The first two steps are iterated until all functions are decomposed into implementable gates or no further progress can be made. Each time a new signal is inserted (step 2), resynthesis is performed for all output signals (step 1). Finally, step 3 collapses decomposed gates and matches them with library gates. The pseudo-code for the technology mapping algorithm is given in Figure 5.

By using a speed-independent initial SG specification, a complex gate implementation for each SG signal is guaranteed to be speed-independent. Unfortunately this gate may be too large to be implemented in a library. The goal of the proposed method is to break this gate *starting from its output* by using *sequential* (if its function is self-dependent, i.e. it has internal feedback) or *combinational* gates.

Given a vector X of SG signals and given one non-input signal $y \in X$ we try to decompose the function $F(X)$ into (line 2 of algorithm in Figure 5):

- a combinational or sequential gate with function $G(Z, y)$, where Z is a vector of newly introduced signals,
- a vector of combinational[1] functions $\mathcal{H}(X)$ for signals Z,

so that $G(\mathcal{H}(X))$ implements $F(X)$. Moreover, we require the newly introduced signals to be speed-independent (line 3).

The problem of representing the flexibility in the choice of the $\mathcal{H}$ functions has been explored, in the context of combinational logic minimization, by [18] among others. Here we extend its formulation to cover also *sequential* gates (in Sections 4.1 and 4.3). This is essential in order to overcome the limitations of previous methods for speed-independent circuit synthesis that were based on a specific architecture. Now we are able to use a broad range of sequential elements, like set and reset dominant SR latches, transparent D latches, and so on. We believe that overcoming this limitation of previous methods (that could only use C elements and dual-rail SR-latches) is one of the major strengths of this work. Apart from dramatically improving some experimental results, it allows one to use a "generic" standard-cell library (that generally includes SR and D latches, but not C elements).

The algorithm proceeds as follows. We start from an SG and derive a logic function for all its non-input signals (line 1). We then perform an implementability check for each such function as a library gate. The largest non-implementable function is selected for decomposition. In order to limit the search space, we currently try as candidates for G (line 2):

- all the sequential elements in the library (assumed to have two inputs at most, again in order to limit the search space),
- two-input *AND, OR* gates with all possible input inversions.

The set of function pairs (H_1, H_2) compatible with the Boolean relation is then checked for speed-independence (line 3), as described in Section 2.2. If both are not speed-independent, the pair is immediately rejected.

Then, both H_1 and H_2 are checked for approximate (as discussed above) implementability in the library, in increasing order of estimated cost. We have two cases:

1. both are speed-independent and implementable: in this case the decomposition is accepted,
2. otherwise, the most complex implementable H_i is selected, and the other one is merged with G.

The latter is a heuristic technique aimed at keeping the decomposition balanced. Note that at this stage we can also implement H_1 or H_2 as a *sequential gate* if the *sufficient* conditions described in Section 4.3 are met.

The procedure is iterated as long as there is progress or until everything has been decomposed (line 4). Each time a new function H_i is selected to be implemented as a new signal, it is inserted into the SG (line 6) and resynthesis is performed in the next iteration.

The incompleteness of the method is essentially due to the greedy heuristic search that accepts the smallest implementable or non-implementable but speed-independent solution. We believe that an exhaustive enumeration with backtracking would be complete even for non-autonomous circuits, by a relatively straightforward extension of the results in [16].

At the end, we perform a Boolean matching step ([10]) to recover area and delay (line 7). This step can merge together the simple 2-input combinational gates that we have (conservatively) used in the decomposition into a larger library gate. It is guaranteed not to introduce any hazards if the matched gates are atomic.

[1]The restriction that $\mathcal{H}(X)$ be combinational will be partially lifted in Section 4.3.

4 Logic decomposition using Boolean relations

4.1 Specifying permissible decompositions with BRs

In this paper we apply BRs to the following problem.

Given an incompletely specified Boolean function $F(X)$ for signal $y, y \in X$, decompose it into two-levels $y = G(Z, y)$; $Z = \mathcal{H}(X)$ such that $G(\mathcal{H}(X), y)$ implements $F(X)$ and functions G and $\mathcal{H}$ have a simpler implementation than F (any such $\mathcal{H}$ will be called permissible).

The first-level function $\mathcal{H}(X) = \{H_1(X), \ldots, H_n(X)\}$ is a multi-output logic function, specifying the behavior of internal nodes of the decomposition, $Z = \{z_1, \ldots, z_n\}$ [2].

At each step of decomposition a small mappable piece (function G) is cut from the potentially complex and unmappable function F. For a selected G all permissible implementations of function $\mathcal{H}$ are specified with a BR and then via minimization of BRs a few best compatible functions are obtained. All of them are verified for speed-independence by checking SIP-sets. The one which is speed-independent and has the best estimated cost is selected.

Since the true support of function F can include the output variable y, it can specify sequential behavior. In the most general case we perform two-level *sequential* decomposition such that both function G and function $\mathcal{H}$ can be sequential, i.e., contain their own output variables in the true supports. The second level of the decomposition is made sequential by selecting a latch from the library as a candidate gate, G. The technique for deriving a sequential solution for the first level $\mathcal{H}$ is described in Section 4.3.

We next show by example how all permissible implementations of decomposition can be expressed with BRs.

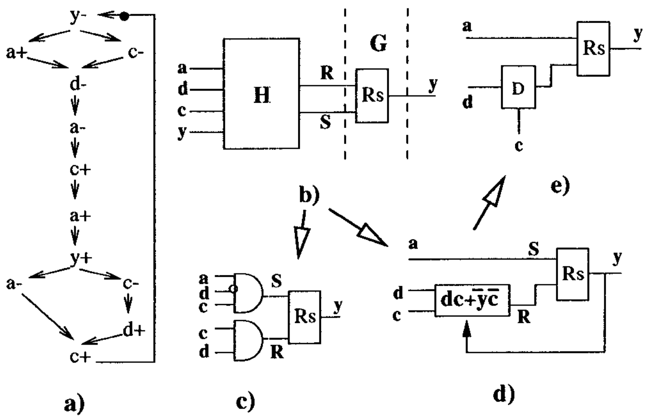
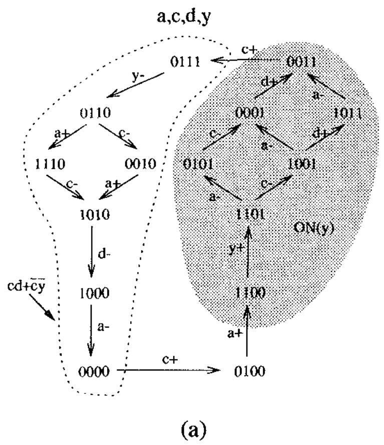


Figure 6: Sequential decomposition for $y = ac\overline{d} + y(\overline{c} + \overline{d})$

Example 4.1 *Consider the* STG *in Figure 6,a, whose* SG *appears in Figure 7,a. Signals a, c and d are inputs and y is an output. A possible implementation of the logic function for y is $F(a, c, d, y) = ac\overline{d} + y(\overline{c} + \overline{d})$. Let us decompose this function using as G a reset-dominant Rs-latch represented by the equation $y = G(R, S, y) = \overline{R}(S + y)$ (see Figure 6,b). At the first step we specify the permissible implementations for the first level functions $R = H_1$ and $S = H_2$ by using the* BR *specified in Figure 7,b. Consider, for example, vector a, c, d, y =* 0000*. It is easy to check that $F(0, 0, 0, 0) = 0$. Hence, for vector* 0000 *the table specifies that $(R, S) = \{11, 10, 00\} = \{1-, -0\}$, i.e. any implementation of R and S must keep for this input vector* either 1 *at*

[2]For simplicity we consider the decomposition problem for a single-output binary function F, although generalization for the multi-output and multi-valued functions is straightforward.



$acdy$	F	RS
0000	0	1–,–0
0001	1	0–
0010	0	1–,–0
0011	1	0–
0100	0	1–,–0
0101	1	0–
0110	0	1–,–0
0111	0	1–
1000	0	1–,–0
1001	1	0–
1010	0	1–,–0
1011	1	0–
1100	1	01
1101	1	0–
1110	0	1–,–0
1111	–	–

(b)

Figure 7: (a) State graph, (b) Decomposition of y by an Rs latch

Region	D-latch C, D	Rs R, S	AND H_1, H_2	OR H_1, H_2
$ER(y+)$	11	01	11	$\{1-, -1\}$
$QR(y+)$	$\{0-, -1\}$	0–	11	$\{1-, -1\}$
$ER(y-)$	10	1–	$\{0-, -0\}$	00
$QR(y-)$	$\{0-, -0\}$	$\{1-, -0\}$	$\{0-, -0\}$	00
unreachable	$--$	$--$	$--$	$--$

Table 1: Boolean relations for different gates

R or 0 *at S. On the other hand, only one solution $R = 0$, $S = 1$ is possible for the input vector* 1100 *which corresponds to setting the output of the Rs-latch to* 1. *The Boolean relation solver will find, among others, the two solutions illustrated in Figure 6,c,d: (1) $R = cd$; $S = ac\overline{d}$ and (2) $R = cd + \overline{y}\,\overline{c}$; $S = a$. Any of these solutions can be chosen depending on the cost function.*

Table 1 specifies compatible values of BRs for different types of gates: a D-latch, a reset-dominant Rs-latch, a two input AND gate and a two input OR gate. All states of an SG are partitioned into four subsets, $ER(y+), QR(y+), ER(y-)$, and $QR(y-)$, with respect to signal y with function $F(X)$ for which decomposition is performed. All states that are not reachable in the SG form a DC-set for the BR. E.g., for each state, s, from $ER(y+)$ only one compatible solution, 11, is allowed for input functions H_1, H_2 of a D-latch. This is because the output of a D-latch in all states, $s \in ER(y+)$ is at 0 and $F(s) = 1$. Under these conditions the combination 11 (clock and data inputs are both high) is the only possible input combination that implies 1 at the output of a D-latch. On the other hand, for each state $s \in QR(y+)$, the output $y = 1$ and $F(s) = 1$, hence it is enough to require either data input to be in 1 or clock input to be in 0 to keep the output of a latch in 1. This is expressed by values $\{0-, -1\}$ in the second line of the table.

4.2 Functional representation of Boolean relations

Given an SG satisfying CSC requirement, each output signal $y \in X$ is associated with a unique *incompletely* specified function $F(X)$, whose DC-set represents the set of unreachable states. $F(X)$ can be represented by three *completely* specified functions, denoted $ON(y)(X)$, $OFF(y)(X)$ and $DC(y)(X)$ representing the ON-, OFF-, and DC-set of $F(X)$, such that they are pairwise disjoint and their union is a tautology.

Let a generic n-input gate be represented by a Boolean equation $q = G(Z, q)$, where $Z = \{z_1, \ldots, z_n\}$ are the inputs of the gate, and q is its output. The gate is sequential if q belongs to the true support of $G(Z, q)$.

We now give the characteristic function of the Boolean relation for the implementation of $F(X)$ with gate G. It represents *all* permissible implementations of $z_1 = H_1(X), \ldots, z_n = H_n(X)$ that allow F to be decomposed by G.

$$BR(y)(X, Z) = ON(y)(X) \cdot G(Z, y) + OFF(y)(X) \cdot \overline{G(Z, y)} + DC(y)(X) \qquad (1)$$

Given characteristic function (1), the corresponding table describing Boolean relation can be derived using cofactors. For each minterm m with support in X, the cofactor $BR(y)_m$ gives the characteristic function of all compatible values for $z_1, \ldots, z_n$ (see example below).

Finding a decomposition of F with gate G is reduced to finding a set of n functions $\mathcal{H}(X) = (H_1(X), \ldots, H_n(X))$ such that

$$BR(y)(X, \mathcal{H}(X)) = 1 \qquad (2)$$

Example 4.2 (Example 4.1 continued.) *The* SG *shown in Figure 7.a corresponds to the* STG *in Figure 6. Let us consider how the implementation of signal y with a reset-dominant Rs latch can be expressed using the characteristic function of* BR. *Recall that the table shown in Figure 7.b represents the function $F(a, c, d, y) = ac\overline{d} + y(\overline{c} + \overline{d})$ and the permissible values for the inputs R and S of the Rs latch. The ON-, OFF-, and DC-sets of function $F(a, c, d, y)$ are defined by:*

$$\begin{aligned} ON(y) &= y(\overline{c} + \overline{d}) + ac\overline{d} \\ OFF(y) &= \overline{y}(\overline{a} + \overline{c} + d) + \overline{a}cd \\ DC(y) &= acdy \end{aligned}$$

The set of permissible implementations for R and S is characterized by the following characteristic function of the BR *specified in the table. It can be obtained using equation 1 by substituting expressions for $ON(y)$, $OFF(y)$, $DC(y)$, and the function of an Rs-latch, $\overline{R}(S + y)$:*

$$BR(y)(a, c, d, y, R, S) = \overline{R}Sac\overline{d} + \overline{R}y(\overline{c} + \overline{d}) + (R + \overline{S})\overline{y}(\overline{a} + \overline{c} + d) + \overline{a}cd(R + \overline{S}\overline{y}) + acdy$$

This function has value 1 *for* all *combinations represented in the table and value* 0 *for all combinations that are not in the table (e.g., for $(a, c, d, y, R, S) = 000001$). For example, the set of compatible values for $acdy = 0110$ is given by the cofactor $BR(y)_{\overline{a}cd\overline{y}} = R + \overline{S}$ which correspond to the terms* 1– *and* –0 *given for the Boolean relation for that minterm.*

Two possible solutions for $BR(y)(a, c, d, y, R, S) = 1$ corresponding to Figure 6,c,d are:

(1) $R = cd$; $S = ac\overline{d}$ (2) $R = cd + \overline{y}\,\overline{c}$; $S = a$

4.3 Two-level sequential decomposition

Accurate estimation of the cost of each solution produced by the Boolean relation minimizer is essential in order to ensure the quality of the final result. The minimizer itself can only handle combinational logic, but often (as shown below) the best solution can be obtained by replacing a combinational gate with a sequential one. This section discusses some heuristic techniques that can be used to identify when such a replacement is possible without altering the asynchronous circuit behavior, and without undergoing the cost of a full-blown sequential optimization step. Let us consider our example again.

Example 4.3 (Example 4.1 continued.) *Let us assume that the library contains three-input AND, OR gates and Rs-, Sr- and D-latches. Implementation (1) of signal y by an Rs-latch with inputs R=cd and S=$ac\overline{d}$ matches the library and requires two AND gates* *(one with two and one with three inputs) and one Rs-latch. The implementation (2) of y by an Rs-latch with inputs $R{=}cd+\overline{y}\,\overline{c}$ and $S{=}a$ would be rejected, as it requires a complex AND-OR gate which is not in the library. However, when input $\overline{y}$ in the function $cd+\overline{y}\,\overline{c}$ is replaced by signal R, the output behavior of R will not change, i.e. function $R{=}cd+\overline{y}\,\overline{c}$ can be safely replaced by $R{=}cd+R\overline{c}$. The latter equation corresponds to the function of a D-latch and gives the valid implementation shown in Figure 6,e.*

Our technique to improve the accuracy of the cost estimation step, by partially considering sequential gates, is as follows:

1. Produce permissible functions $z_1 = H_1(X)$ and $z_2 = H_2(X)$ via the minimization of Boolean relations (z_1 and z_2 are always combinational as $z_1, z_2 \notin X$).
2. Estimate the complexity of H_1 and H_2:
   if H_i matches the library **then** *Complexity* = cost of the gate
   else *Complexity* = literal count
3. Estimate the possible simplification of H_1 and H_2 due to adding signals z_1 and z_2 to their supports, i.e. estimate the complexity of the new pair $\{H_1', H_2'\}$ of permissible functions $z_1 = H_1'(X, z_1, z_2)$, $z_2 = H_2'(X, z_1, z_2)$.
4. Choose the best complexity between H_1 (H_2) and H_1' (H_2').

Let us consider the task of determining H_1' and H_2' as in step 3. Let A be an SG encoded by variables from set V and let $z = H(X, y)$, such that $X \subseteq V, y \in V$, be an equation for the new variable $z \notin V$ which is to be inserted in A. The resulting SG is denoted $A'{=}Ins(A, z{=}H(X,y))$ (or $A'{=}Ins(A, z_1, \ldots, z_k)$ when more than one signal is inserted).

A solution for Step 3 of the above procedure can be obtained by minimizing functions for signals z_1 and z_2 in an SG $A' = Ins(A, z_1, z_2)$. However this is rather inefficient because the creation of SG A' is computationally expensive. Hence instead of looking for an exact estimation of complexity for signals z_1 and z_2 we will rely on a heuristic solution, following the ideas on input resubstitution presented in Example 4.3. For computational efficiency, the formal conditions on input resubstitution should be formulated in terms of an original SG A rather than in terms of the SG A' obtained after the insertion of new signals[3].

Lemma 4.1 *Let Boolean function $H(X,y)$ implement the inserted signal z.Function H can be represented as $H(X,y) = Fy + G\overline{y} + R$, where F, G and R are Boolean functions not depending on y. Let $H'(X,z) = z(F+G) + R$, then SGs $A'{=}Ins(A, z{=}H(X,y))$ and $A''{=}Ins(A, z{=}H'(X,z))$ are isomorphic iff the following conditions are satisfied:*

$$\delta H(X,y)/\delta y \cdot S^* \equiv 0 \quad (1) \qquad F * G * \overline{R} \cap S^+ \equiv 0 \quad (2),$$

where S^* and S^+ are characteristic Boolean functions describing sets of states $ER_A(z+) \cup ER_A(z-)$ and $ER_A(z+)$ in A, respectively.

Informally Lemma 4.1 states that resubstitution of input y by z is permissible if in all states where the value of function $H(X,y)$ depends on y, the inserted signal z has a stable value. The conditions of Lemma 4.1 can be efficiently checked within our BDD-based framework. They require to check two tautologies involving functions defined over the states of the original SG A.

Example 4.4 (Example 4.1 continued.) *Let input R of the RS-latch be implemented as $cd+\overline{y}\,\overline{c}$ (see Figure 6,d). The ON-set of function $H{=}cd{+}\overline{y}\,\overline{c}$ is shown by the dashed line in Figure 7,a. The input border of H (denoted by $IB(H)$) is the set of states by which its ON-set is entered in the original SG A, i.e. $IB(H) = \{0111\}$. By similar consideration we have that $IB(\overline{H}) = \{0100\}$. These input borders satisfy the SIP conditions and hence $IB(H)$ can be taken as $ER_A(R+)$, while $ER_A(R-)$ must be expanded beyond $IB(\overline{H})$ by state 1100 for not to delay the input transition $a+$ ($ER_A(R-) = \{0100, 1100\}$).*

H is negative unate in y and hence Condition 2 of Lemma 4.1 is always satisfied. The set of states where the value of function H essentially depends on signal y is given by the function $\delta H(X,y)/\delta y = a\overline{c}$. Cube $a\overline{c}$ has no intersection with $ER_A(R+) \cup ER_A(R-)$ and Condition 1 of Lemma 4.1 is also satisfied. Therefore literal $\overline{y}$ can be replaced by literal R, thus producing a new permissible function $R{=}cd + R\overline{c}$.

[3]Note that this heuristic estimation covers only the cases when one of the input signals for a combinational permissible function H_i is replaced by the feedback z_i from the output of H_i itself. Other cases can also be investigated, but checking them would be too complex.

5 Implementation aspects

The method for logic decomposition presented in the previous section has been implemented in a synthesis tool for speed-independent circuits. The main purpose of such implementation was to evaluate the potential improvements that could be obtained in the synthesis of speed-independent circuits by using a Boolean-relation-based decomposition approach. Efficiency of the current implementation was considered to be a secondary goal at this stage of the research.

5.1 Solving Boolean relations

In the overall approach, it is required to solve BRs for each output signal and for each gate and latch used for decomposition. Furthermore, for each signal and for each gate, several solutions are desirable in order to increase the chances to find SIP functions.

Previous approaches to solve BRs [2, 17] do not satisfy the needs of our synthesis method, since (1) they minimize the number of terms of a multiple-output function and (2) they deliver (without significant modifications to the algorithms and their implementation) only one solution for each BR. In our case we need to obtain *several compatible solutions* with the primary goal of *minimizing the complexity of each function individually*. Term sharing is not significant because two-level decomposition of a function is not speed-independent in general, and hence each minimized function must be treated as an *atomic* object. Sharing can be exploited, on the other hand, when re-synthesizing the circuit after insertion of each new signal. For this reason we devised a heuristic approach to solve BRs. We next briefly sketch it.

Given a BR $BR(y)(X, Z)$, each function H_i for z_i is individually minimized by assuming that all other functions H_j $(i \neq j)$ will be defined in such a way that $\mathcal{H}(X)$ will be a compatible solution for BR. In general, an incompatible solution may be generated when combining all H_i's. Taking the example of Figure 7, an individual minimization of R and S could generate the solution $R = cd$ and $S = 1$.

Next, a minterm with *incompatible* values is selected, e.g. $\overline{a}\overline{c}\overline{d}\overline{y}$ for which $RS = 01$ but only the compatible values $1-$ or -0 are acceptable. New BRs are derived by freezing different compatible values for the selected minterm. In this case, two new BRs will be produced with the values $1-$ and -0, respectively for the minterm $\overline{a}\overline{c}\overline{d}\overline{y}$. Next, each BR is again minimized individually for each output function and new minterms are frozen until a compatible solution is obtained.

This approach generates a tree of BRs to be solved. This provides a way of obtaining several compatible solutions for the same BR. However, the exploration may become prohibitively expensive if the search tree is not pruned. In our implementation, a branch-and-bound-like pruning strategy has been incorporated for such purpose. Still, the time required by the BR solver dominates the computational cost of the overall method in our current implementation. Ongoing research on solving BRs for our framework is being carried out. We believe that the fact that we pursue to minimize functions individually, i.e. without caring about term sharing among different output functions, and that we only deal with 2-output decompositions, may be crucial to derive algorithms much more efficient than the existing approaches.

5.2 Selection of the best decomposition

Once a set of compatible solutions has been generated for each output signal, the best candidate is selected according to the following criteria (in priority order):

1. At least one of the decomposed functions must be speed-independent.
2. The acknowledgment of the decomposed functions must not increase the complexity of the implementation of other signals (see section 5.3).
3. Solutions in which all decomposable functions are implementable in the library are preferred.
4. Solutions in which the complexity of the largest non-implementable function is minimized are preferred. This criterion helps to balance the complexity of the decomposed functions and derive balanced tree-like structures rather than linear ones[4].
5. The estimated savings obtained by sharing a function for the implementation of several output signals is also considered as a second order priority criterion.

Among the best candidate solutions for all output signals, the function with the largest complexity, i.e. the farthest from implementability, is selected to be implemented as a new output signal of the SG.

The complexity of a function is calculated as the number of literals in factored form. In case it is a sequential function and it matches some of the latches of the gate library, the implementation cost is directly obtained from the information provided by the library.

5.3 Signal acknowledgment and insertion

For each function delivered by the BR solver, an efficient SIP insertion must be found. This reduces to finding a partition $\{ER_A(x+), QR_A(x+), ER_A(x-), QR_A(x-)\}$ of the SG A such that $ER_A(x+)$ and $ER_A(x-)$ are restricted to be SIP-sets (Section 2.2). In general, each function may have several $ER_A(x+)$ and $ER_A(x-)$ sets acceptable as ERs. Each one corresponds to a signal insertion with different acknowledging outputs signals for its transitions. In our approach, we perform a heuristic exploration seeking for different $ER_A(x+)$ and $ER_A(x-)$ sets for each function. We finally select one according to the following criteria:

- Sets that are only acknowledged by the signal that is being decomposed (i.e. local acknowledgment) are preferred.
- If no set with local acknowledgment is found, the one with least acknowledgment cost is selected. The cost is calculated by incrementally deriving the new SG after signal insertion.

As an example consider the SG of Figure 4,c and the insertion of a new signal x for the function $x = c + d$. A valid SIP set for $ER_A(x+)$ would be the set of states $\{1100, 0100, 1110, 0110\}$, where the state $\{1100\}$ is the input border for the inserted function. A valid SIP set for $ER_A(x-)$ would be the set of states $\{1001, 0001\}$. With such insertion, $ER_A(x+)$ will be acknowledged by the transition $z+$ and $ER_A(x-)$ by $z-$. However, this insertion is not unique. For the sake of simplicity, let us assume that a and d are also output signals. Then an insertion with $ER_A(x+) = \{1100\}$ would be also valid. In that case, the transition $x+$ would be acknowledged by the transitions $a-$ and $d+$.

[4]Different criteria, of course, may be used when we also consider the delay of the resulting implementation, since then keeping late arriving signals close to the output is generally useful and can require unbalanced trees.

Circuit	literals/latches		CPU	Area	Area SI	
	old	new	(secs)	non-SI	2i	3i
alloc-outbound	14/4	12/2	8	264	256	264
chu133	12/1	12/1	4	208	210	200
chu150	14/2	8/3	8	144	144	160
converta	12/3	8/3	9	312	232	224
dff	12/2	0/2	1	128	80	80
ebergen	20/3	6/2	3	176	136	136
half	2/2	2/3	1	176	138	130
hazard	12/2	0/2	1	112	96	96
master-read	30/9	26/13	112	688	668	634
mp-forward-pkt	12/3	14/1	5	232	224	232
mr1	36/9	26/10	253	592	762	846
nak-pa	20/4	18/1	12	248	304	328
nowick	16/1	16/1	6	232	232	304
ram-read-sbuf	20/4	22/3	23	360	378	344
rcv-setup	10/1	8/1	1	120	128	128
rpdft	22/1	22/0	12	176	200	216
sbuf-ram-write	22/6	20/3	79	288	378	338
sbuf-send-ctl	22/5	22/2	16	280	262	250
sbuf-send-pkt2	32/5	30/5	113	296	530	304
seq-mix	30/6	30/7	64	592	586	664
seq4	18/7	18/5	131	400	582	450
trimos-send	36/8	12/9	101	480	600	644
vbe5b	10/2	8/4	2	216	204	226
vbe5c	4/3	2/3	2	216	168	168
vbe6a	16/7	32/10	16	600	456	456
vbe10b	28/7	26/11	35	608	552	584
wrdatab	48/7	30/9	290	608	708	714
Total	530/114	430/116		8752	9214	9129

Table 2: Experimental results.

5.4 Library mapping

The logic decomposition of the non-input signals is completed by a technology mapping step aimed at recovering area and delay based on a technology-dependent library of gates. These reductions are achieved by collapsing small fanin gates into complex gates, provided that the gates are available in the library. The collapsing process is based on the Boolean matching techniques proposed by Mailhot et al. [10], adapted to the existence of asynchronous memory elements and combinational feedback in speed-independent circuits. The overall technology mapping process has been efficiently implemented using BDDs.

6 Experimental results

6.1 Results in decomposition and technology mapping

The method for logic decomposition presented in the previous sections has been implemented and combined with the algebraic method presented in [7]. The results obtained from a set of benchmarks are shown in Table 2. The benchmarks correspond to those presented in [7] that were completely decomposed into 2-input gates.

The columns "literals/latches" report the complexity of the circuits derived after logic decomposition into 2-input gates. The results obtained by the method presented in this paper ("new") are significantly better than those obtained by the method presented in [7] ("old"). Note that the library used for the "new" experiments was deliberately restricted to D, Sr and Rs latches (i.e. without C-elements, since they are generally not part of standard cell libraries). This improvement is mainly achieved because of two reasons:

- The superiority of Boolean methods versus algebraic methods for logic decomposition.
- The intensive use of different types of latches to implement sequential functions compared to the C-element-based implementation in [7].

However, the improved results obtained by using boolean methods are paid in terms of a significant increase in terms of CPU time. This is the reason why the boolean method has been combined with the algebraic method (instead of substituting it), in such a way that boolean relations are only solved to find solutions that improve the cost of the solutions previously obtained by using algebraic methods.

6.2 The cost of speed independence

The second part of Table 2 is an attempt to evaluate the cost of implementing an asynchronous specification as a speed-independent circuit. The experiments have been done as follows. For each benchmark, the following script has been run in SIS, using the library `asynch.genlib`: `astg_to_f; source script.rugged; map`. The resulting netlists reported in column `area non-SI` could be considered a lower bound on the area of the circuit regardless of its hazardous behavior (i.e. the circuit only implements the correct function for each output signal, without regard to hazards). `script.rugged` is the best known general-purpose optimization script for combinational logic.

The columns labeled `area SI` report the results obtained by the method proposed in this paper. We report results on two strategies for decomposition before mapping the circuit onto a library:

- Decompose all gates into 2-input gates (`2i`).
- Decompose all gates into 3-input gates (`3i`).

Experiments have also been run for 4-input gates, with no tangible improvements. In both cases, decomposition and mapping preserve speed independence, since we do not use gates (such as MUXes) that may have a hazardous behavior when the select input changes. There is no clear evidence that performing an aggressive decomposition into 2-input gates is always the best approach for technology mapping. The insertion of multiple-fanout signals offers opportunities to share logic in the circuit, but also precludes the mapper from taking advantage of the flexibility of mapping tree-like structures. This trade-off must be better explored in forthcoming work.

Looking at the best results for non-SI/SI implementations, we can conclude that preserving speed independence does not involve a significant overhead. In our experiments we have shown that the reported area is similar[5]. Some benchmarks were even more efficiently implemented by using the SI-preserving decomposition. We impute these improvements to the efficient mapping of functions into latches by using Boolean relations.

7 Conclusions and future work

In this paper we have shown a new solution to the problem of multi-level logic synthesis and technology mapping for asynchronous speed-independent circuits. The method consists of three major parts. Part 1 uses Boolean relations to compute a set of candidates for logic decomposition of the initial complex gate circuit implementation. Thus each complex gate F is iteratively split into a two-input combinational or sequential gate G available in the library and two gates H_1 and H_2 that are simpler than F, while preserving the original behavior and speed-independence of the circuit. The best candidates for H_1 and H_2 are selected for the next step, providing the lowest cost in terms of implementability and new signal insertion overhead. Part 2 of the method performs the actual insertion of new signals for H_1 and/or H_2 into the state graph specification, and re-synthesizes logic from the latter. Thus parts 1 and 2 are applied to each complex gate that cannot be mapped into the library. Finally, Part 3 does library matching to recover area and delay.

This method improves significantly over previously known techniques [1, 8, 7]. This is due to the significantly larger optimization space exploited by using (1) Boolean relations for decomposition and (2) a broader class of latches[6]. Furthermore, the ability to implement sequential functions with SR and D latches significantly improves the practicality of the method.

[5]Taking the best results from `2i` and `3i` the total is 8582

[6]In fact, any sequential gate could be used, including, e.g., asymmetric C elements, the only limit being the size of the space to be explored.

In the future we are planning to improve the Boolean relation solution algorithm, aimed at finding a set of optimal functions compatible with a Boolean relation. This is essential in order to improve the CPU times and synthesize successfully more complex specifications.

References

[1] P. A. Beerel and T. H-Y. Meng. Automatic gate-level synthesis of speed-independent circuits. In *Proceedings of the International Conference on Computer-Aided Design*, November 1992.

[2] R. K. Brayton and F. Somenzi. An exact minimizer for boolean relations. In *Proceedings of the International Conference on Computer-Aided Design*, pages 316–319, November 1989.

[3] S. Burns. General conditions for the decomposition of state holding elements. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems, Aizu, Japan*, March 1996.

[4] T.-A. Chu. *Synthesis of Self-timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, MIT, June 1987.

[5] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Complete state encoding based on the theory of regions. In *International Symposium on Advanced Research in Asynchronous Circuits and Systems, Aizu, Japan*, March 1996.

[6] M. A. Kishinevsky, A. Y. Kondratyev, A. R. Taubin, and V. I. Varshavsky. *Concurrent Hardware. The Theory and Practice of Self-Timed Design*. John Wiley and Sons Ltd., 1993.

[7] A. Kondratyev, J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Technology mapping for speed-independent circuits: decomposition and resynthesis. In *Third International Symposium on Advanced Research in Asynchronous Circuits and Systems, Eindhoven*, April 1997.

[8] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the Design Automation Conference*, 1994.

[9] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for synthesis and testing of asynchronous circuits*. Kluwer Academic Publishers, 1993.

[10] F. Mailhot and G. De Micheli. Algorithms for technology mapping based on binary decision diagrams and on boolean operations. *IEEE Transactions on Computer-Aided Design*, 12(5):599–620, May 1993.

[11] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[12] Enric Pastor, Jordi Cortadella, Alex Kondratyev, and Oriol Roig. Structural methods for the synthesis of speed-independent circuits. In *Proc. of European Design and Test Conference*, pages 340 – 347, Paris(France), March 1996.

[13] P. Siegel and G. De Micheli. Decomposition methods for library binding of speed-independent asynchronous designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 558–565, November 1994.

[14] P. Siegel, G. De Micheli, and D. Dill. Automatic technology mapping for generalized fundamental mode asynchronous designs. In *Proceedings of the Design Automation Conference*, June 1993.

[15] P. Vanbekbergen, B. Lin, G. Goossens, and H. De Man. A generalized state assignment theory for transformations on Signal Transition Graphs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 112–117, November 1992.

[16] V. I. Varshavsky, M. A. Kishinevsky, V. B. Marakhovsky, V. A. Peschansky, L. Y. Rosenblum, A. R. Taubin, and B. S. Tzirlin. *Self-timed Control of Concurrent Processes*. Kluwer Academic Publisher, 1990. (Russian edition: 1986).

[17] Y.Watanabe and R.K. Brayton. Heuristic minimization of multiple-valued relations. *IEEE Transactions on Computer-Aided Design*, 12(10):1458–1472, October 1993.

[18] Y.Watanabe, L.M.Guerra, and R.K. Brayton. Permissible functions for multioutput components in combinational logic optimization. *IEEE Transactions on Computer-Aided Design*, 15(7):732–744, July 1996.