Technology Mapping of Speed-Independent Circuits Based on Combinational Decomposition and Resynthesis

Jordi Cortadella*
Universitat Politècnica de Catalunya, 08071 Barcelona, Spain

Michael Kishinevsky
The University of Aizu
Aizu-Wakamatsu, 965-80 Japan

Alex Kondratyev
The University of Aizu
Aizu-Wakamatsu, 965-80 Japan

Luciano Lavagno†
Politecnico di Torino
10129 Torino, Italy

Alex Yakovlev‡
University of Newcastle upon Tyne
NE1 7RU England

Abstract

This paper presents a solution to the problem of sequential multi-level logic synthesis of asynchronous speed-independent circuits. The starting point is a technology-independent speed-independent circuit obtained using, e.g., the monotonous cover conditions. We describe an algorithm for the factorization of this circuit aimed at implementing it in a given standard cell library, while preserving speed-independence. The algorithm exploits known efficient factorization techniques from combinational multi-level logic synthesis, but achieves also boolean *simplification. Experimental results show a significant improvement in terms of number and complexity of solvable circuits with respect to existing methods.*

1 Introduction

Recent years have seen a revival of interest in the sub-class of asynchronous circuits called *speed-independent* circuits. Such circuits have been characterized by Muller in his seminal paper [9] as being *hazard-free using the unbounded gate delay model.* Even though neglecting wire delays can be restrictive in practice, speed-independent circuits are a good starting point for synthesis and optimization procedures that use more detailed and realistic delay models. On the other hand, very efficient analysis and synthesis techniques, supported by CAD tools, exist for speed-independent circuits today.

Current synthesis techniques still suffer from a severe limitation: either they assume that the implementation library contains *and* gates with unbounded fanin and "free" input inversions ([1, 5, 8]) or they use non-standard "hazard absorbing" flip-flops whose effectiveness *in practice* still needs to be evaluated ([11]). Other results on the implementability of semimodular circuits without inputs using two-input/two-output *and* and *or* gates ([15]) are only interesting from a theoretical standpoint, due to their extremely high implementation cost.

Only recently people have begun to analyze the decomposability of speed-independent circuits using a given, realistic standard cell-like library. The approach described in [13] works only under the *fundamental mode assumption*[1], which is overly restrictive and does not fit well theoretically with the unbounded delay assumption. The same authors describe in [12] a method to perform technology mapping for speed-independent circuits that only decomposes existing gates (e.g., a 3-input *AND* into 2 2-input *AND*s), without any further search of the implementation space. They do not explore complex decompositions, that could use multi-cube divisors or decompose several gates simultaneously. The same limitations also affect the work of [1, 2].

Most recent examples of relevant work can be found in [10, 11, 4]; each of them lacks flexibility either with respect to the gate library, the scope of optimization or the extent of logic sharing.

The main contribution of this paper is an efficient solution of the technology mapping problem for speed-independent circuits. We have developed a body of theory that allows us to prune the search space when looking for solutions. We use classical logic synthesis techniques for combinational multi-level logic in order to find good candidate functions for the decomposition. We then derive efficient filtering conditions that guarantee:

- speed-independent implementability of the new signals, and
- a bound on the global increase in complexity of the circuit, due to the need to acknowledge the new signals.

*This work has been partly supported by the Ministry of Education of Spain (CICYT TIC 95-0419).

†This work has been partly supported by MURST research project "VLSI architectures".

‡This work has been partly supported by the U.K. EPSRC GR/J52327 and the British Council Programme (Spain) Acciones Integradas MDR/1996/97/1159.

[1] I.e., the environment is not allowed to change circuit inputs unless the circuit is stable.

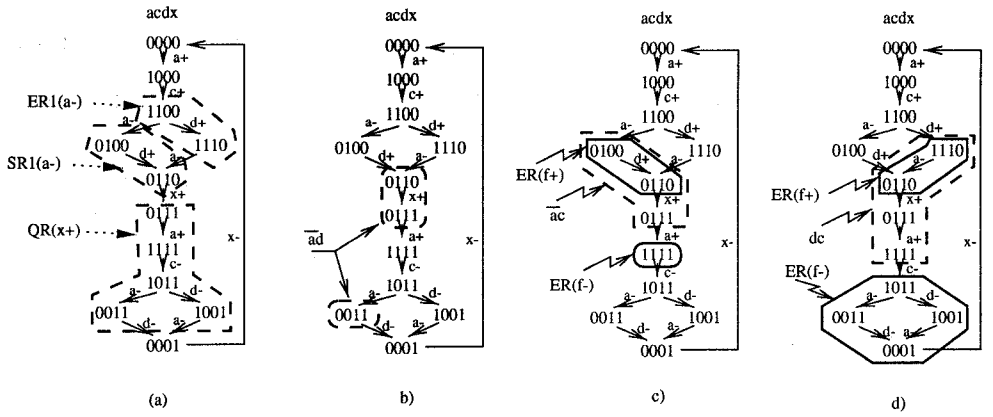

Figure 1: An example of State Graph and logic decomposition (benchmark *hazard.g*)

2 Theoretical background

In this section we introduce theoretical concepts required for understanding our decomposition method. These concepts are subdivided into three parts: (1) circuit specification and its basic logic implementability; (2) conditions of hazard-free decomposition of complex gates; and (3) correctness-preserving transformations to ensure those conditions. They are summarized in the following subsections.

2.1 State Graphs and Logic

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 transition (rising or falling) of an input or output signal of the specified circuit. 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:

- for each edge (s, a_i^+, s'), $v_i(s) = 0$, $v_i(s') = 1$, while $v_j(s) = v_j(s')$ for all $j \neq i$.
- for each edge (s, a_i^-, s'), $v_i(s) = 1$, $v_i(s') = 0$, while $v_j(s) = v_j(s')$ for all $j \neq i$.

Informally, this means that in every transition sequence from the initial state, rising and falling transitions alternate for each signal. Figure 1,a shows an SG which is consistent.

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

The first property is *speed-independence*, required for the existence of a *hazard-free circuit* implementation. It consists of three constituents: determinism, commutativity and output-persistency. 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. Any transformation (e.g., insertion of new, auxiliary, signals for decomposition) if performed at the SG level may affect all three properties.

The second property, *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 is the same. (The SG in Figure 1,a is *output-persistent* and has CSC.) CSC does not however restrict the type of logic function implementing each signal, and therefore CSC and SG speed-independence ensure hazard-freedom only if each signal is implemented as a *single atomic* gate. The complexity of such gate can however go beyond that provided in a concrete library or technology.

2.2 Hazard-free implementability

The decomposition of a complex gate into smaller gates creates new signals, that are not part of the original specification. In order to guarantee that these new signals do not produce hazards, we must ensure that their covers satisfy the important property of *monotonicity*, which is defined in this section.

Necessary and sufficient conditions for output-persistent implementation using unbounded fanin *and* gates (with unlimited input inversions), bounded fanin *or* gates and C elements were given in [8] (extending a previous result of [1]). In this work we are considering a similar basic implementation architecture, called the *standard-C* architecture, which is described in Figure 2. The difference from previous work is that instead of unbounded fanin *and* and *or* gates for the first and second levels, we will allow only *implementable* gates, that is gates which exist in the chosen library. The conditions derived in [8] are correct also in the presence of *input inversions* if the delay of the *inverter* does not exceed that of the remaining logic on

the fastest feedback loop involving the *inverter* itself.

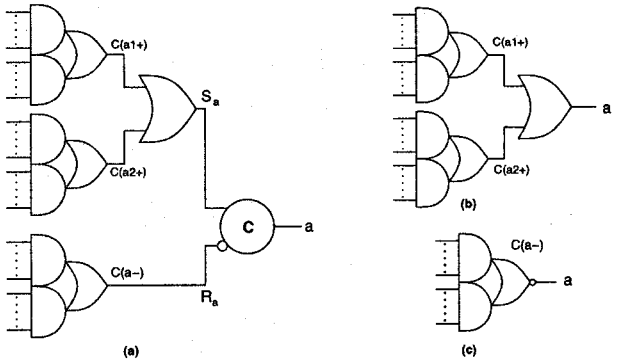

Figure 2: The standard-C architecture extended for complex gates

The concepts of excitation and quiescent regions are essential for defining the hazard-freedom conditions. A set of states is called an *excitation region* for event a^* (denoted by $ER_j(a^*)$) if it is a *maximal connected* set of states such that for every state $s \in ER_j(a^*)$ there is an event $s \xrightarrow{a^*}$. Since any event a^* can have several separated ERs, an index j is used to distinguish between *different connected occurrences* of a^* in the SG. Similarly to ERs, we define *switching regions* (denoted by $SR_j(a^*)$) as connected sets of states reached *immediately after* the occurrence of an event.

The set of events entering states of an excitation region $ER_j(a^*)$ from outside the region is called a set of *trigger events* for event a^*. Looking at a circuit implementation of an SG, signals whose events are trigger for an event of a signal a will *certainly* be inputs (called *trigger signals* for a) to the logic circuit implementing a.

The *quiescent region* $QR_j(a^*)$ of a given signal transition with excitation region $ER_j(a^*)$ is a *maximal* set of states s reachable from $ER_j(a^*)$ such that (1) a is stable in s and (2) s is not reachable from any other $ER_k(a^*)$ such that $k \neq j$ without going through $ER_j(a^*)$ [2]. Examples of ER, SR and QR are shown in Figure 1,a.

Let $c_j(a^*)$ denote one of the first-level complex AND-OR gates in the standard-C architecture. $c_j(a^*)$ is a *correct monotonous poly-term cover*[3] for the generalized excitation region $ER_j(a^*)$ if:

1. $c_j(a^*)$ covers (i.e., its Boolean equation evaluates to 1) all states of $ER_j(a^*)$.
2. $c_j(a^*)$ does not cover any state from $ER_i(a^*) \cup QR_i(a^*)$, where $i \neq j$.
3. $c_j(a^*)$ changes at most once within $QR_j(a^*)$.

Under these conditions, it is possible to show that the outputs of the first-level gates are *one-hot encoded*, and that means that any valid Boolean decomposition of the second-level *or* gates will be speed-independent.

The chosen architecture also covers the case in which a signal in the specification admits a *combinational* implementation (called a *complete* cover). In that case the set and reset network are the complement of each other, and the C element with identical inputs can be simplified to a wire (see Figure 2,b,c).

2.3 Property-preserving event insertion

Our decomposition method is essentially behavioural – the extraction of new signals at the structural (logic) level must be matched by an insertion of their transitions at the behavioural (SG) level. Event insertion is an operation on the SG which selects a subset of states, splits each state in it into two states and creates, on the basis of these new states, an excitation and switching region for a new event. Figure 3 shows the chosen insertion scheme, analogous to that used by most authors in the area [14], in the three main cases of insertion with respect to the position of the states in the insertion set $ER(x)$ (*entrance* to, *exit* from or *inside* $ER(x)$).

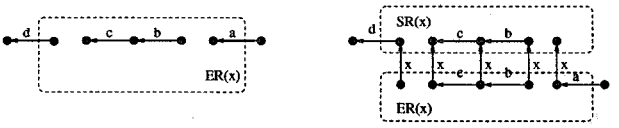

Figure 3: Event insertion scheme

State signal insertion must also preserve the *speed-independence* of the original specification, that is required for the existence of a hazard-free asynchronous circuit implementation. Formally, we say that an insertion state set $ER(x)$, in an SG A' obtained from a deterministic and commutative SG A by inserting event x, is a *speed-independence preserving subset (SIP-set)* iff: (1) for each $a \in E$, if a is persistent in A, then it remains persistent in A', and (2) A' is deterministic and commutative. An efficient method of finding SIP-sets based on the notion of regions has been proposed in [6].

Assume that the set of states S in an SG is partitioned into two subsets which are to be encoded by means of an additional signal. This new signal can be added either in order to satisfy the CSC condition, or to break up a complex gate into a set of smaller gates. In the latter case, a new signal is added to represent the output of the intermediate gate added to the circuit.

Let r and $\overline{r} = S - r$ denote the blocks of such a partition. In order to implement such an encoding, we need to insert appropriate transitions of the new signals in the *border states* between the two subsets.

[2] Note that contrary to [1, 8] in this paper we consider only the so called *restricted* quiescent regions, which do not include states reachable directly from two different ERs of the same signal (condition 2 of the definition)

[3] Here for simplicity we consider the definition of Monotonous Cover without the extension by the so-called *backward quiescent regions* and without considering covering of multiple regions by the same cover. However all the results can be easily generalized for this extension as well.

In this paper we shall consider the so-called *input border* (IB) of a partition block r, denoted by $IB(r)$, which informally is a subset of states of r which have predecessors not in r. We call $IB(r)$ *well-formed* if there are no events leading from states in $r - IB(r)$ to states in $IB(r)$.

Insertion of a new signal can be formalized with the notion of *I-partition* ([14] used a similar definition). Given an SG with a set of states S, an I-partition is a partition of S into four blocks: S^0, S^1, S^+ and S^-. $S^0(S^1)$ defines the states in which x will have the value 0 (1). $S^+(S^-)$ defines ER(x+) (ER(x−)). For a consistent encoding of x, the only allowed events crossing boundaries of the blocks are the following: $S^0 \rightarrow S^+ \rightarrow S^1 \rightarrow S^- \rightarrow S^0$, $S^+ \rightarrow S^-$ and $S^- \rightarrow S^+$.

3 The technology mapping method

As described in the previous section, any deterministic, commutative, output-persistent SG with the CSC property and satisfying the Monotonous Cover conditions can be implemented using the standard-C architecture. This section describes how the potentially large abstract gates derived from the Monotonous Cover implementation can be decomposed into library gates while maintaining speed-independence. The potentially huge search space is limited by an efficient search algorithm that prunes decompositions that are guaranteed to violate speed-independence.

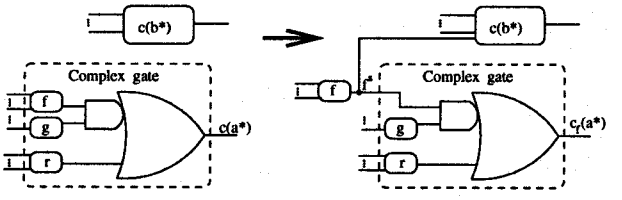

Figure 4: Decomposition of the cover function *c(a*)*

To simplify the implementation of $c(a*)$ a subfunction f is extracted from it and is implemented by a separate gate with output f^s. In a common form $c(a*)$ is represented by $c(a*) = f * g + r$ (where f, g and r are arbitrary boolean functions), as shown in Figure 4. This approach is more general than [12, 4] where switchings of a new signal f^s must be acknowledged by $c_f(a*)$ only and gate f always has a fan-out equal to 1. The acknowledgment of f^s by the covers different from $c(a*)$ (see e.g. the function $c(b*)$ in Figure 4) offers two advantages: the sharing of a gate f by several cover functions can simplify the implementation and, what is more important, succeed in many cases for which a local acknowledgment fails (see the experimental results).

The overall algorithm for signal insertion aimed at logic decomposition is sketched below. The next sections describe each step in more detail.

while circuit is not implementable **do**
  Calculate monotonous covers for all events;
  $a*$ = event with the most complex cover;
  /* Kernels, co-kernels, AND/OR decomposition */
  D = {set of divisors for $c(a*)$};
  for each $f \in D$ **do**
    Find I-partition for f;
    Evaluate progress for decomposition of $c(a*)$;
    /* (property 3.1) */
    Estimate progress for all other covers;
    /* (property 3.2) */
  end for
  if there is no $f \in D$ that can make progress on $c(a*)$
    then /* The cover $c(a*)$ cannot be decomposed */
    return;
    else $f_{best} = f$ with valid I-partition and
      best global decomposition progress;
    Insert a new signal with f's I-partition;
  end if
end while

The algorithm can be tuned by trading-off efficiency and quality of the results. For example, other events different from $a*$ can be also selected for decomposition in case no good divisor is found for $a*$.

Note that the algebraic divisors are only used for a preliminary choice of the function of the new signal to be added to the SG. The well-formedness conditions are then used to refine this function, so that it has a speed-independent implementation. The implementation of *every signal* in the circuit is recomputed at every step. This practically implements *boolean* division and can even obtain *sequential* decomposition. The conditions discussed below are used to guarantee progress at each step. We prune those divisors that would excessively increase the complexity of other signals due to the requirement to *acknowledge* every transition of the new signal to satisfy speed-independence.

We use the example *hazard.g* from the set of asynchronous benchmarks to illustrate our algorithm. Its SG is shown in Figure 1,a and an MC-implementation of the output signals c and d is presented in Figure 5,a. Our target is the decomposition of function S_x into two-input gates, because two-input gates are a standard worst case against which the performance of a decomposition algorithm can be measured.

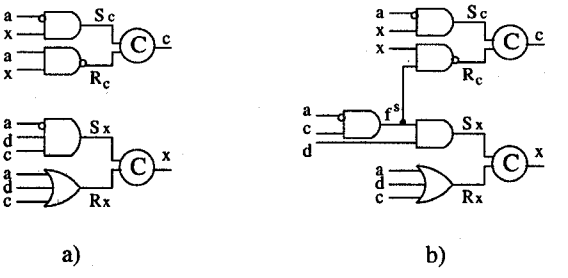

Figure 5: Circuits for a *hazard.g* example before (a) and after (b) decomposition

3.1 Logic decomposition

We assume here familiarity with multi-level logic synthesis (see [3] for more details). As tradition-

ally done in multi-level combinational synthesis, we have chosen algebraic division as the main operation for logic decomposition. However other divisors (e.g. boolean divisors) might also be considered within this scheme. For each event $a*$, there may be several optimal functions that can implement a monotonous cover $c(a*)$. Instead of finding decompositions for all valid covers, we choose only one of the minimum-cost covers and seek algebraic divisors for it, so as to avoid an explosion in the computational cost of the search.

Thus, for each cover $c(a*)$ we seek algebraic divisors, aiming at decompositions of the following type: $c(a*) = f * g + r$ where g is the quotient $c(a*)/f$. AND-decomposition is done when r=0, whereas OR decomposition occurs when g=1.

To find good divisors f for $c(a*)$ the following functions are considered:

- Kernels and co-kernels of $c(a*)$.
- If $c(a*)$ is a poly-term cover, any subset of terms of the sum-of-product expression (OR-decomposition).
- If $c(a*)$ is one cube, any subset of literals of the cube (AND-decomposition).
- Recursive decomposition of the previous candidates, e.g. sub-kernels and AND/OR-decomposition of kernels.

This generation of divisors is heuristically pruned to avoid an explosion of candidates for functions with many terms or cubes with many literals. Experimental results have shown this type of decomposition to be very effective.

Example hazard.g (Figure 1). Function S_x consists of a single 3-literal cube $\overline{a}dc$. It can be decomposed in three ways: by functions $\overline{a}d$, $\overline{a}c$ and dc.

Example 2. For the cover $c(x*) = ab + ac + def$ the following divisors are generated (trivial 1-literal divisors are not considered): the kernel $b + c$, the OR-decompositions ab, ac, def, $ab + ac$, $ab + def$ and $ac + def$ and the AND-decompositions de, df and ef.

3.2 Speed-independent implementation of decomposition function

A boolean function f defines a bipartition $\{S^0, S^1\}$ of the set of states of SG, where S^0 (S^1) contains states in which f=0 (f=1). To insert a signal f^s that realizes function f, it is necessary to find two additional sets of states: $ER(f^s+) \subset S^1$ and $ER(f^s-) \subset S^0$ in which f^s is enabled and fires from 0 to 1 and from 1 to 0, respectively.

Let us denote by $IB(f+)$ the set of SG states in which the function f changes the value from 0 to 1, i.e. $IB(f+) = \{s \in A, \exists s1 \rightarrow s \wedge f(s1) = 0 \wedge f(s) = 1\}$. Clearly $IB(f+)$ must be included into $ER(f^s+)$. Signal f^s can have a speed-independent implementation if and only if its excitation regions are well-formed SIP sets [6]. These sets can be obtained by the following iterative procedure:

Generation of $ER(f^s+)$

1. Start from $ER(f^s+) = IB(f+)$
2. Force well-formedness: recursively insert in $ER(f^s+)$ all states from S^1 that are direct predecessors for some state from $ER(f^s+)$
3. Force SIP properties: if some state diamond intersects with $ER(f^s+)$ in an illegal way remove the illegal intersection by inserting in $ER(f^s+)$ the corresponding states of the diamond
4. Preserve (if necessary) the input-output interface: if an input signal b can be delayed in an insertion of f^s by $ER(f^s+)$, then extend $ER(f^s+)$ beyond the $ER(b*)$, thus removing f^s+ from the set of triggers of b. Return to Step 2.

Calculation of $ER(f^s+)$ stops either if at some step $ER(f^s+)$ intersects with S^0 (then no legal $ER(f^s+)$ exists) or a fixed point is reached. It can be shown [7] that the above procedure always finds the well-formed SIP set $ER(f^s+)$ (if exists) that is minimal and unique. Calculation of $ER(f^s-)$ can be done similarly based on $IB(f-)$.

Example hazard.g. Figure 1,b implies that a decomposition of $S_x = \overline{a}cd$ by a function $f = \overline{a}d$ is illegal, because f intersects illegally with a state diamond $\{1011, 0011, 1001, 0001\}$ and this intersection cannot be removed by extending $IB(f+)$ without hitting the states where f=0.

Decompositions based on functions $\overline{a}c$ and dc are valid and the corresponding excitation regions of signal f^s are shown in Figure 1,c,d. Note, that states $\{0011, 1001\}$ are included into $ER(f^s-)$ (Figure 1,d) to preserve the I/O interface, otherwise input events $a-$ or $d-$ will be delayed by the insertion of f^s. State 0001 is included into $ER(f^s-)$ to preserve speed-independence.

3.3 Progress Analysis

A proper choice of $ER(f^s+)$ and $ER(f^s-)$ guarantees the possibility of a speed-independent implementation of a signal f^s. However it does not guarantee that the speed-independence of a target cover function $c(a*) = f * g + r$ is not disturbed when f is implemented by a separate gate. Conditions for safe substitution of f by signal f^s in $c(a*)$ are given by the following property. These conditions are formulated in terms of states of the original SG, i.e. before f^s is actually inserted. Hence the conditions can be efficiently checked *without reconstructing* the SG (as in [4]). Note, though, that in the formulation of the property we must consider set $QR(a*)' \supseteq QR(a*)$, that also includes states in the excitation regions of the transitions of a after $QR(a*)$, if f^s- becomes a trigger for those transitions.

Property 3.1 *Let $c(a*) = f * g + r$ be a monotonous cover of $ER(a*)$, and $ER(f^s+)$ and $ER(f^s-)$ be the excitation regions of a signal f^s (obtained as discussed above). The implementation $c_f(a*) = f^s * g + r$ satisfies the MC conditions iff:*

1. *If $s1 \in (ER(a*) \cap f*g*\overline{r}) \cap ER(f^s+)$ and $s1 \xrightarrow{a*} s2$ then state $s2 \notin ER(f^s+)$. This ensures that f^s evaluates to 1 in all states of $ER(a*)$ that are covered only by $f*g$.*

2. *For any state s outside* $ER(a*) \cup QR(a*)'$ *we have* $s \notin ER(f^s-) \cap g$ *This ensures that cube* f^s*g *cannot evaluate to 1 outside* $ER(a*) \cup QR(a*)'$

3. *For any state* $s \in (QR(a*) \cap f * g * \overline{r})$, $s \notin ER(f^s+)$, *and*

4. *for any state* $s1$ *predecessor of a state* $s2 \in QR(a*) \cap ER(f^s-) \cap g$, $s1 \in r+g$. *These two conditions ensure the monotonous behavior of* f^s*g *inside* $QR(a*)'$.

The proof is in [7].

Example hazard.g. All the conditions of Property 3.1 are satisfied for the decompositions $\overline{a}c$ and dc and for both of them S_x can be safely decomposed into two AND gates.

3.4 Cost estimation

We now estimate the complexity of the circuit obtained after decomposition with respect to that of the original circuit.

- *Complexity of the implementation of* f^s
  It can be shown, by analyzing the MC conditions ([7]), that $f^s = f$ is a correct complete cover for a signal f^s.

- *Events for which* f^s *is not a trigger*
  The preconditions for these events are not modified by the insertion of f^s, and hence we *can* use the same implementation as before the decomposition[4].

- *Events for which* f^s *is a trigger*
  Here we have two different cases.

  1. Trigger event f^s* replaces another trigger event $d*$ for some $ER(e*)$. In this case we try to substitute the literal f^s for d in the cover function $c(e*) = d * m + n$. This can be done by checking the conditions that are similar to those of Property 3.1.
  2. Trigger event f^s* does not replace any other trigger event for $ER(e*)$. In this case the complexity of the cover function for $ER(e*)$ can in general increase (unless the expanded don't care space induced by f^s* permits to further simplify it). This increase can be moderate (one literal) if the conditions of the following property are satisfied.

Property 3.2 *[7] Let* $c(b*)$ *be a monotonous cover for event* $b*$ *in SG A. If in the SG* A' *obtained from A by inserting the new signal* f^s*:*

1. *event* f^s+ *is a trigger for* $b*$;
2. $ER(f^s+) \cap SR(b*) = \emptyset$ *and*
3. $c(b*) \cap ER(f-) = \emptyset$

then the implementation $c(b*) * f^s$ *in* A' *for event* $b*$ *satisfies the Monotonous Cover conditions.*

This property is used as a heuristic filter to select candidate divisors that are guaranteed not to increase excessively the complexity of the implementation of other signals.

Note that this conservative estimation is applied not only for Case 2 but also for Case 1 when the substitution of an old trigger signal by a new one fails.

Example hazard.g. In the decomposition using dc (Figure 1,d) signal f^s becomes a new trigger event to $x-$ without replacing any other trigger event. Hence the cover for $x-$ will increase by one literal, while the cover for $x+$ will decrease by one literal. Hence this decomposition is not useful.

In the decomposition using $\overline{a}c$ event f^s- is inserted before $c-$ and replaces trigger event $a+$. Function for $c-$ will not increase in complexity. The result of the decomposition by function $\overline{a}c$ is shown in Figure 5,b.

4 Experimental results

The strategy for general logic decomposition previously presented has been implemented and applied to a set of different benchmarks. Results are shown in Table 1.

We have measured the complexity of each gate as the number of literals required to implement it as a sum-of-product gate, either complemented or not. Thus a 2-input XOR gate ($a\bar{b}+\bar{a}b$) is considered to be a 4-literal gate, whereas the function $f = \overline{ab+ac+db+dc}$ is also considered a 4-literal gate ($f = \overline{\bar{a}\bar{d} + \bar{b}\bar{c}}$). This model is slightly different from the one used in [4] in which the complexity was measured as the number of different inputs for FPGA lookup tables.

The first set of columns in Table 1 indicates the complexity of the circuit before decomposition. The second set of columns reports the number of signals inserted for decomposition using gates with at most i literals ($i = 2, 3, 4$). The next column summarizes the results presented by Siegel [12] about the implementability of the circuit with only 2-input gates. All the implementations have been verified to be speed-independent.

From the 32 examples, only 6 were not implemented (`n.i.`) with 2-literal gates. Only one 6-input AND gate in `pe-send-ifc` and two 5-literal gates in `tsend-bm` were not decomposed when attempting to implement these circuits with 4-literal gates. Our results show a significant improvement over those presented in [12], and only one circuit (`pe-rcv-if`) could not be implemented with 2-literal gates from that benchmark suite.

Global acknowledgement allows our method to effectively decompose complex gates with high fan-in (6 or 7 literals). This is shown by circuits like `mr1` and `vbe10b` that were implemented with 2-literal gates. Figure 6 illustrates this fact, depicting the circuit `vbe10b` before and after logic decomposition into 2-literal gates.

The final columns present a rough estimation of the cost for speed-independence-preserving logic decomposition. The cost is evaluated as the number of

[4] It is possible, though, that f^s can be used to simplify the implementation of those signals as well, by increasing the don't care space.

Circuit	# gates with n literals						our tech. mapping			Siegel [12]	lits/latches ($i=2$)	
	$n=2$	3	4	5	6	7	$i=2$	$i=3$	$i=4$	$i=2$	non-SI	SI
alloc-outbound	4	2					1	-	-		16/3	15/4
chu133	2	2					2	-	-	yes	13/2	13/1
chu150	3	2					2	-	-	no	16/1	17/2
converta	5	2					1	-	-	no	21/3	15/3
dff	2	2					2	-	-	yes	14/2	14/2
ebergen	5	2					3	-	-	no	21/2	23/3
half	1	1					1	-	-	no	7/2	3/2
hazard	2	2					2	-	-	yes	14/2	14/2
master-read	4	4	1				8	1	-		37/7	37/9
mmu	4	2	1	1	1		n.i.	n.i.	2			
mp-forward-pkt	2	2					2	-	-	yes	13/3	17/3
mr0	5	1	2	3	1	1	n.i.	5	4			
mr1	3	2	2	1	1		9	4	3		54/7	46/9
nak-pa	8	1					1	-	-	no	24/4	24/4
nowick	6	2					2	-	-	yes	21/3	16/1
pe-rcv-ifc	10	10	3	1			n.i.	5	1	no		
pe-send-ifc	10	7	7	1	1		n.i.	n.i.	n.i.			
ram-read-sbuf	4	3					2	-	-	yes	24/4	23/4
rcv-setup	1		1				2	1	-	yes	9/1	11/1
rpdft		1	3				5	2	-	yes	22/0	22/1
sbuf-ram-write	2	4					4	-	-	no	24/3	29/6
sbuf-send-ctl	2		1				n.i.	1	-			
sbuf-send-pkt2	5	4					6	-	-	no	28/3	30/3
seq_mix	2	3		2			15	3	2		38/6	75/10
seq4	1	2		1			4	1	1		22/5	25/7
trimos-send	6		3				11	3	-		33/6	47/8
tsend-bm	8	6	4	2		1	n.i.	n.i.	n.i.			
vbe5b	3	1					1	-	-	no	13/2	13/2
vbe5c		1					1	-	-	no	7/3	6/3
vbe6a	4		4				8	4	-		38/6	18/6
vbe10b	2	5				1	10	3	1		43/7	33/7
wrdatab	5	7	1				10	2	-		52/5	62/8
Total											624/92	648/109

Table 1: Experimental results

literals of the combinational gates and the number of C elements of the circuit. The column "non-SI" reports the cost of decomposing the original implementation of the circuit into 2-literal gates without preserving speed-independence (`tech_decomp -a 2` command in SIS). The column "SI" reports the cost of the decomposition preserving speed-independence. In some cases, such as `vbe6a`, the number of literals is reduced because the decomposition strategy allows to share logic among different covers. In most cases extra cost is added for the preservation of speed-independence. However, and considering that the area of a C element is roughly equivalent to a 3-input AND gate, we can conclude that the cost of preserving speed-independence is not higher than 10% of the area.

5 Conclusions and future work

In this paper we have shown a solution to the problem of multi-level logic synthesis and technology mapping for asynchronous speed-independent circuits. Let us summarise our two-step approach.

The first step (Section 3.1) chooses a *candidate* for decomposition: algebraic kernels, non-cube-free subcovers, sub-cubes etc. Different versions are evaluated

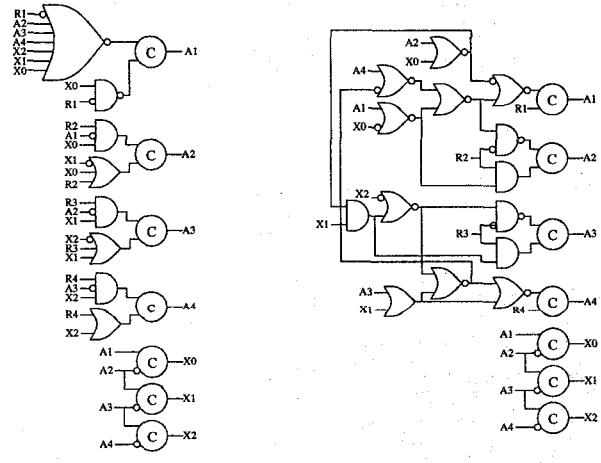

Figure 6: `vbe10b` before and after logic decomposition into 2-literal gates.

and the "best" one is taken. The "classical" combinational decomposition stops here.

The second step (Sections 3.2 and 3.3) performs the actual decomposition – it attempts to find a speed-independent implementation as similar as possible to the candidate obtained at step (1). This is based on a bi-partitioning using SIP-insertion corresponding to signal x obtained by the first step. Functions for all signals are derived from scratch. Our complexity arguments in Section 3.4 show that there is a good chance that x will get exactly the same function which was extracted at step (1). However, there is a chance also that this function will be smaller (thanks to boolean decomposition). Multiple acknowledgements for x appear automatically at this function generation step. Functions for signals which were not decomposed at step (1) may also change. Hence, the chosen implementation for x may correspond to a very general sequential decomposition. Moreover this is not a local, but "global" decomposition since other signals may change as well.

The method is implemented in the tool petrify. The results of the last section, to the best of our knowledge, show that the method appears to produce the most effective and efficient known decompositions of the standard set of asynchronous benchmarks (beating even the fundamental mode solutions). For example, examples such as vbe10 and wrdatab have been decomposed for the first time into two-input AND gates by a software tool.

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] Peter A. Beerel and Teresa H.-Y. Meng. Logic transformations and observability don't cares in speed-independent circuits. In *Proceedings of TAU 1993*, September 1993.

[3] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of IEEE*, 78(2):264–300, February 1990.

[4] 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.

[5] S. Burns and A. Martin. A synthesis method for self-timed VLSI circuits. In *Proceedings of the International Conference on Computer Design*, 1987.

[6] 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.

[7] J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Speed-independent circuit technology mapping: decomposition of combinatorial logic. Technical report, University of Aizu, Japan, September 1996.

[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] D. E. Muller and W. C. Bartky. A theory of asynchronous circuits. In *Annals of Computing Laboratory of Harvard University*, pages 204–243, 1959.

[10] 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.

[11] M. Sawasaki, C. Ykman-Couvreur, and B. Lin. Externally hazard-free implementations of asynchronous circuits. In *Proceedings of the Design Automation Conference*, June 1995.

[12] 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.

[13] 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.

[14] 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.

[15] 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).