using a modified version of the Berkeley synthesis system SIS [1]. The complexity of our synthesis tool is polynomial in the number of the states in the SG. The run-time performance of our program for all the circuits given in Table 1 are under 30 seconds using a DEC Station 3100.

5.1 Comparison with SIS

We have compared our synthesis algorithm with an alternative design technique for asynchronous circuits in which synchronous techniques are used to develop the initial logic, and delay elements are then added to remove hazards. This technique, developed by Lavagno *et. al.*, has been incorporated into SIS [1]. The synthesis results shown in Table 1 indicate that our optimized circuits are on average approximately 25% faster, with an area overhead of about 15%, than those obtained with SIS.

While the synthesis program implemented in SIS takes advantage of the complex gates available in the SIS library, such as AND-OR-INVERT and XOR which are much more area- and speed-efficient than the comparable set of basic gates, our synthesis results were obtained without using these complex gates, as only basic gates were assumed in our canonical structure. It is likely that with an extra step of technology mapping of our basic gates to the complex gates available in the library, our circuits will exhibit even better area and performance than reported in Table 1.

The following two paragraphs describe our comparison strategies. To compare area we had to reconcile the difference in the gate libraries used in the two synthesis systems. For example, the gate library used with SIS contained AND gates up to four inputs with some inputs invertered. However, to implement the pe-send-ic and vbe10b examples, we need AND gates with five and seven inputs respectively. We extrapolated the area of these gates from smaller gates and added an area and speed penalty equivalent to breaking up the large AND gate into two smaller ones. For the purposes of hazard-freedom, the two smaller AND gates are considered to form one atomic gate, since otherwise the added internal signal would have introduced hazards.

The delay of a circuit is taken to be the maximum path delay through a signal network. We model each inverted input of a gate as a separate INV gate and we assume none of the speed benefits of technology mapping. Even with these conservative assumptions our circuits are on average approximately 25% faster than those obtained with SIS.

The reasons for our speed advantage are two-fold. First, the recommended optimization script that is used with SIS seems to optimize for area rather than speed, and secondly, the delay lines that SIS needs to add to ensure hazard-freedom represent a significant fraction of the overall delay through the circuit, whereas a circuit designed with speed-independence would by definition eliminate these unnecessary delay lines.

6 Conclusions

We have demonstrated that gate-level synthesis of speed-independent circuits can be achieved systematically for a wide class of specifications including many examples from industry. The presented CAD tool implements the circuits hazard-freely on the gate level and incorporates speed-independence preserving logic optimizations to improve circuit speed and area. Compared with the most comprehensive benchmark set available [1], our optimized circuits are on average almost 25% faster with an area penalty of only 15% for preserving speed-independence. As asynchronous control circuits, being the largest application of our algorithm, tend to be relatively small and on the critical path, we believe that this area penalty would be more than compensated for by the gains in speed and in the circuit's resilience to delay variations.

Our future work includes breaking up large gates while preserving speed-independence to allow for gate libraries with limited fanin. We will also address maintaining hazard-freedom when bounded wire delays are taken into consideration as the constant wire delay assumption for multiple fanouts may not hold in practice. Furthermore, in some applications achieving speed-independence may not be considered the final goal, since if timing constraints were used in a constructive way, circuit area and performance can be further improved [10]. We will combine the automatic gate-level synthesis described in this paper with timing constraints of the environment and gate library to further reduce circuit complexity and increase circuit speed.

Acknowledgements: The authors would like to thank Professor David Dill of Stanford University for his many valuable suggestions directed towards improving our formalism. We also are greatly indebted to Luciano Lavagno, a Ph.D. candidate at University of California at Berkeley, for his valuable comments on early drafts of this paper, his assistance with installing SIS on our system, and his patience and cooperation in obtaining a fair comparison between our two synthesis methods. We would also like to thank the Synopsys Corporation for hosting the Asynchronous Circuit Seminar and the attendees for many valuable discussions on asynchronous circuits. In particular, we would like to thank Chris Myers for many valuable suggestions on the presentation of this paper and also Steve Nowick, Sheila Hemami, and Andy Hung for their time reviewing earlier drafts of this work.

References

[1] L. Lavagno, K. Keutzers, and A. Sangiovanni-Vincentelli. "Algorithms for Synthesis of Hazard-Free Asynchronous Circuits". In *Proceedings of the 28th ACM/IEEE Design Automation Conference*, 1991.

[2] Tam-Anh Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-theoretic Specifications*. PhD thesis, Massachusetts Institute of Technology, 1987.

[3] Alain J. Martin. "Programming in VLSI: From Communicating Processes to Delay-Insensitive VLSI Circuits". In C.A.R. Hoare, editor, *UT Year of Programming Institute on Concurrent Programming*. Addison-Wesley, 1990.

[4] D. L. Dill. "Trace Theory for Automatic Hierarchial Verification of Speed-Independent Circuits". ACM Distinguished Dissertations, 1989.

[5] David E. Muller and W. S. Bartky. "A Theory of Asynchronous Circuits". In *Proceedings of an International Symposium of the Theory of Switching*, pages 204–243, 1959.

[6] Peter A. Beerel and Teresa H.-Y. Meng. "Semi-Modularity and Testability of Speed-Independent Circuits". Accepted for publication in *INTEGRATION, The VLSI Journal*.

[7] V. I. Varshavky, editor. *Self-Timed Control of Concurrent Processes*. Kluwer Academic Publishers, Dordrecht, The Netherlands, 1990.

[8] Peter A. Beerel and Teresa H.-Y. Meng. "Gate-Level Sythesis of Speed-Independent Asynchronous Control Circuits", 1992. In collection of papers of the *ACM International Workshop on Timing Issues in the Specification of and Synthesis of Digital Systems*.

[9] S.H. Unger. *Asynchronous Sequential Switching Circuits*. New York: Wiley-Interscience, 1969.

[10] Chris Myers and Teresa H.-Y. Meng. "Synthesis of Timed Asynchronous Circuits". To appear in *International Conference on Computer Design, ICCD-1992*.

the l^{th} transition of u with a wire fork to the AND gate of a region network $\widehat{E_{s,k}}$ can be stated as follows:

1. $E_{u,l} \supseteq E_{s,k}$.
2. $E_{u,l'} \cap (Q_{s,k} \cup E_{s,k+1}) = \emptyset$, for all $l' \neq l$ such that $E_{u,l'}$ and $E_{u,l}$ either both constitute falling or both constitute rising transitions of u.

The first condition ensures that when the AND gate of the $\widehat{E_{s,k}}$ block is enabled high, the l^{th} transition of u is enabled. This prevents the wire fork connection from disabling the rising transition of the AND gate and ensures that the wire fork connection does not unduly delay the k^{th} transition of s. The second condition ensures that the internal signal does not glitch before the AND gate would normally fall, thereby guaranteeing that the wire fork connection does not create a 0-spike on the output of the AND gate. Hence, with these conditions satisfied, the added wire fork never introduces hazards and guarantees that the k^{th} transition of s acknowledges that the l^{th} transition of u has stabilized.

In some cases an additional requirement is needed on the choice of the acknowledging signal transition $E_{s,k}$ to ensure hazard-freedom. In the case of acknowledging the rising transition of an AND that is connected to an OR gate (as indicated in Figure 3), $E_{s,k}$ must precede the falling transition(s) of the other input(s) to the OR gate. This added restriction ensures that during the transfer of enabling between OR gate inputs, at least one input of the OR gate is guaranteed to be high, thereby avoiding a static hazard. Similarly, in the case of acknowledging the falling transition of an OR gate that is connected to an AND gate, $E_{s,k}$ must precede the subsequent rising transition(s) of the other input(s) of the AND gate.

There are two rare cases in which our algorithm would fail and adding state variables will be necessary, which can be accomplished by the algorithm described in [8]. The first case is when no $\widehat{E_{s,k}}$ can be found to acknowledge the transition of an internal signal. The second case is when an input choice state transition causes multiple term takeovers so that the cover of a region network derived from its implementation violates the right hand side of the valid states constraint. We have not yet come across a specification for which state variable introduction is needed.

4 Optimizations

The initial implementations derived from the proposed algorithm are in general very inefficient, but our canonical implementation structure permits effective systematic optimization.

The classical logic transformations applied to asynchronous circuits described by Unger in [9] rely on the fundamental mode assumption, and they might introduce hazards in speed-independent circuits. However, it is not difficult to show that simple transformations such as sharing identical gates, removing one-input gates, and merging AND gates in SOP blocks that have a Hamming distance of one, are speed-independence preserving.

In addition, if the covers of the set (reset) regions network of a signal includes their associated quiescent regions, then the set (reset) network can implement the signal network. For simplicity consider the case where the cover of each set region network of some signal s includes its corresponding quiescent region. Then the set network is enabled high in all states that s is enabled high and can therefore implement signal s. While this means the reset network is not needed to implement signal s, it may still be needed to acknowledge some internal signal transition. But as long as no acknowledgement wire fork is connected to any of the reset region networks, they, as well as the C-element for signal s, can be removed, reducing s's signal network to s's set network.

Circuit	$\lvert\mathcal{V}\rvert$	Our CAD tool		SIS
		Unopt. Area/Del	Opt. Area/Del	Area/Del
chu133	24	648 / 6.0	240 / 4.8	352 / 5.2
chu150	26	576 / 6.0	240 / 4.8	232 / 7.0
chu172	12	464 / 6.0	152 / 3.6	104 / 1.6
converta	18	616 / 6.0	520 / 6.0	432 / 6.8
ebergen	18	600 / 6.0	344 / 4.8	280 / 5.6
full	16	336 / 6.0	112 / 2.4	224 / 5.2
hazard	12	352 / 6.0	256 / 4.8	296 / 6.6
hybridf	80	784 / 6.0	152 / 2.4	274 / 6.6
nowick	20	712 / 6.0	504 / 6.0	264 / 6.6
pe-send-ic	117	2448 / 7.2	2072 / 7.2	1232 / 12.2
qr42	18	600 / 6.0	344 / 4.8	280 / 5.6
vbe10b	256	1280 / 7.2	800 / 6.0	1008 / 10.0
vbe5b	24	504 / 6.0	264 / 4.8	272 / 4.2
vbe5c	24	480 / 6.0	264 / 4.8	224 / 5.2
wrdatab	216	1264 / 6.0	840 / 4.8	824 / 7.0
var1	12	744 / 8.8	256 / 5.0	184 / 3.0
xyz	8	504 / 7.4	200 / 4.8	120 / 3.2
Totals		12912 / 109	7560 / 82	6602 / 102

Table 1: Experimental results.

Furthermore, in many cases the acknowledgement wire forks create redundant inputs that can be removed from a gate. In speed-independent circuits, an input is redundant when it does not prevent a transition from firing, does not cause a transition to fire, and does not maintain the enabling of a transition.

In addition to the above mentioned optimizations, variables that are common to both the set and reset networks of a signal network are removable in some cases. If a signal transition causes the set (reset) region network to rise and the reset (set) region network to fall, then under certain conditions the variable can be removed from the set (reset) region network. These conditions ensure that when the signal is removed from the set (reset) region network, the set (reset) region network may turn on early but otherwise must be hazard-free. Removing the signal under these circumstances does not change the behavior of the output signals, because even though the set (reset) region network turns on early, the C-element waits for the reset (set) region network to fall before firing.

These optimizations have been automated in the CAD tool, as will be presented in the next section.

The proofs that these optimizations are speed-independence preserving are omitted because of space constraints. The effectiveness of these optimizations has been very promising. As indicated by Table 1, these optimizations on the average reduce the area of our canonical implementation by about 40% and increase the speed of the circuit by about 20%.

5 Experimental results

The synthesis algorithm and optimization techniques described in this paper have been automated in a CAD tool written in C. To double check the correctness of our CAD tool, we have automated the generation of appropriate specification and implementation files for AVER, the speed-independent circuit verifier developed by Dill [4]. Using AVER, all of our implementations have been automatically verified to be speed-independent implementations of their respective specifications.

To estimate the performance of our synthesis algorithm, we have generated a speed-independent gate-level implementation for all the specification examples taken from the benchmarks given in [1]. The SG specifications for these examples have been obtained from their signal-transition-graph specifications

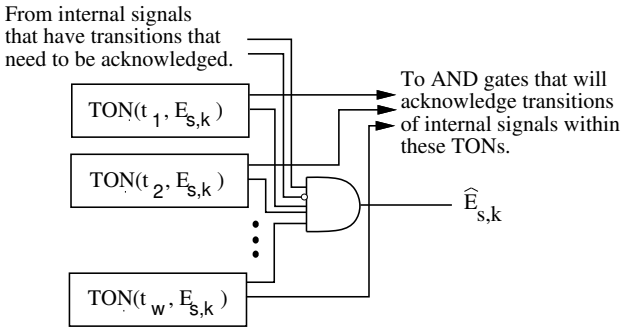

Figure 2: Region network implementation strategy.

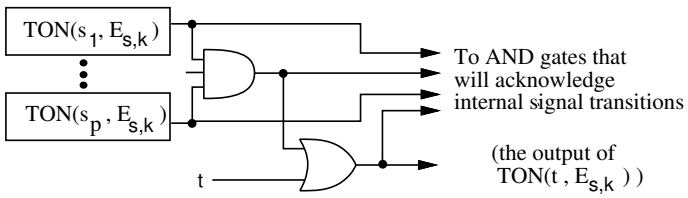

Figure 3: Recursive logic framework for region network expression defined by (1).

3.4.1 The R_Expand() function

The function R_Expand($\alpha, E_{s,k}$) uses a greedy algorithm to find the largest cube that both covers α and satisfies the cube conditions for $E_{s,k}$. First R_Expand() converts its state argument to an equivalent cube defined by the level of each signal specified by the binary code of α. Then R_Expand() iteratively expands the initial cube, using a heuristic that attempts to minimize the number of cubes needed to cover an excitation region, as described below.

At each stage of the cube expansion, R_Expand() determines all the cube literals that can be made into don't-cares without violating the cube conditions. For each such literal R_Expand() determines the corresponding expanded cube and the set of valid states that it covers. The expanded cube that covers the most states in the excitation and associated quiescent region is chosen; it is referred to as *best_cube*. Looping back, the algorithm repeats by expanding best_cube. The algorithm terminates when best_cube cannot be expanded further without violating the cube conditions.

3.4.2 Hazard-free combinational logic design

This subsection presents the algorithm used for obtaining a hazard-free region network $\widehat{E_{s,k}}$.

Let $T^1_{s,k}$ be the term corresponding to the first cube of the region network cover found by R_Expand(). For each literal t_i in $T^1_{s,k}$, we design a combinational network that is enabled high when the corresponding transition of t_i fires and remains enabled high until some time after s fires. As depicted in Figure 2, these combinational networks, called *takeover networks (TONs)*, are combined with an AND gate to form the region network. The presence of the additional wire connections in Figure 2 from the TONs and to the AND gate denotes acknowledgement wire forks to be discussed later.

The generation of the TON of a literal t in the first term is handled according to one of three cases: 1) Signal(t), the signal corresponding to literal t, is not enabled in a state in $E_{s,k}$; 2) Signal(t) is enabled in some state in $E_{s,k}$ that is covered by a cube in current cover; and 3) Signal(t) is enabled in some state in $E_{s,k}$ but in no states that are included in the current cover.

In case 1, the TON is t. The corresponding logic is simply a wire if Signal(t) is high or an inverter if Signal(t) is low.

In case 2, however, Signal(t) changes in a state α covered by the first term and enters a state β which is also in the excitation region but not covered by the first term. In this case a new cube is added to the cover that takes over the enabling of s when Signal(t) fires. This new term $T^2_{s,k}$ is found by expanding β as much as possible using R_Expand() and then constraining the resultant term to have every literal t_i in $T^1_{s,k}$ except t.

We need to constrain $T^2_{s,k}$ so that the sum of $T^1_{s,k}$ and $T^2_{s,k}$ is expressible in a factored form, as follows:

$$T^1_{s,k} + T^2_{s,k} = (t_1 \cdot t_2 \cdot \cdots \cdot t_m)(t + (s_1 \cdot s_2 \cdot \cdots \cdot s_p)). \quad (1)$$

The literals $t_1, t_2, \ldots, t_m$ and t make up $T^1_{s,k}$ and $s_1, s_2, \ldots, s_p$ are the additional literals in $T^2_{s,k}$ found by R_Expand($\beta, E_{s,k}$).

As indicated in Figure 3, the logic for the TON of t essentially follows from the factored form in (1). We first AND the TONs of s_1 through s_p. Then, we OR the output of this AND gate with the logic for the literal t. Because each signal corresponding to $s_i, i = 1, \ldots, p$, can also change in the excitation region, we AND the TONs of s_i that are recursively generated, yielding in general a multi-level logic implementation.

In case 3, Signal(t) changes in a state in $E_{s,k}$ which is not yet covered by any term currently in $\widetilde{E_{s,k}}$. This situation can always be avoided by making TONs of the other literals first, thereby delaying implementing the problematic takeover until an appropriate term is in the cover. This preordering algorithm is relatively simple, and we have not yet come across a specification for which it is needed.

3.4.3 Acknowledgement wire forks

The last step of the region network synthesis algorithm is to ensure speed-independence, i.e. that every transition of every internal signal is acknowledged [3].

To acknowledge a previously unacknowledged transition of an internal signal, we add a wire that connects the internal signal to the AND gate of a region networks that control a different output signal transition (as depicted in Figure 2). This different output signal transition is chosen to be a transition which must fire before the internal signal transition can be disabled. The wire fork connection to the AND gate in the region network guarantees the internal signal stabilizes by ensuring the different output signal cannot fire until the internal signal has stabilized. These acknowledgement wire forks are selected such that they do not reduce the concurrency of the circuit.

As an example, consider the OR gate in the region network depicted in Figure 3. While a rising transition of this OR gate is acknowledged by the rising transition of the signal network's output, the falling transition of the OR gate is not acknowledged. To acknowledge the falling transition, an acknowledgement wire fork is necessary to connect the output of the OR gate to an AND gate of some other region network, as shown in Figure 2. Since the signal transition to be acknowledged is a falling transition, an inverter bubble is needed at the AND gate input.

To characterize the conditions that govern which $\widehat{E_{s,k}}$ can acknowledge an internal signal transition, we first define the concept of an excitation region for internal signals. For an internal signal u, the notation $E_{u,l}$ represents a maximal connected set of states for which in all states $\alpha \in E_{u,l}$ we either always have $\widehat{u}(\alpha) = 0$ or always have $\widehat{u}(\alpha) = 1$. The subscript l indexes the transition number as k does $E_{s,k}$. Unlike input and output signals, internal signals have no associated quiescent regions.

Sufficient conditions for a successful acknowledgement of

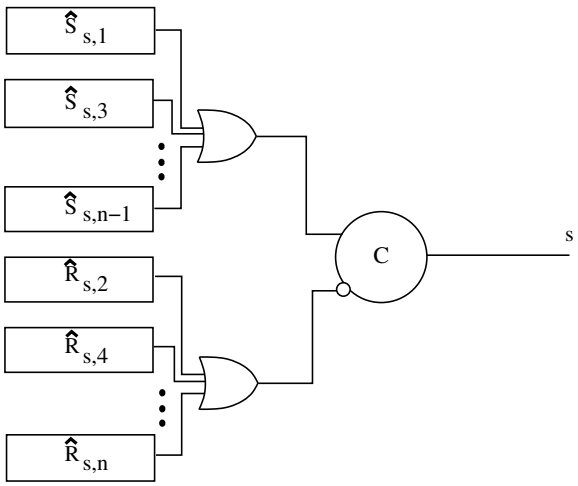

Figure 1: Signal network for s.

The inverter bubble attached to the bottom input of the C-element in Figure 1 provides symmetry in the region networks. That is, in each of the $S_{s,k}$ and $R_{s,k}$ excitation regions, the corresponding Boolean expressions $\widehat{S_{s,k}}$ and $\widehat{R_{s,k}}$ evaluate to 1. This decomposition strategy may not lead to a minimal solution, particularly if logic can be shared in the set and reset region networks. Optimizations presented in section 4 address this issue.

3.2 Region network synthesis: Theory

The set of states in which a region network $\widehat{E_{s,k}}$ evaluates to one is referred to as the *cover* of the region network and denoted by $\widetilde{E_{s,k}}$.

The synthesis of a region network amounts to finding a correct cover of its excitation region and a corresponding hazard-free multi-level logic implementation.

Definition 3.1 (Correct cover) *A cover $\widetilde{E_{s,k}}$ is correct if it satisfies the following conditions.*

1. **Valid states constraint:** $E_{s,k} \subseteq (\widetilde{E_{s,k}} \cap \mathcal{V}) \subseteq (E_{s,k} \cup Q_{s,k})$. *Note that $\widetilde{E_{s,k}} \cap \mathcal{V}$ expresses the valid states in $\widetilde{E_{s,k}}$.*

2. **Entrance constraint:** *The cover $\widetilde{E_{s,k}}$ can only be entered through states in $E_{s,k}$. By saying a cover can be entered through state α, we mean that α is in the cover and has a predecessor state which is outside of the cover.*

The significance of a correct cover is made explicit in the following claim.

Claim 3.1 *If for all signals $s \in \mathcal{O}$ all region network covers $\widehat{E_{s,k}}$ are correct and are implemented hazard-freely by the corresponding region network, then the implementation framework depicted in Figure 1 leads to a speed-independent implementation.*

Sketch of proof: According to Definition 2.4, our proof obligation is to show that this implementation framework satisfies output equivalence and speed-independence.

Given correct covers, it is easy to verify that the Boolean expression of each output signal satisfies the SG specification (i.e. output equivalence) by evaluating each output signal's implied value in each valid state.

The fact that circuit's behavior is independent of gate delays (i.e. speed-independence) follows from the definition of a correct cover. The left half side of the valid states constraint guarantees that a correct cover must contain its associated excitation region. Given this, the entrance constraint ensures that a cover $\widetilde{E_{s,k}}$ will be entered when the circuit is actively trying to fire the k^{th} transition of s and at no other time. The right half side of the valid states constraints ensures that a cover cannot include states of more than one excitation region. Given this, the entrance constraint ensures that no two region network covers of the same signal can be entered at the same time and that when a new cover is entered, the particular state entered must not be in the previous cover of the same output signal. Therefore, for each signal, every valid state can only be in one cover and a maximum of one region network output can be enabled high at any time. Since each region network $\widehat{E_{s,k}}$ is assumed hazard-free, it then follows that mutual exclusion at the input of the OR gates in each half of the signal network is guaranteed. We can then conclude that the changing of s *acknowledges* the transitions of the internal signals within s's signal network [3] and that the implementation is speed-independent. ∎

The correct cover conditions govern the construction of the Karnaugh maps used in the synthesis of each region network: 1's should be placed in states $q \in E_{s,k}$; "pseudo" don't-cares in states $q \in Q_{s,k}$; don't-cares in invalid states (i.e. states not in $\mathcal{V}$); and 0's elsewhere. The "pseudo" don't-cares are different from common don't-cares in that "psuedo" don't-cares can be included in a cover only if their inclusion does not violate the entrance constraint.

3.3 Region network synthesis: Overview of the algorithm

Because the algorithm for region network synthesis is relatively complex, we initially present a simplified overview. We first find all the excitation and quiescent regions for all transitions of all output signals. The synthesis of a particular region network can then be broken down into the following steps.

1. Find the minimum state of the excitation region.

2. Call R_Expand() (described in the next subsection) to find the largest cube that covers this state and satisfies the entrance constraint and the right half of the valid states constraint, together referred to as the *cube conditions.*

3. Add the newly found expanded cube to the region network cover.

4. If the new cover is correct, go to step 5; otherwise, the current cubes in the cover must not cover the entire excitation region. In the latter case, find a state transition from a covered state of the excitation region to an uncovered state of the excitation region. This state transition causes what is called a *term takeover*, as we will describe later. Call R_Expand() to find a new cube which covers this uncovered state and go to step 3.

5. At this step we have a correct cover. We recursively create multi-level logic that implements the takeovers hazard-freely. We guarantee this logic to be hazard-free by synthesizing *wire forks* to *acknowledge* all internal signal transitions, such that all internal signal transitions must stabilize before they can be disabled.

3.4 Region network synthesis: Details of the algorithm

In this subsection we first describe R_Expand(), the core function of the algorithm that finds a correct cover. Then we describe the multi-level logic implementation of each cover and the creation of the acknowledgement wire forks.

2.2 Speed-independent SG specifications

We refer to a speed-independent specification as one in which the specified behavior is independent of the ordering of concurrent signal transitions. In a SG specification $\langle \mathcal{T}, \mathcal{V}, \rightarrow \rangle$ the types of speed-independent state transitions of interest in this paper are semi-modular transitions and input choice transitions, as defined below.

Definition 2.1 (Semi-modular state transition) *The state transition $\alpha \xrightarrow{s} \beta$ is semi-modular if and only if as a result of the transition no enabled signal transition is disabled.*

To formalize input choice behavior, we require the signals modeled in the specification $\mathcal{S}$ to be formed from two disjoint sets, an input signal set denoted by $\mathcal{I}$ and an output signal set denoted by $\mathcal{O}$. We give the following definition of input choice.

Definition 2.2 (Input choice state transition) *The state transition $\alpha \xrightarrow{s} \beta$ is an input choice if and only if s is an input signal, which, when it fires in state α, disables some other input signal(s) but not any other type of signal.*

Transitions corresponding to the outputs of arbiters do not fall in either class because arbiters can non-deterministically choose one of multiple enabled output transitions to fire, thereby disabling other enabled output transitions. We do not present a formalism of this type of behavior as we assume that these gates can be introduced to the circuit design manually. This allows us to model the outputs of the arbiters as inputs to the circuit, and their non-deterministic output transitions as input choice state transitions.

To further refine the class of SG specifications that we can synthesize we must introduce the notion of *excitation regions.* An excitation region of signal s, denoted by $E_{s,k}$, is a maximal connected set of states in which signal s is enabled. The index k is assigned so that no two excitation regions of signal s have the same region index.

We also define a *minimal state* in an excitation region $E_{s,k}$ as a state $\beta \in E_{s,k}$ for which there exists no state $\alpha \in E_{s,k}$ such that $\alpha \rightarrow \beta$. Intuitively the set of minimal states constitutes the first states that can be entered in which a particular transition of s is enabled. It is easily seen that every output signal excitation region has at least one minimal state because we do not allow our specifications to have a cycle of states in which some output signal transition is continuously enabled. Such a cycle of states constitutes a *fake equivalence class* [7] and is never a practical construct.

We now can define the subclass of SGs to which our synthesis algorithm can be applied.

Definition 2.3 (Distributive SG with input choice) *A distributive SG with input choice is a SG in which every transition is either a semi-modular state transition or an input choice state transition and in which all excitation regions have exactly one minimal state, referred to as the* minimum state.

Specifications that have excitation regions containing multiple minimal states include the specification of an OR construct and other disjunctive constructs. We presently cannot synthesize such specifications.

To emphasize the polarity of the k^{th} excitation region of s, we use $R_{s,k}$ in the case of a falling transition (reset) and $S_{s,k}$ in the case of a rising transition (set). To each excitation region $E_{s,k}$ we associate a connected quiescent region $Q_{s,k}$ that can be entered from $E_{s,k}$ and contains states in which signal s is stable.

2.3 Speed-independent implementations

In this paper, we define a circuit implementation as a pair $\langle \mathcal{M}, \mathcal{G} \rangle$. The set $\mathcal{M}$ is a set of circuit nodes, decomposed into three sets of signals, input signals $\mathcal{I}$, output signals $\mathcal{O}$, and internal signals $\mathcal{N}$. The set $\mathcal{G}$ is a set of Boolean functions containing one function for each internal and output signal. The input signals do not have Boolean functions corresponding to them; their behavior is defined by the SG specification.

The Boolean expression of a signal $s \in \mathcal{N} \cup \mathcal{O}$ is expressed in terms of the inputs and outputs of the circuit and is denoted by $\widehat{s}$. Also, we denote the evaluation of this expression in state α by $\widehat{s}(\alpha)$. We define the *implied value* of a signal s in state α as follows: if s is enabled in α then its implied value is the complement of the present value in α (specified by the binary code of α); if s is not enabled in α then its implied value is its present value in α [2]. We now can provide the following definition of a speed-independent implementation.

Definition 2.4 (Speed-independent implementation of a distributive SG with input choice) *A circuit is a speed-independent implementation of a distributive SG with input choice if and only if*

1. **Output equivalence:** *The Boolean expression of each output signal satisfies the specification. More formally, for each $s \in \mathcal{O}$ and for each valid state α we have that $\widehat{s}(\alpha)$ evaluates to the implied value of s in α.*
2. **Speed-independence:** *The actual behavior of the circuit when the inputs behave as specified by the specification is independent of gate delays.*

Hence an implementation is speed-independent if its behavior satisfies its SG specification and is independent of the individual gate delays, assuming constant wire delays for multiple fanouts.

3 Synthesis procedure

Our synthesis strategy consists of four steps: identifying all excitation regions for each signal transition; constructing for each output signal transition a logic block whose primary output is enabled high in all states of its excitation region and some states of its quiescent region; combining logic blocks of different excitation regions of the same signal in the framework depicted in Figure 1; and optimizing for speed and area using speed-independence preserving transformations.

We assume that our gate library contains basic gates such as ANDs, ORs, and Muller C-elements with arbitrary numbers of fanin and fanout and any number of inverters attached to inputs. To ensure hazard-freedom, the library designer must ensure that all gates in the library can be considered *atomic*, which means that each basic gate can be modeled as an instantaneous Boolean function evaluator followed by an arbitrarily long pure delay element.

3.1 Canonical implementation strategy

Figure 1 depicts our decomposition strategy. To implement an output signal s we create one *region network* for each of its excitation regions $E_{s,k}$ and combine the region networks with two OR gates and a two-input Muller C-element. One OR gate combines the set region networks and one OR gate combines the reset region networks. The resulting network is referred to as the *signal network* for s.

The two-input Muller C-element is a memory element with inputs A and B and one output C. The next state equation is $C' = (A + B)C + AB$. Once both inputs of the C-element become 1, the output is enabled high, and will at some time later go high. It will then remain high until both inputs become 0, at which time the output is enabled low. Multiple-input C-elements are similarly defined.

Appeared in IEEE ICCAD-92 Digest of Technical Papers

Automatic Gate-Level Synthesis of Speed-Independent Circuits

Peter A. Beerel and Teresa H.-Y. Meng[1]
Computer Systems Laboratory
Stanford University
Stanford, CA 94305

Abstract

In this paper, we present a CAD tool for the synthesis of asynchronous control circuits using basic gates such as AND gates and OR gates. The synthesized circuits are speed-independent; that is, they work correctly regardless of individual gate delays. We present synthesis results for a variety of specifications taken from industry and previously published examples. We compare our speed-independent circuits with those non-speed-independent circuits synthesized using the algorithms described in [1], in which delay elements are added to remove circuit hazards. These synthesis results show that our circuits are on average approximately 25% faster with an area penalty of only 15%. This work demonstrates that direct synthesis of gate-level speed-independent circuits is not only feasible, but also produces robust and relatively efficient circuits compared to those synthesized with timing constraints.

1 Introduction

While the automated synthesis of speed-independent circuits using complex gates has been investigated [2, 3], automated gate-level synthesis using basic gates has long been considered difficult. Gate-level implementations of hazard-free asynchronous circuits are of particular interest because they facilitate fast prototyping through standard cells or programmable gate-array packages. In this way, asynchronous circuit synthesis need not rely on custom design.

Speed-independent circuits are attractive because they can be proven correct [4] and tolerate delay variations resulting from IC processing or temperature variations. These circuits work correctly regardless of the delays of individual gates, while assuming constant wire delays for multiple fanouts [5]. As a result, achieving speed-independence avoids the need for timing assumptions and delay lines that tend to increase area and reduce the reliability of the circuit. Speed-independent modules can thus easily be replaced whenever a faster or more advanced technology is introduced. In addition, speed-independent circuits have a testability advantage—they are self-checking with respect to output stuck-at-faults [6]. Speed-independent circuits also inherit advantages from their asynchronous nature. These advantages include the absence of clock-skew, freedom from designing for worst-case delay, and automatic power-down of unused circuitry.

Kishinevsky and Varshavsky showed that all *distributive* specifications (to be defined in a later section) can be implemented speed-independently using complex AND-OR-NOT gates or 2-input NAND gates [7]. Their goal, however, was to prove functional completeness, not to optimize speed and area. As a result, their algorithm produces large, complex circuits because it unnecessarily adds many state variables to avoid circuit hazards. This paper describes a CAD tool for the gate-level synthesis of speed-independent circuits that uses combinational logic instead of adding state variables whenever possible. This feature increases gate sharing, which reduces circuit area, and shortens the critical path, which increases circuit speed. Furthermore, our tool accepts input specifications from an extended class of distributive specifications that includes input choice [2].

The organization of the paper is as follows. Section 2 presents the definition of and motivation for our state graph specification style. Section 3 presents the algorithm for synthesizing our canonical hazard-free gate-level implementations. Section 4 describes simple logic optimizations that optimize our canonical implementation while preserving speed-independence. Section 5 presents the experimental results of our synthesis tool when applied to the benchmark set given in [1] and compares our implementations with available counterparts. Finally, we give our conclusions in Section 6.

2 State graph specification

Our synthesis algorithm starts with a low-level state graph (SG) specification that can be obtained through translation from many high-level specifications. Our algorithm can be applied to SG's that include input choice [2] but not disjunctive behavior (to be defined in the next subsection). This specification constraint is commonly assumed in asynchronous circuit synthesis, and this specification class includes all the asynchronous circuit benchmarks presented in [1] and other previously published examples.

2.1 State graph (SG)

A SG is a triple $\langle \mathcal{T}, \mathcal{V}, \rightarrow \rangle$, where $\mathcal{T}$ is a set of transitions, $\mathcal{V}$ is a set of unique *valid* states, and $\rightarrow$ is a transition relation. We define a SG *over* a set of signals $\mathcal{S}$ in the sense that each state $\alpha \in \mathcal{V}$ is a bitvector with one bit for each signal in $\mathcal{S}$. In addition, each signal transition $t \in \mathcal{T}$ is composed of a signal $s \in \mathcal{S}$ and a transition indicator of the set $\{+, -\}$, where s^+ designates the rising transition of s and s^- designates the falling transition of s.

A signal transition is said to be *enabled* in a state α if in state α the signal transition can occur. For example, s^+ is enabled in state α if in state α signal s can rise. For brevity we say s is enabled in state α if either s^+ or s^- is enabled in state α. When a signal transition occurs in some state, moving the circuit to another state, we say the signal transition (or, simply the signal) *fires*. When there exists a state α which contains two or more enabled signals that can fire in any order, we say that the corresponding transitions are *concurrent.*

A state transition $\alpha \rightarrow \beta$ exists if and only if in state α there is some signal transition t enabled which, when it fires, moves the circuit into state β. For example, if s is the only enabled signal in state α, then $\alpha \rightarrow \beta$ if and only if β differs from α in only the bit corresponding to signal s. We use the notation $\alpha \xrightarrow{s} \beta$ to denote that in state transition $\alpha \rightarrow \beta$ signal s changes.

When a signal s (or the corresponding signal transition) loses its chance to fire because of the firing of a concurrent signal transition, the signal (or the corresponding signal transition) is said to be *disabled*.

[1]This research was supported in part by ONR Grant no. N00014-89-J-3036, and by the Center for Integrated Systems, Stanford University.