SYNTHESIS FOR TESTABILITY TECHNIQUES FOR ASYNCHRONOUS CIRCUITS

by

Kurt Keutzer, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M91/77

16 September 1991

COVER PAGE

SYNTHESIS FOR TESTABILITY TECHNIQUES FOR ASYNCHRONOUS CIRCUITS

by

Kurt Keutzer, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M91/77

16 September 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

SYNTHESIS FOR TESTABILITY TECHNIQUES FOR ASYNCHRONOUS CIRCUITS

by

Kurt Keutzer, Luciano Lavagno, and
Alberto Sangiovanni-Vincentelli

Memorandum No. UCB/ERL M91/77

16 September 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Synthesis for Testability Techniques for Asynchronous Circuits*

Kurt Keutzer
Synopsys
Mountain View, CA

Luciano Lavagno
University of California
Berkeley, CA

Alberto Sangiovanni-Vincentelli
University of California
Berkeley, CA

Abstract

Our goal is to synthesize hazard-free asynchronous circuits that are testable in the very stringent hazard-free robust path-delay-fault model. From a synthesis perspective producing circuits satisfying two very stringent requirements, namely, hazard-free operation and hazard-free robust path-delay-fault-testability, poses an especially exciting challenge. In this paper we present techniques which *guarantee* both hazard-free operation and hazard-free robust path-delay-fault testability, at the expense of possibly adding test inputs, and give a set of heuristics which can improve hazard-free robust path-delay-fault testability without requiring such inputs. We also present a procedure that guarantees testability in the less stringent robust gate-delay-fault model.

1 Introduction

In this paper we are concerned with the problem of synthesizing asynchronous sequential circuits from a high level specification, the Signal Transition Graph (STG, [3]). In [13, 12] we presented a set of algorithms to solve this problem. They produce a circuit implementation that is guaranteed to be hazard-free if and only if a set of inequalities among path delays inside the circuit is satisfied. So using a suitable delay model during the synthesis process it is possible to guarantee hazard-freeness in the absence of delay faults. Now our goal is to test those path delays, and be sure that the above mentioned inequalities are satisfied in *each manufactured circuit.*

Moreover, as asynchronous interface circuits typically have absolute delay requirements, it is highly desirable to know that the synthesized circuit is able to operate with the required timing constraints. Also, because asynchronous circuits are often used as interfaces in systems where very high reliability is required, a stringent manufacture test of the interface circuitry is desirable. Unfortunately current testing procedures do not even reliably provide comprehensive stuck-at-fault testing of asynchronous circuits.

Our goal is to synthesize hazard-free asynchronous circuits that are testable in the hazard-free robust path-delay-fault model. Producing circuits satisfying two very stringent requirements, namely, hazard-free operation and hazard-free robust path-delay-fault-testability (*hfrpdft*), poses an especially exciting challenge. In this paper we present techniques which *guarantee* both hazard-free operation and hazard-free robust path-delay-fault testability, at the expense of possibly adding test inputs, and give a set of heuristics which can improve hazard-free robust path-delay-fault testability without requiring such inputs. We also present a procedure that guarantees testability in the less stringent robust gate-delay-fault model (*rgdft*).

The testing scenario that we envision uses *full scan* techniques to make the test generation process manageable. In this scenario every flip-flop (Set-Reset, C-element, ...) can be scanned, to increase both the observability of its inputs and the controllability of its output. This is also necessary because some of the flip-flop types, for example Set-Reset flip-flops, are sensitive only to input transitions in one direction, rising or falling[1]. So we *cannot* test the delay of the transitions in the other direction without resorting to scan techniques.

This paper is organized as follows: We present necessary terminology in Section 2. We show how an initial two-level circuit can be generated, and how hazards can be removed from it, in Section 3. We give a procedure that is guaranteed to turn this two-level circuit in a *hfrpdft* circuit in Section 4. We then give a variety of heuristics that are likely to increase the *hfrpdft* testability of a circuit in Section 5. We give another algorithm for producing a *rgdft* circuit in Section 6. Results on applying these techniques are given in Section 8.

*This work was partially supported by the National Science Foundation under Grant UCB-BS16421

[1]This is useful also to remove *dynamic* hazards from the circuit, see [13].

2 Definitions and Notation

Analysis and synthesis of hazard-free asynchronous circuits which are also delay-fault testable requires spanning the fields of logic optimization, testing and asynchronous synthesis. In this section we provide a minimal amount of terminology which will be used in the following sections.

2.1 Logic Synthesis

A *completely specified single-output logic function* g of n input variables is a mapping $g : \{0,1\}^n \rightarrow \{0,1\}$. Each input variable x_i corresponds to a coordinate of the domain of g. Each element of $\{0,1\}^n$ is called a *vertex*. The set of variables on which g depends[2] is called the *support* of g. The complement of g is the Boolean function g' such that $g(v) = 1 \Leftrightarrow g'(v) = 0$ and $g(v) = 0 \Leftrightarrow g'(v) = 1$ for all $v \in \{0,1\}^n$. An *incompletely specified single-output logic function* f of n input variables (called logic function in the following) is a mapping $f : \{0,1\}^n \rightarrow \{0,1,*\}$.

The set of vertices where f evaluates to 1 is called the *on-set* of f, the set of vertices where f evaluates to 0 is called its *off-set*, the set of vertices where f evaluates to $*$ is called its *dc-set*. Each vertex of the on-set of f is called a *minterm*.

A *literal* is either a variable or its complement. A *cube* c is a set of literals, such that if $a \in c$ then $\overline{a} \notin c$ and vice-versa. It is interpreted as the Boolean product of its elements. The cubes with n literals are in one-to-one correspondence with the vertices of $\{0,1\}^n$.

A cube c_1 covers another cube c_2, denoted $c_2 \sqsubseteq c_1$ if $c_1 \subseteq c_2$, for example $\{a, \overline{b}\} \subseteq \{a, \overline{b}, c\}$, so $a\overline{b}c \sqsubseteq a\overline{b}$. The covering is strict, denoted $c_2 \sqsubset c_1$, if $c_2 \neq c_1$.

The intersection of two cubes c_1 and c_2 is the empty cube if there exists x_i such that $x_i \in c_1$ and $\overline{x_i} \in c_2$ (or vice-versa), otherwise it is $c_3 = c_1 \cup c_2$. The Hamming distance between two cubes c_1 and c_2 is the number of variables x_i such that either $x_i \in c_1 \wedge \overline{x_i} \in c_2$ (or vice-versa).

A cube is called an *implicant* of a logic function f if it covers some minterm of f and it does not cover any off-set vertex of f. An implicant of f is called a *prime* if it is not covered by any other single implicant of f.

An *on-set cover* F of a logic function f is a set of cubes such that each cube of F is an implicant of f and each minterm of f is covered by at least one cube of F. By analogy we can define an off-set cover R of a logic function f as a set of cubes such that each cube of R covers only off-set vertices of f and each off-set vertex of f is covered by at least one cube of R.

A cube c in an on-set cover F of a logic function f can be *expanded* by removing literals from it while it remains an implicant of f. The result of the expansion is not unique (it depends on the removal order), but it is always a *prime implicant* of f.

In the following we shall use "cover" to denote on-set covers. Each cover F corresponds to a *unique completely specified* logic function, denoted by $B(F)$. On the other hand a logic function can have in general *many* covers (also called sum-of-products or two-level representations). A cover is interpreted as the Boolean sum of its elements, so it can also be seen as a two-level sum-of-products implementation of the completely specified function $B(F)$.

A cover F is called a *prime cover* of a function f if all its cubes are prime implicants of f. A cover F is called an *irredundant cover* of a function f if deleting any cube from F causes it to be no longer a cover of f (i.e. if some minterm is no longer covered by any cube of F). A *relatively essential vertex* or an *irredundancy test* of a cube c in a cover F is a vertex that is covered by c and is not covered by any other cube in F.

The *cofactor of a cube* c with respect to a literal x_i, denoted by c_{x_i}, is:

- the empty cube (a cube that does not cover any element of $\{0,1\}^n$) if $\overline{x_i} \in c$.
- $c - \{x_i\}$ otherwise.

The *cofactor of a cover* F with respect to a literal x_i, denoted by F_{x_i}, is the set of cubes of F cofactored against x_i. The empty cube can always be deleted from a cover, since it does not cover any vertex.

The cofactor has the following property (Shannon decomposition), for each $1 \leq i \leq n$: $B(F) = x_i \wedge B(F_{x_i}) \vee \overline{x_i} \wedge B(F_{\overline{x_i}})$.

A function f is *monotone increasing* in a variable x_i if $f(\overline{x_i}, \beta) = 1 \Rightarrow f(x_i, \beta) = 1$ for all $\beta \in \{0,1\}^{n-1}$, that is if "increasing" the value of x_i from 0 to 1 never "decreases" the value of f from 1 to 0. Similarly for *monotone*

[2] I.e. the set of x_i such that there are two vertices v_1 and v_2 that differ only in x_i for which $g(v_1) \neq g(v_2)$.

decreasing. A function f is *unate* in a variable x_i if it is either monotone increasing or monotone decreasing in x_i. Otherwise f is *binate* in x_i. Each *prime* cover of a unate function is also *irredundant*.

A *factored form* is recursively defined as

- a literal, or
- the sum of two factored forms, or
- the product of two factored forms.

A factored form F is obtained *algebraically* from a sum-of-products S representation of a function f if F is obtained from S by applying only the distributive and associative properties of Boolean sum and product (i.e. ignoring that $a \cdot \overline{a} = 0$, $a + \overline{a} = 1$ and so on).

2.2 Testing

A *combinational logic circuit* is represented as a labeled, directed, acyclic graph (*dag*) $G = (V, E)$ with each vertex v labeled with the name of a primitive gate such as *and*, *or* or *not*, or with the name of a primary input or output. There is an edge $< u,\ v >$ in V between two vertices if the output of the gate associated with u is an input to gate v. The members of V that have no fan-in are the only vertices that may be labeled with the name of a primary input. The members of V that have no fan-out are the only vertices that may be labeled with the name of a primary output.

If the output of a gate, g_1, is connected to an input of gate, g_2, g_1 is a *fanin* of g_2. Gate g_2 is a *fanout* of gate g_1.

A *two-level circuit* is a circuit directly implementing a *cover* by implementing each product term by an *and* gate, the sum term by an *or* gate adding inverters on inputs as needed.

An *asynchronous sequential circuit* is implemented, according to the techniques described in [13], using combinational logic, asynchronous flip-flops (Set-Reset, C-elements, ...) and feedback wires. For the purposes of this paper, we will assume that

1. each flip-flop is implemented so that it has a testing mode, in which an appropriate value can be scanned in and out [8].
2. each designer-specified signal that is not implemented with a flip-flop is either a primary input or a primary output of the integrated circuit or is otherwise made accessible (for example by insertion of a scan D-latch normally held in transparent mode).

If these assumptions are satisfied, then every cycle in the circuit is broken by at least one scan memory element. So we shall consider flip-flop inputs as primary outputs, and flip-flop outputs as primary inputs of the combinational logic circuit that we are testing.

2.3 Delay Testing

A *path* in a combinational circuit is an alternating sequence of vertices and edges, $\{v_0, e_0, ..., v_n, e_n, v_{n+1}\}$, where edge e_i, $1 \leq i \leq n$, connects the output of vertex v_i to an input of vertex v_{i+1}. For $1 \leq i \leq n$, v_i is a gate; v_0 is a primary input and v_{n+1} is a primary output. Each e_i is a net.

An *event* is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Consider a sequence of events, $\{r_0,\ r_1,\ ...,\ r_n\}$ occurring at gates $\{g_0,\ g_1,\ ...,\ g_n\}$ along a path, such that r_i occurs as a result of event r_{i-1}. The event r_0 is said to propagate along the path. If there exists a vector pair such that under *appropriate* delays in the circuit, an event could propagate along a path, then the path is said to be *event sensitizable*. If there exists an input vector pair such that under *arbitrary* delays in the circuit, an event propagates along a path, then the path is said to be *robustly event sensitizable*.

A circuit has a *gate-delay-fault* iff there is one gate in the circuit such that the output of the circuit is slow to make a $0 \rightarrow 1$ (or $1 \rightarrow 0$) even when one or more of the gate's inputs change values. Each single gate-delay-fault is assumed to be so catastrophic as to cause a delay along any path through the gate to any output.

A circuit has a *path-delay-fault* iff there exists a path from a primary input to a primary output via a set of gates and nets such that a primary input event is slow to propagate *along the path* to the primary output.

In this paper, we will be dealing with the hazard-free robust path-delay-fault (*hfrpdft*) and robust gate-delay-fault (*rgdft*) models. A two-pattern test $T\ = <v_1,\ v_2>$ is said to be a *robust* delay test for a path π, if and only if, when π is faulty and the test T is applied, the circuit output is different from the expected output at sampling time, independent of the delays along gate input leads not on π [14]. A more stringent model is the *hazard-free robust* delay fault model, treated in [16, 4]. [3] A robust test is said to be a hazard-free robust test if no hazards can occur on the tested path during the application of the test, regardless of gate delay values.

This delay-testing model implies that the scan flip-flops (see Section 2.2) must have the following capabilities:

- select which of the flip-flop inputs (e.g. S or R for a Set-Reset flip-flop) is latched by the test clock and subsequently scanned out.
- memorize two values to be applied in sequence for the application of a two-pattern test.

2.4 Asynchronous Circuit Specification

The *Signal Transition Graph* (STG) was introduced by [2] as a specification formalism for asynchronous sequential circuits. It is an *interpreted free-choice Petri net*: transitions of the FC net are *interpreted* as value changes on input/output signals of the specified circuit.

Two transitions are said to be *concurrent* if there exists some net marking where both are enabled. In this case they can fire in any order. Otherwise they are said to be *ordered*.

An STG is *live* if it is strongly connected and for each signal t all its transitions are ordered and alternating ($t^+ \rightarrow t^- \rightarrow t^+ \ldots$).

In [9] was proved that a live STG can be decomposed into Finite State machine (FSM) components that cover the net (each component is sequential and exhibits non-deterministic choice).

The State Graph (SG, also called reachability graph) is a directed graph, where each node (henceforth called *state*) is in one-to-one correspondence with a live and safe marking of the signal transition graph. An edge joins state s_1 with state s_2 if the corresponding marking M_2 can be reached from M_1 (corresponding to s_1) through the firing of a *single transition*. This transition labels the edge.

An STG has the Unique State Coding (USC) property if there exists an assignment of a *unique* vector v_i of signal values to each state s_i of the SG, such that for each edge $s_1 \rightarrow s_2$:

1. if it is labeled t^+ then signal t must be 0 in v_1 and 1 in v_2.
2. if it is labeled t^- then signal t must be 1 in v_1 and 0 in v_2.
3. otherwise signal t must have the same value in both v_1 and v_2.

A *static* hazard is a $0 \rightarrow 1 \rightarrow 0$ transition (static 0-hazard) or $1 \rightarrow 0 \rightarrow 1$ transition (static 1-hazard) on an output signal of a circuit, in any condition where no transition for that signal should be enabled according to the specification.

A *dynamic* hazard is a $0 \rightarrow 1 \rightarrow 0 \rightarrow 1$ (or $1 \rightarrow 0 \rightarrow 1 \rightarrow 0$) transition on an output signal in any condition where a single positive (or negative) transition for that signal is enabled according to the specification.

2.5 Hazard-free Operation and HFRPDFT

It may be useful to contrast the requirements of hazard-free operation and *hfrpdft*.

For the operation of an asynchronous circuit to be hazard-free it is necessary that for*for all legal* input sequences, i.e. corresponding to valid firing sequences of the STG specification, no static hazard occurs in the circuit. Thus hazard-free operation is a global property governing the normal operation of the circuit.

For a circuit to be *hfrpdft* it is necessary that *there exists* a vector pair that detects each path-delay-fault in a robust and hazard-free manner. Such a vector pair (v_1, v_2) might not be a *legal* input sequence, i.e. there might not exist a valid firing sequence of the STG specification bringing the inputs of the circuit from v_1 to v_2. This means also that it might be impossible to apply v_1 and v_2 without using scan techniques, because the circuitry surrounding the path under test (also called the *environment*), which was designed to implement the STG specification, might not be able to produce those vectors in that sequence.

Despite the apparent similarity between the two properties, neither property implies the other. Moreover, optimizing a circuit for one property can diminish or eliminate the other. See for example the case, described in Section 4, where making a circuit *hfrpdft* introduces hazards during normal operation. Conversely, in the very same example, a cube that appears in the initial two-level cover in order to eliminate a hazard during operation makes the circuit not *hfrpdft*.

[3] The hazard free robust path-delay-fault model is simply called the robust path-delay-fault model in [5, 4].

3 Hazard-free Asynchronous Circuit Synthesis

3.1 Synthesizing Initial Two-Level Circuit

The following procedures, described in more depth in [13], derive an on-set cover F and an off-set cover R for the next-state function f of signal t_i, receiving as input a live STG, S, having the USC property, with initial marking m. Let v and w be vectors of values for the n signals that appear in S, $v, w \in \{0,1\}^n$, and let v^j denote the value of signal t_j in v.

```
Generate_covers ($S$, $i$, $m$)
1. for each signal $t_j$ in the STG, do (determine its initial value):
    (a) let $M_j$ be an FSM component of $S$ that contains all transitions for $t_j$, and let $m_j$ be its initial marking (a subset of $m$).
    (b) find on $M_j$ the first transition $t_j^*$ that can be reached from $m_j$.
    (c) if $t_j^*$ is $t_j^+$, then let $v^j = 0$, otherwise let $v^j = 1$.
2. let $F = \phi$, $R = \phi$.
3. Generate_covers_recur ($S$, $i$, $F$, $R$, $m$, $v$)
4. expand $F$ and $R$ to prime covers, and remove duplicate cubes.
```

```
Generate_covers_recur ($S$, $i$, $F$, $R$, $m$, $v$)
1. if $t_i^+$ is enabled in $m$ then let $v^i = 1$.
2. else if $t_i^-$ is enabled in $m$ then let $v^i = 0$.
3. for each maximal subset $T$ of transitions enabled in marking $m$ such that $t_i^*$ is *not enabled* in the marking $m'$ obtained from $m$ firing all transitions in $T$ do:
    (a) let $c = \{v^j \ s.t. \ t_j^* \notin T\}$.
    (b) if $v^i = 1$ then let $F = F \cup \{c\}$, otherwise let $R = R \cup \{c\}$.
4. for each transition $t_j^*$ enabled in $m$ such that marking $m'$, obtained from $m$ firing $t_j^*$, has not been reached yet, do:
    (a) let $w = v$.
    (b) if $t_j^*$ is $t_j^+$, then let $w^j = 1$, otherwise let $w^j = 0$.
    (c) Generate_covers_recur ($S$, $i$, $F$, $R$, $m'$, $w$)
```

If the next-state function f for signal t depends on t itself, then we initially implement t with a set-reset flip-flop and a pair of two-level circuits,

- one for the *set* input, including all cubes in F that depend on t, cofactored against t,
- one for the *reset* input, including all cubes in F that do not depend on t (plus one *not* gate to invert it).

If f does not depend on t, then our initial circuit is directly the on-set cover of f.

Notice that these initial two-level implementations are guaranteed to be *prime*, but *not irredundant*, and the redundancies cannot be removed without introducing hazards in the operation of the circuit (see, for example, [11]).

3.2 Removing Hazards from the Initial Implementation

The following procedure, described in more depth in [12], detects and removes all possible hazards from a circuit implementation I of the next-state function f of each signal, using information from the State Graph associated with the STG specification.

Let F be the initial two-level on-set cover of f, as obtained in Section 3.1, and let I be a circuit derived from F using only distributivity and associativity[4]. Let R' be the set of all the prime implicants of the complement of $B(F)$, and let F' be the set of all the prime implicants of $B(F)$. Let v_i be a vertex in the domain of f, corresponding to the label of some SG state. The transition cube C associated with a pair of vectors (v_1, v_2) is defined as the cube obtained removing from v_1 all signals that change value going from v_1 to v_2 on the SG. E.g. if we have $v_1 = xyz$, $v_2 = x\overline{y}z$ and signals y and z have a transition between those two SG states, then $C = x$. Let $d(c_1, c_2)$ denote the Hamming distance between cubes c_1 and c_2.

```
Remove_hazards ($f$, $F$, $R$, $F'$, $R'$, $I$)
1. for each vector pair $(v_1, v_2)$ such that
     • $d(v_1, v_2)$ is maximal and
     • $f(v_1) = f(v_2) = f(v_j)$ for all $v_j$ on an SG path from $v_1$ to $v_2$:
   (a) let $C$ be the extended transition cube associated with $(v_1, v_2)$.
   (b) if $f(v_1) = f(v_2) = 1$ then:
       i. for each cube $c_0 \in R'$ intersecting $C$, for each pair of distinct implicants
          $c_1, c_2 \in F \cap C$ such that
          $d(c_0, v_1) = d(c_1, v_1) + 1$, $d(c_1, c_0) = 1$,
          $d(c_0, v_2) = d(c_2, v_2) + 1$ and $d(c_2, c_0) = 1$:
          A. let $t_1^*$ be the transition moving from $c_1$ to $c_0$. Let $t_2^*$ be the transition moving from $c_0$ to $c_2$.
          B. let $d_1$ be a lower bound on the delay along the path in $I$[5] from input $t_1$ to $t$ corresponding to cube
             $c_1$ for transition $t_1^*$.
          C. let $d_2$ be an upper bound on the delay along the path in $I$ from input $t_2$ to $t$ corresponding to cube
             $c_2$, for transition $t_2^*$.
          D. let $d_3$ be a lower bound on the delay between transition $t_2^*$ and $t_1^*$[6].
          E. if $(d_2 - d_1) > d_3$ then a hazard condition exists.
             Then increase $d_3$, e.g. by adding non-inverting buffers, so that $(d_2 - d_1) < d_3$.
   (c) else ($f(v_1) = f(v_2) = 0$):
       i. do the same, exchanging $F \leftrightarrow R$ and $R' \leftrightarrow F'$.
```

4 A Procedure Guaranteed to Generate a HFRPDFT Circuit

This Section describes a technique to implement the two-level initial on-set cover obtained in Section 3.1 as a fully hazard-free robustly path-delay-fault testable circuit that has exactly the same hazard properties as the initial two-level cover, so that the algorithm described in Section 3.2 can be used to make it hazard-free in operation and fully *hfrpdft*.

[10] first gave a procedure, based on Shannon decomposition, to make a combinational circuit robustly path-delay-fault testable. The essence of this procedure is to choose a binate variable, call it x, in a given sum-of-products representation, call it S, of a Boolean function f, decompose S into $x \cdot S_x + \overline{x} \cdot S_{\overline{x}}$, such that the variable x does not appear in either S_x and $S_{\overline{x}}$. Unfortunately, while this procedure results in a hazard-free robust path-delay-fault testable implementation it may not result in an implementation that is hazard-free during normal operation, as is required.

[4]We limit ourselves to such operations because, as shown in [13], I then has exactly the same hazards as F.
[5]There exists only one such path if I is obtained using only distributivity and associativity.
[6]Obtained from a delay analysis of the circuit implementing signal t_2, or assumed to be 0.

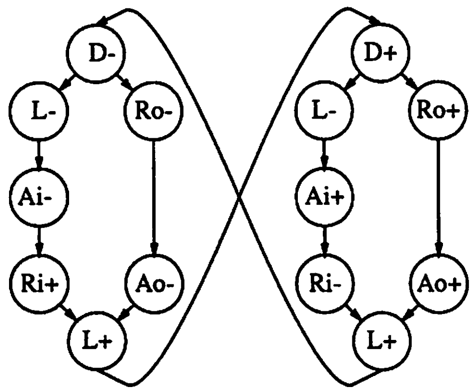

Figure 1: STG with a non-*hfrpdft* next-state function.

Take for example the STG reported in Figure 1 (taken from [3]). The on-set and off-set covers for signal A_i, as obtained in Section 3.1, are respectively $F_{A_i} = D\overline{L} + D\overline{R_i} + L\overline{R_i}$ and $R_{A_i} = \overline{D}\,\overline{L} + \overline{D}R_i + LR_i$. Both covers are redundant (cube $D\overline{R_i}$ can be removed from F_{A_i} and cube $\overline{D}R_i$ can be removed from R_{A_i}). If we choose any variable for the above decomposition, we introduce a hazard whenever that variable changes and A_i must remain constant. Notice that in this case there is also no hope to obtain a *hfrpdft* implementation with any of the heuristics described in Section 5, which cannot remove a redundancy in both the on-set and off-set covers.

We now present a procedure which is guaranteed to produce an *hfrpdft* circuit. This procedure may require the addition of test inputs. At present we know of no procedure that is guaranteed to produce *hfrpdft* circuits, that are also hazard-free in operation, that does not also add test inputs.

4.1 Algebraic Decomposition

Our procedure is similar to the one outlined above in some respects: first a variable at the beginning of an untestable path is identified, call it x, in a given sum-of-products representation, call it F, of a Boolean function f. F is then algebraically decomposed into $x \cdot G + \overline{x} \cdot H + R$, so that the variable x does not appear in any one of G, H and R. The difference between this procedure and that of [10] is the ability to partition out a remainder R. This results in a more area efficient implementation but more importantly the factoring out of the remainder ensures hazard-free operation as we will show in Section 4.2 below. We now give the procedure in detail. It takes as input a *prime*, but possibly not irredundant, two-level representation F of a Boolean function f, and it returns a multi-level implementation of f with exactly the same hazard properties of F:

Make_HFRPDFT(F)

1. If the combinational circuit F is hazard-free robustly-path-delay-fault testable, then return F.
2. otherwise:
   (a) choose (with an appropriate heuristic) an input variable x such that a path beginning from x is untestable in F.
      - Let G be the two-level cover obtained by collecting all the cubes in F that depend on x, cofactored against x.
      - Let H be the two-level cover obtained by collecting all the cubes in F that depend on $\overline{x}$, cofactored against $\overline{x}$.
      - Let R be the two-level cover obtained by collecting all the cubes in F that do not depend on either x or $\overline{x}$.

   (b) Let t_1 and t_2 be two new variables, not in the support of f.

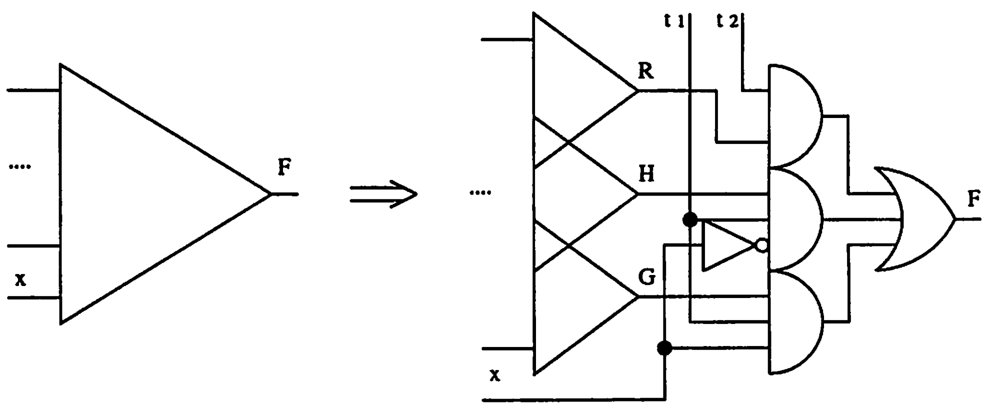

Figure 2: Application of Make_HFRPDFT.

(c) Return the circuit $t_1 \cdot x \cdot$ Make_HFRPDFT(G) + $t_1 \cdot \overline{x} \cdot$ Make_HFRPDFT(H) + $t_2 \cdot$ Make_HFPDFT(R)

Figure 2 shows the result of one step of Make_HFRPDFT(F).
We now proceed to prove that this procedure results in a *hfrpdft* circuit. The key result is the following:

Theorem 4.1 *Let f be a Boolean function, let F be a combinational circuit implementing f and let $G \cdot x + H \cdot \overline{x} + R$ be an algebraic factorization of F such that*

1. *G, H and R are each individually* hfrpdft *circuits*
2. *The on-set of the Boolean function implemented by G contains a vertex not contained in the on-set of the Boolean function implemented by H and*
3. *The on-set of the Boolean function implemented by H contains a vertex not contained in the on-set of the Boolean function implemented by G and*
4. *t_1 and t_2 are two variables not in the support of f*

Then $F' = t_1 \cdot G \cdot x + t_1 \cdot H \cdot \overline{x} + t_2 \cdot R$ is a hfrpdft *circuit.*

Proof: We assume that G, H and R are non-empty. If they are then the analysis is further simplified.
Each of G, H, and R are assumed to be *hfrpdft*. The proof strategy used to detect delay faults in these sub-circuits in F' is to augment the vectors which test those paths in isolation with the appropriate values of t_1, t_2, and x.
Consider a path π in G. By supposition G in isolation is *hfrpdft* so we have a vector pair $< v_1, v_2 >$ that tests π. To test π in F' set $t_1 = 1, t_2 = 0, x = 1$ and apply $< v_1, v_2 >$.
Consider a path π in H. By supposition H in isolation is *hfrpdft* so we have a vector pair $< v_1, v_2 >$ that tests π. To test π in F' set $t_1 = 1, t_2 = 0, x = 0$ and apply $< v_1, v_2 >$.
Consider a path π in R. By supposition R in isolation is *hfrpdft* so we have a vector pair $< v_1, v_2 >$ that tests π. To test π in F' set $t_1 = 0, t_2 = 1, x = 0$ and apply $< v_1, v_2 >$.
The paths associated with x and $\overline{x}$ are equally straightforward. Consider the path π associated with x. By supposition the on-set of G contains a vertex not contained in H. Call that vertex v. To test π set $t_1 = 1, t_2 = 0$, give the values of vertex v to the other primary inputs and let x rise $0 \rightarrow 1$.
Consider the path π associated with $\overline{x}$. By supposition the on-set of H contains a vertex not contained in G. Call that vertex v. To test π set $t_1 = 1, t_2 = 0$, give the values of vertex v to the other primary inputs and let x fall $1 \rightarrow 0$.
The testing inputs do not need to run at speed but it may be simpler to test them than to treat them as a special case. The vectors are created as follows.
Consider the path π associated with the testing input t_1 in $x \cdot t_1 \cdot G$. By supposition the on-set of G contains a vertex not contained in H. Call that vertex v. To test π set $t_2 = 0, x = 1$, give the values of vertex v to the other primary inputs and let t_1 rise $0 \rightarrow 1$.
Consider the path π associated with the testing input t_1 in $\overline{x} \cdot t_1 \cdot H$. By supposition the on-set of H contains a vertex not contained in G. Call that vertex v. To test π set $t_2 = 0, x = 0$, give the values of vertex v to the other primary inputs and let t_1 rise $0 \rightarrow 1$.

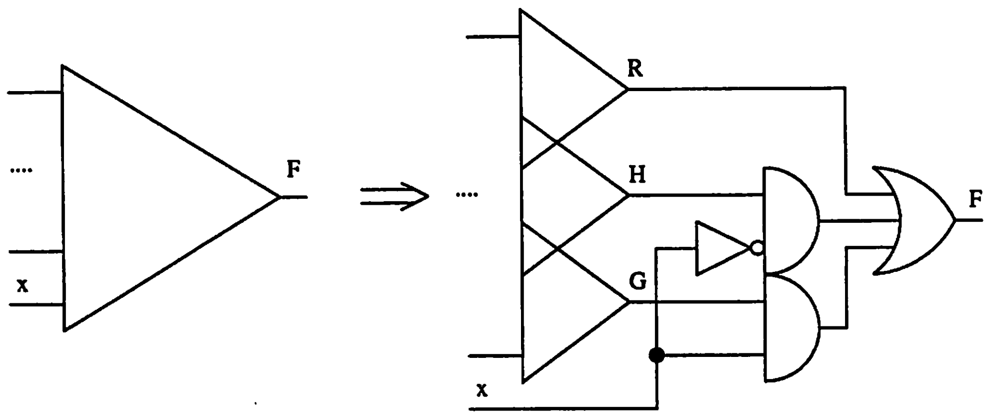

Figure 3: Application of Make_HFRPDFT without test inputs.

Consider the path π associated with the testing input t_2. To test π first find any vertex v such that $R = 1$. Then set $t_1 = 0$, give the values of vertex v to the other primary inputs, and let t_2 rise $0 \rightarrow 1$.

Thus every path in F' is hazard-free robust path-delay-fault testable. ■

Notice also that a similar argument allows us to conclude that if both $G + R$ and $H + R$ are *hfrpdft* circuits, then we do not have to introduce the testing inputs t_1 and t_2, resulting in the circuit structure described in Figure 3.

We prove that we retain primality in the recursion step of Make_HRFPDFT in the following Lemma.

Lemma 4.1 *Let F be a prime on-set cover of a Boolean function f. Let x be a variable in the support of f. Let G be the set of cubes containing x, H be the set of cubes containing $\overline{x}$ and R be the set of cubes containing neither x or $\overline{x}$. Let G_x be the cofactor of G with respect to x and let $H_{\overline{x}}$ be the cofactor of H with respect to $\overline{x}$.*

Then each of G_x, $H_{\overline{x}}$ and R is a prime cover.

Proof: Suppose some cube c in G_x (with corresponding cube $d = x \cdot c$ in F) is not prime. Then we could remove a literal, say y, from it, and the resulting cube, call it c_y, would still be an implicant of f. So also $c' = c_y \cdot \overline{y}$ would be an implicant of g, and $c'' = c' \cdot x$ would be an implicant of f. Then we could expand d to d_y, contradicting the hypothesis of primality of F.

A similar argument can be used for H. As R is unchanged from F, R remains prime. ■

To prove procedure Make_HFRPDFT is correct, we use induction and apply Theorem 4.1 at the induction step.

Theorem 4.2 *Let F be a prime on-set cover of a Boolean function f. Then procedure Make_HFRPDFT results in a* hfrpdft *combinational circuit F' of the form $t_1 \cdot x \cdot G' + t_1 \cdot \overline{x} \cdot H' + t_2 \cdot R'$ implementing f.*

Proof:

- Basis:

  Let f be a Boolean function of two variables. By inspection we can show that for any function of two variables there exists a *hfrpdft* implementation using Make_HFRPDFT.

- Induction step:

  Suppose that any function of n variables can be made *hfrpdft* by the procedure Make_HFRPDFT. Let f be a function of $n + 1$ variables. Suppose F is not *hfrpdft*. Let x be a variable such that a path beginning at x is not testable in F.

  By Lemma 4.1, each of G, H and R (supposed not empty, as in the proof of Theorem 4.1) is a prime cover of a function of n variables, and therefore, by the induction hypothesis, each has a *hfrpdft* implementation that can be arrived at through Make_HFRPDFT. Let us call these *hfrpdft* implementations G', H' and R' respectively.

  Now suppose that H' is not empty, and that it contains no vertex not contained in G'. Then, also H is not empty and it contains no vertex not contained in G. Let d be any implicant of F that is assigned to H, and let $c = d_{\overline{x}}$. Since c contains no vertex that is not contained in G, then we could have removed x from d leaving it an implicant of f. But this contradicts the primality of F.

By a similar argument we can show that G' contains some vertex not contained in H'.

We have now shown that:

1. G', H' and R' are each individually *hfrpdft* circuits
2. The on-set of G' contains a vertex not contained in H' and
3. The on-set of H' contains a vertex not contained in G'.

Let t_1 and t_2 be two variables not in the support of f. Then by Theorem 4.1 $F' = t_1 \cdot x \cdot G' + t_1 \cdot \overline{x} \cdot H' + t_2 \cdot R'$ is a *hfrpdft* circuit.

Thus procedure Make_HFRPDFT produces a *hfrpdft* circuit using additional test inputs. ■

4.2 Retaining Hazard Free Operation

We wish to show that the techniques used to make the circuit hazard-free robustly path delay fault testable do not destroy the possibility to make the circuit hazard-free in operation using the algorithm described in Section 3.2

To do this we will find it useful to apply the following Theorem, of [15]:

Theorem 4.3 *Let T be a two-level representation of a logic function f. Let M be a multi-level representation of f such that it can be obtained from T using only the associative, distributive and De Morgan laws. Then the circuits corresponding to M and T have precisely the same static hazards.*

That is for each pair of input vectors such that the output of M had a hazard for some assignment of wire delays, there exists some wire delay assignment in T such that the same hazard happens at its output, and vice-versa. On the other hand if some hazard could not happen in M under any wire delay assignment, then it could not happen also in S, and vice-versa.

The relevance of this Theorem is that, if we have a two-level implementation T of a logic function f that does not exhibit hazards for some class of input changes, then we can use the transformations listed above, to obtain a multi-level implementation of f that has the same hazard properties.

To see that the circuit F' has the same hazards as F, observe that the inputs t_1 and t_2 do not change during normal operation, therefore their introduction does not create any hazards. Thus in normal operation F' operates as $G \cdot x + H \cdot \overline{x} + R$. The factorization of the initial circuit F into $G \cdot x + H \cdot \overline{x} + R$ can be accomplished simply using associativity. Thus no hazards are introduced in this decomposition and the resulting circuit F' can be made hazard-free in operation.

5 Heuristic Procedures to Improve HFRPDF Testability

In this Section we present a number of heuristics which, while not guaranteeing to produce a *hfrpdft* implementation of a circuit, may be expected to improve the hazard-free robust path-delay-fault testability of a given circuit, without introducing new hazards in its operation.

5.1 Algebraic Factorization

In addition to being a useful technique for circuit optimization, algebraic factorization can be used as a heuristic for improving the delay-fault-testability of a circuit. While a rigorous proof of the utility of this technique would require the introduction of the notion of *ENF Reducibility* from [1, 6], the basic principle in operation is simply demonstrated. Given a cover $F \cdot G + F \cdot H$, if the paths associated with F were *hfrpdft* either in G *or* in H (but not necessarily *hfrpdft* in both), then if the cover is algebraically factored to produce $F \cdot (G + H)$ then F becomes *fully hfrpdft*. This is due to the collapsing of paths that is a natural by-product of algebraic factorization. As a simple illustration of this consider the following example drawn from [6]: $C = a\overline{b} + \overline{b}c + b\overline{c}$ and its algebraic factorization $M = (a + c)\overline{b} + b\overline{c}$. The path associated with literal $\overline{b}$ in cube $a\overline{b}$ of C, call it π_1, is not *hfrpdft*, but the path associated with $\overline{b}$ in cube $\overline{b}c$ of C, call it π_2, is *hfrpdft*. After C is factored into M there is a many-to-one reduction from paths π_1 and π_2 to a single path π associated with literal $\overline{b}$ in the factor $(a + c)\overline{b}$, and the testability of π_2 alone is sufficient to ensure the testability of π. As a result M is completely *hfrpdft* while C is not.

As algebraic factorization is essentially iterative applications of the associative law, it retains the hazard properties of the initial two-level implementation, so that again the algorithm of Section 3.2 can be applied to make the circuit hazard-free in operation.

5.2 Complementation

When the on-set cover of the next-state function of a signal is not *hfrpdft*, then it may be the case that its companion off-set cover (as obtained in Section 3.1) is more easily made *hfrpdft*. Consider the function f implemented in the circuit $F = \overline{a}\,\overline{c} + a\overline{b} + \overline{a}b + c\overline{d} + \overline{c}d$. While this is one of a few prime and irredundant implementations of this function, the paths associated with $\overline{a}$ and $\overline{c}$ in $\overline{a}\overline{c}$ are not *hfrpdft*. Furthermore all prime and irredundant implementations of this function share this problem. Algebraically factoring out $\overline{a}$ produces the circuit $\overline{a}(\overline{c} + b) + a\overline{b}c + \overline{d} + \overline{c}d$. While the path associated with $\overline{a}$ has become *hfrpdft*, the path associated with $\overline{c}$ in $(\overline{c} + b)$ is still not *hfrpdft*. Furthermore, another application of algebraic factorization will not make this path *hfrpdft*. Thus algebraic factorization alone cannot be used to make this circuit *hfrpdft*.

An alternative way of getting a fully *hfrpdft* implementation of f is to implement an *off-set* cover of f, rather than an on-set cover, and complement the output. The off-set cover R of f, as obtained in Section 3.1, is $R = \overline{a}\overline{b}cd + ab\overline{c}\overline{d} + abcd$. This implementation is *hfrpdft*.

Another case in which this can be useful is when the on-set cover of f obtained by the algorithm in Section 3.1 is redundant, while the corresponding off-set cover is not.

Having implemented R, we can apply Lemma 4.7 of [15] that states that introducing an inverter at the output of a circuit can does not introduce or remove hazards. Thus complementing the output of R to produce f retains the hazard properties of R, and the resulting circuit can be made hazard-free.

6 A Procedure Guaranteed to Generate a RGDFT Circuit

In Section 4 we presented an algorithm that is guaranteed to produce a hazard-free robustly *path-delay-fault* testable implementation of a circuit. This algorithm has the potential disadvantage that test inputs may be required. In this Section we present a technique that also requires the introduction of test inputs but may require fewer test inputs, producing an implementation that is robustly *gate-delay-fault* testable. This delay-fault testability model is less stringent than *hfrpdft*, but it can still be sufficient to determine with the desired accuracy whether the delays in the manufactured circuit lie within the bounds assumed during synthesis, thus ensuring hazard-free operation.

6.1 Making the Circuit Robustly Gate Delay Fault Testable

The initial two-level cover produced by the techniques in Section 3.1 is prime but may not be irredundant. The first step required is to make each cube *irredundant* through the introduction of test inputs. For each redundant cube in the cover we add one test input.

Starting now with an irredundant two-level cover we proceed to make the cover *rgdft* through the introduction of a single test input. This procedure is modeled after the procedure given in [17] to improve stuck-open fault testability. To motivate this procedure we will employ the following Lemma from [4, 7].

Lemma 6.1 *: Let C be a two-level single-output circuit and let g be a gate in C. If a path π through g is hazard-free robustly path-delay-fault testable then g is robustly gate-delay-fault testable.*

It is likely that after applying the heuristic transformations described in Section 5 that *at least* one path per gate is *hfrpdft*. If this is not the case then the following procedure will make the circuit *rgdft*.

Make_RGDFT(C)

1. If the prime and irredundant two-level combinational logic circuit C is robustly gate-delay-fault testable, then return C.

2. otherwise:

(a) Let $Q = \{q_1, q_2, \ldots, q_n\}$ be the set of cubes in C that are not *rgdft*.

(b) Let t be an input not in the support of C.

(c) Add t to each cube in Q. For example $q_i' = t \cdot q_i$.

(d) Let $Q' = \{q' | q \in Q\}$.

(e) Let $C' = C - Q \cup Q'$

3. Return C'

We now proceed to prove that the resulting circuit C' is *rgdft*. Let g be a gate in C' associated with cube q. Suppose q is not in Q'. Then by supposition q was already *rgdft*, and the introduction of t does not change this property.

Suppose q' is in Q'. Both the circuit C' and C are prime and irredundant. Let v be a relatively essential vertex of $q \in Q$. Consider $v' = t \cdot v$. Clearly, q' covers $t \cdot v$. Furthermore no other cube in C' covers v' otherwise v would not be a relatively essential vertex of q. So v' is a relatively essential vertex of q'. Consider the vector sequence $< \bar{t} \cdot v, t \cdot v >$. The vector $v' = t \cdot v$ is a relatively essential vertex of q' and it must be the case that the vector $\bar{t} \cdot v$ is in the off-set of C'. Therefore by the necessary and sufficient conditions for *hfrpdft* in two-level networks given in [7, 4] the path associated with t in q', call it π is *hfrpdft*. Now by Lemma 6.1 given above, because π is *hfrpdft* then the gate g associated with cube q' must also be *rgdft*. Furthermore, because π is *hfrpdft*, the output *or* gate of C' must also be *rgdft*, by the reasoning of the same Lemma. Thus the procedure immediately above creates a two-level circuit that is completely *rgdft*. To create a *rgdft* multilevel circuit from this two-level circuit then the constrained factorization techniques of [5, 6] may be applied.

To see that the circuit C' can be made hazard-free, observe that the input t does not change during normal operation, therefore its introduction does not create any hazards. Thus in normal operation C' has exactly the same hazard properties as C.

7 Summary

So far we have described the effective procedures of Sections 4 and 6 and the heuristic techniques of Section 5 as independent procedures, but in fact the most effective use of these techniques involves their integrated application. If the initial two-level circuit produced in Section 3.1 is *hfrpdft*, then algebraic factorization can be freely applied and *hfrpdft* will be retained.

If the initial two-level circuit is not *hfrpdft*, then the best course is to iteratively apply the heuristics of Section 5. If we reach a point at which further applications of algebraic factorization alone will not improve the testability of the circuit, then we have two courses.

1. We can examine the *rgdft* of the current circuit. If the fault coverage in this model is high (as would be expected) and this fault model is acceptable, then we can terminate at this point. We can also achieve complete *rgdft* using the techniques of Section 6.

2. Alternatively, if complete fault coverage in the hazard-free robust path-delay-fault model is desired, then this can be achieved at this point using the techniques of Section 4.

Through the integrated application of these techniques we aim to achieve the desired fault coverage with the least penalty in area overhead and the fewest additional test inputs.

At this point, we have obtained an *hfrpdft* (or *rgdft*) implementation of the STG specification that has exactly the same hazard properties as the initial two-level cover. So we can now apply the techniques described in Section 3.2 in order to obtain a circuit that is hazard-free in operation. These techniques simply change the delays of some signals, so both *hfrpdft* and *rgdft* are maintained in the final hazard-free implementation.

8 Results

In this section we present our experimental results on a number of examples, taken both from industry and academia.

Table 1 gives the area and delay (using a standard cell implementation) of:

- a hazard-free, but not completely testable, optimized implementation (column "Untestable").

example	Untestable		Testable			
	Area	Delay	Area	Delay	Decomp.	Test inp.
chu133	320	5.4	320	5.4	0	0
chu150	208	5.2	264	5.6	1	2
chu172	192	5.2	192	5.2	0	0
converta	432	9.2	432	9.2	0	0
ebergen	304	5.2	304	5.2	0	0
full	224	4.0	224	4.0	0	0
hazard	264	6.4	264	6.4	0	0
hybridf	304	4.8	304	4.6	0	0
nowick	232	4.2	232	4.2	0	0
alloc-outbound	336	5.4	336	5.4	0	0
mp-forward-pkt	304	5.6	304	5.6	0	0
nak-pa	320	5.6	320	5.6	0	0
pe-rcv-ifc	1000	8.4	1000	8.4	0	0
pe-send-ifc	1160	10.8	1160	10.8	0	0
ram-read-sbuf	440	7.0	440	7.0	0	0
rcv-setup	128	2.8	128	2.8	0	0
sbuf-ram-write	456	9.2	456	9.2	0	0
sbuf-read-ctl	320	5.2	320	5.2	0	0
sbuf-send-ctl	344	5.6	360	5.4	0	0
sbuf-send-pkt2	352	5.2	352	5.2	0	0
sendr-done	128	3.8	128	3.8	0	0
qr42	304	5.2	304	5.2	0	0
rpdft	176	4.0	256	6.6	2	3
vbe10b	808	6.8	808	6.8	0	0
vbe5b	392	7.6	352	8.8	0	0
vbe5c	192	3.8	192	3.8	0	0
wrdatab	744	8.2	744	8.2	0	0
total	10384	159.8	10496	163.6	3	5

Table 1: Experimental results

- a hazard-free *hfrpdft* optimized implementation (column "Testable").

The area is the total area, excluding routing, of each circuit. The delay is the maximum *combinational logic* delay, obtained with static timing analysis.

The column labeled "Decomp." gives the number of times procedure Make_HFRPDFT had to decompose the circuit in order to make it *hfrpdft*, while the column labeled "Test inp." gives the number of added test inputs for each circuit.

9 Conclusion

In this paper we are concerned with the problem of synthesizing asynchronous sequential circuits from a high level specification, the Signal Transition Graph (STG, [3]). We have demonstrated both heuristic techniques and effective procedures to synthesize hazard-free asynchronous circuits that are testable in the very stringent hazard-free robust path-delay-fault model.

There are three principal obstacles to comprehensive path-delay-fault testing:

1. the synthesis of fully *hfrpdft* sub-circuits;
2. the inability to achieve comprehensive path coverage because of an unmanageable number of paths; and

3. the inability to apply the vector pairs due to limited controllability/observability.

Our results on both academic and industrial examples show that our synthesis for testability techniques can be used to create asynchronous circuits that are fully *hfrpdft*.

Moreover, in practice we may not have to test *all paths*, but only each path that intervenes in one of the delay bounds used by the procedure described in Section 3.2. This would then be enough to guarantee the hazard-freeness of the manufactured circuit[7].

Finally our testing scenario, where all flip-flops is the circuit can operate in test mode, gives the required controllability and observability.

Thus in this paper we have preliminary indications that the three principal obstacles to comprehensive path-delay-fault testing of asynchronous circuits can be overcome.

The techniques of this paper were applied to circuits synthesized using the asynchronous synthesis procedures described in Section 3. However, because our synthesis for testability techniques so carefully preserve the nature of the original two-level cover, we believe that the same techniques can be applied to achieve fully testable implementations of circuits produced by other asynchronous synthesis procedures, such as the ones described in [18].

References

[1] M. J. Bryan, S. Devadas, and K. Keutzer. Testability-Preserving Circuit Transformations. In *Proceeedings of the Int'l Conference on Computer-Aided Design*, November 1990.

[2] T. A. Chu. On the models for designing VLSI asynchronous digital systems. *Integration: the VLSI journal*, 4:99–113, 1986.

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

[4] S. Devadas and K. Keutzer. Necessary and Sufficient Conditions for Robust Delay-Fault Testability of Logic Circuits. In *Sixth MIT Conference on Advanced Research on VLSI*, pages 221–238, April 1990.

[5] S. Devadas and K. Keutzer. Synthesis and Optimization Procedures for Robustly Delay-Fault Testable Logic Circuits. In *Proceedings of the 27th Design Automation Conference*, pages 221–227, June 1990.

[6] S. Devadas and K. Keutzer. Synthesis of Robust Delay-fault Testable Circuits: Practice. *IEEE Transactions on Computer-Aided Design of Integrate Circuits and Systems*, October 1991. To appear.

[7] S. Devadas and K. Keutzer. Synthesis of Robust Delay-fault Testable Circuits: Theory. *IEEE Transactions on Computer-Aided Design of Integrate Circuits and Systems*, September 1991. To appear.

[8] E. Eichelberger and T. W. Williams. A logical design structure for lsi testing. In *Proceedings of the 14th Design Automation Conference*, pages 462–468, June 1977.

[9] M. Hack. Analysis of production schemata by petri nets. Technical Report TR 94, Project MAC, MIT, 1972.

[10] S. Kundu and S. M. Reddy. On the Design of Robust Testable CMOS Combinational Logic Circuits. In *Proceedings of the Fault Tolerant Computing Symposium*, pages 220–225, 1988.

[11] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis of verifiably hazard-free asynchronous control circuits. Technical Report UCB/ERL M90/99, U.C. Berkeley, 1990.

[12] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Algorithms for synthesis of hazard-free asynchronous circuits. In *Proceedings of the Design Automation Conference*, June 1991.

[13] L. Lavagno, K. Keutzer, and A. Sangiovanni-Vincentelli. Synthesis of verifiably hazard-free asynchronous control circuits. In *Advanced Research in VLSI Conference*, March 1991.

[7]We can also test additional paths to make sure that the circuit does not have other kinds of faults, such as for example stuck-at faults.

[14] C. J. Lin and S. M. Reddy. On Delay Fault Testing in Logic Circuits. In *IEEE Transactions on Computer-Aided Design*, pages 694–703, September 1987.

[15] M. C. Paull and S. H. Unger. Minimizing the Number of States in Incompletely Specified Sequential Circuits. In *IRE Transactions on Electronic Computers*, volume EC-8, pages 356–357, September 1959.

[16] A. Pramanick and S. Reddy. On The Design of Path Delay Fault Testable Combinational Circuits. In *Proceedings of the* 20th *Fault Tolerant Computing Symposium*, pages 374–381, June 1990.

[17] S. M. Reddy and M. K. Reddy. Testable Realization for fet Stuck-Open Faults in CMOS Combinational Logic Circuits. In *IEEE Transactions on Computers*, volume C-35, pages 742–754, August 1986.

[18] S. H. Unger. *Asynchronous Sequential Switching Circuits*. Wiley Interscience, 1969.