A Protocol Extraction Strategy for Control Point Insertion in Design for Test of Transition Signaling Circuits

H.F. Li and P.N. Lam

Department of Computer Science, Concordia University
Montreal, Quebec, Canada H3G 1M8
E-Mail: hfl@vlsi.concordia.ca, ping@vlsi.concordia.ca

Abstract

Control/observation points have been used to detect undetectable faults in delay-insensitive/speed-independent circuits [5,10] but no techniques exist so far for its use in reducing the test length. The major difficulty is in deriving a safe hazard-free test. A theory for control point insertion is presented for the purpose of test length reduction of transition signaling circuits. It is based on extraction of safe behaviors from the original usage protocol via gap detection (identification of unnecessary behavior) and gap matching (jumping from one partial state to another). The area overhead is low, requires only a single input pad, and can give significant reductions in test length.

1.0 Introduction

Transition signaling circuits are asynchronous circuits which use 2-phase non-return to zero signaling. Recent work in the area include [3,4,6-9,13]. In particular, we are interested in the delay-insensitive form of these circuits, i.e., circuits which operate correctly independent of both component and wire delays. Recent work have concentrated on the testability of speed-independent and delay-insensitive circuits [1,2,5,9,10,14] and have not addressed design for test strategies for the purpose of test length reduction. The traditional methods used in synchronous circuits such as scan design are not practical for transition signaling circuits and circuits of [5,10] because of the high ratio of memory components to combinational components. In addition, transition signaling circuits require the input of transitions rather than level values provided by scan design. Transitions can be injected via control points, but there is one major reason which makes the problem difficult: arbitrary injection of transitions is likely to cause hazardous and race behavior. Moreover, the introduction of control points changes the circuit such that the original usage protocol cannot be directly used. In this paper, we introduce a strategy for extraction of safe behaviors via (i) gap detection, (ii) gap matching, and (iii) gap tour finding. Unnecessary behaviors are first identified (gaps) leaving only the necessary segments for the test. These segments are then executed in some order (tour) such that the partial states between segments match. State matching is performed via control point insertion and partial execution of gaps.

2.0 Preliminaries

In this paper, we assume that the Petri net specification/fault model of [9] is used. The set of basic components used in transition signaling circuits, such as (inverter, fork, C-element, toggle, XOR, demultiplexer), are each modeled as a Petri net where net places correspond to wire nodes (I/O ports), net transitions correspond to gate firings, and tokens represent signal transitions. The test is represented by Nt*Nc, the composition of the circuit net Nc and the environment test net Nt, where composition is the co-location (merge) of places which have the same action label. Nc is the composition of the individual component nets forming the circuit. Nt*Nc can be unfolded into a partial order behavior (pomset), similar to the unfolding of a Petri net into a configuration of an occurrence net [11]. Fig. 1 gives an example of the composition of a circuit net and its test net and the obtained pomset unfolding where unlabeled internal places are deleted.

![Fig. 1: (a) Net Nt*Nc, (b) circuit, (c) unfolding.](image)

Some definitions are required. A pomset (partially ordered multiset) [7,12] is defined as the isomorphism class of a labelled partial order. A labelled partial order is a 4-tuple \((V, \Sigma, \Gamma, \mu)\) where \(V\) is the set of events (vertices), \(\Sigma\) is a finite set of actions, \(\Gamma\) is a partial order (set of arcs) expressing necessary temporal precedences among \(V\), and \(\mu: V \rightarrow \Sigma\) is a labeling function mapping events to actions. Let \(p\) and \(q\) be pomsets. \(q\) is a prefix of \(p\) (\(q \leq p\)) if \(q\) is
obtained from p by deleting a subset of events from p such that if an event u is deleted and u precedes v, then v is also deleted. Let u be an event in pomset p. The maximal prefix of u, denoted M(u) is p minus all the successors of u and u. The maximal prefix $M(u)$ denotes the maximal prefix $M(u)$ plus u. The last events of p, denoted $p^\delta$, are the set of events $\{ u \in p \wedge (u,v) \in \Gamma \}$. Pomset subtraction $p-q$ results in a pomset in which all events of q are deleted from p. $A(p)$ denotes the set of actions in p and $E(p)$ denotes the set of events in p.

Let $Nt$ be a test schedule, $Nc$ be the net for a fault-free circuit C, and the pomset action set be the set of input, output and internal nodes of C. $Nt$ satisfies the at-least-k condition if the unfolding of $Nt \otimes Nc$ is a pomset that contains a prefix p such that (i) p contains at least k occurrences of each action in C, and (ii) the last events of p are output events. An inhibitory fault at a place s causes all tokens entering s to disappear and is modeled in the net by deletion of the input arc to s [this is a sa0 (sa1) fault on an initially low (high) wire]. An excitory fault causes a spurious token to appear at a place s and is modeled by the deletion of the input arc to s and the placement of a spurious token at s at circuit initialization [this is a sa1 (sa0) fault on an initially low (high) wire].

[9] shows that for CRF (Critical Race Free) circuits, any test which satisfies the at-least-2 condition is sufficient to detect any single SAF. An at-least-2 test ensures that tokens will traverse every node in the fault-free circuit twice (controllability) and that these tokens will reach some interface output (observability). However, if the circuit is not CRF, such as circuits containing demultiplexers, then a critical race may occur and at-least-2 may not be sufficient (the results of [2] use a more restrictive OUTPUT SAF model and applies for a particular set of components). For general circuits, an at-least-2 test will be sufficient with the aid of the following theorem and design for testability.

**Theorem 1:**
An at-least-2 test is sufficient to detect all inhibitory faults (proof omitted).

Excitory faults can be easily detected in transition signaling circuits by checking for spurious tokens at the inputs of terminal components (C-elements) because any spurious token appearing at circuit initialization will traverse through non-terminal components (inverter, fork, toggle, XOR, demultiplexer) and stop at the terminal components or interface output. Such a check can be implemented by a large OR gate to observe the inputs of C-elements at circuit initialization, giving around 10%-15% area overhead, depending on the number of C-elements in the circuit. Techniques to eliminate the need to observe these nodes exist and will not be expanded upon.

We can therefore concentrate on at-least-2 behaviors and ask how the insertion of control points can reduce the length of the at-least-2 test. The simplest example is the insertion of a control point into the middle of a mod-n counter (Fig. 2) through an XOR. The insertion partitions the counter into two mod-n/2 counters, allowing the two counters to be exercised separately, the first counter through input a and the second counter through control input t. The length of the test is reduced from $O(2^n)$ to $O(2^{n/2})$.

**Fig. 2: Insertion of control point in a mod 64 counter.**

Given K control points to partition the circuit into K subcounters, the length of the test can be reduced to be linearly proportional to the size of the circuit. Fig. 3 shows the general case of control point insertion where $t_i$ are the control points and tokens are distributed to $t_i$ via a sequence generator [6] using a single input pad at t.

**Fig. 3: General case of control point insertion.**

The result of control point insertion/token injection is the skipping of segments of the pomset behavior. Execution of the pomset behavior can allow a jump from one prefix p to another prefix q if the partial states of p and q are the same:

Let $U$ be the universe of actions of a circuit and $B \subseteq U$ be a partial set of actions. The partial state of prefix p is $\Pi(p,B) = \{ a \in p \wedge a = A(u) \wedge a \in B \wedge a = \text{output action} \}$. Let $S_p = \Pi(p,B)$ be the partial state of p, then $S_p = B - S_p$. $S_p$ represents the places in $B$ containing tokens in the subnet and $S_p$ the places without tokens. The (global) state of p is $\Pi(p,U)$.

The ability to jump from one prefix to another, thereby skipping a segment of the test protocol, is given by theorem 2, i.e., if the subnets corresponding to the partial states of p and q are identical, then execution of the future behavior $r-q$ will result in a safe behavior. In the sequel, we will show how control points can be inserted to bridge differences between these two partial states.

**Theorem 2:**
Let $p,q,u$ be pomsets where $p \subseteq q \subseteq r$, and $r$ be the unfolded behavior of a 1-safe net $N=Nt \otimes Nc$. Let $Np \otimes Ncp$ be the net...
after unfolding N up to p and \( N_t \cdot N_c \) be the net after unfolding N up to q. If \( \Pi(p, B) = \Pi(q, B) \) where \( B = A(r-q) \), then the unfolding of \( N_t \cdot N_c \) is 1-safe and unfolds into \( r-q \).

The insertion of control points and extraction of the test protocol is generalized in the next three sections.

### 3.0 Gap Detection

To derive the pomset segments which will be used in the test, we first extract a series of gaps, \( G_1, ..., G_n \) (Fig. 4) from the original at-least-2 pomset \( P \), where a gap is a pomset segment containing events which are redundant and not required to satisfy the at-least-2 condition of the new test behavior. We want to repeatedly find the largest gaps in \( P \) to minimize the test length. Finding the largest gaps reduces the total number of gaps in \( P \), so there are a smaller number of gaps to skip and hence fewer partial states to match, which usually implies fewer control points required.

![Fig. 4: Event gaps in an at-least-2 test behavior.](image)

**Problem 1:** (Gap Finding)

Let \( P \) be an at-least-2 test pomset and \( r,s \) be some prefixes of \( P \), where \( r \leq s \leq P \). Detect the largest size gap (size measured by number of events) \( G = s-r \) such that

(i) \( E(r) \cup E(P-s) \) contains at least two occurrences of every action in \( P \), and

(ii) \( |E(G)| \geq \text{GAP}_\text{SIZE} \).

Problem 1 can be proved to be NP-complete, however, the subproblem of finding the largest size gap starting from a fixed prefix \( r \) and advancing (increasing) \( s \) can be solved in polynomial time. The following algorithm finds a sequence of gaps in \( P \).

**Algorithm 1:** (Gap Finding Algorithm)

\[
p = A; \ i = 1; \ S_A = A(P);
\]

while \( p < P \) do

\[
r = p;
\]

\[
\text{Let } c(a) \text{ be the total order of the event occurrences of action } a \text{ for } a \in S_A;
\]

\[
s = \bigcup M(u) \text{ where (i) } u \text{ is the last event of chain } c(a)
\]

if one event in p is labeled with action a, and (ii) u is the second last event of chain c(a) if no events in p are labeled with action a:

if \( |E(s-r)| \geq \text{GAP}_\text{SIZE} \) then

\[
G_i = s-r;
\]

\[
i = i + 1;
\]

end if

p = s;

if \( p < P \) then

Advance p past immediate successor events of s;

Advance p to next stable state;

\( S_A \) = set of actions not occurring twice in p;

end if

end while

(Where a stable state occurs when no net transitions can fire without requiring an input event from the test interface)

The algorithm advances the prefix p sequentially forward and detects the largest size gap starting from the fixed prefix p. The subproblem of finding the largest gap starting from a fixed prefix requires \( O(mn) \) time where \( m \) is the number of actions in \( P \) and \( n \) is the number of events in \( P \). Fig. 5 illustrates this step. It relies on the fact that events of the same action must appear linearly in a chain (no auto-concurrency) in all delay-insensitive behaviors [7]. The maximum size gap must exclude either the last or second last event of these chains for the actions which have not been covered twice yet. Taking the maximum prefix of these events is the minimum requirement in excluding these events, hence the gap will be maximized.

![Fig. 5: Maximum size gap starting from a fixed prefix r.](image)

Fig. 8 demonstrates the gaps (shaded regions) found by Algorithm 1 in the pomset behavior of the circuit of Fig. 7. The value of \( \text{GAP}_\text{SIZE} \) used is 15 events. Due to space limitation, the internal component actions of the toggle and demultiplexer are not shown (each of these components contain two internal places which record their states).

### 4.0 Gap Matching

Once the initial gaps are found, the test can cover the non-gap behavior by either sequentially skipping gaps from \( G_1 \) to \( G_n \) (executing \( z_0, z_1, ..., z_n \)) or non-sequentially jumping from one gap to another to cover the at-least-2 behavior. Suppose \( r_1 (1 \leq s \leq n) \) is the prefix at the beginning of gap \( G_i \) and \( s_i \) is the prefix at the end of \( G_i \). Sequential skipping of gaps is performed by: for \( i = 1 \) to \( n \), jump from \( r_i \) to \( s_i \) and execute the behavior from \( s_i \) to \( r_{i+1} \). Non-sequential skipping of gaps is performed by repeatedly jumping from \( r_i \) to \( s_j \) (\( 1 \leq i \leq n \)) and executing the behavior from \( s_j \) to \( r_{i+1} \) until all the non-gap behavior is covered.

A correct jump from \( r_i \) to \( s_j \) requires that the partial
state at \( r_1 \) match the partial state at \( s_j \). Prefix \( r_1 \) must also be a stable state. If the partial state matches exactly, then the gap can immediately be skipped (Theorem 2). Otherwise, partial execution of the prefixes is performed to obtain a match: forward advancement of \( r_1 \) and backward advancement of \( s_j \). Since we are jumping from \( r_1 \) to \( s_j \), control points can be used to force \( s_j \) to match \( r_1 \) if the partial state of \( r_1 \) contains a token/action while the partial state of \( s_j \) does not. The token is injected via an XOR at the circuit node corresponding to that action. Since the internal component actions of toggles and demultiplexers cannot be controlled (without changing the components by adding an explicit set/reset) these internal actions are matched by partial execution of the gap.

Let \( G_1, ..., G_n \) be gaps in pomset \( P \), \( G_1 = s_1 - r_1, G_j = s_j - r_j \), \( r_1 \leq g \leq s_1, r_j \leq h \leq s_j \), \( D_j = |E(G_j)| \), and \( D_j = |E(s_j-h)| \) for \( 1 \leq j \leq n \). Let \( S_g = \Pi(g,B) \), \( S_h = \Pi(h,B) \), and \( d(S_g,S_h) = |I(E(G_j)| - D_1 - D_j |D_1 - D_j| \) be the gap size during advancement and \( GAP\_SIZE2 \) be the smallest size gap tolerated after advancement.

**Problem 2: (Gap Matching)**

For some set of partial state actions \( B \) required to be matched and some constant \( K \), find prefixes \( g \) and \( h \) such that \( d(S_g,S_h) \leq K \) and \( GAP\_SIZE(G_j,D_1,D_j) \geq GAP\_SIZE2 \).

\( K \) represents the number of control points available which is used to force the non-matching components of \( g \) and \( h \) to match while \( D_j \) \( (D_j) \) is the number of gap events \( g \) has advanced past \( (h \) has backward advanced past). Fig. 6 illustrates this where \( g \) is advanced forward and \( h \) is advanced backward to find a partial state in which \( g \) and \( h \) match.

---

**Fig. 6: Partial execution of gaps.**

Since Problem 2 can be proved to be NP-complete, the following approximation algorithm can be used:

**Algorithm 2: (Gap Matching Algorithm)**

**detect_state**\( (g, h, B') \)

\[
P' = g \sim h; \quad S_g = \Pi(g,B); \quad S_h = \Pi(h,B');
\]

while \( (S_g \neq S_h) \land \) \( GAP\_SIZE(G_j,D_1,D_j) \geq GAP\_SIZE2 \) do

\[
\text{foreach } a \in \text{actions differing in } S_g \text{ and } S_h \text{ do}
\]

Let \( u \) be last event in chain \( c(a) \) preceding \( h \)

\[
A(u) = a;
\]

if \( (S_g \cap S_h) \) then

\[
h = h - (P' \sim M(u));
\]

if \( (S_g \cap S_h) \) then

\[
h = h - (P' \sim M(u));
\]

endfor

\[
S_h = \Pi(h,B');
\]

endwhile

if \( S_g = S_h \) then Return \( (g, h) \);

else Return \( (\emptyset) \);

end **detect_state**

**main**\( (B, K) \)

\[
g = r_1; \quad S_g = \Pi(s_j,B)
\]

Repeat

\[
\text{Let } \{CB_1, ..., CB_m\} = \text{sets of combinations of } K \text{ actions out of } B;
\]

\[
\text{for } k = 1 \text{ to } m \text{ do}
\]

\[
h = s_j; \quad B' = B - CB_k
\]

if \( \text{detect_state}(g, h, B') \neq \emptyset \) then Return \( (g, h) \)

and exit main.

endfor

Advance \( g \) to next stable state and obtain new \( S_g = \Pi(g,B) \) such that \( d(S_g,S_h) \) is minimal;

Until \( |E(s_j - g)| < GAP\_SIZE2 \)

Report no match found.

end **main**

Algorithm 2 fixes \( g \) first and backward advances \( h \) to find a match. The subroutine **detect_state**() requires polynomial time and guarantees detection if a matching state exists. Combinations of \( K \) actions are subtracted from \( B \) (representing the actions which may be matched with control point insertion) so that the exact match algorithm **detect_state**() can be used. This is justified for small \( K \), which is usually the case when we do not want to insert too many control points.

**5.0 Tour of Gaps**

A non-sequential execution of the behavior can allow less control points to be used. Given a set of gaps extracted \( \{G_1, ..., G_n\} \), the pomset segments between the gaps \( \{z_{0}, ..., z_{n}\} \), and \( G_1 = s_1 - r_1 \) for prefixes \( r_1, s_1 \) a tour of the gaps is given by the sequence \( \langle\pi(1), \pi(2), ..., \pi(n)\rangle \) where \( \pi(i) \) maps \( G_i \) to the new ordering. The test executes the pomset \( \pi(0), \pi(1), \pi(2), ..., \pi(n) \). A jump of a gap from \( r_{\pi(0)} \) to \( s_{\pi(0)} \) requires \( \Pi(r_{\pi(0)},B) = \Pi(s_{\pi(0)},B) \) where \( B = A(\cup_{i=1}^{n} z_{\pi(i)} \cup s_{\pi(i)}) \). The size of \( B \) covers all the actions which appear at the tail end of the behavior \( z_{\pi(i)}; ..., z_{\pi(n)} \) to ensure that there are no token collisions or races.

**Problem 3: (Tour Finding)**

\[
\forall_{i=1..n} \text{ Let } S_{\pi(0)} = \Pi(r_{\pi(0)},B), S_{\pi(0)} = \Pi(s_{\pi(0)},B), \text{ where } B = A(\cup_{i=1}^{n} z_{\pi(i)} \cup s_{\pi(i)}) \). Find a tour \( \langle\tilde{G}_{\pi(1)}, \tilde{G}_{\pi(2)}, ..., \tilde{G}_{\pi(n)}\rangle \) which satisfies \( \sum_{i=1..n} d(S_{\pi(i)}, S_{\pi(i)}) \leq K \).

Problem 3 can be reduced to the traveling salesmen problem. Different approximation algorithms can be used. One possible method is to divide the behavior into a set of partitions containing a constant number of gaps and finding the optimal subtour within each partition. The algorithm should move backward from the last partition to the
first, obtaining the tail end of the tour first so that only partial states need to match. Partial states cannot be used when scanning forward because the partial state requirement \( B \) cannot be known until the tail end of the tour is fixed. Forward scanning requires full global states to match which is more expensive. If there are not enough control points, then some gap can be deleted. Alternatively, an insertion algorithm can be used which starts with a sequence containing the last gap \( G_a \) and inserts the other gaps one at a time into the sequence at the location which gives the least increase in cost.

The example of Fig. 8 shows the case where a sequential tour is used and \( K = 4 \) control points (setting any \( K \geq 1 \) also gives solutions which reduce test length). The location of control points which will be used to inject tokens are circled, \( \{16, 19, 110, 113\} \). The square brackets indicate the actual gaps used after gap matching is performed. With a more intelligent selection of control points, \( 19 \) can be deleted and replaced with \( 16 \). The reduction in the number of events required in testing is from 709 to 178 which is a significant reduction. Generally, circuits which have a high degree of encoding such as this circuit and the circuits of [6] will allow greater reductions. Since the length of the pomset represents the approximate time required for testing, the deletion of 75% of the test pomset gives a corresponding time reduction. A single input pad connected to a sequence generator can be used to distribute the tokens to the control points; it can also be proved that a SAF in the sequence generator or the control points themselves will not invalidate the test. When \( K \) is set to 0, this strategy can be used to shorten test behaviors without using control points. Behaviors which contain redundancies in exercising the circuit by the environment will automatically be detected, such as gap \( G_4 \) of Fig. 4 which can be skipped without requiring any control points.

### 6.0 Conclusion

A design for test strategy is proposed for shortening the test length of transition signaling circuits. Guaranteed hazard free tests are derived from the original usage protocol through the skipping of gaps, gap matching via partial execution and control point insertion, and tour finding. Efficient algorithms are given, which also demonstrates the advantage of performing operations in partial order space rather than state space.

### References


Fig. 8: Event gaps in an at-least-2 test behavior.