Partial Scan Delay Fault Testing of Asynchronous Circuits *

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

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

Luciano Lavagno
Politecnico di Torino
10129 Torino, Italy

Alexander Saldanha
Cadence Berkeley Laboratories
Berkeley - CA 94704

Alexander Taubin
The University of Aizu
Aizu-Wakamatsu, 965-80 Japan

Abstract

Asynchronous circuits operate correctly only under timing assumptions. Hence testing those circuits for delay faults is crucial. This paper describes a three-step method to detect possible delay faults in a sequential *asynchronous circuit. The delays that are to be tested must be provided by the synthesis system. By using this information a set of paths in the circuit that must be tested is identified (step 1). For these paths the circuit is made acyclic by inserting at least one scan latch in every cycle (step 2). Then test patterns are generated for these paths (step 3). These test patterns consist of setup and initialization vectors and the final test vector. We provide effective procedures to solve both the initialization and the test pattern generation problem. The latter problem is solved by reduction to a classical problem of* stuck-at *test pattern generation for a related* combinational *circuit. Finally, a heuristic is proposed to determine which state variables must become part of a scan chain, or for which input variables the positive and negative phase must be driven independently in test mode. Experimental results shows that a high level of path delay fault testability can be achieved with partial scan.*

1 Introduction

Correct operation of asynchronous circuits depends on timing assumptions that are much more complex than those in the synchronous case. In particular, an asynchronous circuit is by construction insensitive to most *delay faults*, because they often affect only its *performance*, not its *functionality*. Some delay faults, though, may have an effect on the correctness as well, and hence it is necessary to be able to test them ([23]). Unfortunately, testing asynchronous circuits is a difficult problem, due to the following main reasons:

- All known asynchronous design methodologies ensure correct *operation* (hazard-freedom) by using some level of *redundancy*, i.e., by sacrificing testability.
- Asynchronous control circuits tend to have more feedback and more registers than their synchronous counterparts. This means that full-scan testing may be unacceptably expensive.

This paper deals with the problem of generating test sequences for a given set of paths in an asynchronous circuit. We assume that the information about which delays in the manufactured circuit must be tested to ensure correct operation is available (e.g., from the synthesis tools). Previous work in the area of asynchronous circuit testing either used greedy heuristic techniques ([4]) to justify and propagate stuck-at faults, or used exhaustive synchronous mode testing for stuck-at faults ([3, 19]) or used manual transformations to ensure that a simple functional testing approach could test all stuck-at faults ([20]), or used a full-scan approach to robustly test all delay faults ([8, 12, 17]).

We consider two versions of the path delay fault testing problem: *robust* path delay fault testing (**RPDFT**) and *hazard-free robust* path delay fault testing (**HFRPDFT**). The former test may allow better coverage and is simpler to generate. It guarantees that hazards in a circuit under test cannot produce false positives, but false negatives can occur. The latter guarantees that during the test, hazards cannot propagate along the paths under test, and does not admit false negatives. For sequential circuits it also guarantees that meta-stability cannot occur in the latches on the paths under test. (See [23] for an in-depth discussion of different versions of the path delay fault testing problems.)

We solve the problem of path delay fault testing for asynchronous sequential circuits as follows.

Step 1: identification of a set of paths that cover all potentially dangerous faults[1] (Section 2.2). This is obtained by finding a set of linear inequalities that bound every relevant delay constraint (e.g., determining that the difference between two delays in a fanout stem is less than a given amount). All known synthesis procedures for asynchronous circuits provide this information either in the form of path delay bounds (e.g. [13]) or in the form of constraints on the relative delays of the branches of a fanout stem (e.g. [10]).

Step 2: reduction to asynchronous circuits with acyclic behavior (Section 2.1). The problem of testing an asynchronous circuit is reduced, by using a *partial scan* approach, to that of testing an object called an *asynchronous net*, in which feedbacks are allowed only inside asynchronous latches (e.g., Muller C elements, with inputs a and b and next state equation $c' = ab + ac + bc$). An asynchronous net is still a sequential object with internal memory, but it can exhibit only *acyclic behavior*.

Step 3: test sequence generation. For a combinational circuit, a delay fault test consists of *pairs* of vectors

*This work has been partially supported by by the Esprit project 21949 - ACID WG (M. Kishinevsky, L. Lavagno and A. Taubin) and EPSRC Visiting Fellowship GR/L24038 (A. Kondratyev and L. Lavagno).

[1] A delay fault is "dangerous" if it violates some assumption made during synthesis, e.g. a fundamental mode constraint, an isochronic fork and so on [13].

0-89791-993-9/97 $10.00

$< v_0, v_1 >$ applied to the primary inputs of a circuit. Vector v_0 sets the outputs and the side inputs of the gates along the path under test to values which allow the propagation of the desired transition when v_1 is applied. For sequential circuits, testing a delay fault requires in general the application of a *sequence* of vectors. The first part of this sequence performs correct *initialization* of latches along the path. The second part of the sequence is a *testing pair* $< v_0, v_1 >$, where the *setup vector*, v_0, sets the outputs and the side inputs of the gates along the path under the condition that all latches are already initialized as required. The *test vector*, v_1 propagates the transition along the path.

We decompose the problem of testing asynchronous nets into that of initializing memory elements, followed by path delay fault testing.

Step 3a: generating testing pairs $< v_0, v_1 >$ (Section 3). This problem is solved by reduction to stuck-at test pattern generation for a *combinational circuit* that can be directly derived from an asynchronous net. This approach was proposed in [21] for RPDFT of combinational circuits. Additional conditions on the generated stuck-at test patterns for reduction of HFRPDFT are given. The method is further generalized for *sequential* nets, by modeling each latch with a combinational model (similar to modeling of latches in time-frame unrolling [1]). We derive conditions on the value of the state inputs under which the test for the combinational circuit is valid also for the asynchronous net, without resorting to time-frame unrolling.

Step 3a: Step 3b: generating initialization sequences (Section 4). Vector v_0 obtained at step 3a is the target of the initialization procedure. We present a heuristic algorithm for monotonous initialization that (if successful) generates initialization sequences bounded by $n^2/2$, where n is the latch count. Otherwise, we resort to classical time-frame unrolling [1] (that has an upper bound on the test sequence length of 4^n).

We improve with respect to previous work because our approach:

- Is complete, because it finds a test sequence for a given fault if one exists (while [4] heuristically maximized the number of tested paths, by using a greedy search algorithm). Note that previous work ([12]) has shown that asynchronous circuits generated with every known synthesis technique can be tested for delay faults by using a full-scan approach, so we can claim that every delay fault can be tested using the proposed method.
- Is automated (while [20] requires the designer to manually insert special circuitry, acting only in functional test mode, under guidance from a testability analysis tool).
- Requires only partial scan (while [8] required full scan and required additional test inputs and [17] is based on full scan and uses transformations of combinational logic increasing the level of testability).
- Requires only the output of a memory element to be scanned (while [12] required both inputs to each element to be independently scanned, that in general can be quite expensive).

The paper is organized as follows. Section 2 reviews the basic notions of delay fault testing and adapts them to asynchronous sequential circuits. Section 3 describes the reduction of HFRPDFT of sequential nets to that of combinational nets. Section 4 presents a procedure for initialization of asynchronous nets. Section 5 provides experimental results.

2 Preliminaries

2.1 Asynchronous circuits and nets

An *asynchronous circuit* is an arbitrary interconnection of logic gates and input nodes, with each gate input connected to strictly one gate output or one input node, and with no two gate outputs tied together. Feedback can be either *local* inside gates (like SR latches or C-elements) or *global* outside gates.

Our strategy for testing asynchronous circuits is based on breaking all *global* feedback loops, by selecting a Minimum Feedback Vertex Set of the circuit graph, and converting all its gates into scan memory elements (like [5, 14] in the synchronous case). Such transformation is obviously easier and cheaper if the selected gates are memory elements (see [12] for a scan SR latch circuit). Outputs of such gates then become simultaneously new primary inputs and primary outputs of the circuit.

We call the resulting circuit, in which feedback can only be local, an *asynchronous net*. In this paper we will consider a particular class of asynchronous nets, which are composed from *simple gates* (AND, OR, NAND, NOR, and NOT) and C-elements. Using the macro-expansion operator [16], any complex gate can be converted to an equivalent connection of simple gates preserving testability properties. Handling asynchronous memory elements other than C-elements is a possible area of future work.

2.2 Identifying the paths to be tested

An asynchronous circuit operates correctly without hazards only if some delay constraints, which differ according to the design style used, are satisfied. All such constraints can be formulated in terms of comparisons among event propagation times along some circuit paths. For example, speed-independent circuits ([10]) operate correctly if and only if all the branches of a multiple fanout point have similar delays (generally, the maximum admissible spread is comparable with one gate delay).

Let us model the delay of each wire i in a circuit by using a variable d_i. In that case, the set of delay constraints that ensure the correct operation of the circuit can be modeled by a set $\mathcal{A}$ of linear inequalities over those variables. The problem, then, is to find a set of paths that allow us to prove that $\mathcal{A}$ is indeed satisfied, by bounding the delay along them. In other words, we would like to be able to find a set $\mathcal{D}$ of linear inequalities, each involving a measurable delay along an I/O path of the corresponding asynchronous net, such that the set of feasible solutions (assignments to the d_is) of $\mathcal{A} \cup \mathcal{D}$ is *the same* as that of $\mathcal{D}$.

The simplest solution to this problem is to greedily add testable path after testable path, until the inequalities in $\mathcal{A}$ all become *redundant* (i.e., the *assumed* delay bounds are *implied* by the *measured* delay bounds). A better solution, that is left to future work, would require to minimize the cardinality of the set of tested paths.

Note that in this case the trade-off between test sequence length and speed at which the circuit can operate ([11]) is possible only if the objective of the test is the determination of the actual performance of the circuit, because no compromise about its correctness is generally possible.

2.3 Path delay faults

In this paper we will use delay fault testing models originally developed for combinational circuits, and extend them to asynchronous nets. Testing for delay faults in that case requires the application of a *pair* of vectors $< v_0, v_1 >$ to the primary inputs to force a signal transition propagation along the path under test $\pi = \{g_0, g_1, g_2, \ldots, g_k\}$. Vector v_0 is called the *setup vector*, vector v_1 is called the *test vector*. The testability conditions are based on considering side-inputs and side-paths.

Definition 2.1 *Let $\pi = \{g_0, g_1, \ldots, g_k\}$ be a path. The inputs of g_i other than g_{i-1} are called* **side-inputs** *of g_i along π and denoted as $S(g_i, \pi)$.*

A path that starts at a primary input and ends at a side-input of $g_i \in \pi$ is a **side-path** *of π. If a side-path starts at the same primary input g_0 as path π, then it is called a* **reconvergent side-path** *of π.*

Definition 2.2 *A* **controlling value** *for a gate g (denoted as $C(g)$) is a value of* one *of its inputs that determines the value at the output independent of the other inputs. Otherwise a value of the input is called* **non-controlling** *and denoted $NC(g)$.*

Note that a C-element has no controlling values, since the next value at the output always depends either on the value at *both* inputs, or on the previous value at the output. We then extend the definition to asynchronous nets as follows.

Definition 2.3 *A* **controlling set** *for a gate g (denoted as $CS(g)$) is a set of values at* some *of its inputs that determines the value at the output independent of the other inputs or the previous state of the gate. Otherwise a set of values at some inputs is called* **non-controlling** *and denoted $NCS(g)$.*

For example, $\{1, 1\}$ and $\{0, 0\}$ are two controlling sets for a two-input C-element and $\{1, 0\}$ and $\{0, 1\}$ are two non-controlling sets.

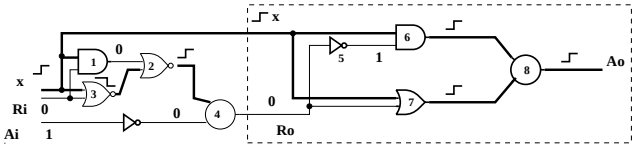

Figure 1: Path delay fault testing in asynchronous nets

A setup vector v_0 for a combinational path has two functions: (1) it sets the outputs of all the gates along the path, and (2) it sets the side inputs of the same gates to values which allow the propagation of the desired transition. The first function is called *initialization*, the second is called *setting*. This may also work for a sequential path, as shown in Figure 1. Assume that path $\pi_1 = \{x, 6, 8\}$ is under test for the rising input transition. It is easy to see that vector $v_0 =< x = 0, Ri = 0, Ai = 1 >$ will initialize the C-elements 4 and 8 into state 0 and also will set the side-input for gate 6 on the path $\pi_1 = \{x, 6, 8\}$ and the side-input for gate 7 on the path $\pi_2 = \{x, 7, 8\}$ at non-controlling values. Then, by applying vector $v_1 =< x = 1, Ri = 0, Ai = 1 >$ a rising transition will propagate from input x to output Ao along two paths π_1 and π_2. If there is a delay fault in at least one of the paths, it will be observed at the output.

Multiple paths can be tested with the same pair of vectors (see Figure 1). In that case, more than one constraint is obviously added to the set $\mathcal{D}$ that is used to bound the timing assumptions.

However, as will be shown in Section 4, testing for delay fault in a sequential path requires in general the application of a *sequence* of vectors. The first part of this sequence is called an *initialization sequence* and is concerned only with correct initialization of C-elements (latches) along the path. Such initialization sequence (which would not invalidate the test) may not exist or may require applying multiple vectors. The second part of the sequence is called a *testing pair* since it always consists of a setup and a test vector. If no such sequence exists for a given fault, we can always introduce new scan registers. In the limit, when all sequential elements are scanned, the circuit becomes testable under very weak assumptions (absence of single-cube-contained implicants [12]).

2.4 Path Delay Fault Testing

In this paper we consider two possible approaches to path delay fault testing ([21, 7, 23, 18, 22]):

RPDFT: A path is *robust delay fault testable* if there is a test pair for the path delay fault that is valid under arbitrary delays along other paths. In other words, hazards cannot invalidate a robust test. However some hazards may propagate to the output node of the path.

HFRPDFT: A path is *hazard-free robust delay fault testable* if there is a robust test vector pair for which hazards may not occur along the path under test.

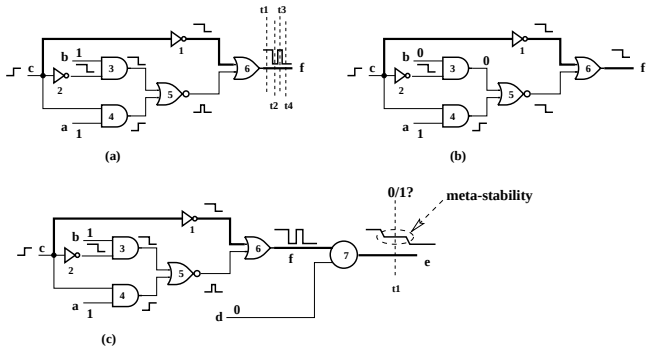

Figure 2: Robust and hazard-free robust delay fault testing for combinational (a,b) and sequential (c) nets.

The difference between these two models for combinational nets is illustrated by checking the testability of path $\pi = \{c, 1, 6\}$ in Figure 2. As shown in Figure 2,a test pair $< v_0, v_1 >=< 110, 111 >$ (the variable order is $< a, b, c >$) may cause a dynamic 1-to-0 hazard at the output of gate 6. However, if π has a longer delay than

expected, then a falling transition at the output of gate 6 is delayed. In this case hazards cannot invalidate the test and $< 110, 111 >$ is a robust test. If the output of gate 6 is observed at t_1, when it has value 1, then we correctly conclude that there is a delay fault along π. If the output is observed at t_2 or t_4, when the output is at 0, we correctly conclude that there is no delay fault along π. However, if the output is observed at t_3, when the output is at 1 due to hazards at the output of gate 5, then a false negative occurs. We incorrectly report a delay fault along π. Therefore, the robust test is conservative and can produce false negatives. As shown in Figure 2,b test pair $< v_0, v_1 > = < 100, 101 >$, propagating transition along two paths $\pi = \{c, 1, 6\}$ and $\{c, 4, 5, 6\}$, is HFRPDFT since no hazards may occur along π.

One may consider that hazard propagation is particularly dangerous for asynchronous circuits, because hazards may lead memory elements into a meta-stable state. Figure 2,c shows an example of such behavior: a C-element (gate 7) with output e at 1 and input d at 0 sees a 1-0-1-0 dynamic hazard on input f. The second vector of a test pair, whose application may cause internal hazards under the RPDFT model, always forces a *controlling set* for all the gates (including C-elements) on the path under test. This means that any C-element that may enter a meta-stable state due to a hazard is also forced to leave the meta-stable state by the time v_1 finishes propagating. Nevertheless, if (due to a delay fault) the output of the C-element is observed at $t1$, when it is in meta-stable state, then it may cause uncertainty in the test machine. A meta-stable output either keeps a value between the logical "0" and logical "1" or oscillates. Hence the effect of meta-stable values on the primary outputs is similar to that of hazards. Both can cause false negative results.

We will then discuss both hazard-free and non-hazard-free testing, because the latter may allow better coverage at the expense of more false negative results.

Definition 2.4 *Let $\pi = \{g_0, g_1, \ldots, g_k\}$ be a path in an asynchronous net. We say that gate $g_i, 0 \leq i \leq k$, has an* **even input parity** *if there is an even number of inverters along the path π from g_0 to the output of g_i. Otherwise, g_i has an* **odd input parity**. *Similarly, we define* **output parity** *as the number of inverters on a path from g_i to g_k. The input parity is denoted as $IP(g_i, \pi)$ and the output parity as $OP(g_i, \pi)$. $IP(g_i, \pi), OP(g_i, \pi) \in \{even, odd\}$.*

Definition 2.5 (RPDFT [21])
A path $\pi = \{g_0, g_1, \ldots, g_k\}$ in an asynchronous net is said to be **robustly path delay fault testable** *for the rising input transition by the vector pair $< v_0, v_1 >$ if for each $g_i \in \pi$ and for each side-input $f_j \in S(g_i, \pi)$ the following conditions hold:*
(1) $g_i(v_0) \neq g_i(v_1)$; (2) if $IP(g_i) = even$, then $g_i(v_1) = 1$ otherwise $g_i(v_1) = 0$; (3) $f_j(v_1) \in NC(g_i)$; (4) if $g_{i-1}(v_1) \in C(g_i)$, then there is no transition on f_j.

This definition also applies to C-elements. In particular, condition 1 requires that the input values of every C-element on π is a controlling set under the vector v_1, and the C-element output has the opposite value under v_0. Conditions 3 and 4 do not apply to C-elements, since they refer to controlling and non-controlling values. Note that there is no constraint on the transitions on side-inputs for C-elements, other than that specified by condition 1.

Definition 2.5 does not restrict transitions at the side-inputs if $g_{i-1}(v_1)$ has a non-controlling value. Therefore hazards may occur.

Testability for the falling input or rising and falling output transitions is defined similarly and differs only in condition 2 (e.g., for the falling output transition condition 2 is as follows: if $OP(g_i) = even$, then $g_i(v_1) = 0$ otherwise $g_i(v_1) = 1$).

For the **HFRPDFT** one more condition must be added to prevent hazards along the path under test:
(5) if $g_{i-1}(v_1) \in NC(g_i)$ or g_i is a C-element, then either there is no transition on f_j or there is one monotonous transition on f_j such that $f_j(v_1) = g_{i-1}(v_1)$.

In the next section we develop the theory for HFRPDFT, since it is the most complex case. A simplified version of the theory, that applies to RPDFT, can be easily derived as well.

3 Reduction of HFRPDFT to stuck-at

The problem of HFRPDF test generation for asynchronous nets is solved by reducing it to classical stuck-at test pattern generation for a combinational circuit which can be directly derived from the asynchronous net. Since asynchronous nets contain latches they exhibit sequential behavior. Hence, in general, a stuck at test for an asynchronous net is not a single vector but may require applying a sequence of vectors. For this reason the reduction is done in two steps:

- Relating sequential HFRPDFT to combinational HFRPDFT (Section 3.1)
- Relating HFRPDFT for a combinational circuit to stuck-at test generation for a slightly modified combinational circuit. This approach was proposed in [21] for RPDFT of combinational circuits. Additional conditions on the generated stuck-at test patterns for reduction of HFRPDFT can be found in [9].

Note that, in practice, the first step is performed *relative to a particular test vector pair* (Section 4 describes one such approach), so the order in the algorithmic implementation is reversed. Moreover, potentially there is a need to backtrack and select another test pair if it does not satisfy the initializability conditions.

3.1 Relating sequential HFRPDFT to combinational HFRPDFT

Given an asynchronous net, C, let us substitute each C-element $c_j = a_j b_j + a_j c_j + b_j c_j$ with its combinational model, that is a majority gate (M-gate) M_j. An M-gate implements the following boolean function: $c_j = a_j b_j + a_j m_j + b_j m_j$, where m_j is an additional primary input (called M-input). Since the majority function is unate and each M-input fans out only to one M-gate M_j, there is no need to decompose M_j into simple gates. In the following we will always consider M_j as an atomic gate. The conversion of C-elements to combinational M-gates is purely logical and is done for the algorithm of test generation for the original sequential circuit, no actual physical transformation of the original sequential circuit is required.

Definition 3.1 *The combinational circuit obtained from an asynchronous net C by replacing each C-element with an M-gate, is called an* **M-net** *and is denoted $M(C)$.*

If an asynchronous net has primary inputs $I = \{i_1, \ldots, i_k\}$ and C-elements $L = \{c_1, \ldots, c_l\}$, then the corresponding M-net has $k + l$ primary inputs $\{i_1, \ldots, i_k, m_1, \ldots, m_l\}$. The following conditions determine the value of the M-input that implies a stable behavior of the M-gate if its output is connected to its input (thus forming a C-element again). Figure 3 shows an example of M-net, corresponding to the asynchronous net from Figure 1.

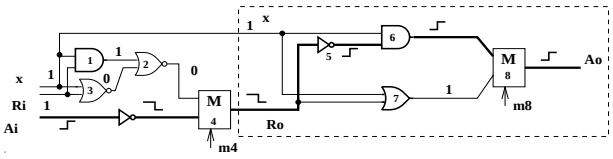

Figure 3: M-net

Definition 3.2 *A vector v of inputs to the M-net is called* **consistent with initialization** *if for each M-gate, M_j, the following condition is satisfied: $c_j(v) = m_j(v)$.*

In other words, the final value c_j at the output of each M-gate after applying v is the same as that of the M-input m_j.

Let $v =< v_0, \ldots, v_k >$ be a binary or ternary vector ($v_i \in \{0, 1, -\}$) for variables from set X and let $Z \subseteq X$. Then $v \downarrow Z$ denotes the sub-vector of v corresponding only to variables from Z. The following theorem states the conditions under which a *combinational* logic test derived for an M-net is valid also for the corresponding *sequential* asynchronous net.

Theorem 3.3 *Let C be an asynchronous net with set of primary inputs I and set of C-elements L. Let $\pi = \{i, g_0, \ldots, g_k\}$ be a path in C and $\pi' = \{i', g'_0, \ldots, g'_k\}$ be the path corresponding to π in the M-net $M(C)$. If $< v_0, v_1 >$ is a HFRPDFT for π' in the M-net and v_0 is consistent with initialization, then $< v_0 \downarrow I, v_1 \downarrow I >$ is a HFRPDFT for C under the following initialization condition: each C-element $c_j \in L$ has the same value $c_j(v_0)$ as $m_j(v_0)$ in the M-net.*

The proof of the theorem can be found in [9]. Note that there is no need to check a similar condition for v_1, because v_1 must impose a *forcing set* on every C-element along the path, and hence the output of the M-gates is independent of the M-input values.

If the condition for v_0 to be consistent with initialization is violated, then the behavior of an asynchronous net and of the corresponding M-net is different and, in general, the test for the latter is not valid for the former. Assume that the order of the primary inputs for the M-net in Figure 3 is as follows: $< x, Ri, Ai, m_4, m_8 >$. A vector pair $< v_0, v_1 >=< 00011, 00111 >$ is a HFRPDFT for the falling output transition at the path $\pi' = \{Ai, 4, 7, 8\}$ in the M-net. Since v_0 is consistent with initialization ($m_4(v_0) = Ro(v_0) = 1$ and $m_8(v_0) = Ao(v_0) = 1$, a vector pair $< v_0 \downarrow I, v_1 \downarrow I >=< 000, 001 >$ is a HFRPDFT for the falling output transition at the path $\pi = \{Ai, 4, 7, 8\}$ in the asynchronous net. The same path can be tested in the M-net with a vector pair $< v1_0, v1_1 >=< 01001, 01101 >$. However, $v1_0$ is not consistent with initialization because $m_4(v1_0) = 0$, while the output of this M-gate $Ro(v1_0) = 1$. It is easy to check that $< v_0 \downarrow I, v_1 \downarrow I >=< 010, 011 >$ is not a HFRPDFT for the falling output transition along the path $\pi = \{Ai, 4, 7, 8\}$ in the asynchronous net.

4 Initialization conditions

Let $< v_0, v_1 >$ be a HFRPDFT for the path π in the M-net and let v_0 be consistent with initialization. Vector v_0 defines the values of the primary inputs and of the M-inputs. By Theorem 3.3 $< v_0 \downarrow I, v_1 \downarrow I >$ will be a HFRPDFT for the asynchronous net only if the values on the outputs of C-elements $L = \{c_1, \ldots, c_l\}$ coincide with those of the majority gates. These values are given by a ternary vector $\alpha = v_0 \downarrow L$. The aim of the initialization procedure is to set all C-elements according to α.

The proposed monotonous initialization procedure begins from those C-elements that are closer to the primary outputs (in backward topological order). The polynomial bound on the (possibly non-existing) monotonous initialization sequence length is due to the fact that a C-element is not disturbed after having been set.

We can associate four Boolean functions (defined over the space of primary inputs and C-element outputs) with each element g (gate or C-element) of an asynchronous net.

- $S1(g)$ and $S0(g)$ – setting g to 1 or 0 respectively and
- $H1(g)$ and $H0(g)$ – holding g to 1 or 0 respectively.

If g is a basic combinational gate with output function f then $S1(g) = H1(g) = f$ while $S0(g) = H0(g) = \overline{f}$. In case of a C-element, the holding and setting functions are different. For C-element c_j with inputs $i_1, \ldots, i_k$ they are: $S1(c_j) = H1(i_1) * ... * H1(i_k)$ and $H1(c_j) = c_j * (H1(i_1) + ... + H1(i_k))$ (similarly for $S0(c_j)$ and $H0(c_j)$).

To set a C-element c_j we must apply an input vector under which the corresponding setting function evaluates to 1. However only for C-elements of the first level the value of a setting function is completely determined by primary inputs. For c_j in level i the setting function depends also on the outputs of C-elements from the lower levels. Therefore the process of setting c_j may require the recursive setting of "preceding" C-elements.

Let us consider in more detail the process of setting c_j to 1 by using an input vector v (resetting it to 0 can be done similarly). Let C_s denote the C-elements from levels higher than i. If v sets c_j then two conditions must be satisfied:

1. there exists cube $\beta \in S1(c_j)$ such that $\beta \downarrow I$ covers v.
2. C-elements C_β whose outputs have a value of 0 or 1 in β (these are the C-elements on which c_j depends) have already been set by previous input vectors $w_1, \ldots, w_n$.

Application of vector v after w_n can lead to the following difficulties:

- $c_m \in C_\beta$ can change its value inside the transition cube between w_n and v. In such case the value of β in v does not evaluate to 1 and c_j is not set.

- $c_m \in C_s$ can change its value inside the transition cube between w_n and v. In such case the requirement of monotonicity of the initialization procedure is violated.

These two conditions restrict the set of valid vectors v that can be applied after w_n, leading towards v_0, that is the objective of initialization. If we denote by $Hold$ the product of the holding functions of C-elements in $C_s \cup C_\beta$, then a valid transition path between w_n and v must belong to $Hold$. The task of finding a valid path can be reduced to a search in a graph with: (1) vertices corresponding to cubes of $Hold$ and (2) edges between every pair of intersecting cubes from $Hold$.

If no valid path exists, then c_j cannot be set by the cube β and another cube from the setting function of c_j is tried. If we fail to find such a path for all cubes c_j, then we need to backtrack. The procedure converges, because full scan testing is always possible.

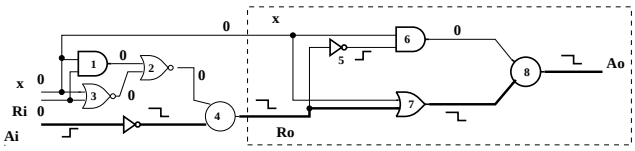

Figure 4: Initialization of asynchronous nets

Return to our example and consider (Figure 4) path $\pi\{Ai, 4, 7, 8, Ao\}$ under the rising transition on Ai. The vector pair $< v_0 = \{x = 0, R_i = 0, A_i = 0\}, v_1 = \{x = 0, R_i = 0, A_i = 1\} >$ will be a HFRPDFT for the falling output transition for π under the condition that Ro and Ao are initialized to 1. Let us find a corresponding initialization sequence. The holding and setting functions for C-elements Ro and Ao are:

$S1(Ro) = \overline{Ai} * (\overline{x}Ri + x\overline{Ri})$

$H1(Ro) = Ro * (\overline{Ai} + \overline{x}Ri + x\overline{Ri})$

$S0(Ro) = Ai * (xRi + \overline{x}\overline{Ri})$

$H0(Ro) = \overline{Ro} * (Ai + xRi + \overline{x}\overline{Ri})$

$S1(Ao) = x * H0(Ro) * (x + H1(Ro))$

$H1(Ao) = Ao * (x * H0(Ro) + x + H1(Ro))$[2]

After substituting of $H1(Ro)$ and $H0(Ro)$ into the functions for Ao we get: $S1(Ao) = x\overline{Ro}Ai + x\overline{Ro}Ri$ and

$H1(Ao) = xAo + AoRo\overline{Ai} + AoRo\overline{x}Ri$

From $S1(Ao)$ it follows that to set Ao at level 2 we first need to reset Ro at level 1. The latter can be done by the vector $w_1 =< Ai = 1, x = 1, Ri = 1 >$. Note that after Ro is reset, the same vector w_1 sets Ao to 1.

The next initialization step is to set Ro. To do this we can try cube $\overline{Ai}\overline{x}Ri$ of function $S1(Ro)$. However, if we apply vector $w_2 =< Ai = 0, x = 0, Ri = 1 >$ after w_1 we cannot keep the value 1 on the output of Ao because $H1(Ao)$ is equal to 0 under w_2 (remember that after w_1, Ro is reset to 0). Therefore no valid path from w_1 to w_2 exists and we need to try the next cube in $S1(Ro)$, that is $\overline{Ai}x\overline{Ri}$. This cube defines vector $w_3 =< Ai = 0, x = 1, Ri = 0 >$ and in any minterm of the transient cube between w_1 and w_3, Ao keeps the value 1 (due to cube xAo of $H1(Ao)$). Therefore, any path inside the transient cube is valid and w_3 can be applied immediately after w_1.

The last task is to check that in the transition from w_3 to v_0 no C-element changes the output. This condition is satisfied, because w_3 and v_1 are adjacent and both belong to $H1(Ao)$ and $H1(Ro)$. Hence, w_1, w_3, v_0 is a valid initialization sequence.

[2] $S0(Ao)$ and $H0(Ao)$ functions are not considered here because in this case Ao is not set to 0.

5 Experimental results

This section presents experimental results which illustrate the RPDFT properties of two classes of asynchronous logic circuits: (1) speed-independent random control logic synthesized from Signal Transition Graph specifications using the so-called Monotonous Cover technique [10]; (2) regular logic from delay-insensitive data-paths [24, 6, 15]. Similar experiments could also be performed with the HFRPDFT model, with obviously lower coverage figures.

The experimental procedure has three main steps:

1. Selecting a set of C-elements to scan.
2. Translating the asynchronous net into an M-net by replacing each C-element with a majority gate. A robust test for each path in the M-net is obtained by test generation of a single stuck-fault in a modified net obtained from the M-net, as described in [9].
3. Generating a sequence of initialization vectors to set the output value of each C-element along the path being tested to the desired value.

As already mentioned, if initialization does not succeed for a chosen test vector pair, additional vector pairs are selected until initialization succeeds for at least one or fails for all.

Circuits from the first class are characterized by a very high density of signal interconnections. We checked four techniques for achieving high-testability: full input scan, full output scan, partial output scan and partial output scan with splitting of primary inputs. Splitting of input connections (that was defined in [12] for true and complemented phases only) is a powerful technique to increase the testability of asynchronous circuits, because it reduces the redundancy level.

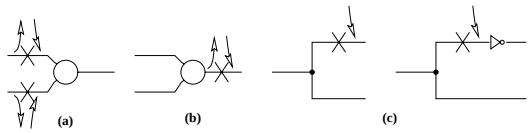

Figure 5: Techniques for testability: input scan (a), output scan (b), fork and phase splitting (c).

As illustrated by Figure 5, input scan requires scan-in and scan-out operations for both inputs of a latch, output scan requires scan-in and scan-out only for the output of the latch. Fork and phase splitting requires scan-in in addition to scan out. Fork splitting means the possibility to scan independently the fanout branches of a fork. Phase splitting means the possibility to drive independently the true and complemented phase of each primary input and sequential element.

Circuit	paths	signals	approach of [12]		partial scan		partial scan with splitting		
			input	output	scan	%	scan	split	%
converta	30	3	100	80	2	64	2	1	71
chu150	22	3	100	91	1	91	1	1	91
chu172	12	3	100	100	0	100	0	0	100
rpdft.map2	36	6	100	97	0	83	0	1	94
alloc-outbound.map2	34	6	100	86	2	86	2	1	97
c3.map2	42	6	100	93	2	71	2	1	83
rcv-setup.map2	20	5	100	100	1	83	1	1	89
master-read-csc.map2	80	16	100	91	5	62	5	2	84
pe-send-ifc-csc	146	5	100	84	4	54	4	2	72
full	16	2	100	50	1	50	1	1	75
dc	36	4	100	89	3	88	3	1	90
qr42	28	3	100	79	1	38	1	1	51
chu133	24	4	100	96	3	89	3	1	96

Table 1: Experimental results for speed-independent control

Table 1 presents the results for speed-independent control logic. The first two columns describe circuit complexity: the number of paths and the number of *non-input* (i.e., feedback) signals of the circuit. There is no column on runtime because for our small circuits test generation always took less then 10 msec. The "approach of [12]" columns show the percent level of testability for input scan and for output scan of all non-input signals respectively, using the technique of [12]. The "partial scan" columns show the level of testability for a selected number of scanned signals for the output scan technique. The last group of columns shows how the level of testability can be increased if the splitting technique is used in addition to partial scan. The column labeled "split" gives the number of split signals. For example, the best level of testability that can be achieved by partial output scan for circuit "converta" (two out of three signals are scanned) is 64%. If one additional signal is split, then testability reaches 71%.

The sets of signals for partial scan and for splitting are selected to achieve the required testability level. In Table 1 we attempt to reach a testability level of 70%, and limit the number of signals which are allowed to be scanned and split as explained in the algorithm Figure 6. For example, we do not allow more than one signal to split for circuit "converta" and to scan more than five signals for circuit "master-read-csc.map2". The requested level of testability cannot be achieved by scanning only five signals in "master-read-csc.map2", but can be achieved by splitting two additional signals.

Our algorithm for selecting a set of signals for partial scan and for signal splitting operates on the directed graph of signal interconnections. It is sketched in Figure 6.

Circuits from the second class are known to have high stuck-at testability. We expected that a high level of path delay fault testability could be achieved with a low scan ratio. Table 2 presents the results for a DIMS-adder [24], for a delay-insensitive adder with a status detector for dual-rail input wires [6] and for a reduced direct logic adder [15]. The numbers are obtained for one bit adders. Since the circuits have no global loops, no scan is required. The last line of the table shows the result for a bit-slice of a serial-parallel carry-save multiplier with pipeline latches

begin
  Select a Minimum Feedback Vertex Set, $Scan$,
  with a maximal sum of vertex degrees among all MFVSs;
  repeat /* Updating scan set */
    Add a vertex with a maximal degree to $Scan$;
    If testability is less than requested **then**
      repeat /* Updating split set */
        Split a signal $g \in Scan$ with a max. out-degree;
        Choose bi-partition of g's fan-out set
        to minimize the number of reconvergent paths;
      until The requested level of testability is achieved
        or the splitting limit is exceeded;
  until The requested level of testability is achieved or
    the scan limit is exceeded;
end

Figure 6: Algorithm

for the two operands, the sum-in and the carry-in, and output latches for the sum-out and the carry-out. To obtain full testability, four feedback wires between the result latches and the operand latches must be scanned. The pipelined adder example is a DIMS-adder incorporated into the pipelined ring.

A few interesting observations can be made:

- Although dependency graphs are very dense for speed-independent random control logic, partial scan augmented with splitting techniques can provide a high level of testability.
- Full *output* scan (which can be achieved by inserting transparent latches after each C-element) provides a relatively good testability level around 80%-90%. This gives, to the best of our knowledge, the first experimental evidence that speed-independent circuits are easily testable even when more accurate fault models than pure output stuck-at are used ([2] first proved their self-checking properties with respect to this kind of fault).
- Experiments with delay-insensitive adders show that the optimization technique used for area efficiency reduces the level of testability from 100% in the DIMS adder to 74% in Martin's adder (which is competitive in

Circuit	Paths	Signals	Testability (%)	Scan (latches)	Scan (% of latches)
DIMS	96	12	100	0	0
DGY	92	18	79	0	0
Martin	84	24	74	0	0
pipelined adder	420	30	100	1	5%
pipelined multiplier	1062	48	100	4	13%

Table 2: Experimental results for delay-insensitive data path

area and faster than a synchronous ripple-carry adder). We may conclude that the optimization transformations which correspond to quasi-delay-insensitive substitution do not retain testability.

6 Conclusions

In this paper we have described a complete path-delay fault testing algorithm for asynchronous *sequential* circuits. We have shown that it is possible to perform such tests by partial scan on a sequential object called an *asynchronous net*. We defined the set of paths that must be tested to check all the timing assumptions. We decomposed the testing problem for sequential circuits into:

1. insertion of enough scan elements to make the asynchronous circuit *functionally acyclic*,
2. initialization (using a heuristic technique, with a fallback strategy for the sake of completeness).
3. test pattern generation, by reduction to combinational ATPG.

Experimental results show that the technique is effective in providing a substantial savings in the number of scan memory elements, versus a small reduction in the testability figures. This is true of control-dominated dense circuits, and even more of regular data path objects.

References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990. Revised printing.

[2] D. B. Armstrong, A. D. Friedman, and P. R. Menon. Design of asynchronous circuits assuming unbounded gate delays. *IEEE Transactions on Computers*, C-18(12):1110–1120, December 1969.

[3] S. Banerjee, S.T. Chakradhar, and R.K. Roy. Synchronous test generation model for asynchronous circuits. In *Proceedings of the Int. Conf. on VLSI Design*, pages 178–185, January 1995.

[4] K.-T. Cheng, V. Agrawal, and E. Kuh. A simulation-based method for generating tests for sequential circuits. *IEEE Transactions on Computers*, 39(12):1456–1463, December 1990.

[5] K.-T. Cheng and V. D. Agrawal. A partial scan method for sequential circuits with feedback. *IEEE Transactions on Computers*, C-39(4):544–548, April 1990.

[6] I. David, R. Ginosar, and M. Yoeli. An efficient implementation of boolean functions as self-timed circuits. *IEEE Transactions on Computers*, 41(1):2–11, January 1992.

[7] S. Devadas and K. Keutzer. Synthesis of robust delay-fault testable circuits: Theory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):87–101, January 1992.

[8] K. Keutzer, L. Lavagno, and A. Sangiovanni-Vincentelli. Synthesis for testability techniques for asynchronous circuits. *IEEE Transactions on Computer-Aided Design*, 14(12):1569–1577, December 1995.

[9] M. Kishinevsky, A. Kondratyev, L. Lavagno, A. Saldanha, and A. Taubin. Hazard free robust path delay fault testing of asynchronous nets. Technical Report TR: 96-2-001, The University of Aizu, Japan, March 1996.

[10] A. Kondratyev, M. Kishinevsky, B. Lin, P. Vanbekbergen, and A. Yakovlev. Basic gate implementation of speed-independent circuits. In *Proceedings of the Design Automation Conference*, pages 56–62, June 1994.

[11] W. Lam, A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli. Delay fault coverage and performance tradeoffs. In *Proc. ACM/IEEE Design Automation Conference*, pages 446–452, June 1993.

[12] L. Lavagno, M. Kishinevsky, and A. Lioy. Testing redundant asynchronous circuits by variable phase splitting. In *Proceedings of the EURO-DAC'94*, pages 328–333, Grenoble, France, September 1994.

[13] L. Lavagno and A. Sangiovanni-Vincentelli. *Algorithms for Synthesis and Testing of Asynchronous Circuits.* Kluwer Academic Publishers, 1993.

[14] D.H. Lee and S.M. Reddy. On determining scan flip-flops in partial-scan designs. In *Proc. International Conf. Computer-Aided Design (ICCAD)*, pages 322–325, November 1990.

[15] A.J. Martin. Asynchronous datapaths and the design of an asynchronous adder. *Formal Methods in System Design*, 1:117–137, 1992.

[16] P. McGeer and R. Brayton. Provably correct critical paths. In *The Proceedings of the Decennial Caltech VLSI Conference*, 1989.

[17] S. Nowick, N. Jha, and F-C. Cheng. Synthesis of asynchronous circuits for stuck-at and robust path delay fault testability. In *Proceedings of the Int. Conf. on VLSI Design*, January 1995.

[18] A. Pramanick and S. Reddy. On the design of path delay fault testable combinational circuits. In *Proceedings of the* 20^{th} *Fault Tolerant Computing Symposium*, pages 374–381, June 1990.

[19] O. Roig, J. Cortadella, M.A. Pe na, and E. Pastor. Automatic generation of synchronous test patterns for asynchronous circuits. In *Proceedings of the 34th Design Automation Conference*, June 1997.

[20] M. Roncken and R. Saeijs. Linear test times for delay-insensitive circuits: a compilation strategy. In S. Furber and M. Edwards, editors, *Proceedings of IFIP Working Conference on Asynchronous Design Methodologies*, pages 13–27, Manchester, UK, 31 March – 2 April 1993, 1993.

[21] A. Saldanha, R. Brayton, and A. Sangiovanni-Vincentelli. Equivalence of robust delay-fault and single stuck-fault test generation. In *Proc. ACM/IEEE Design Automation Conference*, pages 173–176, June 1992.

[22] J. Savir and W.H. Anney. Random pattern testability of delay faults. In *Proceedings of the International Test Conference*, pages 263–273, October 1986.

[23] G. L. Smith. A Model for Delay Faults Based on Paths. In *Proceedings of the Int'l Test Conference*, pages 342–349, September 1985.

[24] Jens Sparsø and Jørgen Staunstrup. Delay-insensitive multi-ring structures. *INTEGRATION, the VLSI Journal*, 15(3):313–340, 1993.