Center for Reliable Computing

TECHNICAL REPORT

Delay Testing for Sequential Circuits with Scan

By Robert B. Norwood and Edward J. McCluskey

97-5

(CSL TR # 97-742)

November 1997

Center for Reliable Computing
Gates 236
Computer Systems Laboratory
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305

Abstract:

Arbitrary two-pattern delay tests are difficult to apply with a standard scan path. However, partitioning the circuit and constraining the scan path order based on this partitioning can result in a scan path where two-pattern tests can be applied. This work presents a technique to identify groups of bistables and form scan paths such that no two bistables from the same group are adjacent in the scan path. Two-pattern tests can then be applied to the various portions of the logic using only bistables from one group to apply any particular two-pattern test.

Funding:

This research was supported in part by the Advanced Research Projects Agency under Contract No. DABT63-94-C-0045, and in part by the National Science Foundation under Grant No. MIP-9107760.

1 Introduction

Current test methodologies typically focus on detecting stuck-at faults. Test pattern generation targets stuck-at faults, and the quality of the test is determined based on the percentage of stuck-at faults detected. Design techniques, such as scan [McCluskey 86], have been developed that allow easy application of stuck-at test patterns to sequential circuits, and high fault coverage for stuck-at faults is attainable.

Recent research [Maxwell 92] [Gayle 93] [Franco 96] shows the need to extend fault models to include delay faults. Franco, et. al., conclude that even for mature processes, timing or pattern dependent defects make up a significant fraction of the defect population. Delay testing should be performed to ensure that parts meet the required performance specifications, or, in other words, that parts can run at the specified clock frequency.

There are many classifications of delay faults, which can generally be broken into two types: gate delay faults and path delay faults. A gate delay fault model [Hsieh 77] assumes that a localized timing failure causes one gate to operate slower than expected. A path delay fault model [Lesser 80] assumes that the timing failure can be distributed along a path, thereby causing the propagation delay along the path to exceed the cycle time. Gate delay faults are a subset of path delay faults.

A test for a delay fault requires the application of two test patterns to create the necessary transitions. The first pattern initializes the logic to a known state. The second pattern activates the targeted fault, causing a transition to propagate along the path under test. The first pattern is applied, and the transients are given time to settle. The second pattern is then applied, and the outputs are sampled after the cycle time. The application of the two test patterns is a *two-pattern test*.

Applying two-pattern tests to combinational logic, and observing the response, is straight forward. Applying delay tests to sequential logic, however, is not as easy.

Simply running the sequential circuit at speed and applying test patterns to the primary inputs will detect some delay faults, but the critical paths are not necessarily tested, and the coverage is typically not sufficient [Barzilai 83]. The sequential circuit can be treated as a combinational circuit during test by inserting a scan path. A scan path turns all the bistables in a circuit into a shift register during test and provides complete controllability and observability to each bistable. A scan path, which greatly simplifies the application of stuck-at test patterns, can be used to apply the two-pattern tests, but some modifications to the scan path must be made, as described here.

This work presents a technique to identify groups of bistables and then form scan paths such that no two bistables from the same group are adjacent in the scan path. The groups are constructed so that bistables from only one group are necessary to test a path for a delay fault. A scan path constructed in this way can be used to apply arbitrary two-pattern tests, as described here. Our technique is generally applicable to any sequential circuit, but the discussion focuses on data path logic with registers.

There are three general techniques for using a scan path to apply two-pattern tests: 1) restricted set of test patterns, 2) modified scan paths, 3) constrained scan paths. Each of these techniques is described in more detail in this section. Our proposed technique is a combination of the modified and the constrained scan path approaches.

1.1 Restricted set of test patterns

By restricting the set of two-pattern tests that can be used to detect the delay faults, the patterns can be applied with a scan path. There are two common ways in which the set of patterns can be restricted. First, the first pattern, used to initialize the logic, can be restricted so that the second pattern, used to activate the delay fault, can be generated by the functional logic with the first pattern as stimulus. With this approach, the first pattern is scanned into the bistables via the scan path. The second pattern is then loaded from the functional logic outputs into the normal bistable inputs. Any arbitrary first pattern can be scanned in, but the second pattern is greatly restricted based on the functional logic. Test

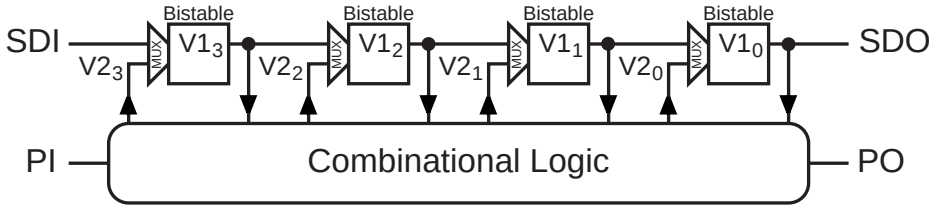


Figure 1. Restricted second pattern (*V2$_3$*, *V2$_2$*, *V2$_1$*, *V2$_0$*) from the functional logic

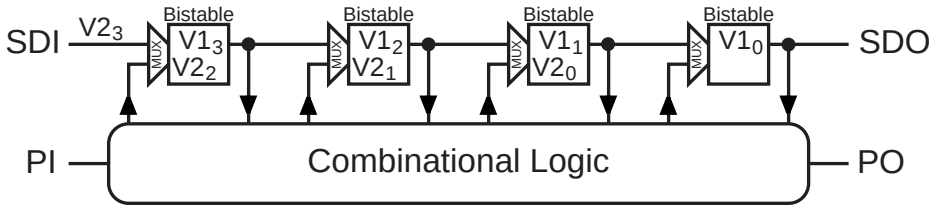


Figure 2. Restricted second pattern (*V2$_3$*, *V2$_2$*, *V2$_1$*, *V2$_0$*) from the scan path

pattern generation for this technique can be complex. Figure 1 illustrates how the two patterns are applied. (Unless otherwise noted, multiplexers are controlled by the test enable signal.) The first test vector (*V1*), is scanned into the bistables via the scan path. The second vector (*V2*), is generated by the functional logic in response to the first vector and primary inputs. This second vector can be loaded into the bistables by the application of a single system clock cycle.

The other approach is to restrict the second pattern to be a single bit shift of the first pattern [Savir 92]. The first pattern is scanned in and the second pattern can be obtained from the first pattern with the application of a single scan clock cycle. The set of second patterns is greatly restricted since it must be a shifted version of the first pattern. Figure 2 shows how this approach works. The first vector (*V1*) is scanned into the bistables. The second vector (*V2*) also happens to be scanned in since the second bit of the first vector ($V1_1$) is equal to the first bit of the second vector ($V2_0$), and so on for the other bits. The last bit of the second vector is ready to be scanned in via the scan data input.

Both of these techniques allow the application of two-pattern tests, but the set of patterns is greatly restricted. A test for a particular delay fault may not be possible since

the necessary sequence of vectors can not be applied. The test pattern generation can also be more complex than when the set of patterns is not restricted.

1.2 Modified scan paths

Arbitrary two-pattern tests can be applied if the bistables themselves are modified. For example, an additional latch can be added to each storage element to allow it to hold two bits of data — one bit that is being applied to the functional logic, and one bit that is being scanned through the scan path. This technique is typically called *enhanced-scan* [Malaiya 83] [Glover 88]. The first pattern can be scanned in and applied to the functional logic. The second pattern can then be scanned in, while the first pattern is still being applied to the functional logic. The second pattern can then be applied to the functional logic. In this way, any arbitrary two-pattern test can be applied. Figure 3 illustrates enhanced-scan. Each bistable holds two values, both vector one (*V1*) and vector two (*V2*). The area overhead of the modified bistables can be high. Partial enhanced-scan [Cheng 91], where only a subset of the bistables are enhanced, can be used to reduce the area overhead, but the set of two-pattern tests that can be applied is reduced since the second pattern must be partially generated from functional logic.

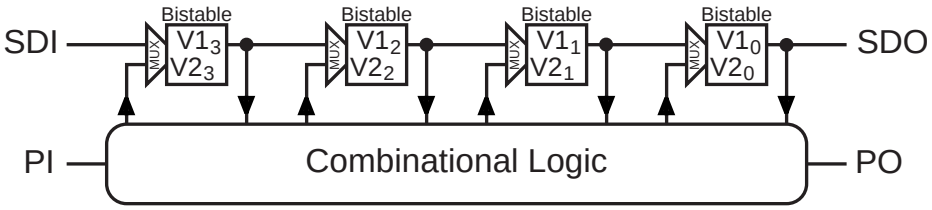


Figure 3. Enhanced-scan

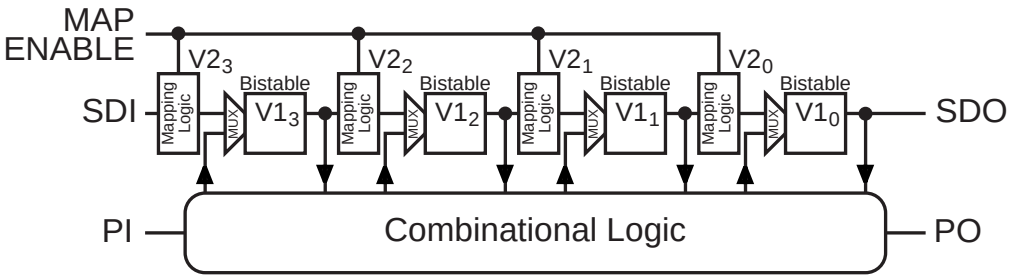


Figure 4. Mapping Logic

The scan path can also be modified with the addition of mapping logic [Touba 96]. Logic is added to the scan path such that the second pattern can be generated from the first pattern. The mapping logic can be designed so that an arbitrary second pattern may be generated, but additional logic is needed to perform the mapping. Figure 4 illustrates the concept. The first vector (*V1*) is scanned into the scan path, and the second vector (*V2*) is obtained by asserting the map enable signal for a single clock cycle.

1.3 Constrained scan paths

The third technique allows the application of arbitrary two-pattern tests by constraining the scan path. The order of the bistables in the scan path can be constrained based on many different criteria, such as the delay fault coverage provided by particular orderings [Mao 90] or the overlapping cones of logic associated with the bistable inputs.

Using the cones of logic, the bistables in the circuit are grouped such that, in order to test any particular delay fault, only bistables from one group are needed. The bistables from the different groups can then be interleaved in the scan path so that no two bistables from the same group are adjacent (next to each other) in the scan path. The first pattern is scanned into the active bistables. A bistable is *active* during a particular set of delay patterns if it is needed to apply either the first or the second pattern. At the same time that the first pattern in scanned into the active bistables, the second pattern is scanned into the inactive bistables that directly precede the active bistables in the scan path. The second pattern can then be scanned into the active bistables with a single scan clock cycle. Arbitrary two-pattern tests can be applied in this manner, but only if there exists an appropriate grouping and ordering of the bistables. Figure 5 illustrates this concept. There are two groups. The first group contains the second and fourth bistables. The other group contains the first and third bistables. Only bistables from the first group are needed to test for delay faults in the first block of combinational logic. Only bistables from the second group are needed to test the second block. In order to test the first block of combinational logic, the test vector is scanned in so that bistables two and four contain

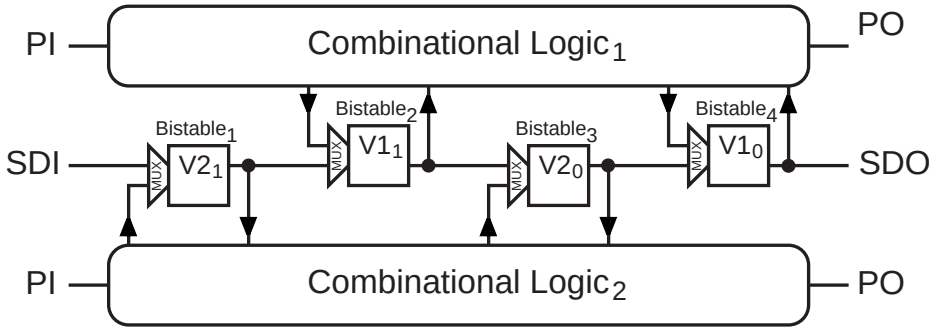


Figure 5. Constrained scan path

the first vector (*V1*) and bistables one and three contain the second vector (*V2*). The first vector is applied to the combinational logic, and then the second vector can be applied with a single scan clock cycle. A similar process is used to test the other block of combinational logic. This approach has been applied to multiple finite state machines, where the bistables from each finite state machine are put into a separate group [Hurst 95].

2 Proposed approach

This work presents a technique that builds on both the modified scan and the constrained scan approaches. The bistables are grouped together using the criteria to be discussed in Sec. 3. The scan path is then ordered based on the groupings, as described in Sec. 4. Specific bistables may need to be modified, as previously discussed in Sec. 1.2, in order to facilitate the application of arbitrary two-pattern tests and improve the fault coverage.

Out technique can be applied to both register-based designs, such as data path logic, and to flip-flop based designs, such as control logic. However, this discussion focuses on register based designs because the regular structure allows the bistables to be grouped as registers instead of individually. The procedure can be directly applied to non-register based circuits by treating the circuit as a one bit wide data path and the bistables as one-bit registers.

The structure of data path logic allows the grouping to be done at a higher level, possibly making use of synthesis information. The registers are grouped so that the data path logic can be tested for delay faults with multiple test sessions, where each test session tests a subset of the data path logic using registers from only one register group. The registers used during a particular test session are active for that session. The registers are formed into multiple, parallel scan paths, much like orthogonal scan [Avra 92] [Norwood 96], such that no two bistables from registers in the same group are adjacent in any of the scan paths, and only registers from one group are used to apply a particular two-pattern test. Two bistables (or registers) are *adjacent* if one is connected to the other in the scan path. In this way, any two adjacent bistables in the final scan path are from different register groups.

Figure 6 illustrates our technique. Figure 6a shows a data path fragment with three registers. There are two register groups with the first group containing registers *1* and *2*, and the other group containing register *3*. There is no direct access to the data path through primary inputs, so test patterns must be applied through the scan path. In order to test for delay faults in the paths through the adder, two-pattern tests must be applied via registers *1* and *2*, — the first register group. The scan path configuration is represented by the highlighted path in Fig. 6a and is shown in more detail in Fig. 6d. There are *n* parallel scan paths, where *n* is the width of the data path. Each scan path is composed of a bit slice of the data path. In other words, one scan path contains bit one of register *1*, bit one of register *2* and bit one of register *3*. Another scan path contains bit two of register *1*, bit two of register *2* and bit two of register *3*. In this way, the register ordering for the bistables in each scan path is the same, and the multiple, parallel scan paths can be treated as a single, *n*-bit wide scan path composed of registers. In this work, unless otherwise noted, the scan path will be treated as a single scan path composed of registers, but the actual implementation is multiple, parallel scan paths composed of the individual bistables in each register.

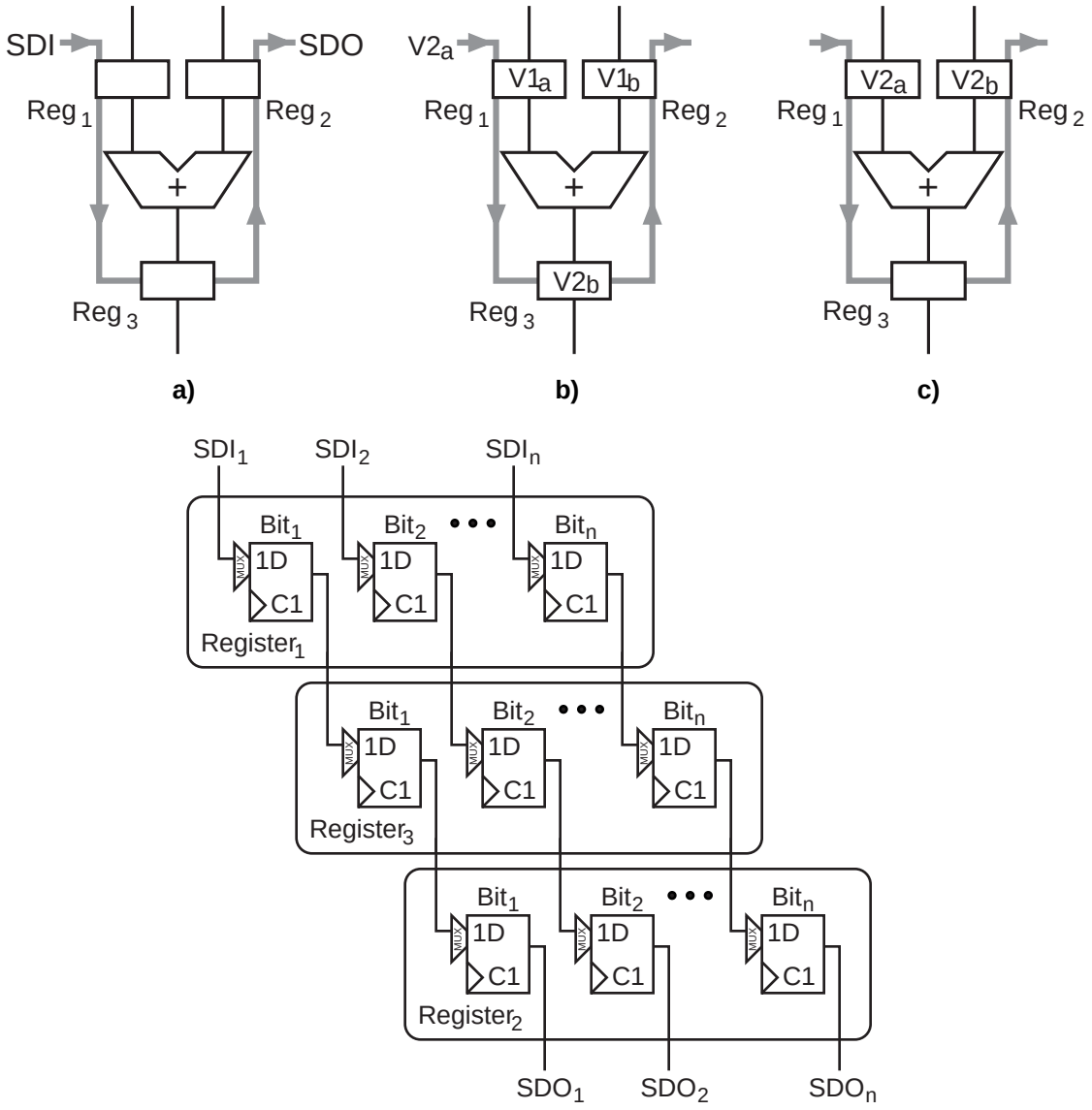


Figure 6. Scan path for delay testing: a) data path fragment; b) applying first pattern; c) applying second pattern; d) multiple scan paths

The scan path order shown in Fig. 6a ($1 \rightarrow 3 \rightarrow 2$), in which no two registers from the same group are adjacent, allows a test vector to be scanned in so that registers *1* and *2* are active and together contain the first pattern, *V1*, needed for the delay test. Register *3* contains part of the second pattern, *V2*, and the rest of the second pattern is ready to be scanned in via the scan data inputs. This situation is shown in Fig. 6b. The first pattern from the active registers, registers *1* and *2*, initializes the combinational logic. A single

scan clock cycle will then load registers *1* and *2* with the second pattern from the inactive registers, and the delay test is implemented. Figure 6c shows the application of the second pattern. The response can be captured in register *3* with a single system clock cycle and can be scanned out through the scan path while the next set of vectors are scanned in.

In this way, the scan path can be used to apply the two-pattern test. We discuss how to construct such a scan path. There are three steps to the procedure:

1. Group the registers into register groups so that only registers from one group are required to apply a two-pattern test to test a particular path for a delay fault.
2. Determine the scan path order so that no two bistables from registers in the same group are adjacent, or, equivalently, so that any two adjacent bistables are not from registers in the same group.
3. Create the final scan path so that the adjacency (non-adjacency) property from step 2 is preserved.

This paper is organized in the following manner: Section 3 discusses methods by which to group the registers. Section 4 covers determining the scan path order, and Sec. 5 shows how to create the final scan path. Section 6 discusses how to use the scan path to apply the delay tests. Some results for the technique are given in Sec. 7.

3 Grouping the registers

The first step in the procedure is forming the registers into groups such that only the registers in one group are used to apply a two-pattern test to test a portion of the data path logic for delay faults, and every part of the data path logic can be tested by the registers in at least one of the groups. The registers are formed into groups by examining the combinational logic between the registers. The register grouping can be performed in a number of ways, depending on how the combinational logic is analyzed. Several methods are discussed here.

3.1 Grouping based on register input support set

An obvious way to group the registers is by looking at the input cones of logic for each register. All the registers whose outputs feed into a particular register input through combinational logic are called the *support set* of that register and are put into the same group. In this way, a data path with *n* registers has *n* groups. For example, using the data path fragment in Fig. 7 (multiplexer select signals come from the control logic and are not shown), registers *1* through *5* support register *6*. There would be one group formed, {*1*, *2*, *3*, *4*, *5*}. All the combinational logic between the registers in the group and the destination register can be tested for delay faults by applying two-pattern tests with the registers in the group since the registers control all the inputs to the combinational logic.

This approach tends to create register groupings with many common registers because each register typically has many registers supporting it. Finding an ordering for the scan path such that no two registers from the same group are adjacent may be impossible without modifying some of the registers, as discussed in Sec. 5, and, therefore, grouping based on register input support sets is not very effective.

3.2 Grouping based on functional input support set

Smaller groups may be obtained by determining the register grouping based on functional unit inputs, rather than on register inputs. As shown in Sec. 4, smaller groups

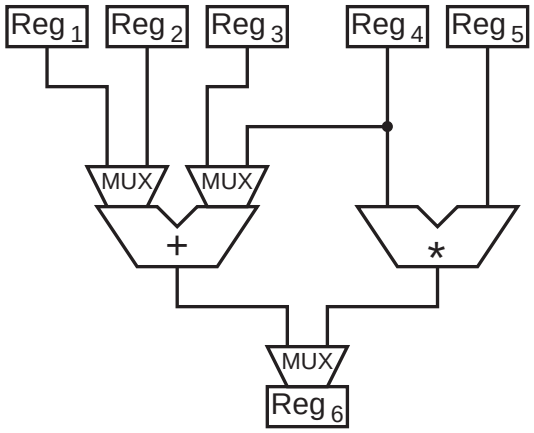
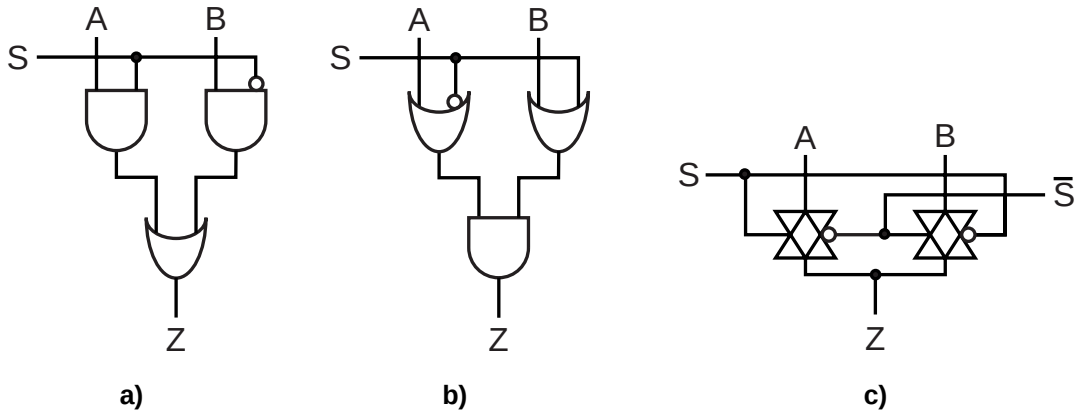


Figure 7. Data path fragment

are preferred because they put fewer constraints on the scan path ordering. For each functional unit, all *possible operations* are determined by taking each function that can be performed by the functional unit and the various registers that can be used as the operands of each function and enumerating the combinations. The registers used for each operation form a group. For example, again using the data path fragment in Fig. 7, the adder can perform four possible operations: *1 + 3*, *1 + 4*, *2 + 3* and *2 + 4*. The multiplier can perform only one operation: *4 * 5*. These five possible operations result in five register groups: {*1, 3*}, {*1, 4*}, {*2, 3*}, {*2, 4*} and {*4, 5*}. This approach will create many groups, but each group will have only a few registers in it, based on the number of inputs to the functional units. Each functional unit can be tested for delay faults using the registers in a few of the groups.

These groupings allow the entire data path to be delay tested piece by piece through the application of two-pattern tests. For example, the paths through the multiplier can be delay tested by applying two-pattern tests to registers *4* and *5* (the fifth group) and capturing the result in register *6*. The adder can be delay tested in a similar fashion. Any particular path in the adder can be tested using one of the first four register groups. The multiplexers can also be delay tested, even if only one data input at a time can have two-pattern tests applied. Figure 8 shows three possible multiplexer implementations. Figures 8a and 8b use logic gates, and Fig. 8c uses transmission gates. Each multiplexer data input can be delay tested individually by holding the select signal at a constant value and applying a two-pattern test to the input being tested. For example, by holding the select signal high and applying a two-pattern test to input *A*, the path through input *A* to the multiplexer output can be delay tested. The path through input *B* can be tested in a similar fashion by holding the select signal low. Only one data input at a time must be capable of having two-pattern tests applied. The path through the multiplexer select, *S*, can be delay tested by holding all the data inputs at a constant value and applying a two-



a) **b)** **c)**

Figure 8. Multiplexer implementations: a) AND-OR; b) OR-AND; c) transmission gates

pattern test to the select. In this way, all the combinational logic can be delay tested through the application of two-pattern tests.

3.3 Grouping based on high-level data flow

The number of groupings can be further reduced by considering only paths that will actually be used during functional operation. Using information from the data flow graph or other high level information, groups are formed only from combinations of registers that are actually used during functional operation. For example, based on knowledge that the adder is only used for *1 + 3* and *2 + 4*, and the multiplier is used for *4 * 5*, the groups formed would be: {*1, 3*}, {*2, 4*} and {*4, 5*}, with {*1, 4*} and {*2, 3*} being removed since the adder is never used to perform these operations. Fewer groups put fewer constraints on the scan path ordering, as shown in Sec. 4. This approach will reduce the constraints on the scan path order, but delay faults that do not affect functional operation will not be detected. If these delay faults must be detected, one of the register grouping techniques described in Sec. 3.1 or 3.2 should be used.

4 Determining the scan path order

Once the registers have been grouped, the scan path order must be determined so that any two adjacent bistables are not from registers in the same group. A *compatibility graph* is generated from the groupings to help determine the scan path order. A node is

created for each register, and an edge is added between two nodes if the corresponding registers are never both present in the same group. Compatibility graphs are shown in Fig. 9 for the three groupings in Sections 3.1, 3.2, and 3.3. The compatibility graph shows which registers may be adjacent in the scan path.

The complement of the compatibility graph, which is a *conflict graph*, shows the grouping between registers. The conflict graph is generated by creating a node for each register and adding edges between two nodes if the corresponding registers are in the same group. The conflict graphs for the previous three groupings are shown in Fig. 10. The conflict graph shows which registers can not be adjacent in the scan path.

From the compatibility graph, paths including all the registers can be found such that no two registers from the same group are adjacent in the scan path. If it is not possible to include all the registers in the path, then the bistables in the registers that can not be added to the scan path must be modified, with enhanced-scan or a shadow register for example, so as to be capable of holding two values at one time. The registers can then be

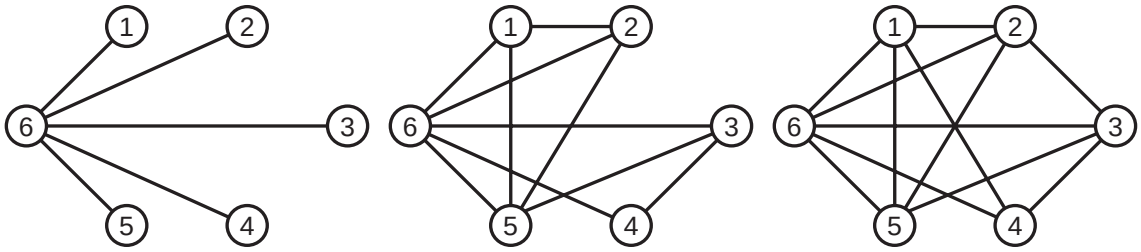


Figure 9. Compatibility graphs: a) based on register inputs; b) based on functional units; c) based on high-level information

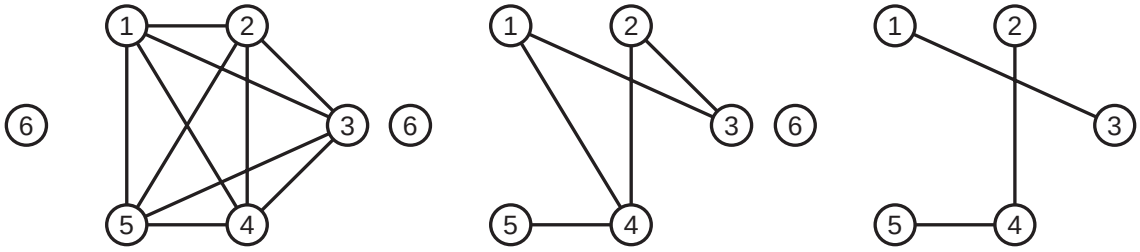


Figure 10. Conflict graphs: a) based on register inputs; b) based on functional units; c) based on high-level information

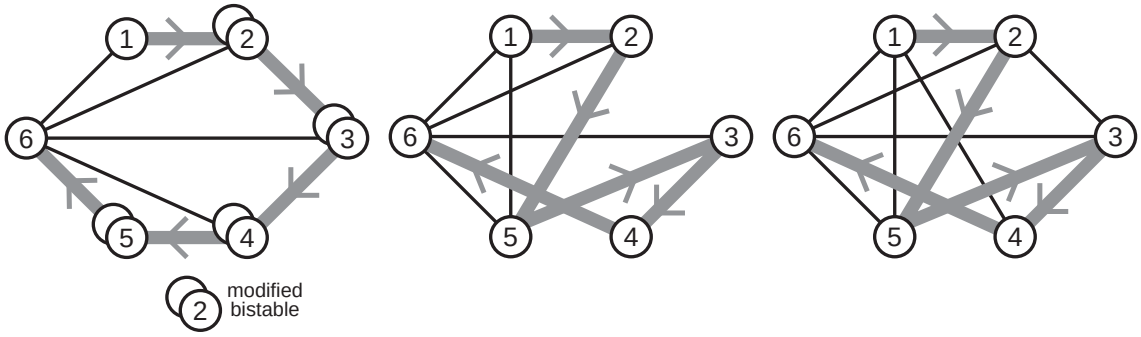


Figure 11. Scan path orders: a) based on register inputs; b) based on functional units; c) based on high-level information

added to the scan path without losing the ability to apply arbitrary two-pattern tests. Figure 11 shows the compatibility graphs with a possible scan path order highlighted. Figure 11a illustrates the need for modified registers because of the many conflicts between registers.

5 Creating the scan path

When the order has been determined, the scan path is created to preserve the property that no two bistables from registers in the same register group are adjacent. The most straightforward approach is to use multiple, parallel scan paths that follow the bit slices of the data path. This is the approach discussed in Sec. 2 and assumed for most of the discussion. The scan path can also be created as a single scan path by interleaving the individual bistables instead of interleaving the registers. Both methods are discussed below.

5.1 Multiple, parallel scan paths

Multiple, parallel scan paths are easily constructed from the register order determined from the compatibility graph. For a data path that is *n* bits wide, *n* scan paths will be created. Each scan path uses the register order to order the bistables in the scan path. Therefore, every scan path has the same order but is composed of bistables from different

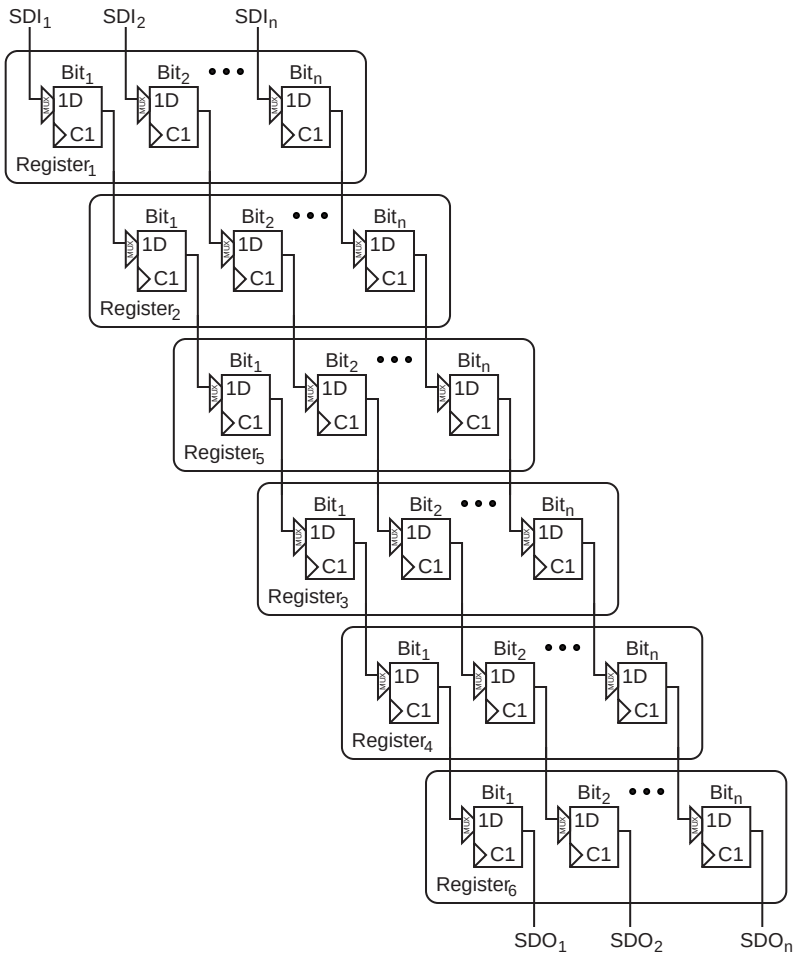


Figure 12. Multiple, parallel scan paths using register order from Fig. 11b

bit slices. As an example, using the register order shown in Fig. 11b from Sec. 3.2 ($1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 6$) for an *n*-bit wide data path, the scan path configuration would look like that shown in Fig. 12. There are *n* scan paths. One scan path for bit position one, one scan path for bit position two and so forth for all *n* bit positions. The bistables within the different scan paths all have the same order. The first bistable is from register *1*. The second bistable is from register *2*. The next to last bistable is from register *4*. The last bistable is from register *6*.

Some register orderings may require that the bistables of certain registers be enhanced so that the two-patterns may be applied. For example, the register order shown in Fig. 11a does not ensure the property that no two registers from the same group may be

adjacent. Registers *2*, *3*, *4* and *5* must be enhanced in order to have a scan path that allows two-pattern tests to be applied. Scan paths constructed in this manner allow the application of two-pattern tests since no two bistables from registers in the same group are adjacent unless the bistable is enhanced to hold two values.

5.2 Single scan path

The register order can also be used to construct a single scan path that allows for the application of two-pattern tests. The individual bistables are interleaved in a single scan path instead of interleaving the registers using multiple scan paths. Figure 13 shows a single scan path, using the register order from Fig. 11b ($1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 6$) and a two-bit wide data path, such that no two bistables from registers in the same group are adjacent. The individual bistables must be interleaved, as opposed to just serially connecting the bits within the registers and then serially connecting the registers, otherwise bistables that are in the same register group will be adjacent in the scan path.

The bistable order is obtained from the register order in the compatibility graph by interleaving the bistables from consecutive registers. As shown in Fig. 13, the first four bistables all belong to either register *1* or register *2*. The next four bistables all belong to either register *5* or register *3*. The final four bistables all belong to either register *4* or *6*. Within each set of four bistables, adjacent bistables belong to different registers. With this configuration, no two adjacent bistables belong to registers of the same group. As mentioned in Sec. 5.1, some bistables may need to be enhanced to ensure the required properties.

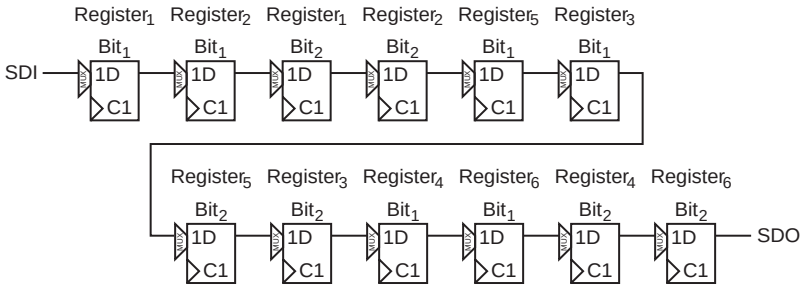


Figure 13. Single scan path using register order from Fig. 11b

Either multiple, parallel scan paths or a single scan path can be constructed from the register order. These scan paths allow two-pattern tests to be applied to detect delay faults.

6 Applying the delay tests

A two-pattern test is applied to the data path through the scan path as follows:

1) Determine the portion of the data path to be tested and the associated registers. These will be the active registers.
2) Determine the two-pattern test.
3) Set the test enable signal high.
4) Scan the first pattern into the active registers and the second pattern into the registers preceding the active registers.
5) Let any transients in the circuit settle.
6) Apply a single clock cycle to scan the second pattern into the active registers.
7) Set the test enable signal low.
8) Apply a single clock cycle to capture the response in the registers.
9) Scan out the response while scanning in the next set of test patterns.
10) Repeat as required.

For example, in order to test the multiplier in Fig. 7, using the scan path order shown in Fig. 11b (*1* $\rightarrow$ *2* $\rightarrow$ *5* $\rightarrow$ *3* $\rightarrow$ *4* $\rightarrow$ *6*), the scan path would look like Fig. 14a after step 4. The first vector is in registers *4* and *5*, and the second vector is in registers *2* and *3*. After step 6 the scan path would look like Fig. 14b with the second vector in registers *4* and *5*. After step 8 the scan path would look like Fig. 14c with the result captured in register *6*.

7 Results for implementations

TOPS, Stanford CRC's high-level synthesis-for-test tool, has been modified to group the data path's registers and create a scan path to allow the application of two-pattern

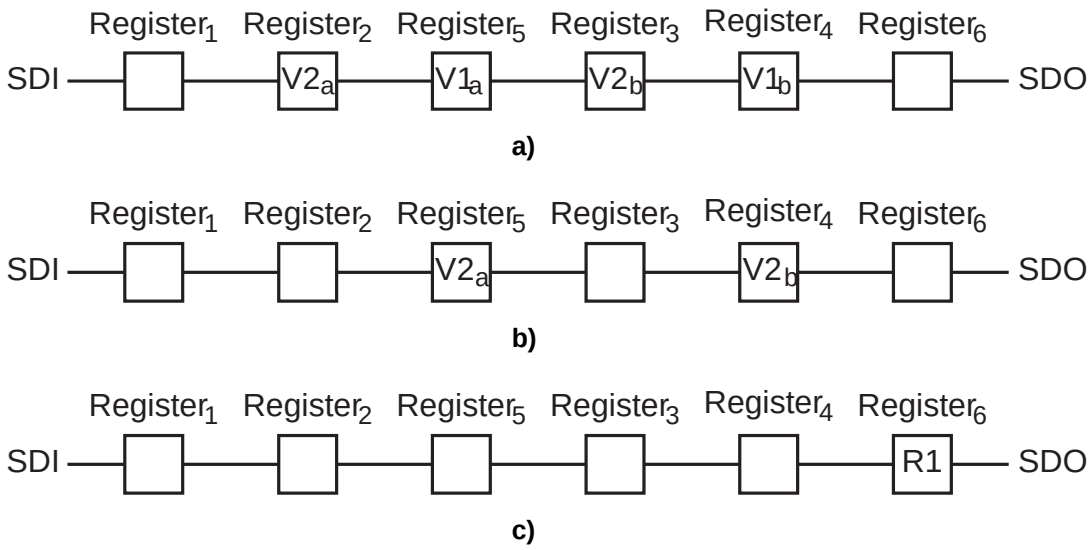


Figure 14. Scan path contents for delay test of Fig. 7 multiplier: a) apply first vector (*V1*); b) apply second vector (*V2*); c) capture results (*R1*)

tests. *Sis* has been modified to group bistables and order the scan path for non-register based circuits. The results for some benchmark circuits are shown in Table 1 and Table 2.

Table 1 shows the results for twenty-seven random logic circuits from the IWLS91 benchmark circuits [Yang 91]. The number of bistables in the circuit and the number of bistables that must be modified to form the scan path is shown. Ten of the circuits require no modified bistables, and fifteen of them require less than twenty percent of the bistables to be modified.

Table 2 shows the results for four high-level synthesis benchmark circuits [Dutt 92] [Tseng 86]. Table 2 includes the number of registers in the data path circuits and the number of registers that would need to be modified to create the scan path. The table also shows the number of conflict edges in the conflict graph. This number gives some indication of the constraints put on the scan path by the particular register grouping. Results are shown for the three different grouping strategies discussed in Sec. 3.

As the number of conflict edges shows, the constraints on the scan path order are greatest when the grouping is done according to the register input support sets. In this

Table 1. Constrained scan path for control logic

Circuit	# Flip-flops	# Modified	% Modified
bigkey	224	0	0
dsip	224	0	0
mm4a	12	0	0
mult16b	30	0	0
mult32b	62	0	0
s1196	18	0	0
s1423	74	27	36
s1488	6	5	83
s1494	6	5	83
s208.1	8	5	63
s27	3	0	0
s298	14	1	7
s344	15	7	47
s349	15	7	47
s382	21	4	19
s386	6	5	83
s400	21	4	19
s420.1	16	13	81
s444	21	4	19
s510	6	4	67
s526	21	4	19
s641	19	0	0
s713	19	0	0
s820	5	4	80
s832	5	4	80
s838.1	32	29	91
sbc	28	0	0

case, many of the registers must be enhanced in order for two-pattern tests to be applied. Using the functional unit inputs or the high-level information to group the registers reduces the constraints and allows for a scan path order that permits the application of

Table 2. Results for various benchmark circuits

Circuit	# Registers	Grouping based on register inputs		Grouping based on functional unit input		Grouping based on high-level info	
		# Modified Registers	# Conflict Edges	# Modified Registers	# Conflict Edges	# Modified Registers	# Conflict Edges
diffeq	7	4	15	0	11	0	7
ellipf	12	9	61	0	29	0	14
gcd	2	1	1	1	1	1	1
tseng	5	2	7	0	6	0	6

two-pattern tests.

8 Summary

Arbitrary two-pattern delay tests are difficult to apply with a standard scan path. However, partitioning the circuit and constraining the scan path order based on this partitioning can result in a scan path such that two-pattern tests can be applied. This work has discussed a technique to group the registers in a data path so that the registers can be formed into a scan path so that no two registers from the same group are adjacent in the scan path. Two-pattern tests can then be applied to the various portions of the data path logic.

The number and size of the register groups directly affect the constraints put on the scan path order — the larger the groups, the greater the constraints. Several register grouping techniques are presented, each forming register groups in different ways and putting different constraints on the final scan path. Scan paths created in this manner require very few, or no, enhanced bistables and allow the application of arbitrary two-pattern tests.

This technique may also be used for non-register based designs. The bistables are grouped together instead of the registers, and the scan path is ordered so that no two bistables from the same group are adjacent. The procedure can be directly applied to non-register based circuits by treating the circuit as a one-bit wide data path and the bistables as one-bit registers.

Acknowledgments

This work was supported in part by the Advanced Research Projects Agency under Contract No. DABT63-94-C-0045, and in part by the National Science Foundation under Grant No. MIP-9107760.

References

[Avra 92] Avra, L., "Orthogonal Built-In Self-Test," *COMPCON Spring 1992 Dig. of Papers*, San Francisco, CA, pp. 452-457, February 24-28, 1992.

[Barzilai 83] Barzilai, Z., and B.K. Rosen, "Comparison of AC Self-Testing Procedures," *Proc. Intl. Test Conf.*, pp. 89-94, Oct., 1983.

[Cheng 91] Cheng, K., S. Devadas, and K. Keutzer, "A Partial Enhanced-Scan Approach to Robust Delay Fault Test Generation for Sequential Circuits," *Proc. Intl. Test Conf.*, Nashville, TN, pp. 403-410, Oct. 29-Nov. 1, 1991.

[Dutt 92] Dutt, N., and C. Ramchandran, "Benchmarks for the 1992 High Level Synthesis Workshop," Technical Report 92-107, University of California, Irvine.

[Franco 96] Franco, P., S. Ma, J. Chang, Y.-C. Chu, et. al., "Analysis and Detection of Timing Failures in an Experimental Test Chip," *Proc. Intl. Test Conf.*, Washington, DC, pp. 691-700, Oct. 21-24, 1996.

[Gayle 93] Gayle, R., "The Cost of Quality: Reducing ASIC Defects with IDDQ, At-Speed Testing, and Increased Fault Coverage," *Proc. Intl. Test Conf.*, Baltimore, MD, pp. 285-292, Oct. 17-21, 1993.

[Glover 88] Glover, C.T., and M.R. Mercer, "A Method of Delay Fault Test Generation," *Proc. ACM/IEEE Design Automation Conf.*, pp. 90-95, June, 1988.

[Hsieh 77] Hsieh, E.P., R.A. Rasmussen, L.J. Vidunas, and W.T. Davis, "Delay Test Generation," *Proc. 14th Design Automation Conf.*, pp. 486-491, June, 1977.

[Hurst 95] Hurst, J.P., and N. Kanopoulos, "Flip-Flop Sharing in Standard Scan Path to Enhance Delay Fault Testing of Sequential Circuits," *Proc. 4th Asian Test Symposium*, Bangalore, India, pp. 346-352, Nov. 23-24, 1995.

[Lesser 80] Lesser, J.D., and J.J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," *IEEE Trans. Computers*, Vol. C-29, No. 3, pp. 235-248, Mar. 1980.

[Malaiya 83] Malaiya, Y.K., and R. Narayanaswamy, "Testing for Timing Faults in Synchronous Sequential Integrated Circuits," *Proc. Intl. Test Conf.*, pp. 560-571, Oct., 1983.

[Mao 90] Mao, W., and M.D. Ciletti, "Arrangement of Latches in Scan-Path Design to Improve Delay Fault Coverage," *Proc. Intl. Test Conf.*, Washington, DC, pp. 387-393, Sept. 10-12, 1990.

[Maxwell 92] Maxwell, P.C., R.C. Aitken, V. Johansen, and I. Chaing, "The Effectiveness of I_{DDQ}, Functional and Scan Tests: How Many Fault Coverages Do We Need?," *Proc. Intl. Test Conf.*, pp. 168-177, 1992.

[McCluskey 86] McCluskey, E.J., *Logic Design Principles*, Prentice-Hall, Englewood Cliffs, NJ, 1986.

[Norwood 96] Norwood, R.B., and E.J. McCluskey, "Orthogonal Scan: Low Overhead Scan for Data Paths," *Proc. Intl. Test Conf.*, Washington, DC, pp. 659-668, Oct. 21-24, 1996.

[Savir 92] Savir, J., "Skewed-Load Transition Test: Part I, Calculus," *Proc. Int. Test Conf.*, Baltimore, MD, pp. 705-713, Sept. 20-24, 1992.

[Touba 96] Touba, N.A., and E.J. McCluskey, "Applying Two-Pattern Tests Using Scan-Mapping," *Proc. VLSI Test Symp.*, Princeton, NJ, pp. 393-397, April 28-May 1, 1996.

[Tseng 86] Tseng, C.-J., and D.P. Siewiorek, "Automated Synthesis of Data Paths in Digital Systems," *IEEE Trans. Computer-Aided Design*, Vol. CAD-5, No. 3, pp. 379-395, July, 1986.

[Yang 91] Yang, S., "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Distributed as part of the IWLS91 benchmark distribution.