Center for Reliable Computing

TECHNICAL NOTE

Preprint

Norwood, R.B., and E.J. McCluskey, "High-Level Synthesis for Scan."

96-4 (CSL TN # 96-406) November 1996	**Center for Reliable Computing** Gates 2A Computer Systems Laboratory Departments of Electrical Engineering and Computer Science Stanford University Stanford, California 94305-9020

Abstract:

This Technical Note contains a preprint of a paper submitted to the 15th IEEE VLSI Test Symposium to be held on April 27-30, 1997 at Monterey, CA.

Funding:

This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant No. N00014-92-J-1782, by the National Science Foundation under Grant No. MIP-9107760, and by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045.

High-Level Synthesis for Scan

Robert B. Norwood and Edward J. McCluskey

CENTER FOR RELIABLE COMPUTING
Stanford University
Gates 236, MC 9020
Gates Building 2A
Stanford, California 94305

Telephone: (415) 723-1258
FAX: (415) 725-7398
E-mail: norwood@shasta.stanford.edu

Designated contact person and presenter: Robert B. Norwood

Topics: Test synthesis, Scan

Short Abstract:

Scan paths are commonly used in digital design to improve the testability of sequential circuits since a full scan path provides complete controllability and observability for every bistable element. A traditional scan path is implemented after the circuit has been designed, with out regard to the actual circuit function. High-level synthesis can exploit knowledge of the circuit function to synthesize a scannable circuit that has less area overhead than a circuit that has scan inserted after synthesis. In this paper, we discuss how synthesis algorithms that target *orthogonal scan* can result in final designs that are fully scanned and have about one-third the overhead of a traditional scan path.

1 INTRODUCTION

Scan paths are commonly used in digital design to improve the testability of sequential circuits. A full scan path provides complete controllability and observability for every bistable element, allowing the sequential circuit to be treated much like a combinational circuit during test. There are many varieties of scan paths used [Eichelberger 77] [Williams 83], each with its own set of advantages and disadvantages, and each adding overhead of some sort to the circuit [McCluskey 86]. A traditional scan path, regardless of its specific implementation details, is implemented independent of the actual circuit function. Bistables are modified and connected to create the scan path with little, or no, knowledge of the exact circuit function. We show that knowledge of the circuit function can be exploited during scan path insertion resulting in a final design that has less overhead than when the scan path is inserted without such knowledge. Previous work has shown this benefit for control logic [Norwood 96a]. Other work has looked at using the controllability of the primary inputs to sensitize certain paths for scan and thereby reuse the existing functional logic for scan [Lin 95]. This paper discusses how high-level synthesis for data paths can target orthogonal scan [Norwood 96b] [Avra 92] and how the resulting designs have less overhead than circuits that have orthogonal scan inserted after design. Synthesis for orthogonal scan also results in fully scannable designs that are smaller than designs with traditional scan.

Orthogonal scan is a full scan implementation for data paths that makes use of the existing functional interconnect to implement the scan path. The control logic must be scanned in some other manner. The data flow during orthogonal scan is parallel to the data flow during functional operation, and this parallelism is exploited to reduce the length of the orthogonal scan path.

Section 2 gives an overview of orthogonal scan, and Sec. 3 describes high-level synthesis. Section 4 presents modifications to the high-level synthesis algorithms to target orthogonal scan. Section 5 gives results for the techniques discussed.

2 ORTHOGONAL SCAN

2.1 Orthogonal Scan Implementation

Traditional scan paths, shown in Fig. 1, connect individual flip-flops within a register and then connect the registers. For example, bit one of register one is connected to bit two of register one, and bit two is connected to bit three of register one, and so on until the last bit of register one is connected to bit one of register two. A traditional scan path requires a multiplexer, or multiplexer equivalent, for each bit of every register.

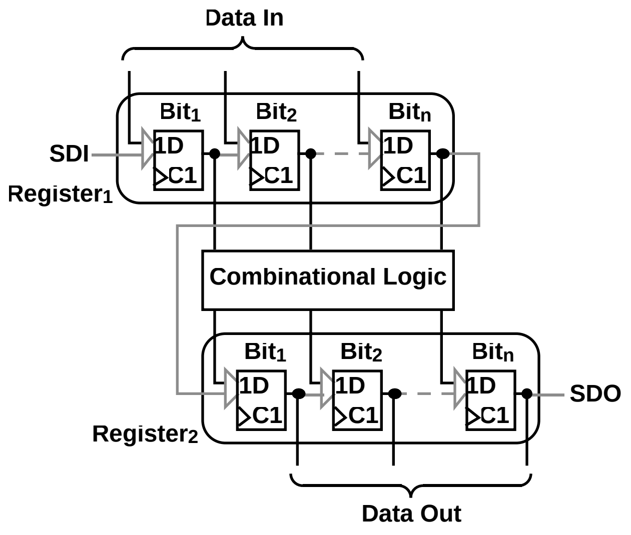

Figure 1. Traditional scan path

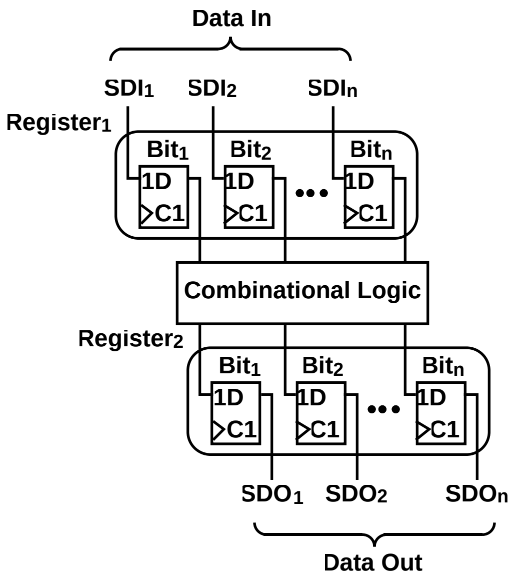

Figure 2. Orthogonal scan path

An orthogonal scan path, shown in Fig. 2, is orthogonal to the traditional scan path and connects corresponding flip-flops between registers. The flip-flops are connected in the scan path so that bit one of register one connects to bit one of register two, and bit two of register one connects to bit two of register two, and likewise for all the bits of the register. In this way, the scan path follows the normal data path flow, but is orthogonal to the traditional scan path flow.

While a traditional scan path usually has a single scan input and a single scan output, an orthogonal scan path has at least *n* scan inputs and *n* scan outputs, where *n* is the bit-width of the data path. A traditional scan implementation can also have multiple scan paths, but the multiple scan paths in orthogonal scan are a direct result of the parallel nature of the orthogonal scan path. The total test application time is thereby reduced for orthogonal scan since the total length of the scan path is reduced by a factor of *n*. The number of orthogonal scan IOs can be reduced by connecting some of the scan data outs to some of the scan data ins. For example, if SDO_1 is connected to SDI_2, and SDO_2 is connected to SDI_3, and so on till SDO_{n-1} is connected to SDI_n, then there will be only one scan data in pin, SDI_1, and one scan data out pin, SDO_n. Of course, this will add interconnect overhead.

Since the data flow during orthogonal scan is parallel to the data flow during functional operation, the functional and scan logic and interconnect can be shared. This sharing can result in significant overhead reduction. Orthogonal scan is inserted so as to

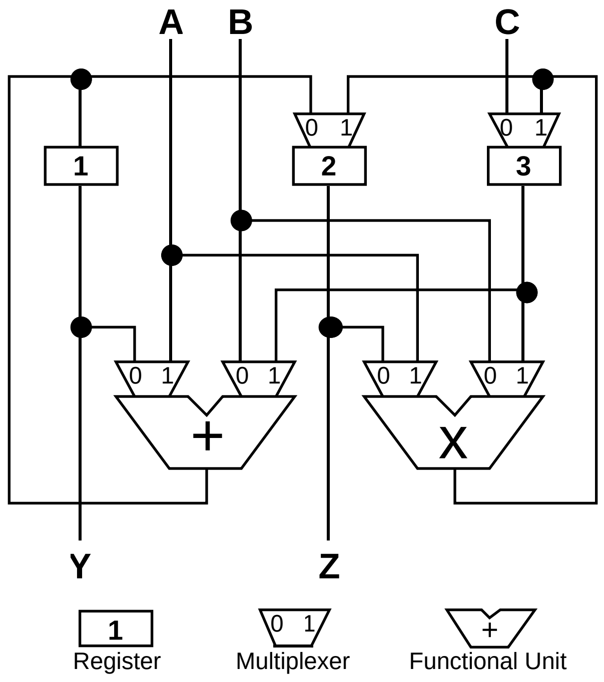

Figure 3. Data Path

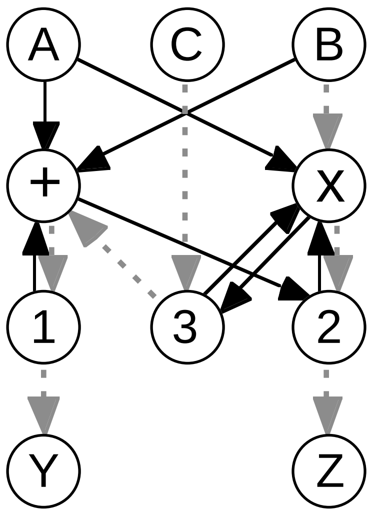

Figure 4. Connectivity Graph

maximize the sharing of the functional elements and to minimize the additional interconnect needed for the scan path.

Figure 3 shows a data path with the control signals and control logic omitted. Figure 4 is a connectivity graph showing the connections between components in the data path of Fig. 3. The nodes of the connectivity graph represent the primary inputs (*A*, *B*, *C*) and outputs (*Y*, *Z*), registers (*1*, *2*, *3*), and functional units (+, *x*) of the data path. Directed edges in the connectivity graph indicate direct connections between components in the data path. Nodes representing multiplexers may also be added to the connectivity graph, but, for simplicity, they are omitted from this discussion.

The connectivity graph can be used to identify orthogonal scan paths. An *orthogonal scan path* is a path in the connectivity graph that starts at a primary input node, includes a subset of the register and functional unit nodes, and ends at a primary output node. An *orthogonal scan implementation* is one or more orthogonal scan paths such that each register is included in one and only one path, and each functional unit is included in at most one path. A *mixed orthogonal scan implementation* is an orthogonal scan implementation that includes only a subset of the registers in the orthogonal scan paths with the remaining registers included in a traditional scan path or some other type of scan. A mixed orthogonal scan implementation may be used when individual bistables in the data paths, such as flip-flops used to hold comparison results, need to be scanned.

Other variations are discussed in [Norwood 96b] in more detail. The dashed edges in Fig. 4 show an orthogonal scan implementation for the corresponding data path in Fig. 3. There are two orthogonal scan paths; one path uses primary input *C* to scan data in, the path through the adder to connect registers *3* and *1*, and primary output *Y* to scan data out ($C \Rightarrow 3 \overset{+}{\Rightarrow} 1 \Rightarrow Y$); the other path uses primary input *B* to scan data in, the path through the multiplier to connect register *2*, and primary output *Z* to scan data out ($B \overset{\text{x}}{\Rightarrow} 2 \Rightarrow Z$). The notation in parenthesis specifies the scan path where $\overset{+}{\Rightarrow}$ indicates that the adder is used for that segment of the scan path, $\overset{\text{x}}{\Rightarrow}$ indicates that the multiplier is used, and $\Rightarrow$ indicates a connection between registers that uses no functional units, other than multiplexers.

Once the orthogonal scan paths are selected, some functional units may need to be modified so that they can transfer the scan data. Functional units included as part of the orthogonal scan path must be able to pass data unmodified, or possibly inverted, from its input to its output so that the scan function may be implemented. Some functional units, such as shifters or certain ALUs, need no modification to pass data; other functional units, however, such as most adders or multipliers, do need to be modified in order to pass data. Logic can be added to the functional unit to force an *identity* value on certain inputs to allow the orthogonal scan data to be passed unmodified. For example, the adder in Fig. 3 can be modified by adding logic to each bit of the left input so that the input will be forced to an arithmetic zero during test mode, and the output of register *3* will be transferred to the adder output. This modification is shown in Fig. 5. The data on the right input will now be passed through the adder since $x + 0 = x$. Zero is an *identity* value for the adder. A similar procedure can be used to modify other functional units.

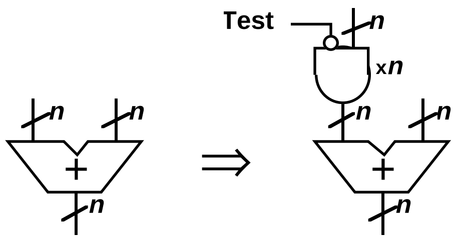

Figure 5. Adder modified for orthogonal scan

The modifications to the data path required for orthogonal scan necessitate some changes to the control logic. The multiplexer address, register enables and functional unit controls of components used in the orthogonal scan path may need to be modified to work correctly during orthogonal scan. Figure 6 shows a multiplexer modified so that input *1* is used during orthogonal scan.

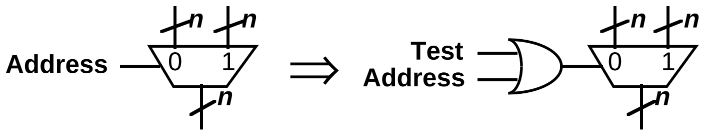

Figure 6. Multiplexer with modified address signal

2.2 Testing with an Orthogonal Scan Path

Using orthogonal scan during test is the same as using traditional scan, except that, because of the parallel structure of orthogonal scan, the test vectors are scanned in and out as words instead of as bits. The test procedure is as follows:

1. Test the orthogonal scan path shift operation. Discussed in Sec. 2.2.
2. Assert test mode signal.
3. Scan in the test vector as words.
4. Deassert test mode signal.
5. Apply test vector to primary inputs.
6. Apply system clock and capture response.
7. Assert test mode signal.
8. Scan out the response as words; scan in test vector.
9. Repeat steps 4 to 8 as required.

The test vectors that are applied via orthogonal scan should be generated for all the combinational logic in the data path — the functional logic as well as the logic added to implement the orthogonal scan path. In this way, faults on the logic added for orthogonal scan that affect the functional operation will be detected. This issue is discussed further in the next section.

2.3 Orthogonal Scan Path Integrity

Since some of the functional logic is also used to implement the scan path, questions arise about the integrity of the test in the presence of faults that affect both the functional operation and the orthogonal scan operation. Correct operation of the orthogonal scan implementation must be assured before the functional logic can be tested. This analysis focuses on single stuck-at-faults.

There are three situations to examine to assure the integrity of the orthogonal scan: faults that affect the functional units used during orthogonal scan, faults that affect the registers used during orthogonal scan, and faults that affect the multiplexers used during orthogonal scan. Faults that affect components that are not used by the orthogonal scan

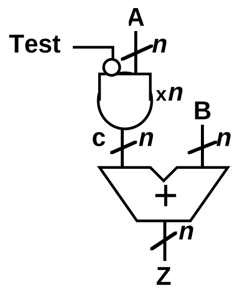

Figure 7. Functional unit modified for orthogonal scan

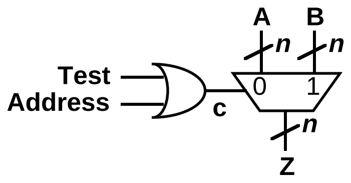

Figure 8. Multiplexer modified for orthogonal scan.

path will not affect the scan operation and can be detected by the application of appropriate test vectors during test.

A functional unit modified for orthogonal scan is shown in Fig. 7. During correct orthogonal scan operation, the data will be passed from input *B* to output *Z* while node *c* is forced to zero by the high *Test* signal. A single stuck-at-fault on one of the bits of *B* or *Z* will be detected by the shift since the shifted data will be modified. A single stuck-at-fault on one of the bits of *A* will be detected during the testing of the functional logic. If *Test* is stuck-at-0 then the left input of the functional unit will not be forced to the identity value, and, as long as *A* is not an identity value, the shifted data will be modified and the shift will detect the fault. If *Test* is stuck-at-1 then the shift will function correctly and the fault can be detected during the testing of the functional logic. If one of the bits of *c* is stuck-at a non-identity value then the shifted data will be modified and detected during the shift. If one of the bits of *c* is stuck-at an identity value then the shift will function correctly, and the fault will be detected during the testing of the functional logic. If a single stuck-at-fault on a functional unit control causes the functional unit to perform the wrong function during orthogonal scan, then the shifted data will be modified, and the fault detected. If a fault does not affect the function performed during orthogonal scan, then the shift will not be affected and the fault can be detected during functional logic testing. All the single stuck-at-faults that can affect the functional units used during orthogonal scan can be detected, with the case of *Test* stuck-at-0 requiring special attention to assure that *A* is not an identity value. This case is discussed below when a set of test patterns is given to test the scan operation before the testing of the functional logic. All the other single stuck-at-faults can be detected during the testing of the shift operation or during the testing of the functional logic.

Faults on the inputs or outputs of registers will modify the shifted data and the fault will be detected during the shift test. Faults on the register enables that cause registers to be disabled or enabled at the wrong time during orthogonal scan will affect the shifted data and can be detected during the testing of the shift operation. Faults on the register enables that do not affect the orthogonal scan operation can be detected during the functional logic testing. All faults on a register can be detected by either the testing of the shift operation or the testing of the functional logic.

A multiplexer modified for orthogonal scan is shown in Fig. 8. *B* and *Z* are used during orthogonal scan. Single stuck-at-faults on any on the bits of input *B* or output *Z* will be detected during the shift test. Faults on input *A* will be detected during functional logic testing. Faults on the multiplexer address signals are more interesting. The multiplexers act as switching logic for the orthogonal scan path. An incorrect address signal on a multiplexer, or multiplexers, can change the orthogonal scan path configuration. Fig. 9a shows a fragment of an orthogonal scan path ordering. The nodes are registers, and the edges show the connections between registers during orthogonal scan. Figs. 9b-d show the possible effects when a single multiplexer has an incorrect address select. Fig. 9b shows a forward edge that bypasses a register and shortens the orthogonal scan path. Fig. 9c shows a backward edge that forms a loop with no entry point. Fig. 9d shows an edge added that does not come from another part of the original orthogonal scan path. The new edge originates from a functional unit that is not part of the orthogonal scan path. A shortened orthogonal scan path, such as in Fig. 9b, can be detected during the testing of the shift operation, as discussed below. Loops in the

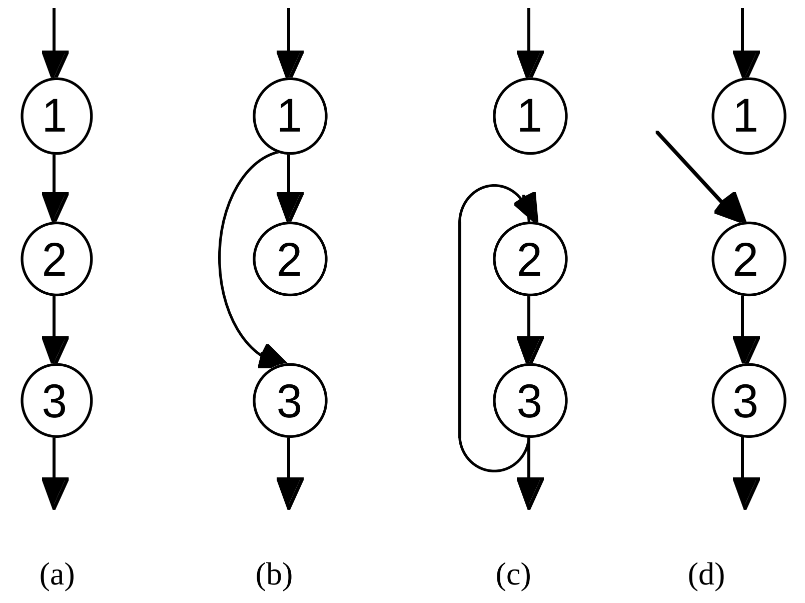

Figure 9. Orthogonal scan path fragments (a) correct ordering (b) bypassed register (c) loop (d) broken path

orthogonal scan path, as in Fig. 9c, are easily detected since the loop has no entry and data can not be shifted through it.

The third case, shown in Fig. 9d, causes the multiplexer to transfer data from some functional unit not originally in the orthogonal scan path. The register with the faulty multiplexer will now either receive data that does not correspond to data being shifted in, or it will receive data that does correspond to data being shifted in. If the data does not correspond to data being shifted in, then the fault is detected by the testing of the shift function. If the data does correspond to data being shifted in, then the orthogonal scan path may be shorter than the original orthogonal scan path, in which case the same test used to detect the shortened orthogonal scan path above will detect the fault. The data being shifted into the register with the faulty multiplexer may actually still come from the correct register through some other path. In this case, the orthogonal scan path may end up being the same length as the original orthogonal scan path, and this situation can not be detected. However, the actually orthogonal scan path is equivalent to the original orthogonal scan path since the order of the registers is the same, and the fault can be detected with the functional test since the test patterns can still be applied. A single stuck-at-fault could cause multiple multiplexers to behave incorrectly if they share the same control logic. Assuming that the data path is not sequentially redundant, the fault will also be detected in a manner similar to that discussed above.

As described above, in order to detect some of the faults during the testing of the shift operation, a set of patterns must be applied to the orthogonal scan path. The set of patterns shown in Table 1 will test the orthogonal scan path. m is the number of registers in the orthogonal scan path, and n is the width of the data path. The first m patterns are the same vector, but the actual vector used is not important, except that it must not correspond to an identity value for any of the functional units in the data path. The orthogonal scan path is filled with this non-identity value so that *Test* stuck-at-0 can be detected, as discussed above. The initial m patterns also set the scan path into a known state so that the length of the scan path can be checked by observing when the m+1 pattern is shifted out of the scan path. The final five patterns put all zero to one and one to zero transitions on each bit of the orthogonal scan path. This will detect any stuck-at faults along the scan path, as described above. Other sets of patterns could also be used in place of these five, if so desired. For example, recent work on checking experiments for flip-flops [Makar 96] could be used to determine the set of patterns. If there are multiple orthogonal scan paths, as in Fig. 4, each orthogonal scan path should be tested separately. During the testing of an orthogonal scan path, any primary inputs not used in

Table 1. Test vectors for testing orthogonal scan path

Apply at cycle #	Observe at cycle #	Test Vector						
		Bit_1	Bit_2	Bit_3	Bit_4	⋯	Bit_{n-1}	Bit_n
m+5	2m+5	0	0	0	0	⋯	0	0
m+4	2m+4	1	1	1	1	⋯	1	1
m+3	2m+3	1	1	1	1	⋯	1	1
m+2	2m+2	0	0	0	0	⋯	0	0
m+1	2m+1	0	0	0	0	⋯	0	0
m	2m	0	1	0	1	⋯	0	1
…	…	…	…	…	…	…	…	…
1	m+1	0	1	0	1	⋯	0	1

that orthogonal scan path should be held at a non-identity value. If the orthogonal scan paths pass the tests, the functional logic can then be tested as described in Section 2.2.

The net result of the above analysis is that the testing of the orthogonal scan path's ability to shift and the testing of the functional logic can detect all single stuck-at-faults. Note that the functional logic test pattern generation includes the logic added for the orthogonal scan path, as mentioned in Sec. 2.2. Once the orthogonal scan path is shown to work correctly, it can be used for diagnosis as well as for test.

3 SYNTHESIS

Section 2.1 described how to insert orthogonal scan into a circuit. The original circuit could have been designed by hand, or it could have been synthesized from a high-level description. Since the focus of this paper is on synthesized circuits, high-level synthesis is discussed in this section to provide a basic understanding of the algorithms involved. Section 4 discusses modifications to the standard high-level synthesis algorithms and builds upon the information in this section.

High-level synthesis takes a behavioral specification of a circuit, such as the VHDL description in Fig. 10 and generates a circuit, both data path and control, that implements that specification. First, a data flow graph (DFG) is derived from the behavioral description. The DFG contains information about the data operations and the data flow of the design. Each node in the DFG corresponds to an operation, and the edges correspond to variables in the initial description. Figure 11a shows the initial DFG for the VHDL description. The five data operations are represented along with their

```
ENTITY example IS
  PORT (SIGNAL go: IN BIT;
        SIGNAL a, b, c: IN INTEGER;
        SIGNAL y, z: OUT INTEGER);
END example;

ARCHITECTURE behavioral OF example IS
BEGIN
  PROCESS
    VARIABLE e, f, g: INTEGER;
    BEGIN
      WAIT UNTIL go = ‘1’;
      e := a + b;
      f := a * b;
      z <= e + f;
      g := f * c;
      y <= g + e;
    END PROCESS;
END behavioral;
```

Figure 10. VHDL behavioral description

relationships to each other which are determined by the data flow. Since the initial DFG contains no timing information, the DFG must be scheduled. In this step the clock cycle boundaries are determined, and the operations are assigned to specific clock cycles. Once the operations are scheduled, they must be bound to particular functional units. Figure 11b shows the initial DFG after scheduling and function binding. Since only a single adder and a single multiplier are used during any one clock cycle, only two functional units are required for the data path — one adder and one multiplier. Data on output *Y* is valid after three clock cycles; data on output *Z* is valid after two clock cycles.

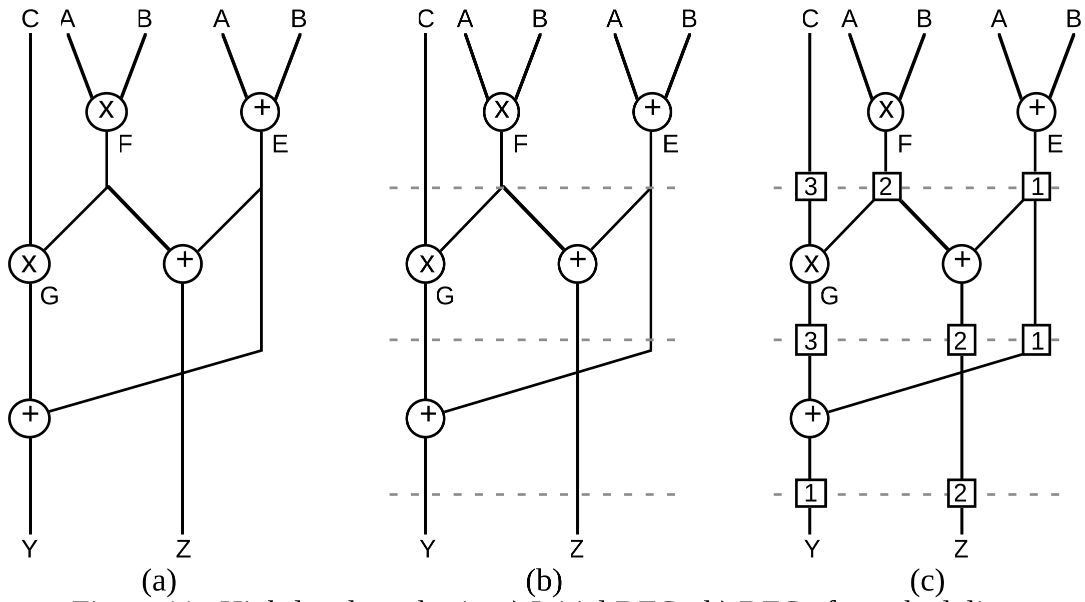

Figure 11. High-level synthesis a) Initial DFG; b) DFG after scheduling and function binding; c) DFG after register binding

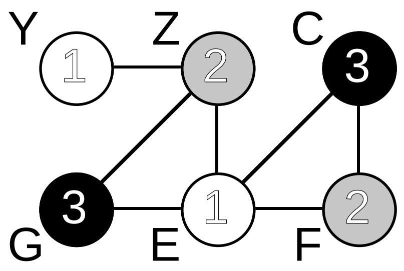

Figure 12. Colored register conflict graph

The final step in high-level synthesis is the allocation and binding of the registers. Each edge in the DFG that crosses a clock cycle boundary must be bound to a register. For any particular clock boundary, a specific register may be bound to only one variable. Register allocation and binding can be formulated as a graph coloring problem. A register conflict graph is constructed from the DFG after scheduling and function binding [Avra 91]. A node is created for each variable in the DFG that crosses a clock cycle boundary. An edge is created between two nodes if the variables corresponding to those nodes can not be assigned to the same register because the variables cross the same clock boundary. The register conflict graph for the DFG in Fig. 11b is shown in Fig. 12. The register conflict graph is then colored with the minimum number of colors where adjacent nodes have different colors. Each color corresponds to a register in the data path — the number of colors indicates the number of registers. The appropriate variables in the DFG are then bound to each register, as shown in Fig. 11c. The data path netlist can be constructed from the scheduled and bound DFG. The data path in Fig. 3 corresponds to the DFG in Fig. 11c.

These basic high-level synthesis steps (scheduling, allocation, and binding) can be modified to take orthogonal scan into consideration. We show in Sec. 4 that the function binding and register binding algorithms can target orthogonal scan in such a way as to improve the final design.

4 SYNTHESIS FOR ORTHOGONAL SCAN

The information associated with a scheduled DFG can be used by the function binding and register binding algorithms to target orthogonal scan. Section 4.1 discusses register binding, and Sec. 4.2 covers modifications to the function binding.

4.1 Register Allocation and Binding

The modified register allocation and binding algorithm for orthogonal scan is comprised of six steps:

1. Create the register conflict graph.
2. Color the register conflict graph.
3. Create the data connectivity graph.
4. Find the orthogonal scan path(s) based on DFG.
5. Modify the register conflict graph.
6. Recolor the register conflict graph.

First, the register conflict graph is created as described in Sec. 3. This register conflict graph provides an initial graph which is then modified to facilitate the implementation of the orthogonal scan. Figure 13 shows the initial register conflict graph for the DFG in Fig. 11b. The initial register conflict graph is colored to indicate how many registers need to be allocated for the data path, but the variables are not bound to these registers. The final coloring used to determine the register binding is obtained after the original register conflict graph is modified in the manner to be described. Three registers are required for the register conflict graph in Fig. 13. This initial coloring provides the number of registers that need to be included in the final orthogonal scan path. Once the number of registers is known, the orthogonal scan paths may be determined.

A *data connectivity graph* is generated from the scheduled, operation-bound DFG. The nodes in the graph are primary inputs and outputs and functional units, and the directed edges connect two nodes if there is an edge in the DFG between the corresponding elements. The data connectivity graph is much the same as the connectivity graph described in Sec. 2, but there are no registers included since the variables have not yet been bound to registers. Register information is included in the data connectivity graph by marking the edges in the graph to indicate whether the corresponding edge in the DFG crosses a clock cycle boundary, thus indicating that the edge will be bound to a register. The data connectivity graph provides information about the connections between functional units. For example, Fig. 16 shows the data

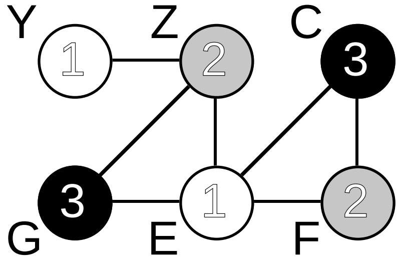

Figure 13. Initial register conflict graph

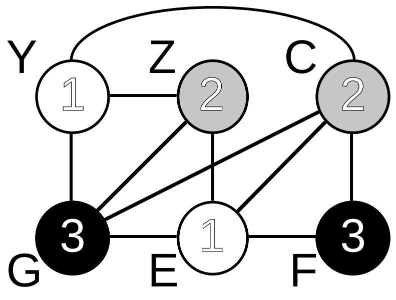

Figure 14. Modified register conflict graph

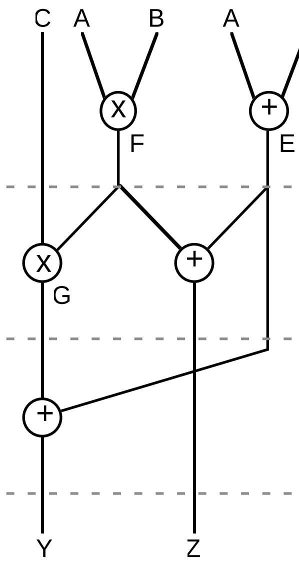

Figure 15. DFG

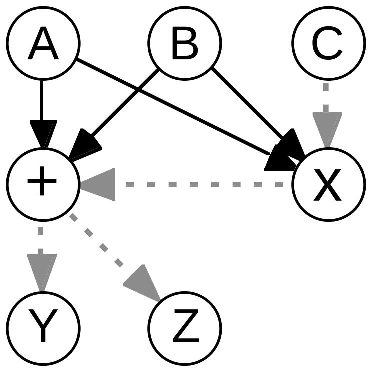

Figure 16. Data connectivity graph

connectivity graph for the DFG in Fig. 15 with the dashed edges indicating DFG edges that will be bound to registers. From this data connectivity graph it is clear that there is no connection from the adder to the multiplier, nor from the multiplier to any primary outputs. Using the data connectivity graph and the DFG from which it is derived, an orthogonal scan path, or orthogonal scan paths, may be determined before the variables are bound to registers.

Examination of the data connectivity graph in Fig. 16 shows that there exist several paths from the primary inputs to the primary outputs. A subset of these paths can be used to implement an orthogonal scan path(s) that includes all three registers required for the design. Only two choices, $C \rightarrow \mathsf{x} \rightarrow + \rightarrow Y$ or $C \rightarrow \mathsf{x} \rightarrow + \rightarrow Z$, can include all three registers. One of these two paths should be chosen so that all three registers (the number or registers is determined from the initial coloring of the register conflict graph) can be included in the orthogonal scan path. If the path $C \rightarrow \mathsf{x} \rightarrow + \rightarrow Y$ is used, then cross referencing with the DFG in Fig. 15 shows that two possible DFG edges (*F* and *G*) connect the multiplier and the adder. Either edge *F* or edge *G* can be used to implement an orthogonal scan path from the multiplier to the adder. If edge *G* is used, then the final orthogonal scan path will include the registers bound to DFG edges *C*, *G*, and *Y*. The registers bound to these three edges must be distinct so that three registers will be included in the orthogonal scan path. Since there are no conflict edges between the nodes corresponding to *C*, *G*, and *Y* in the register conflict graph, these three variables may be bound to the same register(s). Adding conflict edges between these three nodes, to form

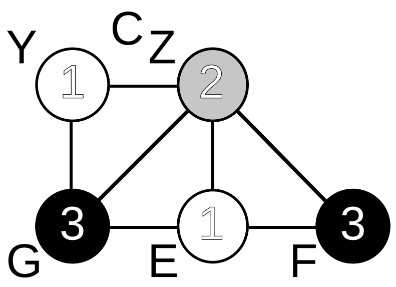

Figure 17. Register conflict graph with merged nodes

a clique, will force the corresponding variables to be bound to three different registers. Figure 14 shows the register conflict graph with conflict edges added between nodes *C*, *G*, and *Y*. The path $C \rightarrow \times \rightarrow + \rightarrow Z$ could also be used in a similar fashion.

In certain cases, it may be necessary to truncate a path in the data connectivity graph because the path includes too many edges that need to be bound to registers. Nodes in the register conflict graph can be merged to force two variables in the DFG to be bound to the same register and thereby shorten a path. For example, if only one register needed to be included in the orthogonal scan path for the DFG in Fig. 15, then DFG edge *C* and DFG edge *Z* could be bound to the same register providing a path from a primary input, through a register, and out a primary output. Merging the nodes in the register conflict graph that correspond to DFG edges *C* and *Z*, as shown in Fig. 17, will force the same register to be bound to those two edges and create an orthogonal scan path (C $\Rightarrow$ 2 $\Rightarrow$ Z). In this way paths can be shortened.

A heuristic for finding an orthogonal scan implementation based on the DFG is outlined here:

1. Create a data connectivity graph from the DFG. The nodes in the graph are primary inputs and outputs and functional units, and the directed edges connect two nodes if there is an edge in the DFG between the corresponding elements. The edges in the graph are marked to indicate whether the corresponding edge in the DFG crosses a clock cycle boundary, thus indicating that the edge will be bound to a register.
2. Choose a path from a primary input to an output in the data connectivity graph. Pick a path that has at least as many tagged edges as there are registers to be included in the orthogonal scan path — as indicated by the initial coloring of the register conflict graph.

3. Choose edges in the DFG to correspond to the path chosen in the data connectivity graph. When multiple DFG edges are possible for a specific connection, choose the DFG edges so that the register conflict graph will have the fewest conflict edges added in step 6.

4a. Truncate the path, if necessary, so that the path will include only the number of registers determined from the initial coloring of the register conflict graph.

4b. Repeat steps 2 to 3, if needed, to include more registers in additional paths.

5. Merge the register conflict graph nodes for DFG edges that should be bound to the same register because of path truncation.

6. Add conflict edges to the register conflict graph so that different registers will be bound to each register in the orthogonal scan path.

The register conflict graph can now be recolored and the resulting DFG and data path constructed. Figure 18 shows the DFG and Fig. 19 shows the data path resulting from this technique for the initial DFG in Fig. 11b. This data path and the data path in Fig. 3 both implement the same VHDL behavioral code, but Fig. 3 was synthesized without considering orthogonal scan, and Fig. 19 was synthesized to target orthogonal scan. The connectivity graph for the new data path, with the orthogonal scan path ($C \Rightarrow 2 \overset{X}{\Rightarrow} 3 \overset{+}{\Rightarrow} 1 \Rightarrow Y$) highlighted, is shown in Fig. 20. The new orthogonal scan path includes only three multiplexers, while the original orthogonal scan path includes four

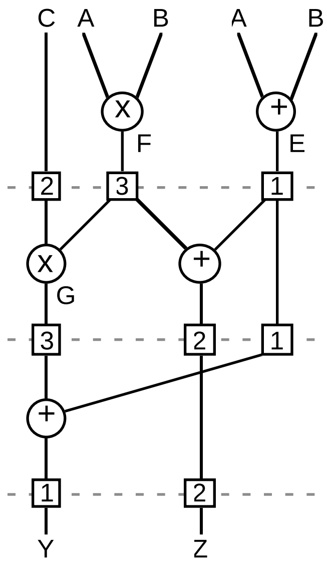

Figure 18. DFG after synthesis for orthogonal scan

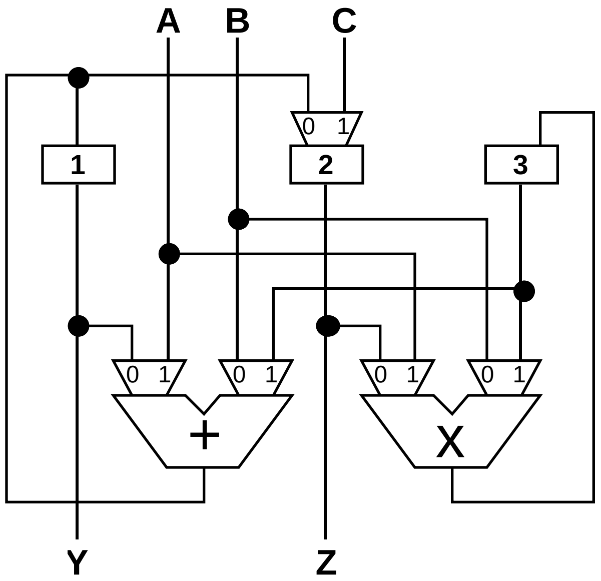

Figure 19. Data path after synthesis for orthogonal scan

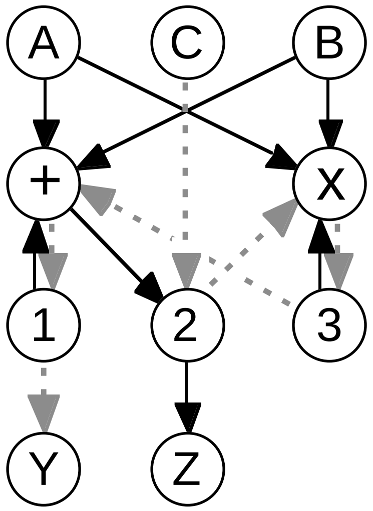

Figure 20. Connectivity graph after synthesis
for orthogonal scan

multiplexers. Therefore, the synthesis for orthogonal scan resulted in less control overhead.

4.2 Functional Unit Allocation and Binding

The register binding algorithm described in Sec. 4.1 uses the connection information contained in the data connectivity graph. The data connectivity graph is directly affected by the functional unit allocation and binding which occurs before the register allocation and binding. An unfortunate function binding may adversely affect the possible connections between the functional units. During functional unit allocation and binding, the units can be bound so that the number of DFG edges that must be bound to registers is maximized on the longest path from input to output. In other words, the longest path from input to output should contain as many DFG edges that cross clock cycle boundaries as possible. This long path in the DFG leads to a long path in the data connectivity graph and allows more registers to be included in a single orthogonal scan path and minimized the number of orthogonal scan paths.

For example, two different functional unit bindings are shown in the DFGs in Figs. 21 and 22. The DFGs are shown after scheduling and functional unit binding, but before register binding. The data connectivity graphs for these DFGs are shown in Figs. 23 and 24. The data connectivity graph in Fig. 23 has at most two functional units included in any path from input to output. By changing the binding of a single operation, the data connectivity graph in Fig. 24 has a path with all three functional units included in one path. The second binding has a single orthogonal scan path that can include three

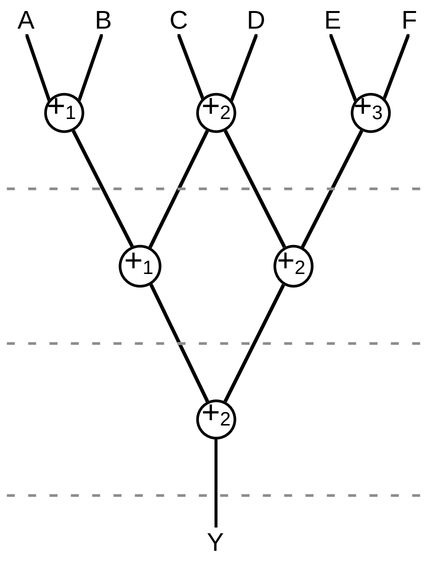

Figure 21. DFG with first function unit binding

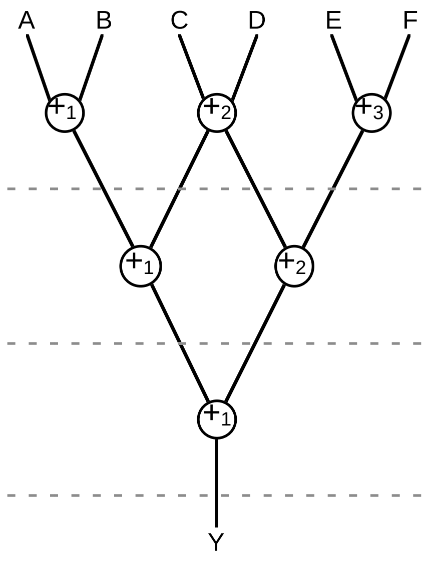

Figure 22. DFG with second function unit binding

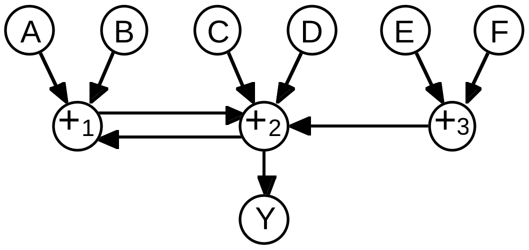

Figure 23. Data connectivity graph with first binding

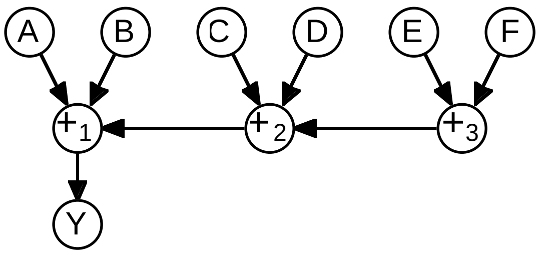

Figure 24. Data connectivity graph with second binding

registers. The first binding can include only two registers in any single orthogonal scan path.

5 RESULTS

Stanford CRC's synthesis-for-test tool, TOPS, has been modified to perform high-level synthesis targeted towards orthogonal scan. Several benchmark circuits [Dutt 92] [Tseng 86] have been synthesized with TOPS, and the results are discussed in this section.

Table 2 shows the final circuit sizes, including data path and control logic area but not the interconnect area, for three benchmark circuits. The area is shown for the circuit with no scan, with traditional scan, with orthogonal scan and normal synthesis, and with orthogonal scan and synthesis for orthogonal scan. The relative circuit sizes are reported in cell units for the LSI G10 technology [LSI Logic 96]. The overhead is calculated as

$$\%\text{Overhead} = \frac{\text{Area}_{\text{Scan}} - \text{Area}_{\text{NoScan}}}{\text{Area}_{\text{NoScan}}} \times 100$$

Where $\text{Area}_{\text{Scan}}$ is the area of the circuit with scan and $\text{Area}_{\text{NoScan}}$ is the area of the circuit without scan.

Traditional scan adds about 11% overhead. Orthogonal scan with normal synthesis reduces this overhead by about 50%. When the circuit is synthesized for orthogonal scan, the overhead is reduced by about 66% from a traditional scan implementation and by about 35% from an orthogonal scan implementation with normal synthesis.

The *diffeq* circuit shows a 66% overhead savings when the synthesis is targeted towards orthogonal scan over when normal synthesis is used and orthogonal scan is implemented, with the savings coming from a reduction in the size of the data path logic.

The *ellipf* circuit has a 42% reduction in area overhead, with the reduction showing up in both the data path and the control. The *ellipf* circuit obtained with the synthesis for orthogonal scan also has less performance impact since only four functional unit inputs are modified, as opposed to five inputs with the standard synthesis.

The *tseng* circuit [Tseng 86] does not benefit from an area savings with the synthesis for orthogonal scan, but it does have a possible performance gain over the orthogonal scan with normal synthesis. The synthesis for orthogonal scan results in an orthogonal

Table 2. Circuit sizes for benchmark circuits in cell units

Circuit	No Scan	Traditional Scan		Orthogonal Scan Normal Synthesis		Orthogonal Scan Modified Synthesis	
	Area	Area	Ovhd	Area	Ovhd	Area	Ovhd
diffeq	13404	14972	11.7%	13908	3.8%	13577	1.3%
ellipf	11115	12459	12.1%	11777	6.0%	11499	3.5%
tseng	10426	11546	10.7%	11105	6.5%	11119	6.6%

Table 3. Orthogonal scan statistics for benchmark circuits

Circuit	Orthogonal Scan Normal Synthesis		Orthogonal Scan Modified Synthesis	
	Control Inputs Modified	Functional Unit Inputs Modified	Control Inputs Modified	Functional Unit Inputs Modified
diffeq	9	3	7	3
ellipf	22	5	19	4
tseng	6	4	7	3

Table 4. Size of data path and control for benchmark circuits in cell units

Circuit	No Scan		Traditional Scan		Orthogonal Scan Normal Synthesis		Orthogonal Scan Modified Synthesis	
	DP Area	Control Area	DP Area	Control Area	DP Area	Control Area	DP Area	Control Area
diffeq	13206	198	14774	198	13686	222	13334	243
ellipf	10704	411	12048	411	11232	545	10992	507
tseng	10273	153	11393	153	10913	192	10945	174

scan path that modifies only three functional unit inputs, as shown in Table 3. The normal synthesis results in an orthogonal scan path that modifies four functional unit inputs, i.e., more paths will have extra logic and delay added to them. The size difference between the two orthogonal scan path implementations is about two AND gates.

6 SUMMARY

High-level synthesis can exploit knowledge of the circuit function to synthesize a scannable circuit that has less overhead than a circuit that has scan inserted after synthesis. Modifications to the functional unit and register allocation and binding algorithms can target orthogonal scan resulting in final designs that are fully scanned and have, on average, about 66% less overhead than a traditional scan path for the circuits shown.

ACKNOWLEDGMENTS

The authors wish to thank Dr. LaNae Avra for her comments and review. This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant No. N00014-92-J-1782, by the National Science Foundation under Grant No. MIP-9107760, and by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045.

REFERENCES

[Avra 91] Avra, L., “Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths,” *Proc. Intl. Test Conf.*, Nashville, TN, pp. 463-472, Oct. 26-30, 1991.

[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.

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

[Eichelberger 77] Eichelberger, E.B., and T.W. Williams, “A Logic Design Structure for LSI Testability,” *14th Design Automation Conf.*, New Orleans, LA, pp. 462-467, June 1977.

[Lin 95] Lin, C., M.T.-C. Lee, M. Marek-Sadowska, and K.-L. Chen, “Cost-Free Scan: A Low-Overhead Scan Path Design Methodology,” *Proc. IEEE Intl. Conf. Computer-Aided Design*, San Jose, CA, pp. 528-533, Nov. 5-9, 1995.

[LSI Logic 96] LSI Logic, *G10-p Cell-Based ASIC Products*, Milpitas, CA, 1996.

[Makar 96] Makar, S., *Checking Experiments for Scan Chain Latches and Flip-Flops*, Stanford University Thesis, 1996.

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

[Norwood 96a] Norwood, R., and E.J. McCluskey, “Synthesis-for-Scan and Scan Chain Ordering,” *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, pp. 87-92, April 28-May 1, 1996.

[Norwood 96b] Norwood, R., and E.J. McCluskey, “Orthogonal Scan: Low Overhead Scan for Data Paths,” *Proc. Intl. Test Conf.*, Washington, DC, Oct. 21-24, 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.

[Williams 83] Williams, T., and K.P. Parker, “Design for Testability — A Survey,” *Proc. IEEE*, Vol. 71, No. 1, pp. 98-112, Jan. 1983.