High-Level Synthesis for Orthogonal Scan

Robert B. Norwood and Edward J. McCluskey

Center for Reliable Computing
Stanford University

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 little 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 as little as 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. Full scan 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 [1], each with its own set of advantages and disadvantages, and each adding overhead of some sort to the circuit [2]. 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. Previous work has shown 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 [3], and data path logic [4] [5]. 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 [6]. This paper discusses how high-level synthesis for data paths can target orthogonal scan [5] [7] and how the resulting designs have less overhead than circuits that have orthogonal scan inserted after design. Synthesis for orthogonal scan 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 overhead of the orthogonal scan path.

Section 2 gives an overview of orthogonal scan, and Sec. 3 briefly 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.

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 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.

As shown in Fig. 2, an orthogonal scan path has at least *n* scan inputs and *n* scan outputs, where *n* is the bit-width

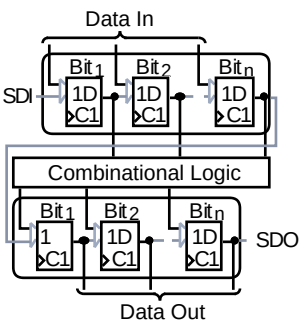

Figure 1. Traditional scan path

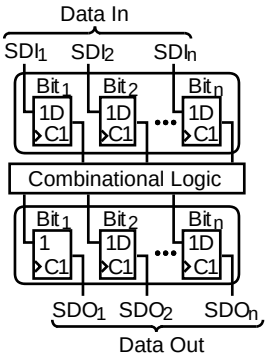

Figure 2. Orthogonal scan path

of the data path, that is, *n* is the number of bits in a register. The total test application time for orthogonal scan is therefore proportional to the number of registers in the data path and not to the number of individual flip-flops because an entire register has data scanned in or out with each clock.

Since the data flow during orthogonal scan is parallel to the data flow during functional operation, the functional and scan logic can be shared. This sharing can result in significant overhead reduction. Orthogonal scan is inserted so as to 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 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, or flip-flops, included in a traditional scan path or some other type of scan. Other variations are discussed in [5] in more detail.

The dashed edges in Fig. 4 show the data flow during orthogonal scan 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 to register *2*, and primary output *Z* to scan data out ($B \overset{x}{\Rightarrow} 2 \Rightarrow Z$). $\overset{+}{\Rightarrow}$ indicates that the adder is used for that segment of the scan path, $\overset{x}{\Rightarrow}$ indicates that the multiplier is used, and $\Rightarrow$ indicates a connection between registers that uses no functional units, other than multiplexers.

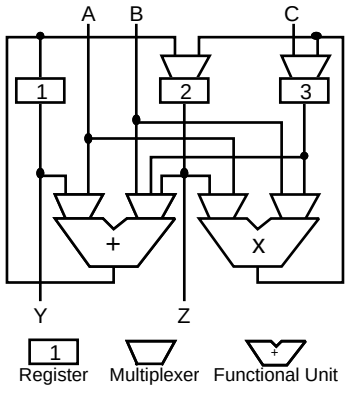

Figure 3. Data Path

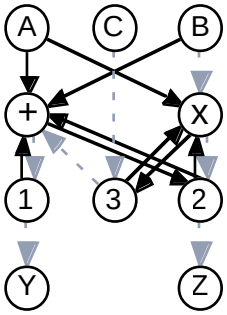

Figure 4. Connectivity Graph

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 the input to the 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, 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 for other functional units.

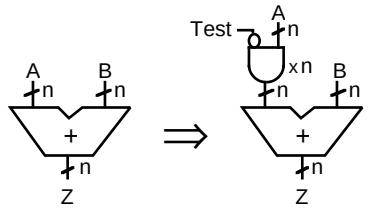

Figure 5. Adder modified for orthogonal scan

The modifications to the data path required for orthogonal scan necessitate some changes to the control logic. The multiplexer addresses, 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 *B* is used during orthogonal scan. Other control signals can be modified accordingly.

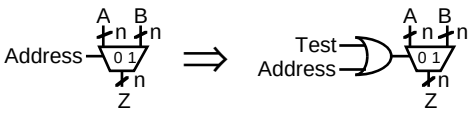

Figure 6. Multiplexer with modified address

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 nature of orthogonal scan, the test vectors are scanned in and out

as words instead of as bits. The test procedure consists of 1) testing the orthogonal scan path shift operation (discussed in Sec. 2.3) and then 2) testing the combinational logic using the orthogonal scan path.

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 scan implementation must be assured before the functional logic can be tested. This analysis focuses on single stuck 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 path will not affect the scan operation and can be detected by the application of appropriate test vectors during the testing of the functional logic.

A functional unit modified for orthogonal scan is shown in Fig. 5. During correct orthogonal scan operation, the data will be passed from input *B* to output *Z* while input *A* is forced to zero by the high *Test* signal. Any single stuck fault on *A*, *B*, *Z*, *Test*, or the function select lines 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 can be detected by either the testing of the shift operation or the testing of the functional logic depending on whether or not the fault affects the scan path.

A multiplexer modified for orthogonal scan is shown in Fig. 6. Input *B* and output *Z* are used during orthogonal scan. Single stuck faults on any of the bits of *B* or *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. A shortened orthogonal scan path can be detected during the testing of the shift operation. Loops in the orthogonal scan path are easily detected since the loop has no entry and data can not be shifted through it. Faults that cause the multiplexer to transfer data from some functional unit not originally in the orthogonal scan path can be detected as long as the data path is not sequentially redundant.

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 any single stuck fault. 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 High-level 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 briefly 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 a VHDL behavioral description, 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. Figure 7a shows an initial DFG. Five data operations are represented along with their 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 7b 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.

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. 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. 7b is shown in Fig. 8. The

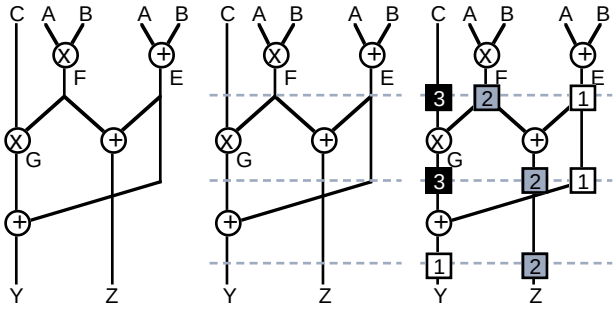

Figure 7. High-level synthesis a) Initial DFG; b) DFG after scheduling and function binding; c) DFG after register binding

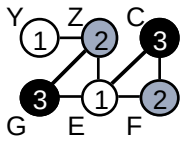

Figure 8. Colored register conflict graph

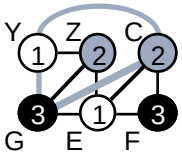

Figure 9. Modified register conflict graph

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 register is then bound to each variable in the DFG, as shown in Fig. 7c. 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. 7c. The information in the scheduled and bound DFG is used to generate the control logic.

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 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 register binding algorithm to target orthogonal scan. The goal of the modified algorithm is to bind the registers so that an orthogonal scan path can be easily inserted with little overall overhead. The size of the data path before the orthogonal scan is inserted may actually increase as compared to the size of the data path synthesized with the original register binding algorithm. However, the desire is to have the final data path size, including the orthogonal scan path, be smaller than the size of the final data path, including the orthogonal scan path, synthesized with the original register binding technique.

The modifications to the register allocation and binding algorithm attempt to insure that an orthogonal scan implementation is possible for the final register binding without unduly affecting the overall data path size. In other words, the orthogonal scan path is just one criteria by which the registers are allocated and bound. The number of registers and multiplexers required for the register binding is also a consideration.

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 an orthogonal scan implementation.
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 path. Figure 8 shows the initial register conflict graph for the DFG in Fig. 7b. The initial register conflict graph is colored to indicate how many registers need to be allocated for the data path, but the registers are not bound to the variables. 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. 8. This initial coloring provides an estimate of 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* (DCG) is generated from the scheduled, operation-bound DFG. The nodes in the DCG 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 DCG 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 DCG 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 DCG provides information about the connections between functional units. For example, Fig. 11 shows the DCG for the DFG in Fig. 10 with the dashed edges indicating DFG edges that will be bound to registers.

The DFG and DCG shown in Figs. 10 and 11 will be used for an informal discussion of the procedure for finding an orthogonal scan implementation. A more detailed description of the heuristic is given later. From the DCG it is clear that there is no connection from the adder to the multiplier, nor from the multiplier to any primary outputs. Using the DCG 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 DCG in Fig. 11 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

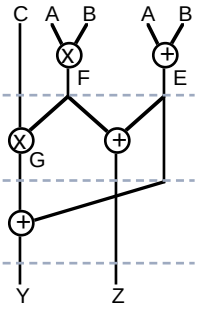

Figure 10. DFG

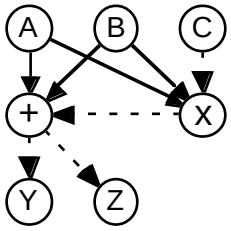

Figure 11. Data connectivity graph

required for the design. Only two choices, $C \rightarrow x \rightarrow + \rightarrow Y$ or $C \rightarrow x \rightarrow + \rightarrow Z$, 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 $C \rightarrow x \rightarrow + \rightarrow Y$ is used, then cross referencing with the DFG in Fig. 10 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 a clique, will force the corresponding variables to be bound to three different registers. Figure 9 shows the register conflict graph with conflict edges added between nodes *C*, *G*, and *Y*. The added edges are highlighted. The path $C \rightarrow x \rightarrow + \rightarrow Z$ 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. 10, 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. 12, will force the same register to be bound to those two edges and create an orthogonal scan path (C ⇒ 2 ⇒ Z). In this way paths can be shortened.

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

1. Create a DCG from the DFG.
2. Choose a path from a primary input to an output in the DCG. 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 DCG. 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 7.
4. Repeat steps 2 to 3, if needed, to include more registers in additional paths.
5. 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.
6. Merge the register conflict graph nodes for DFG edges that should be bound to the same register because of path truncation.
7. Add conflict edges to the register conflict graph so that different registers will be bound to each variable used in the orthogonal scan path.

The register conflict graph can now be recolored and the resulting DFG and data path constructed. The modified register conflict graph is not guaranteed to be colored with the same number of colors as the initial register conflict graph. The entire procedure can be iterated until the correct number of registers is included in the orthogonal scan implementation. Figure 13 shows the

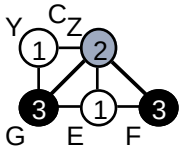

Figure 12. Register conflict graph with merged nodes

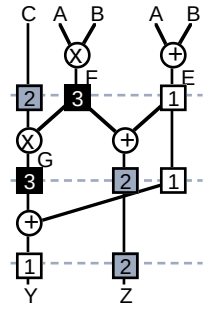

Figure 13. DFG after synthesis for orthogonal scan

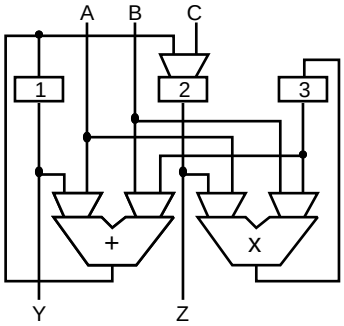

Figure 14. Data path after synthesis for orthogonal scan

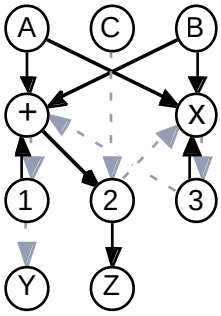

Figure 15. Connectivity graph after synthesis for orthogonal scan

Table 1. 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	14033	15601	11.2%	14886	6.1%	14502	3.3%
ellipf	10775	12119	12.5%	11450	6.3%	11107	3.1%
gcd	2835	2975	4.9%	2886	1.8%	2880	1.6%
tseng	10394	11514	10.8%	11057	6.4%	11057	6.4%

DFG and Fig. 14 shows the data path resulting from this technique for the initial DFG in Fig. 7b (which is the same as Fig. 10). 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. 14 was synthesized to target orthogonal scan. The connectivity graph for the new data path, with the orthogonal scan path ($C \Rightarrow 2 \overset{\times}{\Rightarrow} 3 \overset{+}{\Rightarrow} 1 \Rightarrow Y$) highlighted, is shown in Fig. 15. The new orthogonal scan path includes only three multiplexers, while the original orthogonal scan path includes four multiplexers. The synthesis for orthogonal scan results in a smaller final data path since the data path is smaller and fewer multiplexer addresses signals are modified for the orthogonal scan implementation.

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 [8] [9] have been synthesized with TOPS, and the results are discussed in this section.

Table 1 shows the final circuit sizes, including data path and control logic area but not the interconnect area, for four 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 [10]. 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, on average, about 10% overhead. Orthogonal scan with normal synthesis reduces this overhead by about 50%, on average. When the circuit is synthesized for orthogonal scan, the overhead is reduced by about 60% from a traditional scan implementation and by about 30%, on average, from an orthogonal scan implementation with normal synthesis.

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 60% 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

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

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

[3] 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.

[4] Bhattacharya, S., and S. Dey, “H-SCAN: A High Level Alternative to Full-Scan Testing With Reduced Area and Test Application Overheads,” *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, pp. 74-80, April 28-May 1, 1996.

[5] Norwood, R., 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.

[6] 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.

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

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

[9] 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.

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