Center for Reliable Computing

TECHNICAL REPORT

Synthesis-for-Scan: Reducing Scan Overhead with High-Level Synthesis

By Robert B. Norwood

97-6

(CSL TR # 97-743)

November 1997

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

Abstract:

This technical report contains the text of Robert B. Norwood's thesis "Synthesis-for-Scan: Reducing Scan Overhead with High-Level Synthesis."

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.

SYNTHESIS-FOR-SCAN: REDUCING SCAN OVERHEAD WITH HIGH-LEVEL SYNTHESIS

A DISSERTATION

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL ENGINEERING

AND THE COMMITTEE ON GRADUATE STUDIES

OF STANFORD UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

By

Robert B. Norwood

November 1997

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Edward J. McCluskey (Principal Advisor)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

Oyekunle Olukotun

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

_____

John T. Gill

Approved for the University Committee on Graduate Studies

_____

To my wonderful wife, Sherri,

For her love, support, and prayers.

"Trust in the LORD with all your heart and lean not on your own understanding;

In all your ways acknowledge him, and he will make your paths straight."

Proverbs 3:5,6

ABSTRACT

Scan paths are widely used to improve the testability of sequential designs. A scan path provides direct access to the bistable elements, greatly improving the controllability and observability of the circuit. It is then no longer necessary to generate test patterns for a sequential circuit, a difficult and time consuming task, since the scan path effectively turns the sequential circuit into a combinational circuit during testing. Scan designs, in general, have some overhead, such as increased area, degraded performance, or longer test times, but careful construction of the scan path can reduce the total overhead while still improving the testability.

Designing a scannable circuit is typically a two-step process. First, the circuit is designed to meet the functional specifications, without taking the scan path into consideration. The circuit is then analyzed, and the scan path is inserted based on this analysis. Our work focuses on combining these two steps into one, where the scan path is constructed and inserted during the synthesis of the circuit. In this way, the synthesis algorithms and the scan insertion algorithms can interact to obtain a better result.

We present several new scan synthesis techniques to reduce the scan path overhead by sharing the functional logic and interconnect with the test logic and interconnect. The synthesis algorithms are modified to insert the scan path to maximize the amount of sharing and thereby reduce the overhead due to the scan path. *Beneficial scan* targets control logic and orders the scan path during synthesis to minimize the area and performance overhead. *Orthogonal scan* targets data path logic and exploits the regular data path structure to share the functional and test logic and interconnect. We also present a new technique to constrain the scan path order based on high-level information so that the scan path can be used to apply arbitrary two-pattern delay tests.

ACKNOWLEDGMENTS

I express my appreciation to my advisor, Prof. Edward J. McCluskey, for his direction, encouragement, and instruction during my time at Stanford. He has modeled many of the qualities and characteristics to which I aspire, and I have been greatly inspired by his enthusiasm for both teaching and research.

I am very grateful to my colleagues at the Center for Reliable Computing: Khader Abdel-Hafez, Dave Brokaw, Jonathan Chang, Yi-Chin Chu, Dr. Piero Franco, Vladimir Fridman, Dr. Hong Hao, Rajesh Kamath, Erin Kan, Sunil Kosalge, Wern-Yan Koe, Vincent Lo, Dr. Siyad Ma, Dr. Samy Makar, Subhasish Mitra, Shridar Mukund, Philip Shirvani, Dr. Nirmal Saxena, Rudy Tan, Dr. Nur Touba, and Sanjay Wattal. I want to especially thank Dr. LaNae Avra for her many helpful comments and suggestions. I want to give many special thanks to Siegrid Munda for her excellent administrative support.

I would like to thank Prof. Oyekunle Olukotun, my associate advisor, Prof. Gregory Kovacs, my committee chairman, and Prof. Giovanni De Micheli for being the final member of my committee. Special thanks to Prof. John Gill for being my third reader.

I want to thank my many friends for their prayers and support. I would like to mention just a few by name: Mark Griffie, Loren James, Santosh Poneen, Jeff Siegle, and Jeff Swanson. Thanks for your friendship.

Finally, I wish to thank my family. I thank my wife, Sherri, for her love, support, and prayers. Thanks, also, to my parents for the many blessings they have given me.

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.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

Chapter 1
Introduction

1.1 Motivation

The design of an integrated circuit can be broken down into three basic steps: specification, implementation, and fabrication. *Specification* is the process of determining what the circuit should do. Decisions such as what instruction set or which filtering algorithm will be used are made during this step. *Implementation* is the step where a network of logic gates, transistors, and interconnect is created that performs the functions required by the specification. It is generally possible to automatically generate at least a portion of the implementation with logic synthesis. *Fabrication* results in a piece of silicon (a chip) with transistors and interconnect as specified by the implementation. Figure 1-1 shows this design flow. Two additional steps are also shown in Fig. 1-1: verification and testing. *Verification* involves comparing the implementation to the initial specification. If there are mismatches during verification, then the implementation may need to be modified to more closely match the specification. *Testing* is the process of assuring that the function of each fabricated chip actually corresponds to the function of the implementation. Only chips that pass the tests are shipped to customers.

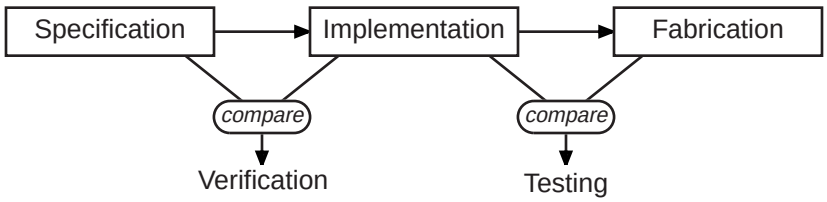


Figure 1-1. Typical design flow

The cost of testing is rapidly increasing due to the decreasing ratio between the number of chip external connections and the number of transistors on the chip. Figure 1-2 shows predictions for this ratio for ASICs (Application Specific Integrated Circuits) for the next several years [SIA 94]. This decrease in the ratio makes controlling and

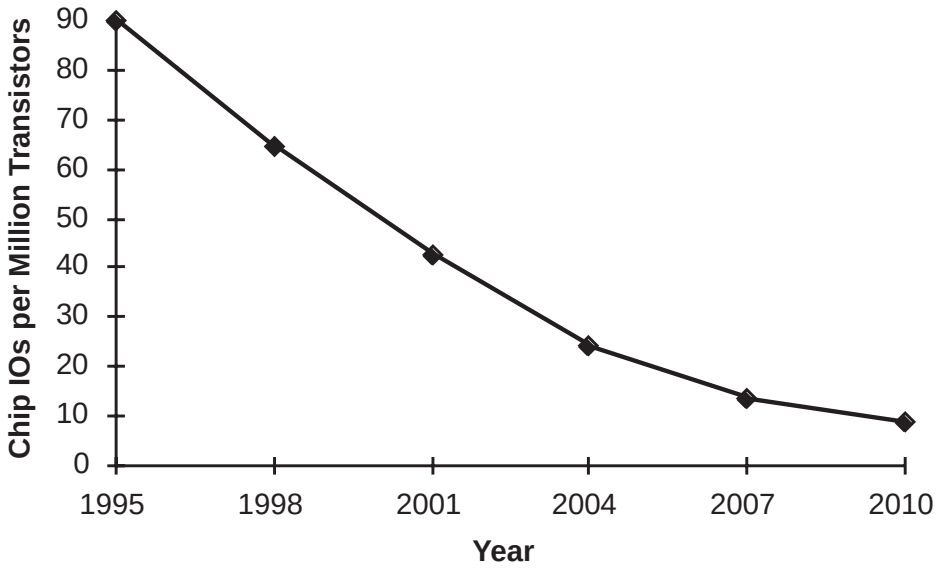


Figure 1-2. Ratio of chip I/Os to transistors on chip

observing the nodes in the circuit more difficult because there are many more nodes that must be controlled and observed from fewer inputs and outputs. The cost of testing can be reduced by including circuitry on each chip to improve testability and facilitate testing. *Testability* is an imprecise term used to describe the costs of test generation and application and the fault coverage. Testability increases when the costs of test generation and test application decrease or the fault coverage increases. Testability decreases when the costs of test generation and test application increase or the fault coverage decreases.

Design techniques intended to increase the testability of a circuit are referred to as *design-for-testability* (DFT). In general, DFT attempts to improve the observability and controllability of designs. *Observability* refers to the ability to determine the values of internal signals by observing the circuit's primary outputs. *Controllability* refers to the ability to force specific values on internal signals through the application of signals to the circuit's primary inputs. A design with low observability or controllability is generally not very testable.

A common DFT practice is the use of scan paths. There are many scan path techniques in use [McCluskey 86], but they all have the same general approach — the

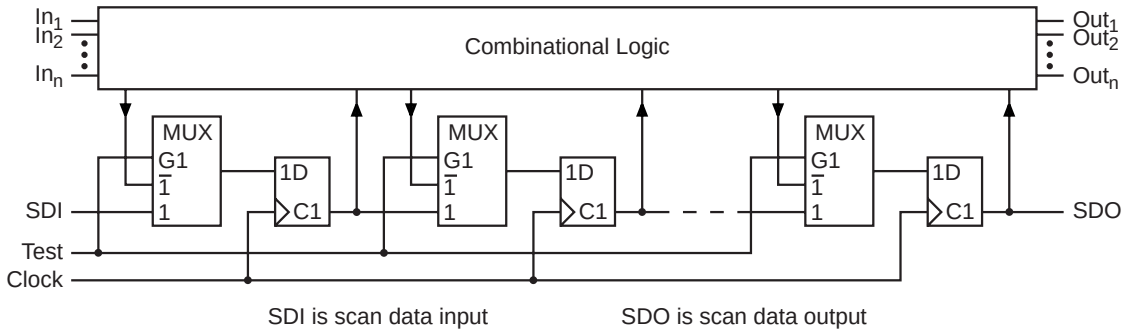


Figure 1-3. Typical scan path implementation

system bistables are connected together to form one or more shift registers during test. There are two modes of operation: normal functional operation, and test operation where the bistables are connected together to form one or more shift registers, called *scan paths*. The scan path provides complete controllability and observability of the system bistables since an arbitrary bit sequence can be shifted into (scanned in) the bistables, and the contents of the bistables can be shifted out (scanned out). Figure 1-3 illustrates the scan path concept.

Scan paths improve the testability of the design by:

1. Reducing test generation cost since sequential circuit test pattern generation is not required. The controllability provided by the scan path allows the circuit to be treated as a combinational circuit during test.
2. Improving fault coverage over that obtainable from sequential circuit test pattern generation. The complete controllability and observability of bistables allows higher fault coverage to be achieved.

However, while scan paths improve testability, they also add overhead to the design that can add to the overall cost of the circuit [McCluskey 86]. The scan path overhead comes from:

1. Increased area due to the additional circuitry and interconnect required to construct the scan path.

2. Possible performance penalty due to the increased propagation delay in the scannable bistables or due to additional loading on the interconnect required for the scan path.
3. Additional pins for the test signals.
4. Increased testing time due to serialization of the test patterns.

We present techniques that reduce the area and performance overhead without compromising the improved testability that scan paths provide. These techniques also lead to easy implementation of multiple scan paths that can reduce the number of additional test pins required and allow reduced test times.

1.2 Contributions

This dissertation presents several new synthesis-for-scan techniques that order the bistables in the scan path to minimize the overhead due to the scan path implementations. The scan paths are ordered during synthesis so that the synthesis process can target the final circuit to the particular scan implementation, thereby reducing the overhead due to inserting a scan path. In particular, the following are the contributions of this dissertation:

- We have identified several *beneficial relationships* between bistables that allow the functional logic and interconnect to be shared with the scan path logic and interconnect. Cost-free scan [Lin 95] has been included as a subset of these relationships.
- We have developed and implemented a *new synthesis-for-scan technique*, beneficial scan, to order the scan path during synthesis to maximize the number of beneficial relationships in the final scan path and thereby minimize the scan overhead.
- We have developed and implemented a *new state assignment for beneficial scan technique* that maximizes the number of beneficial relationships in a finite state machine.

- We have formalized *merged orthogonal scan*, a scan path architecture for data path logic which shares the test logic and interconnect with the functional logic and interconnect.
- We have developed and implemented a *new high-level synthesis-for-scan technique* to insert a merged orthogonal scan path into a data path circuit.
- We have developed and implemented a *new scan ordering technique* to order scan paths for the application of two-pattern delay fault tests.

1.3 Outline

Chapter 2 describes beneficial scan, a synthesis-for-scan technique that targets control logic. Section 2.1 briefly discusses previous work, while Sections 2.2 and 2.3 cover beneficial scan and state assignment for beneficial scan, respectively. Section 2.4 summarizes our work on beneficial scan.

Chapter 3 introduces merged orthogonal scan, a synthesis-for-scan technique for data path logic. Section 3.1 covers previous work, Sec. 3.2 describes merged orthogonal scan, and Sec. 3.3 discusses techniques to modify high-level synthesis to target merged orthogonal scan. Section 3.4 summarizes our work on merged orthogonal scan.

Chapter 4 discusses a technique to constrain scan paths so that arbitrary two-pattern delay fault tests can be applied. Section 4.1 discusses previous work, and Sec. 4.2 describes the new technique. Section 4.3 summarizes our work on using scan paths to apply two-pattern delay fault tests.

Chapter 5 gives some concluding remarks.

Chapter 2
Synthesis-for-Scan for Control Logic

2.1 Previous work

Various techniques have been proposed to reduce the costs associated with scan designs. Specially designed scan elements [Schultz 85] [Zasio 85] [Bhavsar 86] [Giles 91] [Mukund 91] can reduce the additional area or interconnect loading. Careful ordering of the scan path elements can reduce the interconnect or testing time. The long test application time due to the serial shifting of the test data can be reduced by ordering the scan path so that the frequently accessed elements are closer to the scan-in or scan-out pins, thereby making those elements more easily accessed [Gupta 91] [Narayanan 92] [Narayanan 93]. This ordering is done after the circuits have been designed, and the additional interconnect overhead can be high. The test application time can also be reduced by exploiting the inherent sequential nature of the circuit and applying multiple system clock cycles to the circuit for each test vector that is scanned in [Pradhan 92] instead of applying a single system clock cycle as is standard, but the test generation time is increased due to the use of sequential test patterns. By giving up some of the controllability and observability of a fully scanned design, partial-scan designs attempt to reduce the overhead by making only a subset of the system bistables scannable. Since fewer bistables need to be modified, the area overhead of partial scan is typically less than that of full scan [Agrawal 88] [Chickermane 90] [Gupta 90] [Lee 90] [Chakradhar 94], but the circuit is no longer strictly combinational during test, and sequential test pattern generation is necessary. Cost-free scan [Lin 95] attempts to choose a primary input vector to sensitize paths through the combinational logic to form portions of the scan path. The scan path overhead is reduced since fewer bistables must be modified to form the scan path. However, the overhead reduction is constrained by the limited number of cost-free situations in many circuits. Cost-free scan is a subset of

beneficial scan, and all the benefits of cost-free scan can be obtained with beneficial scan. These techniques generally approach the problem of reducing the test overhead after the circuit has been designed.

Other work has considered scan path insertion during the synthesis of the circuit. Testability constraints can be used during synthesis to embed scan paths in a circuit [Cox 94] [Cox 95] Algorithms similar to those of automatic test pattern generation can transform the circuit so that a scan path is embedded and any rule violations corresponding to the constraints are repaired. Instead of modifying the circuit structure directly, the finite state machine (FSM) description of the circuit can be augmented to embed a shift register into the state machine, and the circuit can then be synthesized [Vinnakota 92] [Kanjilal 93]. In this case, a FSM description of the circuit must be available, and it is not obvious how a particular modification to the FSM description will affect the size of the final synthesized circuit. Bistable selection for partial scan can also be performed during synthesis [Bhatia 93].

Our approach, *beneficial scan*, inserts scan during synthesis by sharing the functional and test logic. Cost-free scan is integrated into the technique. Beneficial scan can be applied to partial scan, but this has not currently been examined. Earlier work [Norwood 96a] introduced beneficial scan and described how a beneficial scan path can be inserted into a circuit during synthesis. Here we provide some more details on this procedure and discuss how state assignment of symbolic states can be targeted to beneficial scan.

2.2 Beneficial scan

2.2.1 Overview

We present a technique, called *beneficial scan*, that combines circuit synthesis and scan path insertion into one step. Knowledge of the circuit functions is used to order the scan path elements in such a fashion that the functional logic and the test logic are shared,

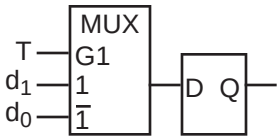


Figure 2-1. MD flip-flop

reducing the cost of the scan path. The scannable bistables are assumed to be implemented with *MD flip-flops* for this discussion. An MD flip-flop [McCluskey 86], shown in Fig. 2-1, is formed by placing a multiplexer at the data input of the flip-flop to allow the selection of two different data inputs — either d_0 for normal system operation or d_1 for test mode — based on the test select, *T*.

This work is based on previous work done at the Center for Reliable Computing looking at reducing the overhead of a built-in self test (BIST) design [Avra 93] [Avra 94] by ordering the scan path so that the functional logic and the test logic — a multiple input signature register (MISR) — can be shared. Beneficial scan uses the same concept of ordering the scan path so that the functional and the test logic can be shared but focuses on reducing the overhead due to the scan path itself.

This synthesis-for-scan method takes the scan path implementation into account during the synthesis of the circuit. A scan path order is found that maximizes the sharing of the functional and the test logic, reducing the overhead due to the scan path. The techniques described here assume a full-scan design.

Every flip-flop input in a sequential circuit has some relationship with every other flip-flop output, as well as with every primary input. Some of these relationships allow some, or all, of the test logic to be shared with the functional logic. When functional and test logic can be shared, it is called a *beneficial relationship.* Other, *non-beneficial*, relationships do not allow the logic to be shared and a multiplexer, or multiplexer equivalent, must be inserted to make a flip-flop scannable during test operations.

Once each flip-flop is classified based on the relationships between it and every other flip-flop, as well as on the relationships between it and every primary input, an order for

the scan path is determined using these relationships. Each type of relationship has a cost associated with it, where the cost reflects the logic overhead required to make the flip-flop scannable. The goal is to obtain an ordering that minimizes the cost and thereby maximizes the logic sharing.

After the order is determined, the circuit is synthesized. Synthesis occurs after the scan path ordering so that the equations for the flip-flop inputs may be factored in such a way as to take advantage of the sharing of the test and functional logic. The factoring is based on the Shannon expansion, as explained in Sec. 2.2.2. The final scan path obtained in this manner may contain inversions between flip-flops, but these inversions may be compensated for during test pattern generation by inverting appropriate data bits in the test vector. Forbidding inversions may result in a less optimal ordering and a reduction in the amount of logic shared.

The concepts behind beneficial scan can be illustrated with a simple example. The circuit in Fig. 2-2 has a traditional scan path where each flip-flop has a multiplexer added to make the flip-flop scannable. These additional multiplexers are highlighted in the figure. Examination of the circuit based on the relationship criteria discussed in Sec. 2.2.2 shows that the input of *flip-flop 1* has a beneficial relationship with primary

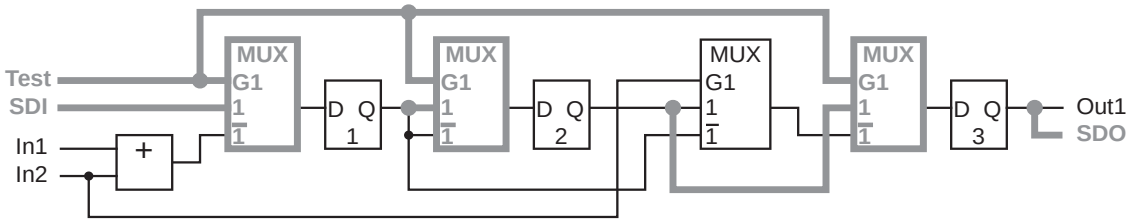


Figure 2-2. Example circuit with a traditional scan path implementation

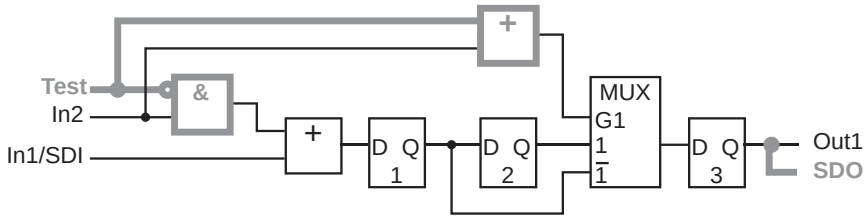


Figure 2-3. Example circuit with beneficial scan

Table 2-1. Flip-flop relationships for example circuit in Fig. 2-2

Output of	Input of		
	flip-flop 1	flip-flop 2	flip-flop 3
flip-flop 1	—	Beneficial	Beneficial
flip-flop 2	Non-Beneficial	—	Beneficial
flip-flop 3	Non-Beneficial	Non-Beneficial	—
input *In1*	Beneficial	Non-Beneficial	Non-Beneficial
input *In2*	Beneficial	Non-Beneficial	Beneficial

inputs *In1* and *In2.* The input of *flip-flop 2* has a beneficial relationship with *flip-flop 1*. The input of *flip-flop 3* has a beneficial relationship with *flip-flop 1*, *flip-flop 2*, and primary input *In2*. These relationships are summarized in Table 2-1.

Figure 2-3 shows the same circuit, but with a beneficial scan path inserted taking advantage of the beneficial relationships. The scan path takes advantage of three beneficial relationships — replacing three multiplexers with one AND gate and one OR gate and reducing the interconnect between flip-flops — resulting in a circuit with less area and probably less delay.

Section 2.2.2 discusses the details of the various relationships, and Sec. 2.2.3 describes how to use the relationships to order the scan path.

2.2.2 Scan element classification

The input equation for each flip-flop in a circuit can be expressed as a function of flip-flop outputs and primary inputs. Based on these input equations each flip-flop in a synchronous circuit has some relationship with every other flip-flop and primary input in the circuit. This relationship may be trivial, as in the case when a flip-flop input expression is vacuous in another flip-flop output. This relationship could be a simple classification whether one input equation includes the output of another flip-flop, or it could be based on more complex input function characteristics.

Some of these relationships allow some, or all, of the test logic to be shared with the functional logic. These are *beneficial relationships*. Other relationships do not allow the

logic to be shared, and a multiplexer, or multiplexer equivalent, must be inserted to make a flip-flop scannable. These are *non-beneficial relationships*.

For our beneficial scan algorithm, we base the majority of the relationships between flip-flops on Shannon's expansion theorem. The Shannon expansion transforms a function, $f(x_1, x_2,\ldots, x_n)$, based on residues. The x_i-residue, also called the positive-phase cofactor of f with respect to x_i, is defined as

$$f_{x_i}(x_1, x_2,\ldots, x_n) = f(x_1, x_2,\ldots, x_i = 1,\ldots, x_n)$$

and the x'_i-residue, or negative-phase cofactor of f with respect to x_i, as

$$f_{x'_i}(x_1, x_2,\ldots, x_n) = f(x_1, x_2,\ldots, x_i = 0,\ldots, x_n).$$

The Shannon expansion is defined as

$$f(x_1, x_2,\ldots, x_n) = x_i\, f_{x_i} + x'_i\, f_{x'_i}.$$

Based on the Shannon expansion, each flip-flop j may be classified with respect to every other flip-flop i. The classifications are summarized in Table 2-2, and a more detailed description of each case, along with the reasoning behind the naming, is given in Appendix I. Notation: D_j is the input of flip-flop j. Q_i is the output of flip-flop i. T is the scan test select signal — '1' for scan mode, '0' for normal system operation. f is the function for normal system operation, that is, the input equation before scan is inserted. g, s and a are arbitrary functions.

Some of these classifications (case B, case A, case X, and case M) are beneficial

Table 2-2. Flip-flop classifications with beneficial cases in bold

Class	Q_i-residue	Q'_i-residue	Equation Form	New Scan Function	Ovhd
case B	constant	constant	$D_j=Q_i^*$	$D_j=Q_i^*$	none
case A	constant not constant	not constant constant	$D_j=Q_i^*+f_{Q'_i}$ $D_j=Q_i^* f_{Q_i}$	$D_j=Q_i^*+T'f_{Q'_i}$ $D_j=Q_i^*(T+f_{Q_i})$	AND OR
case X	$(f_{Q'_i})'$	$f_{Q'_i}$	$D_j=Q_i^*\oplus f_{Q'_i}$	$D_j=Q_i^*\oplus T'f_{Q'_i}$	AND
case M	(s + a)	(s' a)	$D_j=sQ_i^*+s'a$	$D_j=(s+T)Q_i^*+(s+T)'a$	OR
case N	f	f	$D_j=f$	$D_j=T'f+TQ_i$	MUX
case O	not constant	not constant	$D_j=Q_i f_{Q_i}+Q'_i f_{Q'_i}$	$D_j=T'f+TQ_i$	MUX

relationships and allow the test logic to be shared with the functional logic. The other classifications (case N and case O) are non-beneficial, and no logic may be shared. Case O classifications contain any relationships that do not fall into one of the other classifications. These four beneficial classifications include all situations where the sharing of the test logic and the functional logic causes a single AND or OR gate (or no gate) to be added to a flip-flop input to make the flip-flop scannable. The assumption is made that a two-to-one multiplexer has less area than two AND or two OR gates. Therefore, we do not consider situations where more than one logic gate must be added since the resulting overhead would be greater than that of a multiplexer. This assumption is validated by examining several commercial technologies, such as the LSI G10p technology [LSI Logic 96]. Other relationships also exist that may be used to classify the flip-flops, and one, cost-free scan (a beneficial relationship), is described in Appendix I.

2.2.3 Scan path ordering

Once the flip-flops have been classified, a *relationship graph* is constructed. A relationship graph is a weighted, directed graph with a node for each flip-flop and each primary input and edges representing the relationships between the flip-flops and inputs. The edges follow the direction of the shift during the scan operation; therefore, the head of an edge is the node for flip-flop *i*, or input *i*, and the tail is the node for flip-flop *j*. Each flip-flop node has a weighted edge to and from every other flip-flop node, as well as one from every primary input node. An example relationship graph is shown in Fig. 2-4 with a possible scan path highlighted.

The edges are weighted to show the cost of adding test logic between the two flip-flops. Case B and cost-free flip-flops need no additional logic to perform the scan function. Case A, case X and case M flip-flops require the addition of a single gate to add scan capability. Case N and case O flip-flops must have an entire multiplexer added. Based on these overheads, case N and case O edges have a large weight, case B and cost-free edges have a very small weight, and the other cases are weighted in-between.

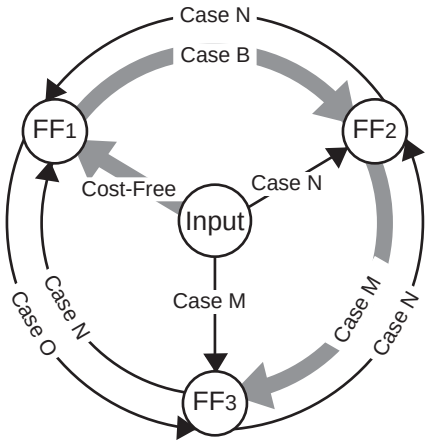


Figure 2-4. Relationship graph

The goal is to find a minimal-weight path through the graph that starts at an input node and covers all the flip-flop nodes. For a circuit with *p* primary inputs and *n* flip-flops, there are $p \bullet n!$ possible scan path orderings since there are $n!$ ways to order the flip-flops and *p* different inputs to use to scan in data. An exact solution is computationally expensive, so a heuristic is required.

Certain characteristics of the relationship graph allow heuristics to perform very well.

- Any ordering of the flip-flops is a valid order for the scan path.
- The cost of choosing any particular flip-flop has little dependence on previous decisions, i.e., a poorly chosen flip-flop will only affect the cost of one other segment of the scan path.

We chose a greedy approach, which is described as follows:

1. Find a lowest-weight edge from a primary input to a flip-flop. Make this primary input the scan data input and make the flip-flop the first in the scan path.
2. Find a lowest-weight edge from the last flip-flop in the scan path to another flip-flop not already in the scan path. Make this new flip-flop the last flip-flop in the scan path.
3. Repeat step 2 until all flip-flops are added to the scan path.

This basic greedy algorithm can be modified to make better choices at each step, such as giving preference to flip-flops with only one beneficial relationship over flip-flops with multiple beneficial relationships in order to leave more options open for later or making sure that a selected flip-flop is not the only beneficial case for another flip-flop. Modifications such as these help guide the algorithm to a better result.

The algorithm attempts to reduce the number of additional test pins needed by using one of the existing primary inputs as the scan-data-in pin. If there is a need to have an explicit scan-data-in pin, a new primary input node may be added to the relationship graph while the other primary input nodes are removed. This new input will have a case N edge to every flip-flop node.

The above procedure will order the flip-flops into a single scan path; Appendix I discusses variations to the procedure to form multiple scan paths.

2.2.4 Results

We have implemented the beneficial scan algorithm in *sis* [Sentovich 92] to order flip-flops for single, or multiple, scan paths. We have used our algorithms to analyze the circuits and order the flip-flops in the scan paths. Existing *sis* functions are used to perform logic minimization and technology map the resulting circuits.

Table 2-3 shows the overhead reduction with beneficial scan compared to a traditional scan path for some of the IWLS91 sequential, multi-level benchmark circuits [Yang 91]. The overhead reduction is calculated as

$$\%\text{Reduction} = \frac{\text{Overhead}_{\text{TradScan}} - \text{Overhead}_{\text{BeneScan}}}{\text{Overhead}_{\text{TradScan}}} \times 100.$$

A negative overhead reduction indicates that the beneficial scan circuit has a larger area than the traditional scan circuit. The average overhead for beneficial scan is about 22%, while the average overhead for traditional scan is about 31%, as shown in Appendix I. These average overheads are both somewhat large due to the high ratio of flip-flops to logic in many of the benchmark circuits. However, the comparison is still interesting

Table 2-3. Overhead reduction — beneficial scan compared to traditional scan

Circuit	% Reduction
dsip	76.1
mm4a	27.0
mult16b	62.9
mult32b	58.3
s1488	33.3
s1494	73.4
s208.1	45.9
s27	52.6
s298	-16.5
s344	-74.8
s349	-66.3
s382	-166.0
s386	69.6
s400	-74.0
s420.1	50.6
s444	7.1
s510	450.1
s526	5.5
s641	2.4
s713	32.0
s820	115.5
s832	-166.3
s838.1	42.7

since the beneficially ordered scan paths have almost 30% less overhead than the circuits with traditional scan paths.

2.3 State assignment for beneficial scan

2.3.1 Overview

There are two steps involved in synthesizing finite state machines (FSM): state assignment and logic synthesis. Section 2.2 described how beneficial scan can be used during logic synthesis to reduce the scan overhead by sharing the functional logic and the test logic; however, state assignment, which is performed before logic synthesis, was not discussed.

Input	Current State	Next State
0	state0	state0
0	state1	state4
0	state2	state0
0	state3	state4
0	state4	state2
0	state5	state6
0	state6	state3
0	state7	state7
1	state0	state4
1	state1	state4
1	state2	state4
1	state3	state4
1	state4	state6
1	state5	state6
1	state6	state7
1	state7	state7

a)

Input	Current State			Next State		
	b_1	b_2	b_3	b_1	b_2	b_3
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	1	1	0
0	1	1	0	0	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	1	0	0
1	0	1	1	1	0	0
1	1	0	0	1	1	0
1	1	0	1	1	1	0
1	1	1	0	1	1	1
1	1	1	1	1	1	1

b)

Figure 2-5. Finite state machine description: a) symbolic description; b) specified description

State assignment is the process of taking a symbolic FSM description, such as that in Fig. 2-5a, and assigning binary codes to the symbolic states to obtain a specified FSM description, such as that in Fig. 2-5b. The state assignment used directly affects the implementation of the final circuit. One state assignment may result in a circuit with little area while another state assignment may lead to a large circuit. The state assignment also affects the relationships between flip-flops since those relationships are really the relationships between bits of the state assignments. The state assignment can be targeted to increase the number of beneficial relationships and thereby reduce the size of the final scannable circuit by allowing more sharing of the functional and test logic.

Take, for example, the four-state FSM *dk15* from the IWLS91 logic synthesis benchmark suite [Yang 91]. A minimum of two state bits, giving four (2^2) possible code words, are needed to implement a four-state FSM. Since all four possible code words are needed to encode four states, there are three distinct state assignments (symmetries reduce the twenty-four possible state assignments down to three distinct state assignments). Out of these three state assignments for *dk15*, one assignment results in a

circuit with one case N non-beneficial relationship and one case M beneficial relationship, one results in a circuit with two case M beneficial relationships, and one results in a circuit with one case M beneficial relationship and one cost-free scan relationship. As this simple FSM shows, the state assignment can directly affect the types of relationships between flip-flops in the final scan path.

2.3.2 State assignment modifications

Modifications can be made to the state assignment algorithm to bias the final state assignment such that there are many beneficial relationships. In this way, the final size of the scannable circuit can be reduced. We have modified the *jedi* state assignment tool [Lin 89] to take beneficial scan into consideration. *Jedi* uses simulated annealing with various cost functions to assign the state codes. The cost function can be modified to account for the number of beneficial relationships and lead to a solution with a maximal number of beneficial relationships. The new cost function is $cost\ function_{beneficial\ scan}$ and is shown in Fig. 2-6.

The basic approach to finding the cost for a particular state assignment involves finding the relationships between pairs of current state bits and next state bits. There are $b(b\text{-}1)$ pairs of current state bits and next state bits to examine, where b is the number of bits used to represent the state, since bits in the same position do not need to be considered. For example, in the FSM description in Fig. 2-5b there are three state bits for each state and six interesting pairs.

```
beneficial_cost_function()
    cost = 0
    for each current state bit position s
        for each next state bit position ns
            if s != ns
                relationship_cost = find_best_relationship_cost(s, ns)
                cost = cost + relationship_cost
    return cost
end beneficial_cost_function
```

Figure 2-6. Pseudo-code for *beneficial_cost_function()*

```
find_best_relationship_cost(s, ns)
    for each transition i
        if i.next_state[ns] always equals i.current_state[s] or
          i.next_state[ns] always equals inverse of i.current_state[s]
            case1 = true
        if i.current_state[s] = 1 always implies i.next_state[ns] = 1 or
          i.current_state[s] = 0 always implies i.next_state[ns] = 0
            case2 = true
        for each transition j
            if i.input implies j.input
                if i.next_state[ns] always equals i.current_state[s] or
                  i.next_state[ns] always equals inverse of i.current_state[s]
                    freescan = true
                if j.current_state equals i.current_state except in bit position s
                    if i.next_state[ns] always equals i.current_state[s] or
                      i.next_state[ns] always equals inverse of i.current_state[s]
                        other_beneficial_case = true
    if freescan or case1 or case2 or other_beneficial_case is true
        return beneficial_case_cost
    else
        return non_beneficial_case_cost
end find_best_relationship_cost
```

Figure 2-7. Pseudo-code for *find_best_relationship_cost()*

For each pair, the relationship between the bits is determined, as described in Fig. 2-7, and the cost for that type of relationship is returned, where beneficial relationships have a lower cost than non-beneficial relationships. The following notation is used: *i.state*[*state_bit*], where *i* indicates the specific transition, *state* indicates either the current state or the next state, and *state_bit* indicates the particular bit position of the *state*; *i.input*, which refers to the input values corresponding to transition *i*.

For example, in the FSM in Fig. 2-5b, the first bit of the current state and the second bit of the next state have a case B relationship since the first bit and the second bit are the same for all sixteen transitions. The criteria for case A and cost-free scan relationships are also met because of the characteristics of the case B relationship, but the case B relationship is a stricter classification. The first bit of the current state and the third bit of the next state exhibit a case A relationship since whenever the first bit is a '0' the third bit is also a '0'. There is not a case B relationship because the fourth transition has the third bit equal to the first bit, and the fifth transition has the third bit equal to the inverse of the

first bit. There is not a cost-free scan relationship because transitions exist with the input equal to '0' and equal to '1' where the third bit is both equal to the first bit and equal to the inverse of the first bit.

Once the relationships are determined for a particular state assignment, the relative cost can be calculated. If the cost function is based solely on the beneficial relationships, without taking into consideration other factors such as output dominance, the number of possible beneficial relationships in the final circuit does increase but so does the circuit size. In order to minimize area while still increasing the number of beneficial relationships, criteria other than just the beneficial relationships must be included in the cost function. The goal is to have the beneficial scan cost guide the state assignment to a solution with many beneficial relationships without unduly affecting the original cost. If the beneficial scan cost function is added to the original cost function, then the original cost function can be made to dominate while the new cost function still guides the solution toward many beneficial relationships. The new cost function is now

$$\text{cost function}_{\text{new}} = \text{cost function}_{\text{original}} + \text{cost function}_{\text{beneficial scan}}$$

where $\textit{cost function}_{\textit{original}}$ is the original *jedi* cost function and $\textit{cost function}_{\textit{beneficial scan}}$ is the cost function described in Fig. 2-6.

2.3.3 Results

Table 2-4 shows the percentage difference in the area obtained with the modified state assignment for beneficial scan compared to the normal state assignment with beneficial scan for some of the IWLS91 benchmark FSM circuits [Yang 91]. The percent difference is calculated as

$$\%\text{Difference} = \frac{\text{Area}_{\text{normal}} - \text{Area}_{\text{modified}}}{\text{Area}_{\text{normal}}} \times 100.$$

Table 2-4 differs from Table 2-3 in that Table 2-4 compares two circuits that both have beneficial scan while Table 2-3 compares a beneficial scan circuit to a traditional scan circuit. A negative percentage difference indicates that the circuit with normal state

Table 2-4. Modified state assignment compared to normal state assignment

Circuit	% Difference
bbara	33.2
bbsse	21.9
bbtas	-9.1
cse	5.2
dk14	-13.1
dk15	9.8
dk16	4.7
dk17	7.4
dk27	24.2
dk512	-3.3
ex1	-12.3
ex2	7.9
ex3	9.5
ex4	0.8
ex5	-20.0
ex6	7.9
ex7	22.7
keyb	57.7
kirkman	-2.1
lion	9.4
mark1	3.8
mc	2.2
opus	0.0
planet	33.7
planet1	33.8
s1	49.8
s1488	-39.3
s1494	68.4
s1a	35.3
s208	-7.4
s27	7.2
s420	5.3
s510	12.9
s820	1.4
s832	0.5
sand	0.7
scf	2.3
shiftreg	1.3
sse	14.7
styr	-1.3
tav	6.9
tbk	0.4
train4	6.4

assignment has a smaller area than the circuit with the modified state assignment. The state assignment targeting beneficial scan results in a circuit with about 9% less area than the circuit obtained with the original state assignment. Appendix 1 contains more detailed results.

2.4 Summary

Beneficial scan is a synthesis-for-scan technique for random logic that orders the scan path(s) during logic synthesis. The order is determined to maximize the amount of sharing that can take place between the functional and test logic. Sharing the functional and test logic reduces the overhead due to insertion of the scan path. Once the scan path order has been determined, the logic is synthesized with the scan path inserted. Circuits synthesized with beneficial scan are fully scanned and are smaller than circuits that have a traditional scan path inserted after synthesis.

The insertion of a beneficial scan path takes place during logic synthesis, and, for a finite state machine, logic synthesis occurs after state assignment. The specific state assignment used directly affects the final order of the beneficial scan path. The state assignment algorithm can be modified to target the final state assignment toward a state assignment that will more likely produce a low overhead beneficial scan path. State assignment targeting beneficial scan usually results in a beneficially scanned circuit with less area than that obtained with normal state assignment. The reduction in circuit size is not guaranteed, but results show an area savings is likely.

Chapter 3
Synthesis-for-Scan for Data Path Logic

3.1 Previous work

Previous work in synthesis-for-scan for data path logic has explored using the data path functionality without modification to load test vectors into the registers [Abadir 85] [Anirudhan 89] [Bhatia 94] [Chickermane 94]. For example, given the data path shown in Fig. 3-1 (Fig. 3-2 shows the logic symbols used) an arbitrary vector *V1* can be loaded into register *3* by setting input *A* to zero and input *B* to *V1* and then applying two system clocks to load register *3*. In this way, register *3* is loaded with *A + B*, and since input *A* is zero, register *3* is loaded with the value on input *B*. No modifications to the data path are required, but since these techniques do not actually implement a scan path, the sequential nature of the circuit can make the application of the test vectors complex. H-SCAN [Bhattacharya 96] is a technique that exploits some of the parallelism in a design to reduce the scan path overhead, but it does not make use of functional units, and it adds interconnect to the design.

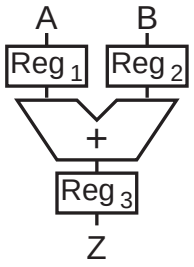


Figure 3-1. Test pattern justification

Orthogonal scan was first presented as a possible means to help reduce the overhead of built-in self-test (BIST) [Avra 94], but details of the technique were not discussed. The basic concepts of orthogonal scan were formalized in [Norwood 96b], and high-level synthesis issues relating to orthogonal scan were presented in [Norwood 97]. Here we present the concepts from earlier work [Norwood 96b] [Norwood 97] and build upon those concepts with some more details and a discussion about orthogonal scan path

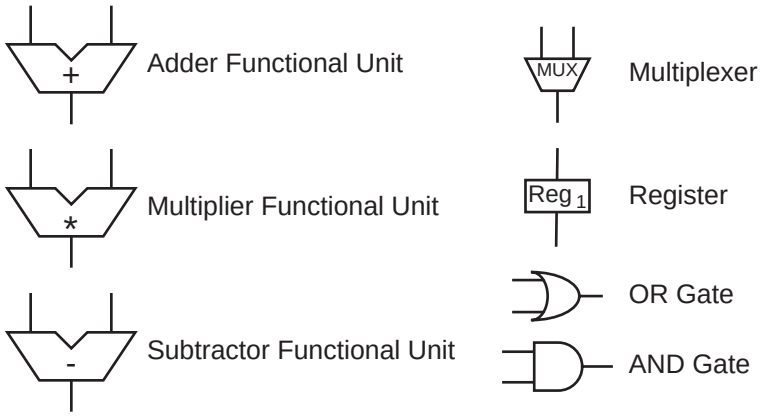


Figure 3-2. Logic symbol definitions

integrity in the presence of faults that affect both the functional and the scan operation. A distinction is made here between orthogonal scan and merged orthogonal scan, as defined in Sec. 3.2.1, where orthogonal scan does not share functional and test logic, and merged orthogonal scan does. Merged orthogonal scan also assumes that the primary inputs and outputs to the data path can be used as scan data inputs and outputs. The earlier work did not make this distinction and assumed that the functional and test logic was shared in any orthogonal scan path. This paper focuses on using merged orthogonal scan to implement full scan paths, where every bistable is included in a scan path. Orthogonal scan may also be useful for other testing techniques such as pseudo-random BIST [Avra 92] or arithmetic BIST [Adham 95].

3.2 Merged orthogonal scan

3.2.1 Overview

In data path type designs, traditional scan paths, represented in Fig. 3-3a, connect individual bistables 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. This type of configuration makes use of the fact that bits within a register are typically close together in the physical design, and the interconnect overhead can

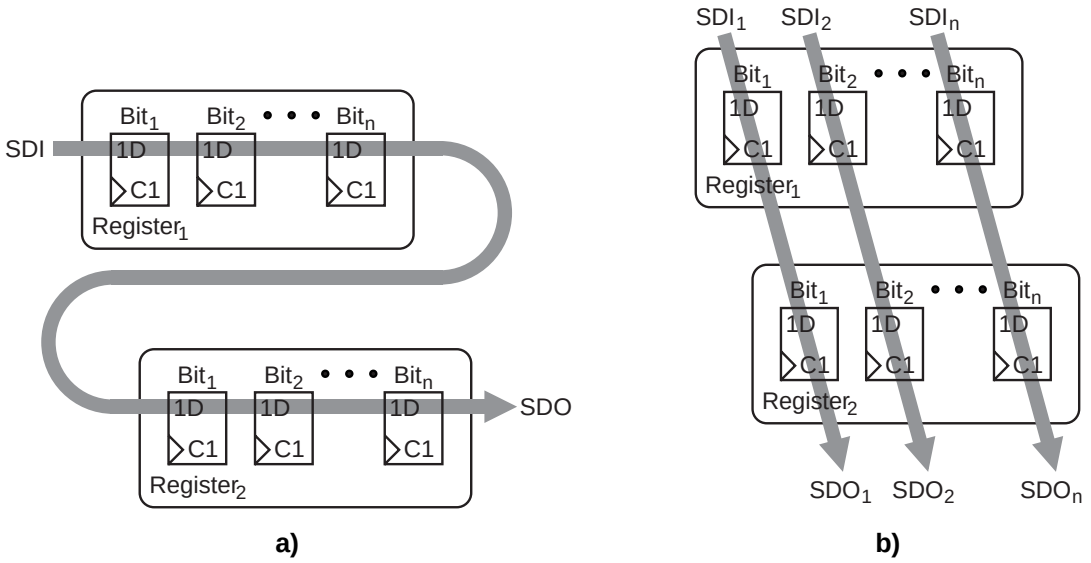


Figure 3-3. Scan path ordering: a) traditional scan path; b) orthogonal scan path

therefore be low within a register. However, replacing each bistable with a scannable element adds overhead. If the scan path is instead connected in the same direction as the flow of data in the data path, existing functionality of the data path can be used to facilitate the sharing of the functional and test logic.

The orthogonal scan path data flow, represented in Fig. 3-3b, is orthogonal to the traditional scan path data flow and connects corresponding bistables between registers. The bistables 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 flow of data in the scan path follows the normal data path flow, but is orthogonal to the traditional scan path data flow.

Since the data flow during orthogonal scan is parallel to the data flow during functional operation, the scan logic, interconnect, and inputs can be shared with the functional logic, interconnect, and inputs. This sharing can result in significant overhead reduction. The orthogonal scan path order is determined to maximize the sharing of the functional elements and to minimize the additional interconnect needed for the scan path.

Figure 3-4a shows a portion of data path logic consisting of two registers and some combinational logic. Adding a traditional scan path, as shown in Fig. 3-4b, requires the addition of a multiplexer, or multiplexer equivalent, to each bit of each register. Additional interconnect is also needed. An orthogonal scan path that does not share any

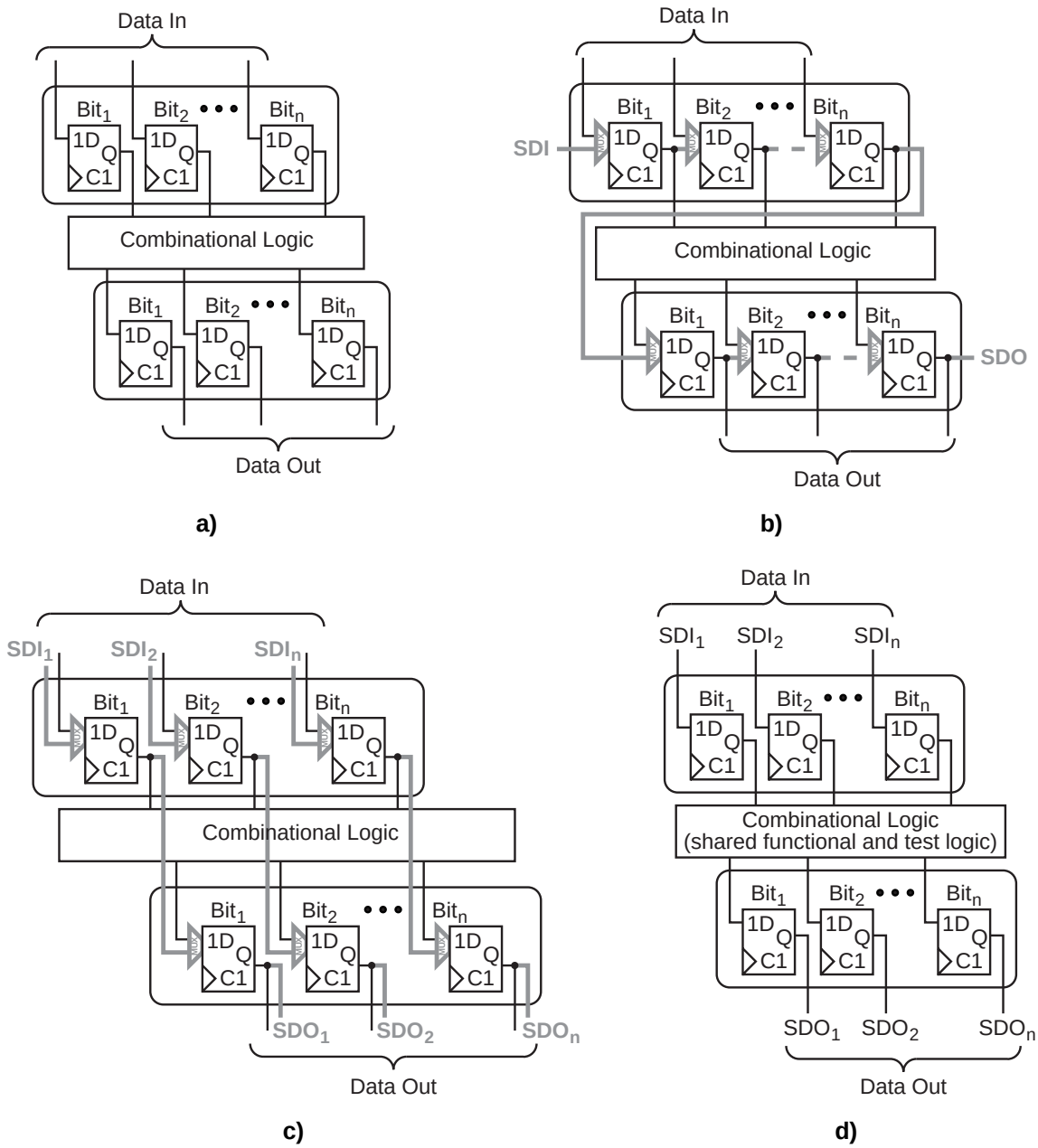


Figure 3-4. Scan path implementation in data path logic: a) data path fragment; b) traditional scan path; c) orthogonal scan path with no logic sharing; d) merged orthogonal scan path with logic sharing

of the functional logic, Fig. 3-4c, also requires the addition of multiplexers and interconnect. However, when the orthogonal scan path shares the functional and test logic, Fig. 3-4d, it may be possible to implement the scan path without adding any multiplexers or interconnect. An orthogonal scan path that shares the functional logic with the test logic is called a *merged orthogonal scan path*. The rest of this paper focuses on merged orthogonal scan paths.

Merged orthogonal scan paths can be inserted into a data path, whether it was designed by hand or through synthesis, and result in lower overhead than if traditional scan was used. However, if the data path is synthesized specifically for merged orthogonal scan [Norwood 97], the final circuit can be smaller than when merged orthogonal scan is inserted after the data path is designed. Various modifications can be made to the synthesis operations to target the final result to a merged orthogonal scan implementation.

Using Stanford CRC's synthesis-for-test tool, *TOPS*, we have synthesized various benchmark circuits using this technique, and results show that merged orthogonal scan paths often require no additional scan in/out pins; little additional interconnect other than for control signals; and only slight modifications to the functional units. This is in contrast to traditional scan paths that require extra interconnect for the scan path and for control, and the addition of a multiplexer to every flip-flop. Because of the parallel structure of merged orthogonal scan paths, the test application time is reduced in the same way that it is for multiple traditional scan paths.

3.2.2 Merged orthogonal scan path insertion

Starting with a high-level specification of the design, there are three basic steps to implementing merged orthogonal scan paths:

1. determine the merged orthogonal scan implementation
2. modify the functional units
3. synthesize the control logic

Each of these steps is described in detail in the following sections.

At some point in the design process, the data path must be constructed out of functional units, registers, multiplexers, and individual logic gates. The data path can either be constructed by hand or with the use of synthesis. The merged orthogonal scan path can be inserted after the data path has been constructed, as discussed in this section, or it can be inserted during the synthesis of the data path, as discussed in Sec. 3.3. The assumption here is that the design is obtained through synthesis of a high-level description.

Once the structure of the data path is determined, the structure can be analyzed to determine an orthogonal scan path order. The merged orthogonal scan path is constructed to take advantage of the data flow in the data path so that the existing hardware and interconnect can be used to implement the scan path. The scan path order is selected to minimize the amount of additional logic and interconnect required to insert the orthogonal scan path.

Figure 3-5a shows a data path with the control signals and control logic omitted. The data path contains two primary inputs, two registers, two multiplexers, two functional units, and one primary output. Figure 3-5b is a *connectivity graph* showing the connections between components in the data path of Fig. 3-5a. The connectivity graph can be used to identify merged orthogonal scan paths. The nodes of the connectivity graph represent the primary inputs (*A*, *B*) and outputs (*Z*), registers (*1*, *2*), and functional units (+, *) of the data path. A directed edge from node *x* to node *y* indicates a connection in the data path from the component corresponding to node *x* to the component corresponding to node *y*. Nodes representing multiplexers may also be added to the connectivity graph, but, for simplicity, they are omitted from this discussion. Since each multiplexer is associated with a single functional unit input or a single register input, including the multiplexers in the connectivity graph does not add any information. The edges in the connectivity graph are weighted where the weight indicates the cost

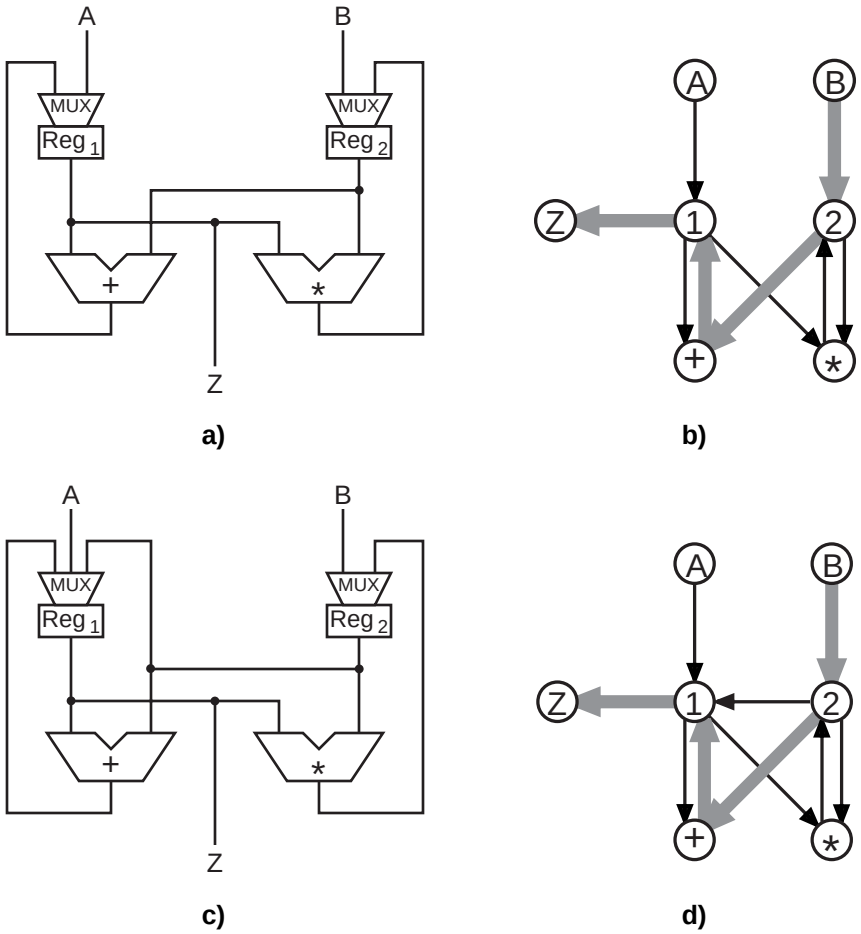


Figure 3-5. Determining merged orthogonal scan path: a) example data path logic; b) connectivity graph; c) modified data path logic; d) modified connectivity graph

associated with using that edge as part of the orthogonal scan path. The cost is calculated based on the amount of interconnect and logic that would have to be added to form that segment of the scan path.

A *merged orthogonal scan path* is a path in the connectivity graph that starts at a primary input node, includes a subset of the register nodes and functional unit nodes, and ends at a primary output node such that data can be transferred between adjacent registers in the path without being changed. Some functional units included in the merged orthogonal scan path may need to be modified to allow the data to be passed unchanged. A *merged orthogonal scan implementation* is a set of one or more merged orthogonal scan paths. In order to allow the merged orthogonal scan paths to be scanned

concurrently, each register must be included in one and only one path and each functional unit must be included in at most one path. If the merged orthogonal scan paths do not need to be scanned concurrently, then these restrictions can be relaxed, and each register and functional unit can be included in multiple paths.

Heuristics can be used to find a minimal-cost path in the connectivity graph such that the path starts at a primary input node and ends at a primary output node. The path is determined by finding or constructing identity transfer paths (I-paths) between registers. An *I-path* [Abadir 85] [Parulkar 95] is a path from a primary input or a register output to a primary output or a register input where the data can be transferred unchanged. For example, in Fig. 3-5a there are three I-paths: the path from primary input *B* to register *2*, the path from primary input *A* to register *1*, and the path from register *1* to primary output *Z*. An I-path can be constructed between register *2* and register *1* by modifying the adder to make it capable of transferring data unchanged. An I-path can always be constructed between two registers with the addition of interconnect and multiplexers. Edges can be added to the connectivity graph to represent this new path. These added edges will have a very large weight since the additional multiplexer and interconnect add significant overhead. The edge weights indicate the overhead involved in forming an I-path between the two components represented by the nodes. Figure 3-5c shows the data path of Fig. 3-5a modified to have an I-path from register *2* to register *1*. Figure 3-5d shows the corresponding modified connectivity graph.

The highlighted edges in Fig. 3-5b show a path for a specific merged orthogonal scan implementation. There is one merged orthogonal scan path using the existing I-path from primary input *B* to register *2* to scan data in, the I-path constructed through the adder to connect registers *2* and *1*, and the existing I-path from register *1* to primary output *Z* to scan data out ($B \Rightarrow 2 \stackrel{+}{\Rightarrow} 1 \Rightarrow Z$). In the shorthand notation, $\stackrel{+}{\Rightarrow}$ indicates that the adder is used for that segment of the scan path and $\Rightarrow$ indicates a connection between registers

that uses no functional units other than multiplexers. $\stackrel{*}{\Rightarrow}$ will indicate that a multiplier is used.

In this example the only logic overhead added to the data path is that required to modify the adder to pass data unmodified and form an I-path. No additional interconnect is needed for the merged orthogonal scan path, nor are any additional scan-in or scan-out pins necessary since existing primary inputs and outputs are used during scan. Appendix II describes several other merged orthogonal scan path implementations.

An interesting benefit of merged orthogonal scan paths is the elimination of hold time problems often associated with scan path insertion. Replacing the bistables in the design with scannable bistables and connecting them to form the scan path, as is done in traditional scan paths, often results in a circuit that does not satisfy the bistable hold times because of the short paths between bistables during scan. These short paths can be padded with buffers to increase the propagation delay, but this can increase the scan path overhead. Since merged orthogonal scan paths attempt to make use of the existing data paths, the scan paths are not any shorter than the functional paths, and there are no hold time violations — assuming, of course, that the original circuit had no hold time problems.

When the merged orthogonal scan implementation is determined, modifications to the data path logic and the control logic may need to be made. Some of the functional units in the data path may need to be modified so that they can transfer the scan data, and control signals must be modified to allow correct operation during scan.

Multiplexers used during merged orthogonal scan require no modification, though multiplexer select signals may need to be augmented as described below. 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

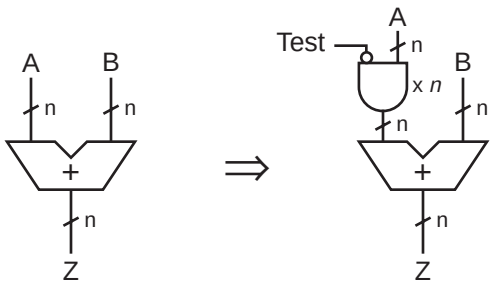


Figure 3-6. Adder modified for merged orthogonal scan

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 merged orthogonal scan data to be passed unmodified. This additional logic is called *masking logic*. For example, the adder in Fig. 3-5a 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 *2* will be transferred to the adder output. This modification is shown in Fig. 3-6. The data on the right input will now be passed through the adder since $B + 0 = B$. Zero is an *identity value* for the adder. A similar procedure can be used for other functional units.

The control logic must also be modified so that the control, enable, and select signals will all be asserted correctly during the scan operation. The multiplexer select signals, register enable signals, and functional unit control signals of components used in the merged orthogonal scan path may require modification so that multiplexers pass the necessary data, registers are enabled at the right times and functional units perform the required operations. These modifications to the control logic make use of the global test mode signals and require at most one logic gate per control, enable, or select signal for each configuration, as shown in Fig. 3-7. The signals may then be forced to appropriate values during the merged orthogonal scan operation.

Once the data path has been synthesized, the merged orthogonal scan path determined, and the functional units and control logic have been modified, the final data path with the merged orthogonal scan implementation is completed. The resulting data

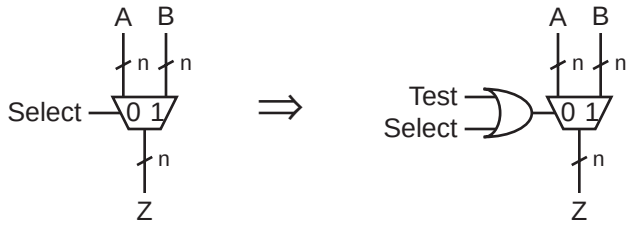


Figure 3-7. Multiplexer with modified select signal

path circuit for the data path in Fig. 3-5a is shown in Fig. 3-8. The additional logic is shaded. Two OR gates are added to the multiplexer select signals, and *n* AND gates are added to the adder, where *n* is the width of the data path.

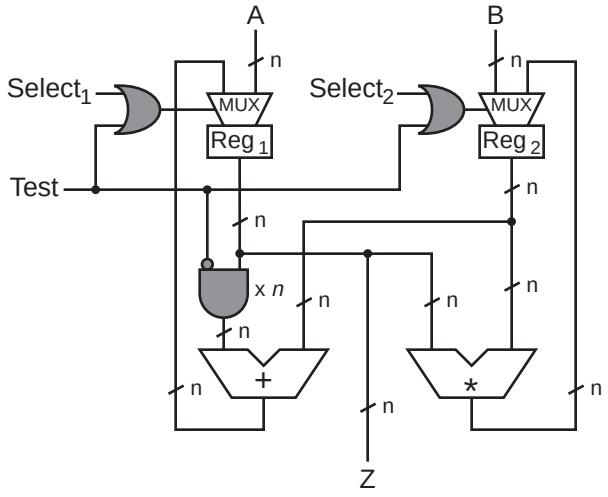


Figure 3-8. Data path modified for merged orthogonal scan

3.2.3 Results

Orthogonal scan has been added to Stanford CRC's synthesis-for-test tool, *TOPS*. Table 3-1 shows the percentage reduction in the overhead for the orthogonal scan compared to the traditional scan approach for four benchmark circuits — three from the HLSW92 benchmark circuits (*diffeq*, *ellipf*, *gcd*)[Dutt 92], and a circuit described in [Tseng 86] (*tseng*). The overhead reduction is calculated as

$$\%\text{Reduction} = \frac{\text{Overhead}_{\text{TradScan}} - \text{Overhead}_{\text{OrthScan}}}{\text{Overhead}_{\text{TradScan}}} \times 100 .$$

Table 3-1. Overhead reduction — merged orthogonal scan compared to traditional scan

Circuit	% Reduction
diffeq	13.3
ellipf	19.5
gcd	46.6
tseng	2.0

The orthogonal scan path has about 20% less overhead, on average, than the traditional scan path.

3.3 Synthesis for merged orthogonal scan

3.3.1 Overview

Merged orthogonal scan paths can be inserted into a data path, whether it was designed by hand or through synthesis, and result in lower overhead than if traditional scan was used [Norwood 96b]. However, if the data path is synthesized specifically for merged orthogonal scan, the final circuit can be smaller than when merged orthogonal scan is inserted after the design. Various modifications can be made to the synthesis operations to target the final result to a merged orthogonal scan implementation.

The register allocation and binding algorithm can be modified to target merged orthogonal scan implementations. The data flow information included in the scheduled and operation bound DFG can be used by the register binding algorithm to guide the final register binding toward one that allows for the insertion of a low overhead merged orthogonal scan path. Before the merged orthogonal scan path is inserted, the data path obtained with this modified synthesis procedure may actually be larger than the data path obtained with the normal synthesis. However, the goal is to make the final data path, with the merged orthogonal scan inserted, obtained with this modified synthesis procedure smaller than the final data path, also with the merged orthogonal scan inserted, obtained with the normal synthesis.

The modifications to the register allocation and binding algorithm attempt to insure that a merged orthogonal scan implementation is possible that does not unduly affect the overall data path size by appropriately allocating and binding the registers. The merged orthogonal scan implementation is just one criterion by which the registers are allocated and bound; the number of registers and multiplexers and the sizes of the multiplexers required for the final register binding is also a consideration, as it is with the original algorithm. Determining the merged orthogonal scan paths during synthesis does not guarantee the best scan implementation due the use of heuristics, but our results for several high-level synthesis benchmark circuits show that the final areas tend to be smaller. Section 3.3.2 describes the modifications to the high-level synthesis algorithms. Appendix II provides more details about the modifications.

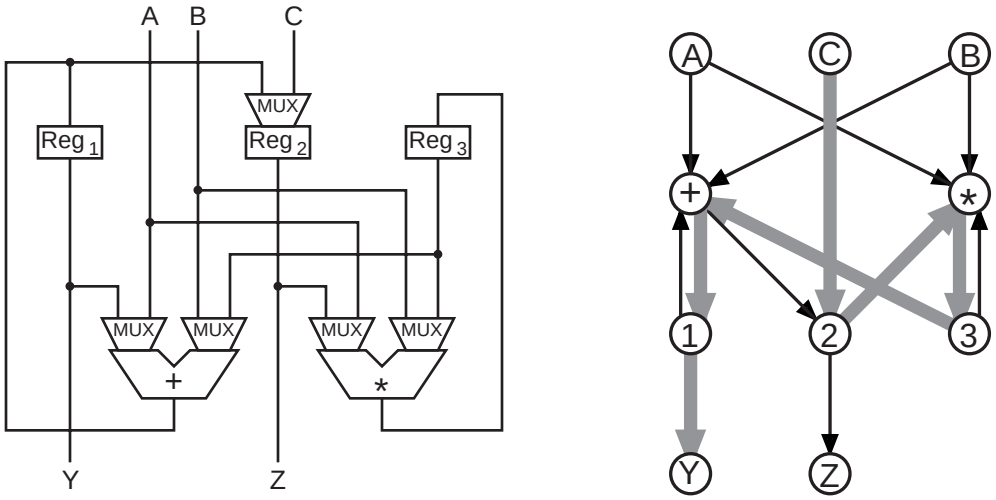
3.3.2 High-level synthesis modifications

Our modified register allocation and binding algorithm for merged orthogonal scan consists of six steps:

1. Create the register conflict graph.
2. Color the register conflict graph.
3. Create the data connectivity graph (defined below).
4. Find a merged orthogonal scan implementation.
5. Modify the register conflict graph.
6. Recolor the register conflict graph.

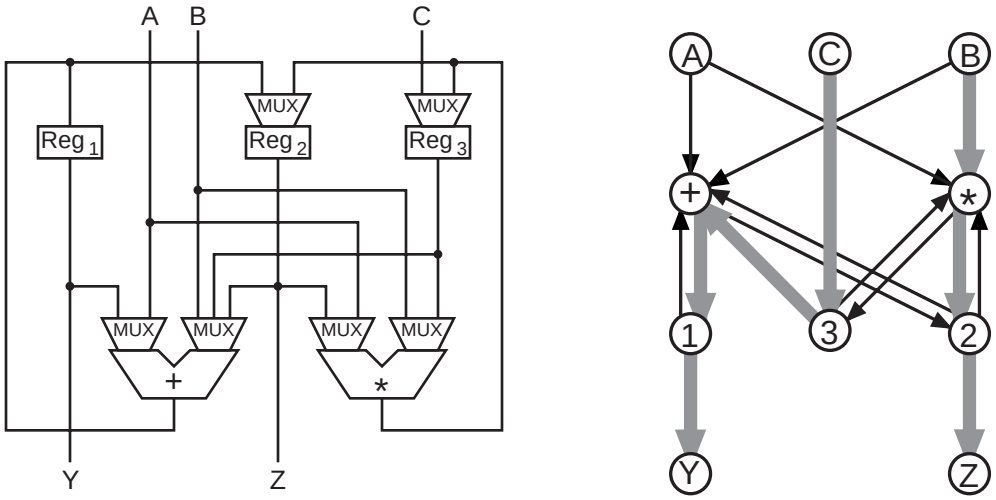
Appendix II describes these six steps in detail.

Figure 3-9 shows a data path synthesized for merged orthogonal scan and the corresponding connectivity graph with the merged orthogonal scan implementation ($C \Rightarrow 2 \stackrel{*}{\Rightarrow} 3 \stackrel{+}{\Rightarrow} 1 \Rightarrow Y$) highlighted. Figure 3-10 shows a data path synthesized normally and the corresponding connectivity graph with the merged orthogonal scan implementation ($C \Rightarrow 3 \stackrel{+}{\Rightarrow} 1 \Rightarrow Y$ and $B \stackrel{*}{\Rightarrow} 2 \Rightarrow Z$) highlighted. These two data paths implement the same function, but they have been synthesized differently. The first



a) **b)**

Figure 3-9. Data path after synthesis for merged orthogonal scan: a) data path logic; b) connectivity graph



a) **b)**

Figure 3-10. Data path without synthesis for merged orthogonal scan: a) data path logic; b) connectivity graph

merged orthogonal scan implementation includes only three multiplexers in the scan path: the multiplexer on the input to register *2*, the multiplexer on the left input of the multiplier, and the multiplexer on the right input of the adder. The second merged orthogonal scan implementation includes four multiplexers in the scan path: the

multiplexer on the input to register *3*, the multiplexer on the right input of the multiplier, the multiplexer on the input of register *2*, and the multiplexer on the right input of the adder. The synthesis for merged orthogonal scan results in a smaller final design since the data path is smaller and fewer multiplexer address signals are modified for the merged orthogonal scan implementation.

3.3.3 Results

The register binding algorithms in *TOPS* have been modified to target orthogonal scan. Table 3-2 shows the percentage reduction in the overhead for the orthogonal scan with normal synthesis and the orthogonal scan with modified synthesis compared to a traditional scan approach. Table 3-2 contains the same benchmark circuits as Table 3-1. The overhead reduction is calculated as

$$\%\text{Reduction} = \frac{\text{Overhead}_{\text{NormSynth}} - \text{Overhead}_{\text{ModSynth}}}{\text{Overhead}_{\text{NormSynth}}} \times 100.$$

The orthogonal scan path with the normal synthesis has about 20% less overhead, on average, than the traditional scan path. Modifying the high-level synthesis algorithms to target orthogonal scan results in about 40% less overhead, on average, than the traditional scan path.

3.4 Summary

Merged orthogonal scan is a synthesis-for-scan technique for data path logic. The orthogonal scan path data flow follows the normal data path data flow and is orthogonal to the data flow of a traditional scan path. This shift in the direction of data flow during

Table 3-2. Overhead reduction — synthesis for merged orthogonal scan

Circuit	% Reduction Normal Synthesis	% Reduction Modified Synthesis
diffeq	13.3	45.4
ellipf	19.5	40.7
gcd	46.6	52.9
tseng	2.0	8.4

scan permits much of the merged orthogonal scan path to be shared with the functional logic and interconnect, thereby reducing the scan path overhead. The merged orthogonal scan path is ordered to minimize the scan overhead. Data path circuits with merged orthogonal scan paths tend to be smaller than circuits with traditional scan paths.

The final size of the circuit with merged orthogonal scan can be further reduced if the merged orthogonal scan path is considered during the high-level synthesis of the data path. The register binding algorithm can be modified to consider the merged orthogonal scan path order so that a merged orthogonal scan path can be inserted into the final circuit with little overhead. Our data shows that synthesis targeting merged orthogonal scan results in circuits with less area than circuits with merged orthogonal scan obtained with normal synthesis.

Chapter 4
Delay Testing with Scan

4.1 Previous work

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. For example, the set of two-pattern tests that can be applied can be restricted [Savir 92], the bistables can be modified to hold two values [Malaiya 83] [Glover 88] [Cheng 91], additional logic can be added to obtain the second pattern [Touba 96], or the scan path order can be constrained to facilitate the application of two-pattern tests [Mao 90] [Hurst 95].

4.2 Constrained scan path ordering

4.2.1 Overview

Building on both the modified scan and the constrained scan approaches, 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.

Our technique can be applied to both register based designs, such as data path logic, and to non-register based designs, such as control logic. However, the discussion in this paper 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] (described in Chapter 3), 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.

The scan path order can be constrained so that two-pattern tests can be applied with little, or no, modifications to the scan path elements. Figure 4-1 illustrates this technique. Figure 4-1a 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. 4-1a and is shown in more detail in Fig. 4-1d. 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

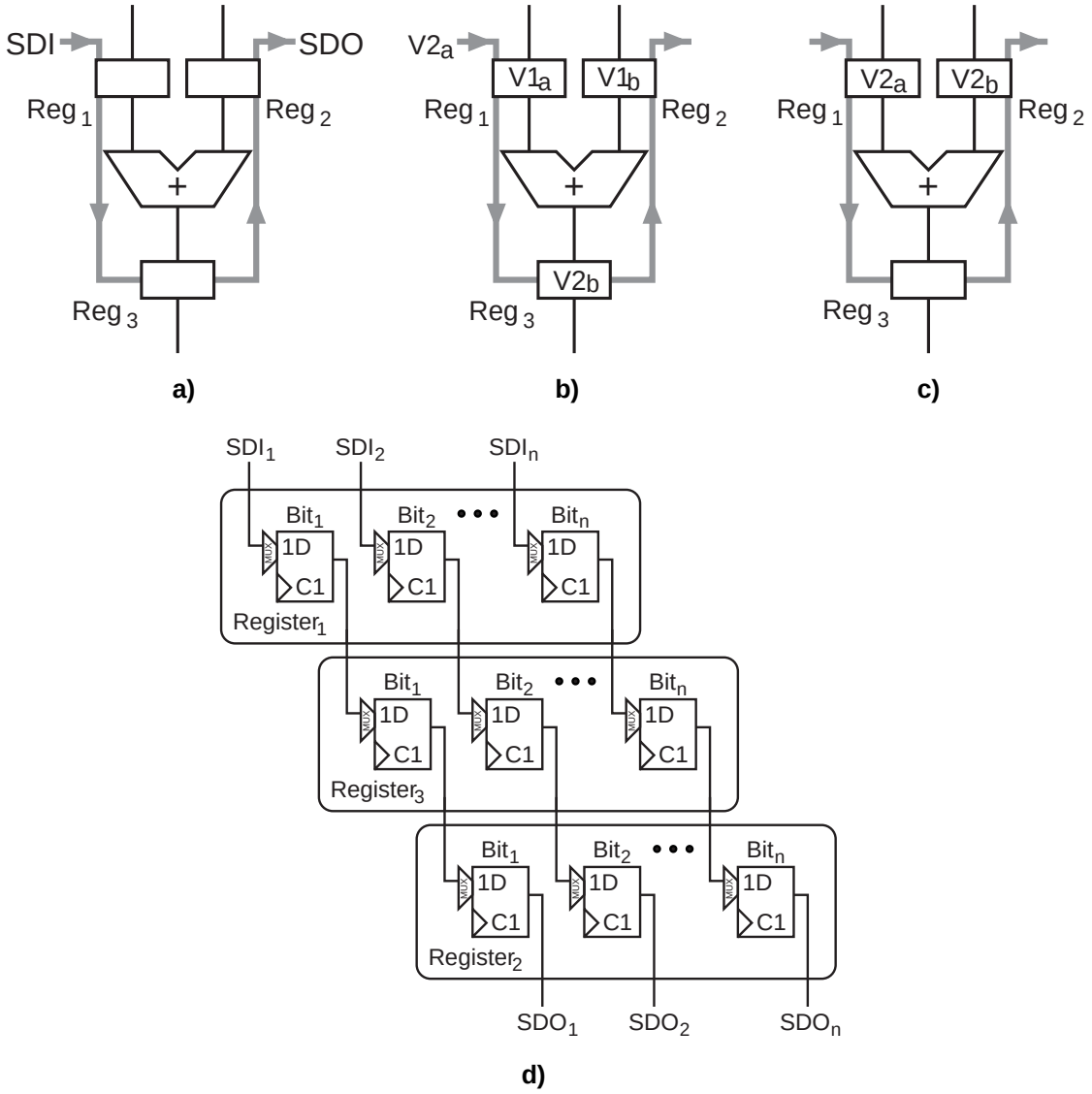


Figure 4-1. Scan path for delay testing: a) data path fragment; b) applying first pattern; c) applying second pattern; d) multiple scan paths

path is the same, and the multiple, parallel scan paths can be treated as a single, *n*-bit wide scan path composed of registers. This forms an orthogonal scan path without logic sharing.

The scan path order shown in Fig. 4-1a ($1 \rightarrow 3 \rightarrow 2$), 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. 4-1b. 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 4-1c 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 is scanned in. In this way, the scan path can be used to apply the two-pattern test when the scan path has been constructed appropriately.

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.

4.2.2 Register grouping

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.

An obvious way to group the registers is by looking at the input cones 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

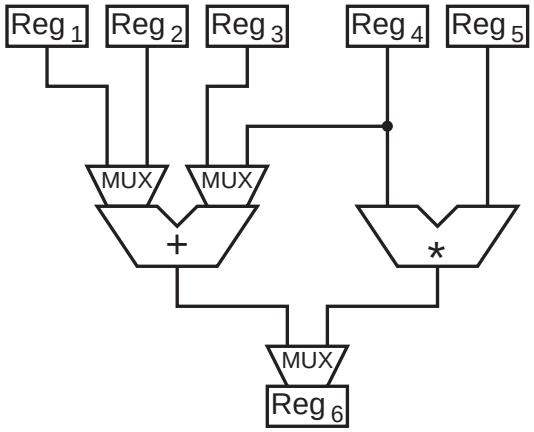


Figure 4-2. Data path fragment

group. In this way, a data path with *n* registers has *n* groups. For example, using the data path fragment in Fig. 4-2 (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, and, therefore, grouping based on register input support sets is not very effective.

Smaller groups may be obtained by determining the register grouping based on functional unit inputs, rather than on register inputs. Smaller groups 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 in each operation form a group. For example, again using the data path fragment in Fig. 4-2, 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. Appendix III provides more details about register grouping using the functional unit inputs.

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. 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 first two register grouping techniques should be used.

4.2.3 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. 4-3 for the three previous groupings. The compatibility graph shows which registers may be adjacent in the scan path.

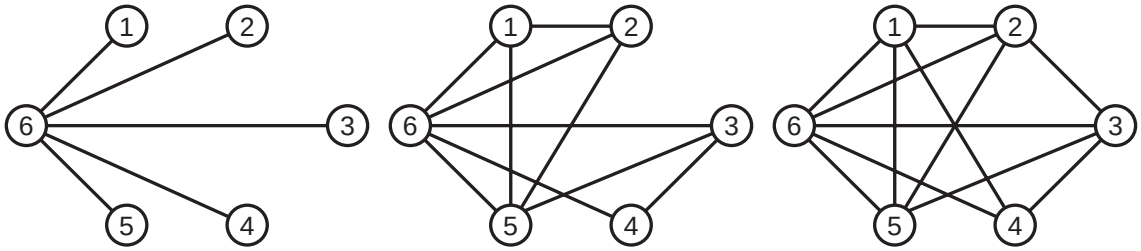


Figure 4-3. Compatibility graph: a) based on register inputs; b) based on functional units; c) based on high-level information

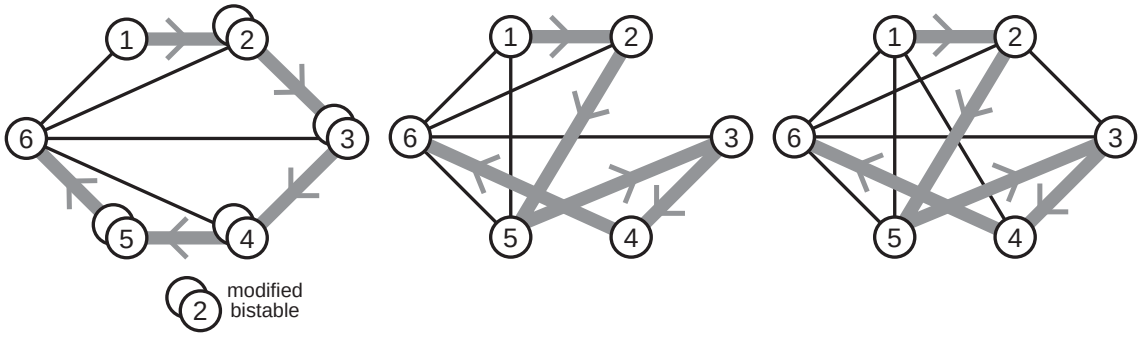


Figure 4-4. Scan path order: a) based on register inputs; b) based on functional units; c) based on high-level information

From the compatibility graph, paths including all the registers can be found such that no two registers from the same group will be 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 can 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 added to the scan path without losing the ability to apply arbitrary two-pattern tests. Figure 4-4 shows the compatibility graphs with a possible scan path order highlighted. Figure 4-4a illustrates the need for modified registers because of the many conflicts between registers. Appendix III discusses details on constructing the scan path and using it to apply two-pattern tests.

4.2.4 Results

TOPS, Stanford CRC's synthesis-for-test tool has been modified to group a data path's registers and create a scan path to allow the application of two-pattern 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 4-1 and Table 4-2.

Table 4-1 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

Table 4-1. 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

Table 4-2. Constrained scan path for data path logic

Circuit	#Registers	Grouping based on register inputs #Modified registers	Grouping based on functional unit input #Modified registers	Grouping based on high level info #Modified registers
diffeq	7	4	0	0
ellipf	12	9	0	0
gcd	2	1	1	1
tseng	5	2	0	0

bistables to be modified.

Table 4-2 shows the results for four data path circuits [Tseng 86] [Dutt 92]. The table 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. Results are shown for the three different grouping strategies discussed in Sec. 4.2.2.

The constraints on the scan path order are greatest when the grouping is done according to the register input support sets. In this 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 two-pattern tests.

4.3 Summary

Arbitrary two-pattern delay fault tests are difficult to apply with a standard scan path. However, careful ordering of the registers in the scan path can result in a scan path such that two-pattern tests can be applied. The scan path order is constrained based on register groupings where only registers in one group are needed to apply any particular two-pattern test. The scan path is ordered so that no two registers from the same group are adjacent to each other in the scan path. Three different approaches to forming the register groupings have been presented, and it is shown that register groupings based on functional criteria result in less overhead than register groupings based on structural criteria.

Chapter 5
Concluding Remarks

This dissertation presents my contributions to the synthesis of circuits with scan path implementations. Scan paths are a design-for-test technique that improves the testability of sequential circuits. However, inserting a scan path can increase the area and decrease the performance of the circuit. Synthesis-for-scan can reduce this scan path overhead by considering the scan path insertion during the synthesis of the circuit. This integrated approach to designing testable circuits can give better results than a two-phase approach where the testability features are inserted after the circuit is designed.

The various scan techniques presented here make use of two basic concepts:

1. Sharing the functional logic and interconnect with the test logic and interconnect can reduce the overhead due to the insertion of a scan path.
2. Taking the scan path implementation into account during the synthesis of the logic can result in more sharing of the functional and test logic and thereby minimize the scan path overhead.

Beneficial scan makes use of the relationships between bistables in random logic to order the scan path during the logic synthesis. The scan path is ordered to maximize the amount of sharing and thereby minimize the scan overhead. Beneficial scan can be considered even earlier in the design process by using a modified state assignment algorithm that targets beneficial scan. The state assignment is made to maximize the amount of sharing that can take place in the final scan path. Appendix I also discusses beneficial scan.

Merged orthogonal scan exploits the regular structure of data path logic to share the functional and test logic and interconnect. The high-level synthesis algorithms can be modified to target merged orthogonal scan and further reduce the scan overhead. Merged orthogonal scan is also described in Appendix II.

The scan path order can be constrained so that arbitrary two-pattern tests can be applied via the scan path. Analysis of the behavior of the circuit to partition the bistables and order the scan path gives better results than analysis of the structure of the circuit. Further details are contained in Appendix III.

All of these scan techniques make use of behavioral information about the circuit to reduce the scan path overhead. As the results show, synthesis-for-scan can reduce the scan overhead. Considering the testability aspects earlier in the design flow can broaden the solution space and lead to a better result.

There are several areas for further investigation. Beneficial scan and orthogonal scan are both full scan techniques, where every bistable is included in the scan path. Both techniques can be applied to partial scan implementations, where only a subset of the bistables are included in the scan path, but the criteria by which to select the subset of bistables is not clear. This work on beneficial scan and merged orthogonal scan has focused on reducing the overhead of scan paths implemented with MD flip-flops. Reducing the scan overhead for other scan implementations, such as dual-port flip-flops or level-sensitive scan design, can also be investigated. The discussion of merged orthogonal scan assumes a data path implemented with multiplexers, as opposed to a bus-oriented structure. Adapting orthogonal scan to a bus structure would require modifications to the scan path ordering algorithms.

References

[Abadir 85] Abadir, M.S., and M.A. Breuer, "A Knowledge Based System for Designing Testable VLSI Chips," *IEEE Design & Test of Computers*, Vol. 2, No. 4, pp. 56-68, August 1985.

[Adham 95] Adham, S., M. Kassab, N. Mukherjee, K. Radecka, et. al., "Arithmetic Built-In Self-Test for Digital Signal Processing Architectures," *Proc. IEEE 1995 Custom Integrated Circuits*, New York, NY, pp. 659-662, May 1-4, 1995.

[Agrawal 88] Agrawal, V., K. Cheng, and D. Johnson, "Designing Circuits with Partial Scan," *IEEE Design and Test* , vol. 5, no. 2, pp. 8-15, April, 1988.

[Anirudhan 89] Anirudhan, P.N., and P.R. Menon, "Symbolic Test Generation for Hierarchically Modeled Digital Systems," *Proc. Intl. Test Conf.*, Washington, DC, pp. 461-469, Aug. 29-31, 1989.

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

[Avra 93] Avra, L.J., "Synthesizing for Scan Dependence in Built-In Self-Testable Designs," *Proc. Intl. Test Conf.*, Baltimore, MD, pp. 734-743, Oct. 17-21, 1993.

[Avra 94] Avra, L.J., "Synthesis Techniques for Built-In Self-Testable Designs," Center for Reliable Computing Technical Report 94-7, Computer Systems Laboratory, CSL TR 94-633, Stanford University, Stanford, CA, Aug. 1997.

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

[Bhatia 93] Bhatia, S., and N.K. Jha, "Synthesis of Sequential Circuits for Easy Testability Through Performance-Oriented Parallel Partial Scan," *Intl. Conf. VLSI Design*, India, pp. 151-154, Jan., 1993.

[Bhatia 94] Bhatia, S., and N.K. Jha, "Behavioral Synthesis for Hierarchical Testability of Controller/Data Path Circuits with Conditional Branches," *Proc. IEEE Intl. Conf. Computer Design*, Cambridge, MA, pp. 91-96, Oct. 10-12, 1994.

[Bhattacharya 96] 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.

[Bhavsar 86] Bhavsar, D., "A New Economical Implementation for Scannable Flip-Flops in MOS," *IEEE Design and Test*, pp. 52-56, June, 1986.

[Chakradhar 94] Chakradhar, S.T., A. Balakrishnan, and V. Agrawal, "An Exact Algorithm for Selecting Partial Scan Flip-Flops," *Proc. Design Automation Conf.*, San Diego, CA, pp. 81-86, June 6-10, 1994.

[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.

[Chickermane 90] Chickermane, V., and J.H. Patel, "An Optimization Based Approach to the Partial Scan Design Problem," *Proc. Intl. Test Conf.*, Washington, DC, pp. 377-386, Sept. 10-14, 1990.

[Chickermane 94] Chickermane, V., J. Lee, and J.H. Patel, "Addressing Design for Testability at the Architectural Level," *IEEE Trans. Computer-Aided Design*, Vol. 13, No. 7, pp. 920-934, July 1994.

[Cox 94] Cox, H., "On Synthesizing Circuits with Implicit Testability Constraints," *Proc. Intl. Test Conf.*, Washington, DC, pp. 989-998, 0ct. 2-6, 1994.

[Cox 95] Cox, H., "Synthesizing Circuits with Implicit Testability Constraints," *IEEE Design and Test*, vol. 12, no. 2, pp. 16-23, Summer 1995.

[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 I_{DDQ}, At-Speed Testing, and Increased Fault Coverage," *Proc. Intl. Test Conf.*, Baltimore, MD, pp. 285-292, Oct. 17-21, 1993.

[Giles 91] Giles, G.L., and J.R. Wilson, "Toggle-Free Scan Flip-Flop," *United States Patent #5,015,875*, May 1991.

[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.

[Gupta 90] Gupta, R., and M.A. Breuer, "The BALLAST Methodology for Structured Partial Scan Design," *IEEE Trans. Comp.*, vol. 39, no. 4, pp. 538-544, April, 1990.

[Gupta 91] Gupta, R., and M.A. Breuer, "Ordering Storage Elements in a Single Scan Chain," *Proc. IEEE Intl. Conf. Computer-Aided Design*, Santa Clara, CA, pp. 408-411, Nov. 11-14, 1991.

[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.

[Kanjilal 93] Kanjilal, S., S.T. Chakradhar, and V.D. Agrawal, "Synthesis Approach to Design for Testability," *Proc. Intl. Test Conf.*, Baltimore, MD, pp. 754-762, Oct. 17-21, 1993.

[Lee 90] Lee, D.H., and S.M. Reddy, "On Determining Scan Flip-Flops in Partial-Scan Designs," *Proc. IEEE Intl. Conf. Computer-Aided Design*, Santa Clara, CA, pp. 322-325, Nov. 11-15, 1990.

[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.

[Lin 89] Lin, B., and A.R. Newton, "Synthesis of Multiple Level Logic from Symbolic High-Level Description Languages," *Proc. IFIP Intl. Conf. VLSI*, Munich, Germany, pp. 187-196, Aug. 16-18, 1989.

[Lin 95] Lin, C.-C., M.T.-C. Lee, M. Marek-Sadowska, and K.-C. 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.

[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.

[Mukund 91] Mukund, S., S. Thanawastien, and T.R.N. Rao, "Efficient Scan-Path and BIST Latches for Static CMOS ASIC Circuits," *Proceedings of the 33rd Midwest Symposium on Circuits and Systems*, pp. 576-579, Aug., 1990.

[Narayanan 92] Narayanan, S., C. Njinda, and M.A. Breuer, "Optimal Sequencing of Scan Registers," *Proc. Intl. Test Conf.*, Baltimore, MD, pp. 293-302, Sept. 20-24, 1992.

[Narayanan 93] Narayanan, S., R. Gupta, and M.A. Breuer, "Optimal Configuring of Multiple Scan Chains," *IEEE Trans. Comp.*, vol. 42, no. 9, pp. 1121-1131, Sept., 1993.

[Norwood 96a] Norwood, R.B., 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.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.

[Norwood 97] Norwood, R.B., and E.J. McCluskey, "High-Level Synthesis for Orthogonal Scan," *Proc. IEEE VLSI Test Symp.*, Monterey, CA, pp. 370-375, April 28-30, 1997.

[Parulkar 95] Parulkar, I., S. Gupta, and M.A. Breuer, "Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead," *Proc. Design Automation Conf.*, San Francisco, CA, pp. 395-401, June, 1995.

[Pradhan 92] Pradhan, D.K., J. Saxena, "A Design for Testability Scheme to Reduce Test Application Time in Full Scan," *Proc. IEEE VLSI Test Symp.*, Atlantic City, NJ, pp. 55-60, April 7-9, 1992.

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

[Schultz 85] Schultz, D.E., "Static Memory Cell with Dynamic Scan Test Latch," *United States Patent #4,554,664*, Nov. 1985.

[Sentovich 92] Sentovich, E.M., K.J. Singh, C. Moon, H. Savoj, et. al., "Sequential Circuit Design Using Synthesis and Optimization," *Intl. Conf. Comp. Design*, Cambridge, MA, pp. 328-333, Oct. 11-14, 1992.

[SIA 94] Semiconductor Industry Association, *The National Technology Roadmap for Semiconductors*, 1994.

[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.

[Vinnakota 92] Vinnakota, B., and N.K. Jha, "Synthesis of Sequential Circuits for Parallel Scan," *Proc. European Conf. Design Automation*, Brussels, Belgium, pp. 366-370, March 16-19, 1992.

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

[Zasio 85] Zasio, J., and L. Cooke, "CMOS Scannable Latch," *United States Patent #4,495,629*, Jan. 1985.