Center for Reliable Computing

TECHNICAL REPORT

Merged Orthogonal Scan

By Robert B. Norwood and Edward J. McCluskey

97-4

(CSL TR # 97-741)

November 1997

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

Abstract:
Merged orthogonal scan is a 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 scan permits much of the merged orthogonal scan path to be shared with the functional logic and interconnect, thereby reducing the scan path overhead. This work describes how a merged orthogonal scan path can be inserted into data path logic in order to minimize the scan path overhead. Modifications to some high-level synthesis algorithms are also presented that can further reduce the scan path overhead.

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.

Copyright © 1997 by the Center for Reliable Computing, Stanford University.
All rights reserved, including the right to reproduce this report, or portions thereof, in any form.

1 Introduction

Scan paths are commonly used in digital designs to improve the testability of sequential circuits since a fully scanned circuit has complete controllability and observability of every bistable element. The sequential circuit can then be treated much like a combinational circuit during test. A scan path is formed by replacing each bistable element in the design with a scannable bistable element. A scannable bistable element has two data inputs and some means to select between the two inputs. One input is used during functional operation, and the other input is used during test operation. The bistables are connected together in a scan path using the bistable test inputs and outputs to form a shift register. During scan operation, data can be scanned (shifted) in and out through the scan path, thereby providing complete controllability and observability to all of the sequential logic. There are many varieties of scan paths [Williams 83], each with its own set of advantages and disadvantages, and each adding overhead of some sort to the circuit [McCluskey 86].

In data path type designs, traditional scan paths, represented in Fig. 1a, connect individual bistables within a register and then connect the registers. For example, bit one

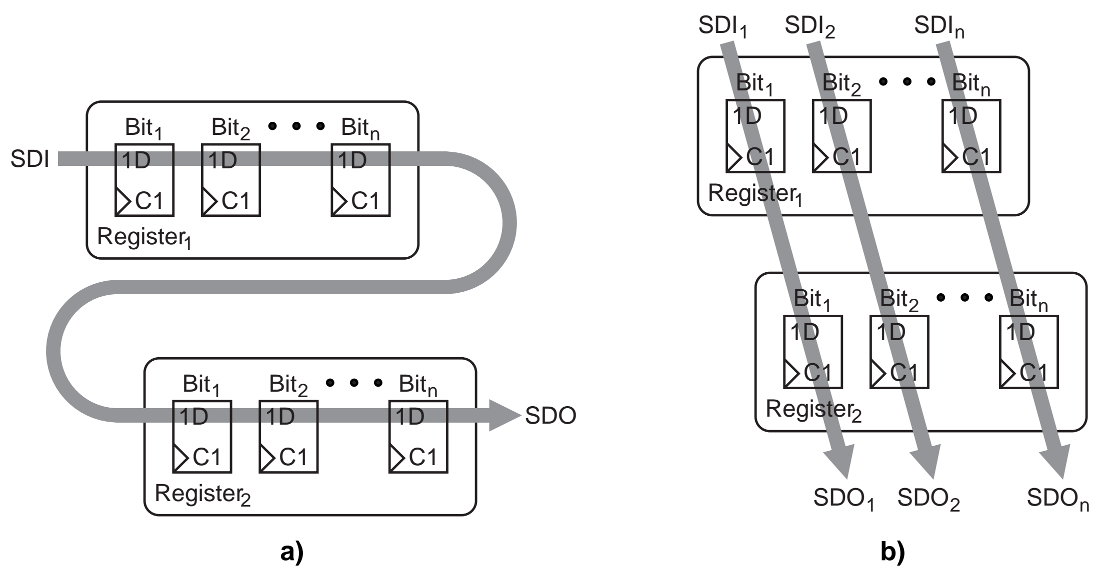


Figure 1. Scan path ordering: a) traditional scan path; b) orthogonal scan path

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 to each other in the physical design, and the interconnect overhead can 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. 1b, 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 2a shows a portion of data path logic consisting of two registers and some combinational logic. Adding a traditional scan path, as shown in Fig. 2b, 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 of the functional logic, Fig. 2c, also requires the addition of multiplexers and interconnect. However, when the orthogonal scan path shares the functional and test logic, Fig. 2d, it may be possible to implement the scan path without adding any multiplexers or interconnect. An orthogonal scan path that shares the functional logic

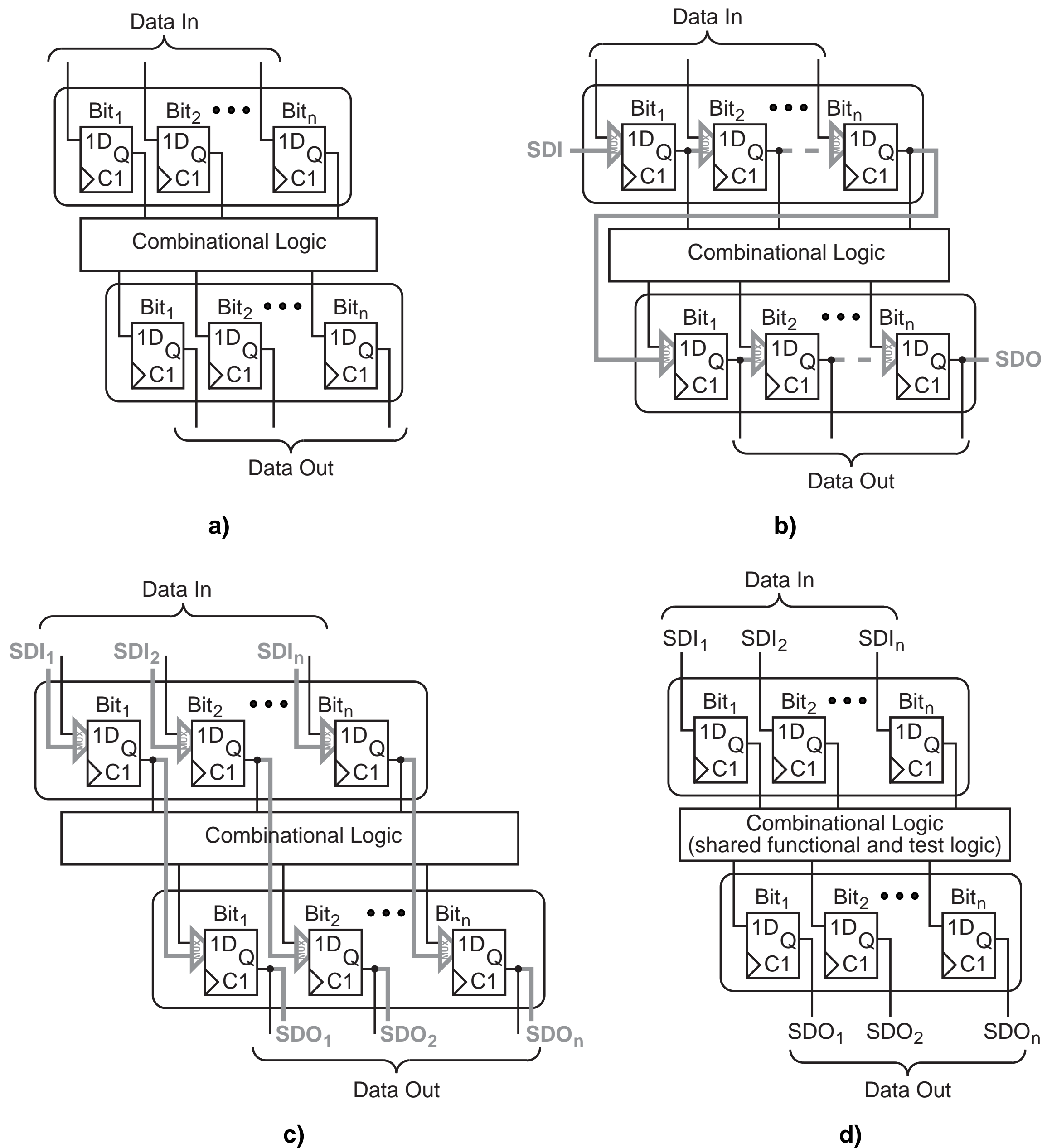


Figure 2. Scan path implementation in data path logic: a) data path fragment; b) traditional scan path; c) orthogonal scan path with no logic sharing; d) orthogonal scan path with logic sharing

with the test logic is called a *merged orthogonal scan path*. The rest of this discussion focuses on merged orthogonal scan paths.

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]

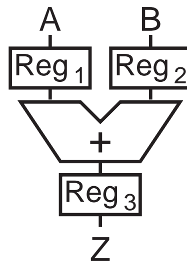


Figure 3. Test pattern justification

[Anirudhan 89] [Bhatia 94] [Chickermane 94]. For example, given the data path shown in Fig. 3 (refer to Sec. 2 for a description of the data path logic symbols used here) 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.

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]. We present the concepts from earlier work [Norwood 96b] [Norwood 97] and builds upon those concepts with some more details and a discussion about orthogonal scan path 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, 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 work focuses on using merged orthogonal scan to implement full scan paths, where every bistable is included in the scan path. Orthogonal scan may also be useful for other testing techniques such as pseudo-random BIST [Avra 92] or arithmetic BIST [Adham 95].

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.

Section 2 describes merged orthogonal scan path insertion. Section 3 discusses issues involved with using a merged orthogonal scan path during test. Section 4 describes a high-level synthesis algorithm that targets merged orthogonal scan path architectures.

2 Merged orthogonal scan

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 Sec. 2.1, or it can be inserted during the synthesis of the data path, as discussed in Sec. 4. The assumption here is that the design is obtained through synthesis of a high-level description, as described in Sec. 4.

Figure 4 defines the logic symbols used in this discussion.

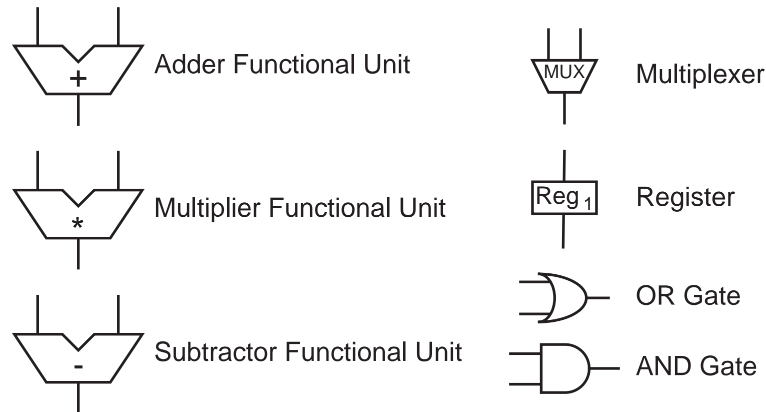


Figure 4. Logic symbol definitions

2.1 Merged orthogonal scan path implementations

2.1.1 Merged orthogonal scan paths

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 merged orthogonal scan path.

Figure 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 5b is a *connectivity graph* showing the connections between components in the data path of Fig. 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

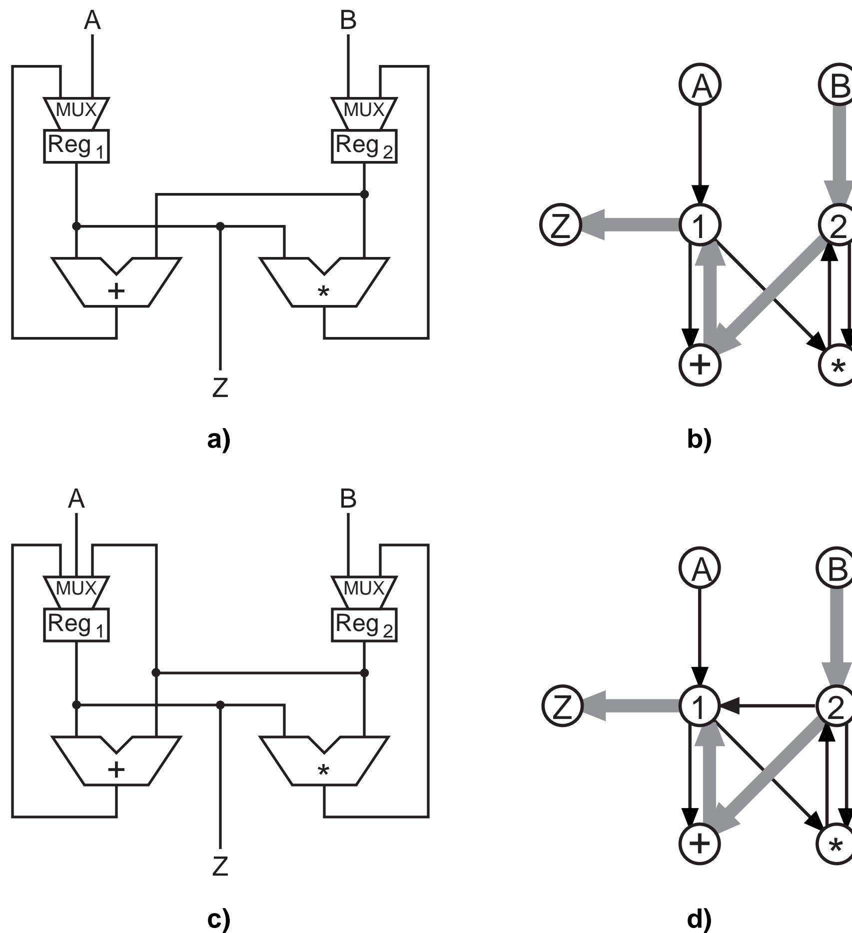


Figure 5. Determining merged orthogonal scan path: a) example data path logic; b) connectivity graph; c) modified data path logic; d) modified connectivity graph

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 associated with using that edge as part of the merged 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. 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 may have

a very large weight since the additional multiplexer and interconnect can add significant overhead. The edge weights indicate the overhead involved in forming an I-path between the two components represented by the nodes. Figure 5c shows the data path of Fig. 5a modified to have an I-path from register *2* to register *1*. Figure 5d shows the corresponding modified connectivity graph.

The highlighted edges in Fig. 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 ($B \Rightarrow 2 \overset{+}{\Rightarrow} 1 \Rightarrow Z$). In the shorthand notation, $\overset{+}{\Rightarrow}$ 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. $\overset{-}{\Rightarrow}$ will indicate that a subtractor is used, and $\overset{*}{\Rightarrow}$ 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. Section 2.2 discusses the modifications to the functional units required to pass data during scan. 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.

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

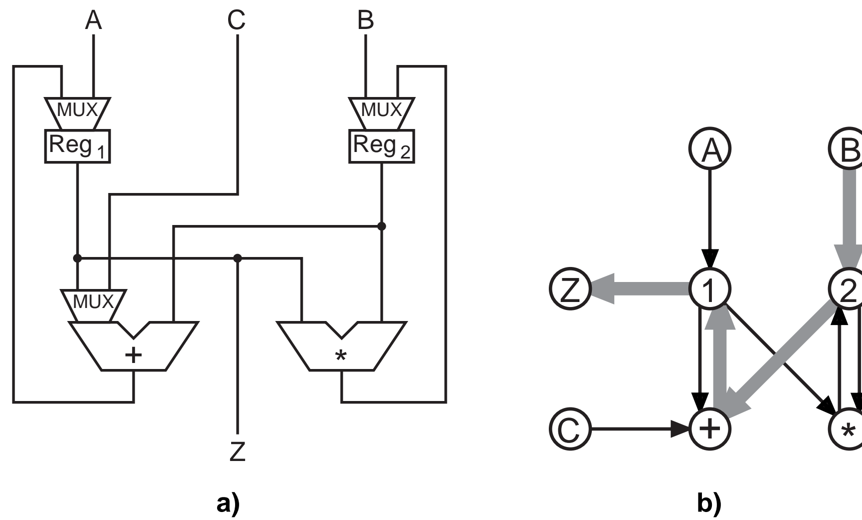


Figure 6. Merged orthogonal scan path using primary input controllability: a) example data path logic; b) connectivity graph

time violations — assuming, of course, that the original circuit had no hold time problems.

2.1.2 Primary input controllability

Some data path designs have primary inputs that connect directly to functional units, such as input *C* to the adder in Fig. 6a. If the merged orthogonal scan path is selected properly, this input controllability can be used to set up the I-path and pass data without having to modify the functional units, much like the cost-free scan technique [Lin 96]. For example, given the data path in Fig. 6a and the merged orthogonal scan path ($B \Rightarrow 2 \overset{+}{\Rightarrow} 1 \Rightarrow Z$) shown in Fig. 6b, no modifications need be made to the adder since input *C* can be used to force a zero on the input and pass the data through the adder during scan. This data path has a scan path with no overhead, other than possible modifications to the control signals (as discussed in Sec. 2.3).

2.1.3 Multiple merged orthogonal scan paths

Multiple merged orthogonal scan paths are also possible. Two or more inputs (and outputs) are used to split the merged orthogonal scan path into multiple parts, each with its own scan-in and scan-out. The test application time is reduced since the length of the

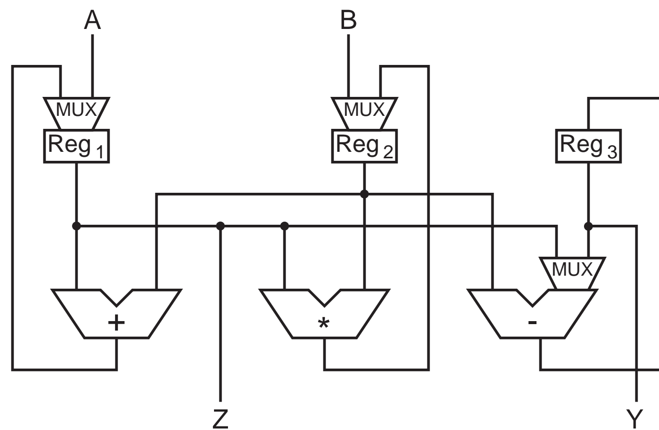
longest scan path is reduced. If the registers in a data path are not all the same size, then multiple merged orthogonal scan paths can be used with registers of different sizes being included in different merged orthogonal scan paths.

Again, heuristics can be used to find multiple paths in the connectivity graph such that each path starts at a primary input node and ends at a primary output node and each register node is included in at most one path. The last requirement of each register node being included in at most one path results from the fact that all the merged orthogonal scan paths in an implementation can be scanned at the same time. This restriction can be removed, but additional control signals are then necessary to select which scan path is active.

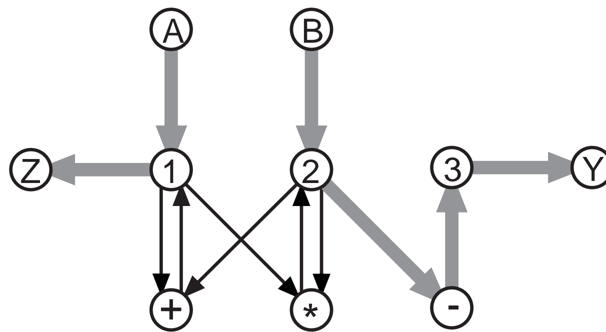
The data path shown in Fig. 7a has the connectivity graph shown in Fig. 7b. There are two merged orthogonal scan paths highlighted in Fig. 7b, $A \Rightarrow \mathit{1} \Rightarrow Z$ and $B \Rightarrow \mathit{2} \overset{-}{\Rightarrow} \mathit{3} \Rightarrow Y$. The connectivity graph in Fig. 7b reflects the fact that only one input to the subtractor (the input from register *2*) may actually be used for the merged orthogonal scan path since the other input can not pass data unmodified, even with the addition of masking logic as described in Sec. 2.2.

2.1.4 Mixed merged orthogonal scan paths

A *mixed merged orthogonal scan implementation* is a merged orthogonal scan implementation that includes only a subset of the registers in the merged orthogonal scan paths with the remaining registers, or bistables, included in a traditional scan path or some other type of scan path. The merged orthogonal scan path and the traditional scan path are scanned concurrently, but they are not necessarily connected together. That is, the traditional scan path has its' own scan in and scan out. For example, the data path in Fig. 7a could have one merged orthogonal scan path including registers *2* and *3* ($B \Rightarrow \mathit{2} \overset{-}{\Rightarrow} \mathit{3} \Rightarrow Y$) while register *1* in scanned with a traditional scan path. This technique might be used if register *1* can not be easily included in a merged orthogonal scan path without adding interconnect and multiplexers. Using a traditional scan path for



a)



b)

Figure 7. Multiple merged orthogonal scan paths: a) example data path logic; b) connectivity graph

a few registers might reduce the overall overhead as compared to adding the interconnect to include all the registers in a merged orthogonal scan path.

2.1.5 Multiple merged orthogonal scan configurations

Data paths that have many more registers than functional units may not be able to share all the test logic with functional logic, and additional interconnect and multiplexers may be needed to form the merged orthogonal scan paths. For example, the data path and connectivity graph shown in Figs. 8a and 8b have four registers, but only one adder, and there is no way to obtain a merged orthogonal scan implementation including all the registers without adding interconnect or using a mixed merged orthogonal scan implementation. If connections are added between register *1* and register *4*, then two

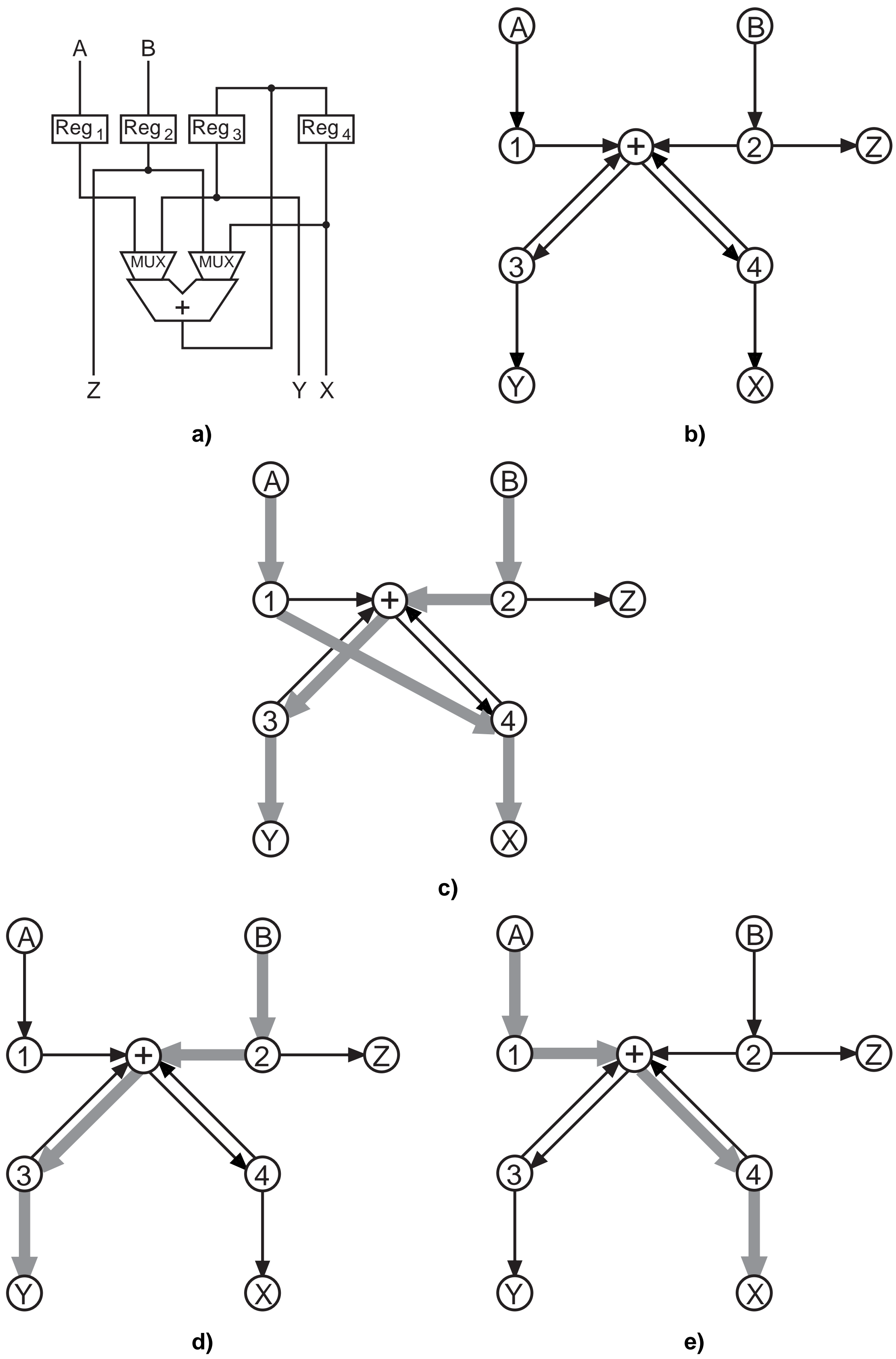


Figure 8. Merged orthogonal scan path with multiple configurations: a) example data path; b) connectivity graph; c) connectivity graph with additional edge; d) configuration one; e) configuration two

merged orthogonal scan paths can be created, as shown in Fig. 8c, but there is additional overhead because of the added interconnect.

However, if the registers already have load enables or gated clocks, or load enables or gated clocks are added, then multiple scan path configurations may be used to scan the registers in phases while the load enables are used to preserve register contents from earlier phases. The net effect of the multiple configurations is the appearance of a single merged orthogonal scan implementation, though different registers are possibly scanned through different hardware configurations. For example, Fig. 8d shows the first configuration, $B \Rightarrow 2 \overset{\pm}{\Rightarrow} 3 \Rightarrow Y$, and Fig. 8e shows the second configuration, $A \Rightarrow 1 \overset{\pm}{\Rightarrow} 4 \Rightarrow X$, for the data path in Fig. 8a. Both configurations use the same adder, and, therefore, only one configuration can be used at a time. Data is scanned into registers *2* and *3* during the first configuration, and then the data is held, using the load enables or gated clocks, while data is scanned into registers *1* and *4*. The net effect is that all four registers have data scanned in and out in four clock cycles, and the multiple configurations can be treated as one logical scan path, though the test mode signals change during the scan operations to select the various configurations. The data is scanned out in a similar manner. Multiple configurations require additional test mode signals so that the various configurations may be selected.

Multiple configurations are distinct from multiple test sessions [Abramovici 90]. The fact that multiple test configurations are used does not change the test pattern generation or test application from when a single configuration is used, other than the need to change test mode signals during scan operations.

If the registers do not already have load enables or gated clocks as part of the normal data path logic, then a subset of the registers can have load enables or gated clocks added so that multiple configurations may be used. In the previous example shown in Fig. 8a, registers *2* and *3* would need to have load enables added. A load enable can be added to a register for roughly the same cost as making the register scannable since both cases

require a multiplexer be added to each bit. The resulting merged orthogonal scan path would still have little interconnect added, and the overall overhead can be less than for a traditional scan path, with the benefit of short test application time.

2.2 Data path logic modifications

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. This section discusses these modifications. Modifications to the control logic are described in Sec. 2.3.

Multiplexers used during merged orthogonal scan require no modification, though multiplexer select signals may need to be augmented as described in the next section. Functional units included as part of the merged orthogonal scan path must be able to pass data unmodified, or possibly inverted, from the input to the output so that the scan function may be implemented. Some functional units, such as shifters or certain ALUs, need no modification to pass data; other functional units, such as most adders or multipliers, do need to be modified in order to pass data. Logic can be added to the functional unit to force an *identity value* on certain inputs to allow the merged orthogonal scan data to be passed unmodified. This additional logic is called *masking logic*. For example, the adder in Fig. 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. 9.

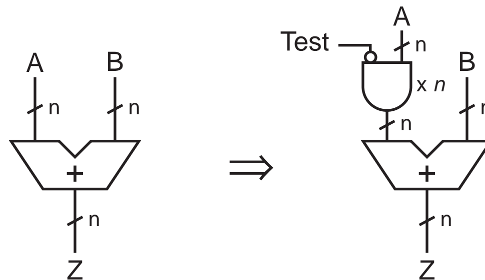


Figure 9. Adder modified for merged orthogonal scan

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 masking logic adds a gate delay to the path between some registers, as opposed to a multiplexer delay being added to every register for traditional scan paths. Traditional scan paths can also require extra interconnect which can increase the load, and consequently the delay, on the flip-flop outputs. The merged orthogonal scan path can be ordered so that a minimum amount of masking logic is added to the critical path, i.e., modify only functional units and functional unit inputs that are not on the critical path. In this way the added delay can be minimized. Instead of adding the 2-input gates directly, the function of the masking logic can be combined with the functional input or the multiplexer associated with that input, if there is such a multiplexer, to reduce the area and delay overhead. These specially designed units would have an extra input, the test mode signal, added to select the merged orthogonal scan mode, as shown in Fig. 9. If multiple configurations are used, then multiple test mode signals are required.

2.3 Control logic modifications

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. 10. The signals may then be forced to appropriate values during the merged orthogonal scan operation.

Figure 10. Multiplexer with modified select signal

2.4 Final data path

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 path circuit for the data path in Fig. 5a is shown in Fig. 11. The additional logic required for orthogonal scan 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.

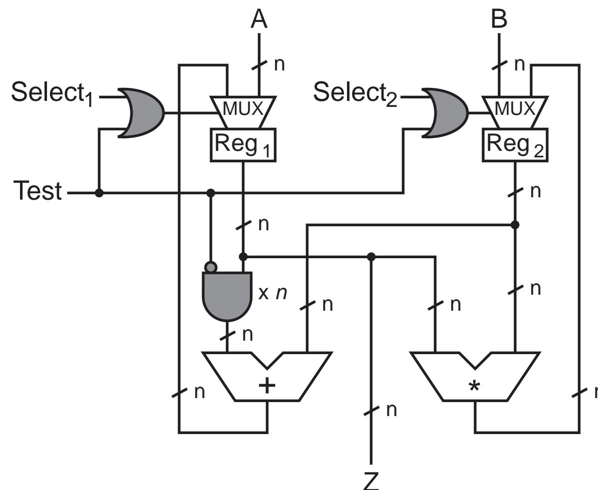


Figure 11. Data path from Fig. 5a modified for merged orthogonal scan

2.6 Experimental results

The Stanford CRC synthesis-for-test tool, *TOPS*, has been modified to add merged orthogonal scan paths to data paths. *TOPS* has been used to add merged orthogonal scan paths to four benchmark circuit examples — three from the HLSW 92 benchmark circuits

Table 1. Benchmark circuit characteristics as synthesized by *TOPS*

Circuit	Data Path Width	# Registers	# Functional Units
diffeq	32-bits	7	2 multipliers 1 adder 1 subtractor 1 comparator
ellipf	16-bits	12	4 adders
gcd	8-bits	2	1 adder/subtractor 3 comparator
tseng	32-bits	5	1 multiplier 3 adders 1 subtractor 1 AND 1 OR

[Dutt 92] (*diffeq*, *ellipf* and *gcd*) and a circuit described in [Tseng 86] (*tseng*). Table 1 shows the data path characteristics of these circuits as synthesized by *TOPS*.

Table 2 compares the areas of 1) the circuit without scan, 2) the circuit with a traditional scan path and 3) the circuit with a merged orthogonal scan path. The area includes the data path logic, the control logic, and the interconnect. The overhead is calculated as

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

where $\text{Area}_{\text{Scan}}$ is the area of the circuit with scan and $\text{Area}_{\text{NoScan}}$ is the area of the circuit without scan. The percent reduction is calculated as

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

where the overheads are determined as shown previously. The area is reported in cell

Table 2. Circuit area in cell units

Circuit	No Scan	Trad Scan	% Ovhd	Orth Scan	% Ovhd	% Reduct
diffeq	102,987	105,458	2.4	105,458	2.1	13.3
ellipf	22,997	24,998	8.7	24,608	7.0	19.5
gcd	6,857	7,063	3.0	6,967	1.6	46.6
tseng	62,523	64,289	2.8	64,254	2.8	2.0

units for the LSI G10 technology [LSI Logic 96].

The data path widths of the four benchmark circuits can be changed without significantly affecting the results. The number of logic gates added for merged orthogonal scan (or multiplexers added for traditional scan) simply scales accordingly.

Table 3 shows the scan characteristics of the benchmark circuits. The number of test pins added to the circuit is shown, along with the logic overhead required to insert the scan path. The last column shows the number of scan shifts that are needed to shift in or out one test vector.

As the results in Table 3 show, the test application time for merged orthogonal scan paths is significantly reduced from that of single traditional scan paths. Merged orthogonal scan paths are shorter because they make use of the buses in the data path and have multiple scan input signals. For an n-bit data path, there are n duplicate scan paths that differ only in bit position — they use exactly the same registers, functional units, inputs, and outputs in exactly the same fashion. The merged orthogonal scan path is at least n times shorter than a single traditional scan path; possibly even shorter if multiple

Table 3. Scan characteristics of benchmark circuits

Circuit	Scan	# Test Pins	Scan Overhead for Data Path	Scan Shifts for Data Path
diffeq	traditional (single path)	1 scan mode 2 scan in/out	224 multiplexers	224
	orthogonal	1 scan mode	128 gates functional 18 gates control	4
ellipf	traditional (single path)	1 scan mode 2 scan in/out	192 multiplexers	192
	orthogonal	1 scan mode	48 gates functional 29 gates control	4
gcd	traditional (single path)	1 scan mode 2 scan in/out	16 multiplexers	16
	orthogonal	1 scan mode	8 gates functional 6 gates control	2
tseng	traditional (single path)	1 scan mode 2 scan in/out	160 multiplexers	160
	orthogonal	1 scan mode	128 gates functional 11 gates control	5

merged orthogonal scan paths are used. The test application time for a traditional scan path can be made comparable to that of a merged orthogonal scan path by using multiple traditional scan paths, but the test application time reduction for merged orthogonal scan comes as a direct result of the orthogonal data flow without any additional effort. Shortening the scan path length increases the number of pins used to scan data in and out. This scan pin increase occurs for merged orthogonal scans, as well as for multiple traditional scan paths, but the total number of pins does not increase since the merged orthogonal scan paths use existing primary inputs and outputs to scan the data in and out.

3 Merged orthogonal scan operation

3.1 Testing with a merged orthogonal scan path

Test application with the merged orthogonal scan path is the same as with a traditional scan path, except that, because of the parallel nature of orthogonal scan, the test vectors are scanned in and out as words instead of as bits. The test procedure consists of:

1. testing the integrity of the merged orthogonal scan path shift operation (discussed in Sec. 3.2) by shifting data through the merged orthogonal scan path;
2. testing the combinational logic using the data in the merged orthogonal scan path to apply the test vectors.

The second step involves the following five steps:

- 2a. shift test vector into the merged orthogonal scan path as n-bit words;
- 2b. apply test vector to primary inputs;
- 2c. apply data to combinational logic and capture response;
- 2d. shift response out and shift next test vector in as n-bit words;
- 2e. repeat 2b through 2d as needed.

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

The automatic test equipment (ATE) used to perform the test can treat the orthogonal scan path(s) as many individual traditional scan paths. Multiple traditional scan paths are commonly in use today, and the ATE capabilities are generally growing to be able to handle more and more scan paths. If the number of scan paths that the ATE can handle is limited, the number of scan I/Os necessary for orthogonal scan can be reduced by concatenating the individual bit-sliced scan paths of the orthogonal scan path. Figure 12 illustrates this concept. Interconnect can be added to internally connect SDO_1 to SDI_2, and SDO_2 to SDI_3, and so on until SDO_{n-1} is connected to SDI_n. This creates a single scan path through the orthogonal scan path with SDI_1 and SDO_n being used to scan data in and out. If instead of concatenating all the bit slices every other bit slice is concatenated, then the number of scan I/Os would be halved. In this way, the number of scan I/Os can be reduced, but, of course, the overhead increases.

This discussion of merged orthogonal scan does not cover the testing of the control

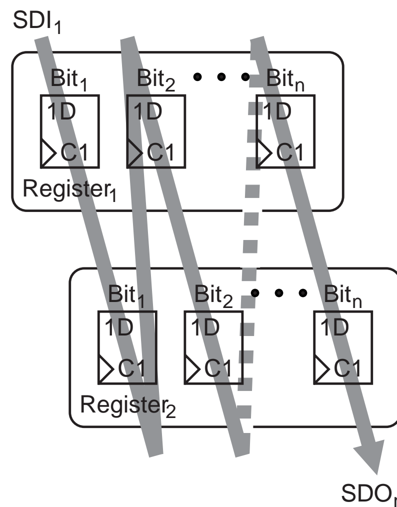


Figure 12. Reducing number of orthogonal scan path scan I/Os

logic. The bistables in the control logic are assumed to be connected in some fashion that is complementary to merged orthogonal scan, e.g., using some form of traditional full scan [McCluskey 86] or beneficial scan [Norwood 96a].

3.2 Merged orthogonal scan path integrity

Since merged orthogonal scan paths use some of the functional logic to implement the scan path, questions arise about the integrity of the scan test in the presence of faults that affect both the functional operation and the orthogonal scan operation. Correct operation of the merged orthogonal scan implementation must be assured before the combinational logic can be tested. If the merged orthogonal scan path is not functioning correctly, some faults may be missed and the merged orthogonal scan path may not be used for diagnostic purposes.

There are three types of faults that must be examined to assure the integrity of the merged orthogonal scan implementation: faults that affect the functional units, faults that affect the registers, and faults that affect the multiplexers. Only faults that affect components included in the merged orthogonal scan path must be considered. Faults that affect components that are not used by the merged orthogonal scan path will not affect the scan operation and can be detected by the application of appropriate test vectors during the testing of the combinational logic.

A functional unit modified for merged orthogonal scan is shown in Fig. 13. During correct merged orthogonal scan operation, the data will be passed from input *B* to output *Z* while node *d* is forced to zero by the high *Test* signal. An error on one of the bits of *B*

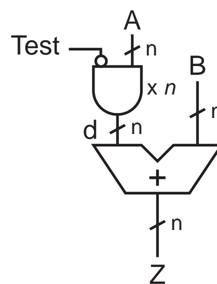


Figure 13. Functional unit modified for merged orthogonal scan

or Z can be detected during the shift operations since the shifted data will be modified. An error on one of the bits of A will be detected during the testing of the combinational logic. If an error causes *Test* to incorrectly be ‘0’, then the left input of the functional unit will not be forced to the identity value, and, as long as input A is not an identity value, the shifted data will be modified and the shift test will detect the error. If an error causes *Test* to incorrectly be ‘1’, then the shift will function correctly and the error can be detected during the testing of the combinational logic. An error causing d to not be an identity value will cause the shifted data to be modified and will be detected during the shift test. An error that does not affect whether d is an identity value will not affect the shift, and the error will be detected during the testing of the combinational logic.

If an error on a functional unit control causes the functional unit to perform the wrong function during orthogonal scan, then the shifted data will be modified, and the error will be detected. If an error does not affect the function performed during orthogonal scan, then the shift will not be affected and the error can be detected during combinational logic testing. All the errors that affect the functional units in the merged orthogonal scan path can be detected, with the case of an error causing an incorrect ‘0’ on *Test* requiring special attention to assure that input A is not an identity value. This case is discussed below with a set of test patterns given to test the scan operation before the testing of the functional logic. All the other errors can be detected during the testing of the shift operation or during the testing of the combinational logic.

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

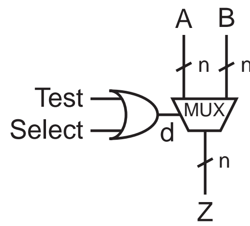


Figure 14. Multiplexer modified for merged orthogonal scan

A multiplexer modified for merged orthogonal scan is shown in Fig. 14. Input *B* and output *Z* are used during the shift operation. Errors on any of the bits of *B* or *Z* will be detected during the shift test. Errors on input *A* will be detected during combinational logic testing. Errors on the multiplexer address signals are more complicated. The multiplexers act as switching logic for the merged orthogonal scan path. An incorrect address signal on a multiplexer, or multiplexers, can change the merged orthogonal scan implementation.

Figure 15a shows a fragment of a merged orthogonal scan path register ordering. The nodes represent registers in the merged orthogonal scan path, and the edges show the flow of data between registers during orthogonal scan. Figures 15b-d show the possible effects when a single multiplexer has an incorrect address select. Figure 15b shows a

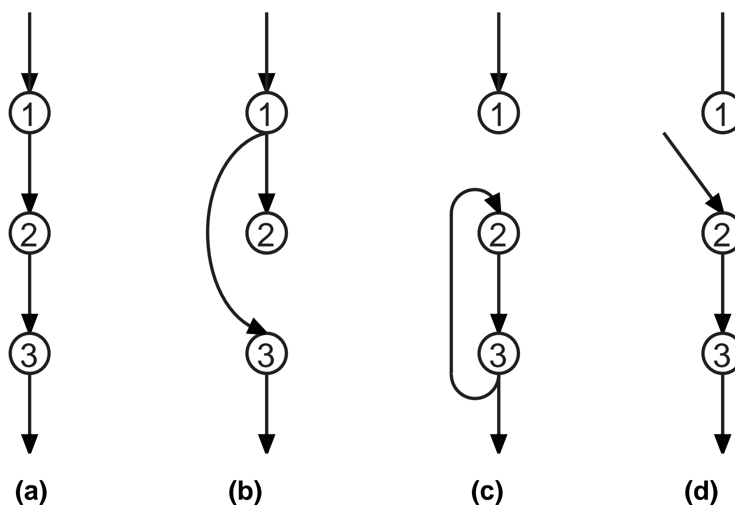


Figure 15. Orthogonal scan path fragments: (a) correct ordering (b) bypassed register (c) loop (d) broken path

forward edge that bypasses a register and shortens the merged orthogonal scan path. Figure 15c shows a backward edge that forms a loop with no entry point. Figure 15d shows an edge added that does not come from another part of the original merged orthogonal scan path. The new edge originates from a functional unit that is not part of the original merged orthogonal scan path. A shortened merged orthogonal scan path, such as in Fig. 15b, can be detected during the testing of the shift operation, as discussed below, since fewer clock cycles are required to scan data through the scan path. Loops in the merged orthogonal scan path, as in Fig. 15c, are easily detected since the loop has no entry and data can not be shifted through it and the scan path can never be initialized.

The third case, shown in Fig. 15d, causes the multiplexer to transfer data from some functional unit not originally in the merged orthogonal scan path. The register with the faulty multiplexer will now either receive data that does or does not correspond to data being shifted in. If the data does not correspond to data being shifted in, then the error is detected by the testing of the shift function. If the data does correspond to data being shifted in, then the merged orthogonal scan path may be shorter than the original merged orthogonal scan path, in which case the same test used to detect the shortened merged orthogonal scan path will detect the error. The data being shifted into the register with the faulty multiplexer may actually still come from the correct register through some other path. In this case, the merged orthogonal scan path may end up being the same length as the original merged orthogonal scan path, and this situation can not be detected. However, the actual merged orthogonal scan path is equivalent to the original merged orthogonal scan path since the order of the registers is the same, and the fault can be detected with the combinational logic test since the test patterns can still be applied. An error could cause multiple multiplexers to behave incorrectly if they share the same control logic. Assuming that the data path is not sequentially redundant, the error will also be detected in a manner similar to that discussed above.

Table 4. Test vectors for testing merged orthogonal scan path

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

As described above, in order to test the scan path for correct operation, a set of patterns must be applied to the merged orthogonal scan path. The set of patterns shown in Table 4 will test a merged orthogonal scan path. m is the number of registers in the merged orthogonal scan path, and n is the width of the data path. The first m patterns are the same vector, but the actual vector used is not important, except that it must not correspond to an identity value for any of the functional units in the data path. The merged orthogonal scan path is filled with this non-identity value so that errors that cause *Test* to incorrectly be a '0' can be detected, as discussed above. The initial m patterns also set the scan path into a known state so that the length of the scan path can be checked by observing when the m+1 pattern is shifted out of the scan path. The final five patterns put all zero to one and one to zero transitions on each bit of the merged orthogonal scan path. This will detect any errors along the scan path, as described above. Other sets of patterns could also be used in place of these five, if so desired. For example, recent work on checking experiments for flip-flops [Makar 97] could be used to determine the set of patterns. If there are multiple merged orthogonal scan paths, as in Fig. 7b, each merged orthogonal scan path should be tested separately. During the testing of a merged orthogonal scan path, any primary inputs not used in that merged orthogonal scan path should be held at a non-identity value to help detect when *Test* is incorrectly a '0', as

described above. If the merged orthogonal scan paths pass the tests, the combinational logic can then be tested as described in Section 3.1.

The above analysis shows that correct operation of the merged orthogonal scan path can be assured with the appropriate tests. Once the merged orthogonal scan path is shown to work correctly, it can be used for diagnosis as well as for test, just like a traditional scan path.

4 High-level synthesis for merged orthogonal scan

Section 2 described how to insert merged orthogonal scan into a circuit. In this section we discuss how the design of the circuit and the determination of the merged orthogonal scan implementation can be combined and performed together resulting in a smaller overall circuit. High-level synthesis is briefly discussed in Sec. 4.1 to provide a basic understanding of the algorithms involved, and Sec. 4.2 discusses modifications to the standard high-level synthesis algorithms to incorporate the insertion of the merged orthogonal scan implementation.

4.1 Overview of high-level synthesis

High-level synthesis takes a behavioral specification of a circuit, such as a VHDL behavioral description, and generates a circuit, both data path and control, that implements that specification. First, a *data flow graph* (DFG) is derived from the behavioral description. The DFG contains information about the data operations and the data flow of the design. Each node in the DFG corresponds to an operation, and the edges correspond to variables. Figure 17a shows an initial DFG for the VHDL description in Fig. 16. Five data operations are represented along with their relationships to each other that are determined by the data flow. Each variable in the DFG is labeled.

The scheduling operation determines the timing of the circuit. In this step the clock cycle boundaries are set, and the operations are assigned to specific clock cycles. Once

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

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

Figure 16. Behavioral VHDL description

the operations are scheduled, they must be bound to particular functional units. Figure 17b shows the initial DFG after scheduling and function binding. Since only a single adder and a single multiplier are used during any one clock cycle, only two

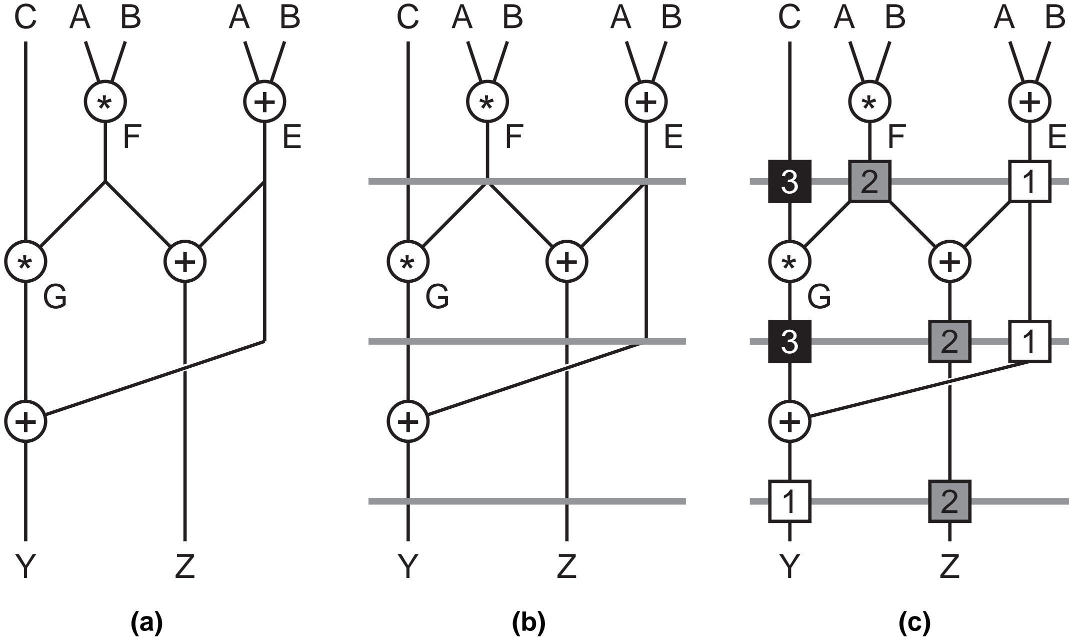


Figure 17. High-level synthesis: a) Initial DFG; b) DFG after scheduling and function binding; c) DFG after register binding

functional units are required for the data path — one adder and one multiplier. Data on output *Y* is valid after three clock cycles; data on output *Z* is valid after two clock cycles.

The final step in high-level synthesis is the allocation and binding of the registers. Register allocation is the process of determining the number of registers that are necessary to implement a specified DFG. Register binding then takes the available registers and maps them to specific variables. Each variable in the DFG that crosses a clock cycle boundary must be bound to a register. For any particular clock cycle boundary, a specific register may be bound to only one variable.

Register allocation and binding can be formulated as a graph coloring problem. A *register conflict graph* is constructed from the DFG after scheduling and function binding. A node is created in the register conflict graph for each variable in the DFG that crosses a clock cycle boundary. An edge is created between two nodes if the variables corresponding to those nodes can not be assigned to the same register because the variables cross the same clock cycle boundary. The register conflict graph for the DFG in Fig. 17c is shown in Fig. 18a. The register conflict graph is then colored with the minimum number of colors where adjacent nodes have different colors. Each color

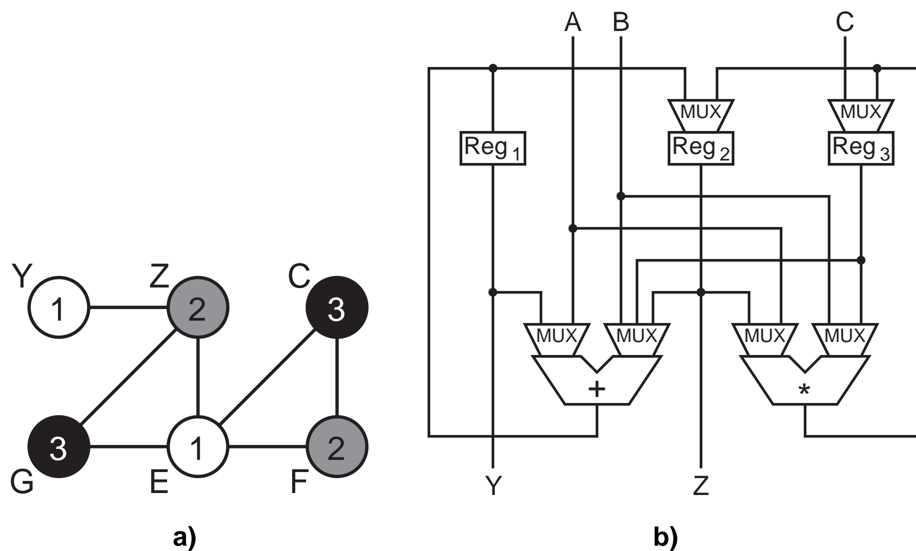


Figure 18. High-level synthesis for DFG in Fig. 17c: a) colored register conflict graph; b) data path logic

corresponds to a register in the data path, and the number of colors indicates the number of registers. One possible coloring of the register conflict graph is shown in Fig. 18a. Each variable in the DFG is then bound to the appropriately colored register, as shown in Fig. 17c. The data path netlist and the control logic can be constructed from the scheduled and bound DFG. The data path in Fig. 18b corresponds to the DFG in Fig. 17c.

4.2 High-level synthesis for merged orthogonal scan

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.

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.

The details of the algorithm are discussed in the rest of this section.

First, in step 1, the register conflict graph is created as described in Sec. 4.1. This register conflict graph provides an initial graph that is then modified to facilitate the implementation of the merged orthogonal scan path. Figure 18a shows the initial register conflict graph for the DFG in Fig. 17b. In step 2, the initial register conflict graph is colored to indicate how many registers need to be allocated for the data path, but the registers are not bound to the variables. This initial coloring provides an estimate of the number of registers that need to be included in the final merged orthogonal scan path. The final coloring used to determine the register binding is obtained after the original register conflict graph is modified in step 5. Three registers are required for the register conflict graph in Fig. 18a. Once the number of registers is known, the merged orthogonal scan paths may be determined.

A *data connectivity graph* (DCG) is generated in step 3 from the scheduled, operation-bound DFG. The DCG, shown in Fig. 19b, shows the flow of data between functional units, primary inputs, and primary outputs. A node is created for each primary input, primary output, and functional unit. A directed edge is created between two nodes if there is an edge in the DFG between the elements corresponding to the nodes. The self-loops (edges with the same head and tail) in the DCG are removed because they do not aid in the formation of a merged orthogonal scan path. The edges in the DCG are

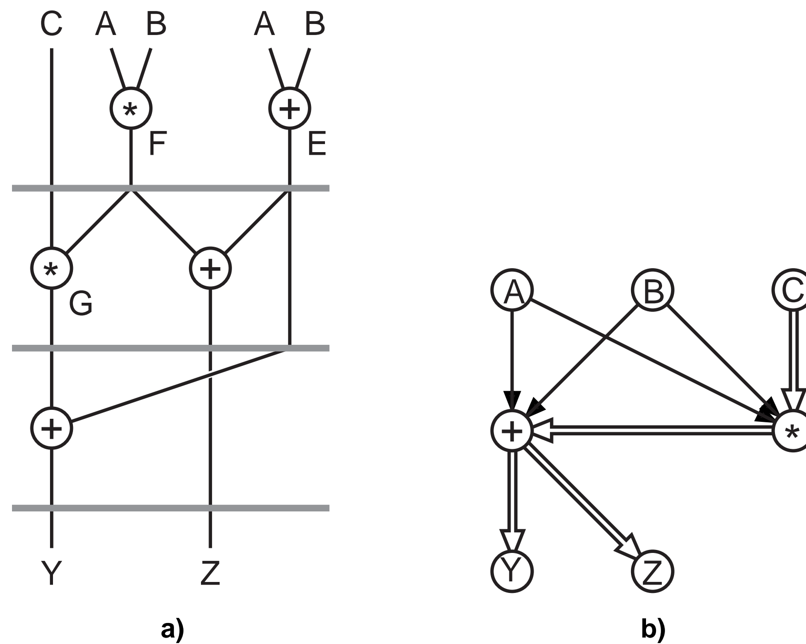


Figure 19. a) DFG before register allocation and binding;
b) data connectivity graph for DFG

marked to indicate whether the corresponding edge in the DFG crosses a clock cycle boundary, thus indicating that the edge will be bound to a register. The DCG is much the same as the connectivity graph described in Sec. 2, but there are no registers included since the variables have not yet been bound to registers. The DCG provides information about the connections between functional units. For example, Fig. 19b shows the DCG for the DFG in Fig. 19a (which is the same DFG as in Fig. 17b) with the outlined edges indicating DFG edges that will be bound to registers. Using the DCG, and the DFG from which it is derived, a merged orthogonal scan path, or orthogonal scan paths, may be determined before the variables are bound to registers.

The DFG and DCG shown in Fig. 19 will be used for an informal discussion of step 4 and step 5, finding a merged orthogonal scan implementation and modifying the register conflict graph. A more detailed description of the heuristic is given later. Examination of the DCG in Fig. 19b shows that there exist several paths from the primary inputs to the primary outputs. A subset of these paths can be used to implement a merged orthogonal

scan path that includes all three registers required for the design. (The number of registers required for the design was obtained in step 2.) Only two choices, $C \rightarrow * \rightarrow + \rightarrow Y$ or $C \rightarrow * \rightarrow + \rightarrow Z$, can include all three registers since only these two paths contain three edges that will be bound to registers. One of these two paths should be chosen so that all three registers can be included in the merged orthogonal scan path. If the path $C \rightarrow * \rightarrow + \rightarrow Y$ is used, then referring to the DFG in Fig. 19a shows that two possible DFG edges (F and G) connect the multiplier and the adder. Either edge F or edge G can be used to implement a merged orthogonal scan path from the multiplier to the adder. If edge G is used, then the final merged orthogonal scan path will include the registers bound to DFG edges C, G, and Y. The registers bound to these three edges must be distinct so that three registers will be included in the merged orthogonal scan path. Since there are no conflict edges between the nodes corresponding to C, G, and Y in the original register conflict graph, these three variables can be bound to the same register(s). Adding conflict edges between these three nodes, to form a clique, will force the corresponding variables to be bound to three different registers. Figure 20a shows the register conflict graph of Fig. 18a with conflict edges added between nodes C, G, and Y. The added edges are highlighted. The path $C \rightarrow * \rightarrow + \rightarrow Z$ could also be used with similar results.

In certain cases, it may be necessary to truncate a path in the data connectivity graph because the path includes more edges that need to be bound to registers than there are registers that need to be included in the merged orthogonal scan path. If each edge is bound to a distinct register, then more registers will be used to form the final data path than are needed. Nodes in the register conflict graph can be merged to force two variables in the DFG to be bound to the same register and thereby shorten a path. For example, if only one register needed to be included in the merged orthogonal scan path for the DFG in Fig. 19a, then DFG edge C and DFG edge Z could be bound to the same register providing a path from a primary input, through a register, and out a primary

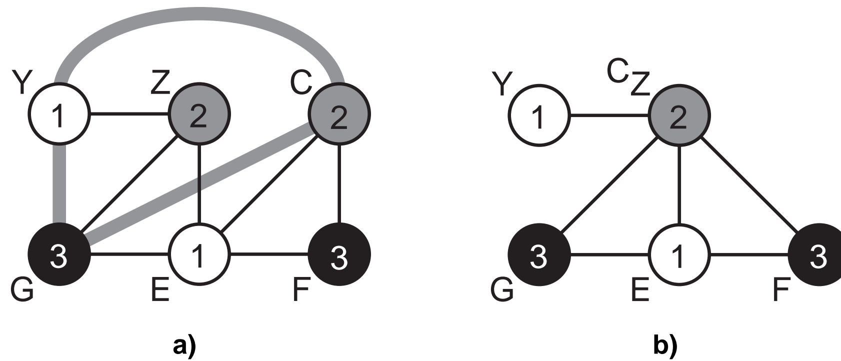


Figure 20. Register conflict graph: a) with added conflict edges; b) with merged nodes

output. Merging the nodes in the register conflict graph that correspond to DFG edges *C* and *Z*, as shown in Fig. 20b, will force the same register to be bound to those two edges and create a merged orthogonal scan path (C ⇒ 2 ⇒ Z). In this way paths can be shortened so that the number of registers allocated is not increased over the initial estimate.

Our heuristic for finding a merged orthogonal scan implementation based on the DFG is outlined here:

1. Create a DCG from the scheduled, operation-bound DFG.
2. Choose a path from a primary input to an output in the DCG that includes edges marked to be bound to registers. Try to pick a path that has at least as many edges marked to be bound to registers as there are registers to be included in the merged orthogonal scan path (as indicated by the initial coloring of the register conflict graph).
3. Choose edges in the DFG to correspond to the path chosen in the DCG. When multiple DFG edges are possible for a specific connection, choose the DFG edges so that the register conflict graph will have the fewest conflict edges added in step 7.
4. Repeat steps 2 to 3, if needed, to include more registers in additional paths.

5. Truncate the path, if necessary, so that the path will include only the number of registers determined from the initial coloring of the register conflict graph.
6. Merge the register conflict graph nodes for DFG edges that should be bound to the same register because of path truncation.
7. Add conflict edges to the register conflict graph so that different registers will be bound to each variable used in the merged orthogonal scan path.

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

Figure 21 shows the DFG and Fig. 22a shows the data path resulting from this technique for the initial DFG in Fig. 19a. This data path and the data path in Fig. 23a both implement the same VHDL behavioral code, but the data path in Fig. 23a was synthesized without considering merged orthogonal scan, and the data path in Fig. 22a was synthesized to target merged orthogonal scan. The connectivity graph for the new data path, with the merged orthogonal scan implementation ($C \Rightarrow 2 \overset{*}{\Rightarrow} 3 \overset{+}{\Rightarrow} 1 \Rightarrow Y$)

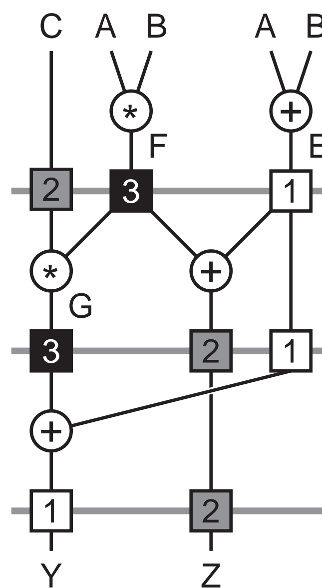


Figure 21. DFG after synthesis for merged orthogonal scan

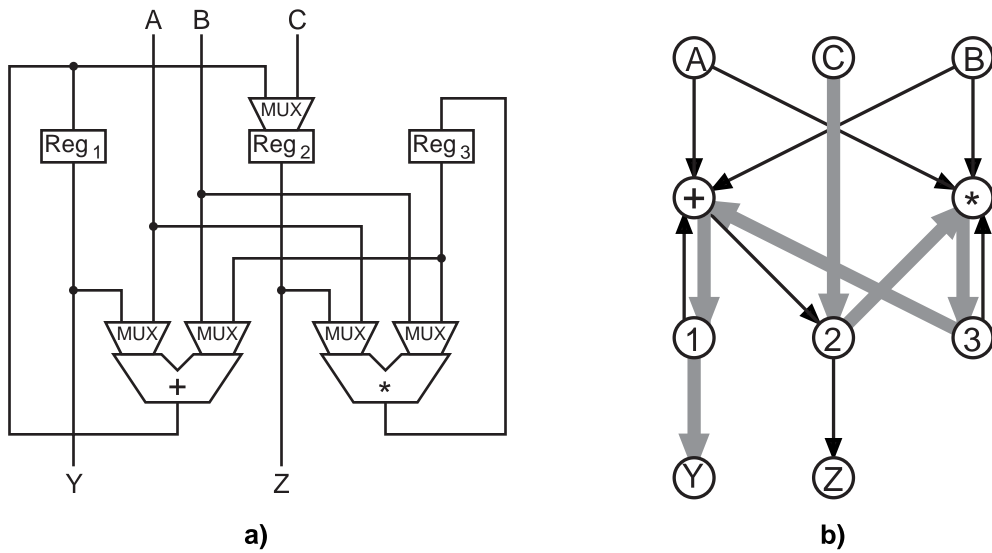


Figure 22. After synthesis for merged orthogonal scan: a) data path logic; b) connectivity graph

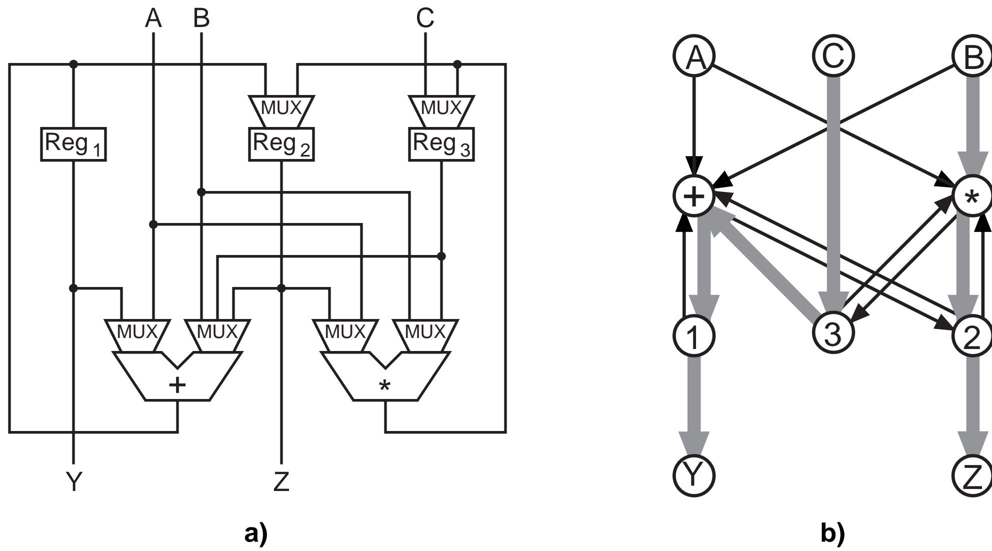


Figure 23. Without synthesis for merged orthogonal scan: a) data path logic; b) connectivity graph

highlighted, is shown in Fig. 22b. The new 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 connectivity graph for the original data path, with the merged orthogonal scan implementation ($C \Rightarrow 3 \overset{+}{\Rightarrow} 1 \Rightarrow Y$ and $B \overset{*}{\Rightarrow} 2 \Rightarrow Z$) highlighted, is shown

in Fig. 23b. The original 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.

4.3 Experimental results

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

Table 5 shows the final circuit areas for four benchmark circuits including the data path logic, the control logic, and the interconnect. The area is shown for the circuit with no scan, with traditional scan, with merged orthogonal scan and normal synthesis, and with merged orthogonal scan and synthesis for merged orthogonal scan. The relative circuit sizes are reported in cell units for the LSI G10 technology [LSI Logic 96]. The overhead is calculated as

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

where $\text{Area}_{\text{Scan}}$ is the area of the circuit with scan and $\text{Area}_{\text{NoScan}}$ is the area of the circuit without scan. Table 6 shows the percentage reduction in overhead for the merged

Table 5. Circuit area for benchmark circuits in cell units

Circuit	No Scan	Traditional Scan		Orthogonal Scan Normal Synthesis		Orthogonal Scan Modified Synthesis	
	Area	Area	Ovhd	Area	Ovhd	Area	Ovhd
diffeq	102,987	105,458	2.4	105,130	2.1	104,336	1.3
ellipf	22,997	24,998	8.7	24,608	7.0	24,184	5.2
gcd	6,857	7,063	3.0	6,967	1.6	6,954	1.4
tseng	62,523	64,289	2.8	64,254	2.8	64,141	2.6

Table 6. Percent reduction in overhead with modified synthesis compared to normal synthesis

Circuit	% Reduction
diffeq	37.1
ellipf	26.3
gcd	11.8
tseng	6.5

orthogonal scan path circuits with the modified synthesis as compared to the merged orthogonal scan path circuits with the normal synthesis. The percent reduction is calculated as

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

where the overheads are determined as shown previously.

For the example circuits shown, traditional scan adds about 5% overhead. Merged orthogonal scan with normal synthesis reduces this overhead by about 20%. When the circuit is synthesized for merged orthogonal scan, the overhead is reduced by about 40% from a traditional scan implementation and by about 20% from a merged orthogonal scan implementation with normal synthesis.

5 Conclusions

Merged orthogonal scan is a 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 scan permits the merged orthogonal scan path to share much of the functional logic and interconnect, thereby reducing the scan path overhead. The merged orthogonal scan path is ordered to minimize the scan overhead. Our results show that 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. Synthesis targeting merged orthogonal scan can lead to smaller circuits than merged orthogonal scan paths alone.

The results of this work show that sharing the test logic and interconnect with the functional logic and interconnect can reduce the overhead due to the insertion of a scan path, and that taking the scan path implementation into account during the synthesis of the data path can result in more sharing of the functional and test logic and further reduce the scan path overhead.

Acknowledgments

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

References

[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.

[Abramovici 90] Abramovici, M., M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, NY, 1990.

[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.

[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 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. 1994.

[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.

[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.

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

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

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

[Makar 97] Makar, S., and E.J. McCluskey, "ATPG for Scan Chain Latches and Flip-Flops," *Proc. IEEE VLSI Test Symp.*, Monterey, CA, pp. 364-369, April 28-30, 1997.

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

[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.

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

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