ORTHOGONAL SCAN: LOW OVERHEAD SCAN FOR DATA PATHS

Robert B. Norwood and Edward J. McCluskey

Center for Reliable Computing
Stanford University
Gates Hall 2A
Stanford, CA 94305

Abstract

Orthogonal scan paths, which follow the path of the data flow, can be used in data path designs to reduce the test overhead — area, delay and test application time — by sharing functional and test logic. Orthogonal scan paths are orthogonal to traditional scan paths. Judicious ordering of the registers in the orthogonal scan path can allow the scan path to be implemented entirely with existing interconnect, resulting in no additional wiring to connect the scan path and no performance degradation due to additional loading on the bistable outputs. Taking the orthogonal scan path into account during high-level synthesis operations such as register allocation allow for a better final solution, but orthogonal scan paths can also be used with non-synthesized data path,. Orthogonal scan paths have roughly half the overhead of traditional scan paths and greatly reduce the test application time. TOPS, Stanford CRC's synthesis-for-test tool, has been modified to implement orthogonal scan paths for synthesized circuits.

1 Introduction

Scan paths are widely used to improve the testability of circuits since a fully scanned circuit has complete controllability and observability of every bistable element. There are many varieties of scan paths [Eichelberger 77] [Williams 83], all of which have overhead due to the additional logic and interconnect [McCluskey 86]. Traditional scan paths are implemented independent of the actual circuit functions. Once the circuit is designed, the scan path is inserted without regard to the logic between flip-flops. By taking the circuit functionality into account during scan path insertion, the overhead due to the test features can be reduced by sharing the functional and the test logic. Previous work has been done in this regard with control paths [Norwood 96], and work is presented here on data paths. The structure of data paths is very well suited to sharing the functional and test logic, and orthogonal scan paths [Avra 92] can be used to reduce the overhead of scan paths.

Traditional scan paths, shown in Fig. 1, connect individual flip-flops within a register and then connect the registers, e.g., 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. An orthogonal scan path, shown in Fig. 2, is orthogonal to the traditional scan path. The flip-flops are connected in the scan path so that bit one of register one connects to bit one of register two, and bit two of register one connects to bit two of register two, and likewise for all the bits of the register. In this way, the scan path follows the normal data path flow, but is orthogonal to the traditional scan path flow.

Judicious ordering of the registers in the orthogonal scan path allows the scan path to be implemented entirely with existing interconnect, resulting in no additional wiring or pins needed to connect the scan path — though some additional interconnect is necessary for the scan path control signals. Orthogonal scan paths also allow functional elements of the data path, such as adders and multipliers, to be used, with slight modifications, to implement the scan path. Orthogonal scan paths are used to scan test vectors in and out with no dependence on which part of the combinational logic is actually going to be tested by which vectors. Other work has looked at using the data path functionality to set up test vectors [Abadir 85] [Anirudhan 89] [Bhatia 94] [Chickermane 94] but these do not actually implement a scan path. [Bhattacharya 96] discusses H-SCAN which exploits some

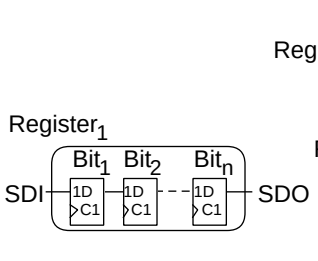

Figure 1. Traditional scan path

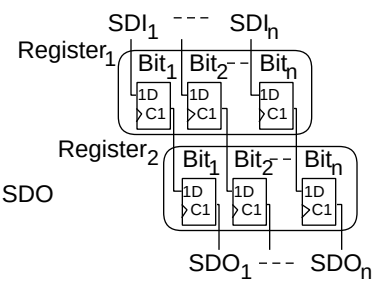

Figure 2. Orthogonal scan path

of the parallelism in a design to reduce the scan overhead, but it does not make use of functional units, and it adds interconnect to the design. This paper focuses on using orthogonal scan to implement full scan paths, where every bistable is included in the scan path, but orthogonal scan may also be useful for other testing techniques such as circular built-in self-test (BIST) [Avra 92] or arithmetic BIST [Adham 95].

An orthogonal scan path is configured to maximize the amount of sharing of the functional elements and to minimize the amount of additional interconnect needed for the scan path. Taking the orthogonal scan path into account during high-level synthesis operations such as function binding and register allocation allow for a better final solution, but orthogonal scan paths can be used with any data path, whether it is synthesized or not. Once the orthogonal scan path is determined, the functional elements are modified to allow them to be used during the scan operations. For example, an adder ($Z = A + B$) can be used to pass data from input A to output Z if the B input is forced to zero. Only a single gate per bit, along with the scan mode select signal, is needed to mask an input.

Using Stanford CRC's synthesis-for-test tool, TOPS, we have synthesized various benchmark circuits using this technique, and results show that orthogonal scan paths can require no additional scan in/out pins; no 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 additional test pins, extra interconnect for the scan path and for control, and the addition of multiplexers to every flip-flop. Orthogonal scan paths also have the added benefit of reducing the length of the scan chain and thereby reducing the test vector application time.

Section 2 describes orthogonal scan path insertion. Section 3 discusses issues involved with using an orthogonal scan path during test.. Section 4 looks at modifications to register allocation and binding during synthesis to benefit orthogonal scan paths. Section 5 gives results for inserting orthogonal scan paths in various benchmark circuits.

2 Orthogonal Scan

Orthogonal scan paths are best inserted during synthesis. At this stage the high level description provides easier analysis of the data path, and the synthesis tools can be enhanced to automatically insert the orthogonal scan path into a design with no additional effort by the designer. There are four steps to inserting orthogonal scan paths:

1. scheduling, allocation and binding
2. determining the orthogonal scan path
3. modifying the functional units
4. synthesizing the control

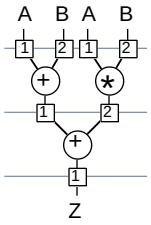

Figure 3. DFG_1

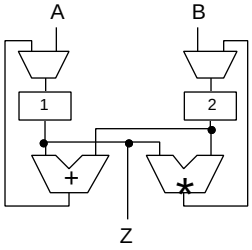

Figure 4. Data path for DFG_1

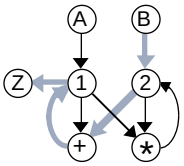

Figure 5. Connectivity graph for data path in Fig. 4

Each of these steps is described in more detail in Sections 2.1 through 2.4.

2.1 Scheduling, Allocation and Binding

Before the orthogonal scan path can be added to the data path, the data path must be synthesized from the data-flow graph (DFG). The DFG specifies the operations to be performed by the data path, and the synthesis process schedules the operations, allocates functional units for the operations and binds the functional units and registers to operations and variables [McFarland 90]. The scheduling, allocation and binding may be performed using any desired methods, but knowledge of orthogonal scan during these steps can improve the resulting orthogonal scan path. Section 4 describes three different register binding algorithms and Section 5 compares the orthogonal scan paths obtained with each.

2.2 Determining Orthogonal Scan Path

Once the DFG has been scheduled, allocated and bound, the structure of the data path is determined. The structure can then be analyzed and an orthogonal scan path found. The 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 DFG shown in Fig. 3 corresponds to the data path shown in Fig. 4; the control signals for the multiplexers are not shown. The numbers inside the boxes in the DFG indicate the register bound to that edge's variable, and the horizontal lines are clock cycle boundaries. Fig. 5 is a connectivity graph showing the connections between components in the data path of Fig. 4. The nodes of the connectivity graph represent the primary inputs and outputs, registers and functional units of the data path.

There are edges between two nodes to indicate a path in the data path between the two corresponding components. Nodes representing multiplexers may be added to the connectivity graph, but since they do not add any information for the data paths discussed here they have been left out of the connectivity graphs in this paper. Analysis of the connectivity graph shows that register *2* can form an orthogonal scan path with register *1* by using the adder. The resulting orthogonal scan path uses input *B* as the scan-in, output *Z* as the scan-out, and the path through the adder to connect registers *2* and *1*. A shorthand notation for this orthogonal scan path is given by $B \Rightarrow 2 \overset{+}{\Rightarrow} 1 \Rightarrow Z$, where $\overset{+}{\Rightarrow}$ indicates that the adder is used for that segment of the scan path and $\Rightarrow$ indicates that no functional unit is used. The scan path is highlighted in Fig. 5.

The only overhead added to the data path is the AND gate (one gate for each bit of the data path) needed to force the first operand of the adder, coming from register *1*, to zero during scan mode. Section 2.3 talks more about the overhead added to the functional units. No additional interconnect is needed for the orthogonal scan path, nor are any additional scan-in or scan-out pins necessary since existing primary inputs and outputs are used during scan. A traditional scan path would require the addition of a multiplexer to each bit of every register, as well as additional pins and interconnect. The additional interconnect required for a traditional scan path adds not only area, but also delay since the loads of the bistable outputs are increased. Orthogonal scan paths do not have this performance penalty due to the loading of the bistable outputs since no additional interconnect is needed to connect the scan path.

Another interesting benefit of orthogonal scan paths is the elimination of hold time problems often associated with scan path insertion. Replacing the flip-flops in the design with scannable flip-flops 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 flip-flop hold times because of the short paths between flip-flops during scan. These short paths can be padded with buffers to increase the propagation delay, or some form of two-phase clocking [LSI Logic 92] can be used to remove the hold time problems, but both of these solutions increase the scan path overhead. Since orthogonal scan paths use 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.

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

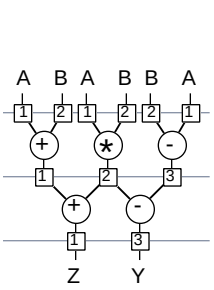

Figure 6. DFG_2

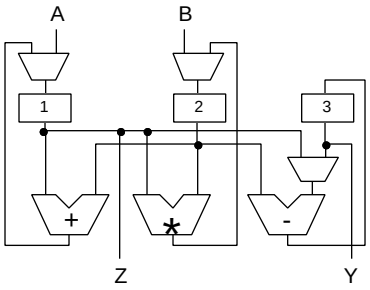

Figure 7. Data path for DFG_2

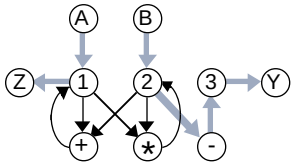

Figure 8. Connectivity graph for data path in Fig. 7 with multiple orthogonal scan paths

since the length of the longest scan path is reduced. The DFG in Fig. 6 has the data path shown in Fig. 7 and the connectivity graph shown in Fig. 8. There are two orthogonal scan paths highlighted in Fig. 8, $A \Rightarrow 1 \Rightarrow Z$ and $B \Rightarrow 2 \overset{-}{\Rightarrow} 3 \Rightarrow Y$. The connectivity graph in Fig. 8 reflects the fact that only one input to the subtractor (the input from register *2*) may actually be used for the orthogonal scan path since the other input can not pass data unmodified.

Data paths that have many more registers than functional units may not be able to include every register in the orthogonal scan path, even with multiple orthogonal scan paths. For example, the DFG, data path and connectivity graph shown in Figures 9, 10 and 11 have four registers, but only one adder, and there is no way to obtain an orthogonal scan path covering all the registers with a single configuration. However, if the registers have load enables 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 scan chain, though different registers are possibly scanned through different hardware configurations. Fig. 12 shows the first configuration, $B \Rightarrow 2 \overset{+}{\Rightarrow} 3 \Rightarrow Y$, and Fig. 13 shows the second configuration, $A \Rightarrow 1 \overset{+}{\Rightarrow} 4 \Rightarrow X$. Data is scanned into registers *2* and *3* during the first configuration, and then they hold their data, using the load enables, 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 clocks, and the multiple configurations can be treated as one logical scan path, though the test mode signals change during the scan operations. The data is scanned out in a

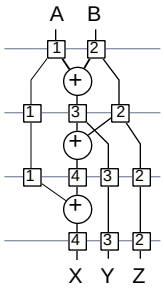

Figure 9. DFG_3

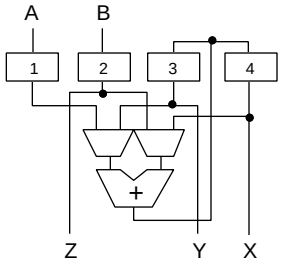

Figure 10. Data-path for DFG_3

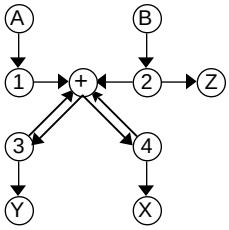

Figure 11. Connectivity graph for data path in Fig. 10 with no single orthogonal scan path configuration

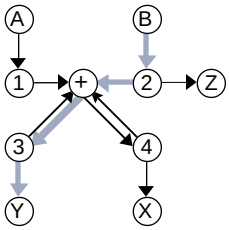

Figure 12. Configuration 1 for data path in Fig. 10

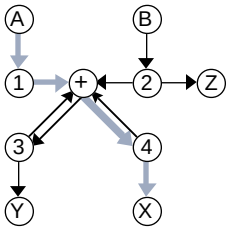

Figure 13. Configuration 2 for data path in Fig. 10

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 as part of the normal data path, then a subset of the registers can have load enables added so that multiple configurations may be used. In the previous example shown in Fig. 10, 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 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.

Different configurations can also be used during scan-in and scan-out, but it becomes harder to overlap the scanning-in and scanning-out of data and the test application time may increase.

2.3 Modifying Functional Units

When the orthogonal scan path configuration is determined, some functional units may need to be modified so that they can transfer the scan data. Multiplexers used during orthogonal scan require no modification, and paths between registers that are composed solely of multiplexers have very little overhead since no functional units must be modified. If a functional unit is used during one of the orthogonal scan configurations, then a logic gate must be added to each bit of the inputs that are not part of the scan path. This additional logic masks the input during scan. For example, the orthogonal scan path in Fig. 5 uses the adder during scan. Therefore the input that is not part of the scan path, in this case the input from register *1*, must have an AND gate added to each bit so that during scan the input is forced to zero. The modified adder is shown in Fig. 14. Other types of functional units are modified in similar fashions.

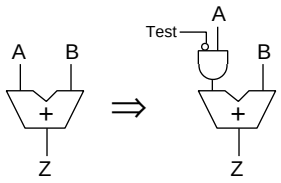

Figure 14. Adder modified for orthogonal scan

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 add extra interconnect which can increase the load, and consequently the delay, on the flip-flop outputs. The orthogonal scan path can be added so that a minimum amount of masking logic is added to the critical path, i.e., modify 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 multiplexer or functional input to reduce the area and delay overhead. These specially designed units would have an extra input, the test mode signal, added to select orthogonal scan mode. If multiple configurations are used, then multiple test mode signals are required.

2.4 Synthesizing Control

The modifications to the data path necessitate some changes to the control. The multiplexer address, register

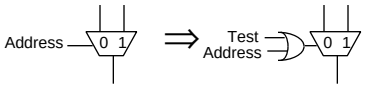

Figure 15. Multiplexer with modified address signal

enables and functional unit controls of components used in the orthogonal scan path may need to be modified to work correctly during orthogonal scan 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 signals make use of the global test mode signals and require at most one logic gate per control signal for each configuration, as shown in Fig. 15. The control signals may then be forced to appropriate values during the orthogonal scan operation.

If the control logic is being synthesized, then knowledge from the DFG may be used to allow the additional control logic to be reduced since some control signals may be shared. For example, the two multiplexers in the data path shown in Fig. 4 may both use the same control signal because of the nature of the data-flow graph.

If the DFG is not available for analysis, the modifications to the control signals must be made without taking advantage of any logic sharing and one logic gate must be added to each control signal. The data path in Fig. 4 would require two additional gates, one for each multiplexer select signal.

2.5 Creating Final Data Path

Once the data path has been synthesized, the orthogonal scan path determined and the functional units and control have been modified, the final data path with orthogonal scan is created. The resulting data path circuit for the DFG and data path of Figures 3 and 4 is shown in Fig. 16.

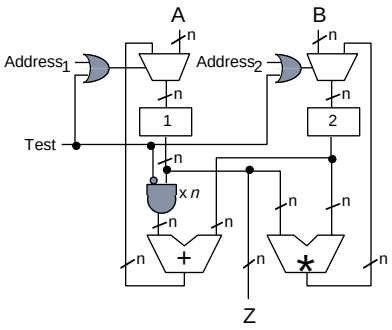

Figure 16. Data path from Fig. 4 modified for orthogonal scan

The additional logic is shaded. The two OR gates added to the multiplexer selects may be reduced to one OR gate if the DFG control information is analyzed. *n* AND gates are added to the adder, where *n* is the width of the data path.

3 Orthogonal Scan During Testing

Test application with the orthogonal scan path is the same as with a traditional scan path, except the test vectors are scanned-in and scanned-out in parallel as words as opposed to being scanned serially as bits. This parallelization of the test data reduces the total test time — the wider the data path, the greater the reduction.

The test vectors are generated in the same manner as for any other full scan design. ATPG is strictly combinational since all of the flip-flops are scanned. The orthogonal scan path itself can be tested prior to the actual circuit testing by shifting a pattern of zeros and ones through the scan path while in scan mode. This initial test verifies the correct shifting of vectors through the scan path and assures a valid test for the circuit. Once the scan path has been verified, it can be used during debugging to help diagnose problems by scanning out the state of the circuit — just as traditional scan paths can help with debugging and diagnosis.

The data paths discussed are assumed to have some primary inputs and outputs that are directly accessible so that test vectors may be applied and examined. If the data paths are embedded so that the inputs and outputs are not directly accessible, then some means of accessing them must be added.

This discussion of orthogonal scan does not cover the testing of the control. The control is assumed to be tested in some fashion that is complementary to orthogonal scan, e.g., using some form of traditional full scan [McCluskey 86] or beneficial scan [Norwood 96].

4 Register Allocation and Binding

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 (edges in the DFG that cross clock boundaries).

The register allocation and binding operations use a register conflict graph. Each node in the register conflict graph represents an edge from the DFG that crosses a clock cycle boundary. Edges, called *conflict edges* [Avra 91], between two nodes indicate that the variables associated with the two nodes cannot be bound to the same register. Register binding assigns a color to each node of the register conflict graph such that adjacent nodes have different colors. Nodes with the same colors can be bound to the same register. The minimum number of colors needed to color the register conflict graph indicates the number of registers allocated to the data path.

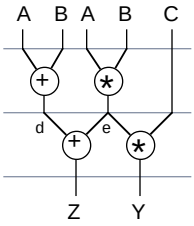

Figure 17. DFG_4 before register allocation

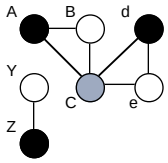

Figure 18. Register conflict graph for DFG_4

4.1 Standard Allocation and Binding

A simple method for allocation and binding creates one node in the register conflict graph for each variable in the DFG [Avra 91]. Conflict edges are added to the register conflict graph to indicate which variables cannot be assigned to the same register. These edges are added between any two nodes that represent variables that are both being used at the same clock cycle boundary. The resulting graph is colored and the variables bound to the corresponding registers.

Fig. 17 shows a DFG before the allocation and binding of registers. There are seven variables in the DFG — *A*, *B*, *C*, *d*, *e*, *Y* and *Z*. The corresponding register conflict graph is shown in Figure 18 with the nodes colored.

This register allocation and binding method provides a standard baseline with which to compare some alternate methods.

4.2 Allocation and Binding with Migration

The basic register conflict graph from Section 4.1 can be modified by creating additional nodes for delayed variables or for variables with multiple targets [Avra 91]. Delayed variables extend across more than one clock cycle boundary and have one node in the register conflict graph for each clock cycle in the variables lifetime. For example, variable *C* in Fig. 17 has two nodes added to the register conflict graph, shown in Fig. 19, one node, C_1, for the first clock cycle boundary and a second node, C_2, for the second clock cycle boundary. These two nodes are treated independently when determining conflict edges to be added. Originally, there were conflict edges between node *C* and nodes *A*, *B* , *d* and *e*, now there are edges between node C_1 and nodes *A* and *B* and between node C_2 and nodes *d* and *e*. This modification to the register conflict graph allows variables to migrate between registers over clock cycle boundaries.

Variables with multiple targets have one node in the register conflict graph for each operation that uses the variable. For example, variable *e* in Fig. 17 has two nodes, *e* and *e'* added to the register conflict graph, shown in Fig. 20, since the variable is used by the addition operation and the multiplication operation. Fig. 21 shows both of these modifications combined for DFG_4.

Both of these modifications create more paths between registers in the data path, making the determination of the orthogonal scan path much easier, but also possibly adding to the number and size of the multiplexers in the data path.

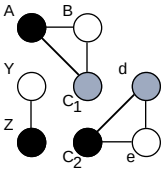

Figure 19. Register conflict graph with delayed variable

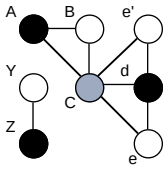

Figure 20. Register conflict graph with multiple targets

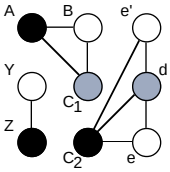

Figure 21. Register conflict graph with variable migration

4.3 Allocation and Binding for Self-Adjacency

The third register allocation and binding method attempts to make the determination of the orthogonal scan path easier without adding the potential overhead of variable migration.

The key observation is that self-adjacent registers are beneficial for orthogonal scan paths, even though they may not be advantageous for other techniques such as circular BIST. A self-adjacent register is a register that is both an input and an output of a functional unit. For example, register *1* in Fig. 23 is a self-adjacent register since it is

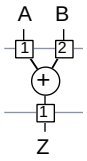

Figure 22. DFG_5

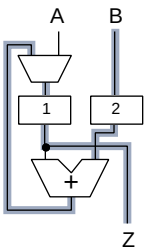

Figure 23. Data path for DFG_5

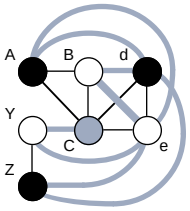

Figure 24. Register conflict graph with cost edges in gray

both the input to the adder and the output of the adder. The DFG shown in Fig. 22 corresponds to the data path in Fig. 23, and the fact that register *1* is both the input and the output of the addition operation indicates that register *1* is a self-adjacent register. Register *2* is not self-adjacent since it is used only as the input to the adder.

Self-adjacent registers reduce the number of distinct registers associated with a specific functional unit. For example, for a functional unit with two inputs and one output, a self-adjacent register results in a functional unit having only two distinct registers as inputs and outputs, as opposed to having three distinct registers if none of them are self-adjacent. With only two registers, an orthogonal scan path, $B \Rightarrow 2 \overset{+}{\Rightarrow} 1 \Rightarrow Z$, through the functional unit can access both registers, as shown by the highlighted path in Fig. 23. With three registers, only two can be included in that portion of the orthogonal scan path and the other register must be included in some other manner — possibly requiring another configuration. Maximizing the number of self-adjacent registers in a data path minimizes the number of distinct registers associated with each functional unit.

A register allocation and binding technique discussed in [Avra 91] can be used to add cost edges to the register conflict graph. *Cost edges* are weighted edges that are added between certain compatible nodes (nodes that do not already have a conflict edge between them) in the register conflict graph, and are used to guide the graph coloring algorithm. A cost edge has a positive edge if it is not advantageous to assign the same color to the adjacent nodes and a negative edge if it is advantageous.

Cost edges with negative weights can be added between nodes to indicate that the variables associated with those nodes should be bound to the same register, if possible, in order to make the register self-adjacent. For example, again using DFG_4 shown in Fig. 17, node *A* in the register conflict graph would have a negative-weight cost edge added between node *d* because if nodes *A* and *d* are colored with the same color, the register bound to both those variables will be a self-adjacent register. The register conflict graph in Fig. 24 shows all conflict edges (black edges) and cost edges (gray edges) for DFG_4.

5 Implementation and Results

The Stanford CRC synthesis-for-test tool, TOPS, has been modified to add orthogonal scan paths to data paths. TOPS has been used to add orthogonal scan paths to five benchmark circuit examples — three from the HLSW 92 benchmark circuits (*diffeq, ellipf* and *gcd*), one from the HLSW 95 benchmark circuits (*fft*) and a circuit described in [Tseng 86] (*tseng*). Table 1 shows the data path characteristics of these circuits.

Table 1. Benchmark circuit characteristics

Circuit	Data Path Width	# Registers	# Functional Units
diffeq	32-bits	7	2 multipliers 1 adder 1 subtractor 1 comparator
ellipf	16-bits	11	4 adders
fft	32-bits	13	4 multipliers 2 adders 2 subtractors 1 divider 1 comparator 4 1024x32 RAMs
gcd	8-bits	2	1 adder/subtractor 1 comparator
tseng	32-bits	5	1 multiplier 3 adders 1 subtractor 1 AND 1 OR

The five benchmark circuits were each synthesized using the three register allocation and binding techniques described in Section 4, and then orthogonal scan paths were added to the circuits. Table 2 summarizes the results. The circuit size before the addition of the orthogonal scan path, the area overhead due to modifying the control signals and the functional units, and the total size of the circuit with the orthogonal scan path are given. The sizes do not include the control logic, other than the modifications due to the orthogonal scan path, nor do they include the routing.

All of the circuits synthesized with the standard register allocation and binding technique can have orthogonal scan paths inserted, but two of them (*diffeq* and *ellipf*) require two configurations. Since all registers in TOPS are synthesized with load enables, the additional configurations do not add a lot of overhead, but two scan mode signals are needed The other three circuits have only one configuration.

When the circuits are synthesized allowing variable migration, the circuits with orthogonal scan paths are larger than the corresponding circuits synthesized with the two other register allocation and binding techniques. However, they also require the smallest overhead to make the original, non-scanned, circuit orthogonal scannable. Both of these factors are due to the same phenomenon — the increase in the number of multiplexer inputs. By allowing the variables to migrate between registers, additional connections are made between the registers,

Table 2. Orthogonal scan overhead for circuits with various register allocation and binding algorithms

Circuit	Register Binding Algorithm	Circuit Size (1000 λ^2)	Control Overhead (1000 λ^2)	Func Unit Overhead (1000 λ^2)	Total Overhead (1000 λ^2)	Total Size (1000 λ^2)	Number Test Pins
diffeq	standard	13385	121	23	144	13529	2
	migration	14452	40	26	66	14518	1
	self-adjacent	13385	121	14	135	13520	1
ellipf	standard	2611	60	47	107	2718	2
	migration	4094	20	55	75	4169	1
	self-adjacent	2500	81	29	110	2610	1
fft	standard	129218	49	282	331	129549	1
	migration	131913	57	0	57	131970	1
	self-adjacent	129054	48	282	330	129384	1
gcd	standard	359	11	11	22	381	1
	migration	366	0	10	10	376	1
	self-adjacent	327	11	8	19	346	1
tseng	standard	8234	121	10	131	8365	1
	migration	8628	40	18	58	8686	1
	self-adjacent	8234	121	9	130	8364	1

connections that do not go through any functional units. This increased connectivity results in very low overhead orthogonal scan paths, but it also results in large multiplexers that add a lot of area to the original, non-scanned circuit.

The circuits synthesized to maximize the number of self-adjacent registers have the smallest total size for all five circuits. The original circuits are the same size, or slightly smaller, as the circuits synthesized with the standard register allocation and binding technique, but the overhead needed to add the orthogonal scan path is much less, resulting in a smaller overall size for the scannable circuit. By guiding the register allocation and binding to maximize the number of self-adjacent registers a much better orthogonal scan path implementation is possible. The rest of this paper uses these circuits, synthesized to maximize the number of self-adjacent registers, to compare orthogonal scan paths to traditional scan paths.

Table 3. Circuit sizes in 1000 λ^2

Circuit	No Scan	Trad Scan	% Ovhd	Orth Scan	% Ovhd
diffeq	13385	13668	2.1	13520	1.0
ellipf	2500	2722	8.9	2610	4.4
fft	129054	129580	0.4	129384	0.3
gcd	327	352	7.6	346	5.8
tseng	8234	8437	2.5	8364	1.6

Table 3 compares the sizes of 1) the circuit without scan, 2) the circuit with a traditional scan path and 3) the circuit with an orthogonal scan path. Again, the sizes do not include the control logic or interconnect. For the technology used, the size of an AND gate is the same as the difference in size of a scannable register and a non-scannable register. In other words, $AREA_{\text{AND gate}} = AREA_{\text{scan flip-flop}} - AREA_{\text{flip-flop}}$. The circuits with the orthogonal scan paths are smaller than the circuits with the traditional scan paths, with roughly half the scan overhead.

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

As the results in Table 4 show, the test application time for orthogonal scan paths is significantly reduced from that of traditional scan paths. Orthogonal scan paths are shorter because they make use of the buses in the data path. For an *n*-bit data path, there are *n* duplicate scan paths that differ only in bit position — they use exactly the same registers and functional units in exactly the same fashion. The orthogonal scan path is at least *n* times shorter than the traditional scan path; possibly even shorter if multiple orthogonal scan paths are used. Shortening the scan path length increases the number of pins that are used for scanning data in and out, but the total number of pins

Table 4. Scan characteristics of benchmark circuits

Circuit	Scan	# Test Pins	Scan Overhead	Scan Shifts
diffeq	traditional	1 scan mode 2 scan in/out	225 multiplexers	225
	orthogonal	1 scan mode	96 gates functional 11 gates control	4
ellipf	traditional	1 scan mode 2 scan in/out	176 multiplexers	176
	orthogonal	1 scan mode	64 gates functional 23 gates control	4
fft	traditional	1 scan mode 2 scan in/out	417 multiplexers	417
	orthogonal	1 scan mode	224 gates functional 38 gates control	13
gcd	traditional	1 scan mode 2 scan in/out	20 multiplexers	20
	orthogonal	1 scan mode	8 gates functional 6 gates control	2
tseng	traditional	1 scan mode 2 scan in/out	161 multiplexers	161
	orthogonal	1 scan mode	96 gates functional 7 gates control	3

does not increase since the orthogonal scan paths use existing primary inputs and outputs to scan the data in and out. Consequently, orthogonal scan paths do not increase the number of channels necessary on the tester. However, tester interfaces with multiple scan capable channels are necessary, but the memory requirements for each channel are significantly reduced.

6 Summary

Orthogonal scan paths follow the path of the data flow and are orthogonal to the flow of normal scan paths. Orthogonal scan paths result in scanned data paths that have much less overhead than traditional scan paths. Less logic must be added to get the scan functionality, the number of additional test pins can be reduced, and little, or no, extra interconnect, along with the associated load, is added. The test application time is also drastically reduced due to the short lengths of the orthogonal scan paths, and orthogonal scan paths do not have the hold time problems that traditional scan paths often have.

Knowledge of orthogonal scan paths can be used during the data path synthesis to improve the final orthogonal scan path. Modifications to the register allocation and binding to maximize the number of self-adjacent registers in the final data path can reduce the overhead of the orthogonal scan path that is inserted into the data path.

Acknowledgments

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

References

[Abadir 85] Abadir, M.S., et. al., “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., et. al., *Digital Systems Testing and Testable Design*, Computer Science Press, New York, NY, 1990.

[Adham 95] Adham, S., 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., et. al., “Symbolic Test Generation for Hierarchically Modeled Digital Systems,” *Proc. Intl. Test Conf.*, Washington, DC, pp. 461-469, Aug. 29-31, 1989.

[Avra 91] Avra, L., “Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths,” *Proc.*

Intl. Test Conf., Nashville, TN, pp. 463-472, Oct. 26-30, 1991.

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

[Bhatia 94] Bhatia, S., et. al., "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., et. al., "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., et. al., "Addressing Design for Testability at the Architectural Level," *IEEE Trans. Computer-Aided Design*, Vol. 13, No. 7, pp. 920-934, July 1994.

[Eichelberger 77] Eichelberger, E.B., et. al., "A Logic Design Structure for LSI Testability," *14th Design Automation Conf.*, New Orleans, LA, pp. 462-467, June 1977.

[LSI Logic 92] LSI Logic Corporation, *Chip-Level Full Scan Design Methodology Guide*, Milpitas, CA, 1992.

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

[McFarland 90] McFarland, M.C., et. al., "The High-Level Synthesis of Digital Systems," *Proc. IEEE*, Vol. 78, No. 2, pp. 301-318, Feb. 1990.

[Norwood 96] Norwood, R., et. al., "Synthesis-for-Scan and Scan Chain Ordering," *Proc. IEEE VLSI Test Symp.*, Princeton, NJ, pp. 87-92, April 28-May 1, 1996.

[Tseng 86] Tseng, C.-J., et. al., "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., et. al., "Design for Testability-A Survey," *Proc. IEEE*, Vol. 71, No. 1, pp. 98-112, Jan. 1983.