Scheduling and Module Assignment for Reducing BIST Resources*

Ishwar Parulkar, Sandeep K. Gupta and Melvin A. Breuer
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089-2562.

Abstract

Built-in self-test (BIST) techniques modify functional hardware to give a data path the capability to test itself. The modification of data path registers into registers (BIST resources) that can generate pseudo-random test patterns and/or compress test responses, incurs an area overhead penalty. We show how scheduling and module assignment in high-level synthesis affect BIST resource requirements of a data path. A scheduling and module assignment procedure is presented that produces schedules which, when used to synthesize data paths, result in a significant reduction in BIST area overhead and hence total area.

1 Introduction

Built-in self-test (BIST) techniques involve the modification of the hardware on the chip such that the chip has the capability of testing itself. How to reduce the BIST area overhead without sacrificing the quality of the test is an important research problem for test engineers. A typical data path consists of registers and functional modules, such as adders and multipliers that are selected from a pre-designed library, and an interconnection network of multiplexers. One cost-effective BIST technique for such data paths, called partial intrusion BIST, involves modifying existing functional registers to generate and supply test patterns and to collect and compress test responses on-chip [1].

Previous research has focused on incorporating BIST resource considerations only during the assignment phases of high-level synthesis [2],[3],[4],[5]. The objectives of synthesis algorithms in all these approaches were variations on the common theme of maximizing sharing of registers as BIST resources and minimizing number of expensive BIST resources required

for a data path. All approaches assumed that a scheduled data flow graph was available. Harris et al. presented a scheduling technique that improved test concurrency and hence the test *time* required to test a data path using partial intrusion BIST [6]. The effect on the number of BIST resources was not considered in this work. In this paper, we present a scheduling and module assignment technique that produces schedules, which when used to synthesize data paths, result in smaller number of BIST resources and low BIST area overhead. The effect of scheduling of operations on the module assignment and the sharing capability of BIST resources between various modules is studied. In [7], a theory for lower bounds on BIST resources for scheduled data flow graphs is presented. The theory has been extended to derive testability metrics that are used as costs to guide the proposed scheduling and module assignment algorithm. Experimental results indicate that schedules generated using our technique lead to data paths with better BIST solutions as compared to data paths synthesized using other schedules. When the proposed scheduling technique is followed by register assignment that can take advantage of sharing potential of BIST resources, further savings in BIST area are obtained.

2 Motivation
2.1 Minimal Intrusion BIST

BIST area overhead concerns can be addressed after synthesis by using minimal intrusion BIST. In minimal intrusion BIST, only a subset of the functional registers are used in the test mode. The registers are selected such that all functional modules are tested and the area overhead for modification of the registers is minimum. The rest of the data path is tested using functional tests. The combination of testing modules with pseudo-random patterns and the rest of the data path with functional patterns ensures a high fault coverage testing scheme at very low cost. Note that in this minimal intrusion BIST methodology the test resources and paths used to generate, transport and collect test

data are a subset of the functional data path. No additional data path components are added for the purpose of testing. In the test mode, some of the registers in the data path are reconfigured to support test pattern generation, some to support signature analysis, and some to perform both of these test functions. Depending on the test function required, four different *types* of BIST resources can be designed: 1) test pattern generation capability only (TPG), 2) test response compression or signature analysis capability only (SA), 3) test pattern generation and response compression capability at different times (BILBO), and 4) simultaneous test pattern generation and response compression capability (CBILBO).

Consider a typical data path as shown in Fig. 1(a). Any register that is connected to an input port of a module through only multiplexers is a candidate for supplying test patterns to that input port. Similarly any register that collects data from the output port of a module through only multiplexers is a candidate to compress test responses. Fig. 1(b) shows such candidate registers for the adder. One choice of BIST resources for the adder is denoted by highlighted registers and paths. The type of BIST register is determined by the function the register performs.

Generally, there are many choices of BIST resources for testing a module. Fig. 1(c) shows an alternate selection of BIST resources for the adder. This choice is better than the one in Fig. 1(b), because R_3 and R_6 can be shared as test resources with the multiplier resulting in fewer BIST resources for the whole data path, as shown in Fig. 1(d). Note that the multiplier and adder have to be tested in different sessions since their test paths to R_6 lie along the same multiplexer. For minimizing BIST area overhead, a data path is analyzed globally to determine BIST registers from the various choices for each module such that all modules are tested with a minimum total BIST register overhead [9]. We have formulated a 0-1 ILP that models the problem of finding a minimal intrusion BIST solution [10].

Once a data path is synthesized there is limited freedom in sharing test resources. From Fig. 1(d) it can be seen that R_1 and R_5 are *dedicated* BIST resources for the multiplier and adder, respectively, and cannot be shared. Traditional synthesis techniques synthesize data paths without any consideration of how input registers or output registers of modules can be shared between modules. If the same behavior is synthesized taking into account BIST resource considerations, the synthesized data path may require fewer such resources.
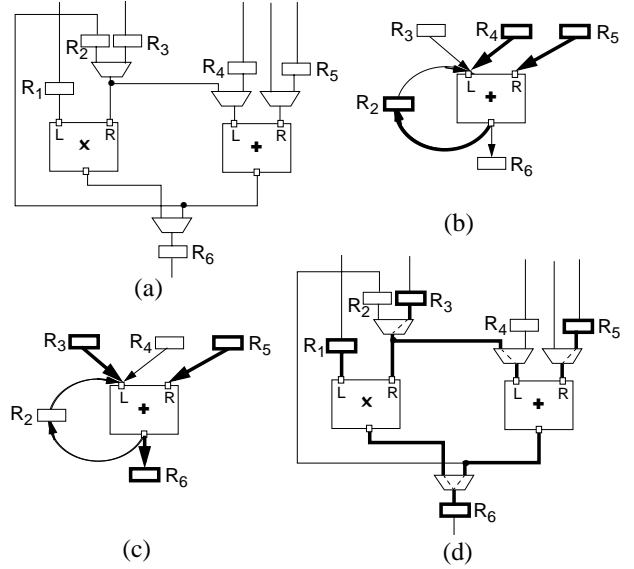

(a)   (b)   (c)   (d)

Figure 1: Minimal intrusion BIST

2.2 Scheduling and BIST Resources

Assignment techniques that synthesize data paths with the objective of minimizing BIST registers required for a BIST solution have been developed [3],[4],[5]. Here a scheduled data flow graph and a module assignment is assumed. The schedule and the module assignment that depends on the schedule have a significant effect on BIST resources of a data path. In behaviors where there is limited freedom for optimization of BIST resources in the register assignment phase, it is beneficial to consider more alternatives during the scheduling and module assignment phases.

For a scheduled data flow graph (DFG), two variables can be assigned to the same register only if their lifetimes do not overlap. Consider the schedule and module assignment shown in Fig. 2(a). Registers to which variables b, c, a and f are assigned are candidates for test pattern generators for module A_1 and registers to which variables d and e are assigned are candidates for test pattern generation for A_2. Because of the lifetimes of these variables, variables d and e have overlapping lifetimes with all the input variables of A_1. Hence A_1 and A_2 will require separate test pattern generators, irrespective of the register assignment chosen. Similarly variable g, the output variable of A_2 overlaps with both the output variables of A_1 which results in separate registers for test response compression of A_1 and A_2. Note that the data path to the right shows *one* possible partial register assignment that corresponds to the *best* BIST solution. Now, if operation $+_3$ is scheduled in control step 2 instead of control step 3, the lifetimes of the variables change. In

this case shown in Fig. 2(b), variable f which is an output variable of A_1 does not overlap with variable g which is an output variable of A_2 and they can be assigned to the same register as shown in the data path to the right. This register assignment enables the sharing of one register for compressing test responses of A_1 and A_2 resulting in savings in BIST resources. Note that the sharing of test pattern generators cannot be improved in this case. In Fig. 2(c), $+_3$ is scheduled in control step 2 as in Fig. 2(b). However operation $+_2$ is scheduled in control step 1 instead of 2 so that a different module assignment is possible. For the schedule in Fig. 2(c), $+_2$ and $+_3$ are assigned to A_2. Now, output variables of A_1 and A_2, namely f and h, can be assigned to the same register. Also, an input variable of A_1, namely c, can be assigned to the same register as input variable f of A_2. Thus a register assignment as shown in the data path to the right can be achieved, which results in only 3 registers for test pattern generation and 1 register for test response compression. Fig. 2(b) and (c) demonstrate how scheduling and module assignment can affect BIST resources. Choosing an appropriate schedule and module assignment creates possibilities for register assignment that can be exploited by an algorithm such as [5] to further minimize BIST area overhead.

3   Scheduling and Module Assignment Algorithm

The behavioral description to be synthesized is assumed to be given in the form of a data flow graph $G = (V, E)$, where V is the set of operations $\{o_1, o_2, ..., o_{|V|}\}$ and E is the set of variables (operands and results of the operations). A schedule of G is a mapping $S : V \to \{1,2,...,L\}$ where for operation $o_i \in V$, $S(o_i)$ corresponds to the control step in which o_i is scheduled. A valid schedule is one in which $(o_i, o_j) \in E \Rightarrow S(o_i) < S(o_j)$. L is called the latency of the schedule. The as-soon-as-possible value $ASAP_i$ of an operation o_i is the earliest control step in which o_i can be scheduled and the as-late-as-possible value $ALAP_i$ of an operation o_i is the latest control step in which o_i can be scheduled. The *mobility* $\mathcal{M}(o_i)$ of an operation o_i is the interval $[ASAP_i, ALAP_i]$, and the *slack* of an operation o_i is $slack(o_i) = |\mathcal{M}(o_i)| - 1$.

A module assignment of a scheduled DFG is defined as $\Pi_M : V \to M$ where M is a set of modules $\{M_1, M_2, ..., M_m\}$. Associated with each module is a *type* of operation it can perform, such as addition or multiplication. Two operations can be assigned to the same module only if they are scheduled in different control steps and if they are of the same type.

In the previous section, we have shown how schedul-

(a) **SCHEDULE 1, MODULE BINDING 1**

(b) **SCHEDULE 2, MODULE BINDING 1**
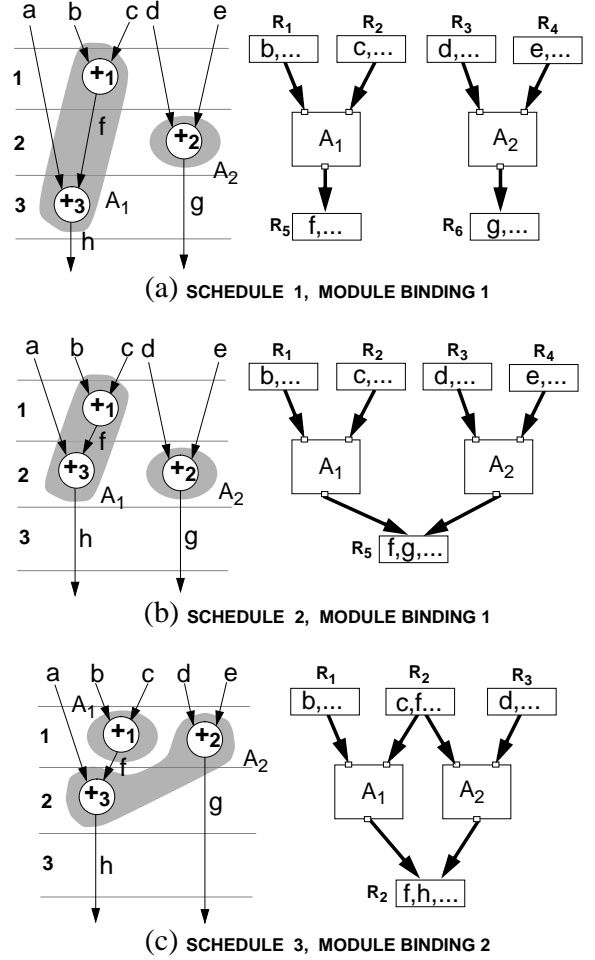

(c) **SCHEDULE 3, MODULE BINDING 2**

Figure 2: Effect of scheduling and module assignment on BIST resources

ing and module assignment can affect the number of BIST resources for a data path. In general, different schedules and module assignments that have a desirable latency and functional area can differ significantly in their BIST resource requirement. We propose a scheduling and module assignment approach that constructs schedules and assign operations to modules such that the desirable latency and functional resource constraints are met and the BIST area overhead is low. It has been shown that scheduling and module binding simultaneously results in a more efficient exploration of the design space [11]. Our approach has two phases: 1) adding temporal testability constraints between selected operations, and 2) performing scheduling and module assignment of each operation in the DFG with the modified constraints. In Phase 1, pairs of operations o_i and o_j are selected such that it is beneficial (in terms of BIST area overhead) for the operations to execute in different control steps and be assigned

to different modules. The scheduling of such operations is constrained to occur in different control steps by adding a temporal constraint (an edge in the DFG) between the operations. Phase 1 can be viewed as *coarse* scheduling, where only concurrency and sequentiality of operations is influenced but the exact control step is not assigned. In Phase 2, *detailed* scheduling is performed where each operation is assigned to a control step and to a module taking into account the constraints added in Phase 1. Scheduling is followed by register and interconnect assignment to synthesize the final data path. After synthesis, BIST resources corresponding to a minimum area overhead are selected.

3.1 Phase 1: Adding Temporal Constraints (Coarse Scheduling)

A schedule induces a temporal sequence on operations. Operations that are executed sequentially can share functional modules. Sequentiality of operations is thus beneficial to minimizing functional resources and desirable when the objective is to minimize functional area.

Definition 1 *The* **storage concurrency** *of a set of variables* Q, $SC(Q)$, *is the maximum number of variables in* Q *alive at the same time.*

The input and output variables of a pair of sequential operations have a high possibility of non-overlapping lifetimes (low SC) and hence a high possibility of being assigned to the same register. Hence if sequential operations are *distributed* across modules (as opposed to sharing modules) then BIST resources and area overhead can be minimized since there is a better possibility of synthesizing common BIST resources. We introduce the concepts of *strictly sequential*, *strictly concurrent* and *weakly sequential* (or *weakly concurrent*) pairs of operations for an unscheduled DFG. Two operations are called *strictly sequential* if they have to be scheduled in different control steps for all valid schedules with minimum latency (L_{min}), and they are called *strictly concurrent* if they have to be scheduled in the same control step for all valid minimum latency schedules. *Weak sequentiality*, on the other hand, implies that it is possible but *not necessary* to schedule the two operations in the same control step for a minimum latency schedule. In the DFG shown in Fig. 3(a), operations $*_1$ and $*_3$ are strictly sequential, operations $*_1$ and $*_2$ are strictly concurrent, and operations $*_5$ and $+_1$ are weakly sequential.

The temporal (order of execution) and spatial (assignment to modules) relationships of a pair of operations determines the nature of the data path as shown in the Table 1. Two operations executing in the same

Table 1: Temporal-spatial relation of operations

Modules	Control steps	
	Same	Different
Same	-	Does not affect BIST resources
Different	I/O variables have high SC (cannot share BIST resources)	I/O variables have low SC (**can share BIST resources**)

control step cannot be assigned to the same module. The other three cases are functionally possible. If two operations are strictly concurrent, they fall in the first column (same control step) and BIST resource reduction is not possible. Strict sequentiality falls in the second column (different control steps). If minimum latency is desired, the flexibility in scheduling cannot be utilized to change the relationship between strictly concurrent or strictly sequential pairs of operations. Strict sequentiality, however, can be leveraged to reduce test resources by deciding an appropriate control step and assigning the operations to different modules. This *fine* (i.e. detailed) scheduling is addressed in Phase 2. The case of interest in Phase 1 is weak sequentiality which spans the whole table. Weak sequentiality can be exploited and the flexibility in scheduling can be used to add temporal constraints so that the operations fall in the second column. Phase 2 can then take advantage of the constraints to push the operations in the direction of the lower right hand box in Table 1.

A temporal constraint is a special edge added between two operations to ensure that they are scheduled in different control steps. In Phase 1 of our approach, pairs of *weakly sequential* operations are identified and testability temporal constraints are added to *selected* pairs. The minimum latency of the DFG is not affected by adding temporal constraints between weakly sequential operations. The number of registers required for the DFG is also not affected because it has been shown to be insensitive to different schedules [12]. But two other effects need to be considered: 1) effect on the mobility of other operations, and 2) effect on the number of modules required. For a weakly sequential pair of operations, the effect of adding a temporal constraint on the mobility of other operations is quantified as the total change in *slack* of operations, Δ_{slack}. All weakly sequential operation pairs are considered in an increasing order of the value of Δ_{slack}. If a temporal constraint between a pair of operations violates the constraint on the number of modules, the pair is
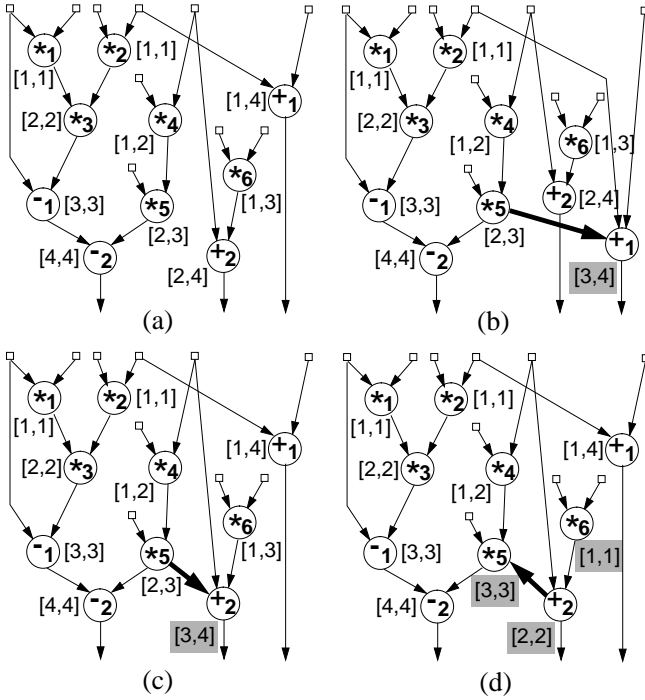
Figure 3: Phase 1 - Adding temporal constraints

3.2 Phase 2: Detailed Scheduling and Module Assignment

The proposed scheduling procedure of Phase 2 is based on classical *list scheduling* techniques [11]. List scheduling techniques are widely used in high-level synthesis because of low computation complexity and near-optimum solutions. We use the *slack* of an operation as the priority function in picking an operation to be scheduled since we desire minimum latency schedules. In addition to assigning operations one by one to a control step, the procedure simultaneously assigns them to the available modules. The types of modules and the number of instances of each type can be computed before scheduling and are known to the procedure.

INPUT: DFG, L_{min} and $M = \{M_1, M_2, ..., M_m\}$

Step 1: Pick an unscheduled o_i with least *slack*

Step 2: Find all possible tuples $T_j = <CStep, Mod>$
      s.t. o_i can be assigned control step $CStep$
      and module Mod

Step 3: **For each** tuple T_j s.t. Mod already
      has an operation assigned to it
      Step 3.1: Calculate $Cost(T_j)$

Step 4: **If** there exists T_j s.t. $Cost(T_j)$ is positive
      Step 4.1: **then** select T_j with highest cost
      Step 4.2: **else** select T_j s.t. Mod has
              no operation assigned to it

Step 5: Assign o_i to selected T_j

Step 6: Go to Step 1

Procedure for Phase 2 - *Schedule_and_Assign()*

The procedure is iterative and at every step from the operations that have not yet been scheduled and assigned, an operation o_i with the smallest *slack* is selected. The set of all control step and module assignment tuples $(T_j = <CStep, Mod>)$ is then determined such that o_i can be scheduled in control step $CStep$ and assigned to module Mod from the set of available modules $M = \{M_1, M_2, ..., M_m\}$. Note that while considering such tuples for o_i, other operations have already been scheduled and assigned to modules. A cost function, $Cost(T_j)$, is computed for each tuple T_j to determine the control step and the module to which the operation should be assigned. $Cost(T_j)$ has two components. The primary component of the cost is Δ_{BIST}, the *decrease* in BIST area overhead corresponding to the assignment as defined by the tuple. A decrease in the number of BIST

dropped from consideration. Of the remaining candidates, a pair is selected such that it is most beneficial for reducing BIST resources. Note from Table 1 that operations in the lower right hand box of the table are most beneficial in this regard. Hence, candidate pairs that have a high probability of being assigned to different modules are selected [10]. A temporal constraint is added to a pair of operations selected in this manner. The ALAP and ASAP values of all operations are updated after addition of a temporal constraint.

Fig. 3 demonstrates addition of temporal constraints to the unscheduled DFG of the *diffeq* benchmark [14]. The $[ASAP_i, ALAP_i]$ values of each operation o_i are indicated in Fig. 3(a). Consider the pair or weakly sequential operations $(*_5, +_1)$. A temporal constraint between these operations and the new mobilities are shown highlighted in Fig. 3(b). The total change in *slack* of operations, Δ_{slack}, is 2. If a temporal constraint is added to the operation pair $(*_5, +_2)$, as shown in Fig. 3(c), $\Delta_{slack} = 1$ and hence this constraint is preferred. Note that a temporal constraint has a preferred *precedence* relationship. The temporal constraint in Fig. 3(c) with the *opposite* precedence relationship is shown in Fig. 3(d). In this case, $\Delta_{slack} = 5$. This constrains the schedule severely and hence the precedence order in Fig. 3(c) is preferred.

resources can adversely affect the multiplexing complexity [10]. Hence another component of $Cost(T_j)$ is Δ_{MUX}, the *increase* in multiplexer area corresponding to the assignment defined by T_j. A tuple T_j is chosen such that the decrease in BIST area overhead after compensating for an increase in multiplexer area $(Cost(T_j) = \Delta_{BIST} - \Delta_{MUX})$, is maximum. A description of Δ_{BIST} is given next.

4   Estimating BIST Cost

Testability metrics based on the theory for lower bounds on BIST resources [7] have been developed in this work to guide the scheduling and module assignment process. The metrics correspond to an estimate of the best optimum BIST area overhead that can be achieved.

Definition 2 *Given a scheduled DFG and a set of modules $\{M_1, M_2,...,M_m\}$ to which the scheduled operations have been assigned, a **maximal concurrent operation set** $V^i_{C_{max}}$ is a set of m operations, each of which is assigned to a different module.*

Let $IVar^i$ be the set of input variables of all operations in $V^i_{C_{max}}$ and $OVar^i$ be the set of output variables of all operations in $V^i_{C_{max}}$. Since $V^i_{C_{max}}$ contains *exactly* one operation from each module, the registers to which variables of $IVar^i$ are assigned correspond to a BIST solution, where these registers generate test patterns for all modules in the data path. Similarly, the registers to which the variables of $OVar^j$ have been assigned correspond to one BIST solution where these registers compress test responses of all modules. The actual *type* of the BIST register is determined by how the variables in $IVar^i \cup OVar^j$ are distributed across the BIST registers. It has been shown that the lower bound on the number of registers required to generate test patterns for all modules is given by $min_i SC(IVar^i)$ and the lower bound on the number of required to compress test responses is given by $min_i SC(OVar^i)$, where the minimum is over all maximal concurrent operation sets [7]. (Recall from Definition 1 that SC is the minimum number of registers required for a set of variables.) Let $IVar^i \cup OVar^j$ be denoted by $TestVar^{i,j}$, the set of *test variables*. For a given schedule and module assignment, each $TestVar^{i,j} (1 \leq i,j \leq l$, where l is the number of maximal concurrent operation sets) corresponds to a minimal intrusion BIST solution.

Estimation of BIST cost in Phase 2 is described next using the unscheduled DFG in Fig. 4(a). The DFG requires 4 modules to implement it - 2 multipliers M_1 and M_2, one subtractor M_3 and one adder M_4. Operations $*_1, *_2, *_3, -_1$ and $-_2$ have a slack of 0 for a minimum latency schedule. Hence they get scheduled in control steps 1, 1, 2, 3, and 4, respectively. Operations $*_1$ and $*_3$ are assigned to multiplier M_1 and $*_2$ to multiplier M_2. The unscheduled operation $*_4$ has a slack of 1 and is considered next. Assigning it to control step 1 would require an additional multiplier, hence it is assigned to control step 2 and module M_2 (since $*_3$ is already assigned to M_1). The next unscheduled operation is $*_5$ and that case is shown in Fig. 4(b). The possible tuples for $*_5$ are $T_1 = <3, M_1>$ and $T_2 = <3, M_2>$. For T_1, $min_i SC(IVar^i) = 3$, $min_i SC(OVar^i) = 1$ and $min_{i,j} SC(TestVar^{i,j}) = 4$. For tuple T_2, $min_i SC(IVar^i) = 2$, $min_i SC(OVar^i) = 1$ and $min_{i,j} SC(TestVar^{i,j}) = 3$. Tuple T_2 is preferable since it corresponds to a data path with a smaller requirement of BIST resources. Hence $*_5$ assigned to control step 3 and module M_2. Fig. 4(c) shows the next scheduling step, that of scheduling operation $+_1$. The operation can be assigned only to the adder module M_4. However, there is a choice in terms of the control step in which it can be scheduled. The possible tuples are $T_1 = <1, M_4>$, $T_2 = <2, M_4>$, $T_3 = <3, M_4>$ and $T_4 = <4, M_4>$. Depending upon the tuple chosen the lifetimes of the input and output variables of $+_1$ change which in turn affects the BIST resources. For T_1, $min_i SC(IVar^i) = 2$ and $min_i SC(OVar^i) = 2$, and for the rest of the tuples $min_i SC(IVar^i) = 4$ and $min_i SC(OVar^i) = 4$. Hence tuple T_1 is preferred and $+_1$ is scheduled in control step 1 and assigned to M_4.

The actual cost function used in scheduling to select a tuple is a more accurate estimation of BIST area overhead. The exact assignment of the variables is $TestVar^{i,j}$ to registers determines the number and type of BIST resources and hence the cost of the BIST solution. Assuming complete flexibility in register assignment, a lower bound $lb_{BIST}(i,j)$, corresponding to $TestVar^{i,j}$ can be found. The minimum of the lower bounds over all $TestVar^{i,j}$ gives a lower bound on the optimum BIST solution of the final data path. Using area overheads of BIST registers from the component library, an accurate metric is used in the scheduling algorithm [8]. The cost $Cost(T_j)$ in procedure *Schedule_and_Assign()*, corresponds to a *decrease* in the BIST area overhead metric and hence a move with a high cost is preferred. A detailed description of the metric is beyond the scope of this paper and is presented in [10].

5   Experimental Results

The proposed scheduling and module assignment procedure has been integrated into the Stanford CRC synthesis-for-test tool, TOPS [13]. To demonstrate the use of the proposed scheduling technique in synthesiz-
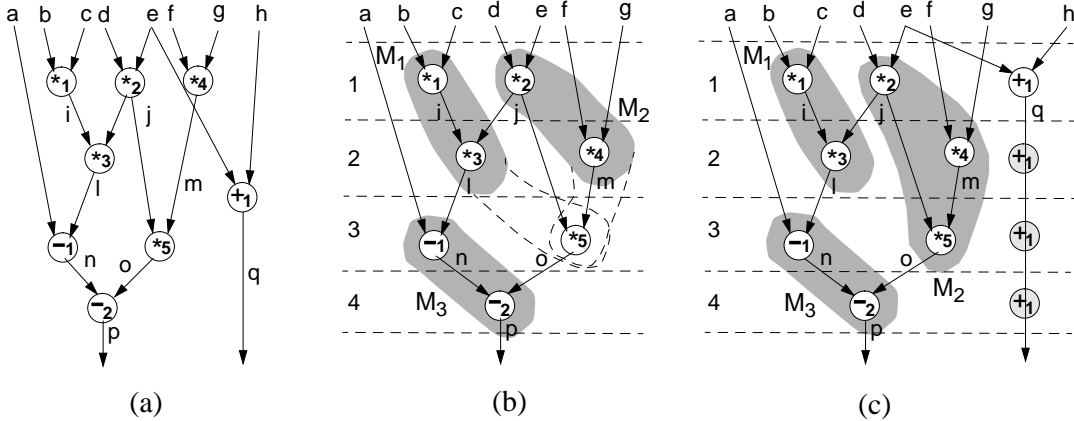
Figure 4: Phase 2 - Detailed scheduling and module assignment

ing data paths with low BIST area overhead, experiments were conducted on the following benchmarks: 1) the 2nd order differential equation - *DIF*, 2) the auto regression filter element - *ARF*, 3) an 8-point FIR filter - *FIR*, and 4) the elliptic wave filter - *EWF* [14]. Different synthesis flows were considered using combinations of two scheduling techniques and two register assignment techniques shown in Table 2 - Flow I (SWT-AWT), Flow II (SFT-AWT), and Flow III (SFT-AFT).

Table 3 shows the characteristics of all the synthesized data paths. The module requirements used in Flow II and Flow III that use the proposed SFT approach were derived from the requirements of Flow I. It can be seen from the results that the latency, number of functional modules and registers in all the three synthesis flows are preserved for all the benchmarks (the exception is *EWF* in which case the number of registers decreases in Flows II and III). However, there is an increase in the multiplexer complexity as BIST considerations are incorporated into the various stages of synthesis.

The data paths synthesized by each synthesis flow were made self-testable using minimal intrusion BIST. The last five columns of Table 3 compare the areas of synthesized data paths, both, before and after making them self-testable. Components designed using a macro-cell library supplied by LSI Logic Corp. were used for synthesis and the area is given in cell units [8]. It can be observed that for all four benchmarks, the BIST area overhead in the case of Flow II is less than that of Flow I and the BIST area overhead for Flow III is less than that for Flow II. This decrease in BIST area overhead is accompanied by an increase in the area of the original data paths (before BIST). The increase in the non-BIST version of data paths is due to the increase in the number of multiplexers. Even in the case

Table 2: Synthesis algorithms used in experiments

TASK	TYPE	DESCRIPTION
Scheduling	**SWT**	Traditional scheduling **W**ithout **T**estability (such as ASAP)
	SFT	Scheduling **F**or **T**estability (proposed in this paper)
Assignment	**AWT**	Traditional register and/or module assignment **W**ithout **T**estability
	AFT	Register assignment **F**or **T**estability (such as in [7])

of *EWF* where the area of non-BIST version actually decreases, the multiplexer area increases as the BIST overhead decreases. The total area of the self-testable data path (area before BIST + BIST ovhd.) decreases from Flow I to Flow II and from Flow II to Flow III. The last column in Table 3 indicates the percentage decrease in total area of the self-testable versions of the data paths with respect to the self-testable version of the data path synthesized without any BIST considerations (Flow I). It can be seen that in the case of *DIF* and *EWF*, most of the reduction in total area comes from SFT and in the case of *FIR* most of it comes from AFT. The results indicate that the proposed scheduling and module assignment technique give a reduction of 30-50% in BIST area overhead an up to 10% in total area over traditional high-level synthesis techniques.

6   Conclusions

For minimizing BIST area overhead, it is desirable to share BIST resources across modules. In this paper we have shown how scheduling and module assignment

Table 3: Characteristics of synthesized data paths

DFG	Flow	L	#Reg	#Mod (type =)			#(n:1) Muxes (n =)						Area before BIST	BIST ovhd.	% BIST ovhd.	Total area	%red. in area
				+	*	-	2	3	4	5	6	7					
DIF	I	4	5	1	3	1	6	4	-	-	-	-	5306	1552	29.25	6858	-
	II	4	5	1	3	1	8	3	-	-	-	-	5306	1136	21.40	6442	5.50
	III	4	5	1	3	1	9	3	-	-	-	-	5402	1008	18.66	6410	6.53
ARF	I	8	16	4	8	-	11	2	2	-	3	-	14496	2560	17.66	17056	-
	II	8	16	4	8	-	11	4	2	3	-	-	14748	1792	12.15	16540	3.03
	III	8	16	4	8	-	14	5	3	3	-	-	15137	1072	7.08	16209	4.97
FIR	I	5	8	4	4	-	6	1	-	1	-	-	7363	3392	46.06	10755	-
	II	5	8	4	4	-	6	2	2	1	-	-	8099	2592	32.00	10691	0.60
	III	5	8	4	4	-	7	2	-	2	-	-	8110	2224	27.42	10334	3.92
EWF	I	14	10	5	3	-	8	3	2	2	1	1	10905	1776	16.28	12681	-
	II	14	8	5	3	-	6	7	6	2	-	-	10608	736	6.94	11344	10.54
	III	14	8	5	3	-	5	6	7	1	1	-	10661	736	6.90	11397	10.13

can affect the number of BIST resources required for a data path. The properties of schedules and module assignments that influence BIST resources have been incorporated into a 2-phase scheduling technique. In Phase 1, coarse scheduling is performed such that the latency and module requirement of the final data path is not compromised. In Phase 2, detailed scheduling is done by assigning operations to control steps and modules. The data paths synthesized by the proposed technique while having significantly lower BIST area overhead are competitive in terms of performance and area with those synthesized by traditional techniques.

Acknowledgments

References

[1] M. Abramovici, M.A. Breuer, and A.D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[2] C. Papachristou, S. Chiu, and H. Harmanani. A Data Path Synthesis Method for Self-Testable Designs. In *Proc. 28th Design Automation Conf.*, pages 378–384, June 1991.

[3] H. Harmanani and C. Papachristou. An Improved Method for RTL Synthesis with Testability Trade-offs. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 30–35, November 1993.

[4] L. Avra. Allocation and Assignment in High-level Synthesis for Self-testable Data Paths. In *Intn'l. Symp. on Circuits and Systems*, pages 463–472, August 1991.

[5] I. Parulkar, S.K. Gupta, and M.A. Breuer. Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead. In *Proc. 32nd Design Automation Conf.*, pages 395–401, June 1995.

[6] I.G. Harris and A. Orailoglu. Microarchitectural Synthesis of VLSI Designs with High Test Concurrency. In *Proc. 31st Design Automation Conf.*, pages 206–211, June 1994.

[7] I. Parulkar, S.K. Gupta, and M.A. Breuer. Lower Bounds on Test Resources for Data Flow Graphs. In *Proc. 33nd Design Automation Conf.*, pages 143–148, June 1996.

[8] Data Book. *G10-p Cell-Based ASIC Products*. LSI Logic Corp., May 1996.

[9] S-P Lin, C.A. Njinda, and M.A. Breuer. A Systematic Approach for Designing Testable VLSI Circuits. In *Proc. Intn'l Conf. on Computer-Aided Design*, pages 496–499, November 1991.

[10] I. Parulkar, S. K. Gupta, and M.A. Breuer. *Scheduling Data Flow Graphs for Minimizing BIST Area Overhead of Data Paths*. CEng Tech. Report 97-05, Univ. of Southern California, Dept. of Elect. Engineering - Systems, February 1997.

[11] G. D. Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[12] L. Stok. *Architectural Synthesis and Optimization of Digital Systems*. Ph.D. Dissertation, Eindhoven University, The Netherlands, 1991.

[13] L.J. Avra, L. Gerbaux, J-C. Giomi, F. Martinolle, and E.J. McCluskey. *A Synthesis-for-Test Design System*. Tech. Report CSL TR 94-622, Computer Systems Laboratory, Stanford University, May 1994.

[14] N. Dutt and C. Ramachandran. *Benchmarks for the 1992 High-level Synthesis Workshop*. Tech. Report 92-107, Univ. of California, Irvine, 1992.