Synthesizing Designs with Low-Cardinality Minimum Feedback Vertex Set for Partial Scan Application

Sujit Dey Miodrag Potkonjak Rabindra Roy
C&C Research Laboratories, NEC USA
Princeton, NJ 08540

Abstract

An efficient partial scan approach for cost-effective sequential ATPG is to select flip-flops (FFs) in the minimum feedback vertex set (MFVS) of the FF dependency graph, so that loops are broken. Through a comprehensive analysis of the sources of loops in the data path, this paper proposes a new high-level synthesis methodology to synthesize data paths which have low-cardinality MFVS, thereby reducing the cost of partial scan significantly. A test efficiency of 100% could be achieved for all designs synthesized by the proposed approach, requiring a significantly less number of FFs to be scanned compared to the original implementations.

1 Introduction

Automatic test pattern generation (ATPG) of sequential circuits is a very difficult problem [1]. While full-scan design solves the testability problem by converting the circuit into a combinational one during testing, each scan flip-flop (FF) incurs certain overhead, both in terms of area and performance. Alternatively, partial scan design has gained wide acceptance, since it requires to scan only a subset of FFs, while making the sequential circuit testable. The goal of a partial scan approach is to select a minimum number of FFs to be scanned such that sequential ATPG can achieve high fault coverage for the circuit under test.

Cheng and Agrawal [2] used the S-graph, which displays the dependencies among the FFs of a sequential circuit, to analyze the difficulty in testing a sequential circuit. In an S-graph, each node corresponds to a FF and there is a directed edge from node u to node v if there is a combinational path from FF u to FF v in the sequential circuit. Based on the observation that cycles in the S-graph are primarily responsible for sequential ATPG complexity, Cheng and Agrawal [2] presented an effective partial scan approach which selects FFs in the minimum feedback vertex set (MFVS) of the S-graph, after deleting self-loops, so that all loops, except self-loops, are broken. Since the problem of finding the MFVS is NP-Complete [3], several partial scan approaches have addressed the issue of finding heuristics to solve the MFVS problem [2, 4].

The mandatory tasks during high level synthesis are allocation, scheduling, and assignment [5], all of which have been shown to have significant impact on the testability of the synthesized designs [6, 7]. Recently, high level synthesis techniques have been used to generate easily testable data path circuits [6, 7, 8, 9, 10, 11, 12, 13]. A technique which selects scan FFs from high level specifications, and reuses the scan FFs during scheduling and assignment to synthesize designs without loops, has been proposed in [14].

In contrast to synthesizing a testable design free of loops by using scan FFs [14], this paper proposes a high-level synthesis methodology to synthesize circuits with low-cardinality MFVS. The design synthesized using the proposed methodology may have as many or more loops, and may be as hard to test, as the design synthesized by conventional high level synthesis techniques. However, the new design will have a MFVS whose cardinality is smaller than the original design, thereby reducing the cost of partial scan significantly.

Since we target datapath-intensive application domains like DSP, whose controller has only a few FFs, the control signals to the data path can be made fully controllable by scanning the state FFs of the controller. Consequently, we concentrate on the data path of the designs.

A circuit synthesized from behavioral specifications has a natural tendency to contain loops, partly due to the presence of loops in the CDFG, and partly due to hardware sharing used to optimize hardware resources like execution units. A comprehensive analysis of the formation of loops, in circuits synthesized by behavioral synthesis, is presented. The analysis uses the data dependencies and the compatibilities of the operations of the Control Data Flow Graph (CDFG) specification, to develop a regular expression-based representation to identify conditions under which loops will be created.

Based on the loop analysis, simultaneous scheduling and assignment algorithms are presented to implement circuits with reduced MFVS cardinality. To make the low-cardinality MFVS implementation cost effective, the scheduling and assignment algorithms simultaneously optimize for resource utilization.

We demonstrate the applicability of the proposed algorithms on some datapath-intensive benchmarks, selected from digital signal processing (DSP) applications. In each of the design, the MFVS of the circuit synthesized using the proposed algorithms is significantly smaller than the MFVS of the original circuit generated by conventional high level synthesis approaches. A gate-level partial scan tool requires to scan a significantly smaller number of flip-flops in the new designs to achieve 100% test efficiency.

2 Hardware Model

The area of ASIC implementation for numerically intensive applications is very often dominated by interconnect requirements [5, 15]. Interconnect requiremnets can be efficiently addressed if the hardware model shown in Figure 1, termed the *register file* model [16, 15], is accepted. In this hardware model, all registers are a priori clus-

0-8186-5440-6/94 $03.00

tered in a number of registers files. While each execution unit can send data to any register file in the general case, each register file is connected to one input of a single executional unit (EXU).

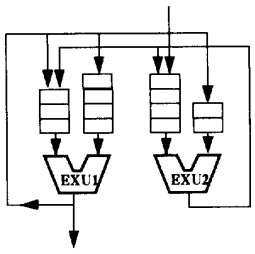

Figure 1: Register File Hardware Model

Numerous variations of the hardware model, where registers are grouped in register files and access to a particular register is limited to only some EXUs, are widely used in industrial designs. The model is used in several high level synthesis systems, e.g. Cathedral-II and Hyper [16, 15]. There are several reasons for adapting the register file model. First of all, its intentional interconnect restrictions powerfully enforce limitation on the number of interconnects and encourages heavy use of local interconnects (between register files and EXUs). Also, the register file model ensures that at any given time, only one EXU can write to a given register file, making all register files easy to implement. Finally, the area needed to layout registers and the combined control logic, which are grouped in register files, is significantly lower than in other hardware models.

While this paper addresses the register file model in particular, a straightforward modification of proposed algorithms is sufficient if other hardware models, without the interconnect restrictions, are used.

3 Formation of Loops in the Data Path

The data path synthesized from a CDFG can contain several types of loops. In this section, we identify and formulate the formation of different types of loops in the data path. We begin by modelling the data dependencies and the compatabilities of the operations of the CDFG by a Data-Dependency and Compatibility Graph (DDCG). Each node in the graph represents an operation of the CDFG. There is a (undirected) compatibility edge between two operations if there is a non-zero probability that both the operations can be assigned to the same module. There is a (directed) data-dependency edge from operation v to operation w if operation w depends on data produced by operation v. We will denote a compatibility edge between v and w as $c(v, w)$, and a data-dependency edge from v to w as $d(v, w)$.

Figure 2(a) shows segments of two paths in a CDFG, and Figure 2(b) shows the corresponding data-dependency and compatibility graph. Assuming that each operation in the CDFG takes one control cycle, the longest path is 3 control steps long. For a schedule which requires 3 control steps, the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) control steps in which each operation can be scheduled [17] is shown by the tuple [ASAP,ALAP] in Figure 2(a). In Figure 2(b), each compatibility edge $c(v, w)$, shown dotted, is weighted by an estimate of the compatibility between two nodes v and w, which is discussed below.

The mobility of an operation v, **mobility**(v), is the number of control steps in which it can be scheduled.

$$mobility(v) = ALAP(v) - ASAP(v) + 1;$$

The overlap between two operations, v and w, **overlap(u,w)**, is the number of control steps in which both v and w can be scheduled. Refering to Figure 2(a), mobility($+_1$) = 1, mobility($+_3$) = 2, and overlap($+_1, +_3$) = 1.

The estimate of the compatibility between two nodes v and w, **Comp(v,w)**, is the probability that v and w can be assigned to the same execution unit (module).

$$\text{Comp(v,w)} = \begin{cases} 1 & if \exists d(v, w) \\ 1 - \text{overlap(v,w)} \,/\, (\text{mobility(v)*mobility(w)}), & \text{otherwise} \end{cases}$$

The compatibility between nodes $+_1$ and $+_3$, Comp($+_1, +_3$) = $1 - (1/2) = 0.5$. All the compatibility edges c(v,w) are weighted by Comp(v,w), as shown in the DDCG in Figure 2(b).

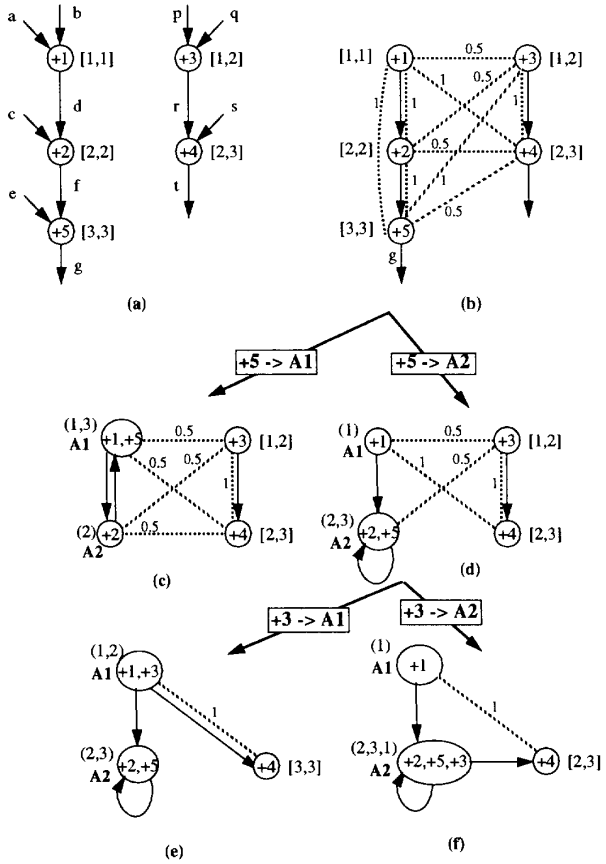

Figure 2: Illustrating the formation of loops: (a) A CDFG, (b) the corresponding DDCG, and (c)-(f) new DDCGs obtained by various assignments

3.1 Representing Paths By Regular Expressions

A **path** p from node x to node y is a sequence of nodes and edges starting with x and ending with y. A path is *simple* if all nodes and all edges on the path, except the first and last node, are distinct. A **cycle** is a simple path which begins and ends with the same node. Throughout this paper, by a path, we refer to a simple path.

The paths of the DDCG can be represented using regular expressions [18]. d^+ represents a path consisting of a sequence of one or more data-dependency edges d. The concatention of two paths p and q is represented by p.q, or simply, pq. For example, the regular expression cd^+ represents a path starting with a compatibility edge c, followed by one or more ocurrence of data-dependency edges. In Figure 2(b), the path $(+_5, +_1), (+_1, +_2), (+_2, +_5)$ can be represented by cd^+.

3.2 Detecting formation of Loops Using DDCG

The first type of loops, data-dependency loops, are formed in the data path due to the presence of cyclic data dependencies in the CDFG. Assignment loops and sequential false loops are formed due to hardware sharing.

(1) Data-Dependency Loop: A Data-Dependency Loop is formed in the data path if there exists a cycle of the form d^+ in the DDCG. In other words, if all the edges of a cycle in the DDCG, or the CDFG, are data-dependency edges, then a loop is formed in the data path, irrespective of the register and module assignment.

(2) Assignment Loop: During assignment of operations to modules (EXUs), we say that a compatibility edge is *used* if the two operations associated with the compatibility edge are assigned to the same module. An assignment loop is formed in the data path if there exists a cycle of the form cd^+ in the DDCG, and the compatibility edge c is used during module assignment. In other words, an assignment loop is formed whenever two or more operations in the path of a CDFG are assigned to the same module.

In the DDCG shown in Figure 2(b), there is a cycle $\{(+_5, +_1), (+_1, +_2), (+_2, +_5)\}$ of the form cd^+. Using the compatibility edge $(+_5, +_1)$ to assign the compatible operations $+_5$ and $+_1$ to the same module, creates an assignment loop in the data path. Let the schedule and assignment of the operations be: $\{+_1 : (1, A1), +_2 : (2, A2), +_3 : (2, A1), +_4 : (3, A2), +_5 : (3, A1)\}$. It satisfies the constraint of three control steps, and uses the minimum number of EXUs (2 adders). The resultant data path, shown in Figure 3(a), and its corresponding S-graph, has an assignment loop (RA1,LA2,RA1).

A self-loop is a special case: it can be formed in the data path either by the presence of a data-dependency loop of the form d, or an assignemnt loop of the form cd.

(3) Sequential False Loop: A sequential loop in the data path is termed false when the loop cannot be senstized. A false loop is a special case of a false path. Let the schedule and assignment of the operations of the CDFG in Figure 2(a) be: $\{+_1 : (1, A1), +_2 : (2, A2), +_3 : (1, A2), +_4 : (2, A1), +_5 : (3, A2)\}$. It satisfies the constraint of three control steps, and uses the minimum number of EXUs (2 adders). The resultant data path, shown in Figure 3(b), has two loops.

Consider the loop in Figure 3(b) shown in bold. To sensitize the loop, the required control signals to the multiplexers M1 and M4, c1 and c2, should be $\{c1 = 1, c2 = 0\}$ (or, $\{c2 = 0, c1 = 1\}$) in any two consecutive control steps. However, this necessitates execution of operations $+_4$ followed by $+_2$ (or $+_2$ followed by $+_4$), which is clearly not possible. Consequently, the sequential loop can never be sensitized, and is a false loop. The S-graph corresponding to the data path has a sequential false loop (LA1,RA2,LA1). Note that the only other loop in the data path is the self-loop (RA2,RA2) which is an assignment loop.

Stok considered the formation of false loops through resource (module) sharing [19]. However, the treatment of false loops involving data paths was limited to combinational loops. Combinational

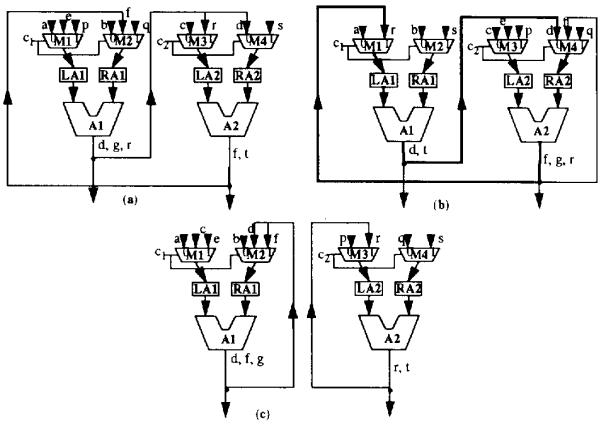

Figure 3: Data Paths formed by different assignments of CDFG in Figure 2(a): (a) Assignment Loop, (b) Sequential False Loop, (c) No Loops except Self-Loops

false loops, as analyzed in [19], are generated in the data path when data-chaining is allowed, that is two or more operations are scheduled in the same control step. Even when data-chaining is not allowed, resource sharing can still lead to false loops in the data path, *sequential* false loops (Figure 3(b)). Since we make the control signals to the data path fully controllable by scanning the state flip-flops of the controller, sequential false loops contribute to the complexity of sequential test pattern generation in the same way as any other loops.

Let $c = (v, w)$ be a compatibility edge in the DDCG such that there does not exist any data-dependency edge from v to w. If there is a cycle of the form $(cd^+)(cd^+)^+$ in the DDCG, and the compatible edges c in the cycle are used during module assignment, a sequential false loop is formed in the data path.

In the DDCG shown in Figure 2(b), there is a cycle $(+_4, +_1), (+_1, +_2), (+_2, +_3)(+_3, +_4)$ of the form $(cd^+)(cd^+)^+$. Assigning the compatible pairs $(+_4, +_1)$ to adder A1, and $(+_2, +_3)$ to adder A2, leads to a sequential false loop as illustrated in the data path in Figure 3(b).

4 Scheduling, Assignment and Allocation Algorithms

In this section, we present integrated scheduling and assignment algorithms to synthesize a design such that the implementation has

minimum feedback vertex set with a low-cardinality, while satisfying the user specified throughput requirements. To make the implementation competitive with respect to the hardware cost, it is mandatory to simultaneously optimize for resource utilization, considering all four components of implementation cost: execution units, registers, multiplexers and in particular, interconnects.

The new high level synthesis approach has three phases. The first phase is allocation of the set of execution units (EXUs), exclusively targetting resource utilization. For this task, we use Hyper [15]. This step guarantees that with respect to the area of EXUs, the synthesized design for MFVS is as good as any other design synthesized by available conventional behavioral synthesis tools which do not consider testability.

In the second phase, we simultaneously schedule and assign each operation of the CDFG while allocating interconnects and registers, so that resulting data path has high global resource utilization and its corresponding S-graph has as small as possible MFVS. In the final phase, the FFs belonging to the minimal feedback vertex set are identified and made scan FFs, using the gate-level partial scan tool OPUS [20].

After the initial allocation of EXUs, we simultaneously schedule and assign each operation of the CDFG, using global testability and resource utilization measures. The aim is to produce a testable data path, by either avoiding the formation of loops in the S-graph, or by ensuring that resulting S-graph has small minimum feedback vertex set. However, equal priority is also given to throughput (number of control steps) and resource utilization, so that the final design is not only cost effective in terms of number of FFs that need to be scanned, but also competitive in terms of hardware cost.

Since we use the path-by-path based scheduling paradigm, in order to provide globally optimal solution, in addition to the two mentioned measures, we also use the flexibility measure, which characterizes the difficulty of the remaining scheduling and assignment tasks after each decision.

At each iteration of the algorithm, from the operations that have not yet been scheduled and assigned, an operation op_i with the smallest slack (ALAP - ASAP) is selected. The set of (module, control step) pairs, $\{(M_i, C_i)\}$, where the module belongs to the set of modules to which the operation can be assigned and the control step belongs to the set of control steps in which the operation can be scheduled, are identified. For each pair, the cost in terms of the size of MFVS, resource utilization and flexibility for scheduling and assignment of subsequent operations, is computed. Subsequently, a pair with the smallest cost is selected. The scheduling and assignment algorithm is outlined below.

schedule_and_assign()

1. *while there exists a node which is not scheduled and assigned* {
2. *Select* op_i *with the smallest slack;*
3. $\{(M_i, C_i)\}$ = *set of (module, control step) pairs to/in which* op_i *can be assigned/scheduled;*
4. $cost(op_i, M_i, C_i) = \alpha * cost_{MFVS}(op_i, M_i, C_i) + \beta * cost_{RU}(op_i, M_i, C_i) + \gamma * cost_{Flex}(op_i, M_i, C_i)$;
5. *select* (M_i, C_i) *with the* **minimum** *cost;*
6. *assign inputs*(op_i) *to registers in the register files;*
7. *update the DDCG, the list of unscheduled operations, and their ASAP and ALAP times;*

}

4.1 Estimating the Size of MFVS Due to an Assignment

An assignment of an operation to a module may introduce assignment loops or false loops in the S-graph, which may require additional feedback vertices to break the loops. Also, a current assignment may force the formation of loops in the future, which will increase the size of the MFVS. In this section, we derive a cost function to measure how the size of the MFVS may increase due to a particular assignment.

The data dependency edges and the compatibility edges and their weights in the DDCG change when an operation is assigned to a module. Let $G_{DDCG}(op_i, M_i)$ be the DDCG obtained if operation op_i is assigned to module M_i. The new DDCGs produced by a sequence of assignments is shown in Figure 2. The initial DDCG is shown in Figure 2(b). Assume that operations $+_1$ and $+_2$ are assigned to adders $A1$ and $A2$ respectively. The DDCGs obtained by assigning $+_5$ to $A1$ and $A2$ are shown by Figures 2(c) and 2(d) respectively.

Let $G_D(op_i, M_i)$ be the subgraph of $G_{DDCG}(op_i, M_i)$, containing only the data dependency edges. A measure of the effect of assigning operation op_i to module M_i is:

$$cost_{\text{MFVS}}(op_i, M_i) = \alpha * \text{MFVS}(G_D(op_i, M_i)) + \beta * \text{MFVS}(G_{DDCG}(op_i, M_i)), \quad (1)$$

where the MFVS terms are the number of feedback vertices required to break all loops, besides self-loops, in the data-dependency graph, G_D, and the DDCG graph (comprising of both data-dependency and compatibility edges) formed after the assignment. The MFVS terms are calculated using heuristics like [2, 4, 20]. The feedback vertices thus found are operations, however since the operations will be mapped to modules, and the hardware model used binds registers to modules, the size of MFVS found using the G_D and G_{DDCG} graphs represent the MFVS size of the corresponding S-graph. Consequently, the first term reflects the size of the MVFS of the *current* S-graph, formed after the current assignment, and the second term reflects the upper bound on the size of the MFVS after *future* assignments.

Consider the choice of assigning operation +5 to either adder A1 or A2. Using equation 1 and the DDCGs shown in Figures 2(c) and 2(d), $cost_{\text{MFVS}}(+5, A1) = 1 + 2 = 3$, while $cost_{\text{MFVS}}(+5, A2) = 0 + 1 = 1$. The first terms can be explained by the observation that assigning $+_5$ to A1 leads to the formation of an assignment loop, and hence needs one feedback vertex, while assigning $+_5$ to A2 does not require any feedback vertex as no loop is formed. Hence, assuming that the size of MFVS is the prime consideration, operation +5 will be assigned to adder A2.

Next, consider the assignment of operation $+_3$ to either A1 or A2, and the corresponding DDCGs shown in Figures 2(e) and 2(f). Neither of the assignments forms any loop in the data path, and neither needs any feedback vertex immediately, reflected by a zero value for $\text{MFVS}(G_D(+_3, A1)$ and $\text{MFVS}(G_D(+_3, A2)$. However, assigning $+_3$ to A2 will create a loop (false loop) during the future assignment of operation $+_4$, which can be only assigned to A1, as shown in Figure 2(f). The future cost in terms of the MFVS size is reflected by the MFVS size of 1 of $\text{DDCG}(+_3, A2)$, as compared to

no feedback vertices required for DDCG($+_3$, $A1$), shown in Figure 2(e). Consequently, assignemnt of $+_3$ to A1 is selected, followed by assigning $+_4$ to A1. The resultant data path is shown in Figure 3(c). It has only two self-loops, and hence a MFVS of size 0.

4.2 Resource Utilization Cost and Flexibility Cost

While the primary target is the cardinality of the MFVS of the synthesized design, high resource utilization has to be ensured to make the design cost-effective for partial scan. The first phase in the overall behavioral synthesis scheme guarantees that there is no overhead with respect to the number of EXUs. We briefly outline the criteria used to achieve high resource utilization of registers, interconnect and multiplexers.

1. The operations which are likely to require additional hardware resources, with low future utilization, are addressed first, while the number of alternatives is still high.
2. Special attention is paid to interconnect. Introduction of interconnects which can not be well reused later should be avoided. Strong preference is given to local interconnect over global interconnect.
3. Registers are also an important part of the implementation cost. Any introduced register should have high likelihood to be effectively reused later.
4. Finally, we should also take into account cost associated with introduction of a new multiplexer.

We schedule and assign one CDFG node at a time, taking into account to the greatest possible extent all global consequences relevant to the four observations. We establish two criteria which are used to predict whether an interconnect will be local or global after the physical synthesis of the design. Obviously, an interconnect from an unit to itself will remain local after placement and routing. The second criteria is based on the observation that the greatest difficulty in routing often arises due to high congestion in some areas of the chip. To avoid congestion, during the interconnect assignment and allocation phase, we try to limit the number of interconnects which originate from or go to a particular register file.

For a particular assignment and schedule choice for an operation, we assign cost, in an increasing order, to the following resources used: (1) new multiplexer, (2) new register, (3) new local interconnect and (4) new global interconnect. When more than one scheduling and assignment decision have the same hardware cost, preference is given to the decision which introduces a new resource with higher likelihood for later reuse. Chance for reusability of an introduced resource is calculated by counting how many unscheduled and nonassigned CDFG nodes can use the new resource.

The flexibility cost component measures to what extent scheduling and assignment of an operation op_i to a particular control step and particular execution unit adversely affects the number of possibilities for scheduling and assignment of still unscheduled operations. Note, that due to data and control dependency edges, assignment and scheduling of one operation in the CDFG can have a number of consequences for assignment and scheduling of other operations. The DDCG graph provides convenient, fast and accurate information for characterizing the flexibility cost. The flexibility cost of the assignment of the operation op_i to the module M_i and control step C_i is proportional to the reduction of the weighted sum of compatibility edges in DDCG after the assignment of op_i to (M_i, C_i). The flexibility cost can be calculated in run time proportional the square of the number of nodes in the DDCG.

5 Experimental Results

We synthesized the following datapath-intensive benchmarks: (1) 3rd order cascade IIR Filter (3rdIIR) [21] (2) Speech Filter (Speech) [21], (3) MA Lattice Filter (MAL) [21], and (4) the popular 5th order elliptical wave digital filter (EWF) [17]. In the sequel, **Orig** (or **O**) refers to the original implementation using conventional high level synthesis techniques to ensure maximal throughput and minimal number of execution units. **SFT** refers to the low-cardinality MFVS implementation, using the new synthesis approach.

Design	B	CS	A	M	Reg		Mux		Inter	
					O	SFT	O	SFT	O	SFT
3rdIIR	16	5	2	3	11	11	12	8	12	9
Speech	20	17	2	3	12	12	20	9	20	9
MAL	16	7	2	2	10	9	10	8	11	10
EWF	16	17	3	3	23	24	29	32	20	27

Table 1: Characteristics of the Designs Synthesized

Table 1 shows the characteristics of the original and the SFT implementations for each benchmark synthesized. The column **Bits** refers to the word length. Both the original and the SFT implementation takes the same number of control steps **csteps** achieving maximal throughput, and need the minimum number of adders **Add** and multipliers **Mult**. The number of registers **Reg**, multiplexers **Mux** and interconnects **Inter**, are shown in Table 1 for the original and SFT implementations. They reflect the area overhead needed, if any, for low-cardinality MFVS implementation.

Design	FFs		SLoops		Edges		Max SCC	
	Orig	SFT	Orig	SFT	Orig	SFT	Orig	SFT
3rdIIR	176	176	64	64	3244	2230	144	144
Speech	240	240	140	100	7899	3461	200	200
MAL	160	144	48	48	2141	2061	128	3
EWF	368	384	272	144	32833	31942	352	352

Table 2: Characteristics of the S-Graphs at Gate Level

Table 2 shows the characteristics of the S-graphs of the flip-flops (gate-level) of both the original and the SFT implementations of the designs. The column **FFs** refers to the flip-flop nodes of an S-graph, the other nodes being inputs and outputs of the circuits. The column **SLoops** refers to the number of self-loops. The column **Edges** shows the number of edges in an S-graph, reflecting the number of dependencies between the FFs of the circuit. We also report the number of FFs in the largest strongly connected component of a circuit **Max SCC**, which shows the number of FFs contained in loops in the S-graph. For example, in the case of 3rdIIR, 144 FFs, out of a total of 176 FFs, are involved in loops in both the original and the SFT implementations.

Table 3 shows the size of the feedback vertex sets required to break all the loops, except self-loops, of the S-graphs of the original and SFT implementations. At the RT-level, the size of the minimum FVS is reported (in terms of registers), except for EWF, where the lower bounds of the FVS are reported. At the gate-level, OPUS [20] was used to identify the FVS (FFs) of the circuits. Table 3 shows that for each design, the size of the FVS needed for the SFT implementation is significantly less than the FVS needed for the original implementation. In the case of 3rdIIR, only 16 FFs are needed to break all loops, except self-loops, for the SFT circuit, compared to 48 FFs needed for the original circuit.

Design	Bits	Feedback Vertex Set			
		RT-Level (Regs)		Gate-Level (FFs)	
		Orig	SFT	Orig	SFT
3rdIIR	16	2	1	48	16
Speech	20	3	1	60	20
MAL	16	3	1	48	16
EWF	16	14†	9†	240	163

†: Lower bound

Table 3: Feedback Vertex Set of S-graphs at Register-Transfer and Gate Level

A timeframe expansion-based gate-level sequential ATPG tool, HITEC [1], was used to identify the testability of the circuits. Table 4 reports the ATPG results for the original and SFT implementations, without using partial scan. The total number of faults and the number of aborted faults (**Abt**) are reporte. Coulumn **FC%** shows the percentage fault coverage, which is the percentage of faults for which a test could be found. Column **TE%** gives the percentage test efficiency, which is the percentage of faults either for which a test could be found or which could be identified as redundant (that is, the percentage of faults not aborted). The ATPG time in seconds on a SUN Sparcstation 2 is also reported. The Table shows that in most cases, both the original as well as the SFT circuits are very hard to test.

Design	Type	Faults		FC%	TE%	CPU (secs)
		Total	Abt			
3rdIIR	Orig	7112	6732	1	5	26200.8
	SFT	6510	6104	1	6	22512.5
Speech	Orig	10004	9677	0	3	23883.7
	SFT	8514	8186	0	4	19614.2
MAL	Orig	6192	191	93	97	1159.6
	SFT	5928	71	95	99	656.0
EWF	Orig	9088	8879	0.3	2	25668.0
	SFT	9791	9514	0.2	3	29666.0

Table 4: Sequential ATPG without Partial Scan

Table 5 reports the ATPG results for each circuit after scanning the FFS of the feedback vertex set identified by OPUS. Column **FFs** shows the total number of FFS (**Total**), and the number of FFS that needed to be scanned (**Scan**). As expected, the effect of scanning the FFs of the FVS is remarkable: a test efficiency of 100% could be achieved for all the circuits. However, to achieve the high test efficiency, the SFT circuits need a significantly less number of FFs to be scanned compared to the original circuits. The experimental results clearly demonstrate the effectiveness of the proposed approach.

6 Conclusions

Partial scan is a popular Design-for-Testability technique for cost-effective sequential ATPG. An effective partial scan approach selects FFs in the minimum feedback vertex set. This paper proposes a high-level synthesis methodology to make partial scan even more cost-effective. The proposed technique can synthesize circuits having small minimum feedback vertex sets, thereby reducing the cost of partial scan significantly. Experimental results on several benchmarks demonstrate the ability of the new synthesis approach to generate circuits amenable to the partial scan methodology.

Design	Type	FFs		Faults		FC%	TE%	CPU (secs)
		Total	Scan	Total	Abt			
3rdIIR	Orig	176	48	7112	3	96	100	94.5
	SFT	176	16	6510	10	96	100	98.7
Speech	Orig	240	60	10004	3	97	100	163.9
	SFT	240	20	8514	22	96	100	192.8
MAL	Orig	160	48	6192	2	97	100	63.7
	SFT	144	16	5928	5	96	100	86.1
EWF	Orig	368	240	9088	0	98	100	209.1
	SFT	384	163	9791	0	97	100	183.3

Table 5: Sequential ATPG after scanning FFs in Feedback Vertex Set

References

[1] T. M. Niermann and J. H. Patel. HITEC: A Test Generation Package for Sequential Circuits. In *Proc. EDAC*, pages 214–218, 1991.

[2] K.T. Cheng and V.D. Agrawal. A Partial Scan Method for Sequential Circuits with Feedback. *IEEE Transactions on Computers*, 39(4):544 – 548, April 1990.

[3] M.R. Garey and D.S. Johnson. *Computers and Intractibility*. W.H. Freeman & Company, 1979.

[4] D.H. Lee and S.M. Reddy. On Determining Scan Flip-Flops in Partial-Scan Designs. In *Proceedings of the International Conference on Computer-Aided Design*, pages 322 – 325, November 1990.

[5] M.C. McFarland, A.C. Parker, and R. Camposano. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2):301 – 317, 1992.

[6] T. C. Lee, W. H. Wolf, and N. K. Jha. Behavioral Synthesis for Easy Testability using Data Path Scheduling. In *Proceedings of the International Conference on Computer-Aided Design*, pages 616–619, 1992.

[7] T. C. Lee, N. K. Jha, and W. H. Wolf. Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and Partial Scan Environments. In *Proc. Design Automation Conf.*, pages 292–297, 1993.

[8] J. Steensma and W. Geurts and F. Catthoor and H. De Man. Testability Analysis in High Level Data Path Synthesis. *Journal of Electronic Testing: Theory and Applications*, 4(1):43 – 56, Februray 1993.

[9] C. Papachristou, S. Chiu, and H. Harmanani. SYNTEST: a method for high-level SYNthesis with self-TESTability. In *Proceedings of the International Conference on Computer Design*, pages 458 – 462, October 1991.

[10] C.-H. Chen and D. G. Saab. Behavioral Synthesis for Testability. In *Proceedings of the International Conference on Computer-Aided Design*, pages 612 – 615, November 1992.

[11] A. Majumdar, K. Saluja, and R. Jain. Incorporating Testability Considerations in High-Level Synthesis. In *Proceedings of the International Symposium on Fault-Tolerant Computing*, 1992.

[12] L. Avra. Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths. In *Proceedings of the International Test Conference*, 1991.

[13] S. Bhattacharya, F. Brglez, and S. Dey. Transformations and Resynthesis for Testability of RT-Level Control-Data Path Specifications. *IEEE Transactions on VLSI Systems*, 1(3):304–318, September 1993.

[14] S. Dey, M. Potkonjak, and R. Roy. Exploiting Hardware Sharing in High Level Synthesis for Partial Scan Optimization. In *Proceedings of the International Conference on Computer-Aided Design*, pages 20 – 25, November 1993.

[15] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast Prototyping of Data Path Intensive Architectures. *IEEE Design and Test*, pages 40 – 51, 1991.

[16] H. De Man et. al. . Synthesis of DSP Systems at Leuven. In *Proc. of the IEEE ICCD*, pages 133 – 145, 1987.

[17] R.A. Walker and R. Camposano. *A Survey of high-level synthesis systems*. Kluwer Academic Publishers, Boston, MA, 1991.

[18] A. V. Aho et. al. . *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[19] L. Stok. False Loops through Resource Sharing. In *Proc. of the International Conference on Computer-Aided Design*, pages 345 – 348, Nov 1992.

[20] V. Chickermane and J. H. Patel. A Fault Oriented Partial Scan Design Approach. In *Proceedings of the International Conference on Computer-Aided Design*, pages 400 – 403, November 1991.

[21] R.A. Haddad and T.W. Parsons. *Digital Signal Processing: Theory, Applications and Hardware*. Computer Science Press, New York, NY, 1991.