Exploiting Hardware Sharing in High-Level Synthesis for Partial Scan Optimization

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

Abstract

A new approach to high level synthesis, which simultaneously addresses testability and resource utilization, is presented. We explore the relationship between hardware sharing, loops in the synthesized data-path, and partial scan overhead. Since loops make a circuit hard-to-test, a comprehensive analysis of the sources of loops in the data path, created during high level synthesis, is provided.

The paper introduces the problem of breaking CDFG loops with a minimal number of scan registers. Subsequent scheduling and assignment avoid formation of loops in the data path by sharing the scan registers, while ensuring high resource utilization. Experimental results demonstrate the effectiveness of the technique to synthesize easily testable data paths, with significantly less partial scan cost than a gate-level partial scan approach.

1 Introduction

The traditional goal of high level synthesis tasks, like scheduling, allocation and assignment, has been the optimization of performance under resource constraints or its dual. More recently, the list of goals was expanded to include fault tolerance and testability, the latter being the main topic of this paper.

Automatic test pattern generation (ATPG) of sequential circuits has been recognized as a difficult problem [1]. While full-scan design solves the testability problem, it can be very costly. Recently, partial scan design has gained wide acceptance, since only a subset of flip-flops (FFs) needs to be scanned to make the circuit testable.

The dependencies of the FFs of a sequential circuit is captured by an S-graph, where each node corresponds to a FF. 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. It has been observed by Cheng and Agrawal [2] that sequential test generation complexity grows exponentially with the length of cycles in the S-graph, and linearly with the longest path (sequential depth) in the S-graph. These observations are used by several gate-level partial scan approaches [2, 3, 4] to select scan FFs such that all loops in the S-graph, except self-loops, are broken and the sequential depth is minimal.

Recently, high level synthesis techniques have been used to generate easily testable data paths [5, 6, 7, 8, 9]. Chen and Saab [8] used a high-level testability analysis program to identify testable structures and synthesize them to improve testability. Lee et. al. [6, 7] developed a method to minimize the formation of loops in the data path by partial scan and proper register assignment.

It has been widely recognized that the implementation area in hardware shared architectures is most often dominated by intercon-

nect requirements [10]. In order to efficiently address interconnection cost, we use the *register file* model, where all registers are clustered 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 functional unit. A number of fully operational high level synthesis systems, such as Cathedral-II and Hyper, are based on the register file model [11, 12].

Since we target computation-intensive application domains, the controller states usually need only a few FFs. In our design for testability framework, we assume all the control signals to the data path to be made fully controllable by scanning the FFs of the controller.

In this paper, we concentrate on generating easily testable data paths from high-level specifications, achieving high resource utilization, and satisfying given performance constraints. We attempt to make the data path easily testable by ensuring that the synthesized data path has no loops, except self-loops. In the rest of the paper, we use loops to refer to loops of length greater than one.

Hardware sharing is a widely used methodology to achieve high resource utilization, but it may adversely affect the testability of a circuit by introducing new loops in the data path. However, when hardware sharing is exploited properly in conjunction with the partial scan methodology, improvements in testability can be achieved despite possible introduction of loops. The scan registers can be shared amongst several variables of the CDFG, to break not only the loops in the CDFG, but also the loops introduced in the data path by hardware sharing.

This paper exploits hardware sharing to minimize the number of scan registers needed to synthesize a minimal-loop data path. Experimental results show that our technique can synthesize very easily testable designs, with low hardware overhead, without compromising the performance of the designs. The partial scan cost incurred by our technique is significantly less than a gate-level partial scan approach.

2 Motivation

Consider the Control Data Flow Graph (CDFG) of the 4th order IIR cascade filter shown in Figure 1(a). Assume that each operation in the CDFG takes one control cycle. The critical path is 6 control cycles long. Figure 1(a) shows one possible feasible schedule and assignment, satisfying a performance constraint of 6 control cycles, and using minimal number of execution units. For instance, the operation $+_2$ is scheduled in control cycle 2, and assigned to be executed in adder A1, shown in Figure 1(a) by the ordered pair (2, A1). The resultant data path is shown in Figure 1(b).

Figure 1: 4th Order IIR Cascade Filter, satisfying performance constraint (6 control steps), and using minimum number of execution units (2A, 3M): (a) Scheduled and Assigned CDFG, (b) Data Path, and (c) S-Graph

Similar to the S-graph of a gate-level sequential circuit, the S-graph of a data path identifies the dependencies between the registers of the data path. The S-graph in Figure 1(c), corresponding to the data path in Figure 1(b), reveals the existence of several loops involving the registers of the data path. The longest loop in the S-graph has length 8. As can be expected, the data path is very hard to test, as indicated in Table 2 by the row *IIR.16 Orig*.

The testability of the data path can be improved using partial scan techniques at the gate-level [2, 3, 4], to break all the loops of the circuit. Breaking all the loops of the S-graph in Figure 1(c) needs scanning at least 3 registers, namely LA1, LA2, and LM1, which translates to $3n$ FFs, where n is the wordsize. However, the associated area and performance overheads due to the large number of scan FFs can be prohibitive.

The example in Figure 1 illustrates that hard-to-test data paths can be generated if testability of the data path is not considered during high-level synthesis. Instead of postponing the task of making the design testable to the gate-level, it is possible to incorporate testability as one of the design goals, besides performance and resource utiliza-

Figure 2: 4th Order IIR Cascade Filter, synthesized for **testability and resource utilization**, satisfying performance constraint (6 control steps), and using minimum number of execution units (2A, 3M): (a) Scheduled and Assigned CDFG, (b) Data Path, and (c) S-Graph

tion, during the various high-level synthesis tasks. Figure 2(a) shows the same flow graph of the IIR filter, with a different schedule and assignment, satisfying the same performance constraints as before. Note that this solution is also optimal with respect of the number of execution units used. The corresponding data path and the S-graph, shown in Figures 2(b) and 2(c) respectively, are significantly more amenable to sequential testability. Notice that the S-graph still has loops; however, scanning register RA2 will break all the loops. The resultant data path, with register RA2 scanned, has no loops, and is very easy-to-test. A test efficiency of 100% could be achieved on the resultant data path, as evidenced by the row *IIR.16 SFT* in Table 2.

3 Source of Loops in the Data Path

We present a comprehensive analysis of the source of loops in the data path, and its corresponding S-graph.

(1) **CDFG Loops:** A CDFG loop is formed in the data path if there exists a cycle consisting of data dependency edges in the CDFG. For example, in Figure 1(a), the CDFG loop (+5, D3, *5, +6, +5) will produce a loop in the data path, irrespective of the register and

module assignments used. If the assignment shown in Figure 1(a) is used, the loop (RA1,A1D1,LM1,RA2,RA1) is produced in the corresponding data path. Similarly, the assignment shown in Figure 2(a) produces the loop (RA2,A'2D1,LM1,RA1,RA2). In general, each cycle in the CDFG always produces a loop in the data-path S-graph, irrespective of the allocation, scheduling, and assignment performed.

(2) **Assignment Loops:** An assignment loop is created whenever two or more operations in the path of a CDFG are assigned to the same module. If two consecutive operations are assigned to the same module, a self-loop (loop of length one) is created. For example, in the CDFG shown in Figure 1(a), assigning the consecutive operations $+_4$ and $+_5$ to the same module A1 creates the self-loop (LA1-LA1) in the data path (Figure 1(b)).

When the operations along a CDFG path from operation u to operation v are assigned n separate modules, with u and v being assigned the same module, a loop of length n is created in the data path. Consider the path $(+_1, +_4, +_5, +_8)$ in the CDFG shown in Figure 1(a). Assigning operations $+_1$ and $+_8$ to the same module A2, and the intermediate operations $+_4$ and $+_5$ to module A1, creates a loop (LA2,LA1,LA2) of length two in the S-graph of Figure 1(c).

(3) **Sequential False Loops:** A sequential loop in the data path is termed false when the loop cannot be sensitized under normal operating conditions. A false loop is a special case of a false path. Figure 3 illustrates the formation of a sequential false loop in the data path. Figure 3(a) shows segments of two paths in a CDFG, where operations $+_1$ and $+_3$ are scheduled in control step 1, and $+_2$ and $+_4$ are scheduled in control step 2. If operations $+_1$ and $+_2$ are assigned adders A1 and A2, respectively, no assignment loop is formed. Similarly, assigning operations $+_3$ and $+_4$ to adders A2 and A1 respectively, ensures maximum resource utilization, while avoiding the formation of any assignment loop. However, the resulting data path in Figure 3(b) reveals the formation of a sequential loop shown in bold. To sensitize the loop, the required control signals to the multiplexors M1 and M2, c_1 and c_2 , should be $\{c_1 = 1, c_2 = 0\}$ (or, $\{c_1 = 0, c_2 = 1\}$) 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 under normal operating conditions, and is a false loop.

Figure 3: Generation of a Sequential False Loop in the Data Path: (a) CDFG fragment, (b) data path

Stok previously considered the formation of false loops through resource (module) sharing [13]. His treatment of false loops was limited to combinational loops, generated in the data path when data-chaining is used, that is two or more data-dependent operations

are scheduled in the same control step. However, even when data-chaining is not used, hardware sharing can lead to *sequential* false loops in the data path, as illustrated by Figure 3. Since we assume that the control signals to the data path are fully controllable, the false loops act as real loops during test generation, and contribute to the complexity of sequential ATPG in the same way as other loops.

(4) **Register File Cliques:** When a module M_i has a register file associated with each input, the registers in the register files may form a clique in the S-graph. Let a register belonging to a register file of module M_i be termed a *self-loop register* if it receives data from M_i , besides possibly other modules. Each self-loop register of module M_i has a path through module M_i to (from) itself and all other self-loop registers of M_i . In the corresponding S-graph, each self-loop register has an edge to (from) the other self-loop registers of module M_i . Consequently, if the left and right register files of module M_i have m and n self-loop registers respectively, then a clique of size $(m + n)$ is formed in the S-graph.

Figure 4: Register Files of Adder A1 in the EWF data path ([14]): (a) Lifetimes and register assignment of variables assigned to left register file, (b) Left and right register files of A1

Figure 4 shows the register files of an adder A1, used in the data path of the elliptical wave filter shown in [14]. Column *variables* and *Lifetime* show the left operands of each operation that was assigned to A1, and the lifetimes of the variables. Each variable has to be assigned to some register of the left register-file of A1, shown in column *Register Assignment*. Note that multiple registers are required due to conflicts in the lifetimes of the variables. For instance, the variables b, c, e and i are all alive in the 7th control step and receive data from A1; hence they need to be assigned to four different self-loop registers. The left register file has 5 registers: {L1 ... L5}, and the right register file has 4 registers: {R1 ... R4}. The inputs of the registers are shown. For instance, register L1 has a single input coming from module A1, while register R1 has three inputs, from A1, A2 and M1.

Since there are 5 self-loop registers in the left register file of A1, and 3 self-loop registers in the right register file, a clique involving the 8 self-loop registers is formed in the corresponding S-graph.

As each register in a clique is completely connected with all the other registers of the clique, breaking all the loops of a clique of size k requires scanning $k - 1$ registers. This means that formation of cliques not only makes test pattern generation very hard, it makes a partial scan solution very expensive.

4 Algorithms for Efficient Use of Partial Scan and Hardware Resources

A complex set of goals is imposed during scheduling and assignment which simultaneously addresses both hardware resource utilization and testability issues while satisfying the throughput constraints. In order to design a hardware competitive solution, it is mandatory to consider all three components of implementation cost: execution units, registers, and in particular, interconnects. Testability imposes requirements regarding all four types of loops in the data path graph, and, to a lesser extent, sequential depth.

Our approach to the allocation, scheduling and assignment problems has three phases. We start with an initial allocation of execution units, targeting exclusively resource utilization. For allocation, we currently use Hyper [12]. In the second phase, all CDFG loops are broken by assigning a subset of variables (scan variables) to scan registers. Each operation which consumes at least one scan variable is assigned to an execution unit (module), and the scan variable is assigned to the associated register file. In the third phase, we simultaneously schedule and assign each operation of the CDFG using global resource utilization and testability measures.

4.1 Breaking CDFG loops with Minimal Number of Scan Registers

In this section, we discuss the problem of breaking the CDFG loops using a minimal number of scan registers. A similar problem has been earlier addressed in the case of S-graphs of gate-level circuits [2, 3], where the minimum number of scan registers required to break all cycles equals the minimum feedback vertex set (MFVS) of the S-graph. Since the problem of finding the MFVS is NP-Complete, several heuristics were successfully used [2, 3].

When hardware sharing is not used, a solution to the MFVS problem can be directly applied to break the CDFG loops. However, when hardware sharing is used, the variables selected to break the loops (scan variables) can share the scan registers. Consequently, the MFVS of a CDFG may not be a good solution.

The new dimension added by hardware sharing to the problem of breaking loops is illustrated by the CDFG of the IIR filter shown in Figure 1. A possible solution to the MFVS problem is the edges: $\{(+1, D1), (+5, D3)\}$. However, since the variables D1 and D3 are simultaneously alive in the first control step, they cannot be shared, thus requiring 2 scan registers to break the CDFG loops. On the other hand, if we select as scan variables $(+2, +1)$ and $(+6, +5)$, all CDFG loops are broken. The scan variables can be now stored in the same scan register, since their lifetimes do not overlap.

Lee, Jha and Wolf [7] proposed an approach to cut CDFG loops using a subset of boundary variables (variables which correspond to the edges with delays). Since all boundary variables are alive simultaneously, each selected variable has to be assigned to a separate scan register. To maximize likelihood of reuse of the scan registers, they select boundary variables with short lifetimes. Later, during register assignment, they share scan registers among variables to minimize the formation of assignment loops in the data path.

While [7] introduces the important idea of sharing scan registers, the technique does not exploit hardware sharing while selecting scan variables to break CDFG loops. All boundary variables are simultaneously alive (in the first control step), and therefore cannot share scan registers. In contrast, considering all variables in CDFG loops as possible candidates for scan variables greatly improves chances

of efficient sharing of scan registers. Consider the IIR filter shown in Figure 1. Limiting the choice of scan variables to boundary variables results in the use of 2 scan registers. However, considering all variables in the CDFG loops results in a solution which uses only 1 scan register, as described earlier.

Also, the length of lifetimes of variables has only indirect and second order effect on hardware sharing. The necessary and sufficient conditions that two variables can share the same register are that they are not simultaneously alive and that proper interconnect for transferring the two variables is allocated.

The goal of our approach to break CDFG loops is to select scan variables such that the following criteria are simultaneously satisfied:

HSC1 All CDFG loops, except self-loops, are broken,

HSC2 The selected scan variables can be assigned to a minimum number of scan registers, and,

HSC3 Reusability of the scan registers, to break the other loops formed during the subsequent scheduling and assignment phase, is maximized;

We refer to the above problem as the minimum *hardware-shared cut* (HSC) problem. We address the minimum HSC problem by using a random walk-based approach which combines probabilistic and heuristic techniques. We use two measures, the loop cutting effectiveness measure, and the hardware sharing effectiveness measure, to capture the effectiveness of a variable in satisfying the three criteria of the minimum HSC problem. Details of the measures, techniques to compute them, and the algorithm which uses the measures to select the scan variables, can be found in [14].

After the scan variables have been selected to cut the CDFG loops, at first, a minimum set of scan registers is identified to which all the scan variables can be assigned. As a second step, the scan registers are selected from as many register files of as many execution units as possible. The second step increases the chances of reusing the scan registers to assign variables to avoid the formation of loops during scheduling and assignment.

5 Scheduling, Assignment and Allocation for Testability and Resource Utilization

After the CDFG loops have been broken using a minimal set of scan registers, in the third and final phase, 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, avoiding the formation of the three types of loops mentioned before. However, priority is also given to use a schedule and assignment which satisfies the constraint on control steps and which maximizes resource utilization, so that the final design is not only testable, but also competitive in terms of hardware cost.

At each iteration of the algorithm, from the operations that have not yet been scheduled and assigned, an operation op_i with least slack (ALAP - ASAP) is selected. The set of (module, control step) pairs, $\{(M_i, C_i)\}$, to which/ in which the operation can be assigned/scheduled, are identified. For each pair, the cost in terms of testability, resource utilization and flexibility for scheduling and assignment of subsequent operations, is computed. Subsequently, a pair with the smallest cost is selected. The cost measures will be described in the subsequent sections. We outline the algorithm for scheduling and assignment of operations.

schedule_and_assign()

```

1. while there exists a node which is not scheduled and assigned {
2.    $op_i = \text{select\_node}()$ ;
3.    $\{(M_i, C_i)\} = \text{set of (module, control step) pairs}$ 
   to in which  $op_i$  can be assigned/scheduled;
4.   compute  $\text{cost}(op_i, M_i, C_i) = \alpha * \text{cost}_{\text{Test}}(op_i, M_i, C_i) +$ 
 $\beta * \text{cost}_{\text{RU}}(op_i, M_i, C_i) + \gamma * \text{cost}_{\text{Flex}}(op_i, M_i, C_i)$ ;
5.   select  $(M_i, C_i)$  with the minimum cost;
6.   assign inputs( $op_i$ ) to scan registers/register files;
7.   update clique graph and data-path graph;
8.   update ASAP of all operations not yet scheduled;
9. }
10. assign all variables in register files to registers;

```

5.1 Testability Cost

A measure of the testability cost is given by equation 1. The first component of the cost measure, ($\text{size}_{\text{assign_loop}} + \text{cost}_{\text{assign_scan}}$), is the cost due to formation of assignment loops, where $\text{size}_{\text{assign_loop}}$ is the length of loops formed, and $\text{cost}_{\text{assign_scan}}$ is the cost of using some existing or new scan registers to break the loops. The second and third components deal with the other two types of loops, false loop and register file clique, while $\text{cost}_{\text{seq_depth}}$ measures the increase in sequential depth due to the assignment.

$$\begin{aligned} \text{cost}_{\text{Test}} = & (\text{size}_{\text{assign_loop}} + \text{cost}_{\text{assign_scan}}) \\ & + (\text{size}_{\text{false_loop}} + \text{cost}_{\text{loop_scan}}) \\ & + (\text{size}_{\text{clique}} + \text{cost}_{\text{clique_scan}}) + \text{cost}_{\text{seq_depth}} \end{aligned} \quad (1)$$

5.1.1 Cost Due to Formation of Assignment Loop

The cost due to formation of assignment loops is computed as follows. It is first checked whether assigning operation op_i to module M_j creates an assignment loop, by traversing the paths in the transitive fanin of operation op_i in the CDFG. For each loop created, attempt is first made to break the loop using any available scan register. If successful, then $\text{cost}_{\text{available_scan}}$ is added to the cost of scan registers, $\text{cost}_{\text{assign_scan}}$, depending upon whether the used scan register could have been used by some other operation in the same control steps. If the loop cannot be broken by any available scan register, and adding a new scan register is allowable by the user-specified limit of $\text{Max_scan_regs_allowed}$, a new scan register is used to break the loop. The cost of a new scan register, $\text{cost}_{\text{new_scan}}$, is added to $\text{cost}_{\text{assign_scan}}$. Note that if a loop is formed by assignment but is broken by using a scan register, then $\text{size}_{\text{assign_loop}} = 0$.

In case neither an available scan register can break the loop, nor a new scan register can be used, the assignment loop will be formed, and left unbroken, in the data path. To discourage assigning the operation op_i to module M_j which leaves a loop in the data path, the size of the loop formed is added to the cost function. In the later case, no scan register is used, and $\text{cost}_{\text{assign_scan}} = 0$.

The computation of the cost associated with the assignment of op_i to module M_j is given below.

cost_assignment_loop(op_i, M_j)

```

1.  $\text{size}_{\text{assign\_loop}} = \text{cost}_{\text{assign\_scan}} = 0$ ;
2. if (assignment_loop_introduced( $op_i, M_j$ ))
3.   for each loop {
4.     if loop can be broken by available scan register
5.        $\text{cost}_{\text{assign\_scan}} += \text{cost}_{\text{available\_scan}}$ ;
6.     else if (#available_scan_regs < Max_scan_regs_allowed) {

```

```

7.       add new scan register to available scan registers;
8.        $\text{cost}_{\text{assign\_scan}} += \text{cost}_{\text{new\_scan}}$ ;
9.     } /* end if */
10.   } else
11.     /* cannot use any new scan registers; allow loop to form */
12.      $\text{size}_{\text{assign\_loop}} += \text{size of loop introduced}$ ;
13.   } /* end for */

```

The cost due to the formation of false loops and cliques is computed in a way similar to assignment loops. The increase in sequential depth due to an assignment can be computed by traversing the transitive fanins of the operation being assigned.

5.2 Resource Utilization and Flexibility Cost

The area overhead for synthesizing a testable design should be minimal, so that the new approach has a significant advantage over gate level design for testability schemes [2, 3, 4]. We briefly outline the criteria used to achieve high resource utilization.

1. The most difficult operations for scheduling and assignment (the operations which are likely to require additional modules) are handled first, while the number of alternatives is still high.
2. Introduction of interconnects which can not be easily reused later is avoided. Strong preference is given to local interconnect over global interconnect.
3. Registers are an important part of the implementation cost. Any new register should have high likelihood to be reused later.

We establish two criteria to predict whether an interconnect will be local or global after the physical synthesis of the design. 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 register, (2) new local interconnect and (3) new global interconnect. When multiple scheduling and assignment choices have the same hardware cost, we prefer a choice which introduces a new resource with higher likelihood for later reuse. Reusability of a new resource is calculated by counting how many unscheduled and nonassigned CDFG nodes can use the resource.

The flexibility cost measures to what extent a particular schedule and assignment of an operation op_i adversely affects the flexibility for scheduling and assignment of subsequent operations in the transitive fanout of op_i . It is calculated by summing up the reduction in slacks of the operations in the transitive fanout of op_i .

6 Experimental Results

To evaluate the effectiveness of the new synthesis for testability technique, we synthesized the following datapath-intensive benchmarks: all-zero FIR wave digital filter (WDF), 4th order cascade IIR filter, and 5th order elliptical wave digital filter (EWF). In the sequel, **Orig** 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 implementation using the new synthesis for testability approach.

Table 1 shows various parameters of the original and SFT implementations after high level synthesis. The number of execution units (EXU) and control steps (CS) needed are same for both versions of

		CS	EXU	Reg	Mux	Inter
WDF	Orig	5	2A, 2M	10	8	11
	SFT	5	2A, 2M	9	6	10
4thIIR	Orig	6	2A, 3M	12	12	20
	SFT	6	2A, 3M	12	9	9
EWF	Orig	17	3A, 3M	23	29	20
	SFT	17	3A, 3M	24	32	27

Table 1: Performance and Hardware Costs of the Designs

the designs. In some cases, for example EWF, the SFT implementation needed a few more registers (**Reg**), multiplexers (**Mux**), and interconnects (**Inter**), indicating that testability improvement may result in a small increase in resource requirements. However, on the average, the area overhead is marginal.

The testability of the synthesized designs was evaluated using the gate-level sequential ATPG tool, HITEC [1], shown in Table 2. The numerical suffix after the name in the first column corresponds to the wordsize of the implementation. For each design, besides the rows corresponding to the original and SFT implementations, the other two rows correspond to circuits obtained from the original design by a gate-level partial scan tool OPUS [4]. The row **GPS_c** refers to the circuit obtained from *Orig* by OPUS, after breaking all loops (except self-loops) using scan FFs. The row **GPS_n** indicates the circuit obtained from *Orig* when OPUS was constrained to use the same number of scan FFs as present in the SFT design. For each version of a design, the total number of FFs and the number of FFs scanned are reported. The results of running HITEC on the four versions of each design are shown in Table 2. The total number of faults, the number of faults aborted by HITEC, the fault coverage and test efficiency achieved, and the ATPG time taken on a SUN Sparcstation 2 are reported.

Table 2 demonstrates that the SFT designs obtained by our technique were consistently more testable than the original designs obtained by traditional high level synthesis techniques. To achieve the same level of testability, the gate-level tool OPUS needed to scan a significantly larger number of FFs (**GPS_c**) than required by the SFT designs. For instance, in the case of EWF.20, OPUS needed to scan 300 FFs to break all loops (except self-loops) and achieve the same level of testability as the SFT design, which had only 60 scan FFs.

Moreover, when OPUS was restricted to scan the same number of FFs (**GPS_n**) as in the SFT design, the testability achieved was significantly lower than that of the SFT design. In the case of EWF.20, while the SFT design achieved 100% test efficiency in only 233 CPU seconds, **GPS_n** could only achieve 40% in 14895 CPU seconds.

7 Conclusions

This paper presented a new technique to exploit hardware sharing to minimize the number of scan registers needed to synthesize a testable data path. Novel algorithms have been proposed to select a minimal number of scan registers to cut CDFG loops, and reuse the scan registers during scheduling and assignment to avoid the formation of further loops in the data path. Experimental results

Design	Type	Total FFs	Scan FFs	Total faults	Abort faults	fault cov (%)	test eff. (%)	ATPG CPU (secs)
WDF.16	Orig	160	0	5798	286	91	95	74347
	GPS_c	160	48	5798	2	96	100	526
	GPS_n	160	16	5798	6	96	100	1515
	SFT	128	16	5534	3	96	100	755
IIR.16	Orig	192	0	7976	7709	0	3	20096
	GPS_c	192	64	7976	3	97	100	820
	GPS_n	192	16	7976	6762	12	15	17206
	SFT	192	16	6776	10	96	100	378
EWF.8	Orig	184	0	3698	3515	2	5	>36h
	GPS_c	184	121	3698	0	97	100	38
	GPS_n	184	24	3698	259	90	93	7167
	SFT	192	24	4108	5	97	100	649
EWF.20	Orig	460	0	10364	10077	1	3	>72h
	GPS_c	460	300	10364	0	98	100	310
	GPS_n	460	60	10364	6216	38	40	14895
	SFT	480	60	10916	16	98	100	233

Table 2: Effect of considering testability during high level synthesis

demonstrate the superiority of selecting partial scan registers during high level synthesis over partial scan selection at the gate level.

Acknowledgments

We thank Professor Janak Patel and Dr. Vivek Chickermane for providing us with HITEC and OPUS tools.

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] 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.
- [4] 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.
- [5] C. Papachristou, S. Chiu, and H. Harmanani. SYNTST: a method for high-level SYNthesis with self-TESTability. In *Proceedings of the International Conference on Computer Design*, pages 458–462, October 1991.
- [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] 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.
- [9] 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.
- [10] 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.
- [11] H. De Man et al. Synthesis of DSP Systems at Leuven. In *Proc. of the IEEE ICCD*, pages 133–145, 1987.
- [12] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak. Fast Prototyping of Data Path Intensive Architectures. *IEEE Design and Test*, pages 40–51, 1991.
- [13] L. Stok. False Loops through Resource Sharing. In *Proc. of the International Conference on Computer-Aided Design*, pages 345–348, Nov 1992.
- [14] S. Dey, M. Potkonjak, and R. Roy. Exploiting Hardware Sharing in High Level Synthesis for Partial Scan Optimization. Technical Report 93-C013, C&C Research Labs, NEC USA, April 1993.