Improving Power, Performance and Area with Test: A Case Study

Teresa McLaurin*, Ignatius P. Lawrence‡
*Arm Austin, TX, USA
‡Texas A&M University, College Station, TX, USA

Abstract

As more low power devices are needed for applications such as IOT, reducing power and area is becoming more critical. Reducing power consumption and area caused by using full scan should be considered as a method to help achieve these stricter requirements. This is especially important in designs using near-threshold technology. In this work, we use partial scan to attempt to improve power, performance and area on a CPU core and a GPU shader core. We present our non-scan DFF selection algorithm that maximizes non-scan DFF count while achieving ATPG results close to those of the full scan design on both a CPU and a GPU shader core. In addition, we present the PPA (power, performance and area) results of these designs for both the full scan and partial scan.

1. Introduction

Scan based testing is widely used in the industry today for manufacturing test. In a mux-D based full scan design, all DFFs in the design are converted into scan DFFs (SDFFs) which adds a 2-input multiplexer onto every DFF at its D-input. One input to this multiplexer is the normal functional input (that would have fed the D-input of the non-scan DFF directly). The other input is the Scan Input (SI) which is generally connected to the Q-output of another SDFF in the design. The selection between these two inputs is controlled by the Scan Enable (SE) signal. When SE=0, the functional input is selected. When SE=1, SI is selected. Thus, when SE=1, all SDFFs in the design are configured as shift register(s) also known as scan chain(s). Automatic Test Equipment (ATE) can shift test data in and out of these scan chains. Thus, all SDFFs become control/observe points and the problem of sequential testing becomes a combinational one. Figure 1 shows a simple scan-based design.

Scan chains introduce much more wiring into a design as there is a wire between the scan out (SO) of one SDFF to the scan in (SI) of the next SDFF in the scan chain. These wires often require hold fixing since these paths have little or no combinational logic in them and the DFFs are often physically close in proximity. Hold fixing can be costly in power and area as well as schedule, since this can become a complex task if there is a lot of hold fixing that must be done [1].

Partial scan is an old topic [2] [3] [4] [5] [6] [7] [8]. [9], [10] and [11]are relatively newer research in this area. In [9], the authors present a "cycle-cutting" approach to determine non-scan DFFs. A minimal set of DFFs are made scannable such that there are no non-scan DFF loops. In our work, we achieve this by non-scan to non-scan paths check (section 2.1). We show through our results that we need several other checks to get good ATPG results with partial scan. The authors in [10] inserted test points to improve controllability/observability in non-scan circuits. Our work does not involve addition of any test points. In [11], faults undetectable by sequential ATPG are identified and SDFFs are selected in order to detect them. While this work reported good fault coverage results, it did not focus on optimizing pattern count.

With the new low power requirements of the Internet of Things (IOT), perhaps it is time to revisit the possibilities of partial scan on current designs. Partial scan is simply leaving some of the DFFs non-scan. This means no extra mux on the DFF and no scan wire to the SI. Partial scan can help mitigate extra logic like hold buffers and wiring caused by full scan as shown in Figure 2.

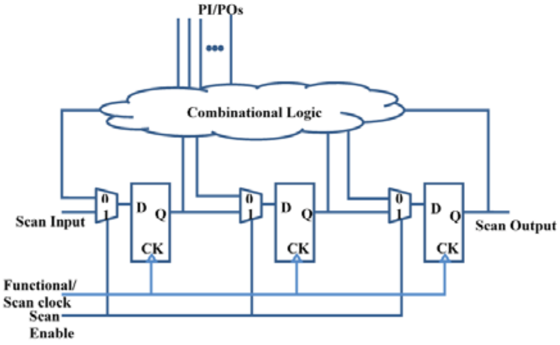


Figure 1: A Simple Scan-based Design

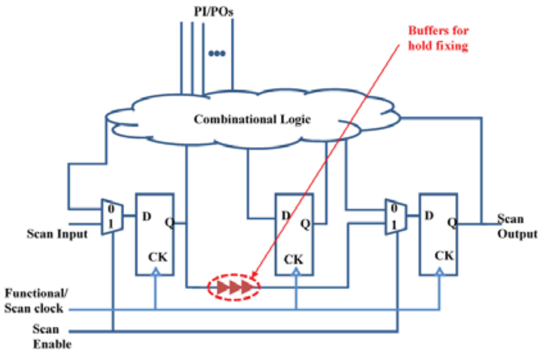


Figure 2: Partial Scan Design

Another result of partial scan is that due to fewer SDFFs, either the number of SDFFs in each scan chain decreases, the number of scan chains decrease, or both. These can help decrease test data volume.

978-1-5386-8382-8/18/$31.00 ©2018 IEEE

Of course, there are issues with partial scan methodology. The test problem becomes sequential for stuck-at fault (SAF) and more sequential for delay fault testing which means the number of capture cycles must be increased. Partial scan also introduces unknowns (Xs). Both of these issues can affect test data volume in a negative way. Values on non-scan DFFs are set through functional paths, making it more difficult for the ATPG tool to set desired values on these DFFs. This can have two negative effects on ATPG results:

i) Reduced Test Coverage (TC)
ii) Increased Pattern Count (PC)

For these reasons, partial scan has been mostly abandoned for over a decade.

An ideal partial scan design includes non-scan DFFs without paying any penalty in TC and PC. Keeping this in mind, we have arrived at a set of checks to classify a DFF as a non-scan DFF. We use the ATPG tool's ability to simulate the last few shift cycles to determine the values of non-scan DFFs mitigating the X propagation issue. We divide our checks into two levels. Level1 checks are run on all DFFs in the design. Level2 checks are a more complicated and time-consuming set of checks. Non-scan candidates that pass Level2 checks are the final non-scan DFFs. As we show through our case studies, for some cores Level1 checks may be sufficient. In that case, the DFFs that pass Level1 checks are the final non-scan DFFs.

We identify a new category of faults in SAF test which we refer to as "Sequentially Redundant" [12] faults. These SAFs are exposed in partial scan designs. We provide a simple yet efficient way to identify these faults and add a check in Level2 to minimize them.

We then perform SAF and transition delay fault (TDF) ATPG to prove the effectiveness of our non-scan DFF selection algorithm (henceforth called partial scan algorithm). Final TC is within 0.1% of the full scan design for both fault models. PPA results prove why partial scan is worth considering for modern designs, especially in areas such as near-threshold technology. Though we believe partial scan will be most ideal for a near-threshold design, case studies are on a CPU core CPU in a multicore processor such as is described in [13] and a GPU shader core as described in [14].

In Section 2, we present Level1 checks of our partial scan algorithm and introduce sequential redundancy. Section 3 describes Level2 checks. ATPG and PPA results are presented in Section 4, and our conclusions are covered in Section 5.

2. Partial scan algorithm – Level1 checks

Level1 checks are a set of preliminary checks run on all DFFs in the design. Subsequent sections present each of these checks. Only Level1 checks were performed on the CPU core as sufficient coverage was achieved with these. The GPU shader core did not achieve sufficient test coverage with these checks alone, so Level2 checks were added.

2.1. Non-scan to non-scan paths check

One of the Level1 rules is that there can be no paths where the launch point and capture point are both non-scan cells. This was done to help reduce the complexity of the logic for the ATPG tool. For n back-to-back non-scan DFFs, the faults captured by the first non-scan DFF need n + 1 capture cycles to be observed at an SDFF at the fan-out of the nth non-scan DFF. The extra complexity of multiple sequential non-scan DFFs may be too much for today's ATPG tools to handle. This also prevents cyclic non-scan paths.

2.2. Shift register check

In the case of a shift register, only the first DFF needs to be an SDFF and the remaining DFFs can be non-scan. Since shift register DFFs are automatically handled by EDA tools and are often already non-scan DFFs, we do not need to include those in our algorithm. For this reason, shift register DFFs fail this check.

2.3. Primary Input (PI) / Primary Output (PO) check

DFFs that are in the fan-out of PIs and those that are in the fan-in of POs are made SDFFs. This is because, at system level, the core will be integrated with other IP. We do not want to jeopardize controllability/observability of logic external to our IP due to their interface with non-scan DFFs. In short, DFFs connected to PIs and POs fail this check.

2.4. RAM check

DFFs that interface with memories are made SDFFs. This is to accommodate the possibility that memories may not include a scan collar. DFFs in the fan-in and fan-out of memories fail this check. Though we always use RAMs with internal scan chains to allow for control and observe of the RAM shadow logic, we cannot rely on this for our many different partners who implement our IP.

2.5. Integrated Clock Gating (ICG) check

DFFs that lie in the fan-in of ICGs are made SDFFs. Our designs are essentially 100% clock gated, so having non-scan DFFs in the logic to the enable of the clock gate as well as in the logic connected to the input of the DFFs becomes too complex for the ATPG tool to handle.

2.6. Self-drive check

DFFs that feedback to themselves fail this check and are scan inserted. DFFs that feedback to themselves are difficult to set to a desired value without any hardware modifications.

2.7. Clock Domain Crossing (CDC) check

Any DFF that has a fan-in DFF from a different clock domain is made an SDFF. The capture DFFs on CDC paths (that are either false or multicycle) will always be an X after the shift procedure. We do not consider such DFFs as non-scan to reduce the amount of Xs in the design before entering into capture cycles (we try to make the design as X-free as possible).

2.8. Fan-in and Fan-out check

Fan-in and fan-out to a DFF are considered because the more complex the control or the observe logic must be, the less likely the ATPG tool will be able to get coverage with a moderate number of patterns. We found that different types of designs (e.g. CPU1 vs CPU2 vs GPU) had different thresholds to achieve the partial scan percentages that were desired to affect PPA. Fan-in and fan-out became variables in the partial scan algorithm to achieve the minimum desired percentage of non-scan DFFs. In addition, when selecting non-scan DFFs with these criteria, the check starts with the DFFs with low fan-in/out as discussed in Section 3.4.

2.9. Sequential redundancy

By introducing non-scan DFFs, we effectively revert testing from being a purely combinational problem to a sequential one. This creates a new set of SAFs that do not exist in a full-scan design. Figure 3 shows an example of a “Sequentially Redundant” fault.

In Figure 3, DFFs FF1, FF2, FF3 and FF6 are scan DFFs. DFFs FF4 and FF5 are non-scan DFFs.

The values at the Q outputs of FF4 and FF5 in the current cycle depend on the Q outputs of FF1, FF2 and FF3 in the previous cycle. After shifting values into all SDFFs, two capture cycles are needed to propagate the faults captured by non-scan DFFs FF4 and FF5 to the SDFF FF6.

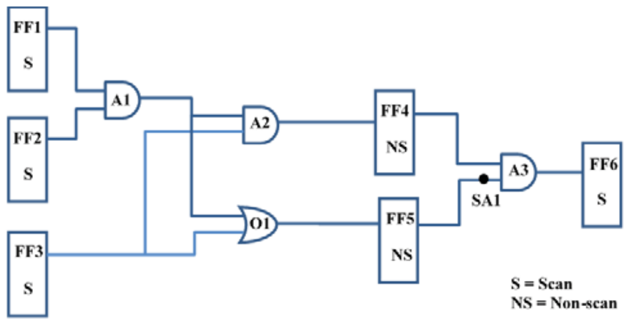


Figure 3: Sequential Redundancy Illustration

Table 1 shows the truth table for Q outputs of non-scan DFFs FF4 and FF5 as a function of Q outputs of scan DFFs FF1, FF2 and FF3.

Cycle n			Cycle n+1	
Q_{FF1}	Q_{FF2}	Q_{FF3}	Q_{FF4}	Q_{FF5}
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	0	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1: Truth Table for Circuit for Figure 3

To detect the stuck-at 1 (SA1) fault shown in Figure 3 we need the output values of FF4 and FF5 (QFF4, QFF5) set to 1 and 0, respectively. As shown in Table 1, (QFF4, QFF5) can never achieve the value set (1, 0). This SA1 defect is a sequentially redundant fault. If FF4 and FF5 had been SDFFs, they could have been set to any of the four possible values during shift mode and detected the SA1 fault.

We define a sequentially redundant fault as a fault whose presence does not affect the behavior of the sequential circuit. Combinational redundant faults (which are automatically detected by the ATPG tool) are not included in our definition of sequentially redundant faults. Sequential and combinational redundant logic can be present in a design due to improper RTL coding style or ineffective optimization by the synthesis tool. Sometimes, they are deliberately intended to address timing issues. Sequentially redundant faults were found in our partial scan design and must be addressed if they are significant in number.

Classical research work on identifying sequential redundancy has focused on logic optimization during synthesis. Sequential redundancy was detected in [15] by looking for don’t care conditions in state transition diagrams. In [16], a circuit was made feedback free by cutting at the feedback lines assuming they are fully controllable and observable. Test generation and fault simulation are then performed on this feedback free circuit. C-cycle redundancy was presented in [17]. In this work, an arbitrary set of inputs are provided to the circuit for c clock cycles (where c >= 0). The possible states of the circuit at the end of these c cycles are analyzed for both fault-free and faulty cases to determine sequential redundancy. If there exists a state S_f in the faulty circuit and a state S in the fault-free circuit such that the response of both the circuits to any input I is the same, then the fault is considered sequentially redundant. The circuit is then optimized by removing the region associated with that fault. It was shown that different benchmark circuits needed different values of c to detect all the redundant faults.

We came up with a simple way of determining the possibility of sequentially redundant faults even before generating the partial-scan design. We first perform SAF and Launch-off Capture (LOC) TDF ATPG on the full-scan design. Sequentially redundant faults are uncovered during TDF ATPG. If we consider the sequentially redundant SA1 fault in Figure 3, the slow-to-fall fault on the same node will remain uncovered. This is because, during the launch of capture transition, all DFFs are in their functional mode and sequentially redundant faults are exposed. During SAF ATPG, the sequentially redundant faults are hidden since there is control and observe at every DFF. We determine sequentially redundant faults as those that are undetected during full-scan transition delay ATPG and detected during full-scan SAF ATPG. This technique does not involve using synthesis or test generation/fault simulation engines. Instead, it only relies on existing full-scan ATPG results.

Determining sequentially redundant faults beforehand, allows us to use that information in a Level2 check of our partial scan algorithm. This enables us to generate a partial scan netlist with a minimal amount of sequentially redundant faults.

2.10. Other X-generation issues

Even though we simulate the last few shift cycles, we found that there was still some X-generation. Since the value in the last DFF of a scan chain is known only after all shift cycles, the non-scan DFFs that lie in the fan-out of these DFFs propagate X's during the capture phase. Figure 4 shows two non-scan DFFs in the fan-out of last SDFF in a scan chain. These DFFs will capture X's during the first capture cycle since the tool is unable to simulate shift values for the last DFF in a scan chain. To prevent this, we override the set/reset ports of all non-scan DFFs that are functionally adjacent to the last SDFF of each scan chain with the scan enable signal so that they are either 1 or 0 at the end of shift instead of X as described in [18]. If these non-scan DFFs are neither set nor reset DFFs, then we use a set or reset DFF to be able to initialize them during shift. Figure 5 shows an example of how a resettable non-scan DFF on the fan-out of last DFFs in scan chains is handled. A settable DFF would be handled in the same manner where shift enables the set, rather than the reset. This resolved the issue that was causing the X-generation.

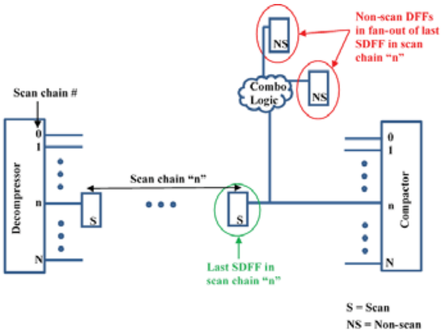


Figure 4: Non-scan DFFs in Fan-out of Last SDFF in Scan Chain

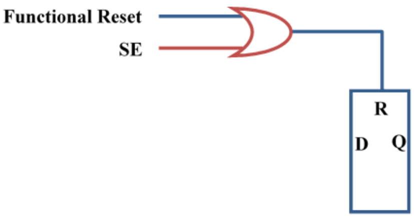


Figure 5: Set/Reset Overriding for Non-scan DFF on Fan-out of Last DFF in Scan Chain

3. Partial scan algorithm – Level2 checks

Level2 checks are performed on non-scan candidate DFFs, i.e. DFFs that pass Level1 checks. Subsequent sections specify each of these checks. These checks were added for the computationally heavy GPU shader core which could not achieve the test coverage numbers of the CPU core with only the Level1 checks. These additional checks have not been evaluated on a CPU core.

3.1. Overlap threshold check

The overlap threshold check was added to control the number of non-scan DFFs that feed into a single combinational logic cloud. The GPU is a computationally intensive design, resulting in DFFs whose fan-in ranges from 1 to 18000 and fanout from 1 to 4000. We looked at modules that included a high level of computational logic and identified a common problem: high fan-in from non-scan DFFs into combinational logic resulted in too many uncovered faults in that logic. This condition also exacerbated the propagation of faults captured by non-scan DFFs that needed to be propagated through that logic. With these observations, we concluded that in certain modules, when high fan-in includes many non-scan DFFs, this can reduce test coverage in the area significantly. We call this the "overlap threshold" problem. Figure 6 illustrates this condition.

In Figure 6, FF1, FF2, FF3, FF4, FF5, FF7 and FF8 are non-scan DFFs feeding-in to the combinational logic fan-in to SDFF FF6. For faults captured by FF2 to propagate to FF6, FF1, FF3, FF4 and FF8 must be a 1 while DFFs FF5 and FF7 must be a 0. Since non-scan DFFs are weak control points, it might be tough to satisfy all the constraints.

To limit the number of non-scan DFFs feeding in to a combinational logic cloud, we limit the number of non-scan DFFs that fan-in to an SDFF. In Figure 6, if we limit the maximum number of non-scan DFFs that fan-in to SDFF FF6 to some value n, it is guaranteed that the combinational logic cloud (AND gate + OR gate) shown in the figure will not have more than n non-scan DFFs feeding in. This is a simple yet effective way of implementing the overlap threshold check. In the next section, we will reveal how we addressed both sequential redundancy and the overlap threshold problem by using only the overlap threshold check.

Sequential redundancy was described in Section 2.9. In this section, we will talk about how the sequential redundancy check was implemented using the overlap threshold check. To create a simpler implementation, we chose an aggressive way to tackle sequential redundancy.
Non-scan DFFs that illuminate sequential redundancy have common fan-in and fan-out DFFs. This is evident in Figure 3. In the event of two or more non-scan candidate DFFs having common fan-in and fan-out DFFs, only one of those candidates will pass the sequential redundancy check.

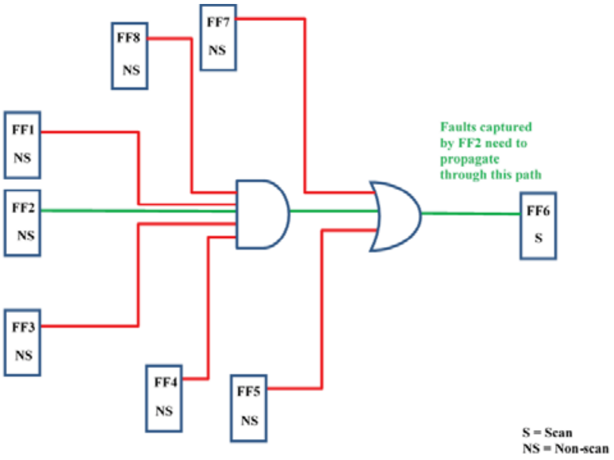


Figure 6: Non-scan DFFs Feeding into Combinational Logic

3.2. Sequential redundancy check

In Figure 7, only FF4 or FF5 will end up passing the check. Just because there are non-scan candidates that have common fan-in and fan-out DFFs does not mean there will be sequential redundancy. However, the simplified flow is more aggressive in replacing these DFFs with SDFFs. Figure 7 illustrates the parallels between sequential redundancy and the overlap threshold problem.

In Figure 7, there are two non-scan DFF candidates FF3 and FF4 that need to be evaluated for sequential redundancy and the overlap threshold condition. The candidates have common fan-in DFFs: FF1 and FF2, and common fan-out DFFs: FF5 and FF6.

Hence, only one of them will end up passing the sequential redundancy check based on the order in which they are processed. If FF3 is processed before FF4, FF4 will fail the sequential redundancy check. While checking for the overlap threshold condition, we consider the DFFs in the fan-out of the candidate and ensure that the number of non-scan DFFs that fan-in is below the set threshold.

When FF3 is processed, we look at the number of non-scan DFFs that fan-in to DFFs FF5 and FF6; the same is true for FF4. Now, if we set the overlap threshold limit to one (i.e. only one non-scan DFF can fan-in to a combinational logic cloud), FF5 and FF6 will have only one non-scan DFF in their fan-in. Assuming FF3 is processed before FF4, FF3 will end up passing the overlap threshold check and FF4 will not. Thus, the overlap threshold limit of one meets the sequential redundancy requirements.

Implementing an overlap threshold limit of 1 is very aggressive and we ended up losing a considerable number of non-scan DFFs. Our non-scan DFF count dropped from ~30% to ~20% after adding this check. But by losing only a third of the non-scan DFFs, we regained two thirds of the lost SAF coverage.

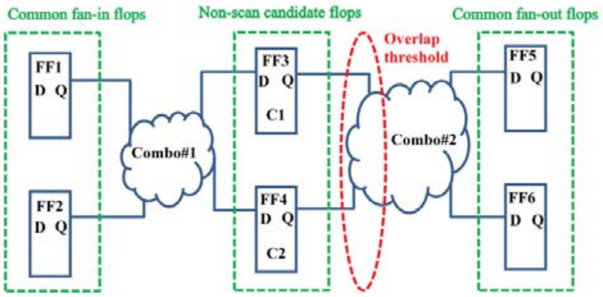


Figure 7: Parallels between Sequential Redundancy and Overlap Threshold Conditions

The overlap threshold check enables a lot of flexibility. We can set the overlap threshold limit to one to push the coverage as high as possible. An overlap threshold limit of one is the most conservative value: it completely gets rid of sequential redundancy and the overlap threshold problem. However, the threshold can be varied anywhere from one to the maximum fan-in minus one. With higher threshold values, we can get a higher non-scan DFF count usually at the cost of lower coverage and/or a higher pattern count due to an increase in sequential redundancy and the overlap threshold problem. However, as in LOC transition delay testing, one could argue that the lost coverage is for faults that can never occur during functional operation.

The overlap threshold step was not required for CPU to meet test coverage requirements.

3.3. Identification of problematic DFFs

After addressing the overlap threshold and sequential redundancy problems, we still saw around 0.1% lower test coverage as compared to the full scan design. The non-scan DFFs that affected these faults, either in the observe or control paths, were collected. We refer to these non-scan DFFs as “problematic” DFFs. We associated a weight with each problematic DFF. Weight indicates the number of uncovered faults affected (on a control or observe path) by the corresponding problematic DFF. So, the higher the weight, the more problematic the DFF is with regard to affecting test coverage. We then ranked these problematic DFFs in decreasing weight order to identify the worst ones.

We identified the below properties of the problematic DFFs:

i) Many were high fan-in/fan-out DFFs. High fan-in/fan-out DFFs affect more logic hence can be problematic. Based on this, we adjusted the maximum fan-in threshold to 120 and maximum fan-out threshold to 100 on the GPU shader core. Note that these thresholds have nothing to do with overlap threshold. These are the thresholds described in Section 2.8.

ii) There were low fan-in/fan-out problematic DFFs. These DFFs were involved in arithmetic intensive logic and the combinational logic depth between these DFFs and the SDFFs in their fan-out was high.

There are two ways to address the low fan-in/fan-out problematic DFFs problem:

i) We select the problematic DFFs whose weight is above a certain value and exclude them while processing non-scan DFF candidates. This is a more accurate approach with some iterative work involved. We call this the "semi-automatic" way of identifying problematic DFFs. For the GPU shader core, we selected 100 as the cut-off weight. So, any problematic DFF whose weight was above or equal to 100 was excluded and made SDFF.

ii) We check if a candidate is involved in arithmetic logic and exclude it if none of its fan-out DFFs have a fan-in below a certain threshold. To check if a candidate is involved in arithmetic logic, we look for adder cells in the fan-in/fan-out. By having at least one fan-out DFF with low fan-in, the non-scan DFF has at least one easy path to propagate the faults captured. Here, we assume that a low fan-in DFF has smaller combinational logic in its fan-in. This is an automatic but less accurate technique than i).

3.4. Ordering non-scan candidate DFFs

In three of our checks: overlap threshold check, sequential redundancy check and non-scan to non-scan connectivity check, the order in which we process the non-scan candidate DFFs has a significant impact on the non-scan DFF count and ATPG results. Non-scan candidates that are processed early and pass all checks jeopardize the chances of other candidate DFFs that overlap with them. Besides, it is guaranteed that the non-scan candidates in their fan-in and fan-out will be made SDFFs. This can be disastrous to non-scan DFF count if high fan-in/fan-out DFFs are processed early on.

We process non-scan candidate DFFs in order of increasing fan-in. All candidates with a given fan-in are processed in order of increasing fan-out. We gave preference to low fan-in over low fan-out DFFs because it is easier to set desired values in low fan-in DFFs due to smaller combinational logic in the fan-in cone.

Figure 8 through Figure 10 show our partial scan algorithm. Note that the flow charts show the various checks in the order in which they are implemented. Non-scan to non-scan paths check which is a Level1 check is moved to the end of the flow.

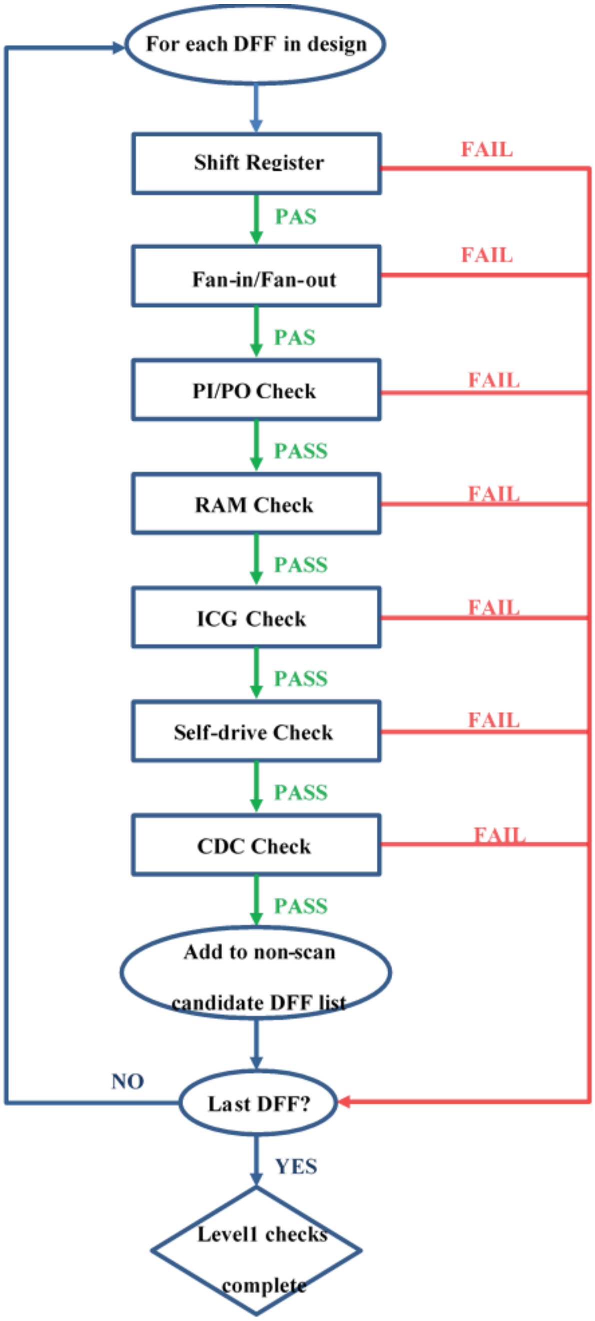


Figure 8: Level1 Checks (except non-scan to non-scan connectivity check)

4. Results and discussion

All ATPG results shown in this section are with compression enabled.

The same compression hardware configuration is used for both full scan and partial scan. Since the number of scan chains is fixed, there are fewer DFFs per scan chain in partial scan when compared to full-scan. This means fewer shift cycles per pattern. Hence, to make a fair comparison with regard to pattern count, we introduce the parameter "pattern volume" which is defined below:

Pattern volume = (Pattern count) x (Number of DFFs in longest scan chain)

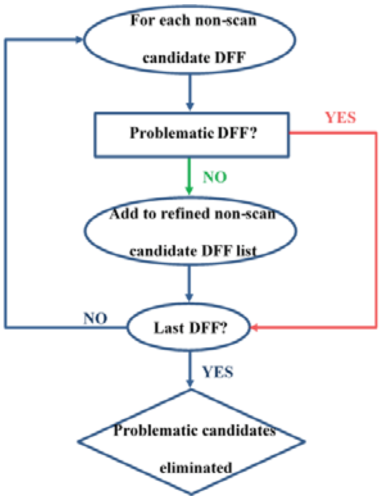

For each non-scan candidate DFF
Problematic DFF?
YES
NO
Add to refined non-scan candidate DFF list
NO
Last DFF?
YES
Problematic candidates eliminated

Figure 9: Elimination of problematic DFFs

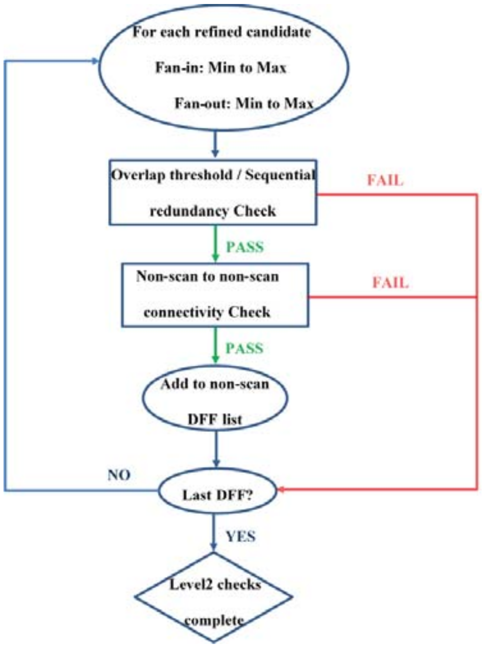


Figure 10: Level2 Checks and Non-scan to Non-scan Connectivity Check

4.1. CPU Core results

Four different experiments were run on the CPU core with different non-scan DFF percentages. The percentage was varied by changing the fan-in and fan-out variable during non-scan DFF selection. The CPU core contains approximately 250K DFFs and only uses Level1 checks. There can be multiple CPU cores in a microprocessor which multiplies the PPA advantages of partial scan. Figure 11 shows the CPU core SAF test coverage and pattern count/volume.

The four different netlists use 0% (full scan), 9%, 15% and 35% non-scan DFFs. Full scan achieved 99.79% test coverage with 7164 patterns. Note that by final release of Arm® IP, the stuck-at TC is required to meet 99.9% or a path on how to get to 99.9% stuck-at TC must be described. When the test coverage data was collected, this design was early in schedule, so this work was not complete. The SAF TC did drop slightly for the 9% (99.74%) and 13% (99.72%) netlists. The 35% non-scan netlist did achieve over 99% SAF test coverage, as well. The pattern volume for the 9%, 13% and 35% partial scan netlists, when compared to the full scan netlist was 32%, 62% and 143% larger, respectively.

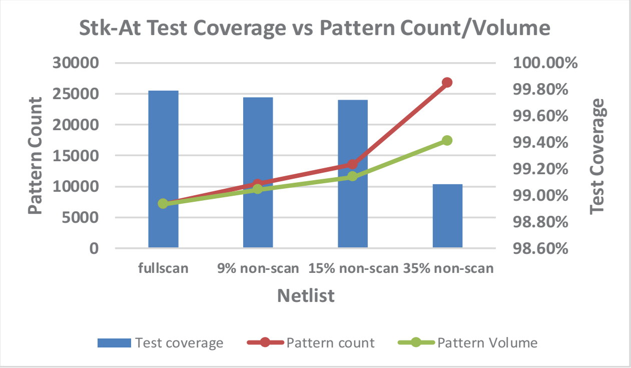


Figure 11: CPU Core SAF TC

Figure 12 shows the TDF TC and pattern count/volume. As can be seen, the LOC transition delay coverage is very high on the CPU core with full scan (97.49%). The test coverage remains high for most of the netlists and the 35% non-scan netlist still achieves greater than 90% test coverage.

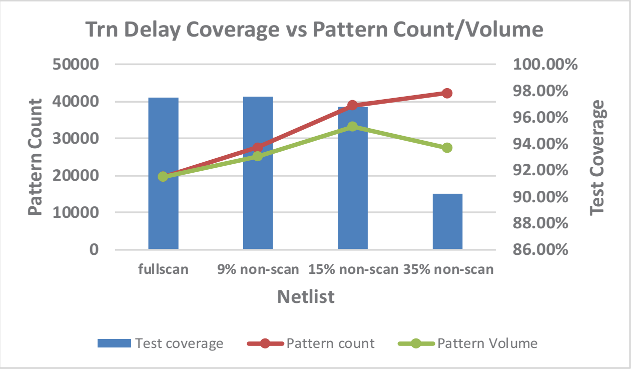
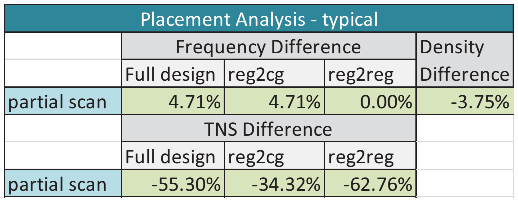


Figure 12: CPU Core TDF TC

The 9% and 15% partial scan netlist test coverages are respectively 97.56% and 96.79%. Interestingly, the 9% partial scan netlist achieves slightly higher test coverage than the full scan netlist. The reason behind this was not evaluated. The 35% partial scan netlist only achieves transition delay TC of 90.25% and may not be a good candidate for manufacturing. The pattern volume increases by 28% and 69% for the 9% and 15% netlists respectively. The test coverage loss for both SAF and TDF on the 9% and 15% partial scan netlists, though minimal, is still coverage loss. It must be considered what end markets may still be targeted for SoCs using the partial scan methodology. For instance, automotive would likely not be a good choice.

The question we then asked is whether the PPA improvement is worth the extra test time. We evaluated the PPA of this core with a 19% non-scan netlist. The full flow comprises synthesis → placement → clock tree synthesis (CTS) → postCTS hold → route → postroute → postroute hold. Table 2 shows the improvements shown in the partial scan netlist after the placement step.

.

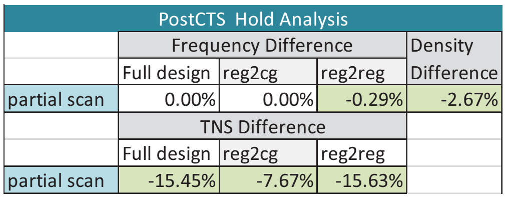


Placement Analysis - typical				
	Frequency Difference			Density
	Full design	reg2cg	reg2reg	Difference
partial scan	4.71%	4.71%	0.00%	-3.75%
	TNS Difference			
	Full design	reg2cg	reg2reg	
partial scan	-55.30%	-34.32%	-62.76%	

Table 2: CPU Core Post-Route Timing Analysis

There was an increase of 4.71% for frequency which came from the reg2cg (register to clock gate) paths, not the reg2reg (register to register) paths. The TNS (total negative slack) is the sum of all negative slacks. Though frequency did not improve for reg2reg, the number of paths with negative slack (paths missing the targeted frequency) vastly improved. TNS improved for all paths combined by 55% and the density of the circuit improved by 3.75%. When the density is reduced, it means that there is more room for routing, the frequency may be higher, power may be improved, and/or the design floorplan may be able to be reduced. These all contribute to helping the design reach its target frequency and routing more easily

Table 3 shows the post-CTS hold analysis. In this run, the clock tree has been added and hold time analysis has been done.

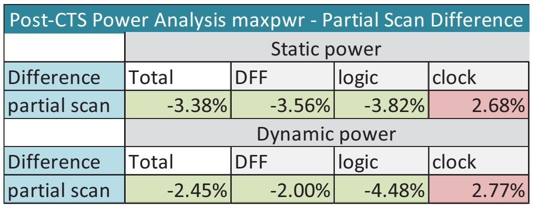


PostCTS Hold Analysis				
	Frequency Difference			Density
	Full design	reg2cg	reg2reg	Difference
partial scan	0.00%	0.00%	-0.29%	-2.67%
	TNS Difference			
	Full design	reg2cg	reg2reg	
partial scan	-15.45%	-7.67%	-15.63%	

Table 3: CPU Core postCTS/Hold Analysis

Now the frequency between the two netlists is very close. However, the TNS for all paths combined improves by 15.45% on the partial scan netlist and this is considered a more important metric since the tools have less paths to work on and so don't have to work as hard. The density is better by 2.67%. The reg2reg paths have a slightly slower frequency on the partial scan netlist (.008 GHz). However, looking at the total picture, it is believed that the partial scan netlist is better than the full scan netlist with regard to meeting the PPA goals. But is it enough to make it worthwhile doing partial scan on this type of design?

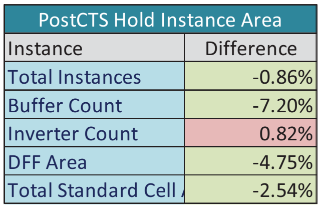
We then evaluated dynamic power with our maximum power (maxpwr) test. Table 4 shows that indeed the power during functional mode did reduce on the partial scan netlist by 2.41%, which is significant. The clock power increased. In the past this has been caused by the floorplan not being reduced for the smaller area of the partial scan netlist, which results in the clock signals traveling further.



Post-CTS Power Analysis maxpwr - Partial Scan Difference				
	Static power			
Difference	Total	DFF	logic	clock
partial scan	-3.38%	-3.56%	-3.82%	2.68%
	Dynamic power			
Difference	Total	DFF	logic	clock
partial scan	-2.45%	-2.00%	-4.48%	2.77%

Table 4: CPU Core Maxpwr Results

Though we did not shrink the floorplan, we did analyze how much the area reduced for various instances. Table 5 shows the difference in total instance area as well as a breakdown of several types of instances with regard to area or count. If this were resolved, then the total power savings would increase.



PostCTS Hold Instance Area	
Instance	Difference
Total Instances	-0.86%
Buffer Count	-7.20%
Inverter Count	0.82%
DFF Area	-4.75%
Total Standard Cell /	-2.54%

Table 5: Partial Scan Instance Area/Count Differences

The two largest reductions in the design were buffer count and flop area. Buffer count indicates that less hold fixing was needed and so less buffers were needed. Though inverter count is slightly higher, the reduction of buffers far surpasses the increase in inverters. The DFF area reduction has a direct relation to the number of DFFs that remained nonscan.

4.2. GPU shader Core results

GPU shader cores are larger than CPU cores and could potentially show a bigger PPA advantage when utilizing partial scan. There can be multiple shader cores in a GPU, multiplying the benefits of partial scan. This GPU shader core contains over 500k DFFs. Initially we used just the Level1 checks and the ATPG results were below

Netlist	Non-scan DFF % w.r.t. total DFFs (%)	Stuck-at ATPG			Transition Delay ATPG		
		Test coverage (%)	Pattern Count	Pattern volume (K)	Test coverage (%)	Pattern count	Pattern volume (K)
Full scan		99.75	11573	11631	97.46	27200	27336
Partial scan w/o overlap threshold/sequential redundancy and problematic DFF checks	29.87	99.01	22396	15946	95.84	54243	38622
Partial scan w/ overlap threshold/sequential redundancy check	19.98	99.54	17194	13910	96.98	45101	36487
Partial scan w/ overlap threshold/sequential redundancy and semi-automatic problematic DFF check	20.36	99.68	15667	12612	97.40	42167	33945
Final partial scan w/ overlap threshold/sequential redundancy and automatic problematic DFF check	20.34	99.65	16077	12959	97.14	37250	30024

Table 6: ATPG Results for GPU Shader Block

expectations. We evaluated the reasons behind this and created the Level2 checks.

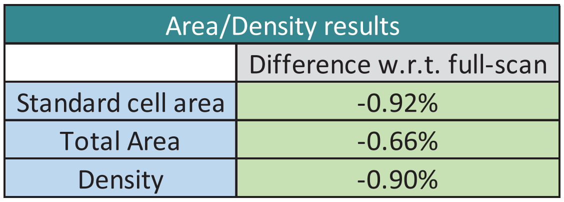
Table 6 shows the GPU shader core ATPG results after adding the Level2 checks. We first performed full scan SAF and TDF ATPG and use these results as the benchmark. The first partial scan netlist (referred to as original partial scan netlist) was generated without overlap threshold/sequential redundancy/ problematic DFF identification checks. The resulting netlist is 30% non-scan. The SAF and TDF test coverage (TC) drop by 0.74% and 1.62% respectively with regard to full scan. Pattern volume increases by 37% and 41% for SAF and TDF ATPG respectively.

Next, we determine the modules that have sequential redundancy and overlap threshold problems and apply overlap threshold and sequential redundancy checks for these modules. This results in a 20% non-scan netlist. We lose 10% non-scan DFFs by adding these checks but we gain 0.53% stuck-at TC and 1.14% transition delay TC compared to the original partial scan netlist. Also, pattern volume drops by 13% and 5.5% for stuck-at and transition delay ATPG respectively compared to the original partial scan netlist. While we lose a third of the non-scan DFFs, we regain two-thirds of the lost stuck-at TC. When compared to full scan, stuck-at TC is 0.21% less and pattern volume increases by 19%. Transition delay TC is 0.48% less and pattern volume is 33% more.

To improve the TC, we now perform problematic DFF identification on top of the existing checks. For the semi-automatic problematic DFF identification, using the partial scan netlist with only overlap threshold/sequential redundancy checks, we gained 0.14% stuck-at TC and 0.42% transition delay TC. With 20% non-scan DFFs, we are within 0.1% of full scan TC for both stuck-at and transition delay. The pattern volume increased by 8% for stuck-at and 24% for transition delay ATPG when compared to the pattern volumes for the full scan netlist.

Next, we worked to reproduce the same results, seen with the semi-automatic problematic DFF identification, with automated problematic DFF identification. The results for both ways of identifying problematic DFFs are very close. When compared to the semi-automatic technique, the automatic technique showed a stuck-at TC loss of 0.03% and a transition delay TC loss of 0.26%. The pattern volume for TDF showed an 11% increase while that for SAF remained almost unchanged.

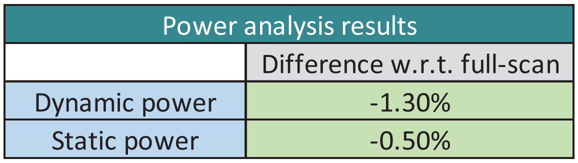
PPA analysis was performed for the final partial scan netlist (using semi-automatic problematic non-scan DFF identification). Table 7 through Table 10 show the results. The numbers shown in these tables are percentage change in partial scan netlist with respect to the full-scan netlist. Standard cell area dropped by 0.92% and this is reflected on the total cell area. There is no change in RAM area which is expected. No work was done to reduce the floorplan, so there is no change in total die area. Physical utilization of the standard cells has dropped by 0.9%. We learned from our standard cell team that due to their limited usage, the non-scan DFF standard cells are not optimized to the same extent as their SDFF counterparts.



Area/Density results	
	Difference w.r.t. full-scan
Standard cell area	-0.92%
Total Area	-0.66%
Density	-0.90%

Table 7: GPU Shader Core Area/Density results

Our vector-less power analysis results showed that leakage power dropped by 0.5% while dynamic power dropped by 1.3%. Dynamic power simulations with vectors were not run on this design.

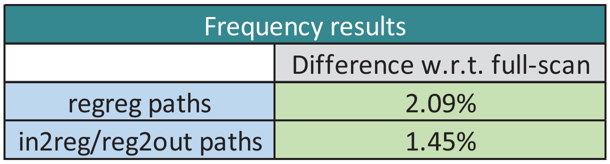


Power analysis results	
	Difference w.r.t. full-scan
Dynamic power	-1.30%
Static power	-0.50%

Table 8: GPU Shader Core Power analysis results

Frequency results are shown in Table 9. Register to register (reg2reg) paths can now be clocked at a frequency that is 2.09% higher with respect to full-scan. Maximum frequency for input to register (in2reg) and register to output (reg2out) paths increased by 1.45%.

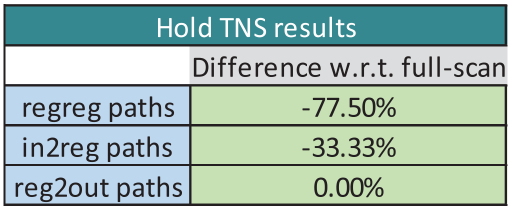
As seen with the CPU core, the biggest benefit of partial scan is reflected in the total negative slack (TNS) results shown in Table 10.



Frequency results	
	Difference w.r.t. full-scan
regreg paths	2.09%
in2reg/reg2out paths	1.45%

Table 9: GPU Shader Core Frequency results

On reg2reg paths, TNS improved by 77.5%. On in2reg paths, TNS improved by 33.33%. There is no change in TNS for reg2out paths.



Hold TNS results	
	Difference w.r.t. full-scan
regreg paths	-77.50%
in2reg paths	-33.33%
reg2out paths	0.00%

Table 10: GPU Shader Core TNS results

5. Conclusion and future work

We presented our partial scan algorithm as a complex amalgamation of several checks. We introduced a new class of faults that manifest themselves during partial scan SAF ATPG and presented a way, not only to determine them, but also eliminate/contain them. We presented two different ways to identify problematic non-scan DFFs.

We showed that different types of logic need different considerations when choosing non-scan DFFs and that there are benefits worth extra exploration in this area.

One way to extend this work would be to analyze the presence of overlap threshold problem in an automated way. We determined overlap threshold manually by debugging the uncovered faults in the ATPG tool. Another area for future work would be to involve test point analysis to determine locations of problematic DFFs. Partial scan benefits should also be explored on a near-threshold design.

Acknowledgements

The authors would like to thank Mark Nathan, Rich Slobodnik, Abhijith Gokuldas, Leo Prakash and Mark Appleton for their help in collecting this data.

References

[1] P. Narayanan et al, "Modified DFF Architecture to Reduce Hold Buffers and Peak Power during Scan Shift Operation," *VLSI Test Symp. (VTS)*, pp. 154 - 159, 2011

[2] K. -T. Cheng, V. D. Agrawal, "A partial scan method for sequential circuits with feedback", *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 544 – 548, 1990

[3] D. H. Lee, S. M. Reddy, "On determining scan DFFs in partial-scan designs", *IEEE International Conference on Computer-Aided Design. Digest of Technical Papers*, pp. 322 – 325, 1990

[4] R. Gupta, R. Gupta, M. A. Breuer, "The Ballast methodology for structured partial scan design", *IEEE Transactions on Computers*, Vol. 39, No. 4, pp. 538 – 544, 1990

[5] V. Chickermane, J. H. Patel, "An optimization based approach to the partial scan design problem", *International Test Conference*, pp. 377 – 386, 1990

[6] V. D. Agrawal et al, "Designing circuits with partial scan*", IEEE Design & Test of Computers*, Vol. 5, No. 2, pp. 8 - 15, 1988

[7] V. Chickermane, J. H. Patel, "A fault oriented partial scan design approach", *IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pp. 400 – 403, 1991

[8] S. T. Chakradhar et al, "An Exact Algorithm for Selecting Partial Scan DFFs", *Design Automation Conference*, pp. 81 – 86, 1994

[9] S. Sharma, M. Hsiao, "Combination of Structural and State Analysis for Partial Scan*", International Conference on VLSI Design*, pp. 134 – 139, 2001

[10] D. Xiang, Y. Xu, H. Fujiwara, "Non-scan Design for Testability for Synchronous Sequential Circuits Based on Conflict Resolution", *IEEE Transactions on Computers*, pp. 1063 – 1075, 2003

[11] X. Lin, I. Pomeranz, S. M. Reddy, "Full scan fault coverage with partial scan", Conference on Design, automation and test in Europe, 1999

[12] M. A. Iyer et al, "Surprises in Sequential Redundancy Identification", *European Design and Test Conference*, pp. 88 - 94, 1996

[13] McLaurin, Frederick, Slobodnik, "The DFT challenges and solutions for the ARM Cortex-A15 microprocessors", *Proc. of IEEE International Test Conference, 2012.*

[14] McLaurin, Kulkarni, "The DFT Challenges and Solutions for the Arm Mali-Mimir GPU", *Proc. of IEEE International Test Conference India, 2017.*

[15] M. Damiani et al, "Synchronous Logic Synthesis: Circuit Specifications and Optimization Algorithms", *IEEE International Symposium on Circuits and Systems*, pp. 2566 - 2570, 1990

[16] K.-T. Cheng., "On removing redundancy in sequential circuits", *Design Automation Conference*, pp. 164 - 169, 1991

[17] M. A. Iyer et al, "Identifying Sequential Redundancies Without Search", *Design Automation Conference*, pp. 457 - 462, 1996

[18] M. Abramovici et al, "On selecting DFFs for partial reset," *International Test Conference,* Baltimore, MD, pp. 1008 - 1012, 1993