

Hierarchical DFT Methodology – A Case Study

Jeff Remmers, Moe Villalba
Plexus Design Solutions, Inc., Sudbury, MA
Richard Fisette
Mentor Graphics Corporation, Waltham, MA

Abstract

A hierarchical approach to DFT is presented to address the issues encountered when inserting DFT into large SOC designs. The case study uses a production design with Sandburst, Inc (0.13μ, 4M gate chip).

1.0 Introduction

In recent designs, the methodology of applying scan on a flat design has become unpractical. Designs are being implemented using a hierarchical approach to address design issues with place and route tools, synthesis tools and timing closure. In addition, when devices move into operations and production, debugging and diagnosing problems using a “flat” design becomes more difficult.

An implementation using the hierarchical methodology is shown using a 4M gate, 0.13μ process design. In the case study with a Sandburst, Inc design, this multi-phase scan methodology exhibits characteristics that address the issues found in the traditional scan techniques.

2.0 Current SOC Design Practices

The tile approach is a technique that is recommended by a number of EDA suppliers of place and route tools. This flow provides a level of relief to the timing closure issues found in current designs. This tile approach has been extended to a hierarchical DFT methodology.

2.1 Current Scan Methodology

It is first necessary to look at the scan methodologies currently being implemented. In many companies, design methodologies are being driven more by the physical design process than any other factor. Specifically, there are two main factors. One is timing closure, which is common in every design using nanometer technologies. Also, the capacity limitations of physical design tools dictate how a large design is partitioned, which in turn drives architectural and coding decisions. It follows that the DFT methodology should be compatible with both the front-end and back-end design processes.

2.2 Physical Design of Current Scan Methodology

Currently, the physical design approach is to lay out the device as several individual blocks that are hard macros (hardmacs), which are then assembled at the chip level. In addition, there is top-level logic that is placed around

the blocks. Figure 1 illustrates an example of twelve hardmac blocks that have been separately laid out and then placed together at the chip level. Also shown is the top level logic distributed around the hardmac blocks. The top level logic often consists of clock generation logic, test control logic and miscellaneous glue logic.

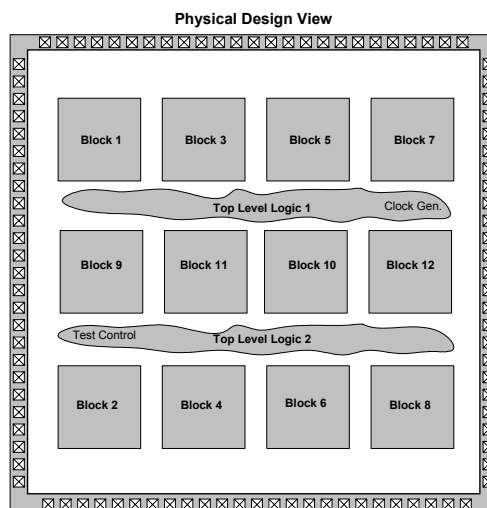


Figure 1 – Physical Design View

2.3 Current Scan Chain Architecture

Scan is inserted at the block level. When the blocks are assembled at the top level, the chains can be connected in one of two ways: concatenated or direct to I/O.

In the concatenated scan chain approach, scan chains from one block are concatenated with chains from another block. In some cases, the chains will traverse multiple blocks before coming out of the chip.

The benefits of this approach are: 1) Automated tools have the ability to assemble block level chains together at the chip level, 2) Assuming that the block level chains are relatively balanced it is easy for tools to balance chip level chains and 3) Having a limited number of pins for scan chains is not a problem.

The disadvantages are: 1) Designs containing multiple clock edges can cause shifting problems when traversing back and forth between positive and negative edge triggered flops as the scan chains traverse through the blocks, 2) Care must be taken to ensure that lock-up

latches are placed properly when crossing between clock domains at the chip level, to avoid timing problems, 3) Since the chains go through multiple blocks, timing problems in a single block can break an entire chain affecting other blocks and 4) Verification of scan chains must wait until the end of the design process, when the entire chip is assembled

With direct to I/O scan chains, the chains inserted at the block level are routed directly to chip I/O, through some muxing logic, without connecting to any other block. In this configuration all block level chains are accessed in parallel for chip level pattern generation.

The pros of this approach are: 1) Clock domains and clock edges can be isolated to certain chains even at the chip level, thereby, minimizing the chances of shifting problems, 2) Blocks are isolated from each other, preventing timing problems in any one block from affecting scan chains in other blocks and 3) It is also easy to identify which blocks have problems, based on which scan chains fail

The disadvantages are: 1) More information about the whole chip is necessary to plan for the correct number of scan chains and to balance the chains and 2) There may not be enough pins available to accommodate the number of blocks.

3.0 Proposed Hierarchical Scan Architecture

An alternative scan implementation to what is currently being done is a more formalized “hierarchical” approach. This proposed implementation describes the design requirements and highlights how they address the specific issues. Hierarchical DFT essentially divides and conquers large SOC designs and isolates blocks from each other. The benefits include: faster runtime, no tool or machine capacity problems, early detection of testability issues, simplified verification, isolation of potential problem blocks, a universally applicable methodology for managing scan chains at the chip level, and simplified debug of scan chain failures on the tester. This methodology is broken down into what is to be done at the block level and then at the chip level.

3.1 Block Level

The “block level” is the level at which the physical design defines the block. Effective block level DFT requires that the blocks be well isolated from the rest of the chip. The goal is to have all block level DFT issues addressed before chip integration.

3.2 Registered I/O ports

The most important requirement is that all I/O ports of the block be registered. For DFT purposes, this serves to isolate the block when these I/O registers are scanned.

There are more formal DFT implementations, specifically IEEE P1500 that insert a boundary scan-like collar around

the block. Registration of I/O ports in a block is a common design technique used to better control inter-block timing paths (and alleviate the timing closure problems). It is also a common technique for design groups sharing cores or “IP” blocks.

3.3 Partition Scan Chain

The I/O registers described above can be hooked together into a single chain called a partition scan chain. Since these registers are already in the design, the DFT process takes advantage of them by using it as the isolating “collar” for that block. This isolation incurs no additional area overhead because the registers are there by design and not added for DFT purposes. The rest of the core logic in the block is scanned and assigned to other scan chains making sure that partition chains only contain the I/O registers.

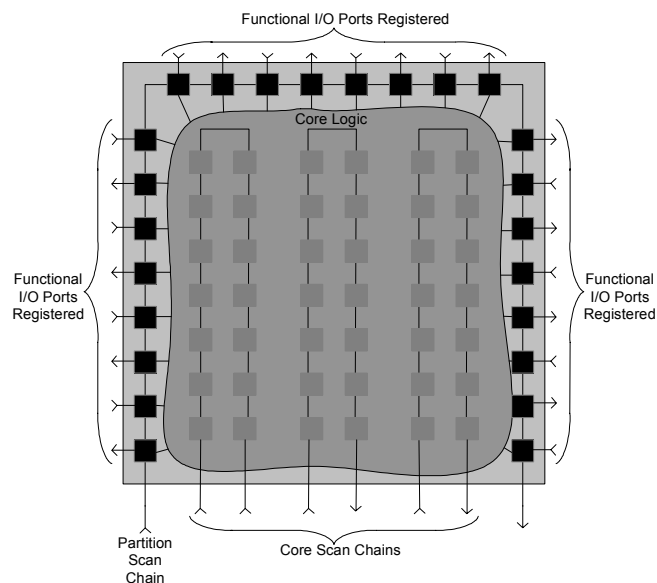


Figure 2 – Block Level Scan Insertion

Figure 2 shows a single partition chain, however multiple chains that contain the I/O registers can be used, as long as only port I/O registers are in those chains. Like all scan chains, they can also have multiple clock domains and clock edges. Scan chain re-ordering, if necessary, is also handled like any other chain.

3.4 Skeleton Netlist

After scan insertion has been completed, a “skeleton” representation of the block is saved out for chip level integration activities. This skeleton netlist will primarily contain the partition scan chain and some numbers of representative scan flops for each of the other core scan chains. The number of clock domains used on those chains determines the number of flops for each chain. This is done so the clock connectivity to each block can be verified at the chip level. When the chip level netlist is integrated, inclusion of the skeleton blocks will result in a netlist that can be processed very quickly by ATPG DRCs. These standard ATPG DRCs will be used to verify that the clocks and scan chains are connected correctly at the top level. It uses the DRC capability of ATPG tools to quickly check for chip level interconnects of control signals (and chains) to the blocks. Chip level ATPG that targets only the interconnect logic between blocks need only use the skeleton netlists resulting in very fast runtime

3.5 Pattern Generation

For the purposes of identifying test coverage issues early in the design schedule, pattern generation should be done at the block level. Functional inputs will be constrained to unknown state and functional outputs will be masked. Only the scan chain inputs and outputs are used for pattern generation.

The resulting test coverage will directly reflect the test coverage that can be obtained for that block, once it is integrated at the chip level, since the chip level access to the block will be through the scan chains. Any test coverage issues will be quickly identified and can be more easily addressed at the block level. The only faults not detected at the block level will be at the I/O ports to the block. This will result in a minimal loss in coverage, because these ports are registered hence only the “D” pin of the input registers and Q pin of output registers are missed. It also guarantees that signals from other blocks will have no impact on the controllability and observability of the faults in this block regardless of how the full chip is integrated. I/O fault detection will be addressed during chip level pattern generation.

Some blocks are inherently difficult to test because they are challenging to ATPG algorithms. These blocks typically achieve high test coverage but they require a large number of patterns, and long tool runtimes. Block level pattern generation identifies these types of blocks.

4.0 Verification

Block level verification allows designers to identify DFT problems early in the design cycle. It also addresses tool capacity and runtime issues because a block is significantly easier to process than the whole chip. Once the layout of the block level netlist is complete,

verification can begin. Both Static Timing Analysis (STA) and gate level simulations, with timing information back-annotated, can be run quickly on the relatively small netlist. This can occur well in advance of the full chip layout. Verification will catch scan shift issues in the block and scan capture issues for blocks with a single clock domain. Chip level timing verification must still be done to account for insertion delay in the chip level clock trees, but if there are timing problems at the block Level they will be caught and addressed early. This is consistent with the goal of achieving DFT closure in step with functional closure.

5.0 Chip Level

The chip level implementation takes advantage of the fact that the block level DFT isolates the block, rendering it portable and immune to changes outside the block. The first chip level issue to address is how to bring the block level chains to the chip I/O.

6.0 Multi-phase Scan Chain Merging

Multi-phase scan chain method addresses the issue of chip level scan chains. Similar to the direct to I/O approach described above, multi-phase scan chain merging brings block level scan chains directly to the chip level pins through muxing logic. The difference with this approach is that there are several mux configurations used in scan mode. Each mux configuration, or phase, brings out a different group of block level scan chains. Figure 3 shows the physical design example shown in Figure 1 in which hierarchical DFT, using the partition scan chain, was applied at the block level. I/O muxing was added to access the scan chains and was controlled by input signals Scan_phase_sel(2:0). In the following example, where each block has four scan chains, two blocks can be accessed per phase at the chip level. A phase is defined by how the mux select signals, Scan_phase_sel(2:0), are set.

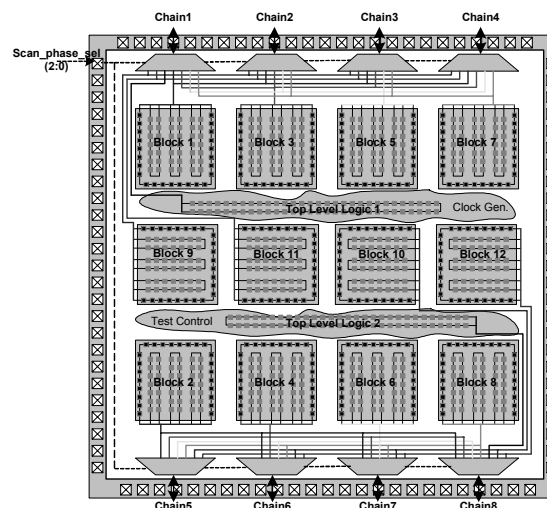


Figure 3 – Multi-phase Scan Chain Merging

Phase #0 is defined by the values on the Scan_phase_sel(2:0) signals set to 000. It brings the four scan chains from Block 1 to chip level scan chains 1-4 and the Block 2 chains to chip level scan chains 5-8. With this configuration, scan patterns are applied to those two blocks only. In Phase 1, Scan_phase_sel is set to 001, which in turn accesses Blocks 3 and 4. This process continues on until the final phase, Phase 6, Scan_phase_sel = 110, in which top level logic and interconnects between the blocks are targeted. In this phase the partition scan chains for all the blocks and the chains for the top level logic are brought out together.

This approach provides the same block isolation of the direct to I/O scheme. At the same time it can accommodate any number of scan chains like the Concatenation method without the potential timing problems associated with traversing multiple blocks. An additional benefit is that scan chain balancing is made simpler. Rather than trying to balance all the chains for the entire chip, it is only necessary to balance the chains for a given phase.

7.0 Pattern Generation

Pattern regeneration at the chip level demonstrates how the multi-phase scan chain method combined with hierarchical DFT and the skeleton netlist addresses tool runtime, debug/diagnose tester problems and recover from tester problems. Typically scan patterns must be run at the chip level once the entire design has been assembled. For very large designs, this can present difficulties for ATPG tool capacities and runtimes. This becomes particularly problematic in the late stages of layout when ECO's can frequently occur and ATPG must be re-started many times.

With the multi-phase scan chain approach, the chip level ATPG process can be dramatically simplified. Only the blocks that are accessed for a given phase need be present in the netlist along with the collection of top level logic that contains clock generation and test control logic. This is just a variation on the skeleton netlist concept, whereby only the logic you are targeting need be present.

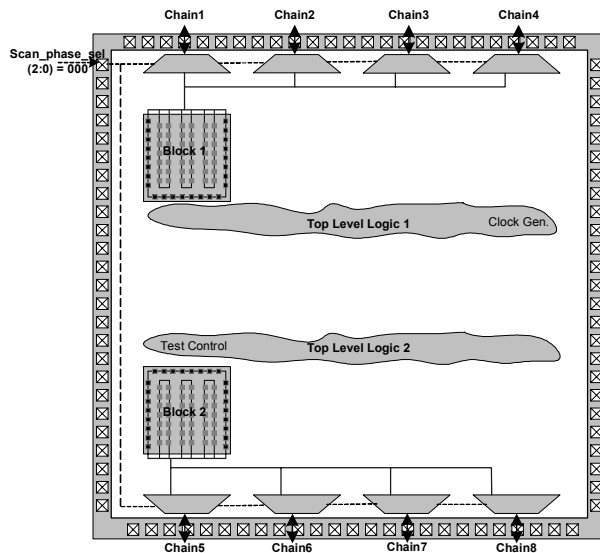


Figure 4 – Phase0

Figure 4 shows that for Phase 0, the only logic necessary is Block1, Block2, top level logic, I/O pads and multi-phase muxing. With this configuration (Scan_phase_sel = 000), ATPG can be run on a netlist that uses much less memory and because it targets fewer faults, results in a much shorter runtime. Patterns are saved out as a separate scan file just for Phase 0, and are run on the tester that way.

Each phase will progress in this same manner. The one different phase will be the last one, which addresses top level logic and interconnects between the blocks.

Figure 5 shows that in Phase 6 (Scan_phase_sel = 110), all the top level logic that is scanned and the interconnect logic between the blocks are targeted. The skeleton netlists for the blocks are used, because only the partition scan chains are accessed in this phase.

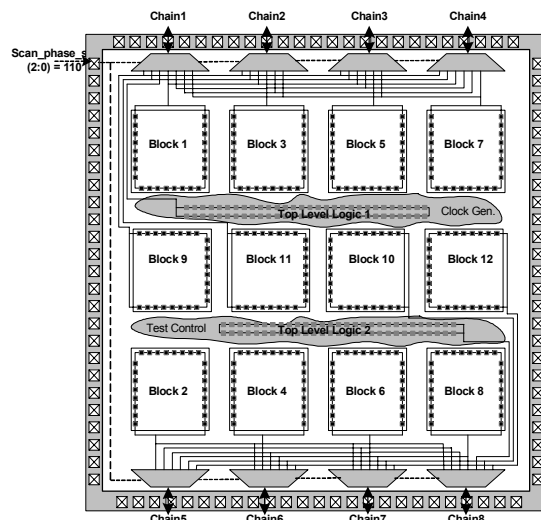


Figure 5 – Scan Phase 6

The impact of Engineering Change Orders (ECO's) and block re-spins is reduced using this approach. If netlist changes occur in only a single block, scan patterns need only be regenerated for the phase containing that block. None of the other patterns need to be regenerated as a result of the change.

ATPG runtime can be improved over and above what is gained by the processing of only a few blocks at a time. Since each phase can exist independently, ATPG can be performed for multiple phases simultaneously.

8.0 Determining Chip Level Test Coverage

Because the phase level netlists do not include all chip logic as well as logic that isn't targeted for fault detection, the test coverage results for a given phase is incomplete. It is necessary to fault grade the patterns from each phase against the full chip netlist. When pattern generation is complete for each phase, a fault list containing detected and undetected faults must be saved. An additional run of the ATPG tool is invoked on the full netlist and the fault lists from each phase are loaded. Once a fault is determined to be detected, it cannot be overwritten as undetected by a fault list from subsequent phases. In this manner, the ATPG tool will tally up the list of detected faults as each phase is loaded and give final chip level test coverage.

9.0 Verifying Chip Level DFT Logic

block level DFT logic is verified at the chip level to ensure that scan chains are enabled, all clocks reach the scan flops, and there are no scan chain blockages. This is simple to do because there is complete access to all the needed pins. The process is done by the ATPG tool DRCs that trace the scan chains before pattern generation. When the blocks are combined at the chip level, access to the block level DFT pins is no longer direct but must go through I/O pads, muxing logic, clock/reset generation logic and other glue logic. With the chip level netlist configured for scan, the ATPG tool DRCs trace scan paths through the top level logic down to the block level chains.

Even though large netlists can take a long time (up to hours) to run DRCs, a chip level skeleton netlist will dramatically reduce runtime for DRCs. It is created by combining the block level skeleton netlists with the top level logic of the chip.

All the logic that is outside the blocks is kept in the chip level skeleton netlist. This includes the clock generation logic and any test control logic. Although for each block, only the skeleton netlist for that block is instantiated. It is not necessary to verify all the flops in the block level chains (that was already done at the block level), but to verify that the chains can be reached through all the chip level logic. That is why each block level chain has only

one flop per clock domain. Scan chain tracing rules will verify that the chains can be accessed and the correct clocks are enabled. A process that takes hours to run on a full netlist is reduced to just a few minutes. This reduced runtime makes it feasible to use ATPG DRCs for verification.

10.0 Case Study

The focus of this case study is on the multi-phase scan methodology developed by Plexus Design Solutions, Inc., which was implemented during the design of the Petronius chip for Sandburst, Inc. The Petronius design is a 4M gates, 0.13µ TSMC process, having a core frequency of 250 MHz. Sandburst, Inc. is attuned to the importance of DFT and has made a significant effort to achieve the highest possible test quality. There are a number of DFT features designed into the Petronius chip, such as Full Scan, Boundary Scan, Memory BIST, Functional Test and Functional Binning. The following is a list of the DFT features and requirements for Petronius:

Full scan - Petronius is a nearly "full scan" design using Mux-D flops. With the exception of a small number of registers in the clock generation logic, all the 270K flops were scanned.

Stuck-at - The target coverage for stuck-at faults was 99+%.

Transition Faults- Because 0.13µ processes have exhibited an increased number of at-speed related defects, it was deemed necessary to target transition faults. While the core frequency of 250 MHz is achievable by available testers, at-speed clocks were generated using the on-chip PLL (include references). This was done in order to prove the concept for future planned designs that will require frequencies beyond the tester's capability. The goal for transition coverage was not established but depended on how much memory was available on the tester after running all the other tests. Stuck-at fault testing is the focus of this paper.

Path Delay - Initial plans did not include targeting Path Delay faults. It was decided that path delay patterns could be added if the defect level warranted it.

Boundary Scan - Petronius has Boundary Scan on all the I/Os except for the SERDES interfaces.

Memory BIST - Memory BIST was used on all the Artisan memories. There was not enough total memory in the chip to necessitate Repair capability.

Functional - There were additional "functional" tests used to target the SERDES blocks, the PLL and clock generation logic.

Functional Binning - Functional Binning refers to the ability to test specific parts of the design to determine

that, if there is a failure, whether or not it can be salvaged as a partially functioning part. Petronius has two distinct modes of operation. If one of those modes does not work, the part can still be used for the other mode provided it can test for that mode separately. This effectively increases yield.

10.1 Chip Level Scan Chain Planning

There were enough pins to accommodate 36 scan chains at the chip level. This was one of the main factors in determining how the multi-phase mux block was created to accommodate all the hardmac scan chains. Since functional tester memory was used, there were no tester limitations to the number of chains allowed. The goal was to use as many scan chains as possible, to minimize chain length. The limiting factor was that quite a few of the I/Os were not available for scan, because of the high speed SERDES macros.

10.2 Hardmac Scan Chains

There were a few factors that made Petronius a good candidate for implementing the multi-phase scan methodology. First, the functional blocks were designed from front-end through physical design as hard macros to be pieced together at the chip level; secondly, for reasons of timing closure, all I/O ports of the hardmacs were registered; and thirdly, some hardmacs were instantiated multiple times. Table 1 summarizes the hardmac information related to scan.

Hardmac Name	# Hardmac Instances	# Flops/ Hardmac
pt_pi	1	2359
pt_xb	1	27764
pt_if	40	2631
pt_bw	1	14690
pt_ina	32	2702
pt_nm	1	24194
pt_fo	1	1177
pt_ac	1	457
serdes	10	446
Top Level Logic	1	3827
Chip Total	88	270632

Table 1 – Hardmac Data

project. To start, the number of chains per hardmac was driven by balancing the length of the chains. The length of the chains was determined by the length of the partition chain. The scan insertion tool reported on how many

flops made up the partition scan chain for each block. Table 2 summarizes the partition scan chain information and determined how many more chains were needed to maintain fairly balanced scan chains for each block. No attempt was made to balance the chains of one block to another block.

Block	# Flops/ Block	# Flops in Partition Chain	Total # Chains (includes Partition Chain)	Length of Non-Partition Chains
pt_pi	2359	788	3	786
pt_xb	27764	2187	14	1968
pt_if	2631	1146	2	1485
pt_bw	14690	1075	14	1048
pt_ina	2702	367	9	292
pt_nm	24194	791	28	867
pt_fo	1177	190	3	494
pt_ac	457	240	2	217
serdes	446	0	1	446
Top Level	3827	0	2	1914

Table 2 – Partition Chain Data

Looking at each block in isolation, it is evident that the chains are not perfectly balanced. The limiting factor here is that the flops on the partition scan chain must be kept separate from any other chain. Even with multiple partition scan chains per block, this requirement can make it difficult if not impossible to match the chain lengths without creating an unreasonable number of chains. It was decided that giving up some efficiency in scan chain balancing would be acceptable. Once the scan chains were inserted in the block netlist, the scan version of the block was sent on to layout, and a “skeleton” netlist was written out by the scan insertion tool. This “skeleton” netlist was used for chip level DFT verification and pattern generation in the final phase.

10.3 Scan Phase Planning

In determining the number of phases required to access all the block level chains, there are some guidelines that were established. First, a block had to be fully contained within a phase; and secondly, there could be multiple blocks per phase, but effort had to be made to balance the chains at the chip level, as best as possible. One other less formal consideration was to minimize the number of phases. At the scan phase planning point in the project it was not known how many gates were needed to implement the mux block or, more importantly, how many interconnect wires were required. The concern was that this block could have a significant impact on routability and timing closure. With these factors in

mind, several configurations were considered that grouped multiple blocks in various phases to try to balance chains for a given phase, while using all 36 chip level scan chains. This exercise resulted in changing some block level scan chain configurations. The final assignment of blocks to phases was put into a spreadsheet format.

10.4 Generating Scan Mux Block

Sandburst created a perl script to read the information in the scan phase planning spreadsheet and generate verilog RTL for the scan mux block. This scan mux block granted access from the block level chains indicated in each phase to the 36 chip level chains. In the case of the final phase (Phase 12), the chains for the top level logic, the chains in the SERDES blocks and the partition scan chains of all the blocks were accessed in a single phase. This required that some of the chains be concatenated in the scan mux block. To avoid scan shift timing problems, lock-up latches were inserted at the end of each chain during block level scan insertion. The resulting scan mux block contained about 2K gates and 1,116 ports. This block was absorbed in to the top level logic block.

10.5 Block Level ATPG Results

During the scan phase planning process, work continued on each of the blocks. Table 3 shows the stuck-at test coverage for each of the blocks and the number of required scan patterns.

Block	Test Coverage	# Scan Patterns
pt_pi	96.69%	209
pt_xb	99.43%	338
pt_if	96.96%	251
pt_bw	98.90%	564
pt_ina	99.53%	2026
pt_nm	99.83%	2472
pt_fo	98.65%	196
pt_ac	95.02%	101

Table 3 – Block Level Stuck-At Results

This exercise detected test coverage problems early in the design process, when they were easier to rectify. Some of the block level test coverage may look relatively low (i.e. less than 99%). This is because these results were achieved by using scan chain access to the blocks only. The functional I/Os of the block were not used. By using scan chain access only, we simulated the same access we would have had to this block, once it was integrated into the chip, meaning the test coverage achieved at block level is the same as at chip level. The partition scan chain of each block guaranteed that there were no controllability or observability problems due to logic outside the block. The I/O faults will be detected when running pattern

generation in the final phase targeting top level faults. Note that the size of the block doesn't necessarily dictate the number of pattern (e.g. pt_ina is one of the smallest blocks but also has one of the highest pattern counts).

10.6 Complex Blocks

Looking at the results in Table 3, you can see that test coverage is reasonably high for each block. Another item that stands out is the difference in the number of patterns required to test each block. The pt_nm and pt_ina blocks in particular (and pt_bw to a lesser extent) require significantly more patterns than the other blocks. These complex blocks are inherently more challenging to ATPG algorithms and result in a higher number of scan patterns. Test coverage is still high, it just requires many patterns to achieve the maximum coverage. Once a complex block is introduced into a full chip netlist, it affects ATPG performance for the entire design, despite the fact that there may be many other simple blocks. What we discovered was that identifying the complex blocks had an impact on the scan phasing that was planned. To minimize the effect of the complex blocks, we created as many chains with the shortest length as possible. The goal was to reduce the tester memory requirement and tester time. The complex blocks were isolated to their own phases, so the chains would be as short as possible and other blocks would not be affected. This realization came relatively late in the project, however. When the scan phasing was re-examined from the perspective of complex blocks, it was determined that the pt_ina blocks were reasonably well isolated and had short chains to begin with. The effort to replan the pt_nm block was determined to be more than was necessary, at that stage of the project.

10.7 Verifying Chip Level DFT Logic

RTL level test benches were not created to verify the scan mux block and other chip level connections. Instead, the DRCs contained in the ATPG tool (specifically scan chain tracing rules) were used to verify that block level chains were correctly wired to the chip level pins. This was done by building a netlist for each phase that included only the logic required for testing that phase. This typically included all the top level logic plus whatever blocks were targeted for that phase. Rather than instantiating the entire block netlist, the "skeleton" netlist for that block was used. The resulting phase level netlist was not much larger (in gate count) than just the top level logic, and it could be processed very quickly by the ATPG tool. This allowed for quick identification of problems connecting the blocks to the scan mux logic.

10.8 Building Phase Level Netlists

Phase level netlists were created by using verilog include statements to incorporate just the logic blocks needed for that phase.

The pt.v contained pads and top level logic and instantiated all the hardmacs, but did not have the module definitions for each hardmac. The pt_if.v contained the module definition of that hardmac. When the ATPG tool read in the phase0.v netlist, any block that did not have a module definition would be automatically “black-boxed” by the tool. Because the partition scan chain isolated the targeted block from other blocks, the Xs generated by the black-boxes had no impact on test controllability or observability. A further refinement to this netlist building process was to black-box instances of a block that were not targeted in a certain phase. For instance, only 18 instances of pt_if were tested in Phase0, another 18 in Phase1 and four others in Phase2. Including the pt_if.v netlist means the ATPG tool would see all 40 instances of the pt_if even though only 18, at most, would be targeted. Modifying the pt.v netlist to rename the pt_if instances not being targeted for that phase would allow the ATPG tool to black-box those as well. In all there were 13 of these phase level netlists. The Phase12 netlist was originally going to contain just the top level logic and the skeleton netlist of each block, since only the partition scan chains of each block was needed for this step. Because of tool problems in generating the skeleton netlist, based on the post-layout, scan-reordered netlist, Phase12.v ended up including all the logic of the entire chip. As the project progressed, there were many changes made to each of the blocks. To keep up with the latest changes, the phase level netlists had to be constantly updated to point to the latest releases of the block.

10.9 Chip level ATPG

Even though block level ATPG results can be used to determine the block level test coverage and a reasonable indication of the number of patterns required, ATPG still had to be run from the chip level pins. Once the scripts were generated and working for one of the phases, it was a simple matter of modifying the scripts to change the Phase Select bits for each phase. Because each phase can operate autonomously from any other phase, it was possible to run ATPG for multiple phases simultaneously. Sandburst had two ATPG licenses, so it was possible to batch up all the phases and run two at a time, thus reducing the net ATPG runtime significantly.

In examining the chip level ATPG results, it can be calculated how much tester memory will be required to run the patterns. In this case the tester memory was calculated for each phase and all the phases were added up. For each phase, the required tester memory was defined by the number of scan patterns multiplied by the length of the longest chain. The tester memory total was 15.56Meg.

10.10 Chip Level Fault Grade

Since the entire netlist is not present during the pattern generation of each phase, the test coverage indicated for that phase is not a true indication of the overall test coverage. To determine the true test coverage of the chip, a fault list, including which faults have been detected, must first be written out for each phase. By invoking the ATPG tool on the full netlist once more, each fault list can be loaded into the tool using a “-protect” switch. This allows the tool to keep a running tally of faults detected for each phase. Once a fault is detected in one phase it cannot be overwritten as undetected when loading subsequent phases. By contrast, faults that are undetected in one phase will be overwritten, as detected by a subsequent phase. Table 4 shows how the test coverage increments higher for each phase and the final total test coverage for the chip:

Phase #	Cumulative Fault Grade	Phase #	Cumulative Fault Grade
0	12.23%	7	67.60%
1	24.45%	8	72.62%
2	36.44%	9	77.65%
2a	42.45%	10	82.68%
3	47.48%	11	83.37%
4	52.51%	11a	96.23%
5	57.54%	12	98.89%
6	62.57%		

Table 4 – Phase Level Fault Coverage

10.11 Functional Binning

The Petronius design is well segmented into two functions; the crossbar (fully functional device) and arbiter, which only requires that a subset of blocks be functional. The test patterns were ordered so that the subsets of blocks need for the arbiter functionality were tested first. If these patterns run cleanly then the device was categorized as a good arbiter part. If, when running the rest of the patterns a failure is encountered, the part can be binned as a good arbiter device rather than thrown away in the fail bin. This practice can effectively increase the yield.

10.12 Physical Design

During layout there were congestion problems in the Top level logic. A number of measures were taken to relieve the congestion. The scan mux block was reconfigured so that blocks were grouped within a phase based on physical placement. Specifically Phases 0-2 contain 40 pt_if blocks and Phases 3-10 contain 32 pt_ina blocks. The two top level scan chains were also reordered. These

actions relieved some of the congestion. The most significant contributors to congestion were determined to be large busses related to the processor interface and memory BIST. Once these were addressed, the design was routable. The lesson learned was the importance of considering physical design in the planning of the scan phases.

10.13 Static Timing Analysis

STA was performed on the entire design for both scan shift and scan capture modes of operation. This activity was rolled into the functional STA tasks and, while an important verification step, did not really benefit from this DFT methodology.

10.14 Gate Level Simulation

Gate Level Simulation was run much like ATPG. It was done in phases on partial netlists that included only logic necessary to control the DFT logic and represent the targeted blocks. However, additional steps were taken in order to “black-box” the non targeted blocks. Since the simulator cannot auto black-box missing modules (as the ATPG tool can), an empty module definition must be created to represent the non-targeted block. SDF back annotation was again done just for the included logic for that phase. These simulations benefited greatly from this methodology. Not all the runtime data has yet been compiled, but all the patterns, parallel and select serial for all the phases, can be simulated in less than a day. In previous designs at Sandburst, simulations extended beyond one week. Not only are the smaller netlists inherently faster, but the phasing provides the advantage of multiple simulation licenses and multiple machines to run phases in parallel. This has been a huge factor in keeping DFT verification out of the critical path to tape-out.

10.15 Tester Activities

This device was up and running on the tester in hours for the stuck-at patterns and days for the transition patterns.

10.16 Comparison to Traditional Scan Methodology

To determine some of the benefits of this methodology, it was necessary to run ATPG on the full chip, bringing all the scan chains out in a single phase, as is done with traditional scan techniques. Work in this area is preliminary at the time of this writing. Initial experimental results indicate that running ATPG on the full design required about 3600 scan patterns. The ideal chain length would be 8200, which means the required tester memory would be 29.52M, at minimum. The limit on tester memory for this design is 28M/per pin for all test vectors, so it would have been necessary to truncate scan patterns in order to run scan and all the other tests. Compare this to the 15.5M required using the multi-phase scan methodology, which leaves room for the JTAG, memory BIST and functional patterns, as well as some

transition patterns. With the data available at this time in the project, the following can be said about the hierarchical methodology versus a traditional scan methodology:

Design Flow Compatibility - Front-end and back-end designs are done in a hierarchical manner. When a block is “completed” from a functional perspective, DFT for that block can also be completed at the same time.

Early Detection of DFT Problems - block level results are clear indicators of what can be expected at the chip level.

DFT Tool Runtime - Working on smaller pieces of the design results in shorter tool runtimes and lower compute resources (smaller machines). We could take full advantage of multiple ATPG licenses by running multiple phases in parallel.

Less Tester Memory Required - Initial results indicate there is over a 2X reduction in tester memory required (Table 5).

	Pattern Volume (per pin)
Hierarchical Flow	12,883,507
Flat Design Flow	31,349,299

Table 5 – Pattern Volume Comparison

ECO Tolerant - ECO’s late in the project have less impact since only the affected phases require pattern regeneration.

Simulation Runtime - As with DFT tools, the simulation environment runs much more quickly when dealing with smaller pieces of the design.

Fast Recovery From General Problems - A variety of problems were encountered when working through the details of tapeout. Minor fixes to timeplates and masking of certain scan cells can be quickly turned around for any or all of the phases.

Reduced Machine Requirements - Because of the reduced netlist size of each of the phases, most of the ATPG and simulation can be run on relatively small, 32 bit Linux machines. Only the final phase required a 64 bit machine.

There was a cost associated with these benefits such as additional front-end design tasks to direct the block level scan insertion and implement the scan muxing at the chip level. Also additional gates and interconnects resulting from the scan mux block impacted top level routing and timing closure.

11.0 What Worked

By staying compatible with the block level design approach, testability problems for blocks were identified

and addressed early on in the project. The partition scan chain, as expected, provided isolation of the blocks from other blocks. This is what made it possible to use reduced netlists for each phase, resulting in large improvements in ATPG and simulation runtimes.

11.1 What Did Not Work

The skeleton netlists for the blocks were not fully used as planned. STA verification was an area that did not benefit from this methodology. It is necessary to run STA on the full netlist with SDF back annotation to take into account all the crosstalk, coupling effects and clock insertion delays.

11.2 What Should Be Changed/Modified

The scan phase planning needs to be more automated. Factors, such as which blocks are complex blocks and what is the physical placement of blocks needs to be part of the planning process. Improvements in the generation of the skeleton netlist will make that a more useful tool during pattern generation for the final phase.

12.0 Conclusion

There were challenges in implementing this methodology but the effort paid off in a number of ways. The real motivation for implementation of a hierarchical DFT is to align with front-end and physical design practices. This was achieved on Petronius and many additional benefits were also achieved. These benefits include reduced runtime of tools (Pattern Generation - ATPG and Pattern Verification - simulation) and reduced scan pattern size. The phase planning was a learning process as design details such as complex blocks blocks and physical placement became prominent factors. These lessons will be applied on future designs. The front-end design effort required was deemed well worth it as savings were realized in ATPG effort and verification effort. As a result of the additional tester memory available using this technique, more testing was implemented. The final test quality was significantly higher.

Special thanks to Tony Martuscello, Leo Keegan and the team at Sandburst for their help in this design.

References

1. Alfred L. Crouch, *“Design for Test for Digital IC’s and Embedded Core Systems”*.
2. Sengupta, Kundu, Chakravarty, Parvathala, Galivanche, Kosonocky, Rodgers, Mak, *“Defect-Based Test: A Key Enabler for Successful Migration to Structural Test”*, Intel Technology Journal Q1’99.
3. Ron Press, Janusz Rajski, *“Heading off test problems posed by SoC”*, EE Times, October 16, 2000.
4. Hari Balachandran, Kenneth M Butler, Neil Simpson, *“Facilitating Rapid First Silicon Debug”*, International Test Conference 2002.
5. Bailey, Metayer, Svrcek, Tendolkar, Wolf, Fiene, Alexander, Woltenberg, Raina, *“Test Methodology for Motorola’s High Performance e500 Core Based on PowerPC Instruction Set Architecture”*, International Test Conference 2002.
6. Wilson, Ron, *“DFT Takes on Test Cost in Final Combat”*, Integrated System Design, October 2001
7. Sematech *“Overview of Quality and Reliability Issues in the National Technology Roadmap for Semiconductors”* 98013448A-TR
8. Sabada, Sudhakar *“Keynote: Nanometer Design Challenges”* Magma Nanometer Design Seminar October 2002
9. Rick Fisette, Gary Gebenlian, Jeff Remmers *“Efficient DFT Verification Methodology Using Skeleton Netlist and ATPG tool, a Case Study”* International Test Synthesis Workshop 2003
10. Vikram Iyengar y , Krishnendu Chakrabarty and Erik Jan Marinissen *“Test Wrapper and Test Access Mechanism Co-Optimization for System-On-Chip”*, International Test Conference 2001
11. Eric Larsson, Zebo Peng *“A Reconfigurable Power-Conscious Core Wrapper and its Application to SOC Test Scheduling”*, International Test Conference 2003
12. Sandeep Kumar Goel, Erik Jan Marinissen *“Effective and Efficient Test Architecture Design for SOCs”*, International Test Conference 2002
13. Rainer Dorsch, Ramón Huerta Rivera, Hans-Joachim Wunderlich, Martin Fischer *“Adapting an SoC to ATE Concurrent Test Capabilities”*, International Test Conference 2002