

Can commercial tools handle DFT of digital SoCs?

Kim Petersén

*Hardware Design Center
SE-723 50 Vasteras Sweden
Email: kim.petersen@hdc.se*

Ove Andersson

*Ericsson Radio Systems
SE-164 80 Kista Sweden
Email: ove.andersson@era.ericsson.se*

key words: Logic BIST, Memory BIST, SoC, IP.

ABSTRACT

This paper presents the experiences gained from insertion and verification of DFT-structures into two digital state of the art mega gate deep submicron ASIC designs. The tools used are a suite of commercial DFT tools. The different types of tests inserted are scan, boundary scan, memory BIST and logic BIST. Both hard and soft IP modules have been used together with user created logic. The soft IP had a RAM with a pre-created memory BIST, not created with the commercial tool.

1. INTRODUCTION

Shorter time to market of as complex ASICs as SoC (system on a chip) requires efficient methods and tools to carry out design for test (DFT).

The commercial suit of tools is designed to implement and verify the different types of physical structures needed to carry out an efficient production test of digital deep submicron designs as SoC. The tools can also create the corresponding test vector files required by a tester in the production.

The tools have been used, by the authors of this paper, to implement and verify DFT at two digital deep submicron ASICs of type SoCs. The experiences gained are included in this paper.

From user point of view: One suit of tools always comprises the three main parts:

- The tools.
- The documentation of the tools.
- The support of the tools.

It is the total quality of all these three parts that together defines how efficient it is to use the suit of tools.

The project was started with using version of the tools available in 1999 and ended with versions that were updated with bug fixes.

The workstations used to run DFT at was a SUN Ultra 60 with 2Gbyte or RAM and a SUN Ultra 80 with 4Gbyte of RAM. The Ultra 60 had one CPU running at 360 MHz and the Ultra 80 had two CPUs running at 450 MHz.

2. COMPLEXITY OF THE DESIGNS

One of the two ASICs consists of the main parts as:

- Seven identical DSP IPs, Each with an embedded SRAM
- One ARM IP, with an embedded ROM.
- 31 embedded SRAMs
- Two PLLs
- User created logic (UL)

The other ASIC consists of the parts as:

- Eight identical DSP IPs, of similar type as in the first ASIC.
- 33 embedded SRAMs
- User created logic

The total size of each individual ASIC is approximately 2 M gates plus embedded memories. Both ASICs uses three different clock frequencies, in the range of 100 to 200 MHz.

The number of functional IOs is appr. 300 for each ASIC.

From production test point of view: The physical area of both the ASICs can be described as consisting of embedded memories with surrounding logic.

3. STRUCTURE OF THE DESIGNS

Structure of the ASICs is as shown in figure 1.

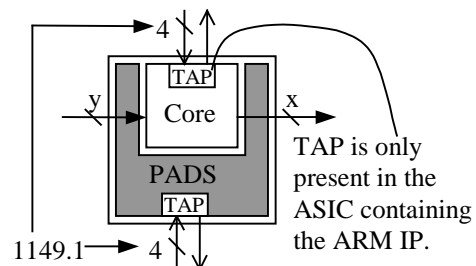


Figure 1. Structure of the ASICs

Core contains all the functional logic.

The PADS module contains the boundary scan structure, the logic BIST controller, and associated embedded test logic. See ref. 2 for further information.

Most of the IOs are connected to boundary scan cells. Non boundary scan IOs are only the TAP connections and those IOs that are extremely time critical.

From now on, the experiences gained from both the ASICs will be presented by only using the ASIC with the embedded ARM IP in the illustrations.

4. THE DFT WORKFLOW USED

The DFT structures are inserted and verified as a bottom-up sequence divided into three main steps, as shown in figure 2. First the operations are carried out at the Block level followed by the Core level and finally the Top level.

The staff in the project was divided into three different teams, as shown in figure 2. One team (team1) delivered the IP called ARM as one block with surrounding glue logic and support functions, the output was a synthesised netlist with all DFT structures inserted and verified. Another team (team2) tailored the soft and configurable IP module called DSP into a ready to use block, the output was a synthesised and routed netlist with all DFT structures inserted and verified. The third and last team (team3) inserted and verified DFT for the user defined ULs and the remaining parts of the SoC.

The internal logic in both the DSP and ARM IPs has critical timing requirements, handled by using separate teams in the project.

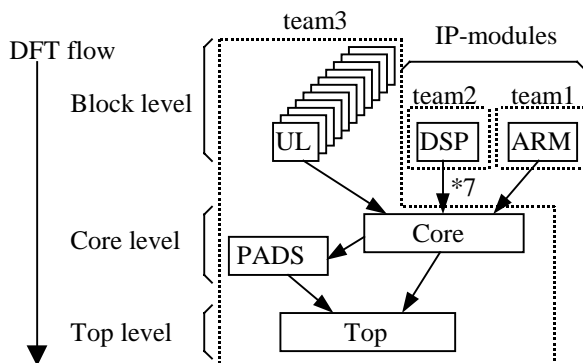


Figure 2. The three main levels used in the DFT flow.

It is important to build together the entire design from Block level up to Top level as fast as possible in a project, since it requires a lot of work (read time) to do this the first time. Examples of reasons why it takes a long time are:

- It is a lot of work to create the different scripts needed to automate all steps in the DFT flow.
- Detect all the bugs and limitations in the DFT tools and find out workarounds to handle the bugs.
- Pipe clean the total flow of tools used.

Creation and verification of all scripts needed to automate the different DFT insertion and verification operations were more time consuming task than foreseen. The total number of script files written for the UL-blocks at the Block-, Core- and Top-level were 126 (224 A4 pages),

several of the files were common for all the different UL-blocks.

It is fairly easy (and fast) to rerun all operations as many times as needed, as soon as the design once has been built up from Block-level up to Top-level.

5. DFT AT BLOCK LEVEL

One of the goals in the project was to try to insert the DFT-structures, at the Block-level, in such a way that no further modifications of test structures, in any block, was needed at Core or Top level. This strategy worked perfectly through out the project after first time the Top level had been assembled.

Test points have been inserted in all blocks, except those that are extremely time critical. It was easy to insert test points.

Scan chains and test points are inserted at gate level after synthesis.

5.1. Scan and logic BIST for ULs

The DFT flow to insert scan chains with the DFT tool worked rather straightforward without any special problems. The only problem was how to exclude certain flip-flops from being inserted into any scan chain. The number of scan chains inserted into any block was controlled by the number of different clocks in the block and the feature that allows definition of the maximum length of any scan chain.

The only block that did create some trouble was the ARM control platform. Problems occurred not due to the DFT tools used but due to the design style used for the ARM control platform (see figure 5). It had an internal tri-state bus and several latches, due to both edges on the clock was used for clocking.

First run: Scan chains are connected according to the logic hierarchy (arbitrary with respect to physical layout).

Script redefines the order according to physical layout.

2nd run: Scan chains are connected as described in the re-ordered list.

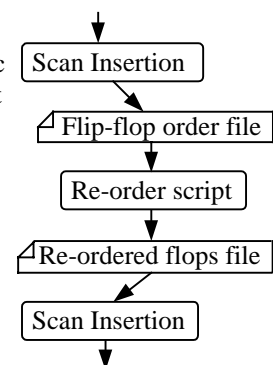


Figure 3. Re-order flip-flops in scan chains

For some blocks it was necessary to control the order in which flip-flops were to be connected into the scan chains in order to avoid routing congestion. The order of the scan chains was achieved by using the flow shown in figure 3. A user created script was used to define the order of the flip-flops in each scan chain.

Logic BIST requirements did not add any special problems at DFT insertion at the Block level. No special verification for logic BIST was carried out at the Block-level.

Verification of inserted structures was fast and easy to carry out, creation of testbenches was easy and worked without any problems in the simulator used (VerilogXL).

5.2. Non-scan flip-flops

Full scan was used, but even so, not all flip-flops could be included into scan chains. This situation is valid for flip-flops involved in clocking, creation of reset, scan enable pipeline or when placed in areas having extremely time critical paths.

The tools had no embedded functionality that made it possible to define what flip-flops to exclude from being inserted into a scan chain. This limitation was handled by using two different ad-hoc methods, one for each ASIC. Non of the methods used was optimal, but the results were satisfactory.

The first method: The procedure to insert scan chains in a block started with running a user created script. For each flip-flop that was not to be inserted in a scan chain, the script added the label "_nonscan" at the end of the device name. The scan insertion tool was used to insert scan chains into the modified netlist, and the flip-flops with the extension "_nonscan" were not inserted into a scan chain.

To make this method work: A "dummy" library of all the vendor flip-flops was created. It was only used during DFT and as a complement to the regular library supplied by the ASIC vendor. The contents of each flip-flop, in the created "dummy" library, were a black box and the device name was the same as in the vendor library but with the additional "_nonscan" added at the end of the device name. The scan insertion tool will not exchange the "_nonscan" flip-flops into scan flip-flops since it also doesn't know that they are flip-flops. After scan chains have been inserted and verified, synthesis software (Synopsys) was used to remove the "_nonscan" extension.

The problem with this method is that a modified netlist is used when running the DFT tools. Using a modified netlist is acceptable at the Block- and Core-level but is not what you want to use when running rule check at the Top level. With this method the procedure with inserting "_nonscan" had to be repeated at the Core and Top levels.

The second method: This approach makes use of the behaviour that Verilog uses the last definition used. The "_nonscan" label is added into the netlist once and never has to be removed. At end of the netlist, of the design, is the Verilog attribute "celldefine" added, one for each type of flip-flop not to include into a scan chain. The attribute "celldefine" is used to define the nonscan flip-flops as regular flip-flops, this makes the simulator and synthesis tools interpret the nonscan flip-flops as regular flip-flops. When running the DFT tools, an additional file

is included on the command line. This additional file also contains the "celldefine" definitions of the nonscan flip-flops, but now they are defined as black boxes. Since the additional file is included on the command line after the netlist, it overrides the "celldefine" definitions in the netlist and the nonscan flip-flops will not be inserted into any scan chain.

With the second method the "_nonscan" attribute only has to be added once and never has to be removed. This means that the netlist does not have to be temporarily modified in the DFT flow.

5.3. Memory BIST for ULs

Two memory BIST controllers were used to implement test of the 31 SRAMs . Memory BIST was generated and assembled at the RTL-level. Analysis and verification was carried out at the gate level as shown in figure 4.

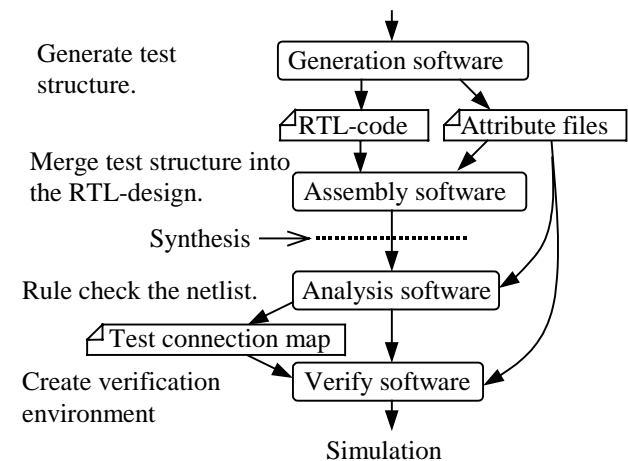


Figure 4. Basic memory BIST flow

Both memory BIST structures were synthesised and verified without any special problems.

The attributes file from the generation software is required as input during verification in order to rule check the connections throughout the netlist and automatically create a verification environment (testbench plus stimuli).

It was easy to automatically add scan logic directly at the inputs and outputs to/from the memory areas. This feature was used with all memories, and improves the fault coverage at scan test.

5.4. IP DSP

The DSP is an in-house IP. The IP was tailored for the project and consisted of a DSP core, one SRAM and a collar. The collar made it possible to use the scan chains, in the IP, for either debug or production test.

Scan chains were inserted with the DFT tool. The TAP, in the PADS module, is used as interface during debug mode. The only difficult part was to connect the logic that controlled debug to the PADS module. This problem was solved by "fooling" the DFT tool that the debug controller interface was a memory BIST controller.

The memory BIST for the SRAM was not inserted with the DFT tools used. Even so: It had a hardware wrapper to make it compatible with the commercial DFT software chip-level interface. This memory BIST could not be verified with the commercial DFT tool because the attributes file needed was missing (see figure 4). This attribute file could not be created by hand since it must be in a binary format.

The DSP IP was delivered to team3 both as a soft IP (netlist) used for simulation and a hard IP used for floorplanning.

5.5. IP ARM

The IP ARM consisted of a μ Processor (see ref. 3) of type hard IP, as a blackbox, and a Control platform (CP), as shown in figure 5.

The μ Pr IP is tested with its own test interface. The CP is tested together with the remaining parts of the Core. The μ Pr IP is tested with parallel test vectors supplied by the company owning the μ Pr IP. The test vectors were used without any special problems.

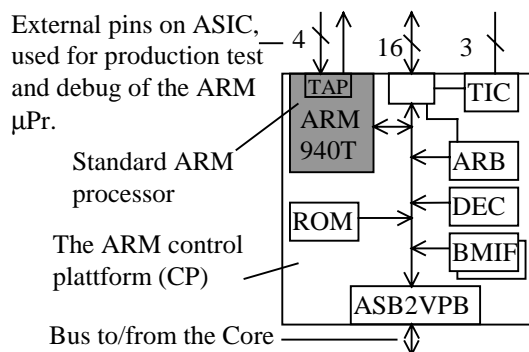


Figure 5. The different parts in the IP ARM

The CP was developed with a third party menu based tool. The output was a description at RTL-level. The commercial DFT tool was used to insert and verify scan chains and memory BIST in the CP. A lot of work was required to insert scan chains. The source of trouble was the menu based tool that created a DFT unfriendly design structure. Both edges of clocks were used, a huge amount of latches existed in the design and the embedded tri-stated bi-directional bus had four different bus master.

The problem with the latches: It was solved by creating a script that searched through the netlist for all latches and inserted logic that broke all the feedback loops during test. This script had to be modified each time the CP was synthesised.

The problem with the four possible bus masters: It was solved by adding logic that assured that only one of the four bus masters could become a master on the bus at a time.

No special problems were encountered at insertion and verification of memory BIST for the ROM.

6. DFT AT CORE LEVEL

The PADS module and connection of scan chains throughout the Core were created by using the commercial DFT tool.

The Core must be created before the PADS module can be created, because creation of the PADS module requires one input file that is generated, by the DFT tool, during the Core flow.

The IP-module ARM was connected without any problem regarding DFT. The IP-module DSP was a little tricky to connect.

Logic BIST is not verified at the Core level, only at the Top-level.

6.1. Scan and debug

Initially, the DFT tools couldn't handle hierarchical scan insertion. As soon as the supplier of the tools understood the problem, they enhanced the applicable parts of the software. Hierarchical scan insertion was supported with the enhanced software.

In general all scan chains were connected in such a way that each scan chain only pass through one block at the Core level, see figure 6. The total amount of scan chains was 153 at the Core level.

The work to connect the scan chains is described below.

First the VHDL description of the Core (top-level design block) was modified to add-scan-chain ports at the top-level "component" and "port map" declarations in the Core file. This task was completed by creating a script that extracted the scan-chain input and output port names from the gate-level descriptions of the individual blocks containing the scan chains and added ports with the extracted names to the top level VHDL entity representing the Core. Scan-chain inputs were connected to logical low and scan-chain outputs were connected to "OPEN". The VHDL netlist was next converted to a Verilog netlist, using synthesis software. The scan insertion software was then used with the Verilog netlist to connect the scan-chain input and output ports from the block boundaries to the newly created companion ports at the boundaries of the Core.

Two of the scan chains were always connected in the wrong way by the scan insertion software. These two scan chains were connected through two blocks instead of one, as depicted in figure 6. The problem was solved by running two passes of the scan insertion software. The first run was used to check if the number of scan chains at the Core level was the same as the sum of the scan chains in all the individual blocks. If the number of scan chains did not match, the output file from the scan insertion software describing the chains was edited and used as input for the second run. In the second run, the scan insertion software correctly connected the chains exactly as described in the edited input file.

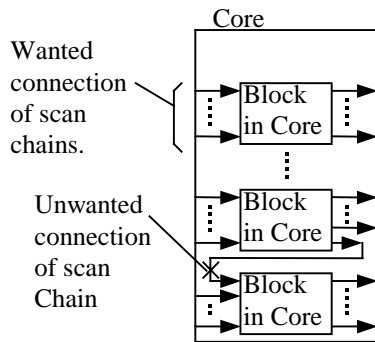


Fig. 6. Connection of scan chains at Core level.

One scan chain in each DSP IP was connected through the Core in a different way from the approach described above. This difference is due to the scan chains in the DSP IPs also being used for debug of the DSP IPs. The scan chains through each DSP IP are connected, through the Core, as shown in figure 7a.

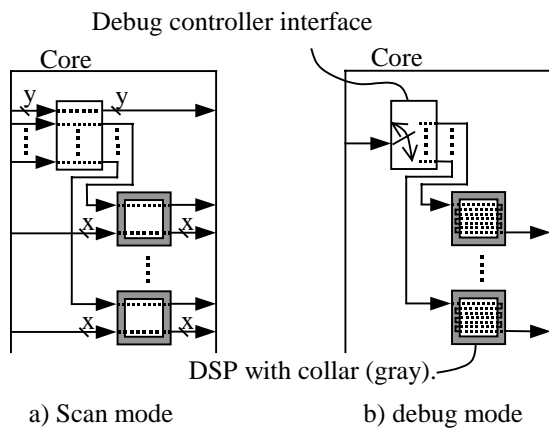


Fig. 7. Connections of scan and debug for DSP IPs

During debug of a DSP IP the scan chains are connected as described in figure 7b. All "x+1" scan chains in a DSP IP are in debug mode merged together into one long scan chain, this is performed by the grey marked collar surrounding each DSP IP. The scan insertion software always connected one scan chain to each DSP IP in an incorrect way. The tool short circuited the scan chains as shown in figure 8. The unwanted connections were removed by using a synthesis tool as part of the script flow.

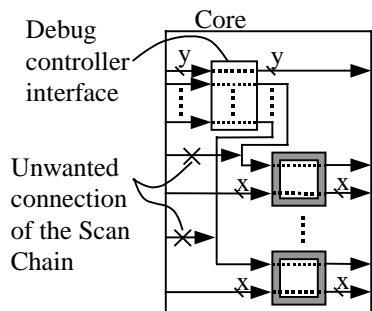


Fig. 8. Short circuit of scan chains in DSP structure.

The ARM IP also had an embedded debug structure, which was accessed through a dedicated interface at the

ARM IP. No problem occurred during connection of the ARM IPs scan chains through the Core level.

It took approx. 17 hours of computer run time to connect the scan chains through the Core level and rule check the Core and Block levels. It took another 12 hours of run time to automatically create the testbench. The simulation itself took just 40 minutes of run time.

6.2. Memory BIST

All memory BIST controllers were initially verified at the block level. Even so: It was a time consuming task to repeat the verification at the Core level. The main reasons are as follows:

First reason: The DFT software extracts the connectivity of the test structures from the netlist and places the information in a test connection map file, as shown in figure 4. The verification software uses this connectivity file, the attributes file from the generation software, to generate a design specific verification environment. A problem was encountered during this operation because the synthesis software slightly modified the names of the memory BIST controllers at an earlier step in the flow. The software that extracts the test connectivity did not recognise the "modified"-names given to these controllers by the synthesis software. Missing information in the documentation, that the name of the memory BIST controllers must be identical to the names of the attributes files, was the reason for the mistake.

Second reason: Verification of the memory BIST test structure in each DSP IP could not be carried out with the DFT tools, because the memory BIST in the DSP IPs were not generated by using the DFT tools. Verification of memory BIST can only be carried out if an attribute file, as shown in figure 4, exists for the memory BIST test structures. It was not possible to create an attribute file by hand for the "foreign" memory BIST test structures because the file format is proprietary to the DFT tool supplier and is not available in a readable format. Therefore, only the memory BIST test structures created with the DFT software could be taken through the flow of figure 4. Testbench and test stimuli had to be created by hand to verify the memory BIST test structure in each DSP IP.

Third reason: The analysis software had a problem tracing through multiplexers. Solved by providing a DFT-specific model, one for each component not extracted successfully, in order for the extraction to be done properly.

Fourth reason: Several runs through the analysis software were required due to these problems. The loop time through the analysis software was one hour or more.

Information in the documentation describing how to avoid these issues, and a more user-friendly version of the analysis software, would have saved several days of work.

6.3. PADS module

The flow used to create and verify the PADS module is depicted in figure 9.

Once the TAP was created it was easy to recreate new modified versions of the TAP as many times as needed. Some bugs and limitations were detected as follows.

The PADS structure created by the DFT software consisted of hundreds of VHDL-files. A few of the files created were not compatible to the VHDL-standard. They could be compiled after corrections with help of a user created script.

One clock pin on the package was bonded to two pads on the silicon. This was not supported by the DFT software. The problem was solved by adding one non existing package pin in an input file and, as part of synthesis, eliminating the non existing package pin. All lines addressing the non existing package pin were then removed from the BSDI-file created by the DFT software.

The DFT software did not allow the user to select the active level of scan-enable signals. In our case the ASIC vendor library used the opposite level of scan-enable compared to the one created when the PADS module was generated. This problem was solved by using a user created script that added inverters at all the different scan enable ports from the PADS module.

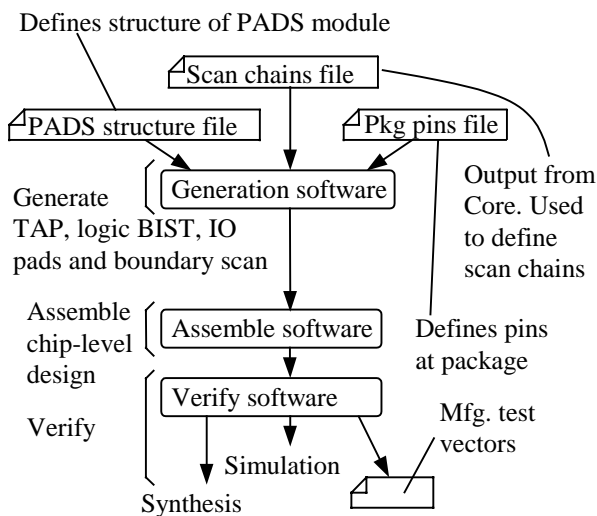


Fig. 9. The process to create and verify PADS module

The boundary scan structure created was not totally IEEE 1149.1 compatible. The TCK clock forwarded to the Core used falling edge, the standard demands rising edge (see ref. 1). User defined bits are added into the instruction register instead of into a separate data register, but the user defined bits are not used at decoding of the instruction register. It is only possible to create one data register in the boundary scan structure. This limitation is not included in the 1149.1 standard.

No straightforward way was supported to connect the debug controller for the DSP IPs in the Core. The debug

controller was connected into the PADS structure by fooling the tool that the debug controller was a memory BIST controller.

Total computer run times needed, to create and verify the PADS module, was just a few hours.

7. DFT AT TOP LEVEL

Verification of all the DFT structures implemented was carried out and test vectors files were created with the different steps as described in figure 10.

Verification and creation of test vector files must be carried out as separate operations and for one test type only to avoid computer crashes. This limitation is not depicted in the documentation.

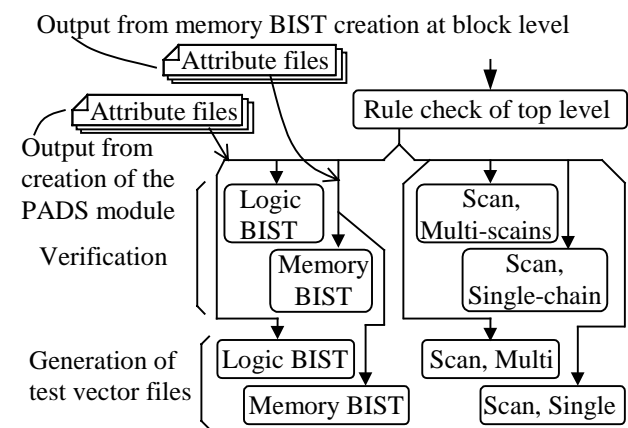


Fig. 10. The steps at Top level

Scan in single- and multi-chain configurations, memory BIST and logic BIST are fully controlled through the TAP in the PADS module.

Rule checking at the Top level was easy to perform.

Verification of scan configurations was easy to perform, and without any special problems.

Verification of memory BIST and logic BIST was more difficult to carry out at the Top level than at the Core level. Once again the problem was to get through the test connectivity extraction step, as shown in figure 4.

Test vector files were extremely easy to create for all the different test types included in figure 12. Test vector files for scan (single and multi chain configuration) can be created incrementally, in as many and small portions as wanted. This is a very useful feature making it possible to save calendar time by running creation on several computers at the same time.

Computer run time needed to carry out rule check for each test type was approximately 60 hours. Verification of one scan configuration type, using 10000 test vectors, required approximately 19 hours of computer run time.

Gigabytes of output files was created each time verification was performed or a test vector file was created.

Logic BIST was included to try to increase the fault coverage for timing related faults. Most parts of logic BIST are executed at full clock speed. Faults in paths between clock domains are not checked at speed but all clock domains are checked simultaneously.

The total number of flip-flops inserted into the scan chains is 71909. This is the number of flip-flops connected between the TDI and TDO for the single-chain scan configuration.

There are 16 scan chains in the multi-chain scan configuration. One of these chain is connected through the PADS module, and contains 1315 flip-flops. The other 15 chains are connected through the Core, and have lengths in the range 3600-5316.

153 scan chains are used in the logic BIST configuration. The shortest scan chain contains three flip-flops, and the longest contains 602 flip-flops.

8. SCAN FAULT COVERAGE

It is not easy to understand what the total fault coverage reached is by using the DFT tool because the tool reports fault coverage as a percentage of all faults that the tool assume can be tested.

Tried to compare fault coverage by using a competitors commercial DFT tool on the implemented scan test structure. Failed to compare fault coverage because never managed to make the competitor DFT tool to understand the scan test structure.

9. PROTOTYPE EXPERIENCE

Logic BIST was not verified on prototype devices because no time was available for this activity.

The boundary scan, memory BIST and scan tests worked as expected.

10. DOCUMENTATION AND SUPPORT

The documentation that follows together with the DFT tools is of good help to understand the basics about the tools at a high level, but are of limited help when using the tools at a real design. The reason for that is mainly due to that the examples included are too few, small and incomplete, and no executable tutorials are included. Another reason is that important user information sometimes is not highlighted or even missing.

The support, from the DFT tool supplier, has normally been fast and of good quality.

Ericsson is a world wide company with design centers all around the world. Problems, addressing the DFT tools, reported from one of the design centers were a few times not forwarded, by the DFT tool supplier, to the other design centers.

11. CONCLUSION

It is possible to implement and verify DFT of SoCs by using the commercial DFT tools. The implementation required a significant level of direct support from the commercial vendor. Much of this support could have been avoided through improvements to the documentation. It was time consuming for a first time user, as well as for an experienced DFT engineer to learn to use the tools. Better documentation would contribute significantly to improving this situation.

It is very easy to create test vectors with the DFT tool.

For features not directly supported by the DFT tools:

- It was easy to define the order in which flip-flops are inserted into the scan chains.
- It was possible to exclude certain flip-flops from being inserted into a scan chain.
- Debug structures using scan chains were connected into the DFT structure.

It was also easy to automatically create testbenches for verification of scan chains. But it was very time consuming, and difficult, to generate the testbenches for memory BIST and logic BIST, mainly due to the difficulties with extracting the test connections from the design.

Boundary scan was implemented and verified without any special problems. The generated boundary scan structure was not fully compatible with the IEEE 1149.1 standard. This did not introduce any problem from a functional point of view.

12. ACKNOWLEDGE

Since this project was completed, the commercial DFT vendor responded that its documentation has been enhanced and its flow has been simplified, addressing most of the issues described in this paper.

13. REFERENCES

- [1] IEEE Std 1149.1-1990; IEEE Standard Test Access Port and Boundary-Scan Architecture.
- [2] DESIGN FOR AT-SPEED TEST, DIAGNOSIS AND MEASUREMENT: Benoit Nadeau-Dostie. ISBN 0-7923-8669-8.
- [3]. ARM documentation at ARMS own WWW site.