BOSCH
Invented for life

How Logic and Memory BIST can help to address some functional safety requirements in automotive ICs

Christophe Eychenne

Robert Bosch SAS

06901 Sophia Antipolis, France

www.bosch.fr

ABSTRACT

With increasing use of electronic systems in vehicles for managing critical functionality, the requirements for safety in automotive chip architecture is becoming mandatory.

Thus automotive chip must implement functional safety feature in addition to the general functionality.

The Logic and Memory BIST (Built-In Self-Test) are one of the option to facilitate safety at chip level with a minor area penalty. It goes through logic and memory to ensure that the chip has no permanent fault due to aging at power up, but also during functional mode.

Hence this paper deals with the Synopsys LBIST and MBIST proposed solution.

It describes an efficient DFT architecture to detect permanent fault on design part even in functional mode.

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Electronic devices used in safety critical automotive applications require a periodic in-field testing. All these automotive safety requirements are being driven by ISO 26262 standard. This standard is a risk-based safety standard, where the risk of hazardous operational situations is qualitatively assessed and safety measures are defined to avoid or control systematic failures and to detect or control random hardware failures, or mitigate their effects.

The functional safety critical device needs to be robust to following errors that could violate safety goal:

- Permanent error, Permanent fault that creates a systematic error which leads to an inability to execute safety critical functionality.
- Transient error, Random fault that creates a short time error which leads to an inability to execute safety critical functionality, but disappears after reset

Several hardware options have been proposed to facilitate safety detection and fault tolerance at chip level:

- Redundant Critical hardware module implemented in lock-step of each other, processing the same inputs, but this option must be limited to very critical part due to the duplicated area penalty.
- Hardware monitors module to detect failures, like sudden temperature change, change in clock frequencies, change in signal/power voltage levels, etc.
- Watchdog timers to specify a period in which a task need to be executed.
- Glitch filters on critical inputs to stop noise and transient spikes from getting in.
- Error Correcting Code (ECC) and Cyclic Redundancy Check (CRC) mechanism to preserve data integrity within the system.
- Built-in Self-Test (BIST) to detect any permanent hardware fault due to aging.

Among all these existing hardware solutions for fault detection, monitoring and timers solutions helps to detect global issue, but are not efficient to detect logic or memory problem in device part.

Hence, only hardware lockstep, ECC/CRC and BIST solutions are able to catch problems on logic and memory with more or less efficiency. For sure, the hardware lockstep is an exhaustive solution to prevent all type of faults, as soon as one can be sure that the hardware is safe at start-up. Unfortunately, this hardware duplication is not manageable for large IC due to huge area overhead and requires a thorough check of the comparison logic. Thus it is limited to very critical part, like safety control unit.

On the other side, ECC and CRC have a detection limited to data corruption in the system, and thus can only protect memories, registers or bus protocol data from permanent or transient faults,

Therefore, Built-in self-test (BIST) is becoming an effective way of enhancing the safety of automotive SOCs demanding in-field self-testing of the functionality and integrity of the system against permanent fault, with a small area penalty.

In this paper, we will describes an implementation for in-field self-test for logic and memories for an automotive System-On-Chip which has been based on an evaluation of the Synopsys solutions for both Built-In Self-Test technique.

The paper is organized into five sections. Section 2 provides an overview of the Build-in self-test implementation to support online safety needs. A Synopsys LBIST architecture/implementation for safety use is detailed in Section 3, with also a dedicated paragraph for potential improvements. Section 4 describes a Synopsys MBIST architecture/implementation for safety according design constraints with a dedicated paragraph for potential improvements. Section 5 concludes this paper.

2. Built-in Self-Test implementation to serve functional safety needs

Primary goal of the DFT hardware and architecture is mainly the manufacturing test, based on 2 techniques.

On one hand, the scan testing is a deterministic logic testing, where Test patterns are pre-generated by an ATPG using a gate-level representation of the design. Scan testing requires Automated Test Equipment (ATE), which controls the test patterns supplied to the SOC. Patterns are stored in tester memory and loaded into the circuit using parallel scan chains, in the same way, captured values are unloaded from the circuit and compared with an expected value at tester level. The automotive coverage target for scan testing is very high:

- Test coverage > 99% for stuck-at faults
- Test coverage > 90% for transition faults.

On the other hand, since decades, memories are tested with Memory Build-In Self-Test (BIST). The memory BIST tests the memory cells thanks write and read access to all locations of the memory to ensure that the cells are operating correctly. This process gives additional test coverage of the address and data paths that Memory BIST uses and one can consider that implementing thorough memory BIST algorithm gives a 100% test coverage on the memory.

All this already existing DFT hardware appears as an Eldorado for the functional safety needs on safety critical device part, looking for the ability to achieve desires testing target in limited time with small area overhead. Indeed DFT hardware is able to target these high test coverage requirement with a minimal area overhead, because already implemented for manufacturing test.

Unfortunately, if the memory BIST hardware can be reused as it is for in-field testing, the scan logic is inadequate for such purpose due to the strong interaction with ATE. Thus the logic must be tested with a specific Logic BIST hardware that should reuse as much as possible existing scan hardware, like scan segments and clock control logic. The Logic BIST testing is random ATPG where an on-chip pattern generator feeds the scan chains, an on-chip result compressor compresses the scanned out response of all patterns into a final signature. Even if the Logic BIST solves the problem of pattern generation and comparison, the inherent non-deterministic pattern generation of the random generator, leads to lower test coverage compared to deterministic scan patterns and so higher runtime for a given test coverage target.

Furthermore, the BIST runtime flexibility must be carefully taken into account to fit in a given time frame, as well as the extra power consumption during self-test.

Thus this paper will explore the use of these ideal logic and memory BIST technics for in-field testing when the device is switched on (power-up), but also to run on the fly test on some device critical part (online testing). This study will used the Synopsys STAR Memory System solution and the DFTMAX Logic BIST solution and is intended to propose a methodology for logic and memory online testing.

From safety point of view, the design is split into 2 parts, as shown in Figure 1:

- The safety sub-system part or safety control unit is in charge of checking the integrity of the device parts and move the device into safe state in case of detected violation. This small safety critical part of the device is self-tested only at power-up for the logic and the memories. But during functional run time, no more in-field testing is feasible, the monitoring must remain active and available all the time. Here specific technics are required to detect fault during run time, like hardware lock-step.
- The rest which concerns the biggest part of the design, is well adapted to power-up and online testing. Indeed if system allows unavailability of some parts or sub-systems for a given time frame, logic and memory self-test are perfect technics to detect permanent fault inside at low

cost. Hence only protection against transient fault will be required for these parts thanks ECC, CRC or local hardware redundancy for critical parts. Thus design will be architected not only into sub-system function, but also into self-tested part ("safety element") that could be excluded from the system for a given time frame for in-field self-test.

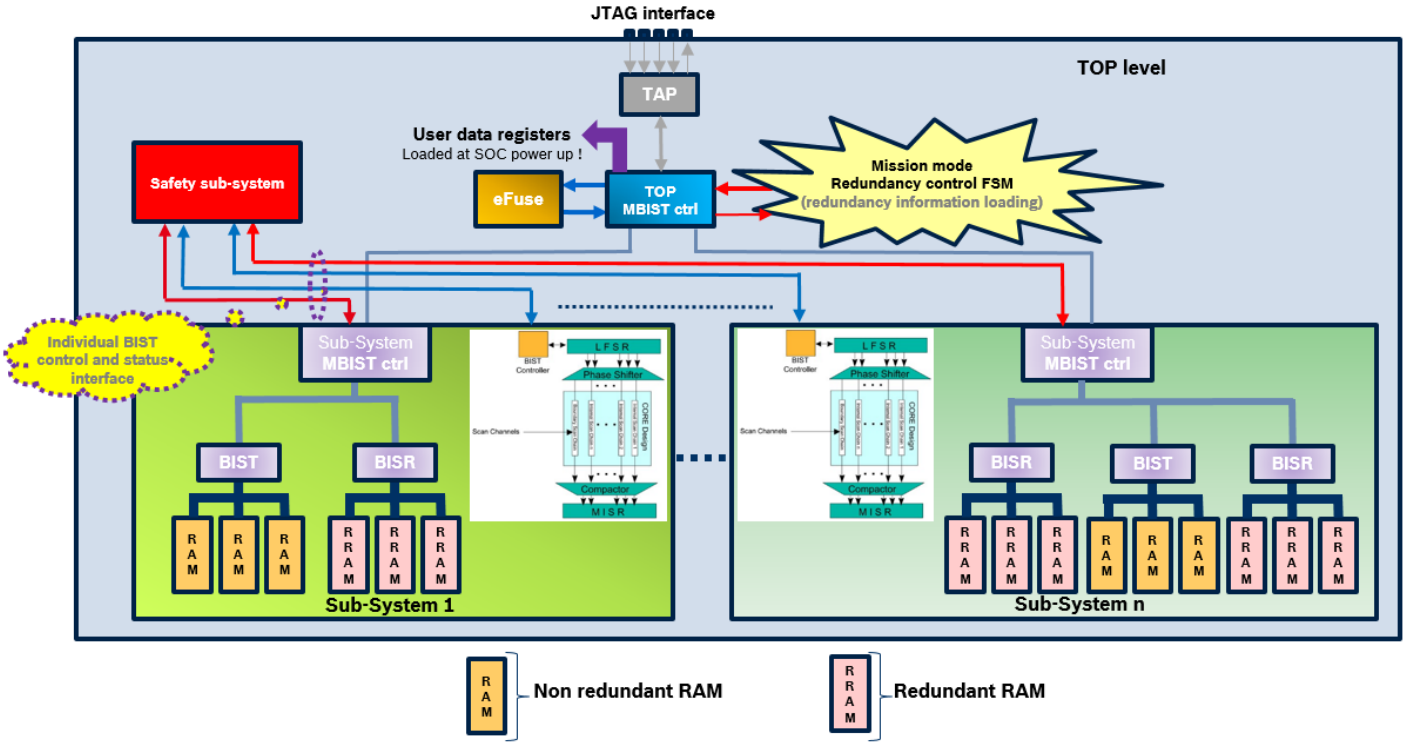

Figure 1. System architecture for in-field self-test

3. Logic BIST implementation for functional safety

This section gives a brief introduction on Logic BIST, in a second step an overview of the Synopsys solution, then a proposed architecture, followed by results and finally a way of improvement in term of feature to better support “in-system” testing.

The logic BIST architecture most widely used to apply patterns and observe responses on a chip is the self-testing using MISR and parallel shift register sequence generator. The basic mechanism uses a pseudo-random pattern generator (PRPG) to generate the inputs to the device's internal scan chain, initiate one or more functional cycles to capture the response of the device, and then compress the captured response into a multiple input signature register (MISR). The compressed response that comes out of the MISR is called the signature. Any corruption in the output signature indicates a defect in the device.

Because of such architecture, logic BIST has more strict rules concerning any source of unknown value which LBIST cannot tolerate. Indeed, during LBIST, any unknown value in the logic will corrupt test response and result in an incorrect signature. Thus black-box, non-scan cells and multicycle/false paths must be isolated compared to classic scan architecture where ATPG can masks any potential unknown states when creating deterministic patterns.

Finally, with the pseudo-random patterns used in logic BIST, there are sometimes faults that are difficult to observe and difficult to control. Testing can be improved by adding observe and control test points, targeting this pseudo-random resistant faults.

So Logic BIST technology has been in use for decades and was initially intended to be used as low cost test solution during manufacturing to support IC growth. When test compression technology has been introduced, it became a lot more cost effective to use test compression at the tester rather than LBIST.

Now, LBIST use becomes relevant for repeated testing in the field to detect permanent fault due to aging. Thus, for in system testing, the LBIST goal is to reach the highest coverage with a minimal amount of pattern, to minimize unavailability of the tested part.

Of course, for manufacturing test, the classic test compressed ATPG pattern will be used and LBIST will be turned off and treated like functional logic.

3.1 LBIST architecture proposed

The LBIST architecture must serve an in-field system testing at power-up and cyclic run time execution on part of the system. In order to keep the system available during application mode only sub-system part can be self-tested, when the rest of the system is still working in mission mode.

Thus the architecture must support local LBIST run on specific sub-system parts during mission mode and generates safe output during logic BIST run to not disturb surrounding logic kept running in mission mode. In other words, sub-system under LBIST test is unavailable till end of the LBIST like when a power domain is switched off.

Moreover, in order to minimize area overhead, scan and LBIST constraints are aligned to share as much as possible the infrastructure, like scan segments.

The goal of LBIST is to produce higher quality results with fewer patterns to minimize runtime and unavailability of the sub-system. In this way, Synopsys proposed a reseeding method that allows to perform incremental run and so achieve better coverage with small amount of pattern.

In parallel, for pure design constraints, in order to support late ECO and late system requirements in term of LBIST allocated run time, all the LBIST control parameters must remain programmable by software. Hence, seed value, associated signature value, pattern number value, and shift cycle number value must remain configurable at device level.

Finally, the LBIST architecture has been implemented with the following assumptions at sub-system level:

- Local clock control required thanks OCC hook-up to all the functional clock(s).
- Local reset control required on all functional asynchronous reset(s)
- No internal clock divider allowed.
- All primary input ports must be isolated by a wrapper to ensure controllability and so preserve test coverage
- All primary output ports must be isolated by a wrapper to ensure observability and so preserve test coverage
- Safe state is required on all primary output ports
- No unknown value generator allowed (X-bounding logic required to prevent X's propagation):
    - No black-box allowed (isolation required)
    - Memory must implement a bypass mechanism feature
    - No non-scan element allowed
    - Multi-cycle path and false path got an hold mechanism (RTL or gate level fix)

3.2 Synopsys solution overview

Synopsys DFTMAX Logic BIST solution is a synthesis-based solution for in-system self-test of digital integrated circuits. Design Compiler synthesizes logic BIST, scan, and compression for manufacturing test in the same step. These DFT structures are automatically integrated into the design architecture using DFTMAX.

DFTMAX Logic BIST works in conjunction with DFTMAX compression and TetraMAX ATPG to provide test point analysis and insertion, seed and signature computation and coverage calculation like detailed in Figure 2.

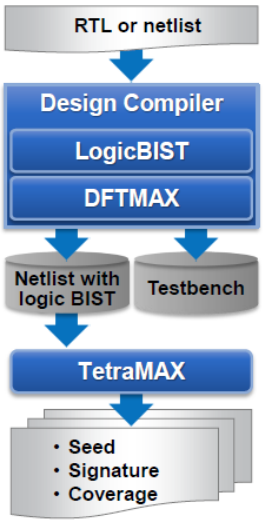

Figure 2. Synopsys synthesis-based logic BIST flow

The DFTMAX Logic BIST compression enables a design part to test itself using the same scan chains already implemented for manufacturing test. It uses a pseudo-random pattern generator (PRPG) to create scan data, and a multiple-input signature register (MISR) to capture the design response. At the end of the test, if the actual signature matches the expected signature, the self-test asserts status signals (Figure 3).

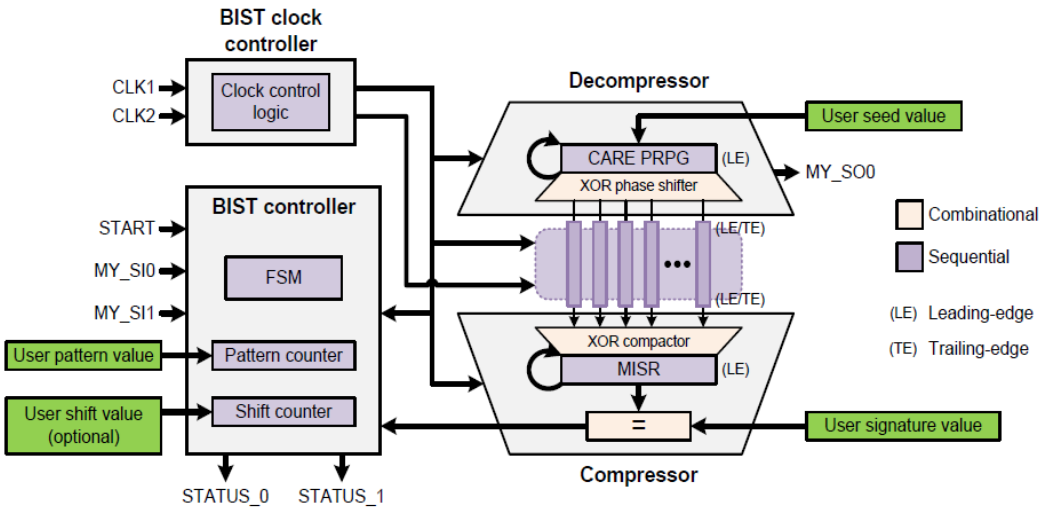

Figure 3. DFTMAX Logic BIST architecture

Note that Clock controller, reset controller, boundary wrapper and output safe state isolation will be automatically inserted at gate level during DFT insertion phase and avoid any safety intrusion at RTL level. Only chip safety controller unit will be developed in RTL and integrated in top level to manage the safety activity and generates adequate actions.

The self-test is initiated by simply holding the START signal high. If no error is detected, the PASS/FAIL output STATUS [0] will be low. One additional output signal STATUS [1] informs when the LBIST run is done (Figure 4).

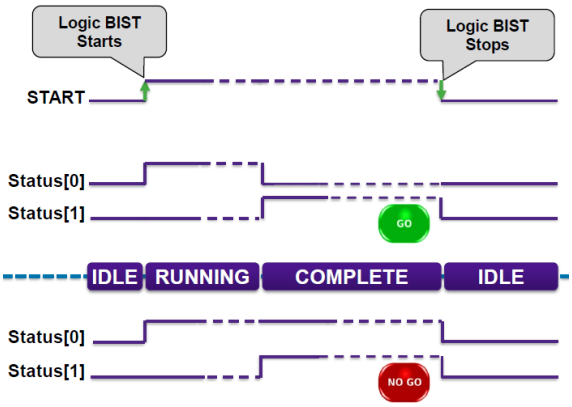

Figure 4. DFTMAX Logic BIST control interface

For more detail, please refer to the DFTMAX Logic BIST user guide (ref. [1])

3.3 Experiments and results

An experiment has been done on a sub-system including Synopsys STAR Memory System (SMS) BIST infrastructure and network to get a relevant safety solution. Hence the sub-system includes a full standalone safety solution for permanent fault detection on memory and logic, covering both stuck-at fault and transition fault.

The design is an image processing module, consisting of 38,2 k-flops and 23 memories, representing a total of 4 M-bits, with 2 clock domains (1 functional / 1 SMS BIST) and 3 asynchronous resets (1 functional / 2 SMS BIST).

The following DFT architecture has been implemented to support both in-field testing and manufacturing testing:

1. For manufacturing test: 1 pipelined test compression has been integrated with 8 channels to drive 191 scan segments of 200 flops.
2. For LBIST test: 1 test compression has been integrated with 191 scan segments of 200 flops. A boundary isolation has been implemented around the self-tested part with safe output state. Moreover Test point insertion flow has been used to evaluate coverage improvement.

All memories includes a bypass feature mechanism that must be enabled during LBIST. In addition, OCC, integrated during DFT insertion, have been configured to be able to deliver up to 4 capture clock pulses to allow memory write and read access with ATPG scan patterns.

For this experiment, the first step was to check the quality of the design in terms of ATPG coverage.

Thus, with the previous scan configuration, a test coverage of 99.13% (fault coverage of 98.87%) was achieved for stuck-at test and 97.09% for transition test (fault coverage 96.84%). So a clean design from pure manufacturing point of view!

Then several LBIST architecture have been tried to check strategy impact on LBIST test coverage:

- Clock ratio: OCC weighted capture probability attribution for clocks and asynchronous resets
- Test point insertion flow: “pattern_reduction” vs “testability” vs “random resistant analysis"
- Capture clock pulse number
- Sub-system isolation with safe state output: “core wrapping” vs “user-defined test point”

3.3.1 Reset and clock weight attribution for capture groups

As mentioned in the architecture, for in-field self-test on part of the device, one can’t afford any disruption on the system functional clock. Hence it’s mandatory to integrate OCC on primary input clocks at the boundary of the self-tested part and get local control of the clock during LBIST.

So with DFT-inserted OCC controller flow, the tool uses an OCC controller design with additional Logic-BIST clock control logic. In our experiment, the OCC slow clock used to load scan segment is common for scan ATPG and BIST.

3.3.1.1 Weighted clock/reset groups logic structure

By default, all OCC clocks capture in each Logic BIST pattern. If capture paths exist between clock domains, additional logic is required to selectively enable non-interacting capture clocks in each pattern. This avoids capturing an X value from an asynchronous clock domain that is also clocked in that pattern.

To prevent such capture issue, Synopsys proposes to split clocks into groups and assign a weight to each group. Thus in each pattern, a single clock group can pulsed for capture proportionally to the weight values attributed.

Because Logic BIST does not use the OCC clock chain registers, they are repurposed for clock group selection during Logic BIST self-test. In each pattern, the clock chains load a pseudo-random value from the PRPG. This value feeds a weighted clock group selector that enables the pulse pattern for one of the capture groups, as shown in Figure 5. The number of generated pulse can be programmable thanks to the “lbist_clk_pattern” control bus, but must remain constant all along a LBIST run. For more detail, on this structure, please refer to the DFTMAX Logic BIST User Guide to get more detail (ref. [1]).

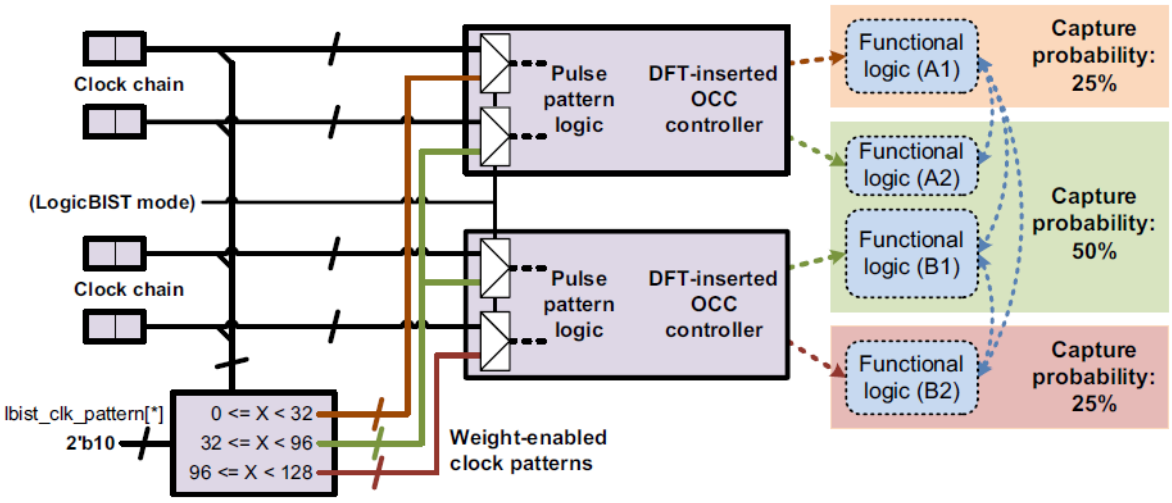

Figure 5. Weighted capture groups logic structure

In parallel, one can't imagine to play with the asynchronous reset signal of the system to check a device part. Here again, a local control at the self-tested sub-system boundary is required. Indeed, from test point of view, asynchronous resets are similar to clocks as they cause sequential cells to capture a value. Thus one must ensure that there's no capture interaction between clock and reset during LBIST and all asynchronous reset must be part of a dedicated weighted clock capture group, as shown in Figure 6. During Logic BIST, reset are under control of LBIST clock/reset weight decoder during capture and disabled during shift phase.

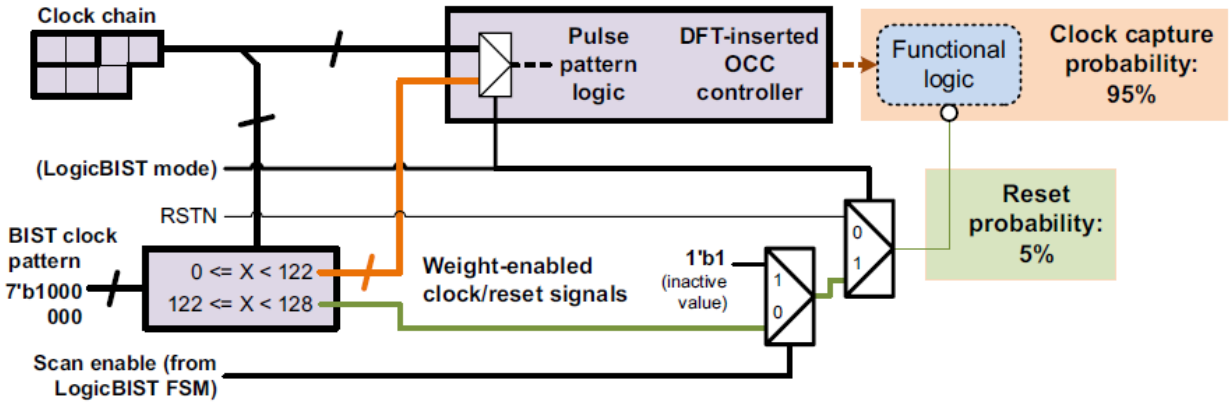

Figure 6. Weighted clock/reset capture groups logic structure

After the concept description arises the challenge on how to distribute this clock probability when the sub-system has several clock domains to get the best coverage with the fewer pattern number.

How to define an optimal weight attribution for a hardwired function???

3.3.1.2 Clock/reset capture probability distribution

A predictable method would be welcome and the first approach to get this information is to perform a basic stuck-at ATPG run enabling OCC. One can suppose that the optimal LBIST clock distribution can be close to the ATPG clocking strategy for the same fault dictionary. Thus, based on this first assumption, a detail flow is described below to attribute an optimal clock/reset weight:

1. First step is to perform a basic stuck-at ATPG run using OCC and collect ATPG clocking strategy per pattern (OCC enabled):
   - set_faults -fault_coverage -model stuck
   - set_atpg -capture_cycles 0 (Only basic scan pattern generated)
   - run_atpg -auto
   - report_pattern -all –clocking (Collect ATPG clocking strategy per pattern)
2. Second step consists to count occurrence of each clock domain for all patterns (experiment with 2 clock domains clk and SMS wrck):
   - Get the number of pattern involving an OCC in the list (Total pattern number):

     grep "capture_cycle=0" report_clocking_per_pattern.rpt | wc -l
   - For each OCC, Get the number of pattern involving "clk_i OCC instance name" clock domain:

     grep " clk_i OCC instance name" report_clocking_per_pattern.rpt | wc -l

3. As mentioned previously in the weight clock/reset logic structure, all asynchronous reset are controlled together (same group) and one can assume a low probability to detect fault on reset tree (direct primary control) compared to clock pulsing.
4. Finally, clock weight formula per clk_i clock domain is as follow:

   clk_i weight = [clk_i pat number / Total pattern number] x [128 counter value - asynchronous reset weight]

   with "clk_i pat number" = result of the grep " clk_i OCC instance name " report_clocking_per_pattern.rpt | wc -l

   with "Total pattern number" = result of the grep "capture_cycle=0" report_clocking_per_pattern.rpt | wc -l

With this method, the following figures were obtained for the evaluation:

- All asynchronous reset weight = 1 (0.78%)
- SMS wrck weight = [104 / 1435] x [128 - 1] = 9 (7.19%)
- clk weight = [1331 / 1435] x [128 - 1] = 118 (92.03%)

3.3.1.3 Clock/reset weight ratio impact on test coverage

Assuming the optimal weighting per clock domain is attributed, now the goal is to validate this method and determine if this is really the optimal distribution to get the highest coverage with the minimal number of pattern.

The six experiments below show the significant influence of the clock/reset ratio on the test coverage. For all these trials, the OCC clock capture pulses were set to "2" to support both stuck-at and transition test, the pattern count was limited to 150 patterns and test coverage was evaluated for both stuck-at and transition test.

The description below details each trial and its efficiency, as results are shown in the Figure 7:

- ✓ Trial 1:
  - ✓ Experiment:

    In this first trial, one made a rough estimation of the clock ratio according to the higher amount of logic on "clk" clock domain compared to "wrck" clock domain and the assumption that a small ratio on reset can be acceptable based on the easy way to control and observe fault on reset tree.

  - ✓ Results:

    The results collected for this first trial will create a reference model for the others trials.
- ✓ Trial 2:
  - ✓ Experiment:

    This second trial checks the benefit of the previous clock weight formula and used clock ratio calculated from an ATPG run, but to trust this first comparison, the same reset ratio as trial 1 was used.

  - ✓ Results:

    Here one can clearly observed a coverage improvement compare to trial 1.

- ✓ Trial 3:
  - ✓ Experiment:

    For this third trial, as previously, the reset ratio was preserved, but "wrck" ratio has been upgraded and "clk" ratio has been downgraded compared to ATPG calculated weight of trial 2.

  - ✓ Results:

    Here one can observed a slight coverage decrease compare to trial 2 that validates the ATPG calculated distribution.
- ✓ Trial 4:
  - ✓ Experiment:

    For this forth trial, here the reset ratio has been minimized to its minimal weight of 1 and the remaining weight allocated to "clk" ratio, whereas "wrck" ratio has been maintained compared to ATPG calculated weight of trial 3.

  - ✓ Results:

    On this trial, one can remark that the reset ratio can be decreased to its minimal value of 1, without a huge impact on test coverage.
- ✓ Trial 5:
  - ✓ Experiment:

    In this fifth trial, as a minimal reset weight in trial 4 seems to provide good results in term of test coverage, the reset weight is maintained and ATPG calculated weight is applied on both "clk" and "wrck" clock domains.

  - ✓ Results:

    This time, one gets the optimal test coverage compare to all the previous trials and confirms that clock ratio must be calculated based on an ATPG run. Moreover reset weight can be configured to the minimal value of 1.
- ✓ Trial 6:
  - ✓ Experiment:

    Finally this sixth trial confirms the observations of the trial 4 and 5. Indeed, in this case, the goal is to preserve reset weight like trial 4 and 5, but increase "wrck" clock ratio and decrease "clk" clock ratio to create a deviation compared to ATPG calculated weight.

  - ✓ Results:

    Thanks to this latest check, one can finally conclude that it's mandatory to calculate clock probability based on ATPG run to get the optimal architecture.

The figure below illustrates the impact of the clock/reset probability distribution on test coverage.

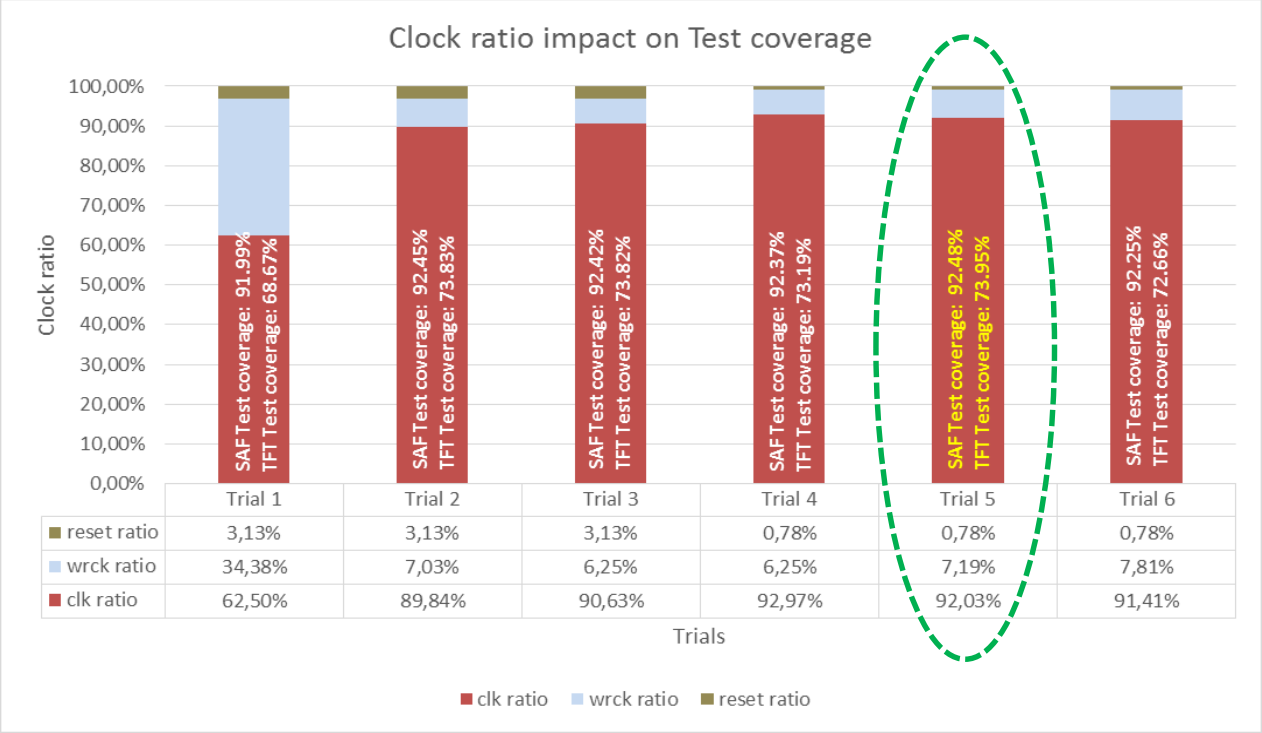

Figure 7. Clock ratio impact on test coverage

3.3.1.1 Conclusion

With this formula to calculate clock/reset capture probability distribution based on ATPG scan technics, we propose a quick and efficient solution to be implemented in the LBIST controller hardware.

This flow produces confident values for an optimal hardware clock pulse architecture which is mandatory for safety on-line self-test where run time and test coverage are critical.

3.3.2 Test Point insertion impact on test coverage

In a complex design, there are usually some uncontrollable or unobservable logics parts. Because these logics are unable or difficult to control and/or observe, it is very difficult or impossible to test them. The consequence is low test coverage. To balance such problem, Test Point Insertion is an efficient technique to improve a design's testability and improve its test coverage by adding some simple controllable and/or observable logic.

This chapter presents three technics of Test Point Insertion proposed by Synopsys and evaluates the best flow to improve test coverage with Logic BIST pseudo-random pattern generation.

The two first classic standalone flow, evaluated at the beginning, are integrated in DFT compiler tool. During DFT insertion, the tool inserts the optimal set of test points that meets the requirements, like the maximum number of test points or the maximum additional area overhead according to the selected strategy:

- "pattern_reduction" – Enables only observe points
- "testability" – Enables both control and observe points

The last flow evaluated in this section is a new methodology proposed by Synopsys. Indeed new technics arise for design that may be resistant to random patterns which means the probability of controlling some nodes randomly to a 0 or 1 value, or probability of observing some nodes to a scan register is low.

The identification of test points is done using random resistant fault analysis (RRFA) method. Controllability and observability measures of each signal in the circuit are calculated and is given a weight by measures of delta coverage gains using probability models. Based on the analysis of the data from fault simulation RRFA lists the possible candidates for the test point insertion and categorizes them as control 0/1 or observe point.

Following this new scheme, a first SpyGlass DFT ADV random resistant analysis run generates test points selections. Then this Spyglass test point selection generated report is used by DFT compiler to complete the physically-aware test point insertion during DFT insertion:

- "random_resistant" – Enables both control and observe points based on SpyGlass DFT ADV random resistant analysis report

3.3.2.1 Test Point insertion flow impact on test coverage

Regarding the test coverage drop on the module in Logic BIST compared to deterministic ATPG, the need to improve the coverage with the minimal number of patterns is clear.

The four experiments below show the significant influence of the test point on the test coverage for a given number of pattern. For all these trials, the OCC clock capture pulses were set to "2", the pattern count was limited to 150 and 1000 patterns, and test coverage was evaluated for stuck-at test only.

The description below details each trial and its efficiency shown in the next Figure 8 and Figure 9:

- ✓ Trial 1: "No test point" flow
  - ✓ Experiment:

    This first trial without Test Point flow establishes the reference figures for later comparison of the Test Point flow influence on Logic BIST efficiency.
  - ✓ Results:

This trial will be used as reference flow and results are detailed in next Figure 8 and Figure 9, for experiments respectively with 150 and 1000 patterns.

- ✓ Trial 2: "pattern-reduction" flow
  - ✓ Experiment:

In this second trial, the integration of the LBIST was done with the test point insertion flow enabled with the "pattern-reduction" option. The number of test point has been limited to 500 observe points and also 5 test points shared per test point flop.

  - ✓ Results:

Here one can clearly observed a test coverage improvement from 0.5% up to 1% compare to trial 1. In this trial, DFT compiler has inserted 263 observe points and 53 data sink registers. Results are detailed in next Figure 8 and Figure 9, for experiments respectively with 150 and 1000 patterns.

- ✓ Trial 3: "testability" flow
  - ✓ Experiment:

This third trial checks the benefit of the "testability" test point insertion flow, which should give in theory better results compared to trial 2. The number of test point has been limited to 500 observe points and 500 control points. Like the previous trial, 5 test points are shared per test point flop.

  - ✓ Results:

For this trial, one can see that this flow does not give the best result in term of coverage reached, compared to previous flow, DFT compiler has inserted 500 observe points, 500 control points and 100 data sink registers. One can observe minor difference in favor of the "pattern_reduction" flow. Results are detailed in next Figure 8 and Figure 9, for experiments respectively with 150 and 1000 patterns.

- ✓ Trial 4: "random_resistant" flow
  - ✓ Experiment:

This fourth trial checks the benefit of the "random resistant" flow divided in two steps (ref. [8]):

- Selecting Test points using SpyGlass DFT ADV feature that identifies the random resistant faults by using probabilities of detection and thus should improve fault detection on hard to test logic cones during LBIST. The test point number has been aligned also to 500 test points, like previous trials.

- Inserting Physically-Aware Test point using DFT compiler. Here again, 5 test points are shared per test point flop.

  - ✓ Results:

This last trial clearly demonstrates the real benefit of the random-resistant analysis, compared to previous flow. DFT compiler has inserted 229 observe points, 17 control points from selection report and 52 data sink registers. One can observe an improvement of more than 1% of the test coverage compared to the same design without test point and at least 0.2% compared to all previous test point flow, as shown in Table 1. Results are detailed in next Figure 8 and Figure 9, for experiments respectively with 150 and 1000 patterns.

The two figures below illustrates the efficiency of the Test Point to achieve a higher Logic BIST test coverage with a minimal run time. And finally, at least for this Test Point insertion experiment, "random_resistant" flow gives the best results with same area impact as any other flow, with the benefit to be less intrusive on timing thanks to physically aware test points insertion which considers timing information when implementing test points.

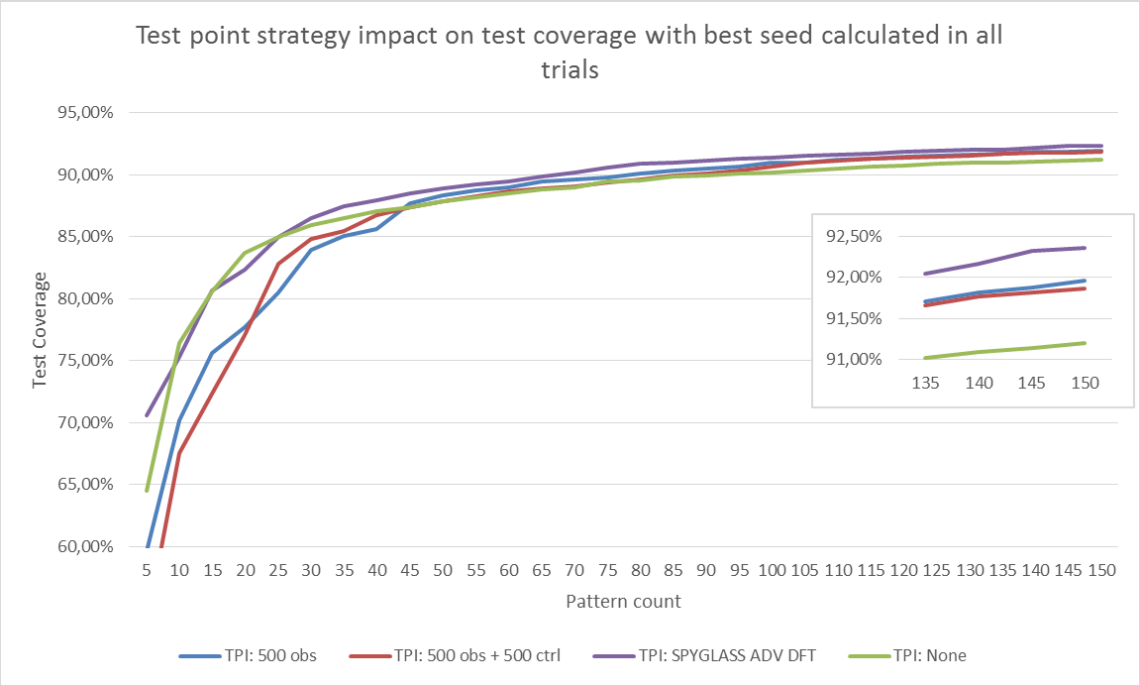

Figure 8. Test point flow impact on test coverage for 150 patterns

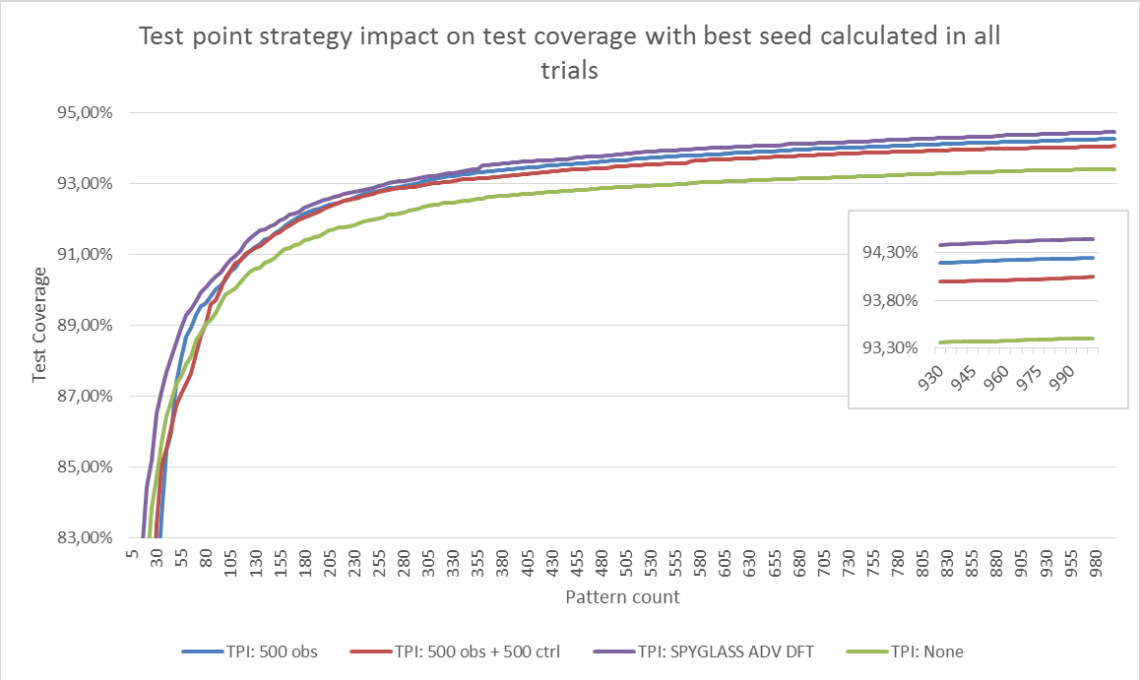

Figure 9. Test point flow impact on test coverage for 1000 patterns

3.3.2.2 Conclusion

Designs with Logic BIST exhibit random pattern resistance because of the random nature of Logic BIST test, thus leading potentially to low test coverage. To handle this, the two first evaluated test point insertion flow available in the Synopsys solution help to recover coverage loss and so pattern count, but are more adapted to ATPG deterministic test. Indeed, the result of the experiments clearly shows that "pattern-reduction" flow gives better results than "testability" flow for LBIST efficiency, whereas we could expect an inverted situation compared to the goal of each flow.

However the SpyGlass DFT ADV test point selection flow, based on a random-resistance analysis on synthesis netlist, is really promising. Indeed, as shown in Table 1, the preliminary experiment of this flow provides really good results and goes in the right way to improve Logic BIST efficiency, to achieve the highest test coverage with a minimum pattern count and so decrease Logic BIST run time. SpyGlass test point selection can be improved with some internal parameters, but this preliminary evaluation just provides some trends to tune the flow and improve the test coverage for a given amount of pattern with an estimated target coverage limit of 96.61%, fixed with the 100k patterns run.

Please refer to the user manual for more details on the SpyGlass flow (ref.[8]).

Test point strategy	**SPYGLASS ADV DFT effort option** dft_rrf_tp_count_for_cutoff_incremental_gain	**Test coverage 150 patterns**	**Test coverage 1k patterns**	**Test coverage 100k patterns**	**Test point summary**
"no test point" flow	NA	91,20%	93,40%	NA	▪ 0 test points ▪ 0 test point reg
"pattern_reduction" flow	NA	91,96%	94,25%	NA	▪ 263 test points ▪ 53 test point reg
"testability" flow	NA	91,86%	94,05%	NA	▪ 1000 test points ▪ 100 test point reg
"random_resistant" flow	10	92,36%	94,44%	96,61%	▪ 246 test points ▪ 52 test point reg
"random_resistant" flow	20	92,62%	94,83%	NA	▪ 387 test points ▪ 81 test point reg
"random_resistant" flow	30	92,84%	95,00%	NA	▪ 501 test points ▪ 102 test point reg

Table 1 . Test point strategy impact on test coverage

As a conclusion, one can claim that the SPYGLASS ADV DFT test point selection flow ("random_resistant" flow) is the most efficient and must be preferred to any others.

In case of unavailability of this feature, according to previous trials, "testability" test point flow alone clearly shows weakness for Logic BIST use in term of area overhead compared to the test coverage benefit, and thus "pattern_reduction" flow has to be used.

3.3.3 OCC capture clock pulse impact on test coverage

As mentioned in chapter 3.3.1 , it's mandatory to integrate OCC on primary input clocks at the boundary of the self-tested part and we've demonstrated the influence of the clock/reset probability distribution on test coverage. In order to share DFT hardware between classic scan ATPG and Logic BIST, the DFT architecture for scan has to reuse as much as possible the safety LBIST implementation. Hence, the reuse of the OCC seems to be clear and as ATPG scan pattern requires 4 capture clock pulses to be able to perform a write and read access inside memory, one can question what could be the influence of the capture clock pulse number on test coverage.

The chapter below detail this test coverage impact based on the ability of each OCC to generate from "1" up to "4" capture clock pulses.

3.3.3.1 OCC capture clock pulse impact on test coverage

The four experiments below show the influence of the capture clock pulses on the test coverage. For all these trials, both stuck-at test and transition test were targeted and test coverage was evaluated with respectively "1", "2", "3" and "4" capture clock pulses for stuck-at test and with respectively "2", "3" and "4" capture clock pulses for transition test.

At OCC level, the number of capture clock pulse is hardcoded by default to one with the "lbist_clk_pattern" bus tied value "4'b0001" at LBIST controller level. Thanks to DFT Compiler, specific commands to disconnect and add new connections allow through a script to make this value programmable:

- lbist_clk_pattern[3:0]=4'b0001 => 1 capture clock pulse
- lbist_clk_pattern[3:0]=4'b0011 => 2 capture clock pulses
- lbist_clk_pattern[3:0]=4'b0111 => 3 capture clock pulses
- lbist_clk_pattern[3:0]=4'b1111 => 4 capture clock pulses

The Figure 10 below shows that one can get the best coverage in both stuck-at and transition test with "3" capture clock pulses at least for this design, but the Logic BIST seed for this evaluation was fixed and extracted as best seed for a LBIST run with 2 capture pulses and 150 patterns.

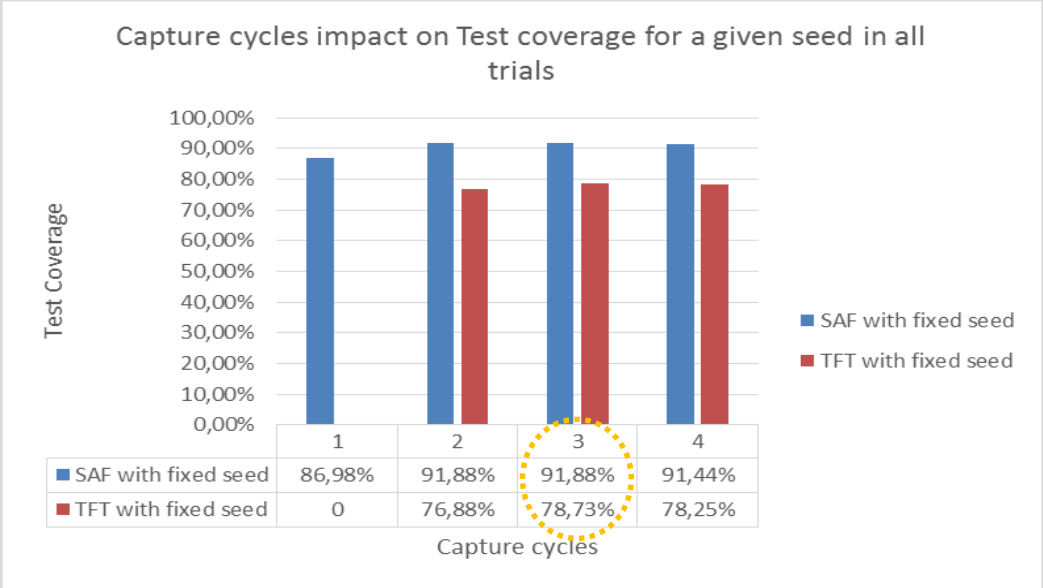

Figure 10. Capture cycles impact on test coverage with same seed for all trials

In order to be objective, the same experiment must be done with a best seed calculated in each run to provide good evidence. The next experiment uses the script provided by Synopsys to calculate the best seed over a predefined number of trials. In this case, a limit of 50 seeds has been experimented and the Figure 11 clearly demonstrates that "3" capture clock pulses for a Logic BIST run provide the best results in term of test coverage for a limited run time (150 patterns).

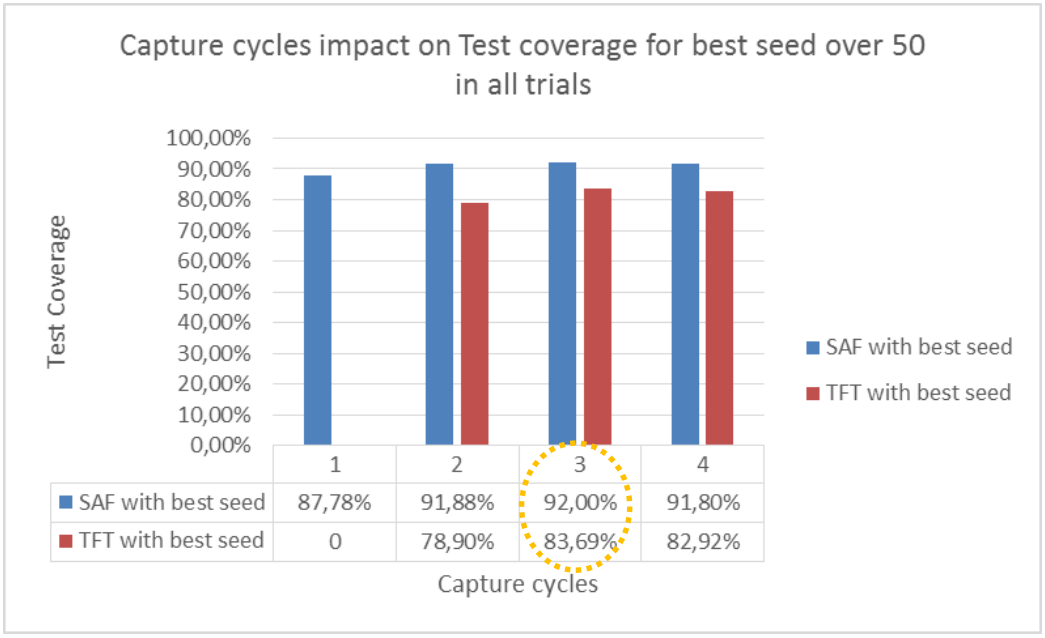

Figure 11. Capture cycle impact on test coverage with best seed calculated in all trials

3.3.3.2 Conclusion

As conclusion, we've proved that both for stuck-at test and transition test, at least a "2 capture clock pulses" configuration must be used during LBIST to get a better coverage. Moreover, as all the clocks have to remain in mission mode even during LBIST, all clocks are pulsing at-speed both stuck-at and transition tests and so it's possible to address each test at the same time with a single LBIST run. It will save a huge amount of time and one gets a better test coverage for free.

3.3.4 Logic BIST part isolation technics

To perform in-field Logic BIST on part of a design during application mode, an isolation of this sub-system is required during self-test. DFT compiler provides two isolation technics implemented during DFT insertion, that isolate the Logic BIST part from surrounding logic during self-test operation:

- Core wrapper isolation mechanism
- User-Defined Test point isolation mechanism

Whatever the mechanism used, this isolation mechanism is mandatory for the testability of the core to avoid the propagation of unknown value from the inputs to the MISR and ensure observability on the outputs to preserve test coverage.

On one hand, the core wrapping technic creates an area overhead because all core primary ports are isolated with a dedicated wrapper isolation cell, built with a flop and a multiplexer to perform the isolation mechanism. However DFT Compiler includes a "maximized-reuse" feature to minimize this area impact, when reusing the existing primary port registers to build the wrapper chain.

On the other hand, the User-defined test point technic reduces area compared to previous wrapper technology, because the isolation cell is built with a multiplexer and a shared control register for the inputs, and with a XOR tree and few shared observe registers for the outputs. Multiple test points can share a single test-point register, which reduces significantly the area and makes this solution more attractive.

However, the output isolation cells does not prevent output toggling when Logic BIST is running and this situation is unacceptable during in-field Logic BIST where surrounded logic of the LBIST part remains in mission mode. No unintended output toggling must occur!

To avoid such situation, a safe state value is required on all the outputs and this safe value implementation capability is only supported in the wrapper isolation mode. In this flow, DFT Compiler uses a specific wrapper cell that contains an additional multiplexer at its output to drive a static safe logic value, enabled by a dedicated control signal.

Thus only wrapper isolation remains available for the in-field self-test when looking for an automated solution.

The safe state isolation of the outputs will be reused also for in-field memory self-test at sub - system level. This kind of isolation at gate level allows to limit RTL intrusion for safety thing.

3.3.4.1 Experiments

All the previous analysis and results have been made with the core wrapper flow with safe state output isolation and also shared wrapping style flow to limit area overhead.

The wrapping has been done with the following configuration:

- Wrapper clock: Sub-system functional clock clk
- Wrapper shift: DFT scan enable signal
- 129 ports not wrapped:
  - All Clocks
  - All asynchronous Resets
  - LBIST control signals
  - SCAN control, access and enable signals
- 215 ports wrapped:
  - 179 input ports:
    - ✓ 90 Shared WC_S1 cell type
    - ✓ 89 Dedicated WC_D1 cell type
  - 36 output ports:
    - ✓ 32 Shared safe WC_S1_S cell type
    - ✓ 4 Dedicated safe WC_D1_S cell type

Please refer to DFT compiler User Guide to get more information on wrapper cell description (ref.[2]).

During this experiment, with about 56% of shared registers reused in the wrapper cell, the area overhead for the Logic BIST component and the scan architecture has been evaluated on a synthesis netlist with the flops already converted in scan flops and results shown in the table below (Table 2).

Module	**Area overhead (%)**
Sub-system before DFT insertion (flop already swapped to scan flop)	NA
Sub-system after DFT insertion (LBIST + SCAN components)	
LBIST controller	0.06%
LBIST decompressor	0.09%
LBIST compressor	0.13%
LBIST wrapper	0.10%
SCAN decompressor	0.06%
SCAN compressor	0.20%
DFT MISC: Lockup latch / OCC / config test mux / test decoder / TPI / …	0.50%
Total area overhead:	1.14%

Table 2 . Logic BIST area overhead overview with wrapper isolation

The global cost for both Logic BIST and SCAN hardware is therefore around 1.14% which is really acceptable in term of area overhead compared to the core duplication in a lockstep architecture.

In order to measure the extra area overhead of this core wrapping solution compared to optimal isolation with test point, an additional DFT insertion trial has been done with the same DFT configuration, except for the isolation flow, results are shown in next table below (Table 3).

Module	Area overhead (%)
Sub-system before DFT insertion (flop already swapped to scan flop)	NA
Sub-system after DFT insertion (LBIST + SCAN components)	
LBIST controller	0.07%
LBIST decompressor	0.08%
LBIST compressor	0.12%
SCAN decompressor	0.06%
SCAN compressor	0.20%
DFT MISC: Lockup latch / OCC / config test mux / test decoder / TPI /…	0.50%
Total area overhead:	1.03%

Table 3 . Logic BIST area overhead overview with Test Point isolation

The extra cost for the wrapper cell implementation compared to the test point isolation is around 0.11% which is really negligible to get an automated flow for safe state outputs.

3.3.4.2 Conclusion

Core wrapping with "shared" style and "maximize_reuse" option must be preferred to limit area overhead and ensure safe output state during in-field self-test in mission mode for logic and memory. This automated isolation flow is for sure at the moment the most reliable for safe state output isolation.

3.4 Way of improvement

This chapter presents few missing features in the evaluated Synopsys LBIST hardware (DFTMAX Logic BIST version K-2015.06-SP4) that could help the designers, but also improve its safety use.

3.4.1 Automated control of DFT signal involved in LBIST

During LBIST insertion, some mandatory control signals, like input clocks, asynchronous reset and clock gate test input pin are automatically controlled based on LBIST configuration, as follow:

- Primary input clock(s): OCC added on all sub-system input clock(s) declared
- Primary asynchronous reset(s): Automatic control logic added thanks reset type declaration
- Clock gate TE pin: Automatic control logic added thanks clock-gating control signal definition

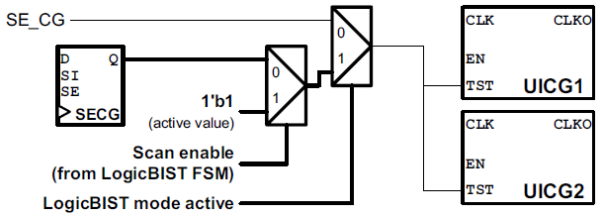

Figure 12. Clock-gating logic control during LBIST

As LBIST insertion is performed in parallel with the SCAN insertion, user defines the configuration for both mode before DFT insertion run. Unfortunately, except the previous listed signals, all others sub-system DFT control signals remain uncontrollable when LBIST mode is active. Hence, specific

interception at RTL level or gate level is required to configure properly the sub-system in self-test for this non-exhaustive list of signals:

- Test Point control: tst_testpoint_enable
- OCC control: tst_occ_reset / tst_occ_enable / tst_occ_bypass
- Memory control: tst_mem_bypass / tst_mem_power_down
- STAR Memory System control: tst_sms_dm[2:0] / sfp_dft_mode
- Test mode: tst_safe_enable / tst_wrap_enable / tst_wrap_mode / tst_compress_enable
- SCAN control: tst_bypass_rst / tst_scan_mode / …

For all the previous listed DFT signals, it would be fine to be able to collect those signals into two lists to define an automated control "to logic 1" or "to logic 0" during LBIST mode, as shown in Figure 13:

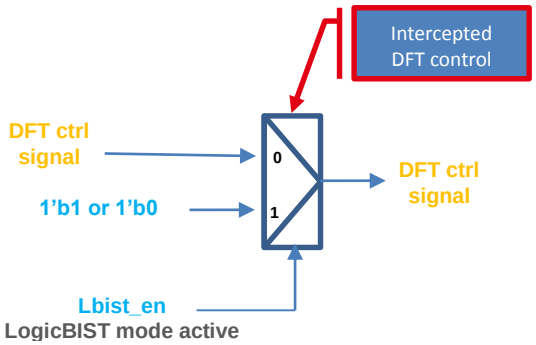

Figure 13. LBIST automated control for DFT signals

The first list would define all the signals with a required control value to '0' when LBIST mode is active. In the same way, the second list would define all the signals with a required control value to '1' when LBIST mode is active.

3.4.2 Capture clock pulse programming

As demonstrated in this paper, the control of the clock pulse number during LBIST is mandatory to improve test coverage. Thus an automated flow to get the "lbist_clk_pattern" bus value programmable like others LBIST parameters would be welcome.

3.4.3 Safety weakness around LBIST controller

Like mentioned previously, the safety control unit is a safety critical component of the system. Any feature built around this module to check its correct functioning can help making the system more reliable. The article in reference (ref.[5]) proposes to add a CRC in this part to guarantee its correctness and to improve the safety robustness. Moving back now to Logic BIST hardware, the LBIST controller and the MISR comparator are considered also as safety critical part, because untested during in-field self-test.

In a similar approach, it would be nice to implement a standard CRC within the LBIST component to ensure that all its execution is performed as planned. The input data for the CRC could come from the LBIST controller state machine, LBIST controller internal counters and MISR comparator output, observed during the LBIST run. Even if the LBIST status toggling is important for safety, it's not enough to guarantee the LBIST expected behavior. A trusted and reliable status report is required for safety use.

3.4.4 Test Point flow based on random resistive fault

A random resistant test point insertion flow will be the key point to reduce the unavailability of a device part in the system. Decreasing LBIST pattern count and so its run time will boost the deployment of the Logic BIST for safety. So the new flow based on SPYGLASS random fault analysis is welcome.

However, deterministic test point insertion could be required in parallel to achieve a higher test coverage for some specific safety critical part.

3.4.1 Switching activity evaluation and reduction

Clearly, the faster the frequency while shifting the scan chains, the shorter the test time. However, clocking and shifting data through all of the scan cells can cause much more power than during functional mode. During shift phase, all of the clock gates are enabled and most of the registers are changing state. Thus frequency must be limited to prevent IR drop and problems on remaining non tested logic, but also to avoid false positive detection.

Like scan activity, one can easily imagine that Logic BIST will create an extra activity compared to functional mode. Hence, it would be fine to get reports of the switching activity in capture and shift mode for a given LBIST run (seed defined) in the TetraMAX tool like for scan patterns. This would help to potentially improve power grid, to redefine LBIST architecture or simply decrease shift frequency.

Moreover, a mechanism to mask shift-in value on few scan segments of the random generator would be nice to reduce shift switching activity, at least as workaround solution. One can even imagine a cyclic masking of the scan chains on a pattern basis, during an LBIST run. Even if less efficient, such power reduction mechanism could help to solve struggling situations on silicon.

For capture mode, the OCC already minimizes activity (1 clock domain group active at a time) and the functional clock gating inserted during synthesis should help also to minimize the capture switching activity.

Thus the shifting phase appears as the most critical phase in term of switching activity.

4. Memory BIST implementation for functional safety

Memory cells are designed using transistors and/or capacitors, and therefore they cannot be modeled by logic gates. Structural test based on gate level netlist cannot be applied to memory testing. However, memory cores have a rather regular structure caused by identical memory cells and very simple functional operations (only read and write) which are very suitable for functional test. Unlike the case of random logic testing, which needs a large set of deterministic test patterns to reach the desired test coverage, functional test programs for embedded memory cores can be generated by compact and scalable on-chip test pattern generators. Furthermore, since written data is unaltered in a fault-free memory, the expected responses can easily be re-generated on-chip and low overhead comparison circuitry can check the correctness of output responses. Therefore, the complexity of memory BIST circuit is lower than that of logic BIST. Due to the deterministic nature and high test quality of memory test algorithms, memory BIST has emerged as the state-of-the-art practice in industry for manufacturing test since a long time.

Hence, as the memory BIST logic is already implemented for manufacturing, why not using it for in-field memory self-test also.

In this chapter, we will illustrate the use of the Synopsys STAR Memory System BIST solution to support both manufacturing test but also in-field memory self-test for safety.

In contrast to the manufacturing test, the in-field memory self-test in mission mode can only be ran on part of the design and the memory self-tested part must not corrupt the surrounding logic of the device. This is the major difference in term of memory BIST architecture.

4.1 Synopsys solution overview

The STAR Memory System BIST is a composite of infrastructure IP including, as shown in Figure 14:

- Wrappers to provide memory access during test mode
- STAR Memory System processors to perform test, diagnosis and redundancy analysis
- A STAR Memory System server to control and schedule the test, diagnosis and redundancy analysis of memories through STAR Memory System processors

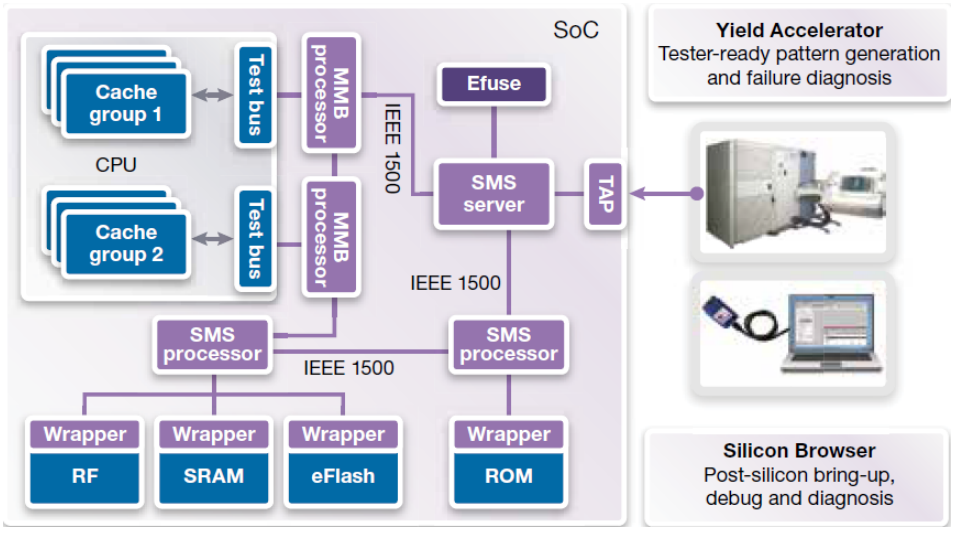

Figure 14. STAR Memory System overview

4.2 Memory BIST architecture to support in-field self-test

The STAR Memory System offers 2 types of control interface:

- JTAG interface for manufacturing test
- SMART mode interface for in-field test

At production phase JTAG TAP will be used as it gives the full access to the SMS data and instruction registers. Specifying the proper instructions sequence, the user will be able to run the desired test on the memories to comprehensively test the memories under all corners to identify all the potential failures and then generate a potential repair signature stored in an e-Fuse according detected defects. At this stage the SoC is being tested on ATE and controlled through TAP interface to shift-in setup and shift-out status.

In contrast, Smart mode is to be used on the field (application mode or so called mission mode). It is assumed that the chip is already tested and all the failures are identified and repaired. E-fuse is programmed with the golden repair signature. This is a chip normal usage. After power up to use the memories functionally these need to be repaired. Toggling just one pin (smart interface) the SMS server will initiate a predefined sequence of actions which will load the repair data from e-fuse into memories and perform other checks to make sure the repair has been successful.

Why limiting this SMART mode to start-up, one can imagine to launch BIST test also during runtime mode only on some part of the design?

4.2.1 Memory BIST architecture to support design constraints

The coming section will describe an optimal STAR Memory System architecture able to support both production test and in-field self-test without any compromise on efficiency and quality.

The STAR Memory System gives the flexibility of defining various groups of SMS processors connected to the TOP level Server through an IEEE 1500 standard serial link, as shown in Figure 15.

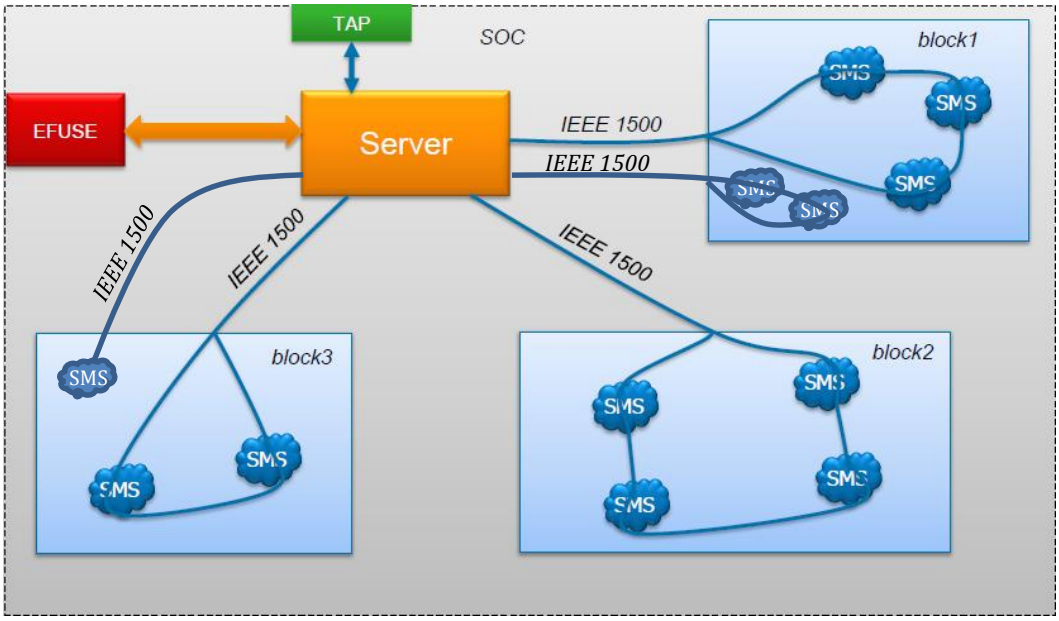

Figure 15. STAR Memory System network in hierarchical flow: ring approach

Due to SOC design integration constraints where sub-systems are delivered to the SOC and developed in parallel, a hierarchical flow must be used to apply an IP oriented approach where sub-systems get an embedded autonomous MBIST solution.

Moreover, from top level point of view, SOC design integration must get a frozen interface for each sub-system whatever the amount of memories embedded and whatever the SMS architecture implemented internally. Thus the latest design constraint comes from this fix number of IEEE 1500 interface required between SOC top level server and the SMS group implemented inside the IP. Indeed, each SMS processor group in the IP requires a dedicated IEEE 1500 link to the top level server, if implemented in the same way as Figure 15. This is not acceptable at SOC level in case of late change of SMS groups in a sub-system that will impact the interface. So a predictable number of IEEE 1500 interface is required at the beginning of the project during the partitioning of the system.

Fortunately, the STAR Memory System gives also the flexibility to implement locally inside a sub-system a dedicated sub-server, limiting the IEEE 1500 interface for Memory BIST to one single interface without any concession on memory BIST feature, which will ease also IP reuse for next SOC.

So the previous architecture presented in Figure 15 can evolved to a local star architecture where SMS processors are connected to a local sub-server within the sub-system, as shown in Figure 16.

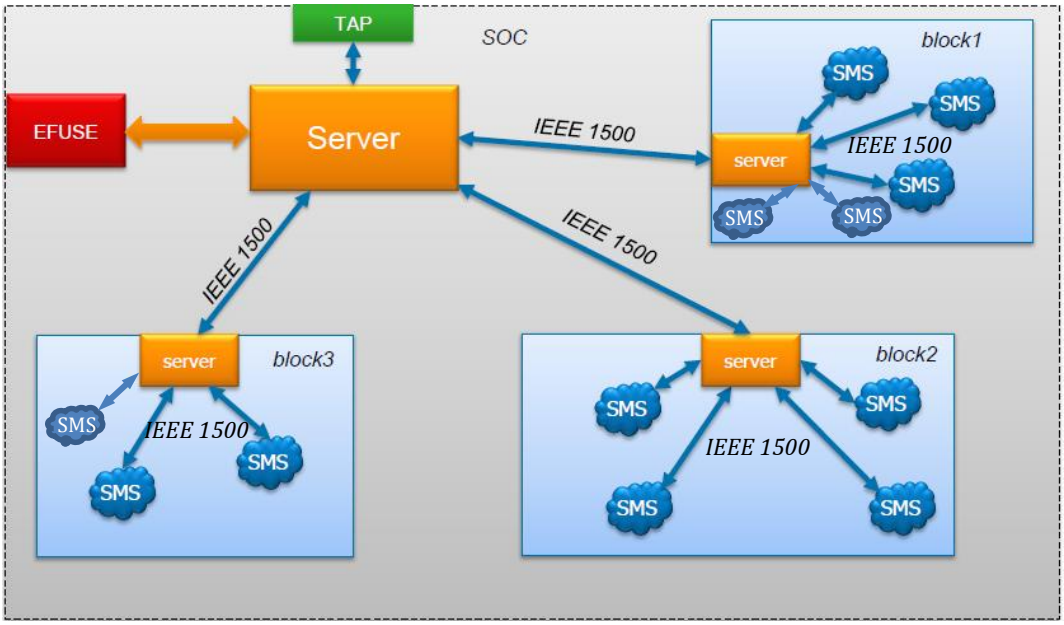

Figure 16. STAR Memory System network in hierarchical flow: star approach

Such architecture can also be relevant for functional safety use also to perform an in-field memory BIST only on a dedicated sub-system part thanks to the SMART mode interface available at sub-server level.

4.2.2 Memory BIST architecture to support in-field self-test constraints

As the overall Memory BIST architecture is now defined in term of networking, let's come back to this SMART mode interface, used usually to run memory test during chip power-up. Based on the SMS architecture proposed, we've got now the freedom to launch in-field memory self-test on some parts of the chip. However the safety control unit must be able to schedule the memory BIST run within a sub-system to minimize peak power. By default, the server integrates a predefined BIST scheduling sequence during SMART mode configured by the user during server generation. Once generated, the scheduling will be hardwired in the server logic and will always function in the same order. This predefined solution is really binding for an in-field test use, where the system constraints and the time allocated for safety checks is known late in the IC development process.

Luckily, the required flexibility is given by the "ring_bypass" control bus at sub-server/server level.

The “ring_bypass” bus will exclude the SMS processor group corresponding to the “ring_bypass” active high bit from the BIST execution. Thus, to get the maximum flexibility for BIST run scheduling, no default scheduling must be predefined in the server configuration, all SMS groups can run in parallel, but any of them can be excluded from the BIST run with this “ring_bypass” control bus. Thus scheduling is now software dependent at system level and no more hardwired.

The SMART mode sequence is initiated with a high pulse on the smart input pin of the server/sub-server, then four main modes are available from system level, controlled with dedicated pins:

- udr_load: If high, SMART state machine reads e-fuse user data and load it into user data register. This feature is useless for safety use and concerns only top level server.
- bihr_run: If high, SMART state machine reads e-fuse container, decompresses repair signature and shifts it into memories redundancy register. This mode is the mandatory default power-up mode for an IC to repair memory (yield recovery) and concerns only top level server (e-Fuse shared at top level).
- bist_run: If high, SMART state machine performs memory BIST test according grouping predefined during server generation and also according “ring_bypass” control bus configuration, as described previously. For in-field self-test, this control signal available at server level and sub-server level allows a perfect control of the memory BIST for safety purpose.
- bisr_run: if high, SMART state machine tests memories, calculates soft repair signature, repairs memories and executes final memory BIST. This feature can’t be used in the field, because it could lead to non-deterministic behavior.

In addition to the previous “bist_run” control signal, the “ring_bypass” control bus allows the exclusion of SMS processors from a BIST run and thus increases the control granularity of the memory BIST run time according to system constraints and free time allocated for self-test during application mode.

Server/Sub-Server SMART mode interface provides additionally three status signals to inform the system:

- spf_fail: If high, it informs the system that memory BIST status is KO and at least one memory got a defect.
- spf_ready: If high, it indicates to the system that no SMART mode is active at this time and SMART mode state machine is ready for any SMART mode function.
- udr_rdy: If low, it informs the system that user data e-Fuse extraction is on-going.

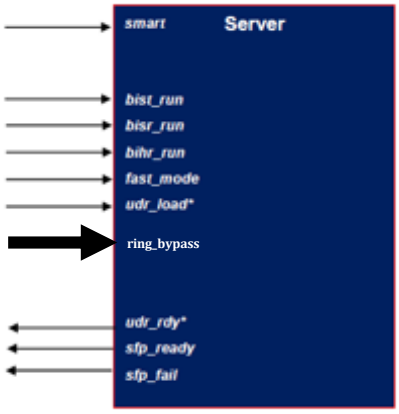

Figure 17. STAR Memory System server SMART mode interface

For in-field self-test usage, the SMART mode interface of all sub-server must be propagated to the chip safety controller component. Scheduling will be organized at this central level by software to launch on demand self-test during sub-system idle time.

Like for Logic BIST on part of the system, for memory BIST also, an isolation of the self-tested part is required and in this case it's planned to reuse the existing Logic BIST isolation. Thus there's no corruption of the surrounding logic during BIST, a safe state will be applied on all the sub-system outputs during self-test.

4.3 SMS BIST Experiments and results

The STAR Memory System BIST architecture has been defined according to the previous chapter to support in-field self-test. For this trial, the plan was to evaluate both LBIST and MBIST solution for the same sub-system and check also the memory BIST solution control at top level, to evaluate the memory repair solution. Thus the same sub-system was used for MBIST insertion at RTL level and then LBIST insertion at gate level, but the MBIST insertion flow has been also evaluated on the design top level to check the hierarchical flow.

Hence, the memory BIST insertion at sub-system level is the first step of safety customisation to sustain the online in-field self-test solution proposed.

4.3.1 Sub-system SMS BIST architecture overview

Even if the experiment of the SMS BIST solution has been evaluated on a full design, following the previous presented architecture, the discussion will be limited to a sub-system including an SMS BIST, which is the single relevant part for online in-field self-test.

The design is an image processing module with a single clock domain "clk", consisting of 38, 2 k-flops and 23 memories:

- 16 * memories ram 1200 words of 144 bits
- 2 * memories ram_1920 words of 76 bits
- 2 * memories ram_1920 words of 77 bits
- 2 * memories ram_1920 words of 126 bits
- 1 * memories ram_1920 words of 108 bits

The SMS BIST architecture has been defined in the following way:

- 1 local sub-server
- 3 SMS processors
- 23 SMS memory wrappers

The SMS memory wrappers were distributed as follow:

- 1 SMS processor 0 to control the first 8 memories 1200x144
- 1 SMS processor 1 to control the next 8 memories 1200x144
- 1 SMS processor 2 to control the 5 remaining memories

This architecture is fully in line to support both online in-field self-test and manufacturing test.

4.3.2 SMS BIST evaluation on a sub-system

During this experiment, the area overhead for the SMS BIST component and the impact on test coverage has been evaluated and results are shown in the table below (Table 4).

Module	Instance Number	Area overhead (%)
Sub-system without SMS BIST insertion	NA	NA
Sub-system with SMS BIST insertion		
SMS BIST sub-server	1	0.06%
SMS BIST processor 0	1	0.17%
SMS BIST wrapper ram 1200x144	8	0.04%
SMS BIST processor 1	1	0.17%
SMS BIST wrapper ram 1200x144	8	0.04%
SMS BIST processor 2	1	0.19%
SMS BIST wrapper ram 1920x76	2	0.03%
SMS BIST wrapper ram 1920x77	2	0.03%
SMS BIST wrapper ram 1920x126	2	0.04%
SMS BIST wrapper ram 1920x108	1	0.03%
	Total area overhead:	1.40%

Table 4 . Sub-system SMS BIST area overhead overview

The global cost for SMS BIST solution is therefore around 1.40% of area overhead, which is really acceptable. If we define the optimal SMS architecture in term of area with only a sub-server to support design integration constraint and a single processor to test all memory in parallel (at least 1 processor per clock domain), one can extrapolate the area overhead of the safety architecture with the additional SMS BIST processors inserted, to 0.19% per processor. Indeed, the main targets of the online in-field self-test are the power peak consumption due to SMS BIST extra activity compared to functional mode and the self-test run time that must be integrated inside free time frame of the system. To play on these 2 parameters, the only freedom allowed at user side is the processor distribution, which means the number of memories allocated to a dedicated processor and that will be tested in parallel during self-test.

Thus to limit IR drop, in the experiment, a maximum of 8 memories can be tested in parallel and to get some flexibility on the run time, memories are also grouped per number of words which will dimension for sure the BIST execution according to memory clock frequency.

In the presented SMS BIST architecture, we've got the freedom to launch in parallel from 1 up to 3 SMS processors, which means 7 up to 23 memories tested in parallel.

Note that the presented architecture doesn't contain any memory redundancy feature, this additional feature has been evaluated in a second step, but it's not relevant for the in-field safety usage.

In this experiment, the impact on the sub-system test coverage has been evaluated and no difference has been observed as soon as all the SMS logic can be properly controlled during scan or LBIST. No special action need be taken on this side!

4.4 Way of improvement

This chapter presents few missing features in the evaluated Synopsys Self-Test and Repair Memory System hardware (STAR Memory System, version K-2015.09) that could help the designers, but also improve its safety use.

4.4.1 Memory BIST run time and power calculation for processors

Like for the Logic BIST, the dimensioning of the MBIST run time is a key point for safety use. At the opposite of the LBIST where a run can be performed for a given number of pattern and incremental run is feasible with reseeding from one LBIST run to another. Nothing similar is feasible for a memory test and one is forced to wait the end of a memory BIST run to reach the coverage. No incremental run is feasible. So the memory processor grouping is mandatory according system requirements and power activity. To ease this memory grouping per processor, it would be fine to get an automatic computation of a SMS processor run time per memory group for a given MBIST architecture, based on memory clock frequency.

Currently, grouping is just made based on memory size, physical placement and functional clock frequency.

In parallel it would be fine to get an estimation of the power consumption during MBIST to limit IR drop.

4.4.2 Safety weakness around SMS BIST server status flag

Like mentioned previously for Logic BIST, the MBIST server is considered also as safety critical part, even if this component is tested during in-field Logic BIST, there's no protection against transient fault during MBIST run.

In a similar approach, it would be fine to implement a standard CRC within the SMS BIST server component to ensure that all its execution is performed as planned. The input data for the CRC could come from the SMS server SMART mode state machine and all memory BIST processor runtime are compliant with the expected runtime. Even if the Memory BIST status "sfp_ready" and "sfp_fail" toggling is required for safety, it's not enough to guarantee the MBIST expected behavior. A trusted and reliable status report is required for safety use.

From safety permanent fault detection on all SMS BIST modules, there's no problem as memory BIST infrastructure is part of the online Logic BIST detection.

4.4.3 Reduce runtime for SMART mode BIST test

By default, an exhaustive memory BIST algorithm is implemented for manufacturing in each SMS processor to cover all potential type of faults in the memory. Even if the implemented SMS BIST algorithm can be selected by user configuration, it would be welcome to be able to define two sets of algorithm.

Indeed, for in-field self-test, one can distinguish 2 types of phases in the system:

- On one side, the power-up phase, where the system can have a large amount of time to wake up. In this phase, one can imagine to have an exhaustive test of the memories. For this step, a long runtime could be acceptable and is not intrusive for the system. Thus the classic memory BIST algorithm used for manufacturing are perfect to perform the sanity check of the system.
- On the other side, the idle mode on sub-part of the system, where the sub-system can be unavailable only for a limited time frame. This free time frame must be as short as possible and in such situation an exhaustive memory BIST algorithm can be too long for large embedded memory and make this memory self-test solution non pertinent.

One can see that memory BIST duration can be a brake to the use of this solution for memory permanent fault detection. However, the combination of ECC protection and limited memory BIST algorithm (small run time) could represent a back door to such runtime issue. Indeed, the memory BIST does not prevent transient fault aggression on memory and a mandatory ECC protection is required on critical memories. The ECC solution cover both transient and permanent fault, but unfortunately it just protect the data inside the memory, but not the memory address decoder. Thus to get a robust hardware from safety point of view, the data and the corresponding ECC bits must be stored in 2 different memories to detect any defect in the address decoder.

Hence, to avoid the duplication of memories for robust ECC detection against permanent fault, a smaller run time memory BIST algorithm would be welcome to solve this permanent error detection around address decoder with a minor area overhead, because Memory BIST hardware is already implemented for manufacturing. If ECC is implemented just for permanent fault detection and don't care about transient fault, here this methods would save area.

Based on previous statement, it would be fine to get various algorithm (at least 2) embedded in the SMS processor, that could be selected through the SMART mode interface. Hence the use of the correct algorithm could be managed by the safety controller component according system constraints according allocated time for self-test. Flexibility is required on this side to avoid hardwired solution without any fallback provision.

4.4.1 Minor drawbacks observed during the SMS BIST evaluation

This final part just list minor drawbacks reported to Synopsys after the evaluation:

- The first point concerns the SMS BIST frequency ratio of 8 required between slow clock and fast clock. This type of limitation is not acceptable at system power-up, where at speed clock is not available before PLL lock, but system can decide to launch a memory BIST using an internal oscillator clock.
- The last point concerns the SMART mode interface misalignment which differs in case of sub-system including redundant memory and other sub-system without redundant memory. During the experiment, the architecture has been defined with a sub-server located in each sub-system and a single shared e-Fuse placed at the SOC top level to store memory repair information. Unfortunately, during the trial with redundant memory, we've observed a misalignment for the SMART mode interface, as follow:
  - ✓ No redundancy => SMART mode interface placed at sub-system server level
  - ✓ With redundancy and shared e-Fuse => SMART mode interface placed at TOP server level

  This misalignment of control even if manageable is a real drawback in term of safety control architecture consistency.

5. Conclusions

With increasing use of electronic systems in vehicles for managing critical functionality, the requirements for safety in automotive chip architecture is becoming paramount. Synopsys test solution already offers all required key components for safety-critical automotive ICs, including advanced manufacturing test and in-field self-test, to ensure compliance with the ISO 26262 functional safety standard.

This paper describes Build-In Self-Test implementation for logic and memories in an automotive SOC, based on Synopsys solution. It demonstrates that Memory BIST and Logic BIST technics are well adapted to sustain safety requirements with a small compromise on the area side. It is shown how to define an optimal implementation for in-field logic self-test and we've proposed an efficient Memory BIST architecture to sustain also in-field memory self-test, both to support the requirements of in system test and achieve the best test coverage with a minimal run time.

However, even if Synopsys proposes performant solutions to address functional safety fault detection, some improvement have to be done to ease in-field self-test use. Indeed, an exhaustive Memory BIST run time is quite long for large memory outside IC power up phase and the automation level of Logic BIST integration must be improved to ease designer workload, like for classic scan insertion (report of switching activity, control of DFT signal, ATPG debug). All these technics comes from manufacturing to achieve a high reliability, but must be improved for in-field safety use.

Hence it is shown how these BIST techniques have to be augmented to better support the functional safety requirements of in-field self-test.

6. References

[1] DFTMAX Logic BIST User Guide, version K-2015,06-SP4

[2] DFT Compiler, DFTMAX, and DFTMAX Ultra User Guide, version K-2015.06-SP4

[3] TetraMAX User Guide, version K-2015,06-SP4

[4] Improving fault coverage for random-pattern resistant designs - Abhishek Mahajan - December 03, 2015

[5] Adding CRC to BIST improves SOC safety & reliability - Shreya Singh, Abhinav Gaur & Amit Pal - April 23, 2014

[6] Logic BIST: State-of-the-Art and Open Problem - Nan Li, Gunnar Carlsson, Elena Dubrova & Kim Petersen - March 16, 2015

[7] New approach moves logic BIST into mainstream - R. Kapur, R. Chandramouli & T.W. Williams - October 14, 2002

[8] Beta documentation Physically-Aware Test Points selection and synthesis, version L-2016.03

[9] DesignWare Self-Test and Repair (STAR) Memory System, version K-2015.09