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

Christophe Eychenne

Robert Bosch SAS

June 25, 2016

SNUG Munich

Agenda

Built-In Self-Test to serve functional safety needs

Logic BIST implementation for functional safety

Memory BIST implementation for functional safety

Conclusion

Built-In Self-Test to serve functional safety needs

Introduction

BIST to serve functional safety needs

- Status
  - A typical car today relies on electronics for controlling safety-critical components such as engine, transmission, air bags, steering, and braking system.
  - As such safety-critical devices in the system must be able to routinely perform in-field self-test to verifying that the circuitry operates without structural defects !
  - Available technics to detect defects :
    - Redundant Critical hardware module implemented in lock-step of each other, processing the same inputs
      - ✓ Most efficient option, but limited due to the duplicated area penalty !
    - Error Correcting Code (ECC) and Cyclic Redundancy Check (CRC) mechanism
      - ✓ Efficiency limited to preserve data integrity within the system.
    - Built-in Self-Test (BIST) to detect any structural hardware fault due to aging.
      - ✓ Efficient option to detect permanent defect in logic and memories using already existing test manufacturing hardware with a minimal area overhead !

BIST to serve functional safety needs

- Built-In self-test represents a cheap answer to detect permanent faults on logic and memories, at power up and for cyclical runs.

- Challenges for in-field self-test use:
  - Increase the fault detection of Logic BIST (pseudo-random pattern generation)
  - Control self-test Runtime to minimize unavailability of the tested part for both MBIST and LBIST
  - Control switching activity to avoid disturbing remaining functional parts of the design

- How to convert these DFT hardware structures to serve functional safety needs ?
  - With an optimal Logic BIST implementation to improve fault detection
  - With a specific Memory BIST architecture to control self-test runtime

Logic BIST implementation for functional safety

Implementation experiments

L-BIST implementation for functional safety

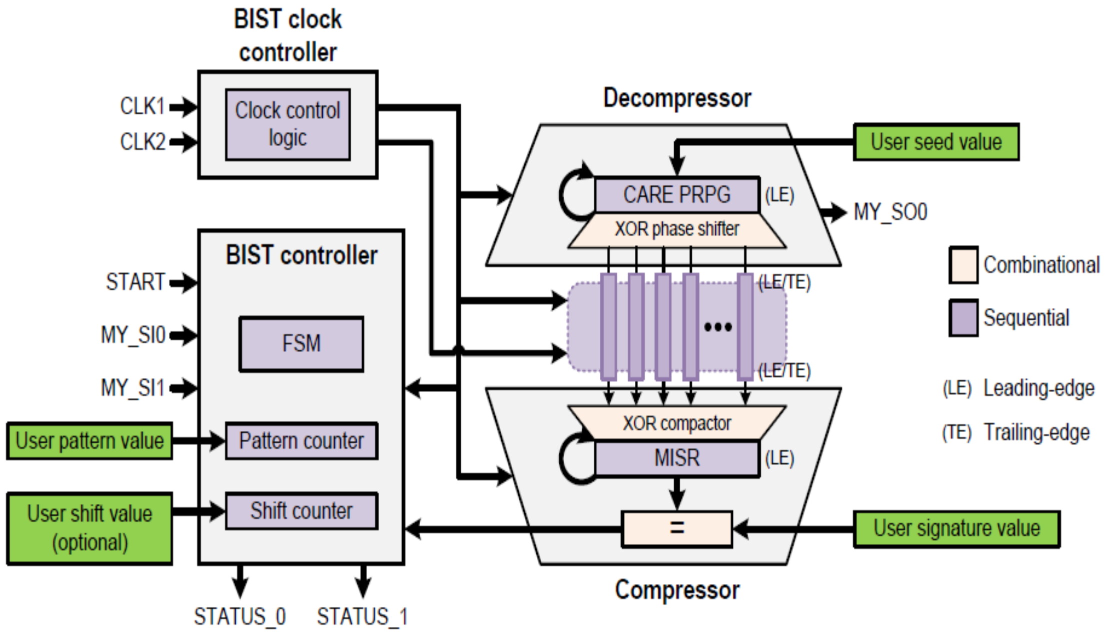
Targets and constraints

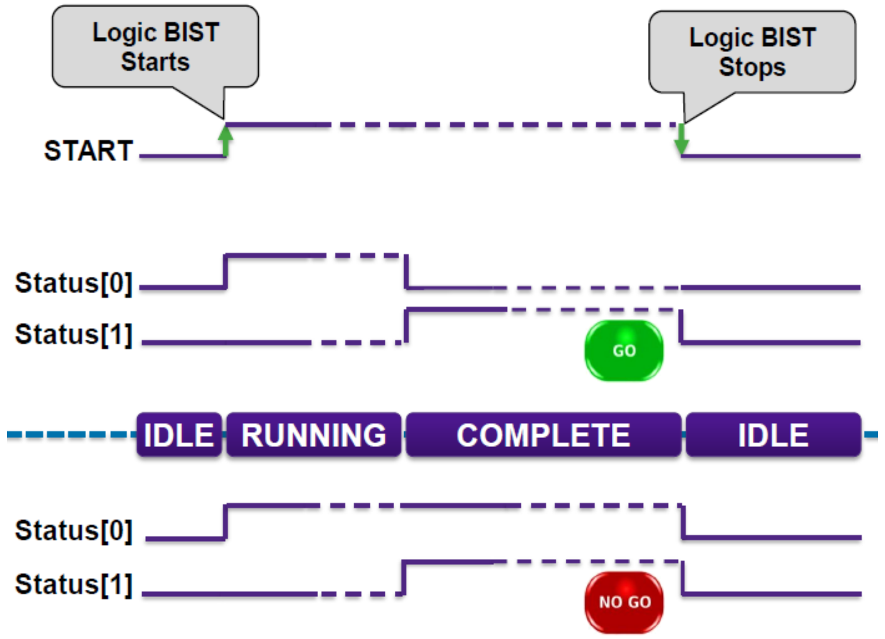
- L-BIST architecture must serve the following topics:
  - LBIST integration inside sub-system
    - ⇒ Architecture and tools to support hierarchical design flow
    - ⇒ Shared SCAN infrastructure as much as possible to minimize area overhead
  - Safety usage at system power up and cyclic on the fly in-field self-test execution on part of the system without disturbing rest of the design:
    - ⇒ Architecture to support local Logic BIST run on a specific part of the system in mission mode
    - ⇒ Core boundary isolation to preserve test coverage
    - ⇒ Safe state output during BIST to avoid disturbing boundary logic remaining in mission mode
    - ⇒ Local clock and reset control at sub-system level
    - ⇒ Flexible run time to integrate LBIST during idle mode of the sub-system
    - ⇒ Control switching activity compared to mission mode to limit peak current
  - No use during production test

L-BIST implementation for functional safety

Synopsys DFTMAX Logic BIST overview : synthesis-based solution

- It consists of an LBIST controller, a PRPG, a MISR and a signature comparator





- The self-test is initiated by simply holding the START signal high, and results can be observed on status outputs:
  - Status[0]: PASS/FAIL status
  - Status[1]: LBIST done

L-BIST implementation for functional safety

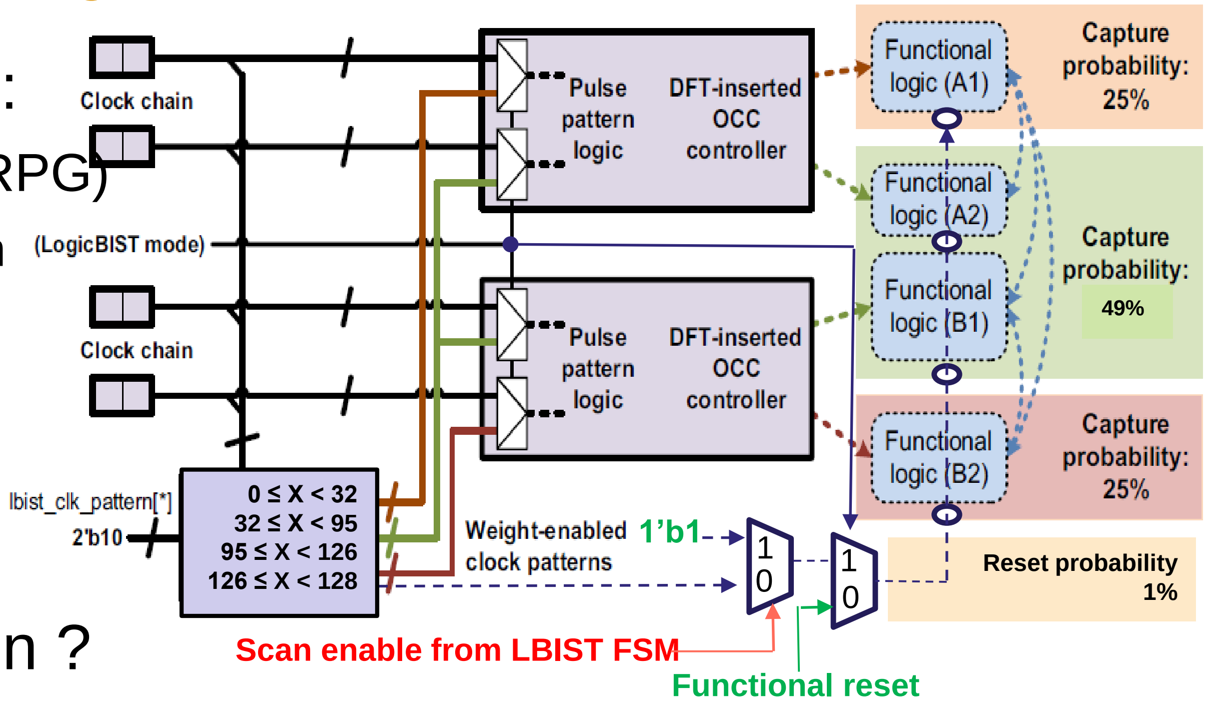
L-BIST implementation to get the best coverage for a minimal runtime

- Based on the L-BIST Synopsys solution, the goal was now to serve the safety needs with the existing proposed hardware:
  - Get the best test coverage, with a minimum pattern count to minimize unavailability of the tested sub-system
  - Get a correct isolation, to avoid disturbing remaining part of the design in mission mode
- Following part investigates on L-BIST architecture to achieve the previous constraints:
  - Test coverage improvement:
    - Clock ratio: OCC weighted capture probability attribution for clocks and asynchronous resets
    - Capture clock pulse number
    - Test point insertion flow: “pattern_reduction” vs “testability” vs “random resistant analysis"
  - Sub-system isolation with safe state output thanks “core wrapping”

L-BIST implementation for functional safety

L-BIST implementation to get the best coverage for a minimal runtime

- LBIST clock control thanks to local OCC:
  - No more deterministic clock pulse control (PRPG)
    - LBIST uses a clock pulse selection mechanism based on clock group's capture probability
    - Each group's capture probability is relative to the sum of all clock and reset group weights (128)



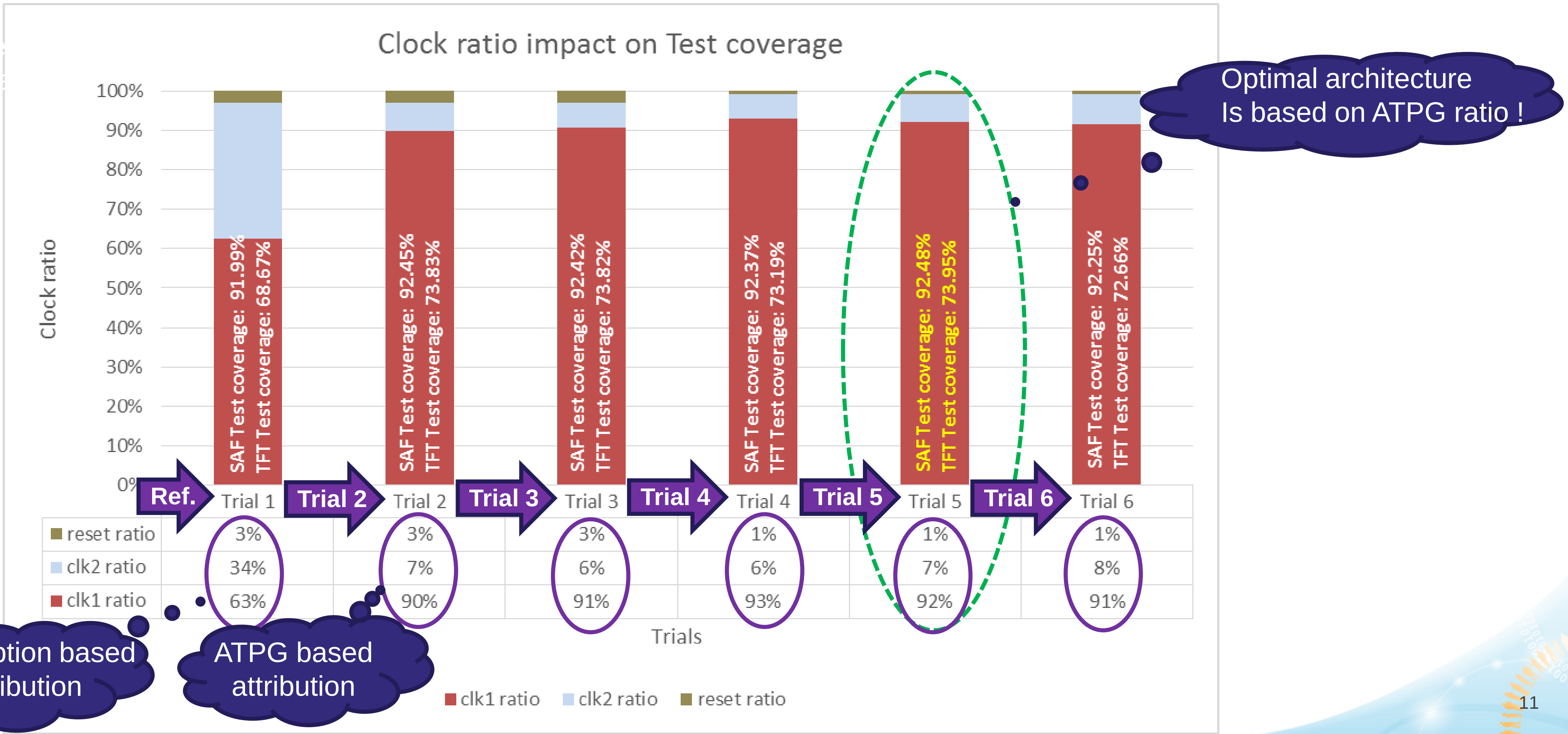
- How to define this clock weight attribution ?
  1. Perform first a basic stuck-at ATPG run using OCC and report pattern clocking
  2. Count occurrence of each clock domain for all pattern in the previous ATPG report
  3. All asynchronous reset are controlled together and require a low probability to be checked compared to clock pulsing
  4. Clock weight capture probability based on ATPG results:

Clock weight = [clock pat number / Total pattern number] x [128 counter value - async reset weight]

L-BIST implementation for functional safety

L-BIST implementation to get the best coverage for a minimal runtime

- Clock weight impact on test coverage: SAF / TFT - 2 capture clock pulses - 150 patterns
  - 2 clock domains (clk1/clk2) + 2 asynchronous reset : 95% of the logic on clk1 clock domain

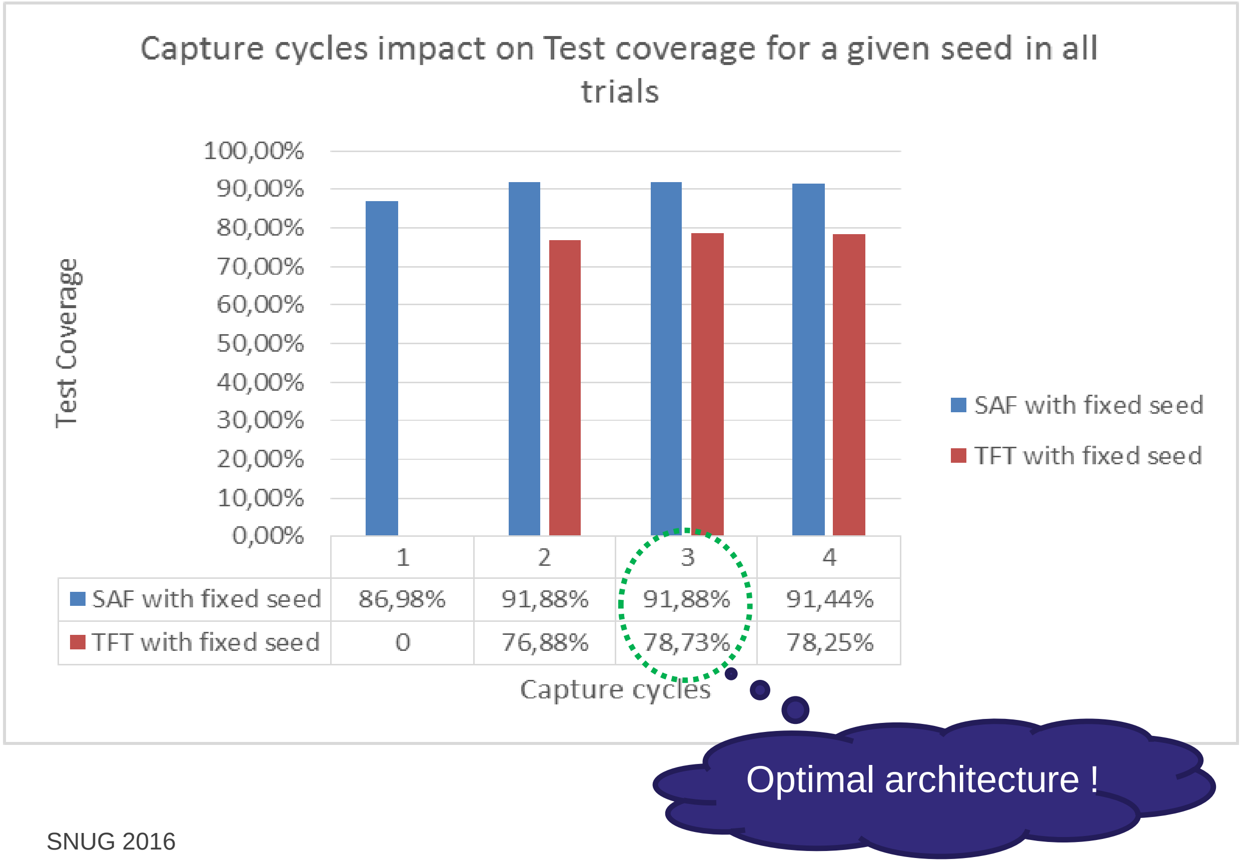


	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5	Trial 6
🟫 reset ratio	3%	3%	3%	1%	1%	1%
🟦 clk2 ratio	34%	7%	6%	6%	7%	8%
🟥 clk1 ratio	63%	90%	91%	93%	92%	91%

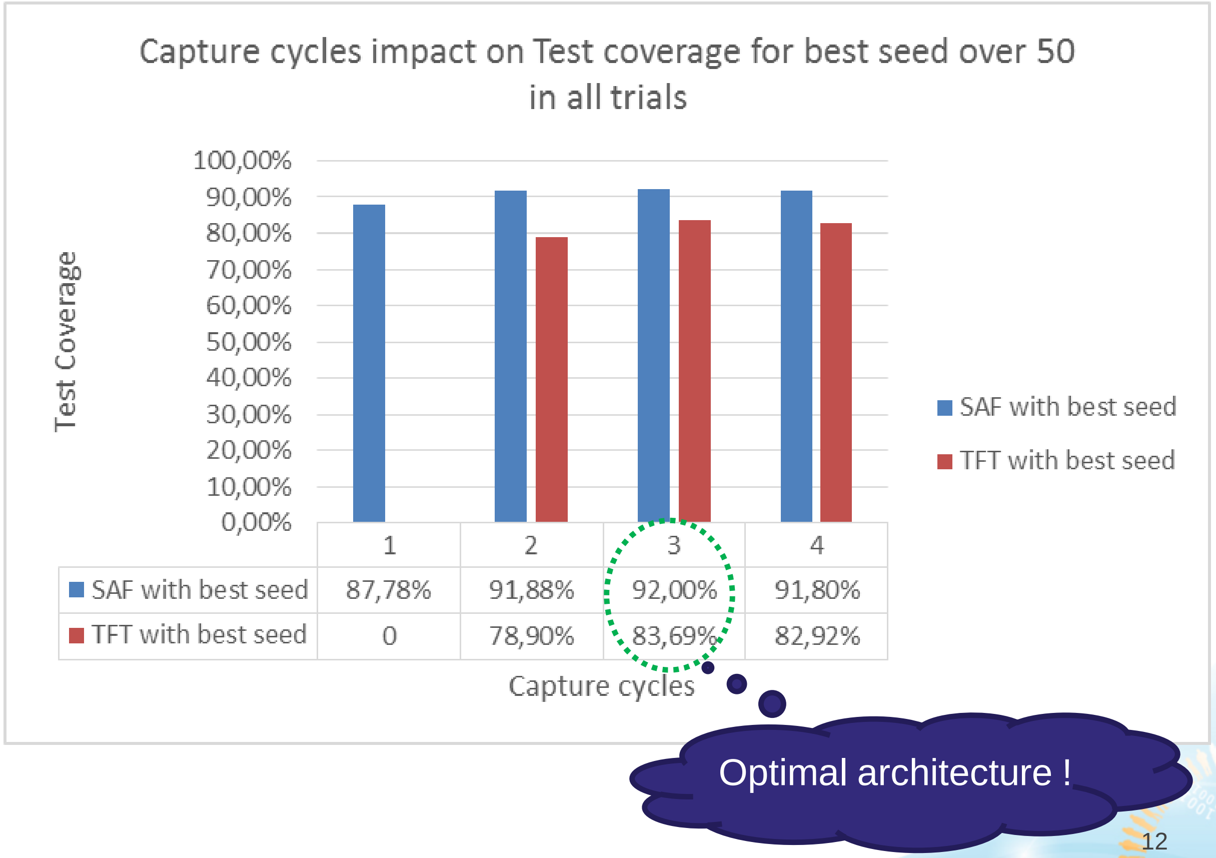
L-BIST implementation for functional safety

L-BIST implementation to get the best coverage for a minimal runtime

- Capture clock pulse number impact on test coverage: SAF / TFT - 150 patterns
  - OCC architecture already defined with 4 clock bit chain to support scan ATPG RAM access
  - Setup of LBIST capture clock pulse thanks lbist_clk_pattern[3:0] bus from 1 to 4 capture clock pulses



	1	2	3	4
🟦 SAF with fixed seed	86,98%	91,88%	91,88%	91,44%
🟥 TFT with fixed seed	0	76,88%	78,73%	78,25%



	1	2	3	4
🟦 SAF with best seed	87,78%	91,88%	92,00%	91,80%
🟥 TFT with best seed	0	78,90%	83,69%	82,92%

L-BIST implementation for functional safety

L-BIST implementation to get the best coverage for a minimal runtime

- Test point type impact on test coverage: SAF – 2 capture clock pulses
  - 3 types of test point flow available in the tool:
    - "pattern_reduction" : Only observe point added
      DFT compiler : **500 observe / 0 control** test points max + 5 test points shared per test point register
    - "testability" : Control and observe point added
      DFT compiler : **500 observe** / **500 control** test points max + 5 test points shared per test point register
    - "random_resistant" : SpyGlass ADV DFT random resistant analysis report used to insert control and observe
      DFT compiler: **500 observe** / **500 control** test points max + 5 test points per test point register
      Spyglass ADV DFT : dft_pattern_count = 500 + dft_rrf_tp_count = **1000** + dft_rrf_tp_count_for_cutoff_incremental_gain = 10

Test point strategy	Test coverage 150 patterns	Test coverage 1k patterns	Test coverage 100k patterns	Test point summary	
"no test point" flow	91,20%	93,40%	NA	- 0 test points - 0 test point reg	Reference
"pattern_reduction" flow	91,96%	94,25%	NA	- 263 test points - 53 test point reg	++ Area + Coverage
"testability" flow	91,86%	94,05%	NA	- 1000 test points - 100 test point reg	- Area + Coverage
"random_resistant" flow	92,36%	94,44%	96,61%	- 246 test points - 52 test point reg	++ Area ++ Coverage

L-BIST implementation for functional safety

L-BIST implementation conclusion

- Using the proposed implementation, Synopsys LBIST is a performant solution:
  - Test point insertion based on random resistant fault analysis is the right approach to minimize test time without compromise on the quality and also push back the current test coverage limit

SPYGLASS ADV DFT effort option	Test coverage 150 patterns	Test coverage 1k patterns	Test coverage 100k patterns	Test point summary
dft_pattern_count 500 / dft_rrf_cutoff_inc_gain 10	92,36%	94,44%	96,61%	246 test points & 52 test point reg
dft_pattern_count 500 / dft_rrf_cutoff_inc_gain 30	92,84%	95,00%	NA	501 test points & 102 test point reg
dft_pattern_count 500 / dft_rrf_cutoff_inc_gain 50	92,98%	95,05%	NA	673 test points & 138 test point reg
dft_pattern_count 500 / dft_rrf_cutoff_inc_gain 80	93,34%	95,34%	NA	990 test points & 201 test point reg
dft_pattern_count 150 / dft_rrf_cutoff_inc_gain 80	93,76%	95,84%	96.88%	1807 test points & 364 test point reg

- LBIST execution run time is one of the major concerns now for an on the fly safety use
- LBIST switching activity is also one of the other concerns to avoid disturbing remaining mission mode parts

Memory BIST implementation for functional safety

Architecture experiments

MBIST implementation for functional safety

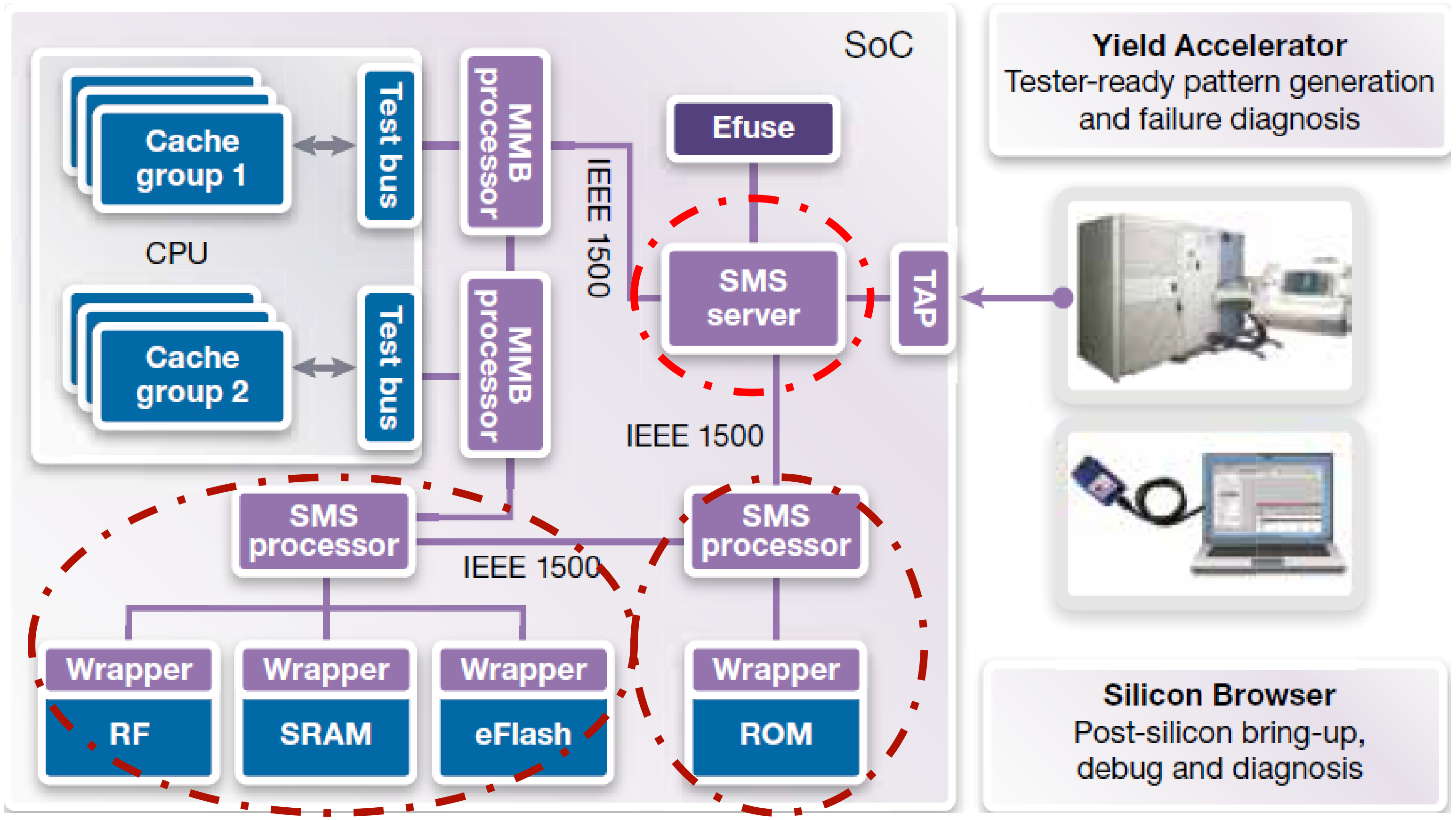
Targets and constraints

- MBIST architecture must serve the following topics:
  - Non equal sub-system maturity that should not impact the overall system
    - ⇒ Architecture to support hierarchical design flow
  - Safety usage at system power up and cyclic on the fly BIST run execution on part of the system
    - ⇒ Architecture to support local Memory BIST run on a specific part of the system in mission mode
    - ⇒ Safe state output during BIST to avoid disturbing boundary logic remaining in mission mode
    - ⇒ Flexible run time to integrate Memory BIST during idle mode of the sub-system
    - ⇒ Control switching activity compared to mission mode to limit peak current
  - Production test pattern
    - ⇒ Architecture to support tester constraints
    - ⇒ Memory repair mechanism
    - ⇒ Failure diagnosis

MBIST implementation for functional safety

Synopsys START Memory System (SMS) overview

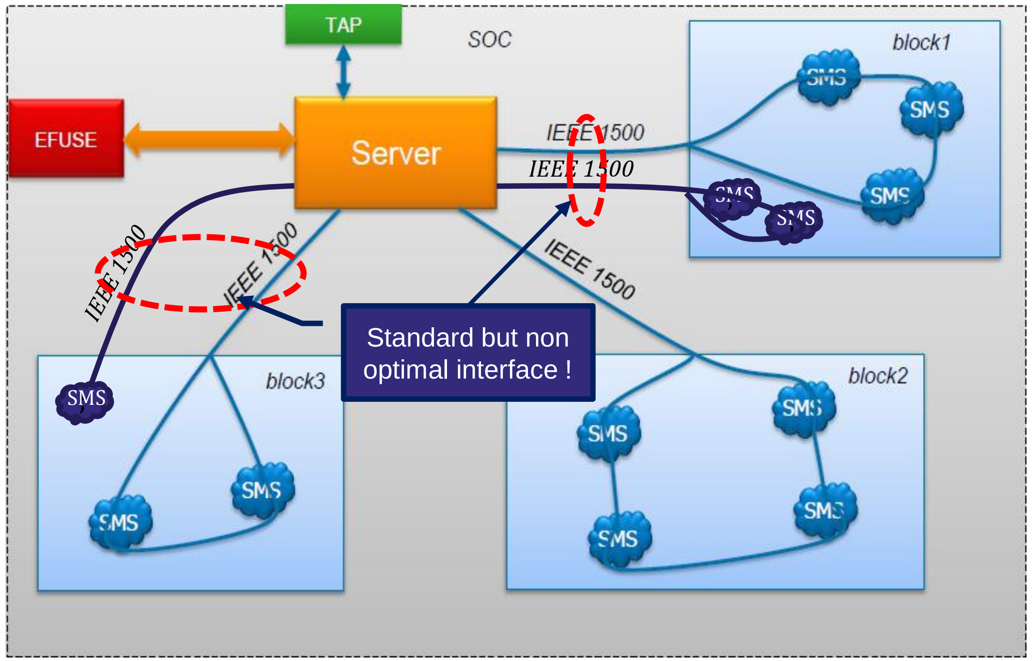
- The Synopsys Memory BIST solution (SMS) consists of :
  - protocol converter / eFuse handler “SMS server”
  - memory-wrapper “wrapper” and common master “SMS processor”

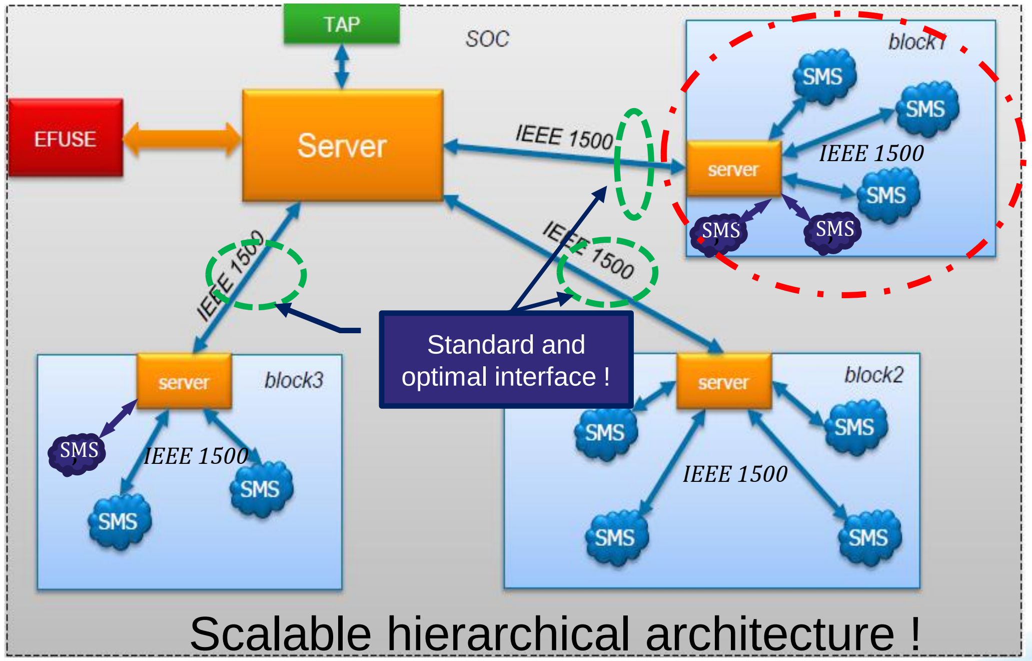


MBIST implementation for functional safety

Synopsys SMS BIST architecture for design

- SMS architecture for design constraints: IP-oriented approach for IP reuse
  - Each Sub-System contains its own complete MBIST solution:
    - Local IP server for a single IEEE 1500 interface connection to TOP level whatever SMS network
    - At least 1 processor(s)
    - Associated memory wrappers for all memories present in this IP.



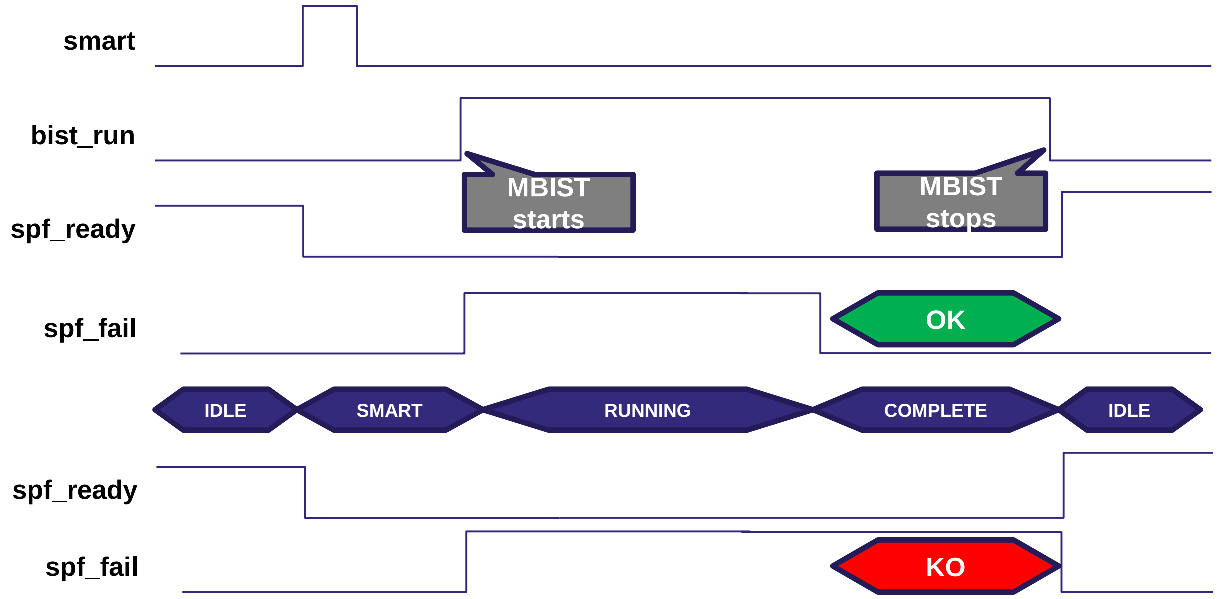


Scalable hierarchical architecture !

MBIST implementation for functional safety

Synopsys SMS BIST implementation for safety

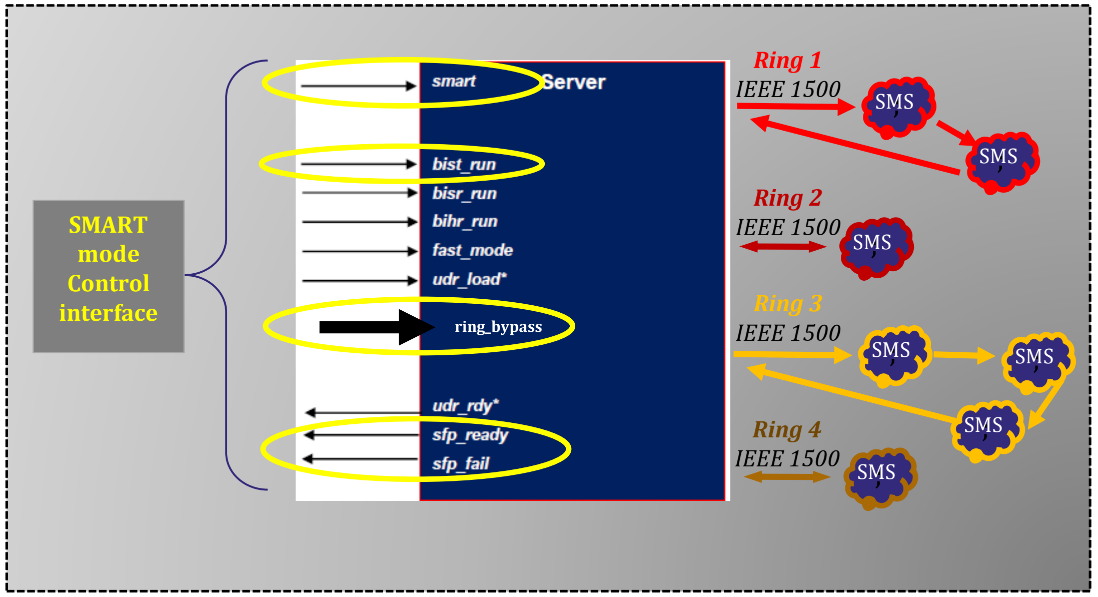
- SMS BIST feature to support safety: SMART mode interface at server level
  - The self-test is initiated by simply holding the bist_run signal high after selecting the SMART mode sequence, results can be observed on 2 SMART status outputs:
    - spf_fail: PASS/FAIL status
    - spf_ready : MBIST done
  - SMART state machine performs memory BIST test according grouping defined thanks to “ring_bypass” control bus configuration (No predefined hardcoded configuration for flexibility)

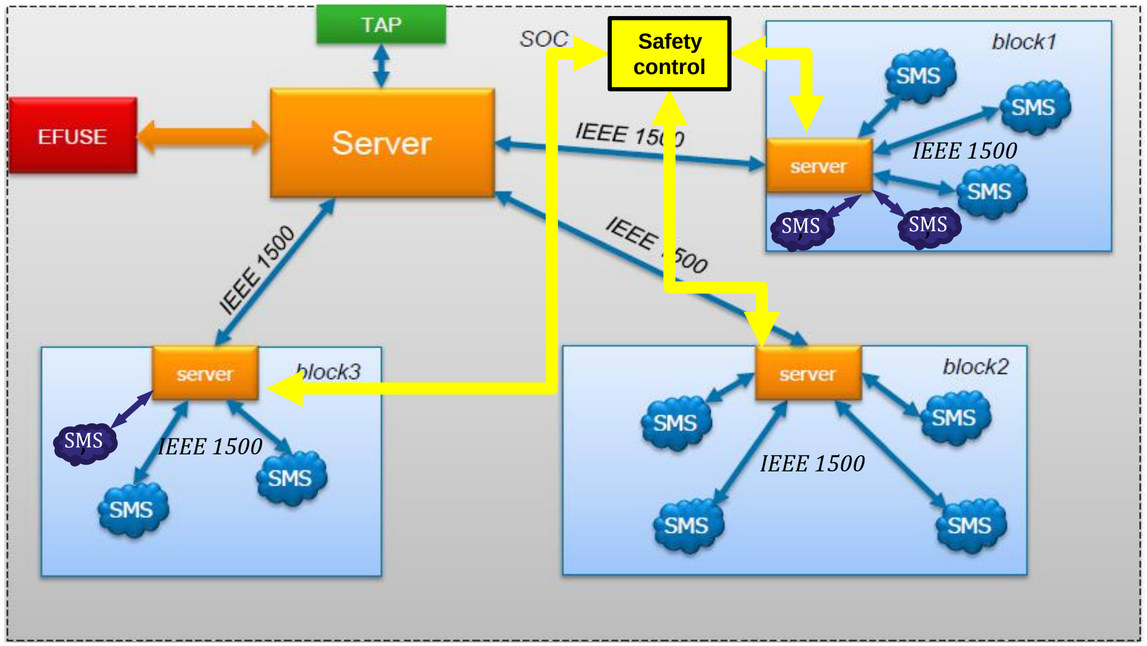


MBIST implementation for functional safety

Synopsys SMS BIST implementation for safety

- SMS BIST feature to support safety: SOC integration
  - The IP-oriented MBIST architecture is already aligned to support safety requirements
    - ⇒ Server implemented at IP level with SMART mode interface





  - SMS ring_bypass signal used to include/exclude memory from an MBIST run at sub-system level
    - ⇒ Allow to limit switching activity + control run time execution thanks relevant grouping of memory

MBIST implementation for functional safety

Synopsys SMS BIST implementation conclusion

- SMS scalable hierarchical architecture is perfectly in line with:
  - Design constraints thanks to a single and common interface at sub-system level
  - Safety needs thanks to the simple SMART mode control interface in functional mode
- No test coverage loss at sub-system level after SMS modules insertion
- Small area overhead of this strategy (only sub-server area overhead)
- MBIST execution run time is the major concern for on the fly use of the MBIST
  - Would be fine to get at least 2 selectable algorithms inside the MBIST solution to select on the fly the execution run time according system constraints !

Conclusion

Conclusion

- By combining various DFT techniques
  - Synopsys Logic BIST : Core wrappers / Test Point / Local clock and reset control
  - Synopsys Memory BIST : STAR server hierarchical flow
- We tackle the main today in-field automotive safety concerns
  - High level of detection for permanent fault inside the logic thanks to LBIST
  - Exhaustive detection of memory defect thanks MBIST
  - Self-test at power up and also on the fly during mission mode (core wrapper isolation)
  - Limited area overhead
  - Ability to achieve desires testing target in a limited run time
  - Limit switching activity
- What’s next ?
  - Deployment of these technics and flows in future Bosch automotive project