2021 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE OCTOBER 26-27, 2021

A Novel Approach To In-System SRAM Repair Verification in Embedded SoC to Bridge Gap between Test and Functional Mode

Harshal Kothari, Associate Staff Engineer, SSIR, Bengaluru, India (harshal.k1@samsung.com)

Eldin Ben Jacob, Associate Staff Engineer, SSIR, Bengaluru, India (eldin.jacob@samsung.com)

Ajay Vamshi Krishna, Senior Engineer, SSIR, Bengaluru, India (a.krishnaka@samsung.com)

Sriram Kazhiyur Sounderrajan, Associate Director, SSIR, Bengaluru, India (sriram.k.s@samsung.com)

Somasunder Kattepura Sreenath, Director, SSIR, Bengaluru, India (soma.ks@sasmung.com)

***Abstract*—Embedded memories are a huge part of any modern SoC and play a vital role in the performance of the design. The purpose of memories in systems is to store massive amounts of data. Since memories do not include logic gates and flip-flops, various fault models and test algorithms are required to test the memories. The process of testing the memories in a chip on automated testing equipment involves the use of external test patterns applied as a stimulus, the device's response being analysed by the tester, comparing it against the golden data stored as part of test pattern data. This is a complex and time-consuming procedure that involves external equipment. MBIST solution makes this easier by placing all these functions within the test circuitry surrounding the memory itself. It implements a finite state machine (FSM) to generate stimulus and analyse the response with the expected response. This self-testing circuitry acts as an interface between the high-level system and the memory. The challenges of testing embedded memories are minimalised by this interface as it provides direct observability and controllability. The FSM generates the test patterns for memory testing and reduces the need for external test pattern to be stored. Since the MBIST design is now responsible for the performance of vital memories, it is imperative to verify the MBIST design with complex use-case scenarios in a methodical manner. This paper discusses the approach to verify MBIST based SRAM Repair in embedded SoC in functional mode.**

Keywords – MBIST; SoC ; SMS ; DFT; Memory Repair; Functional Mode

I. INTRODUCTION

Memory Repair is implemented in two major steps: the first step is to analyze the failures diagnosed by the MBIST controller during the test for repairable memories, and the second step is to determine the repair signature to repair the memories which should be shifted to Built-In Self Repair registers (BISR). The repair packet generated by the MBIST model is then electrically fused (eFused) in a non-programmable memory on the chip. This repair signature is finally shifted to the corresponding memories using a complex memory system. In our SoC intended for AR/VR applications, we employed Synopsys STAR (Self-Test and Repair) Memory System (SMS) server and sub-servers network connecting to all subsystems via standard IEEE1500 interface. The SMS repair architecture and process of memory repair in our SoC is presented in Figure 2. Modern SoCs have hundreds of repairable memories embedded in various blocks. The memory repair becomes complicated to verify, as there are numerous possible combinations. Hence we attempt to bridge the gap between DFT block level verification in test mode and DV SoC level In-System verification in functional or mission mode by adopting functional and test coverage driven co-verification.

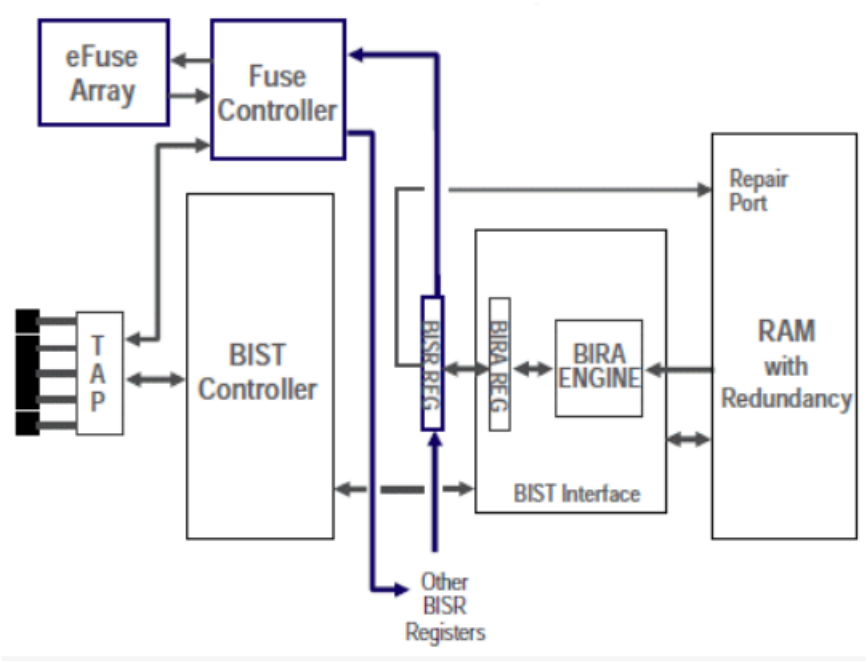

Figure 1. MBIST Memory Model

In functional mode, we attempt to cover end-to-end verification from eFuse to BIST controller to BIRA engine to memory redundancy ports. One of the major challenges in the verification of repair logic is determining the repair signature to replace the faulty SRAM columns with redundant columns. However, it is very inefficient to simulate all the cases of fault injection and BISR packet shifting in SoC level simulation because MBIST simulation is extremely time-consuming. To tackle this issue, we split the memory repair verification into two parts:

- MBIST verification
- Repair shifting verification.

Core SRAM Repair shifting verification has the following steps:

i) Load the repair data into eFuse memory
ii) Programming the SRAM repair server module to shift the repair signature to corresponding memories
iii) Inject fault in the memories at locations corresponding to the repair signature
iv) Testing Memory to check successful repair

The usage of memories varies with the block in which it is used and the method of verification also varies accordingly. For example, let us consider the embedded memory of a SoC's boot processor and power firmware master. This memory needs to be repaired before the hardware Finite State Machine (FSM) or the processor booting from Read Only Memory (ROM) releases reset for the boot processor and before the boot processor implements the next part of boot code. The memory repair verification needs to include the verification of the HW FSM RTL that performs repair shifting of the memory. Thus, the memory repair verification can be broadly classified into three sections and each of the sections involve a different technique to verification:

i) Repair shifting
ii) Control signals (reset, clock, etc.) and hardware controls
iii) Negative testing

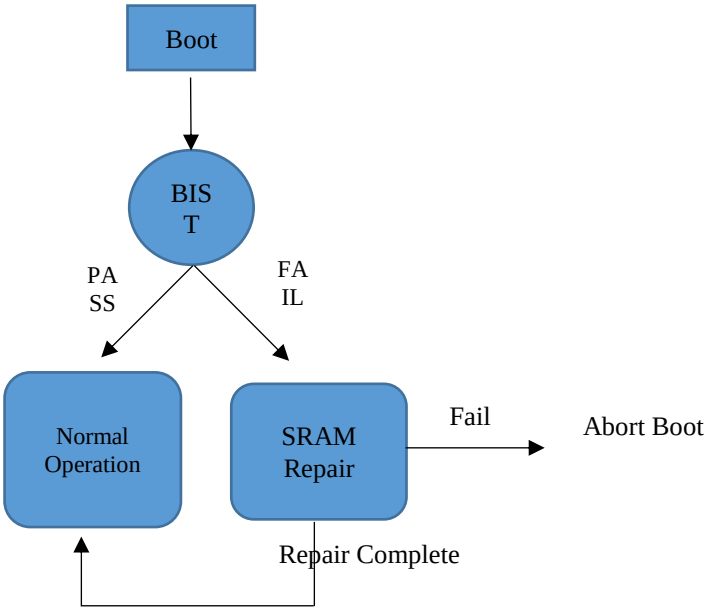

Figure 2. SRAM repair flow

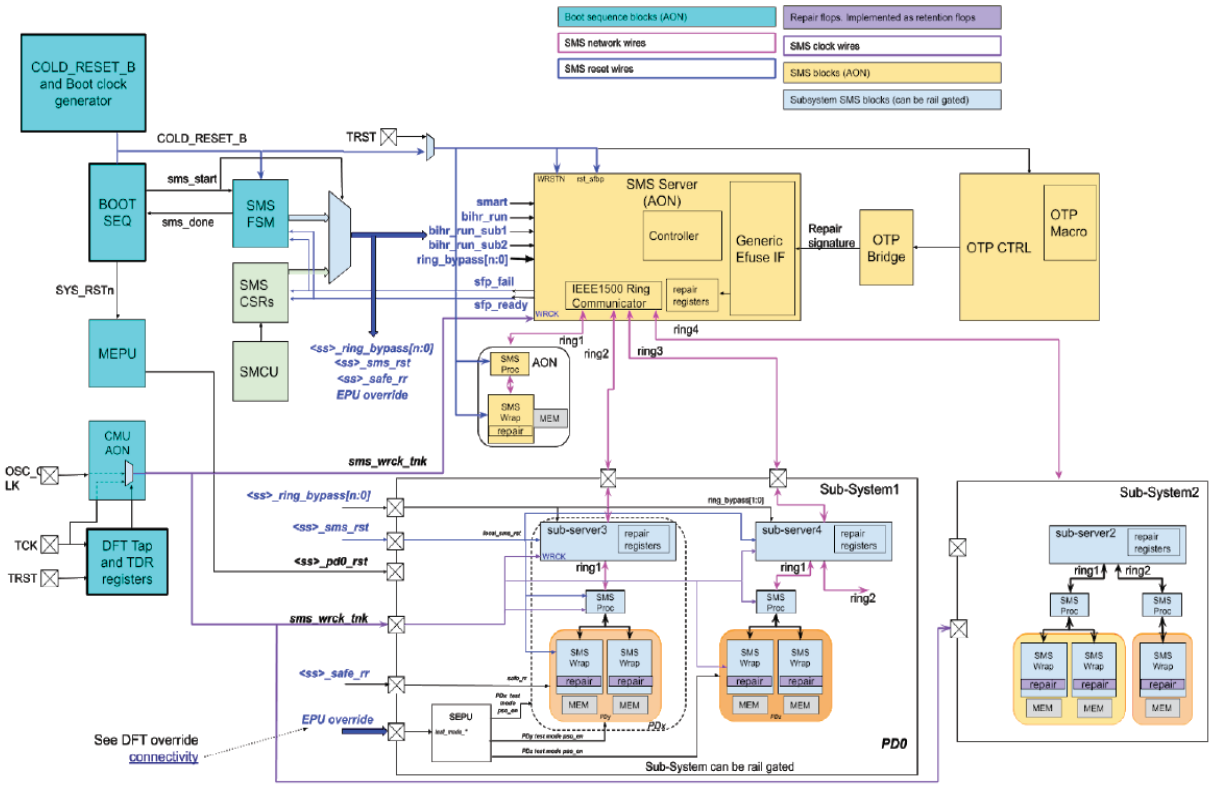

Figure 3. SMS repair architecture in SoC

SRAM repair verification involves verifying memory repair for all possible types of memory faults. Many of these faults, such as transition fault, can only be implemented in gate level simulation with timing. However, the timing GLS simulations are extremely slow. Hence, it is important to identify the scenarios that can only be verified in timing GLS without compromising on quality. Apart from transition faults and parallel repair, we also need to target the retention feature of memory repair cells in power-aware timing GLS simulation. Once the repair information is available after running BIST simulations, the repair bits need to be correctly shifted from OTP or eFuse interface to the physical memory's redundancy ports. Coverage for all 3303 memories in SOC was ensured via multiple iterations of simulations run to cover this connectivity from OTP memory, which is preloaded with the repair signatures corresponding to failing SRAM, to various SRAM's CRE and FCA ports. Preloading the repair bits for all memories' index together also helps save simulation time by targeting all blocks' SRAM repair in parallel. In this paper, we will discuss in detail the various steps involved in identifying the repair scenarios that need to be targeted in different simulation environments.

II. METHODOLOGY

SRAM repair shifting test sequences can be broadly divided into five main categories:

i) Core Repair Scenarios
ii) Retention Check
iii) Connectivity Check
iv) Error Injection
v) Hardware Control Checks

These are verified in functional mode at SoC level SV-UVM testbench using below attributes:

A. CORE REPAIR SCENARIOS

In core repair scenarios, the core functionality of SRAM repair is verified. The repair signature is preloaded in the OTP using the readmem task as can be seen in *Figure 4*. This contains the 18,653 fuse bits required to repair memories in all 12 SMS rings. The SMS related registers are then programmed to trigger the "SMART" interface to shift the repair signatures to the corresponding memories. The system is reset through WRSTN, rst_sms, rst_sfbp and provided clock WRCK. Server level smart control pins (bihr_run, bihr_run_sub_*, ring_bypass) are then asserted. Depending on which SMS cores need to be repaired, sub-server level ring_bypass pins are set. This is followed by activating a pulse on SMART pin which starts the SMART sequence. Server will put sfp_ready low and if not done before, will read OTP repair segment for the entire chip and load into internal server repair registers. This step happens only once on cold power up. Next, the repair data is loaded from Server into Sub-Servers followed by each SMS cores that are not bypassed. During this process, we check for sfp_ready at server level to come back to high (when all repair sequences at sub-servers level will be done) and check for sfp_fail to ensure the SMART shifting happened as intended. Finally, the FCA and CRE bits of the destination memory is checked to verify that the repair signature is shifted properly. Once this completeness is ensured, we run functional IP datapaths accessing these SRAMs to qualify the entire operation. System Verilog Assertions are also employed to check the SMART sequence in *Figure 5*.

```
$readmemb("[redacted]otp_all_ones.txt",[redacted].mem1,0);
$readmemb("[redacted]full_repair_signature.txt",[redacted].mem1,9216);
$readmemb("[redacted]otp_all_ones.txt",[redacted]mem1,28800);
```

Figure 4. OTP Preloading

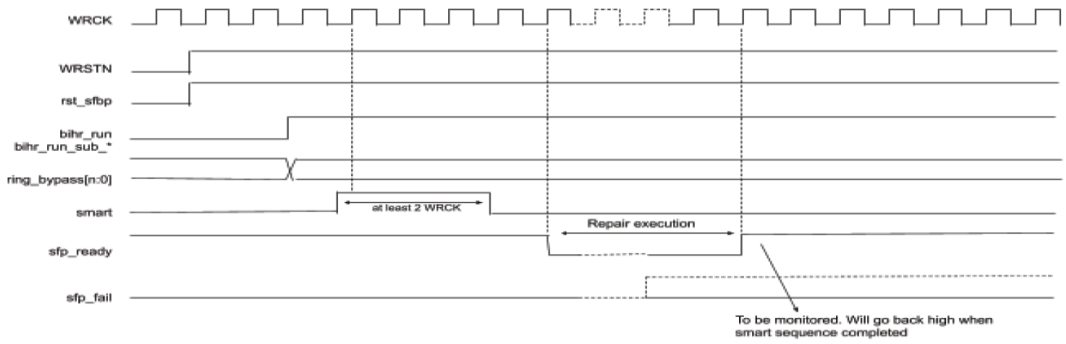

Figure 5. SMS server SMART interface[1]

B. *RETENTION CHECK SCENARIOS*

When cold boot reset is released and the chip is booted up for the first time, SRAM repair signature is shifted to the corresponding memories. These repair values are stored in the reconfiguration registers of the memories. For reconfiguration register protection, SMS wrappers have an option called “Safe Memory Reconfiguration Register” that enables “safe_rr” pin on the wrapper to prohibit reset operation (*Figure 6*). This allows preserving the reconfiguration information if the SMS is reset. It is necessary to retain these values in the reconfiguration registers since the repair cannot be shifted after every time the power domain of the memory is gated and ungated. Hence, it is important to verify the retention feature of these reconfiguration registers. The test sequence checks the retention feature by depositing random value in the reconfiguration register, powering down the power domain containing memory and verifying that the value is retained. The same check is repeated when power is ungated to check redundancy ports have got the correct repair signature again. The sequence is depicted in *Figure 7*.

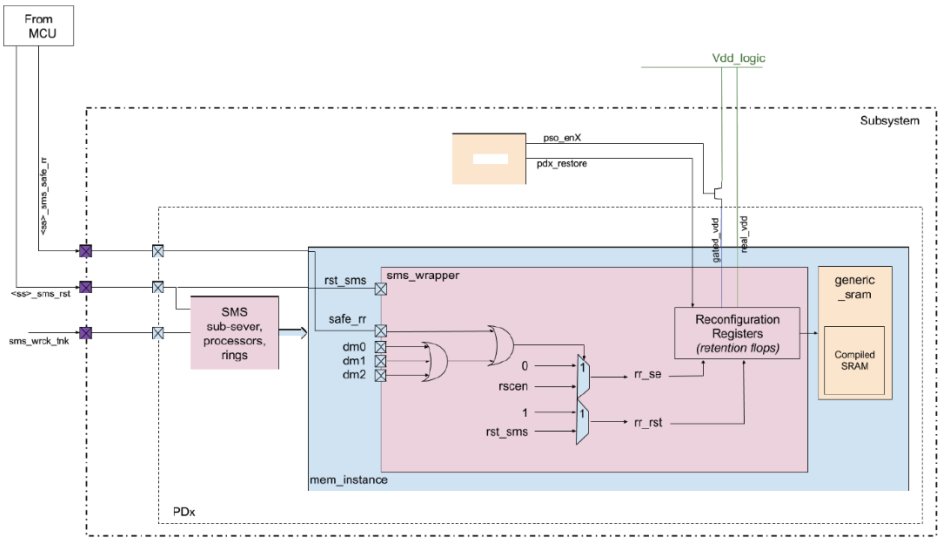

Figure 6. Safe reconfiguration register

```
$xm_deposit("                                                    reconfig_r_reg_19_.D","1'b1");
if(
                                                                 reconfig_r_reg_17_.Q != 1'b1)
  `uvm_error(get_type_name(), "FLCN_PD1_reconfig_reg_17_ retention fail")
```

Figure 7. Retention Check

C. CONNECTIVTY CHECK SCENARIOS

Modern SoCs contain hundreds of memories that are scattered throughout the chip. The repair signature originates at the OTP memory, shifted to eFuse controller, and then the SMS controller reads this via OTP bridge and shifts the repair packets to the memories. There is a possibility of connection mismatch in this path. Connectivity check tests verifies end-to-end connectivity for memory repair registers. Random repair data is loaded into the OTP memory along with the corresponding server ID and memory ID. The repair data is then shifted using the SMS controller. Data sanity is checked between the memory repair registers and the OTP memory containing repair packets to verify connectivity for all 3303 memories as seen in the *Figure 8.*

```
if(                                                                    FCA1[2] !== data_otp[4015])
  `uvm_error(get_type_name(),$psprintf("Data mismatch for
.FCA1[2]: %b , expected : %b",                                                          FCA1[2],data_otp[4
)))
```

Figure 8. Connectivity Check

D. ERROR INJECTION SCENARIOS

So far, the verification of various elements of SRAM repair shifting has been discussed in detail. However, it is also necessary to ensure that the signature is repairing the fault in the memory as intended. Error injection scenario verifies the memory repair by using fault injection tasks in the memory model. In the first iteration of repair, the repair signature is loaded with all zeros and the fault injection is verified as negative scenario. Then the golden repair signature packet which enables column redundancy for faulty bit and sets the address for repair is loaded in OTP and shifted to the memory to verify fault correction. After sfp_pass, this memory is accessed using functional datapath for a write followed by read. Thus we can ensure that the memory repair is effective and upon a transaction being directed at this memory, the data sanity is intact.

E. HARDWARE CONTROL CHECKS

The usage of memories varies with the block in which it is used and the method of verification varies accordingly. For example, let us consider the embedded memory of a SoC's boot processor. This memory needs to be repaired before the HW FSM releases reset for the boot processor and before the boot processor implements the next part of boot code. The memory repair verification needs to include the verification of the HW FSM code that performs repair shifting of the memory. After the basic boot is completed, the repair shifting for other memories can be performed by reusing this boot FSM. The register overrides are used to enable the boot FSM to trigger SRAM repair for a particular block. The FSM waits for the done (sfp_ready/sfp_fail) signal to be asserted from the SMS server to conclude execution. The status registers can be used to verify the pass or fail status of the repair. The Power Management Unit also needs to work in tandem with SRAM repair process to ensure the power domain for which SMS repair is targeted in ungated and the save and restore control for retention flops implementing the reconfiguration repair registers is handled correctly. Connectivity checks between RTL sequencer, SMS server and SMS sub-servers for WRSTN, rst_sms, rst_sfbp, bihr_run, ring_bypass, safe_rr etc. are also verified.

III. CONCLUSION

Memory repair is a very crucial part of the SoC and as such it is important to verify the quality of verification for memory repair. Coverage was measured for SRAM repair verification to ensure that all possible scenarios are simulated. The metrics for the functional and code coverage are discussed below.

Functional Coverage helps measure the extent of the verification with respect to various features by measuring the changes in control signals and register values. Some of the major cover points used in the functional coverage of SRAM repair verification is given below:

- Boot FSM controlled SRAM repair – the repair for some memories need to be shifted before the boot processor is powered up. The reset for boot processor is used in conjecture with the SMS control signals to cover this feature.
- Post-Boot SRAM repair – the repair after boot completion is implemented by the main processor using system registers. The repair runs for this method is done with SMART server and ring bypass option.
- Repair Retention – the repair cannot be shifted each time the memory is reset or powered down, so retention checks are imperative during SRAM repair verification. The individual scenarios for all possible reset and power-down scenarios are created and cover points help to ensure they are covered.
- SRAM Repair Connectivity- the SRAM repair information is saved in the eFuse memory and these OTP packets are shifted to the memories. The connectivity to each and every memory with the correct mapping to the BISR registers and redundancy ports to be verified is essential for the SoC level verification.
- SMS repair exclusive features – The below stated are some special features of the SRAM repair logic we are using which has to be covered to check the correctness of control and status updating process: bihr_run, ring_bypass, safe_rr and smart signal.
- Parallel repair – the SRAM repair done one after the other for all blocks will be a time consuming activity, due to which the option of parallel processing of SRAM repair is required.
- Serial repair – the memories present in various blocks and power domains can be repaired sequentially one after the other. This is important to cover using serial memory repair to ensure SMS repair for 1 ring does not impact the repair of others.
- Signature based repair – There are some golden reference signatures provided by the DFT team to run repair with post-BIST signatures. This cover-point is added to ensure that the signature of every block's memory is covered.

Code coverage on RTL sequencer that drives SRAM repair controller was generated and the 2415 bins that directly correspond to the SMS related signals were covered fully.

The extensive number of tests and 100% coverage closure helped find many issues in SRAM repair. There was a limitation on the SMS server wherein repair cannot be run for the main server and sub servers parallel. Thus, the hardware sequence had to be modified to accommodate the repair for main server and sub servers in a sequential manner. The retention scenarios helped to correct the safe_rr signal value for memories in always on power domain to avoid uninitialized memory data. Furthermore, the end-to-end connectivity tests helped ensuring that the repair signature is shifted to the memories properly. There are some limitations in SRAM repair DV primarily stemming from design constraints. Since the design does not support in-system BIST, functional mode DV has to rely on sms_done/fail signals from the SMS server to ensure that the repair is shifted. The actual BIST repair can only be verified in test mode with BIST executed. With this functional mode DV at SOC level with 78 core scenarios and 42 end-to-end connectivity checks, 15+ bugs as well as few architectural limitations were unearthed and fixed with SMS based SRAM repair before tape-out of the chip.

References

[1] STAR Memory System: https://www.synopsys.com/implementation-and-signoff/test-automation/star-memory.html