Melioration of Enhanced Low Power Memory Retention and Low Power Feature Validation Using API's

Nithin Venkatesh
Senior Technical Lead
Nithin.Venkatesh@intel.com
Akula Hareesh
Verification Engineer
hareesh.akula@intel.com
Intel Technology India Pvt Limited, No 23/56-P, Outer Ring Road, Bellandur, Bengaluru – 560103

Abstract- **With ever increasing SoC complexity and advanced power-aware architectures, the challenges involved with SOC Low Power verification have grown substantially. Flows and Methodologies adopted using standard scripting for extracting design information and enabling automation are not scalable. As memories are designed very tightly to the limits of the technology, they are more prone to failures than logic. The overall yield of an SOC thus is heavily affected by the yield of memory cores in the chip. This article proposes a robust methodology using API's and System Verilog assertion to effectively verify the low power intent. This methodology majorly overcomes limitations, manual efforts put on validating low power features.**

I. INTRODUCTION

With ever increasing SoC complexity and advanced power-aware architectures, the challenges involved with SoC Low Power verification have grown substantially. Flows and methodologies adopted using standard scripting for extracting design information and enabling automation are not scalable. There is continuous improvement in methodologies to reduce the cost and improve debug throughput that can have huge impact in the time-to-market of the products.

Built-in self-repair (BISR) techniques have widely been used for enhancing the yield of memory cores in SOCs. However, built-in self-repairable memory poses verification challenge in design phase. Efficient verification strategies for built-in self-repairable memory systems thus are essential and imperative. In one of the previous project, it was observed that BISR registers weren't converted to retention register that was causing data loss during power down mode which leads to removal of complete BISR chain for that particular partition containing ~8% of memories in the design, which resulted in a substantial yield loss.

Based on increase in complexity in low power verification done on multiple SOCs, we propose robust low power verification methodology using API's from debug tool and System Verilog Assertions. This methodology majorly overcomes limitations, manual efforts put on validating low power features.

II. LOW POWER VERIFICATION CHALLENGES

FIGURE 1 How the verification challenge has grown over the past two decades [1]. From the case study and analysis done by the team from Synopsys, the complexity for doing the verification has increased significantly for past 2 decades and the design size has also grown many folds. Due to many IP's are getting integrated in the SOC and number of power domains getting added into SOC, verification engineers are facing lots of challenges in day-to-day debugging.

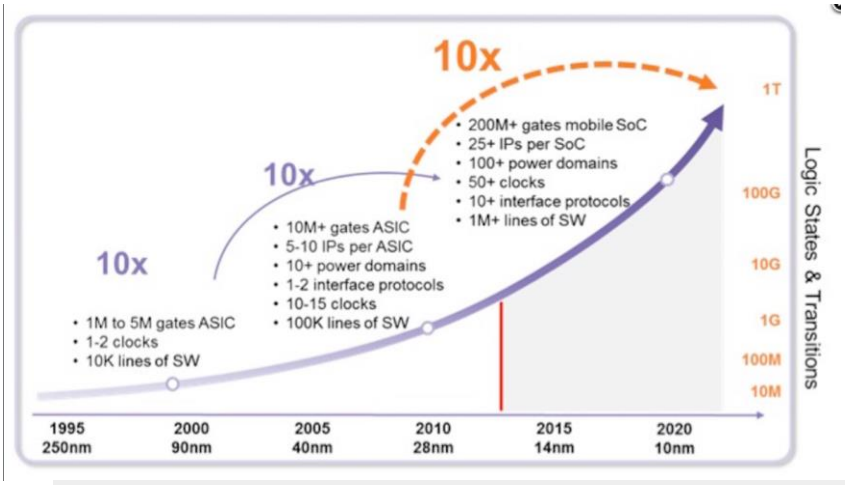


Figure 1. How the verification challenge has grown over the past two decades [1]

For low power verification, it is critical to have a methodology which defines verification flow which can seamlessly help in finding bugs at earlier stage with ease. For validating BISR Retention capable registers in low power verification is a challenge which comes up with a lot of manual work i.e. eyeball verification through waveforms of retention register features before and after power cycle and connectivity with functionality check of ~35k low power memory signals. With every new design release, significantly the memory list and signal list will change eventually resulting in addition of work. This makes it very difficult to verify low power feature without any defined methodology without which will end up in spending more time.

Figure 2 shows the problem statement of manual effort in signal extraction, assertion evaluation process and simulation re-run post debug of failing assertions.

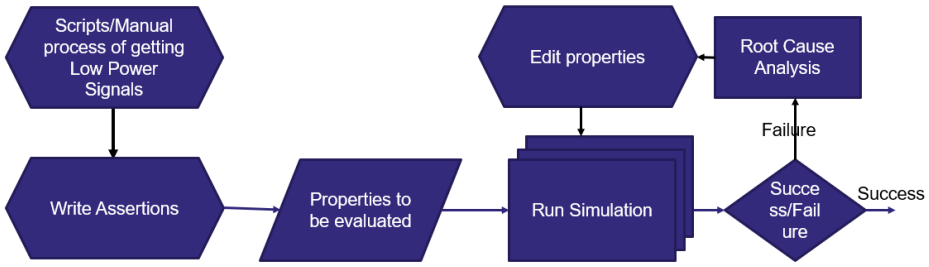


Figure 2. Problem Statement

Major challenges faced during low power verification are as mentioned below:

1. Signal extractions from design being an overhead to complete validation task on time
2. Tedious process of validating more than 50k low power memory signals
3. Time consuming process on validating low power signal connectivity
4. Need to run multiple iterations of simulation to evaluate properties

III. API Based Methodology

This paper defines a new methodology to verify the retention of BISR registers in SoC power aware simulations and thereby making sure that there is no compromise on yield loss because of memory repair issues.
This paper also specifies the usage of API's from the debug tool. We verified the functionality of ~35K low power Memory signals spread across different blocks at SOC level. We are also able to get initial values of all outputs/inputs at different ports of multiple hierarchies. Starting from extracting required signals across hierarchies followed by generating assertions evaluating them offline, the methodology helped us to reduce the manual effort of eyeballing waveform. Turnaround time has been reduced to 1-2 hours from 5-6 hours.

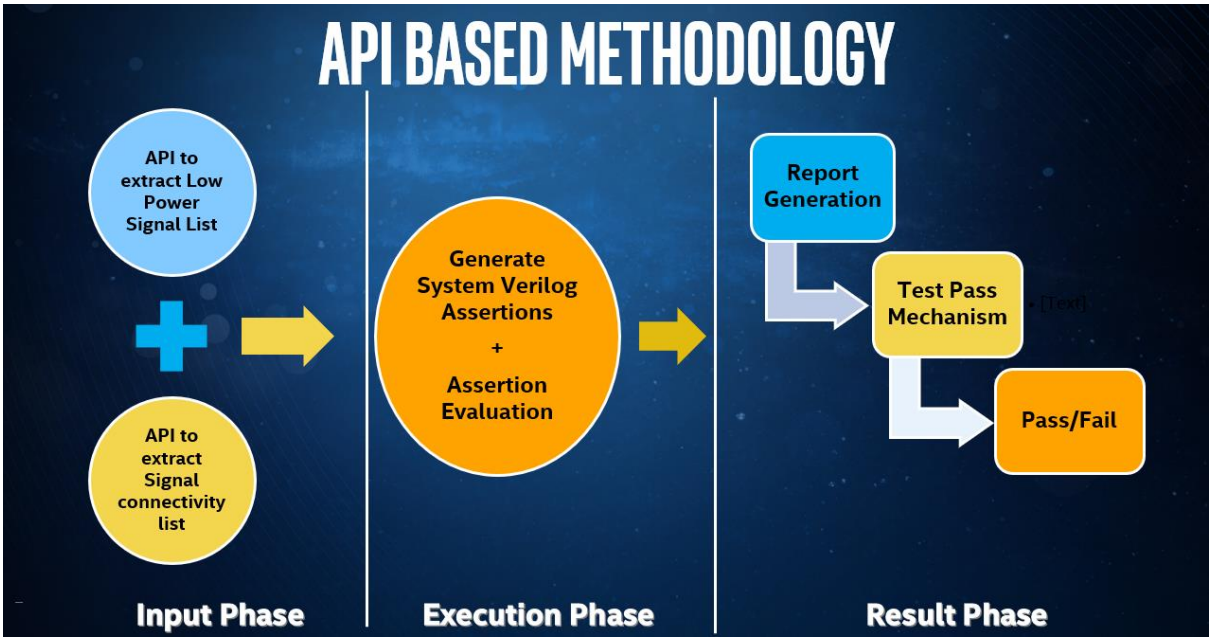


Figure 3. API based Methodology with 3 Phases

1. Input Phase:
    a. Retention register list and power control signal extraction using API's which will create a single csv file.
    b. Low power memory signal extraction from entire design hierarchy using APIs from the debug tool
    c. Signal connectivity check by extracting output ports from multiple design hierarchies.

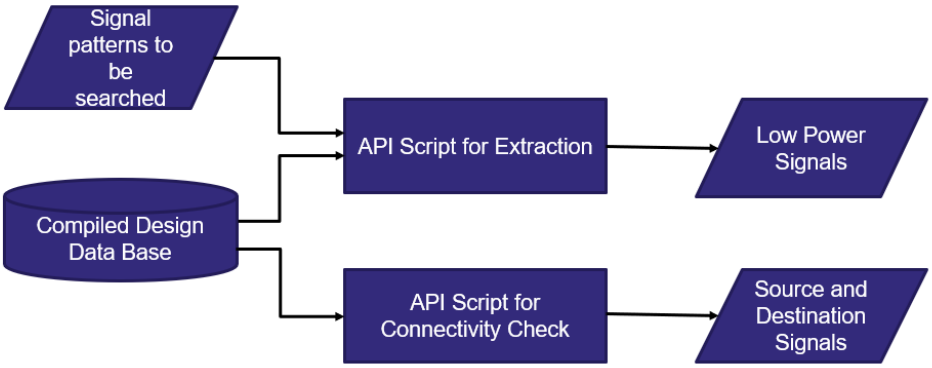


Figure 4: Input Phase

2. Execution Phase:
    a. Extracted signal list from first phase taken as input for our script which generates automated assertions
    b. Evaluate properties using SVA evaluate from the debug tool
    c. Offline assertions helped us in debugging in quick time and execution time as well. Because in our system, SOC RTL+UPF compilation/elaboration of the design used to take ~10 hours. So, if we go with “simulating assertions in live simulation”, any assertion compilation or post assertion failure to recompile the assertions we must go through the whole SOC compile/elab process. By executing assertions offline or during post processing after simulations, saves us a lot of time and debugging the failures were much faster
    d. Dumping of required signals were taken care in our “Internal Assertion Generator” which reads those signal list and generates an required hierarchy list for FSDB dumping for that simulation. So, this eases our effort of manually adding desired signals or hierarchies as we are discussing here ~50k signals

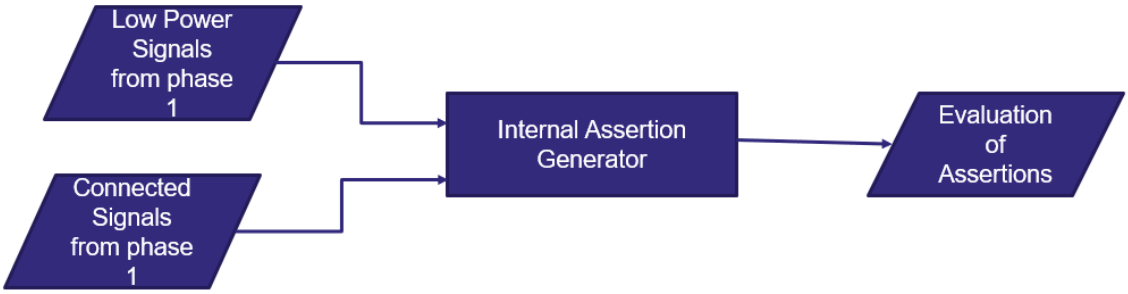
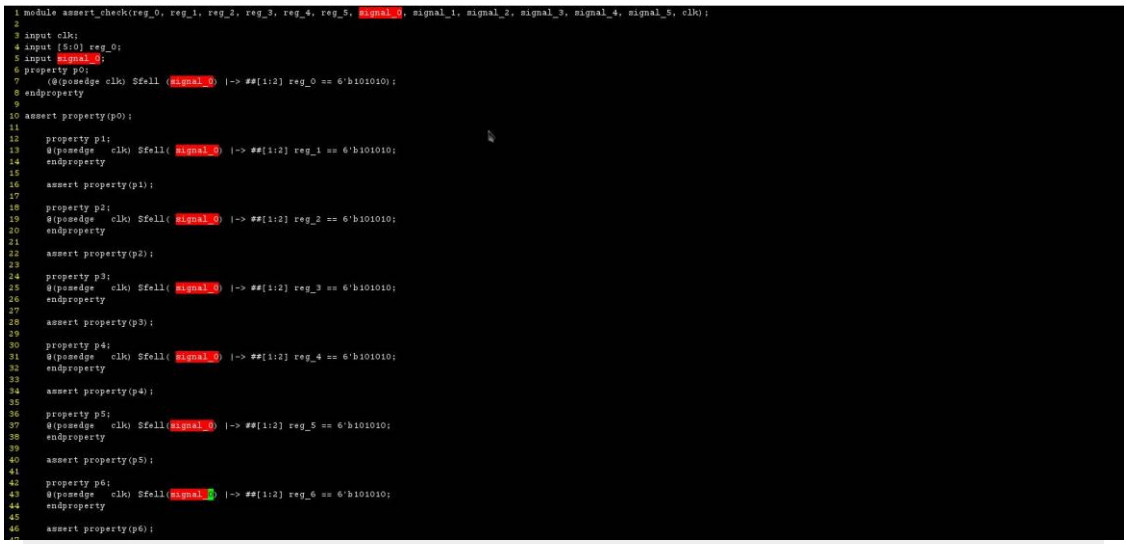


Figure 5: Execution Phase



```
module assert_check(reg_0, reg_1, reg_2, reg_3, reg_4, reg_5, signal_0, signal_1, signal_2, signal_3, signal_4, signal_5, clk);

input clk;
input [5:0] reg_0;
input signal_0;
property p0;
    (@(posedge clk) $fell (signal_0) |-> ##[1:2] reg_0 == 6'b101010);
endproperty

assert property(p0);

    property p1;
    @(posedge   clk) $fell( signal_0) |-> ##[1:2] reg_1 == 6'b101010;
    endproperty

    assert property(p1);

    property p2;
    @(posedge   clk) $fell( signal_0) |-> ##[1:2] reg_2 == 6'b101010;
    endproperty

    assert property(p2);

    property p3;
    @(posedge   clk) $fell( signal_0) |-> ##[1:2] reg_3 == 6'b101010;
    endproperty

    assert property(p3);

    property p4;
    @(posedge   clk) $fell( signal_0) |-> ##[1:2] reg_4 == 6'b101010;
    endproperty

    assert property(p4);

    property p5;
    @(posedge   clk) $fell(signal_0) |-> ##[1:2] reg_5 == 6'b101010;
    endproperty

    assert property(p5);

    property p6;
    @(posedge   clk) $fell(signal_0) |-> ##[1:2] reg_6 == 6'b101010;
    endproperty

    assert property(p6);
```

Figure 6: Generic Snippet of Automated Assertions from Internal Assertion Generator Script

3. Result Phase:
    a. After assertion evaluation, reports are generated and analyzed and test passing mechanism is defined based on assertion status and functional simulation status
    b. Figure 8 shows the report snippet of all the properties evaluated with the status. The pass/fail mechanism of the test will be decided on this assertion report and functional simulation result.

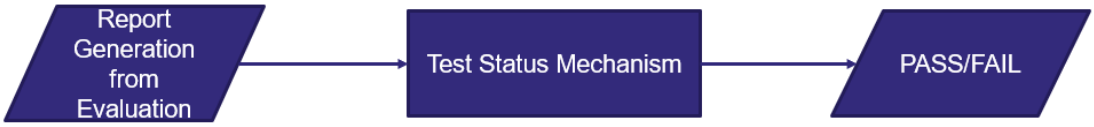


Figure 7: Result Phase

no	Name	Success	Fail	Incomplete	Total
1	property_p1	22034	243	0	0
2	property_p2	5004	69	0	0
3	property_p3	32004	459	0	0
4	property_p4	24090	0	0	0
5	property_p5	764	0	0	0
6	property_p6	256	0	0	0

Figure 8: Report Snippet

IV. RESULTS

This methodology has really helped us in reducing debug efforts significantly and increase quality in delivery. Debugging of any failing assertions can be done offline by just having dump which reduces the entire simulation time methodology also helped us to reduce the manual effort of eyeballing waveform for low power memory signal verification. Turnaround time reduced to 1-2 hours from 5-6 hours. As shown in figure 9, our execution time v/s RTL release has significantly improved over time by 5x.

Usage of api's are mainly for signal extraction using pattern matching and collecting all the signal list in an excel sheet, which will be parsed as an input to our "Automated Assertion Generation Script" (details of this automated script cannot be shared due to proprietary concerns). This is a unique technique as it involves api's, generation of assertions and evaluating those assertions while simulation flow itself.

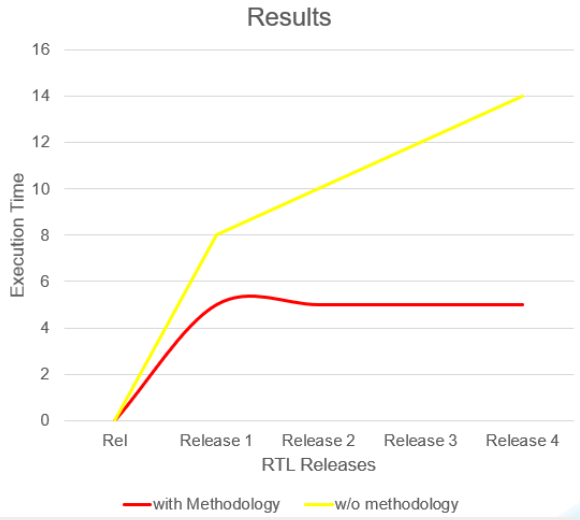


Figure 9. Results: Execution Time v/s RTL Releases

V. CONCLUSION

This methodology eased low power verification with the use of API and automated assertions by overcoming the low power verification challenges thereby enabling to achieve the goal of quality verification. Extraction of required signals from the complete and complex design followed by data processing to declaration of results is achieved in an efficient and enhanced way. Re-usability is an additional takeover from this methodology which can significantly push us to embrace this framework in the low power verification. This methodology has improved validation time cycle. There is enormous scope of the above framework to expand further for addressing challenges seen in Low Power Verification and System Verilog Assertion centric verification. This paper indeed is a direct contribution to low power verification, as mentioned using API's we develop automated assertions for BISR Retention registers, and these assertions are used to validate the retention register functionality with ease. Using API's, we have also validated low power memory signal functionality by again generating automated assertions and validate the same.

REFERENCES

[1] https://www.techdesignforums.com/practice/technique/setting-objectives-measuring-outcomes-low-power-verification/