2019 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Ensuring Deadlock-Free ASIC Operation: A Comprehensive Integration of Frequency and Operation Coverage Matrices

Sohel R. Langardar, Micron Technology, Bangalore, India (slangardar@micron.com)
Naman Kothari, Micron Technology, Bangalore, India (nkothari@micron.com)
Balwinder Pal Sethi, Micron Technology, Bangalore, India (bsethi@micron.com)

***Abstract*— In ASIC SoCs, determining the optimal clock frequency combinations is a complex challenge and difficult to analyze each one, including idle and active scaling during operation. The final combination is often determined much later in the process, based on power and performance results along with real silicon data. Functional correctness across the operational frequency spectrum is often under-verified. This paper introduces a functional coverage-based methodology using System Verilog to address this gap. We define two key coverage matrices: (i) a Frequency Coverage Matrix to track frequency combinations across IPs and subsystems, and (ii) a Frequency and Operation Coverage Matrix that captures meaningful activity under each frequency scenario. By leveraging architectural inputs to constrain the verification space, our methodology ensures high-impact combinations are tested thoroughly.**

Keywords*— *SOC, Functional Coverage, Covergroup, STA Frequencies

I. INTRODUCTION

The complexity of modern ASIC SoCs continues to grow, driven by the need to integrate multiple IPs and subsystems, each operating under different clock domains. To achieve energy efficiency and performance optimization, frequency scaling—both static and dynamic—is widely employed. These variations introduce a verification challenge: ensuring that all modules operate correctly across a wide range of frequency scenarios, including corner cases and intermediate operating points.

Typically, the minimum and maximum supported frequencies for each domain are determined by the ASIC architect and timing analysis tools. However, actual operating frequencies are finalized late in the design cycle based on post-layout simulations, power-performance trade-offs, and silicon characterization. Consequently, there is a risk that some critical frequency combinations are insufficiently verified before tape-out.

This paper proposes a systematic approach using functional coverage to track and validate frequency combinations. By incorporating activity monitoring, we go beyond traditional frequency verification and provide a holistic framework for SoC-level frequency validation.

II. MOTIVATION AND BACKGROUND

Existing functional verification strategies largely focus on protocol correctness, interface integrity, and corner-case functional tests. Clock-related verification is often limited to Clock Domain Crossing (CDC) checks and basic STA timing assertions. These approaches do not comprehensively address whether each IP behaves as expected across all architecturally supported frequency ranges.

Furthermore, verifying every permutation of clock frequency combinations in a large SoC is computationally infeasible. For example, with just five clock domains and four possible frequencies per domain, the total number of combinations grows exponentially (4^5 = 1024). Most of these combinations are unlikely or irrelevant.

To make frequency combination verification practical and meaningful, input from SoC architects is essential. They can narrow down the frequency sets to those that are feasible, functionally valid, and most likely to occur in silicon. By defining a constrained yet representative subset, verification becomes targeted, achievable, and valuable.

III. PROPOSED METHODOLOGY

To address the under-verification of frequency-dependent behavior in ASIC SoCs, our solution introduces a structured coverage-driven approach centered around two key matrices, implemented using System Verilog:

A. IP/Sub-System Frequency Coverage Matrix

This matrix captures which frequency combinations have been applied to each IP or subsystem during simulation. Coverage points are defined per domain and are sampled dynamically as frequencies change during testbench execution. This ensures visibility into the breadth of frequency scenarios exercised across the design.

B. IP/Sub-System Frequency and Operation Coverage Matrix

While individual frequency application is important, verifying functional activity under cross-IP frequency combinations is essential to uncover hidden bugs and ensure system-level correctness. This matrix extends the frequency coverage by monitoring user-defined activity signals.

A frequency combination is considered fully covered only when:

i. All IPs are operating under a valid frequency configuration as defined in the architectural combination set.

ii. Each IP exhibits expected functional behavior under that configuration.

This matrix captures scenarios where frequencies were applied across multiple IPs, but no meaningful activity occurred in one or more domains—highlighting potential verification blind spots. By correlating frequency combinations with operational signals, the methodology ensures that high-impact, cross-IP scenarios are not only exercised but also validated for correctness.

C. Role of Architectural Constraints

Architectural constraints are leveraged to limit the number of required frequency permutations. This targeted approach improves regression efficiency and simplifies analysis complexity, all while maximizing verification relevance. By focusing on high-impact combinations, the methodology ensures that verification efforts are both scalable and strategically aligned with system-level goals.

IV. METHODOLOGY IMPLEMENTATION

To systematically implement the frequency verification methodology, we designed a modular testbench within SoC environment. The testbench is composed of several modular units, each contributing to the coverage infrastructure. The overall architecture is illustrated in Figure 2. This infrastructure enables real-time detection of exercised frequency combinations and corresponding operational activity for functional coverage reporting.

A. Header

This file defines the foundational data types and structures used throughout the testbench. It includes:

- SOC_IP_FREQ_t: Enumerates supported frequency values for each IP. An example is shown in Figure 1(a).

```
//ENUM : Multiple Frequencies supported for each IP
//IPx_Freq = 'hx<2bytes>_Freq<2bytes>;
//x=IP index
//Freq= frequency in Mhz
typedef enum {
    IP1_100 = 'h10064, //100MHz
    IP1_400 = 'h10190, //400MHz
    IP2_300 = 'h2012c, //300MHz
    IP2_500 = 'h201f4  //500MHz
} SOC_IP_FREQ_t;
```

Figure 1(a).

- SOC_IP_FREQ_COMB_t: Enumerates unique identifiers for valid cross-IP frequency combinations. An example is shown in Figure 1(b).

```
//ENUM : unique value for each supported combinations
typedef enum {
    COMBO_1 = 'haaaa0001,
    COMBO_2 = 'haaaa0002,
    COMBO_3 = 'haaaa0003,
    COMBO_4 = 'haaaa0004
} SOC_IP_FREQ_COMB_t;
```

Figure 1(b).

- Soc_combs_t: A structure that maps each combination enum to its corresponding frequency values. An example is shown in Figure 1(c).

```
//Structure type to hold information for each combination
typedef struct {
  SOC_IP_FREQ_COMB_t freq_comb_enum;
  SOC_IP_FREQ_t freq_comb_values[`IP_N];
} SOC_COMBS_t;

//Information for each frequency combination across IPs
SOC_COMBS_t SOC_FREQ_COMB_ENTRIES[`COMB] = {
  {COMBO_1 , '{IP1_100, IP2_300}},
  {COMBO_2 , '{IP1_100, IP2_500}},
  {COMBO_3 , '{IP1_400, IP2_300}},
  {COMBO_4 , '{IP1_400, IP2_500}}
};
```

Figure 1(c).

B. Frequency Monitor

As shown in Figure 2, this module receives clock signals from each IP and outputs a stable frequency value. It filters out transient glitches and ensures only valid frequency states are propagated to the control unit.

C. Activity Monitor

This module monitors activity signals from each IP. When an activity signal remains stable for a predefined duration, it asserts an IP-specific activity event. This ensures that only meaningful operational states are considered for coverage.

D. Control Unit

The control unit is the heart of the coverage infrastructure. It operates two parallel threads:

- Thread 1: Monitors each IP's frequency independently. When a valid frequency is detected, it writes the corresponding SOC_IP_FREQ_t enum to a GPR register. This feeds the Frequency Coverage Matrix.
- Thread 2: Monitors cross-IP frequency combinations and their activity status. When all IPs are operating under a valid combination from SOC_FREQ_COMB_ENTRIES and show stable activity, it writes the corresponding SOC_IP_FREQ_COMB_t enum to the GPR. This feeds the Frequency and Operation Coverage Matrix.

Both threads continue monitoring until all combinations are covered. Once a combination is hit, its check is disabled, but the thread continues checking remaining combinations.

E. Coverage Collector

This unit consists of two submodules:

- CSR Monitor: Samples valid writes to the GPR register for both enum types.
- Covergroups: Two covergroups are defined—one for SOC_IP_FREQ_t and another for SOC_IP_FREQ_COMB_t. These coverpoints receive sampled enum values from the CSR monitor and build the respective coverage matrices.

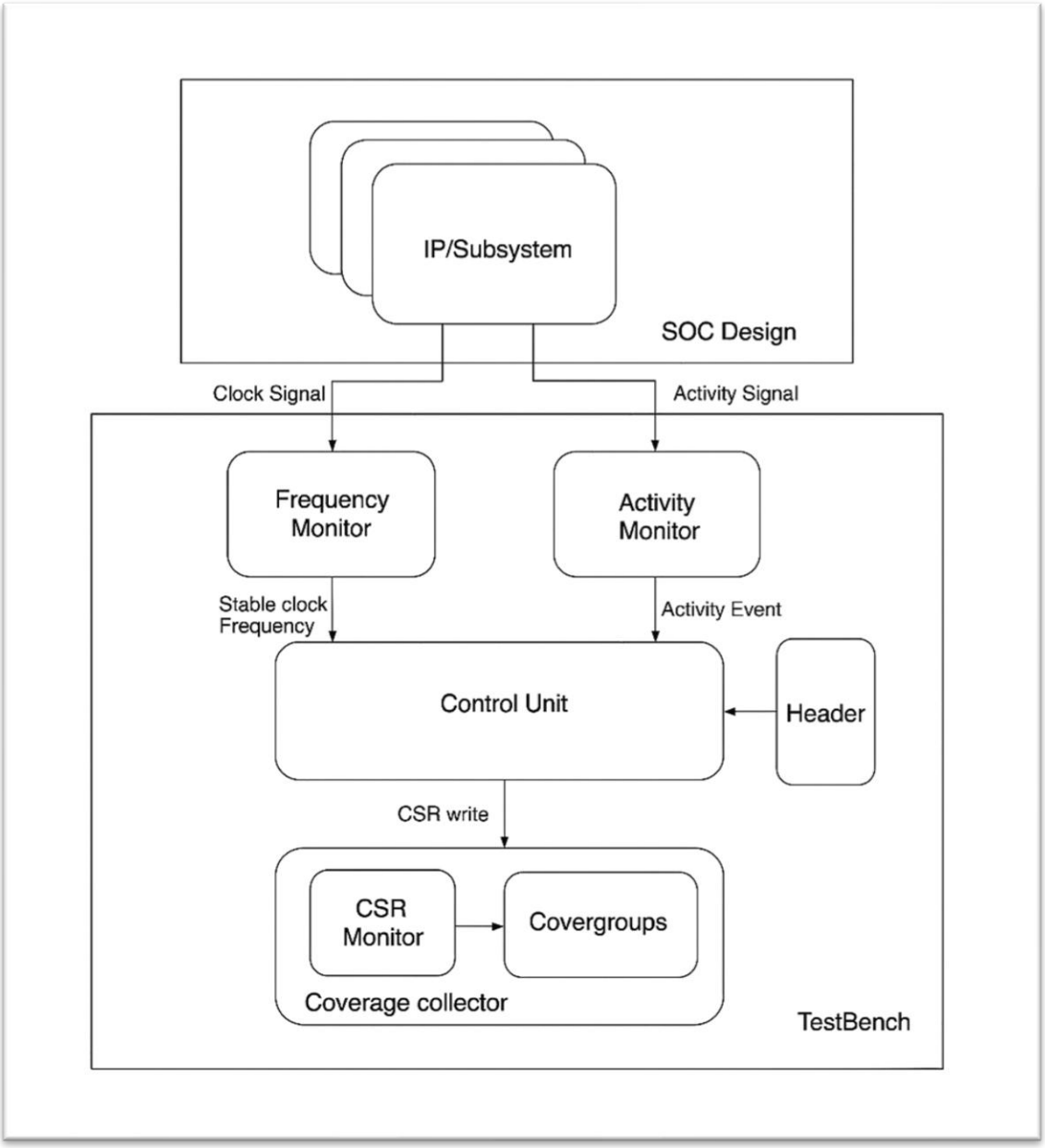


Figure 2. Frequency and Activity Model Architecture

V. DESIGN FLOW SUMMARY

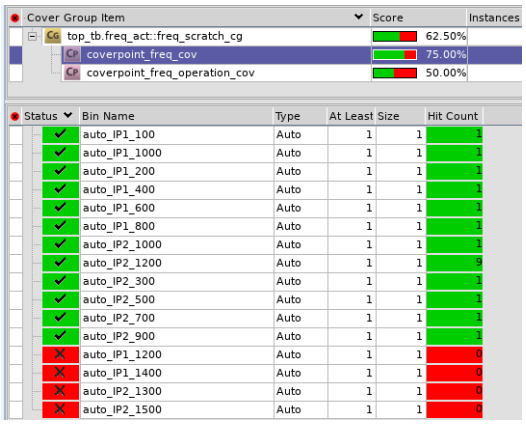
1. IP/Subsystem's clock and activity signals are passed to Frequency and Activity Monitor.
2. Frequency Monitor and Activity Monitor process these signals to produce stable frequency and activity events.
3. Control Unit uses these inputs along with the header-defined ENUMs to write frequency and combination identifiers to the GPR register.
4. CSR Monitor captures valid writes and forwards them to the Covergroups.
5. Covergroups collect coverage data, forming the two matrices:
    - Frequency Coverage Matrix: Tracks individual IP frequency scenarios.
    - Frequency and Operation Coverage Matrix: Tracks meaningful operational combinations across IPs.

VI. RESULTS AND ANALYSIS

Created a dummy subsystem consisting of two IPs, each with a single input clock signal. Targeted verification with 12 different frequency combinations. Out of 12 combinations, only 8 frequency combinations are performed with both IP's operations and for the remaining 4 combinations while both of the IPs remain idle. This is done specifically to catch uncovered combinations in both coverage matrixes. Created dummy requirement of 16 possible valid frequency combinations for the selected subsystem.

A. Frequency Coverage Matrix

- This coverage matrix is shown in Figure 3(a).
- Total defined combinations: 16
- Simulated combinations: 12
- Achieved coverage: 75%
- Gaps identified: 4 combinations never triggered



Cover Group Item	Score	Instances
top_tb.freq_act::freq_scratch_cg	62.50%	
coverpoint_freq_cov	75.00%	
coverpoint_freq_operation_cov	50.00%	

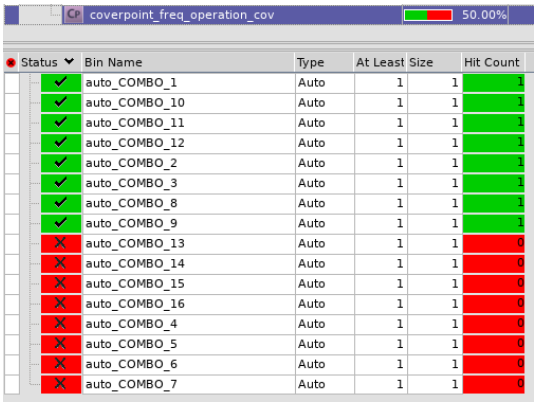
Status	Bin Name	Type	At Least	Size	Hit Count
✓	auto_IP1_100	Auto	1	1	1
✓	auto_IP1_1000	Auto	1	1	1
✓	auto_IP1_200	Auto	1	1	1
✓	auto_IP1_400	Auto	1	1	1
✓	auto_IP1_600	Auto	1	1	1
✓	auto_IP1_800	Auto	1	1	1
✓	auto_IP2_1000	Auto	1	1	1
✓	auto_IP2_1200	Auto	1	1	9
✓	auto_IP2_300	Auto	1	1	1
✓	auto_IP2_500	Auto	1	1	1
✓	auto_IP2_700	Auto	1	1	1
✓	auto_IP2_900	Auto	1	1	1
✗	auto_IP1_1200	Auto	1	1	0
✗	auto_IP1_1400	Auto	1	1	0
✗	auto_IP2_1300	Auto	1	1	0
✗	auto_IP2_1500	Auto	1	1	0

Figure 3(a).

This analysis highlighted scenarios overlooked by directed tests, enabling targeted regression planning.

B. Frequency and Operation Coverage Matrix

- This coverage matrix is shown in Figure 3(b).
- Full coverage (frequency + operation): 8 combinations
- Gaps identified: 8 combinations
- Insight: Some tests applied frequencies but failed to stimulate meaningful activity in one or both IPs.



coverpoint_freq_operation_cov 50.00%

Status	Bin Name	Type	At Least	Size	Hit Count
✓	auto_COMBO_1	Auto	1	1	1
✓	auto_COMBO_10	Auto	1	1	1
✓	auto_COMBO_11	Auto	1	1	1
✓	auto_COMBO_12	Auto	1	1	1
✓	auto_COMBO_2	Auto	1	1	1
✓	auto_COMBO_3	Auto	1	1	1
✓	auto_COMBO_8	Auto	1	1	1
✓	auto_COMBO_9	Auto	1	1	1
✗	auto_COMBO_13	Auto	1	1	0
✗	auto_COMBO_14	Auto	1	1	0
✗	auto_COMBO_15	Auto	1	1	0
✗	auto_COMBO_16	Auto	1	1	0
✗	auto_COMBO_4	Auto	1	1	0
✗	auto_COMBO_5	Auto	1	1	0
✗	auto_COMBO_6	Auto	1	1	0
✗	auto_COMBO_7	Auto	1	1	0

Figure 3(b).

These insights allowed us to refine test vectors to stimulate missing behaviors, revealing corner-case dependencies in timing and initialization sequences.

VII. CONCLUSION

This paper presents a systematic, scalable methodology to verify clock frequency combinations and functional behavior in ASIC SoCs using functional coverage. By defining coverage matrices for frequency and operation, and incorporating architectural constraints, we bridge a critical verification gap not addressed by traditional timing or CDC checks.

REFERENCES

[1] "IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," in IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012) , vol., no., pp.1-1315, 22 Feb. 2018, doi: 10.1109/IEEESTD.2018.