Streamlining Low Power Verification for Multi-die SoCs: A Comprehensive Framework

Jaein Hong, Junha Jeon, Yujin Oh, Moonki Jun, Sungcheol Park, Sanghune Park

Foundry Business, Samsung Electronics Co., Ltd.
1, Samsung-ro, Giheung-gu, Yongin-si, Gyeonggi-do, Republic of Korea, 17113

E-mail: jaein.hong@samsung.com,
junha96.jeon@samsung.com

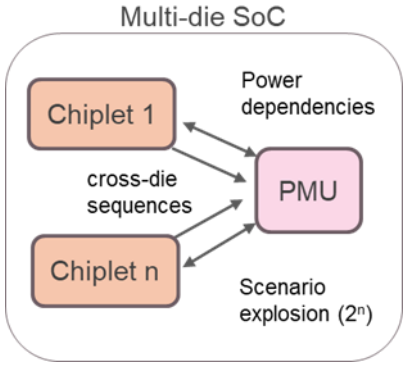
62 THE CHIPS TO SYSTEMS CONFERENCE

SAMSUNG

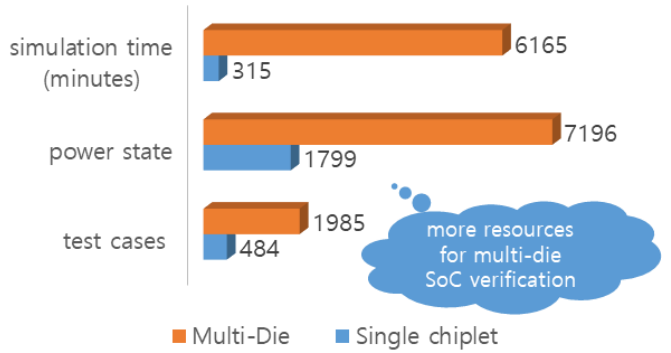
SPONSORED BY CEDA IEEE Council on Electronic Design Automation sigda acm

Motivation

- **The growing importance of power management in multi-die SoCs**
  - Multi-Die SoCs integrate multiple chiplets to meet high performance and energy efficiency demands
  - The Power Management Unit (PMU) plays a critical role in
    - synchronizing power states across chiplets
    - Implementing advanced low-power techniques e.g. power gating, isolation, power rail off, dynamic voltage frequency scaling, multiple supply voltage domain
  - Challenges: Mismanagement of power can lead to energy inefficiency, overheating, and system reliability issues

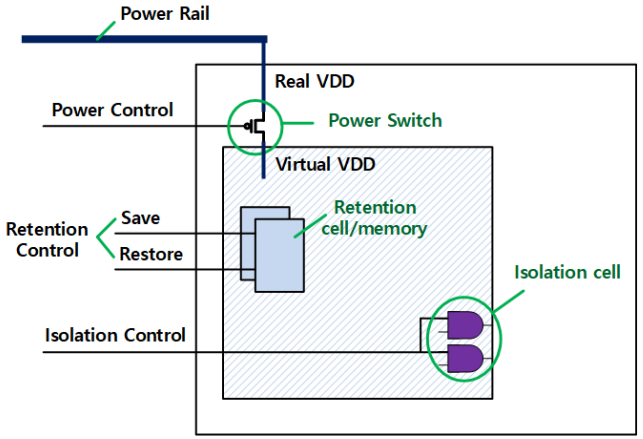


- **Low power verification challenges**
  - Complexity increase:
    - Inter-die power state transitions and communication add verification difficulty
    - Exponential growth in test cases and power states compared to single-die systems
  - Limitation of existing verification methods:
    - Manual, specification-driven processes are time-consuming and error-prone
    - Scalability issues with existing verification methods for multi-die designs

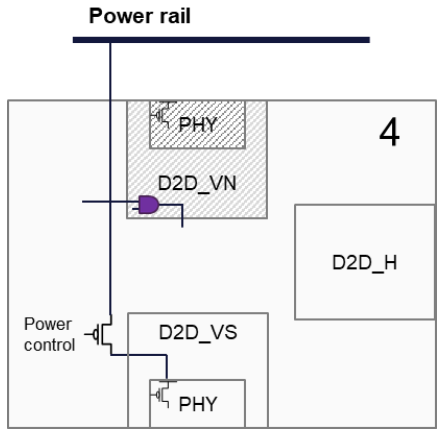


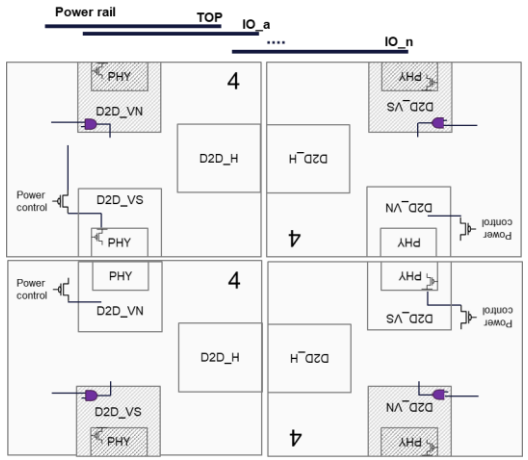
Low Power Verification Scope

- **Low power verification in monolithic SoCs**
  - Key low power verification targets
    - Power control – Power rail on/off sequencing and power gating (PG) logic
    - Isolation control – Domain level isolation for power off scenarios
    - Retention/Restoration – Flop/memory state preservation, GPIO retention during low power modes
    - Clock & Reset – Power mode dependent clock behavior and system level reset control (HW/SW reset) including DVFS and DRAM self-refresh



- **Challenges in multi-die SoCs**
  - Scenario explosion: Exponential growth in test space
  - Shared power rails & merged enable signals
  - Die-level dependency on physical position
  - D2D interface: valid signaling across die power states
  - Cross-die sync & power-good signal timing
  - Manual validation misses inter-die transitions & rare states





Verification Complexity

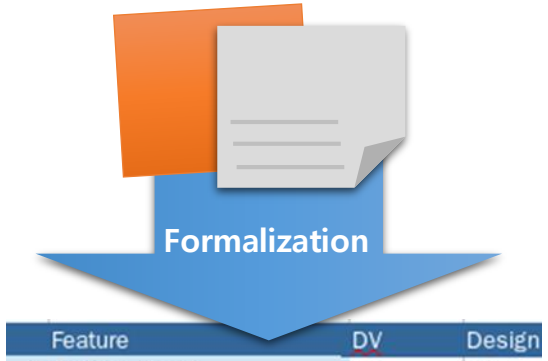
- **Key verification aspects comparison: Monolithic SoC vs. Multi-die SoC**

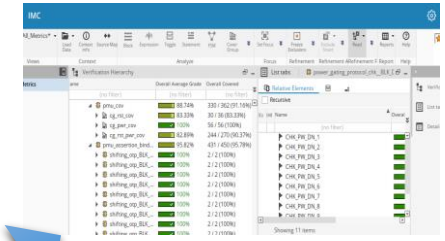
Aspect	Monolithic SoC	Multi-Die SoC
Power Control	Local power rails and control signals	Shared rails and merged enable signals across dies
Isolation	Intra-domain isolation only	Cross-die isolation across different power states
Retention	Single-domain state preservation	Per-die retention with synchronization between dies
Clock & Reset	Local DVFS and reset control	Cross-die reset timing and DVFS coordination
Interface Validity	Valid signals under unified power domains	Signaling across dies with potential power state mismatches
Verification Space	Linear growth with system complexity	Exponential scenario growth with number of dies
Manual Coverage Risk	Generally manageable	High risk of missing rare inter-die transitions

- These structural and functional differences significantly increase verification complexity, making manual approaches impractical for scalable multi-die SoCs

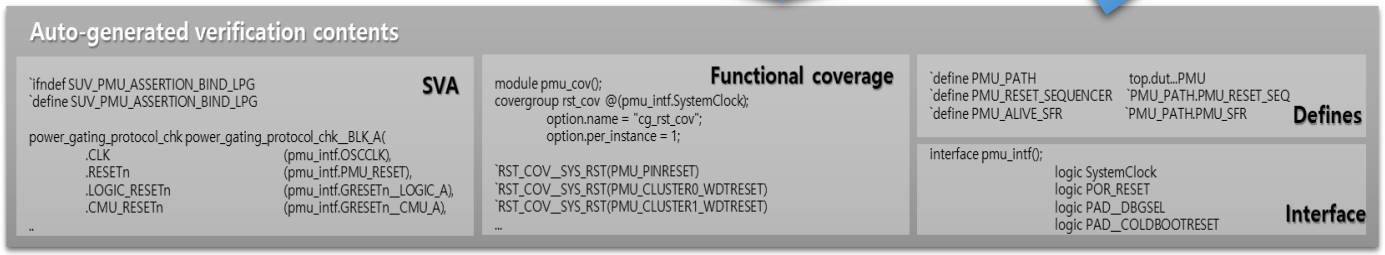
Proposed Verification Framework

- **Automating low-power verification for multi-die SoCs**
  - A fully automated framework enables seamless low-power verification across dies
  - From specification to coverage analysis, each step is script-driven and scalable
- **Key flow components**
  1. **Specification definition**
     - Low-power features are formalized in using metadata
  2. **Verification planning**
     - Checklist-based validation ensures all required conditions are covered
  3. **Environment generation**
     - Automatically builds checkers, coverage models, and testbench interface files
  4. **Simulation and reporting**
     - Scripts handle test execution, coverage analysis, and report generation



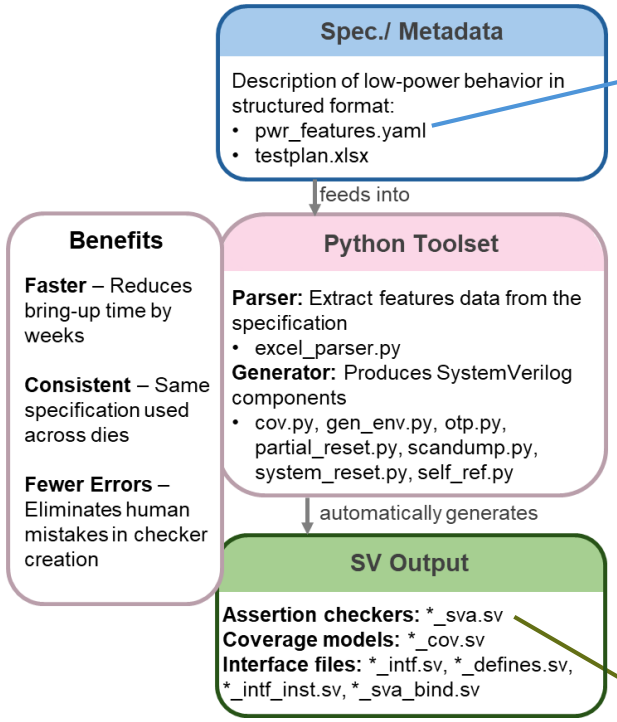


Coverage analysis



Framework components & implementation

- **What powers the framework**
  - Metadata-driven architecture
    - All low-power features are described in structured metadata (e.g., pwr_features.yaml)
    - Enables spec reusability and consistency across dies
  - Script-based generator tools
    - Python-based scripts parse metadata to automatically generate
      - Checkers (e.g., *.sva), Coverage models (e.g., *.cov.sv)
      - Testbench interface files (e.g., *_intf.sv, *_bind.sv)
    - Unified environment
      - The same spec drives all artifacts
      - Guarantees alignment between design intent and test infrastructure
      - Eliminates redundant setup across dies




```
//-------------------------------------------------
// Power Gating Entry (Power Down)
//-------------------------------------------------
// 1.PMLINK handshaking ('h74->'h78->'h5C ->'h11->'h30)
// 2.CEN goes high. (If SRAM is in the block)
// 3.PGEN goes high. (If SRAM is in the block)
// 4.GPIO PDN
// 5.PAD retention
// 4.Output isolation enable. (OUTISO_EN goes high)
// 5.GRESETn goes low (logic reset)
// 6.GRESETn_CMU goes low (cmu reset)
// 7.SCPRE/SCALL goes high. (power switch off)
```

```
##################################################
# System level power gating entry sequence
##################################################
power_gating_entry:
  description: "Power Gating Entry (Power Down)"
  steps:
    - id: 1
      name: "PMLINK handshaking"
      sequence: ["0x74", "0x5C", "0x11", "0x30"]
    - id: 2
      name: "CEN goes high"
      condition:
    - id: 3
```

Example code snippet for power gating features

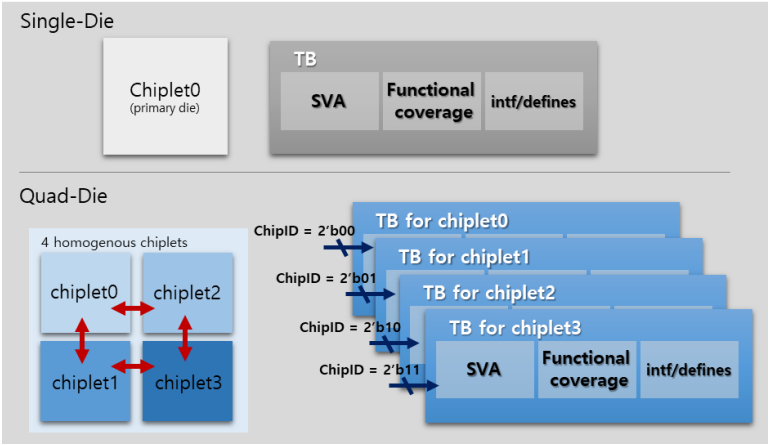
```
//reset chk
property RESET_CHK;
  @(posedge CLK) disable iff(!RESETn||SCANDUMP_EN)
    $fell(IN_RSTn) && (!MASK) |-> ##[0:1000] reset_seq(OUT_RSTn, TOGGLE_RSTn);
endproperty
property NOT_RESET_CHK;
  @(posedge CLK) disable iff(!RESETn||MASK)
    $fell(IN_RSTn) ##[0:1000] $fell(OUT_RSTn) |-> ##0 (STABLE_RSTn)[*1:100];
endproperty

//reset mask chk
property RESET_MASK_CHK;
  @(posedge CLK) disable iff(!RESETn)
    $fell(IN_RSTn) && (MASK) |-> ##0 (STABLE_RSTn&&TOGGLE_RSTn)[*1:$];
endproperty
```

```
RST_CHK_TOGGLE     : assert property (RESET_CHK) $info("PASS: RESET_CHK");
    else $error("RESET_CHK failed");
RST_CHK_NOT_TOGGLE  : assert property (NOT_RESET_CHK) $info("PASS: NOT_RESET_CHK");
    else $error("NOT_RESET_CHK failed");
RST_CHK_MASK       : assert property (RESET_MASK_CHK) $info("PASS: RESET_MASK_CHK");
    else $error("RESET_MASK_CHK failed");
```

Example code snippet of auto-generated reset checker

Application to Quad-die SoC

- **Seamless scaling from single to multi-die**
  - The framework, initially developed for a single chiplet, seamlessly scales to handle multi-die SoCs
  - Verified on a homogeneous 4-die SoC configuration
  - Verification artifacts such as test checkers and coverage models are automatically expanded to ensure full coverage across all dies.
- **Key features of expansion**
  - Metadata Reuse
    - Structured spec format is reused per die, maintaining consistency and avoiding redundant input
  - Chiplet-Specific Scaling
    - The framework supports validation at both the individual die level and the multi-die SoC level
  - Unified Verification Environment
    - All chiplet-specific artifacts are consolidated into a single, cohesive environment, allowing parallel and scalable validation



Evidence

- **Verification artifacts improvements**
  - Checkers: Expanded from 2,264 to 3,152 in the Quad-Die system, achieving a 39% improvement
  - Coverage Models: Increased from 852 to 1,080, reflecting a 27% improvement
  - These enhancements demonstrate the framework's ability to scale effectively for multi-die systems, ensuring comprehensive coverage across all chiplets
- **Development time reduction**
  - Spec-to-Test Cases: Reduced from 12 weeks to 1 week for quad-die systems (~90% faster)
  - Test Environment Setup: Cut from 16 weeks to 6 weeks (~63% faster)
  - Checkers: Generated in 2 weeks instead of 8 weeks, achieving ~75% improvement
  - Coverage Models: Reduced from 3 weeks to 1 week (~67% faster)
  - The framework delivers significant time savings, accelerating project timelines while maintaining accuracy and quality

Verification Artifacts (Quad-Die)

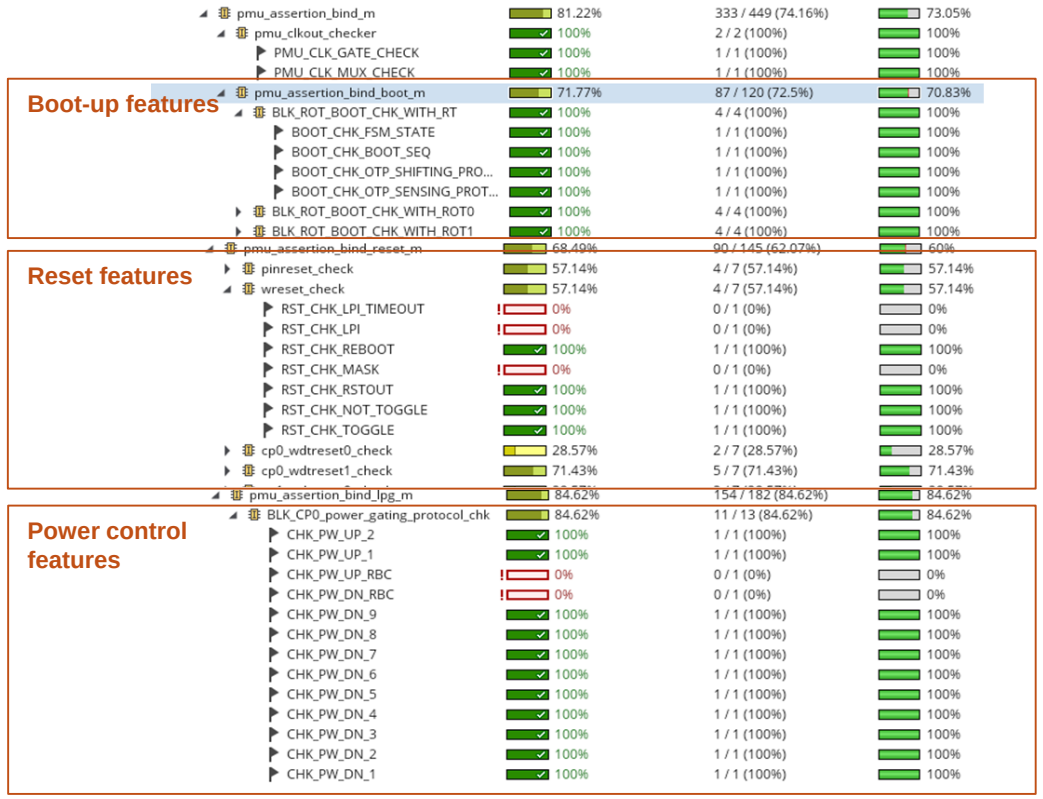
Metric	Existing Methods	Automated Framework	Improvement
Checkers	2,264	3,152	+39%
Coverage Models	852	1,080	+27%

Development TAT (Quad-Die)

Stage	Existing Methods	Automated Framework	Improvement
Spec-to-Test cases	12 weeks	1 week	~90% faster
Test Environments	16 weeks	6 weeks	~63% faster
Checkers	8 weeks	2 weeks	~75% faster
Coverage Models	3 weeks	1 week	~67% faster

Evidence (cont'd)

- **Coverage and quality improvement**
  - Auto-generated coverage helped uncover missing edge cases:
    - Isolation state transitions under partial power
    - Unspecified wakeup timing between dies
  - Coverage improvement: more bins exercised earlier in the cycle
  - Reduced debug effort and faster test closure
- **Enhancements**
  - Accurate Checker Generation
    - Auto-generated checkers eliminated errors caused by manual processes, identifying subtle issues missed by existing methods
  - Enhanced Test Coverage
    - By identifying gaps in coverage, the framework enabled additional testing, which uncovered previously undetected bugs
  - These improvements highlight the reliability and consistency of the automated approach, ensuring higher verification quality



Summary

- **Motivation**
  - Verifying low-power features across complex SoC configurations, especially multi-die systems requires scalable, reliable, and automated methods to avoid manual errors and improve efficiency
- **Methodology**
  - We developed a formalized and automated validation framework that integrates specification-driven checker generation, coverage modeling, and reporting. This framework supports both homogeneous and heterogeneous SoCs through per-die customization
- **Key takeaways**
  - Scales from single-die (788 checkers) to multi-die (3,152 checkers) configurations
  - Achieved up to 90% reduction in development time for key verification stages
  - Improved early bug detection via auto-generated checkers and coverage analysis
  - Proven reliability through formal spec management and minimized manual effort
  - Framework extensible to heterogeneous SoCs with per-die rules