基于形式验证的低概率边界条件识别方法研究

孙逸睿[1,2]，郝一鸣[2]，谢昕洋[1,2]，徐畅[2]，苏国彬[2]，赵双妹[2]，罗力川[2,*]

[1] 电信科学技术研究院，北京 100083
[2] 宸芯科技股份有限公司，北京 100083

摘要： 随着信息快速发展，芯片设计复杂度持续攀升，内部数据通路与控制逻辑高度耦合，片上系统（SoC）集成密度急剧增加。传统验证方法学在扩展、复用及平台标准化等方面存在局限，难以在有限资源内实现对全部芯片行为高效覆盖。尽管已有研究借助大规模自动化工具，通过通用验证方法学（UVM）增强随机测试与回归仿真，仍难以在项目周期内高效覆盖可能引发致命故障的极端边界场景，致使验证存在盲区。本文提出一种动态仿真与形式验证（FPV）相结合的混合验证方法，以 PWR 模块的关键路径为例进行穷尽分析。首先，基于 UVM 构建动态仿真平台，通过随机测试对模块主要功能与常规场景进行验证，覆盖率达到 94%以上；随后，采用 FPV 方法编写关键属性断言，对设计进行数学化分析，有效识别出 UVM 难以覆盖的边界条件，推导出设计中隐含的使用约束，进而揭示出潜在缺陷。实验表明，该方法高效发挥动态仿真与形式验证的互补优势，显著提升验证完备性和效率，为芯片设计可靠性提供有力保障。

关键词： 形式验证；边界条件; SoC 验证；属性验证; FPV; UVM

DOI: 10.57237/j.cst.2025.04.002

Formal Verification-Based Identification of Low-Probability Corner Cases

Sun Yirui[1,2], Hao Yiming[2], Xie Xinyang[1,2], Xu Chang[2], Su Guobin[2], Zhao Shuangmei[2], Luo Lichuan[2,*]

[1]China Academy of Telecommunication Technology, Beijing 100083, China
[2]Morningcore Holding Co., Ltd., Beijing 100083, China

Abstract: With the rapid development of information technology, the complexity of chip design continues to increase, with highly coupled internal data paths and control logic as well as sharply rising integration density in System-on-Chip (SoC), which impose greater demands on verification methodologies. Traditional approaches show inherent limitations in scalability, reusability, and platform standardization, making it difficult to efficiently achieve comprehensive coverage of all chip behaviors within limited resources. Although large-scale automated tools and the Universal Verification Methodology (UVM) have been widely adopted to enhance random testing and regression simulation, they still fail to efficiently capture extreme boundary scenarios that may lead to fatal failures, leaving potential blind spots in verification. To address this challenge, this paper proposes a hybrid verification method that

*通信作者：罗力川，luolichuan@buaa.edu.cn

收稿日期: 2025-08-25；接受日期: 2025-09-13；在线出版日期: 2025-10-16

combines dynamic simulation with Formal Property Verification (FPV), using the critical path of the PWR module as an example for analysis. A UVM-based verification platform is constructed to execute randomized test cases on key functions and typical scenarios, achieving more than 94% coverage in line, toggle, branch, and functional metrics, which confirms the completeness of basic functional validation but also indicates the difficulty of covering corner cases. Then, FPV is applied by formulating property assertions, such as state transition constraints, and performing exhaustive mathematical analysis of the Register Transfer Level (RTL) design, thereby detecting extremely low-probability boundary conditions and revealing implicit usage restrictions in design logic. By incorporating these restrictions into formal constraints, the verification results are corrected and latent design defects are confirmed. This hybrid method effectively leverages the complementary strengths of UVM dynamic simulation and FPV formal analysis, significantly improving the detection capability of corner cases, enhancing verification completeness and efficiency, and strengthening design reliability, while also providing a theoretical basis and practical reference for optimizing verification strategies in future SoC design projects.

Keywords: Formal Verification; Corner Case; SoC Verification; Property Verification; FPV; UVM

1 引言

随着信息技术的快速发展，集成电路设计规模持续扩大，SoC 集成度不断提高，验证工作面临诸多挑战[1]。一方面，芯片设计从单一功能模块发展为多模块、多时钟域协同的复杂系统，设计内部的数据通路和控制逻辑高度耦合，传统的验证方法难以有效覆盖全部设计行为；另一方面，芯片在低功耗、高性能、高可靠性等多目标约束下，系统架构越发复杂，导致验证所需资源和周期大幅上升[2]。早期的验证方法学如 VMM、OVM，虽在特定项目中取得成果，但仍存在扩展性差、重用率低、平台规范不统一等问题。因此业界提出了 UVM 作为统一的验证架构，现已成为复杂 SoC 验证领域的事实标准[3, 4]。李智超[5]等人通过在序列组件中采用面向对象的层次化建模方法，简化了验证平台产生激励的复杂度，提高了验证效率；王鑫[6]等人基于 UVM 搭建验证平台，并设计验证方案，采用约束随机测试为主，辅以定向测试的方式对片上网络进行验证。

然而，传统 UVM 验证平台存在两大瓶颈：算法实现复杂和验证效率不足，这导致逐渐无法满足更加严苛的验证需求，推动了国内外学者将其与其他技术相结合做研究[7]。彭莉[8]等人基于 DeepSeek 大模型智能验证框架的创新解决方案，重构芯片验证覆盖率收敛范式；Abelrahman [9]等人设计了一款 ART 工具，能自动创建 UVM 验证平台，生成的代码结构可以由用户自定义或者自动从 RTL 派生。

但上述研究均未对边界条件（corner case）做详细验证，corner case 往往代表着在极端或低概率输入下可能触发的异常行为，其在芯片实际应用中虽发生概率极低，却可能引发严重的系统性故障，甚至导致整个芯片功能失效[10]。因此，在芯片设计阶段对 corner case 进行充分挖掘和验证，对于提升设计可靠性和产品稳定性具有重要意义。而传统的基于 UVM 的动态仿真，使用大量随机测试、直接用例和长时间回归测试，不仅拉长验证周期，增大资源消耗，还难以遍历所有可能的行为组合[2, 11, 12]。因此本文以 PWR 模块为研究对象，基于 UVM，创新性结合形式验证工具 FPV 对模块关键控制路径进行穷尽性分析，对其 corner case 做检验，并进一步分析其触发机制，提出有效约束方案，证明形式验证在 corner case 覆盖方面的优势与价值。

2 基于 UVM 验证

2.1 PWR 模块

PWR 模块是实现功耗控制、时钟管理、复位控制及电源管理的模块，位于芯片系统控制路径的核心位置，与功耗管理单元、SOC 控制器以及其他外围控制模块协同工作，通过一系列状态机和控制逻辑，实现芯片在不同电源域之间的切换与控制。模块内部设计包含多个计数器、控制寄存器及组合逻辑，存在较多异步触发路径和状态交叉，设计复杂度较高。其行为逻辑正确与否将直接影响整个芯片在切换多工作模式时时序是否一致。

2.2 基于 UVM 的验证

图 1 为动态仿真验证流程：(1) 分解测试点，编写对应的直接用例和随机用例；(2) 搭建验证平台并展开仿真；(3) 通过仿真过程与结果判断设计是否满足功能需求；(4) 若功能满足则收集覆盖率，保证验证完备性。为确保 PWR 模块功能的正确性，本文基于 UVM 搭建动态仿真验证平台，平台结构如图 2 所示。

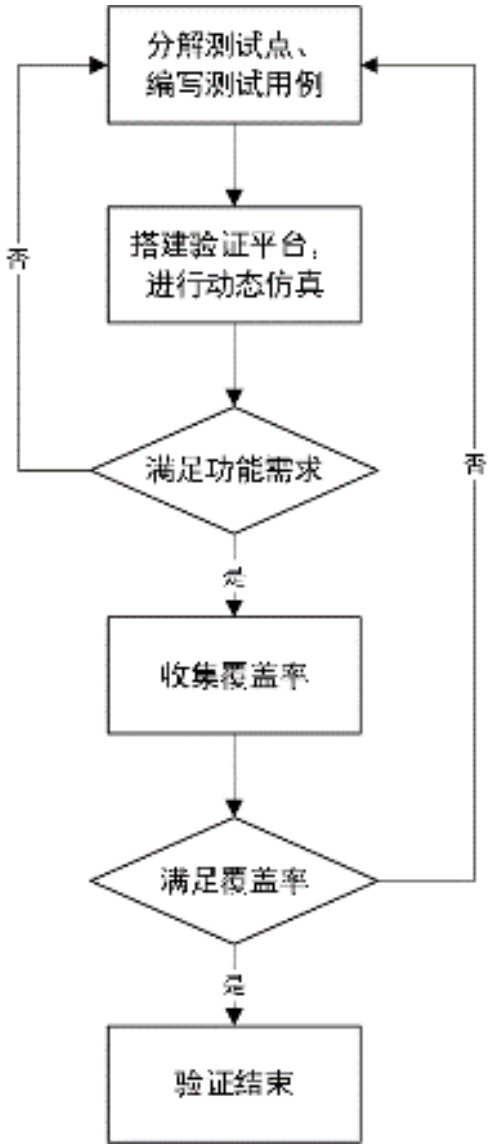

图 1 动态仿真流程图

PWR_ENV 层包含了各功能模块的组件、接口声明（include 文件）与环境参数配置文件（configuration 文件），构成完整基础的验证平台结构。

RAL（Register Abstraction Layer）为寄存器抽象层模型，用于对 PWR 模块内的寄存器空间进行访问与配置，即进行读写操作，通过控制寄存器可以改变 DUT（Design Under Test，待验设计）的工作状态。

seq 为测试序列。针对 PWR 模块的功能特性，编写多个测试用例，并根据种类封装为多种测试序列。其中 pwr_reset_powerup_seq 为验证上电、解复位等多个特定功能的定向测试序列，以确保模块在关键阶段的行为与时序符合设计规格；而功能性序列以及 DUT 与 APB 总线交互序列则为随机测试序列，生成多种状态组合以提升验证覆盖率，实现更全面的功能验证。

验证平台采用 virtual sequence 机制，以协调组件之间的交互，它本身不直接驱动接口，而是在顶层统一调度多个下层 sequence 的执行顺序与参数传递，解决跨接口、多场景等复杂问题。

APB_IF 和 PWR_IF 为两个接口。前者为 APB 总线信号；后者为 PWR 模块端口上 APB 信号以外的所有信号，包括时钟信号、复位信号、电源管理信号、睡眠信号、低功耗控制信号、中断信号、其他控制信号等。

apb_master_agent 是一个主动 agent，为 APB 的 VIP（Verification IP，验证用知识产权核），通过寄存器模型将各测试用例中的寄存器操作转换为真实的寄存器操作。其中 sequencer 负责生成并调度测试激励；driver 负责接收激励并将其转换为 DUT 可识别的信号，驱动接口完成相应操作；monitor 负责监视信号，并从中提取 transaction 用于检查。

另外，验证平台还包含一个 cfg 文件，用来向寄存器模型传递参数和验证变量，验证前需先 source 此文件内的 cshrc 文件，以完成环境变量的配置，同时初始化验证过程中使用到的工具。

针对模块功能，分解测试点并编写测试用例，包括寄存器验证、上电时序验证、时钟验证、复位验证、睡眠与唤醒状态转换验证、中断验证低功耗控制验证和电源管理验证。同时，为确保验证质量，在测试用例全部通过后的回归阶段，采取 CDR（Coverage-Driven Regression，覆盖率驱动回归）策略收集覆盖率，如图 3 所示。其中图 3b)，由于“进入睡眠状态”与“未进入睡眠状态”在逻辑上属于互斥关系，因此出现四个未覆盖点。

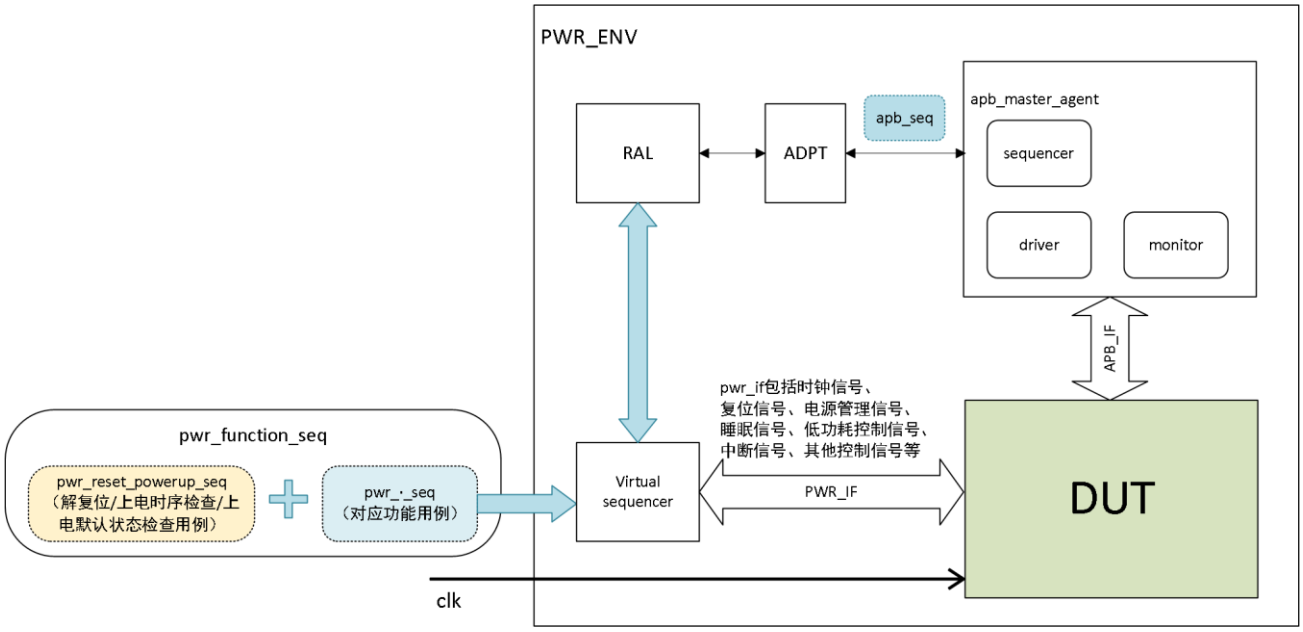

图 2 PWR 模块验证平台结构框图

Name	Score	Line	Toggle	FSM	Condition	Branch	As
tb_top	76.87%	98.43%	83.37%	88.13%	95.36%	95.96%	
U_ap_pwr_top	94.40%	98.80%	93.62%	88.13%	95.39%	96.06%	
U_ap_pwr_glue_logic	99.11%	99.75%	96.87%	100.00%	99.61%	99.33%	
U_pll_ctl	94.35%	100.00%	79.25%		98.16%	100.00%	
U_pmu	92.98%	98.85%	82.61%	89.58%	99.11%	94.74%	
U_pslave	96.58%	98.79%	93.48%		97.96%	96.10%	
U_pwr_clkgen	94.29%	98.52%	95.50%	94.38%	87.89%	95.15%	
U_rsu	99.64%	99.46%	100.00%	100.00%	99.45%	99.29%	
U_slpu	88.58%	99.53%	96.40%	52.24%	96.30%	98.41%	

a) 总体覆盖率

Hierarchy | Modules | Groups | Asserts | Statistics | Tests

Avg. Group Score:94.79% U+C:735 U:23 C:712 X:0
Avg. Group Inst. Score:94.79% U+C:735 U:23 C:712 X:0

Group	Score	Instances	U+C	U	C	X	Goal	Weight	AtLeast	PerInst	Overlap	AutoBin	Missing	Comment
tb_top::ap_pwr_slp_en_with_slp_enter0_seq::CovSlp_slp_en_withe_slp_enter0	100.00%		8	0	8	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_en_with_slp_enter1_seq::CovSlp_slp_en_withe_slp_enter1	100.00%		8	0	8	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_en_with_slp_enter2_seq::CovSlp_slp_en_withe_slp_enter2	100.00%		8	0	8	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_en_with_slp_other_seq::CovSlp_slp_en_withe_slp_other	100.00%		11	0	11	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_intr_mk_seq::CovSlp_slp_enter_intr_mk	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_intr_seq::CovSlp_slp_enter_intr	100.00%		24	0	24	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_26msync_seq::CovSlp_enter_slp_stop_26msync	90.62%		14	3	11	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_32ksync_seq::CovSlp_enter_slp_stop_32ksync	75.00%		14	5	9	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_cnt0_seq::CovSlp_enter_slp_stop_cnt0	75.00%		14	5	9	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_cntfull_seq::CovSlp_enter_slp_stop_cntfull	75.00%		14	5	9	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_maskraise_seq::CovSlp_enter_slp_stop_maskraise	75.00%		14	5	9	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_enter_stop_seq::CovSlp_enter_slp_stop	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_fp1_2dramwk_seq::CovSlp_fp1_2dramwk	100.00%		44	0	44	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_fp1_cond_seq::CovSlp_idle_mask	100.00%		42	0	42	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_int_wake_pll_async_seq::CovSlp_int_wake_pll_async	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_int_wake_pll_produce_seq::CovSlp_int_wake_pll	100.00%		388	0	388	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_intr_mask_seq::CovSlp_slp_intr_mask	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_slp1_2dramwk_intr_mk_seq::CovSlp_slp1_2dramwk_intr_mk	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_slp1_2dramwk_intr_seq::CovSlp_slp1_2dramwk_intr	100.00%		24	0	24	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_wkfp_2dramwk_intr_mk_seq::CovSlp_wkfp_2dramwk_intr_mk	100.00%		14	0	14	0	100%	1	1	0	1	64	64	
tb_top::ap_pwr_slp_wkfp_2dramwk_intr_seq::CovSlp_wkfp_2dramwk_intr	100.00%		24	0	24	0	100%	1	1	0	1	64	64	

b) 功能覆盖率

图 3 基于 UVM 验证覆盖率

3 基于形式验证

基于 UVM 的动态验证本质上是一种“采样式验证”，其测试用例的编写十分依赖人工，因此很难保证可能的输入与状态组合已全部触发。而 PWR 模块作为芯片功耗与时钟管理的核心部分，其控制信号往往与多个子模块进行交互，状态转换情况复杂。所以为保障设计的鲁棒性与一致性，采用形式化验证方法对模块的关键控制路径、状态转换逻辑以及潜在 corner case 进行全面的数学化分析。

3.1 形式验证

形式验证提供了从数学上证明系统正确性的方法和技术，通过静态分析，穷尽系统的所有状态，全面分析逻辑与行为，发现潜在缺陷[13]。例如定理证明和模型检查[14]。定理证明通过将系统和规格说明形式化的转换为逻辑公式，并借助工具推理，验证系统是否满足要求；模型检测将系统建模为有限状态机，用逻辑公式和工具遍历所有可能状态，检查是否违反规格说明，其能穷尽验证系统行为，特别适用于硬件设计和协议验证，有效发现并发与时序问题，保证系统可靠性[13]。

3.2 FPV

FPV 工具不依赖验证平台的搭建及测试用例的编写，而是基于属性进行遍历式验证，证明 SVA（System Verilog Assertion）断言中定义的属性是否被 RTL 逻辑所满足[15]，即根据数学规则对问题建立数学模型，工具会根据此数学模型，分析所有 DUT 行为的可能性来证明此设计的正确性，或是举出反例来证伪，从理论上来说，形式验证方法可以 100%覆盖设计状态的所有子集[15]。

FPV 是一个基于属性的形式验证工具，用于对 RTL 设计进行遍历性验证。其工作时需要三个输入文件：RTL 设计代码、SVA 文件，以及用于流程控制的 Tcl 脚本。其中 SVA 文件用来定义期望满足的验证属性，通常包括断言（Assertions）、假设（Assumptions）以及覆盖（Cover properties）；Tcl 脚本用于执行具体的验证流程，包括 RTL 的分析与综合（Analyze&Elaborate）、时钟与复位的创建（create clock/reset）、形式验证的启动（Run check_fv）。

FPV 通过构建设计的状态空间模型，对所有可能的输入组合和状态转换情况进行验证。针对每一条 SVA 文件中编写的属性，判断其在所有情况下是否始终成立，从而输出详细的验证结果：对于断言属性，FPV 会报告其是否“证明通过（Proven）”、“被反例否定（Falsified）”或“被非空路径覆盖（Non Vacuous）”；假设属性反馈其是否可覆盖、不可达或存在 vacuous 路径；覆盖属性与假设属性相似。此外，若发现某一属性未验证通过，FPV 能提供完整的反例波形以定位问题所在。其所需的输入与输出如图 4 所示。

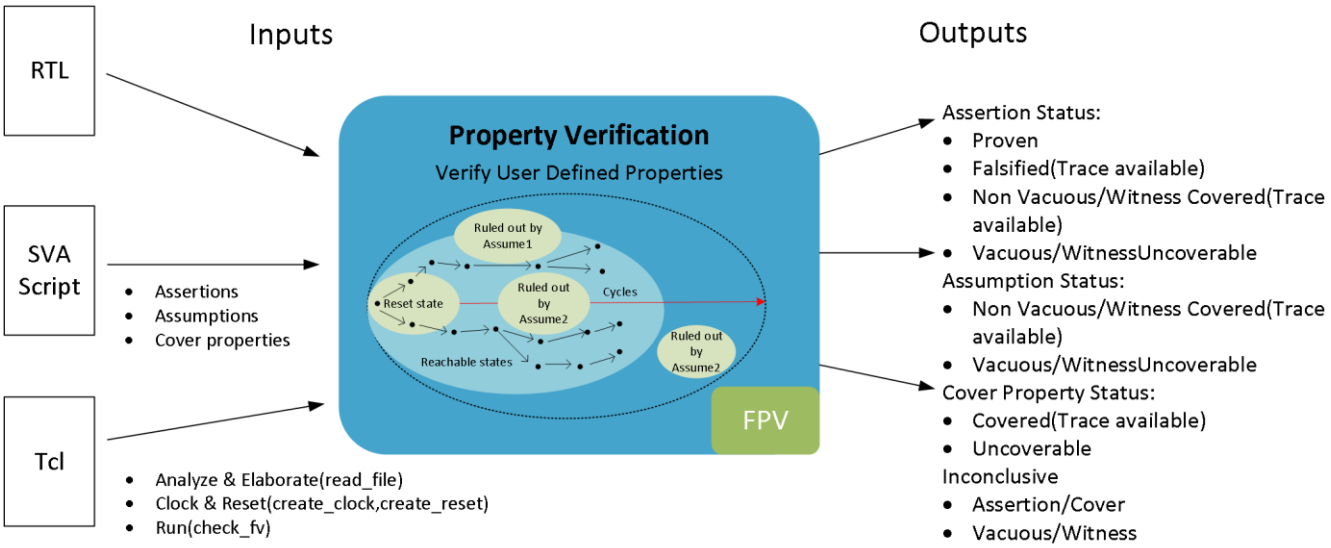

图 4 FPV 输入与输出图示

图 5 为 FPV 的验证流程：(1) 使用 vcf -f run.tcl -verdi 命令启动 FPV 工具，加载设计文件与属性描述；

(2) 对输入设计与断言进行解析和编译，形成可验证模型；(3) 根据验证需求添加必要的约束条件，以避免不合理场景影响结果；(4) 执行属性证明过程；(5) 若产生失败断言，则追踪反例波形并调试。

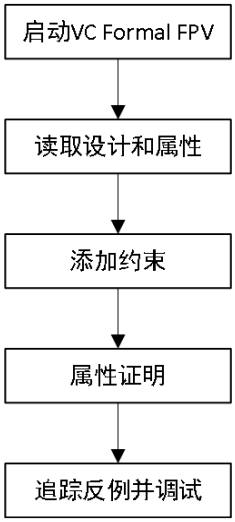

图 5 FPV 验证流程

3.3 属性编写

属性是根据设计的行为进行形式化描述，主要包括三类：断言属性、假设属性和覆盖属性，采用 SVA 语言实现。

SVA 是一种描述性语言，可以描述时序相关的情况。语言的描述性本质提供了对时间卓越的控制，语言本身非常精确且易于维护，SVA 也提供了若干个内嵌函数来测试特定的设计情况，并且提供了一些构造以自动收集功能覆盖数据[16]。

结合 PWR 模块的功能特点，对状态转换的多种组合情况进行属性编写。如针对状态机由 BOOT 状态跳转至 WAKE_FP 状态的情况，编写了三条断言检查：检查解复位后，状态机的初始状态是否为预期状态；检查内部逻辑完成前，状态跳转的指示信号是否始终保持低电平；检查状态跳转的指示信号被拉高后，状态机是否已跳转至预期状态。

通过 FPV 对每个属性的分析，成功捕捉到一个 corner case。图 6 为 FPV 验证结果。

```
Summary Results
 Property Summary: FPV
 -----------------
 > Assertion
   - # found        : 15
   - # proven       : 13
   - # falsified    : 2

 > Vacuity
   - # found        : 15
   - # non_vacuous  : 15

 > Witness
   - # found        : 12
   - # covered      : 12

 > Constraint
   - # found        : 19
```

a)

```
List Results
 Property List:
 -------------
 > Assertion
   # Assertion: 15
   [       0] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_BOOT_to_BOOT
   [       1] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_BOOT_to_WAKE_FP
   [       2] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_DRAM_WAKE_to_DRAM_WAKE
   [       3] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_DRAM_WAKE_to_WORK
   [       4] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_WAKE_FP_to_DRAM_WAKE
   [       5] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_WAKE_FP_to_WAKE_FP
   [       6] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_WKFP_2DRAMWAKE_INTR
   [       7] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_dram_wake_cnt_hold
   [       8] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_dram_wake_count
   [       9] falsified   (depth=7)   (non_vacuous)  -  ap_pwr_top.chk.ast_osc_stable_count
   [      10] falsified   (depth=7)   (non_vacuous)  -  ap_pwr_top.chk.ast_osc_stable_hold
   [      11] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_pll_stable_count
   [      12] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_pll_stable_hold
   [      13] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_reset_state
   [      14] proven                  (non_vacuous)  -  ap_pwr_top.chk.ast_step_check
```

b)

```
> Constraint
  # Constraint: 19
  [  115819] constrained                          -  fault_stbl_constr_0 - STABLE(cerfaultenable)
  [  115820] constrained                          -  constant_115820 - prst_n==1
  [  115821] constrained                          -  constant_115821 - test_mode==0
  [  115822] constrained                          -  constant_115822 - sys_clk26m_sel==1
  [  115823] constrained                          -  constant_115823 - pll0_bp_test_mode==0
  [  115824] constrained                          -  constant_115824 - U_pslave.pll_stable_time0[0]==1'b0
  [  115825] constrained                          -  constant_115825 - U_pslave.pll_stable_time0[1]==1'b1
  [  115826] constrained                          -  constant_115826 - U_pslave.pll_stable_time0[2]==1'b1
  [  115827] constrained                          -  constant_115827 - U_pslave.pll_stable_time0[3]==1'b1
  [  115828] constrained                          -  constant_115828 - U_pslave.pll_stable_time0[4]==1'b1
  [  115829] constrained                          -  constant_115829 - U_pslave.pll_stable_time0[5]==1'b0
  [  115830] constrained                          -  constant_115830 - U_pslave.pll_stable_time0[6]==1'b0
  [  115831] constrained                          -  constant_115831 - U_pslave.pll_stable_time0[7]==1'b0
  [  115832] constrained                          -  constant_115832 - U_pslave.dram_wake_perido[0]==1'b0
  [  115833] constrained                          -  constant_115833 - U_pslave.dram_wake_perido[1]==1'b1
  [  115834] constrained                          -  constant_115834 - U_pslave.dram_wake_perido[2]==1'b1
  [  115835] constrained                          -  constant_115835 - U_pslave.dram_wake_perido[3]==1'b1
  [  115836] constrained                          -  constant_115836 - U_pslave.dram_wake_perido[4]==1'b1
  [  115837] constrained                          -  constant_115837 - U_pslave.dram_wake_perido[5]==1'b0
```

c)

图 6 FPV 验证结果报告

4 Corner Case 问题解决

4.1 问题发现

根据 PWR 模块功能特性，可通过配置寄存器来指定状态 1 与状态 2 之间的跳转条件。其具体行为表现为：向寄存器写入配置值 config value，内部计数器开始计数，在计数器尚未达到该配置值之前，状态跳转指示信号 osc_stable 始终保持低电平；当计数器计数到该配置值时，osc_stable 信号才会被拉高，指示状态跳转完成。跳转情况如图 7 所示。

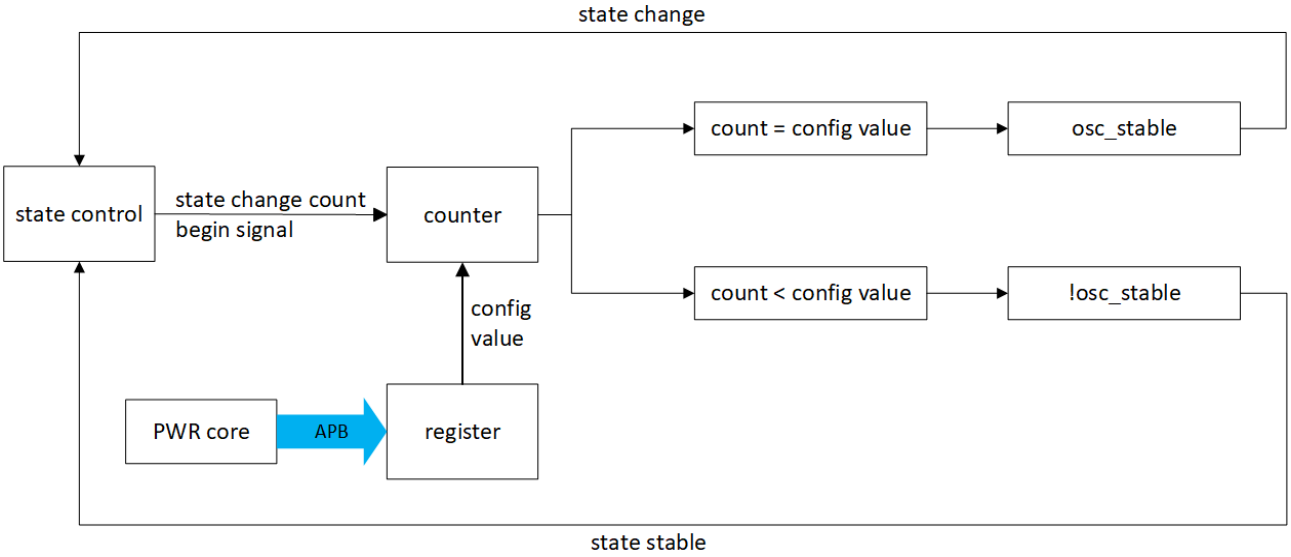

图 7 模块状态跳转逻辑示意图

针对此情形设置如下断言：

property p_osc_stable_hold;

@(posedge ap_pwr_top.U_slpu.clk32k) disable iff (!ap_pwr_top.U_slpu.rstn)

$rose(start_BOOT)|->(!$past(`AP_PWR_SLP_FSM.osc_stable,1)throughout((`AP_PWR_SLP_FSM.pll_state_R==`PLL_BOOT)[*0:$]##1(`AP_PWR_SLPU.slp_int_cnt!7:0]==`AP_PMR_PSLAVE.osc0_stable_time[7:0])));

endproperty

如图 8 所示，该条断言检查失败。并产生图 9 的反例波形，左侧信号名称一栏，ast_osc_stable_hold 信号名称前出现向上箭头，表示该信号造成断言失败，右侧波形栏的横线表示断言失败产生的时间。

4.2 问题解决

断言失败的原因在于计数器的计数值并未达到配置值，但 osc_stable 信号被错误拉高。进一步分析发现，RTL 代码存在两个使用限制：

寄存器配置值不能为 2；

由于跳转指示信号 osc_stable 相较于计数逻辑存在一拍延迟，当内部计数器的计数值等于配置值减一时，不能重新配置寄存器，即此时不允许向寄存器写入新数据，否则可能导致计数尚未完成，但跳转信号 osc_stable 被提前拉高造成逻辑异常。

为验证断言失败是否与上述限制有关，在 SVA 文件中添加对应约束，以阻止上述受限行为发生。编写约束如下所示：

Assume

assume_count_not_change:

assume property (p_count_not_change);

assume_count_not_2:

assume property (p_count_not_2);

重新运行后，断言检查成功，如图 10 所示。证明该限制条件导致 corner case 发生。

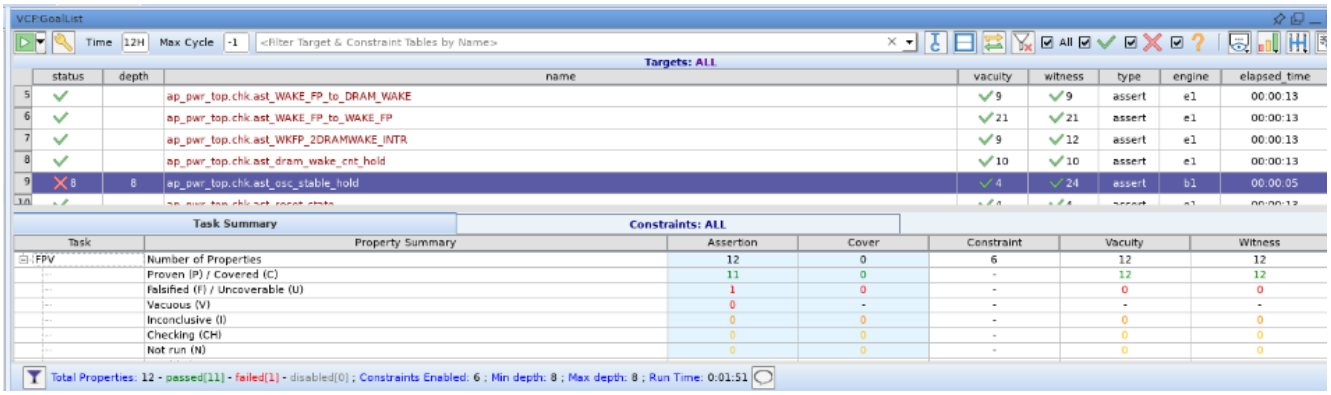

图 8 失败断言

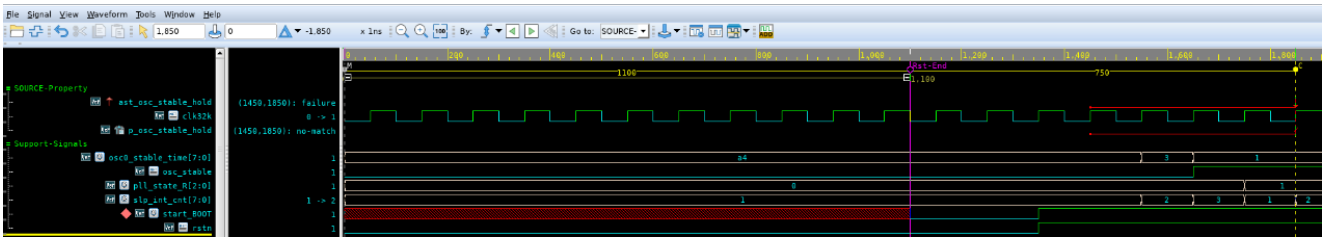

图 9 断言失败触发的反例波形

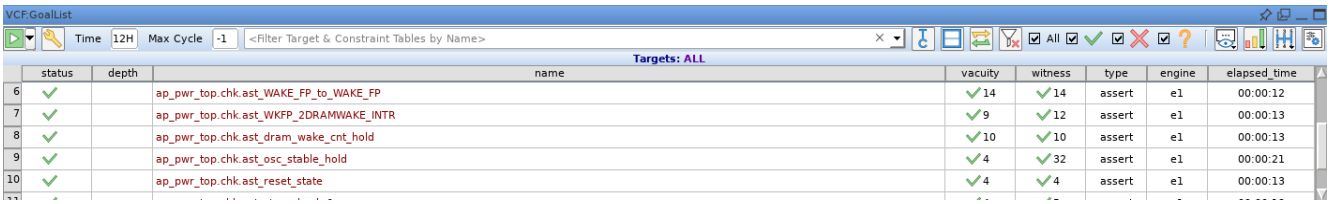

图 10 FPV 断言成功结果图

5 结论

针对现有研究对低概率边界条件关注不足的问题，本文在完成基于 UVM 的功能性验证后，引入形式化验证对 PWR 模块的关键控制路径展开穷尽性分析。动态仿真结果显示，模块验证覆盖率总体较高，如 ap_pwr_glue_logic 语句覆盖率达 99.75%，rsu 状态机与分支覆盖均接近 100%。然而，在 UVM 已达成期望覆盖率的前提下，形式验证工具 FPV 仍成功识别出动态仿真未触及的低概率触发路径，进而找到潜在的设计使用限制与风险。该结果表明，动态仿真与形式验证具备显著互补性：前者通过定向与随机测试高效实现基本功能覆盖；后者在极端状态与深层边界条件识别方面表现出更强能力。两者结合有效提升了验证完备性与设计可靠性。

基于上述发现，本文形成了面向 corner case 的验证思路：先基于 UVM 完成基础功能与主要场景的覆盖，再以 SVA 属性驱动形式验证工具如 FPV 对状态空间进行系统扫描，定位异常触发序列，进而将存在的风险与使用条件反馈至设计与后续验证环节。该方法为 UVM 的直接测试用例编写与约束配置提供了依据，同时也明确了设计可完善的接口约束与使用规范，从工程角度大大提升了验证前瞻性与设计可靠性。

综上，本文基于 UVM、使用 FPV 工具，验证了动态仿真结合形式验证的方式在挖掘 corner case 与提升验证完备性方面的可靠性。该实践对控制逻辑复杂、状态交互密集的模块具有可推广价值，也为后续芯片项目在验证策略制定与资源配置上提供实践参考。

参考文献

[1] 田劲, 王小力. 基于 UVM 验证方法学的 AES 模块级验证[J]. 微电子学与计算机, 2012, 29(08): 86-90.

[2] 曹庆年, 姚翌, 孟开元. 数字电路芯片验证方法研究综述 [J]. 工业控制计算机, 2024, 37(02): 53-55+83.

[3] 李晨思. 基于 UVM 的 AMBA 总线桥及通信接口的验证研究 [D]. 西安: 西安建筑科技大学, 2024.

[4] 李子豪. 基于 UVM 的 USB 接口 IP 核验证平台的设计与应用 [D]. 北京: 北方工业大学, 2024.

[5] 李智超, 刘冬培, 吕平, 等. 基于 UVM 的可重用协议转换模块验证平台设计 [J]. 信息工程大学学报, 2024, 25(06): 653-659.

[6] 王鑫, 张畅. 基于 UVM 的片上网络验证平台设计 [J]. 计算机测量与控制, 2025, 33(03): 323-329.

[7] 徐晓贝. 基于 UVM 和 Python 的 RSFEC 验证平台 [J]. 中国信息界, 2025, (05): 223-225.

[8] 彭莉, 王定. DeepSeek 加速 GPU 芯片验证覆盖率收敛技术的应用 [J]. 集成电路应用, 2025, 42(05): 62-63.

[9] SABRY A, SAID A, MOHAMMAD A, et al. [ART]: A Novel Approach to Automated UVM Testbench Generation [C]//2024 International Conference on Microelectronics (ICM). IEEE, 2024: 1-5.

[10] OWEN A. Corner Case Coverage Challenges in Stress Testing DDR Controllers under Variable Load Conditions [J]. 2024.

[11] 邵奇. 形式验证在数据路径验证中的应用研究 [D]. 北京: 北方工业大学, 2023.

[12] 李旭元. 基于 UVM 平台的 DMA 控制器验证 [D]. 沈阳: 辽宁大学, 2024. https://doi.org/10.27209/d.cnki.glniu.2024.002066

[13] 黄迪. 面向数据库管理系统的形式化验证 [J]. 信息记录材料, 2025, 26(07): 155-157.

[14] GRIMM T, LETTNIN D, HÜBNER M. A survey on formal verification techniques for safety-critical systems-on-chip [J]. Electronics, 2018, 7(6): 81.

[15] 张牧翔. 适用于 AHB Matrix 的形式验证优化技术研究 [D]. 西安: 西安电子科技大学, 2023.

[16] 维加亚拉哈文. SystemVerilog Assertions 应用指南 [M]. 北京: 清华大学出版社, 2006.