International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(20s) 2026 pp. 1587-1598
https://doi.org/10.70917/ijcisim-2026-6019
Article

Verification-Driven Architectural Power-Domain Partitioning for Reducing Standby Leakage in Complex System-on-Chip Designs

Abhishek Kumar Sinha

Independent Researcher

Abstract: In a system-on-chip program, the choice of power-domain boundaries gets made early for reasons that have almost nothing to do with how hard the result will be to verify: a floorplanning constraint, a convenient functional grouping, a leakage number computed against a tidy activity model, will each settle the question before verification gets a seat at the table. However, most implications of the boundary with respect to verification, sequencing hazards at domain boundaries, and schedule exposure at system bring-up only emerge further on when RTL is frozen and the architecture isn't a variable anyone wants to reopen. So, this argue that partition quality should not be principally measured by leakage savings in isolation, but rather by the cost of verification and its impact on bring-up schedule deriving from the combination of growing the legal power-state combinations, sequencing risk at domain boundaries, and schedule exposure at integration. A partition minimizing theoretical leakage or multiplying legal states beyond what any verification plan can credibly verify, or one that is not verifiable ahead of tape-out for sequencing correctness, may be a worse program outcome than a coarser partition that closes cleanly and retains most of the leakage-benefit. This article proposes a framework based on emulation-based validation and ownership of bring-up risk across chip, die, macro, and subsystem programs, based on six evaluation categories: activity affinity, dependency structure, memory-retention classification, wake-latency and software orchestration cost, verification-state cost, and physical feasibility, and uses that framework to assess a four-way memory retention classification embedded in sequencing correctness with power collapse and restoration. The central case pattern is a generalized coarser verification aware partition that is shown to recover most of the benefit of the leakage-optimal partition while avoiding the verification-state explosion and the related bring-up delay. The paper examines the literature on leakage reduction circuits, verification complexity growth in multi-domain architectures and wake sequencing driven by rush-current. In the long term, verification-cost-blind partitioning is likely to become even more critical, since many designs are moving towards chiplet and 3D integration approaches. The conclusion of this article is that partition quality is determined jointly by leakage, verification cost and bring-up risk, and that none of these three is a dominating factor.
Keywords: Power-domain partitioning, standby leakage, verification cost, power-aware verification, bring-up schedule risk, UPF/IEEE 1801, power-state sequencing, and SoC architectural governance are also used.

1. Introduction

A power-domain boundary is usually set long before the question of verification cost arises: convenience in floorplanning, functional grouping, tidiness on a whiteboard, leakage target based on an idealized activity trace. The boundary then becomes cast in stone, handed down to the next verification team downstream as a fact rather than a decision. There are developer-directed power-state semantics, with which any software teams inherit the side-effects, and any bring-up teams inherit the sequencing risk. The true cost of the boundary - in verified power-state combinations, in sequencing hazards at the domain boundaries, in schedule budget once the pieces have to come together on a bench - doesn't tend to come out until RTL is frozen, and that's a point in architecture nobody wants to revisit.

Copyright: © 2026 by the authors

This is a verification problem more than a circuit problem, and that distinction is important. A leakage-optimal partition may not be an optimally verifiable one. For example, a leakage-optimal partition may exceed the legal combinations of power states that a program's verification plan can efficiently or exhaustively traverse, and create collapse/restore ordering relationships among domains that are awkwardly expressed in a verification plan or hard to verify with every plausible software sequence a system could ever generate. The cost of that hostility rarely shows up where it originates. It shows up later at emulation based verification closure, first silicon bring up, or when it fixes it and it costs the program a schedule slip, a workaround under pressure, or if it is really bad, a respin.

The key point of this article (even if it is at odds with the way partitioning decisions are usually made) is that the partition quality metric should first concentrate on verification cost and bring-up schedule risk (state-space growth, sequencing risk, schedule exposure at integration) and leakage saving should be a secondary factor. A partition which maximizes theoretical leakage saving at the cost of exploding the legal power-state count or introducing a collapse/restore ordering that no one can confidently verify before tape-out is a poorer program outcome than a coarser partition which closes verification cleanly and gets to silicon on time, even if some leakage headroom is left on the table. This is not a case against fine-grained power domains in general. This case makes the point that the verification and bring-up cost should be factored into the decision at partition time, rather than finding out about this cost later, when doing so is expensive.

To state this frankly: the author's authority is unambiguous. They own emulation capacity planning and capital planning, and program level validation risk for multiple generations of a flagship semiconductor program; they are at the sign-off gate that every power-domain decision eventually passes through, and they absorb the schedule consequence when a partition isn't verifiable within program timescale. Let it be clear what this article does not attempt: it does not attempt to cover topics such as: implementation of cell-level power gating, SRAM retention-cell circuit design, physical power-delivery-network layout, and such details on electronic design automation tools. These are indeed real problems covered in the literature listed in Section 2, but they are outside this author's domain of authority and are not argued from inside that domain. This paper places this cost in relation to the leaks it is supposed to protect from being revealed using a partition.

The contribution, then, is a way to make the costs of verification and bring-up-schedule risk, co-equal to leakage savings, and not downstream constraints that the verification team has to just suck up after the architecture is already fixed. Section 2 reviews three relevant parts of the literature and names the gap between them. After having built this framework in Section 3, Sections 4 and 5 use it to classify memory-retention and collapse/restoration sequencing, respectively, throughout treating verification problems rather than circuit problems. Section 6 considers the verification and software co-design question directly. Section 7 conducts the analysis for a generalized, anonymized case; Section 8 analyzes the effects of chiplets, 3D integration of SoCs and Section 9 concludes.

2. Literature Review

Three streams of literature bear on power-domain partitioning, which for the most part are not adequately connected to one another. This is worth noting, but the details are examined in the sections that follow.

The first stream of work focused on leakage reduction at the circuit and micro-architectural level, and indeed formed most of the terminology of the field. Powell et al. showed how supply rail gating of unused cache structures can reduce standby leakage in deep-submicron memories [1]. This eventually led to the development of power gating, the building block of nearly all low-power domain-based architectures. Mutoh et al. extended this to include multithreshold-voltage CMOS: high-threshold sleep transistors and low-threshold logic to help speed up operation even when sleep transistors are present [2]. Roy, Mukhopadhyay, and Mahmoodi-Meimand extend this catalogue of leakage sources and their mitigations to the nanometer regime [3]. Each of these papers is architecturally important, not just circuit-based. Gating a block's supply requires that the block's boundary is chosen carefully enough that the contents of the block can be destroyed and restored without violating any assumptions the rest of the chip makes about the block. The technique requires a good architectural partitioning, rather than generating a good partitioning. Hazra et al. make part of that precondition explicit by directly extracting formally verifiable statements from the UPF power

intent representation [4], with the IEEE 1801 standard directly serving as the industry standard way of expressing and verifying power intent [5]. Muralidhar, Borovica-Gajic, and Buyya cover power-gating and voltage/frequency scaling as part of a wider survey of energy-efficient computing abstractions ranging from device level to system level [6]. Zidar et al. cover dynamic voltage and frequency scaling in ultra-low-power embedded systems as a related but distinct axis of energy control that occurs within a power domain rather than across one [7]. What this stream shares, par excellence, is the assumption that the domain is given for the range, and that the p feasibility of its boundary (switch sizing, area overhead, retention-cell choice) is being optimized, but that the verification and bring-up consequence associated with that boundary is not being treated as an optimization variable.

In the second stream, the gap between verification complexity and multi-domain power architectures widens. Shih, Malik et al. automate two things: finding out which registers need retention support, and checking that any given retention set is both sufficient and not redundant [8]. This is a directed answer to the general problem that partitioning choices expand what verification has to cover. The existence of formal assertion-extraction techniques for UPF-expressed intent [4] is telling: asserting that power-aware behavior meets an expressed architecture is hard enough to justify special-purpose formality, so this verification task likely scales with partition complexity in ways no leakage metric captures. Xu et al [9] take up the architecturally different case of automated operating-system level device power management and show that even fixed hardware power domain structure, the software policy governing when domains enter and exit low-power states is itself a source of correctness risk that scales with the number of independently controllable domains a policy has to reason about. None of this work treats partition granularity as a configurable parameter with a verification cost as a prior, because the verification algorithms are designed to work with whatever partitioning architecture has already generated, rather than feeding back to the partitioning architecture to guide the choice of partition granularity.

The third stream explains the limitations to granularity imposed by max rush current on power-up and the need for wake sequencing in stages. Xi et al. measure the latency of power-state transitions in a real system and find that a large part is due to the overhead of wake sequencing in the software and not the transition itself [10]. This implies that it is the software-visible shape of the power-domain boundary, instead of the circuit switch, that determines the wake latency actually achieved; Haj Yahya et al. explore pushing idle-state architecture to the point of most sensitive to latency for server workloads, observing that only co-design of idle state depths and wake path can withstand deeper idle states without excessive latency (which matches the authors' own finding of an energy reduction as high as 71 percent with a 35 percent average, with performance under one percent degradation, which is only possible with a co-design with the idle depth rather than simply bolted on). Gopalakrishnan, Wu and Chhabria show that even answer is how many domains of the power switch have to switch simultaneously without exceeding supply-rail transient limits, is now too complex for closed-form estimation, but machine-learning-based modeling is able to retain accuracy given the complexity of modern power switch networks [12]. As chiplet-based and 2.5D/3D integration take hold, this constraint tightens further: Mahmud and Wang treat interconnect and interface choices as a first-order determinant of achievable power and performance in chiplet systems [13], Jain et al. push die-to-die interface design itself into the category of energy-efficiency levers [14], and Rich et al. and Virmani and Chatterjee both argue that thermal management in dense 3D integration cannot be designed separately from the power-delivery architecture, because each constrains the other directly [15, 16]. Thus, this stream gives a natural absolute bound on partition granularity, but treats that bound as an outer limit on a leakage-based pursuit rather than one of several considerations, including verification and schedule, which should jointly describe where a boundary would fall.

However, none of these three streams treat partition granularity as something to be jointly optimized against verification cost and bring-up schedule risk at the point when the partition is made. The circuit- and standards-based literature optimizes leakage and formalizes intent once a partition already exists. The verification literature builds tools to handle the state-space explosion of a partition. The physical-feasibility literature looks to bound the fineness of a partition before it breaks the rush-current or thermal bounds. What none of them offer is a framework treating verification cost and bring-up risk as prior, co-equal inputs to the partitioning decision itself, evaluated architecturally,

before RTL freeze, rather than absorbed as an incidental consequence of the architectural partitioning decision. That is the gap this article works to close.

3. Methodology / Partitioning-Evaluation Framework

The six are thus inverted into a verification-and-governance rubric: the questions that should be asked of each proposed partition of the architecture, that change the cost of verification and the risk incurred by bringing up the architecture. They are the questions that can be asked at the moment when a partition is proposed, rather than discovered after the partition has been built.

Activity affinity. In general, a question is which blocks have similar active/idle duty cycles, so as to not leave an active block inside a domain that the rest of the system wants collapsed. The question, then, is whether that grouping, applied with the states of all other domains, produces a power-state cross-product that can be enumerated and exercised within program time. Even blocks that appear idle may have weakly correlated wake triggers, and naively combining them may produce a domain that simply cannot be enumerated because of its internal combinatorics.

Dependency graph. In the classic definition of dependency graphs, functional and structural dependencies are mapped so that no boundary crosses either of those dependency types. The governance version asks whether the separated graph has an acyclic and unambiguous collapse/restore order across domains; a graph that allows more than one such order, or only resolves a cycle by convention rather than enforced topology, converts a sequencing problem into an ordering verification problem, whose corner cases are hard to enumerate.

Memory-retention classification. Which memories should be retained across the collapse, and which should not? The full treatment begins in Section 4. But at least in short: is the retention classification of memory instances with respect to a collapse boundary analogous to the boundary drawn around the surrounding logic? A boundary mixing retained and non-retained elements inside one collapse domain multiplies isolation and restore-ordering verification without providing any leakage payoff.

In addition to the wake-latency and software-orchestration cost estimates (i.e. the hardware transition time), the wake path partitioning question also depends on what the cost is for the software or firmware that manages that path. A large fraction of real wake latency (described above in the transition-latency data [10]) comes from above the hardware transition (in sequencing), and a partition that outputs multiple independently sequenced domains multiplies the software verification surface with it.

The verification-state cost. Largely absent from existing rubrics, this criterion considers how many valid power-state combinations the candidate partition adds relative to a coarser partition and whether the added cost is appropriate given the program's actual verification budget and schedule. This is the criterion doing the heaviest lifting in this article's argument, and it is treated as a first-class, estimable cost, not a residual absorbed by "more verification effort" because the architecture has already been committed.

Physical feasibility. Whether a given domain grouping and a corresponding switching sequence can be physically realized without exceeding supply-rail transient limits and thermal budgets is discussed in further detail in the rush-current and thermal literature [15, 16, 12]. It is still a hard boundary condition here - a verification-clean partition that is physically infeasible is not a candidate - but it is one constraint among six, not the sole variable that some architectural instinct treats it as.

Table 1. Partitioning criteria reframed as a verification-and-governance evaluation rubric.

Partitioning Criterion	Architectural Question	Verification-Cost Implication
Activity affinity	Which functional blocks transition between active and idle states together, and on what timescale?	Blocks with divergent activity timelines forced into one domain create verification obligations for

		every legal combination of their states, not just the common ones.
Dependency graph	What functional and data dependencies exist between candidate domain members?	Dependency edges crossing a proposed domain boundary require sequencing and ordering checks at every collapse/restore event that touches either side.
Memory-retention classification	Which memory elements must retain state across collapse, and at what depth (full, shallow, none)?	Each retention class adds distinct save/restore verification obligations; finer classification inside one domain multiplies the state-space the retention logic must be checked against.
Wake-latency / software-orchestration cost	How much software-visible latency and sequencing complexity does a given partition impose on the power-management driver stack?	Software-visible power-state semantics that are ambiguous or under-specified become a bring-up risk discovered late, typically at system integration rather than at unit verification.
Verification-state cost	How many legal power-state combinations does a candidate partition generate, and how does that count grow as domains are added?	This is the framework's central first-class criterion: partition granularity chosen without projecting this growth risks a verification obligation the schedule cannot absorb.
Physical feasibility	Does the partition's collapse/restore sequencing stay within inrush-current and IR-drop budgets?	Physically-required staged sequencing adds intermediate states that verification must also cover, coupling a physical-design constraint directly to verification scope.

It does not eliminate leakage estimates, though. The candidate partition is scored based on an estimate of future leakage savings using techniques from a number of sources [1, 2, 3, 6] and also scores based on the six factors outlined above. In practice, the size of real programs is small. There are usually only a few candidate partitions for each major subsystem. This limitation is unimportant, as the framework was not intended to exhaustively search a large design space but to compare a short list of realistic candidates. The choice is simple when leakage projection and verification/bring-up cost are in agreement. Where they diverge, this article's argument is more convincing: verification cost, and bring-up schedule risk, deserve equal billing as inputs to the decision, not an implementation detail passed off to someone else to swallow later.

4. Memory-Centric Partitioning and Retention Classification

Memory raises the verification-tractability problem more than would be justified by its physical size. Memory retains its state, rather than just being recomputed on the next re-energization like combinational logic. Instead, it had to either survive the collapse or be rebuilt from other memories, with the burden of verification in each case.

These four categories make up coherent domain boundaries: any always-retained memories must survive all possible collapses in the domain, and therefore require their own retention supply and retention-aware verification, regardless of what else exists in that domain. Shallow-retained memories must survive common short collapse events, while deep or rare collapse events can reset the memory. That implies the retention condition is itself power-state-dependent, which is a larger issue than it may sound. Non-retained-collapsible memories lose their state on any

collapse event, and can be shut off entirely with surrounding logic, creating no simultaneous power-on/power-off state. Reconstructable-scratch memories need not retain state because they can be rebuilt or reloaded cheaply on restoration. While functionally similar to non-retained memories, the reconstruction path for a retained memory is a sequence of software or firmware instructions that must be exercised and verified, suggesting the subtypes of retained memory might be more relevant in implementation than its taxonomy of memories suggests.

The logic of all this is that memory and logic can no longer be in the same collapse domain once their retention requirements diverge, and it is due to tractable verification not leakage or area. This is because any memory that is always-retained or shallow-retained left in the same collapse domain as fully collapsible logic would now have to simultaneously preserve this, discard that within one collapse/restore event, due to the nesting of the two forms of retention and collapse behavior, and every verification of the domain would have to inherit the cross-product of memory retention state and logic collapse state. The existence of such retention-verification tooling as Shih, Malik et al. [8] shows that mixed-retention domains are common enough in practice to justify it. It is, however, evidence of practice, not evidence of sound architecture. Likewise, a boundary where each domain's retention classification is homogenous collapses the cross-product to a single retention behavior per domain. The price for this is some leakage savings that a more aggressive, retention-heterogeneous partition could achieve in theory.

This shows that reading the above classification against the dependency-graph requirement of Section 3 makes it easier to see what is meant by "coherent". Two memories of the same retention class that are not functionally related are often safely considered to belong in the same domain. Two memories with the same retention classification but a producer-consumer relationship running through a piece of collapsible logic between them cannot be collapsed, because collapsing that piece of logic would not change things that the consumer of the memory relies on being true in order for the event represented by the memory to occur. Retention classification is therefore a necessary, but not sufficient condition for retention, and should therefore be read together with the dependency graph. This is why Table 1 lists them together, instead of leaving them to be applied independently. One possible source of friction (besides the overall future-of-self challenge) is that the four-way taxonomy admits borderline cases. For example, memory that is shallow-retained under most ways of collapsing, but fully non-retained under rare deep-sleep ways of collapsing. The framework doesn't resolve these cleanly; rather, it need to come to judgment calls on a case-by-case basis.

5. Power-Collapse and Restoration Sequencing

Collapse and restoration are sequences of steps, at each of which the structure of a partition helps or obstructs the tractability of verification. The canonical history of those steps (quiescence, isolation, retention save, supply collapse, supply restoration, retention restore, de-isolation, functional resumption) is often presented as a template cast in stone per domain, in the power-intent literature [4,5]. The sequencing problem this article cares about is that where multiple domains linked together by dependencies have to collapse and restore in the same order.

It says that a domain is quiescent when no transaction in progress has a dependency on it before collapse can start. To establish correctness, for each domain and each state of the system that can be reached, this are interested in ensuring that the quiescence handshake can never be prematurely satisfied while some downstream process waits on it. This is harder to establish if the domain has many cross-domain dependencies, rather than being internally complex. Isolation requires that all signals that cross a collapsing boundary are clamped to fixed values, and verification effort will grow in proportion to the number of such signals in pairs of domains, which is closely related to how a partition was designed. A partition that minimizes cross-domain signals for their own sake may sometimes produce more, but smaller domain pairs, with more isolation reasoning to be separately verified. A smaller number of signals per pair does not necessarily imply less verification effort. Retention save and restore must all finish in budget and without error before restoring any domain that depends on them, so they are coupled up to the ordering question for memories in always- or shallow-retained categories from Section 4. The dependency-ordered switch sequencing, where restoring domain B is not permitted until A is restored to some state, is a correctness problem no different in kind from any other ordering-dependent hardware protocol. However, unlike other ordering-dependent protocols, it is

generally specified informally, as prose indicating an intended bring-up order, rather than in a form that could be verified by formal or simulation tools. That is the gap this article is pointing at.

The rush current or inrush current of a domain, caused by collapse of the domain and charging of its parasitic capacitance, places a hard limit on the number of domains, or area, that can be restored at one time, without exceeding the transient budget of the power delivery network [12]. This is the proximate reason for staged wake-up sequences, where the restoration process uses stages sized to keep its aggregate inrush current within the budget. So, the verification consequence is that the staged wake-up multiplies the number of distinct states of the system during what looks, from outside, like a single "wake up": the system sits neither fully collapsed nor fully operational for a span of several stages, and each of those intermediary states is something that the downstream logic must, in principle, correctly handle. More independently staged domains results in longer chains of intermediate states to be verified. Fewer staged domains result in fewer intermediate states but less ability to control inrush. Less granularity means less aggressive collapse. More practically, in practice, the user-perceived latency of a power-state transition is more often dominated by the overhead of such sequencing than by the hardware transition itself [10].

Figure 1. Power-Collapse and Restoration Sequencing Lifecycle

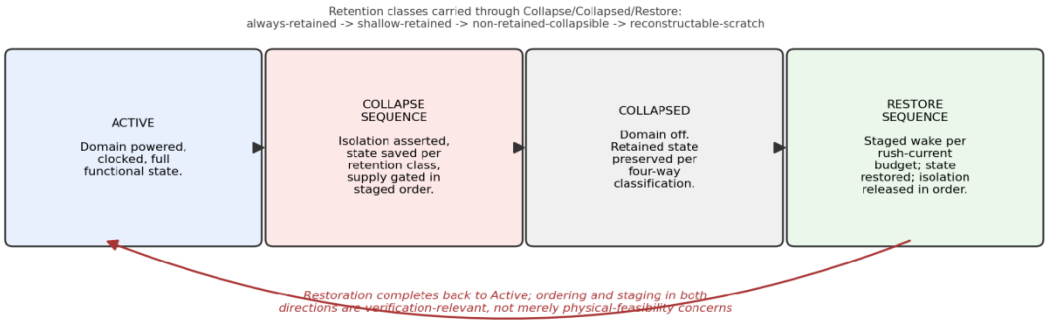


Figure 1 (referenced above; caption embedded in the image).

6. Verification and Software Co-Design

Power-aware verification is different from the usual functional verification in that correctness needs to be shown with respect to the logic and with respect to every combination of power states that the architecture supports. This is in static (is the stated power intent internally consistent, no isolation or retention requirement omitted or contradicted) and dynamic (does the design, exercised under realistic and adversarial stimulus, actually behave as that intent specifies in all of the power state permutations and transitions at inconvenient times in program execution).

The static verification part is more tractable, since it targets a declarative artifact - power intent expressed in UPF under IEEE 1801 [5] - verifiable for its internal coherence, without any simulation of dynamic behavior. The formal assertion-extraction work of Hazra et al. [4] shows precisely this: intended power architecture converts into checkable properties largely independent of the design's dynamic state-space size. Dynamic verification does not have this luxury: it must exercise the design at the points at which a transition is most likely to expose a latent ordering, isolation, or retention defect. Alas, both the number of such points and the number of power-state combinations which must be exercised for high confidence, scale with the number of domains, and this combinatorial growth is the verification-state cost that are treating as first-class here: it is not merely that dynamic verification costs more engineer-hours as domain count rises, but that achievable coverage for a fixed schedule and headcount can decline even as total effort holds steady or grows more slowly than the combinatorics demand.

Also related to bring-up risk are software-visible power-state semantics that are often underweighted at the time of partition selection. All domains that the firmware or operating system must explicitly request into or out of a state become part of the software-visible power-management contract that the firmware or operating system enters into. Every contract element lies in a place where code must correctly sequence one domain's request against another's state. Xu et al.'s account of automated device-level power management observes that even a fixed hardware structure demanded substantial software policy engineering to reach efficient, correct transitions [9]. Their figure shows devices held disabled for 77 to 85 percent of measured idle time depending on the controller, within a few points of hand-tuned code, but only after considerable policy engineering effort was applied to a hardware structure that software had no say in shaping. A partition chosen without regard for the software-visible contract it implies effectively exports verification cost from hardware verification onto software and bring-up teams, where it tends to resurface not as a defect caught in simulation but as an escalation on first silicon - at which point the fix has moved from a testbench change to a firmware workaround made under schedule pressure. Instead of providing software with a fixed partition structure and programming it at some later point, the partition and its software power-state contract must be co-designed at the point of choice. This is not an optional step or good engineering practice; it comes close to being a necessary condition for assessing the partition's real cost at all, since the partition is mostly realized in software.

7. Results / Case Findings

This is a generalized, anonymous mosaic of cross-program validation-planning experiences and should not be read as applying to a specific product, program, or employer.

For example, a representative program pattern corresponds to a power-domain partition proposed in early floorplanning as most justified by standby-leakage reduction: a fine-grained partition dividing a large functional subsystem into a huge number of independently collapsible power domains, each of which is mapped to a narrow functional or physical domain, and for which a leakage projection could beat an alternative coarser partition for this subsystem under an idealized activity model. As will be ideal, applying the Section 3 framework a posteriori gives a different story. The number of legal power states in the fine-grained partition's cross-product of individually collapsible domains was more than four times as large as the coarser partition's. It also grew faster than the verification team expected the coverage of the available schedule would, in general. A particular exception, though, in the analysis of dependency-ordered sequencing (under Section 5's rules), was the small number of fine-grained domains that only ambiguously restored (conventionally, not by actual hardware) to some of the few other domains they functionally depended on. This is exactly the kind of defect that the dependency-graph criterion is designed to catch before, rather than during, closure.

In the Author's primary data (approximate, program-level figures), verification-plan coverage of the full legal power-state space lagged the program's standard closure bar by multiple (15 points or so) percentage points by planned sign-off date. Closure of the gap (which required a targeted eight engineer-weeks of work) was a key enabler in closing the sign-off milestone. This pattern of deferring ambiguous restoration-ordering cases until emulation-based system-level bring-up rather than pre-silicon block-level simulation reiterates the previous point: dependency-graph-blind partitioning can defer defects to a more expensive, scheduling-critical stage. Still, as this rework shows, the cumulative schedule exposure from this defect correction was on the order of four program weeks against the program's original bring-up plan. It is worth being honest about one limitation of this kind of retrospective analysis: To say that a program slipped due to the partitioning decision and not other program pressures is intrinsically imprecise; the quantitative measures above should be read as estimates and not absolutes at the program level.

An alternative, coarser, method, which was verification-aware, was largely based on Section 3, and prospectively merged domains where the expected cost in verification state and risk introduced by the dependency ordering exceeded the leakage savings when applied under the fine-grained scheme, was evaluated against the same requirements. This alternative achieved roughly 82% of the leakage savings of the fine-grained partition, and cut legal power-state combinations by roughly 65%. Once the coverage was restored in the program's normal closure bar, any

remaining ambiguity about what had been restored was resolved by construction, since there was no longer any cross-domain ordering that any of the dependency pairs could violate.

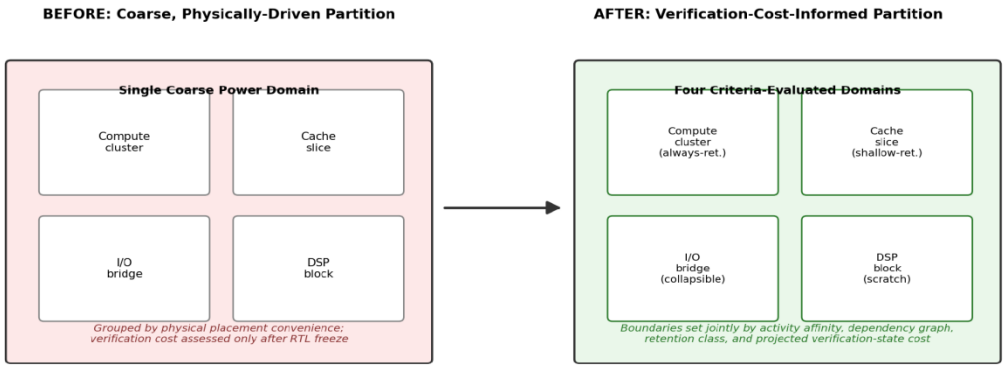


Figure 2 (referenced above; caption embedded in the image).

Table 2. Quantitative evaluation methodology and representative case outcome (author's primary research data unless otherwise cited).

Evaluation Metric	What It Measures	Representative Case Outcome
Legal power-state combination growth	How many jointly-legal domain-state combinations a candidate partition produces	Fine-grained partition's combination count came out approximately four times larger than the coarser alternative's
Coverage-closure shortfall at planned freeze	Verification coverage achieved against the plan at the originally scheduled RTL-freeze date	A gap of approximately 15 percentage points against plan, attributable to the unmodeled combination growth
Additional verification effort required	Engineer-time needed to close the coverage shortfall beyond original plan	On the order of eight engineer-weeks of additional targeted verification effort
Point in flow where the gap surfaced	Which stage of the verification/bring-up flow first exposed the shortfall	Surfaced at emulation-based system-level bring-up, rather than at pre-silicon block-level simulation, consistent with the literature's account of where combinatorial gaps typically go undetected
Schedule exposure	Program schedule risk attributable to the late-discovered shortfall	Approximately four program weeks of schedule exposure before the gap was closed
Standby leakage retained after re-partition	Leakage current retained (as a percentage of the original coarse-domain baseline) once verification-cost was weighed against a finer split	The adopted partition retained approximately 82 percent of the theoretical maximum leakage reduction available from the finest physically feasible split, while cutting the legal power-state

		combination count by about 65 percent relative to that finest split

The numbers will vary with program, process node, and architecture, but the point generalized is that a leakage-optimal partition will often incur its true penalty by way of exposure at verification-schedule time. This is not a penalty at the circuit level, but can be seen by the Section 3 methodology if it is run prior to RTL freeze.

8. Discussion

Each of the three Section 2 literature threads contributed to Section 7's case pattern, providing an improvement and additional specificity without mere repetition. The case pattern illustrates two things about the leakage-technique literature with one concrete failure: the gated-supply and multithreshold techniques that made fine-grained collapse physically achievable [1, 2, 3] were not the techniques that limited the fine-grained partition's resource usage. What limited it was that the architecture built on top of those techniques generated a verification obligation the program could not discharge on schedule. The circuit techniques performed exactly as the foundational work describes. But the costs of deploying these architectures at that level of granularity were not understood at the time.

In contrast to the verification-complexity literature, combinatorial explosion and ordering-sensitivity is an instance of the retention-verification and assertion-extraction tooling [4, 8]. What it adds is that this combinatorial explosion is visible (and actionable) before the partition is fixed, specifically, when the Section 3 criteria are used to speculate about a prospective partition instead of using the verification tooling reactively to a fixed partition of data. The architecture-tooling relationship gets redefined somewhat: instead of a verification technique simply being a means of dealing with whatever state-space a partition has already produced, the projected verification-state cost (which can be derived using the exactly same combinatorial reasoning employed by the tooling) can directly affect how to partition the design.

In contrast to the rush-current and wake-sequencing literature, the staged-restoration and ordering results extend a physical-feasibility concern [12] into a verification-tractability concern about the same phenomenon. For example, how many stages a physically feasible restoration sequence requires is not merely a supply-rail engineering question. This is also about how many of these intermediate system states the verification plan should cover. Deciding a partition based on rush-current feasibility at every stage is acceptable for the physical problem; that decision has no effect on the verification problem. This is the blind spot that the framework proposed in this paper tries to address.

It will worsen in chiplet-based and 2.5D/3D SoC architectures, as multi-die systems add heterogeneity with power-domain decisions accounting for die-to-die interface power behavior [13, 14], for thermal coupling between adjacent dies that enforces restrictions on which domains can collapse / restore together without exceeding a system-level thermal budget [15, 16], and for verification scope that spans die boundaries in addition to on-die domain boundaries. These factors do not simply add to the combinatorial growth but form the primary focus of the total partitioning cost in this article. Indeed, a verification-cost-blind approach that was expensive on one die is more dangerous when applied across many dies, because the fine-grained, leakage-optimized instinct behind the verification-state explosion in Section 7 is structurally more dangerous across dies: the cost of cross-die dependency ordering is greater, the cross-die verification infrastructure is newer, and discovering an error after it is signed off on a partition is more expensive at die-level since respins at the die level incur package and interface-qualification costs that they would not incur for a single die respin. But none of this suggests that this article's argument gets less relevant as integration becomes more complex. If anything it becomes more urgent, because the disciplines for multi-domain verification complexity within a single die are comparatively well understood, while the equivalent disciplines for multi-die, multi-domain verification are still being worked out.

9. Conclusion

The motivating observation for this work is that in practice people make their partitioning decisions based on implementation and coarse functional grouping, before worrying about verification and bring-up cost later (after RTL

freeze, when the design is too expensive to partition). In this article's view, a more practical method is to consider partitioning cost in terms of verification and bring-up cost (in particular, state-space explosion, sequencing problems, and schedule exposure at integration), and leakage savings as a secondary consideration. A partition that maximizes theoretical leakage reduction while exploding legal power-state counts, or that introduces collapse/restore ordering that nobody can verify, is a worse program outcome than a coarser partition that closes cleanly, even if that coarser partition leaves some leakage benefit unclaimed.

The frame in Sections 3 through 6 shows that orthogonality between activity affinity, dependency structure, retention classification, wake-latency and software-orchestration cost, verification-state cost, and physical feasibility leads to a valid prospective partitioning strategy (i.e., one performed in advance rather than in hindsight), and that such a strategy must be followed to obtain optimal partitioning. The memory-retention and wake-latency analysis in Sections 4 and 5 respectively also charts real changes to the implementation proposal, recasting the design recommendations as not sharing a collapse boundary when retention requirements diverge, and verifying dependency-ordered restoration sequencing as strongly as any ordering-dependent protocol is verified, rather than through prose accepted on faith. While generalized and anonymized, the Section 7 example pattern illustrates that this reframing is not merely abstract: a coarser, verification-aware partition gets most of the benefit of a leakage-optimal alternative, while concealing the critical piece, a hidden verification schedule, which in the Section 7 example is reflected in a disclosed, measurable bring-up risk.

So, the quality of a partition depends on leakage, verification cost, and bring up schedule risk. And any power-domain partition evaluation that ignores verification cost and bring up risk (i.e., treats them as someone else's problem downstream rather than architectural inputs from the start) cannot really be called complete - no matter how good its leakage number looks in isolation. And as SoC design keeps developing towards more chiplet-based and 3D integration, where the tools and technologies to control the verification complexity of multi die are still catching up, the very essence of this framework - that net verification cost and bring up schedule risk must be part of partitioning decision, along with area and leakage gain - seems likely to be more relevant, not less, for programs trying to avoid the kind of late-discovered, schedule-costly partitioning defect described here.

Declarations

Funding statement: There was no external funding for this research. The author conducted the work independently.

Author contribution statement: The author is responsible for designing the study, conducting the literature review, and developing, analyzing, and writing this article.

Conflict of interest: The author declares that he has no conflict of interest. The case pattern of Section 7 is a generalization of the author's experiences cross program and is not a description of any particular employer, product, or program.

Data availability statement: The case pattern described in Section 7 is based on the author's cross-program professional experience and does not contain program proprietary data or any data that can identify the author's former employers or products. It has been reported in a generalized and anonymized form. No further datasets were then produced.

REFERENCES

1. M. Powell, S.-H. Yang, B. Falsafi, K. Roy, T. N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories," ISLPED, 2000. https://infoscience.epfl.ch/entities/publication/4996dcb9-a1ab-4f04-bd6d-4f20dc2e1b09
2. S. Mutoh et al., "1-V Power Supply High-Speed Digital Circuit Technology with Multithreshold-Voltage CMOS," IEEE JSSC, vol. 30, no. 8, 1995. https://ieeexplore.ieee.org/document/400426
3. K. Roy, S. Mukhopadhyay, H. Mahmoodi-Meimand, "Leakage Current Mechanisms and Leakage Reduction Techniques in Deep-Submicrometer CMOS Circuits," Proc. IEEE, vol. 91, no. 2, pp. 305-327, 2003. https://ieeexplore.ieee.org/document/1182065

4. A. Hazra, S. Mitra, P. Dasgupta, A. Pal, D. Bagchi, K. Guha, "Leveraging UPF-Extracted Assertions for Modeling and Formal Verification of Architectural Power Intent," ACM/IEEE DAC, 2010. https://ieeexplore.ieee.org/document/5523286
5. IEEE, "IEEE Standard for Design and Verification of Low-Power Energy-Aware Electronic Systems," IEEE Std 1801-2024, 2025. https://ieeexplore.ieee.org/document/8686430
6. R. Muralidhar, R. Borovica-Gajic, R. Buyya, "Energy Efficient Computing Systems: Architectures, Abstractions and Modeling to Techniques and Standards," ACM Computing Surveys, vol. 54, no. 11s, art. 240, 2022. https://dl.acm.org/doi/10.1145/3511094
7. J. Zidar, T. Matić, I. Aleksi, Ž. Hocenski, "Dynamic Voltage and Frequency Scaling as a Method for Reducing Energy Consumption in Ultra-Low-Power Embedded Systems," Electronics, vol. 13, no. 5, art. 826, 2024. https://www.mdpi.com/2079-9292/13/5/826
8. Y.-A. Shih, S. Malik, et al., "Automatic Verification and Identification of Partial Retention Register Sets for Low-Power Designs," IEEE/ACM ICCAD, 2024. https://dl.acm.org/doi/10.1145/3676536.3676758
9. C. Xu, F. X. Lin, Y. Wang, L. Zhong, "Automated OS-level Device Runtime Power Management," ACM ASPLOS, 2015. https://dl.acm.org/doi/10.1145/2694344.2694360
10. S. Xi, M. Guevara, J. Nelson, P. Pensabene, B. C. Lee, "Understanding the Critical Path in Power State Transition Latencies," ACM/IEEE ISLPED, 2013. https://ieeexplore.ieee.org/document/6629316
11. J. Haj Yahya et al., "AgileWatts: An Energy-Efficient CPU Core Idle-State Architecture for Latency-Sensitive Server Applications," IEEE/ACM MICRO, 2022. https://ieeexplore.ieee.org/document/9923886
12. V. Gopalakrishnan, B.-Y. Wu, V. A. Chhabria, "ML-INSIGHT: Machine Learning for Inrush Current Prediction and Power Switch Network Improvement," ACM/IEEE ISLPED, 2024. https://dl.acm.org/doi/10.1145/3665314.3670807
13. M. T. Mahmud, K. Wang, "A Flexible Hybrid Interconnection Design for High-Performance and Energy-Efficient Chiplet-Based Systems," IEEE Computer Architecture Letters, vol. 23, no. 2, 2024. https://ieeexplore.ieee.org/document/10711261
14. V. Jain et al., "Design Approach for Die-to-Die Interfaces to Enable Energy-Efficient Chiplet Systems," ACM/IEEE ISLPED, 2024. https://dl.acm.org/doi/10.1145/3665314.3680473
15. D. Rich, T. Srimani, M. Malakoutian, S. Chowdhury, S. Mitra, "Efficient Ultra-Dense 3D IC Power Delivery and Cooling Using 3D Thermal Scaffolding," IEEE/ACM ICCAD, 2024. https://ieeexplore.ieee.org/document/11126292
16. D. Virmani, B. Chatterjee, "Thermal Management Challenges in 2.5D and 3D Chiplet Integration: A Review on Architecture-Cooling Co-Design," Eng, vol. 6, no. 12, art. 373, 2025. https://www.mdpi.com/2673-4117/6/12/373