World Journal of Advanced Engineering Technology and Sciences
eISSN: 2582-8266
Cross Ref DOI: 10.30574/wjaets
Journal homepage: https://wjaets.com/

(REVIEW ARTICLE)

Check for updates

PCIe L0p low-power state verification: Pre-silicon approaches and challenges

Deepak Kumar Lnu *

Principal Engineer, USA.

World Journal of Advanced Engineering Technology and Sciences, 2025, 15(01), 388-396

Publication history: Received on 25 February 2025; revised on 03 April 2025; accepted on 05 April 2025

Article DOI: https://doi.org/10.30574/wjaets.2025.15.1.0208

Abstract

The PCIe 6.0 specification introduces the L0p low-power substrate, enabling dynamic link width scaling to reduce power consumption without interrupting data flow. This feature presents unique verification challenges due to its complex handshake mechanism, precise timing requirements, and backward compatibility needs. Pre-silicon verification using formal methods and simulation is essential to validate L0p logic before silicon fabrication. Formal verification confirms the correctness of state transitions, entry/exit conditions, and deadlock freedom, while simulation examines lane activation/deactivation sequences and data integrity during width changes. This article details the verification methodology for L0p, including formal proofs of the handshake process and extensive simulation of lane scaling events. Case studies highlight subtle design bugs uncovered and resolved in pre-silicon, including simultaneous width-change request deadlocks and lane reactivation sequence errors, avoiding costly post-silicon debugging. Comprehensive pre-silicon validation ensures PCIe 6.0 devices can safely leverage this power-saving mechanism while maintaining performance and interoperability.

Keywords: Power management; Link width scaling; Formal verification; Pre-silicon validation; Protocol compliance

1. Introduction

Peripheral Component Interconnect Express (PCIe) employs a Link Training and Status State Machine (LTSSM) to manage link initialization, normal operation, and power management transitions. The PCIe 6.0 specification introduced a significant new power-management substate called L0p (L0 power state) to improve active-state power efficiency [1]. L0p allows a PCIe link to dynamically reduce its lane count (link width) during operation to save power while keeping at least one lane active so that data traffic is never interrupted. This is a departure from previous generations where any link width change required the link to enter a recovery/training process (causing a traffic stall of several microseconds) or to use L0s, a lighter sleep state that provided only modest power savings. In essence, L0p enables proportionate power consumption to bandwidth usage – the link can scale down to fewer lanes when full bandwidth is not needed and scale back up when demand increases, all without halting data flow.

The L0p state is negotiated and used only if both link partners support it, and it is restricted to Flit mode operation (a PCIe 6.0 feature using fixed-size Flow Control Units at the data link layer) [1]. During initial link training, devices exchange capabilities; if L0p is supported on both sides, it is enabled at the end of training (in the Configuration.Complete phase via a TS2 ordered-set exchange). Thereafter, either device can initiate a transition into L0p by requesting a lower link width when appropriate. The specification mandates symmetric width changes – the link must downscale or upscale symmetrically (e.g., x16 to x8, x4, x2, or x1) and does not allow asymmetric configurations (like x4 in one direction and x8 in the other). This design choice avoids added complexity and potential interoperability issues, focusing L0p on the standard PCIe lane widths (×1, ×2, ×4, ×8, ×16). It also ensures that at least one lane in each direction remains in L0 (active) state at any time, carrying forward progress for data packets or keep-alive ordered sets.

* Corresponding author: Deepak Kumar Lnu

Verifying the L0p mechanism is critical because it involves intricate handshake protocols and new timing constraints in PCIe 6.0. A design flaw in L0p logic could manifest as a link that fails to properly change width or, worse, a link deadlock or instability when power management triggers occur. Unlike always-active L0 or fully-idle L1 states, L0p is a hybrid mode intermixing active and quiescent lanes, making validating less straightforward. Moreover, the L0p feature must coexist with backward compatibility: a PCIe 6.0 device supporting L0p must interoperate gracefully with older generation devices (which will never invoke L0p). Thus, the L0p control logic must be robust under mixed-generation scenarios (e.g., ensuring L0p requests are not sent if the partner didn't enable L0p or handling legacy 8b/10b encoding modes with different timing). Given these challenges, pre-silicon verification of the L0p state is indispensable. This paper applies formal verification and simulation to thoroughly validate the L0p low-power state logic in a PCIe 6.0 dut design before tape-out. The scope of digital pre-silicon verification (logical correctness of state-machine and protocol handling) and aspects like analog signal integrity during L0p (e.g., actual power consumption or electrical transients on lane disabling) are beyond the scope. By ensuring the design's L0p behavior is correct in pre-silicon, one can avoid errata in the field and ensure that the promised power savings of L0p are realized without sacrificing link reliability or performance [3, 6].

2. Technical Background

SW Control bits @ Tx/Rx		Tx side behavior			Rx side behavior		
Hardware Autonomous Width Disable	L0p Enable	Issue Nonpriority L0p request for width reduction?	Issue Priority L0p request for width reduction?	Issue L0p request for width increase?	Response to Nonpriority L0p request for width reduction?	Response to Priority L0p request for width reduction?	Response to L0p request for width increase?
1	x	Not permitted	Not permitted	Must issue to achieve fully configured width, if at lower width	Recommend Nak. But 'No-response' is permitted as well.	Recommend Nak. But 'No-response' is permitted as well.	Ack for full width increase. Otherwise, recommend Nak but 'No-response' is permitted as well.
0	0	Not permitted	Not permitted	Must issue to achieve fully configured width, if at lower width	Nak	Nak	Ack for full width increase. Otherwise, recommend Nak but 'No-response' is permitted as well.
0	1	Permitted	Permitted	Permitted	Ack/Nak	Ack (with exception noted in § Section 4.2.6.7)	Ack

Figure 1 Summary of L0p Transmitter/Receiver Behavior [1]

PCIe Active State Power Management and L0p: The fully active state in PCIe is L0, where all lanes transmit data. Before PCIe 6.0, active-state power management relied on L0s (L0 Standby) and L1 substates to save energy when traffic was idle. L0s is an electrical idle state entered on idle periods, but exiting L0s back to L0 uses a limited Fast Training Sequence and yields only minor power savings with very low latency. L1 is a deeper standby state that powers down much of the link, achieving significant power reduction at the cost of higher exit latency (multiple microseconds) due to requiring a full link retraining handshake upon wakeup. Additionally, some implementations experimented with dynamic link width (DLW) change in earlier generations, where a link could retrain to a lower lane count (or lower speed) when throughput needs to drop. However, the conventional DLW approach forced the entire link into a Recovery/Configuration state for re-training, stalling traffic for tens of microseconds and negating most benefits. A new approach was needed to allow on-the-fly bandwidth scaling without link resets, which motivates L0p.

L0p is a substate of L0 (the active state), enabled only in Flit Mode for PCIe 6.0 and above links. The link always initializes at the maximal negotiated width during training, then may enter L0p dynamically based on runtime conditions. Entry into L0p involves a handshake between the two link partners. According to the PCIe 6.0 specification, when one side decides to reduce link width (for instance, an endpoint experiencing low traffic wants to save power), it issues an L0p Request for a target width (which must be narrower than current) via a special Link Management DLLP (Data Link Layer Packet) and/or a coded indicator in the periodically ordered sets. Upon receiving this request, the partner must respond with an ACK or NAK within a specified timeout (1 μs in high-speed modes or up to 4 μs if operating at legacy Gen1/Gen2

speeds with 8b/10b encoding). If the request is acknowledged, both sides will downscale the link; if it is negatively acknowledged or times out, the width remains unchanged (and the initiator may retry later). To prevent deadlock when both sides coincidentally request a width change at the same time, the spec defines an arbitration: typically, the request for a lower width (or the one designated with higher priority, depending on an L0p.Priority flag) will take precedence, and the other request is nacked. This ensures a deterministic outcome even for simultaneous requests.

3. L0p Lane Transition Sequence

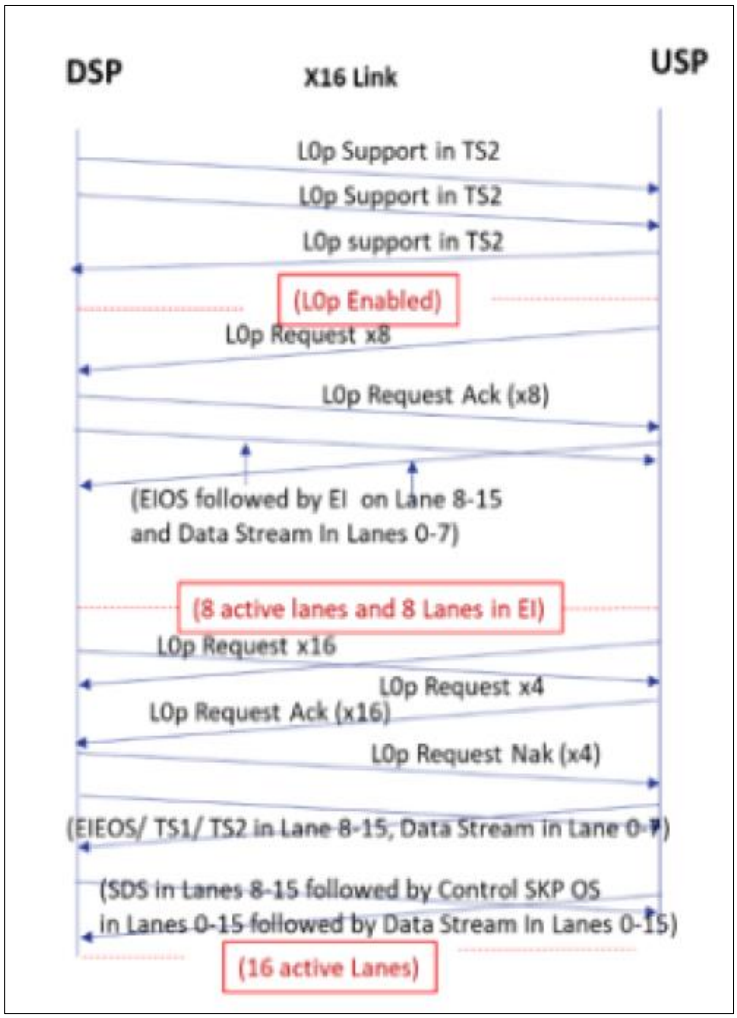

Figure 2 Illustration of L0p dynamic link width scaling on a PCIe 6.0 ×16 link [4, 5]

Once an L0p width change is agreed (e.g., from ×16 down to ×8), the link enters a transition phase where half of the lanes will be turned off. This process is orchestrated carefully to avoid data corruption. The spec defines that lane deactivation occurs at an SKP Ordered Set boundary – PCIe links periodically send SKP symbols for clock correction, and these form natural checkpoints where lanes can be gracefully taken offline or brought online [1]. For a width reduction, a Control SKP Ordered Set is sent on all currently active lanes to signal the upcoming change, and then the lanes that will be turned off send an Electrical Idle Ordered Set (EIOS) followed by entering an electrical idle (quiescent) state. Meanwhile, the remaining lanes carry the data stream (Flits and SKP symbols) at a reduced width. The link has effectively dropped to the new lower width, and normal data transmission continues on the active lanes.

For a width increase (returning to a higher lane count), the sequence is somewhat reversed and more involved since previously idle lanes need to retrain and synchronize. The lanes to be activated exit electrical idle in a coordinated fashion: a special Electrical Idle Exit Ordered Set (EIEOS) is issued (for data rates that require it), and training sequences (TS1/TS2) are exchanged on those lanes. In contrast, the other lanes maintain the ongoing data stream. The new lanes train up to link-ready status, including receivers locking and equalization if needed, similar to a Recovery substate but done in parallel with live traffic on the other lanes. After a successful TS1/TS2 exchange on the new lanes, the link

partners send a Start-of-Data Stream (SDS) indication on those lanes (at ≥Gen3 speeds) to mark the point at which those lanes will carry real flit data. Finally, a Control SKP OS is sent to all lanes indicating completion, and the data stream then spans all lanes at a higher width. Throughout this upsizing sequence, forward progress was maintained on the originally active lanes; once the formerly idle lanes join, full bandwidth operation resumes [4, 8]. The entire entry or exit of L0p is thus transparent to software and higher protocol layers – there is no packet loss, only a momentary reduction in aggregate throughput during the transition. Figure 1 conceptually illustrates an L0p width downscale and upscale event for a ×16 link going to ×8 and back to ×16

In the downscale (left), lanes 8–15 send EIOS and go idle while lanes 0–7 continue carrying traffic (resulting width ×8). In the upscale (right), lanes 8–15 retrain (TS1/TS2 exchange) and then resume carrying data, restoring width ×16. At least one lane in each direction remained active throughout both sequences, so no data transfer interruptions occurred [1].

Notably, L0p is incompatible with L0s—in fact, the PCIe 6.0 spec disallows using L0s in Flit mode because L0p provides a superior solution for active power management [5, 7]. Any lanes turned off during L0p behave essentially as if they were in an L1-like idle (the spec expects similar electrical idle residency as L1 for those lanes). From a system perspective, L0p is intended to be completely transparent aside from the power reduction. Hardware designers anticipate that L0p could save several picojoules per transmitted bit on an ×16 link by powering down transceivers on unused lanes, a significant gain at 64 GT/s data rates [3, 7]. As PCIe continues to evolve (PCIe 7.0 and beyond), the L0p mechanism may be further refined (for instance, future specifications might allow asymmetric lane configurations or additional widths) [5]. Still, scaling link resources to match bandwidth demands is expected to remain integral to high-speed I/O power management.

4. Pre-Silicon Verification Methodology

Verification of the L0p state logic was approached with two complementary pre-silicon methodologies: formal verification and directed simulation. Both were employed to ensure the design correctly implements the L0p protocol under all specified scenarios. Importantly, the verification environment models the PCIe link at the transaction/data link/physical layer level to exercise the L0p behavior purely in simulation or formal analysis without needing physical hardware [8]. This section outlines how each methodology was used for L0p and how they complement each other.

5. Formal Verification

Formal methods were used exhaustively to prove the critical properties of the L0p implementation. A formal model of the LTSSM and associated link control logic was constructed (Verification IP), interfacing with a simplified environment model representing a link partner. Using a PCIe Verification IP (VIP) model for formal, encoded the rules of the L0p handshake as protocol checks. Key properties checked include: (1) Correct Entry Conditions: The design must only enter the L0p substate when specified (i.e., when both devices have advertised L0p support and a valid L0p Request/response handshake has occurred). The spurious or out-of-order triggers would not bring the LTSSM into L0p, is proved. For example, the check was written that if the LTSSM transitions to L0p, then an L0p Request must have been sent by the device or received from the partner and acknowledged appropriately within the timeout. Conversely, if a valid request is issued and acknowledged, the LTSSM should eventually enter L0p. (2) State Transition and Deadlock Freedom: Verifying that once in L0p, the link can always cleanly exit back to L0 (or other states like Recovery if external events trigger a retraining) without deadlocking. One error test injected an assumption that one side issues an L0p Request and then checked that the link width eventually changes (entering the L0p state) or the request is aborted by a NAK/timeout – in all cases, the protocol progresses. This ensures the design cannot get stuck waiting indefinitely (e.g. if the partner misbehaves or an internal flag is missed). Additionally, a mutual exclusion check was done for simultaneous requests, proving that if both sides initiate L0p simultaneously, the design logic resolves it such that only one side's request succeeds (the other is nacked), per the spec arbitration rule. This was achieved by modeling two independent request initiations and ensuring a safety property that they don't both reach the "accepted" state. (3) Lane Transition Sequencing: The Verification IP was also used to verify the sequence of control events on each lane during L0p transitions. Formulating temporal checks to confirm that whenever the design initiates a width downscale, it will send the proper Control SKP and EIOS on the affected lanes before turning them off and that those lanes enter an idle state. Similarly, for width upscale, check that idle lanes receive TS1/TS2 training and an SDS indicator before the design uses them for data. These properties required reasoning over a cycle-accurate timeline of events aligned with SKP intervals. Leveraging bounded model checking with carefully chosen bounds (on a few microseconds of link time) to cover the handshake and ordered-set sequences [7]. The formal proofs increased confidence that the dut's finite-state

machines and timer counters exactly match the specification's L0p protocol. Any violation of these properties would produce a counterexample trace, which could then be analyzed to pinpoint logic errors.

Notably, one area of formal verification targeted was the timing constraints. For instance, the spec requires an L0p responder to reply within 1 μs (128b/130b encoding) or 4 μs (8b/10b mode). Modeling the passage of time cycles formally proved that if the requester does not see a response within the required window, the requester's state machine cancels the pending request as expected. This involved checking that a timeout counter reaches its threshold and triggers an abort signal without an ACK. Another formal check ensured that after an L0p exit (upsize), the design does not re-enter L0p too quickly, respecting any spec-defined minimum residency or hysteresis (the Q&A indicates a lane should stay idle at least as long as an L1 minimum, preventing rapid flapping) [2]. By exhaustively exploring all corner cases (different initial widths, simultaneous requests, edge-case timing of messages), formal verification provided a mathematical guarantee for many aspects of L0p logic correctness.

6. Simulation-Based Verification

While formal proof covers logical correctness, simulation validates the design's behavior over time with realistic scenarios and checks aspects more easily observed in waveform-level detail (such as data payload integrity and interoperability with a more detailed partner model) [6]. Developing a directed and random simulation test suite using a PCIe 6.0 simulation environment (with a partner model that supports L0p). Directed test cases targeted specific scenarios: for example, a test where the root complex triggers L0p entry after a period of low utilization, expecting the endpoint to ACK and the link to transition to half width; another test where the endpoint requests L0p but the root NAKs the request (perhaps simulating that the root is not ready to reduce bandwidth), ensuring the endpoint returns to normal L0 operation promptly. Crafting a scenario of simultaneous L0p requests: both testbench models issue an L0p Request nearly simultaneously (within the same microsecond). In the simulation, verifying the outcome – one request succeeded, and the other was rejected – and that the link still ended up in a stable width (per spec, the higher-priority or higher-width request wins). This test was cross-checked with the formal results for consistency.

Crucially, lane-level monitoring was implemented in the testbench to validate the ordered-set sequences during transitions. Attaching checkers that listen for the expected Control SKP, EIOS, EIEOS, TS1, TS2, and SDS symbols on each lane at the right moments. For instance, in a downscale test from ×8 to ×4, the checker ensures that lanes 4 to 7 output EIOS and then go quiet, while lanes 0 to 3 carry flits. In an upscale test from ×1 back to ×8, the checker looks for TS1/TS2 on lanes 2 to 7 while flits continue on lane 0, then an SDS on lanes 2-7, etc., before verifying that flits are seen on all 8 lanes thereafter. Any deviation (such as a lane starting to send data without an SDS or an EIOS not being sent when it should) is flagged as a failure. These monitors essentially serve as a scoreboard for protocol compliance at the physical layer.

Randomized traffic was also applied during L0p transitions to maintain data integrity. Running long simulations where the device under test (DUT) continuously exchanges Transaction Layer Packets (TLPs) with the partner while periodically triggering L0p entry and exit at random intervals. Synthetic variabilities such as random packet sizes, varying inter-packet gaps, and random points in time for L0p requests (sometimes when buffers are full, sometimes when idle) are introduced to stress the DUT. A data integrity checker compared the transmitted and received flit streams across width changes to confirm that no flits were lost, duplicated, or disordered. Because Flit mode uses sequence numbers and end-to-end CRC, the simulation could also detect if any flit was corrupted or dropped due to a bug during a transition (which would violate flow control or CRC checks). No such data corruption was observed after the initial bugs were fixed, indicating the L0p mechanism can be transparent to the data stream.

Another aspect of the simulation was verifying fallback and corner cases. Simulating a mixed-generation link where the DUT (Gen6 capable) was connected to a Gen5-only partner (which does not support Flit mode or L0p). In this case, L0p was not enabled after link training and checked that the DUT correctly refrained from sending any L0p requests. This ensured backward compatibility logic (advertising and detecting L0p support) was correct. Performing error-injection tests: for example, deliberately dropping the partner's ACK DLLP in one test to see if the DUT times out and recovers (it did, by canceling the request and continuing in full width after 2 μs of no response). In another test, an L0p entry was forced. Immediately triggered a conventional recovery (simulating an unrelated error on the link) to ensure the state machines handle an abrupt exit from L0p to Recovery without issues.

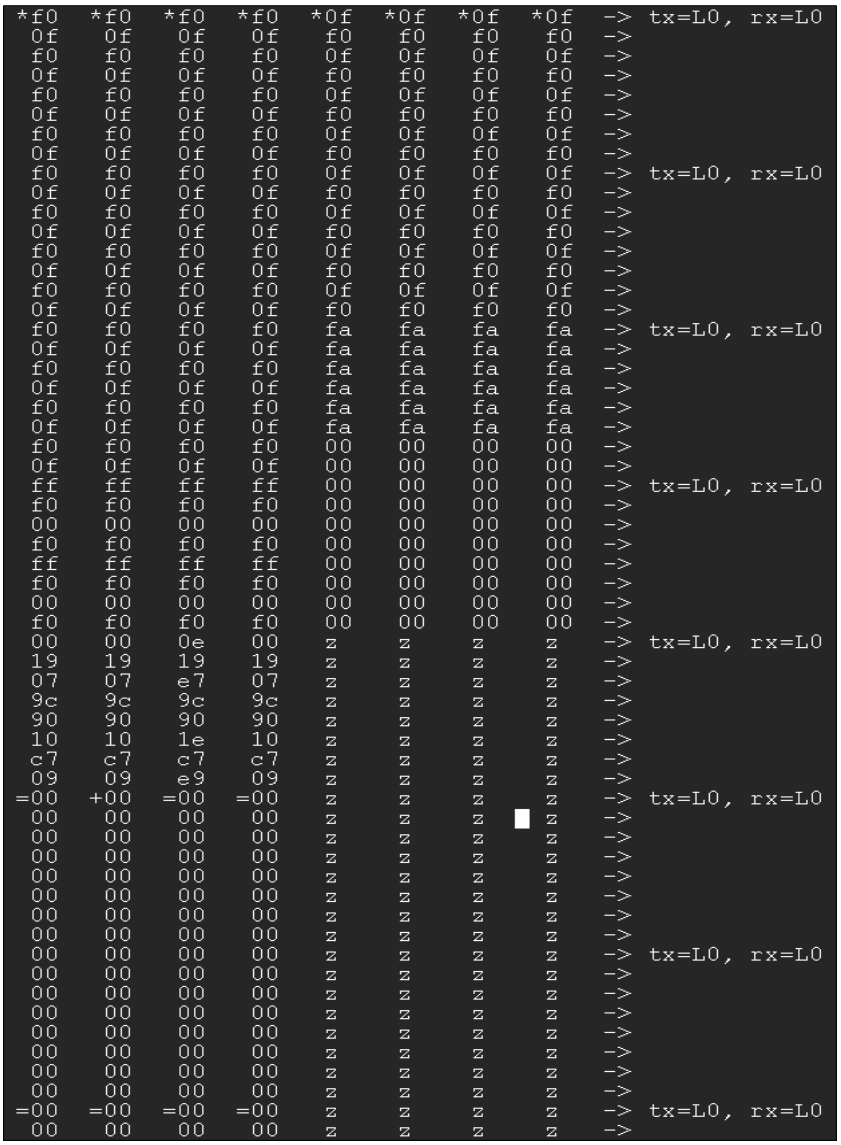

Figure 3 Link going down from x8 to x4. Control SKIP on lanes 0-3 and EIOS on lanes 4-7

Throughout the simulation, coverage metrics were gathered. Achieving coverage of all L0p request types (downsize to each possible width and upsizes back to full width), both roles (DUT initiating vs responding), all response types (ACK, NAK, timeout), and the lane sequencing variations (different lanes being turned off/on depending on which portion of the link is shed). The coverage also ensured that the boundary conditions were met, such as the maximum allowed latency for response and the minimum interval between successive L0p actions. By the end of the simulation campaign, reached near-100% coverage on all L0p-related events, increasing confidence that no untested scenario remained [3, 8].

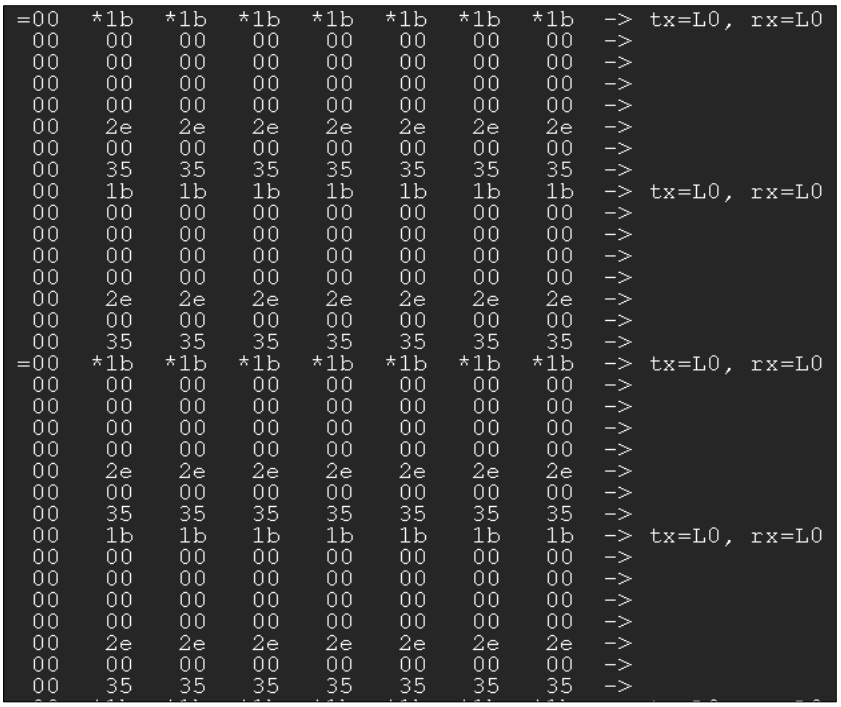

Figure 4 Link going up from x1 to x8. TS1 OS on lanes 1-7 and data flit on lane 0

7. Case Studies

To illustrate the effectiveness of the verification approach, present two representative pre-silicon bug scenarios ("case studies") uncovered during L0p state verification. Each case describes the issue, how it was detected and debugged via the methodology, and the resolution implemented in the design. These cases emphasize rigorously verifying the new L0p logic early in the design cycle, as subtle protocol bugs could otherwise slip through to silicon [4].

7.1. Case Study 1: Formal Detection of a Simultaneous Request Deadlock

Context: The PCIe 6.0 spec allows either end of the link to initiate an L0p width change, raising the possibility that both ends initiate nearly concurrently. The design under test had an arbitration scheme intended to handle this: if both sides requested L0p, the one requesting the lower link width (or with a priority flag) should win, and the other side should detect the conflict and abort its request. This arbitration logic was implemented in the LTSSM control block.

- **Issue:** Formal verification produced a counterexample trace where the LTSSM became stuck, and L0p was never entered, nor was the link returning to normal L0. The scenario involved both sides initiating an L0p Request for a moderate width reduction (each requested to go from ×16 down to ×8). In the problematic trace, each side's state machine sent out an L0p Request DLLP and waited for an ACK from the other side. Because both were waiting, neither side ever sent an ACK – a classic deadlock situation. The arbitration signals in the design was not triggered because both requests were identical (same target width), and the design did not correctly implement the spec's rule that one request should be nacked. Essentially, the logic to compare the two simultaneous requests and issue an NAK to one of them was missing. As a result, both sides of the model sat indefinitely, waiting on each other's ACK. This violated the protocol check that a request must eventually be resolved. The checker flagged this as a failure and provided the event trace showing both request signals high and no progress.
- **Debugging:** Using the formal counterexample, inspect the internal arbitration signals. Discovering that the design only handled simultaneous requests if the requested widths differed (it would pick the higher width request to favor, per spec). However, it did not handle the case of both sides requesting the same width change. The spec's guidance is that either side could have an implementation-defined priority or use the L0p.Priority bit to force one to yield [1]. The design hadn't implemented the optional L0p.Priority feature and assumed asymmetric requests. To confirm, write an additional check reflecting the spec: "If both sides request L0p to width N concurrently, the DUT should respond to the partner with a NAK (or receive a NAK) such that only one request succeeds." This check failed, confirming the design's shortcomings.

- **Resolution:** The design was updated to handle simultaneous identical requests by introducing a simple tie-break mechanism: if both sides request the same target width, the DUT will internally choose to defer its request (send an NAK to the partner's request) if, for example, the partner's port ID is numerically higher (an arbitrary but consistent rule). This effectively ensures one side wins and the other backs off. Also added support for the L0p.Priority bit, as defined in the spec, is used so that a port entering L0p for critical reasons (like thermal throttling) can set this bit and guarantee it wins over a partner's routine power-saving request. After these fixes, the formal deadlock trace was eliminated – the revised model showed that in concurrent request scenarios, one side always proceeded and the other aborted, allowing the link to transition to the intended width or remain at full width deterministically.
- **Impact:** If unresolved, this bug would have appeared when two PCIe 6.0 devices aggressively tried to enter L0p (for instance, two power-conscious endpoints). In silicon, the link would stall in an in-between state (each waiting for an ACK that never comes), likely requiring a forced retrain or reset by higher-level software to recover. This could manifest as a serious interoperability issue between devices from different vendors. Catching it in formal verification ensured the final design handled simultaneous L0p negotiations gracefully, preventing a potential post-silicon deadlock situation.

7.2. Case Study 2: Simulation of an Upsizing Sequence Glitch

- **Context:** L0p exit (increasing link width) is more complex than entry. The design must bring previously idle lanes back online in sync. The specification-prescribed sequence (for Gen3+ speeds) is: EIEOS → TS1/TS2 training on new lanes → SDS indication → resume traffic on all lanes. The DUT's implementation of this sequence relied on a micro-coded finite state machine, which would orchestrate sending the ordered sets and enabling the lanes at the right time.
- **Issue:** In a directed simulation test, the DUT (root complex side) attempted to scale the link from ×2 back up to ×4. The partner model sent the expected training sequences on lanes 2 and 3, indicating readiness. The DUT should have sent the "Start of Data Stream" (SDS) marker to incorporate those lanes into the active data flow. However, in the simulation waveform, it was observed that as soon as the partner completed TS2, the DUT immediately started transmitting flits on lanes 2 and 3 without sending an SDS or waiting for a Control SKP alignment. This was a protocol violation – the SDS must mark a clean handoff of those lanes to normal traffic [1]. Consequently, the partner ignored the unexpected flits on those lanes (since it hadn't seen an SDS) and treated them as training bits. This led to a momentary loss of flit synchronization: the first few flits sent on lanes 2-3 were not accepted by the partner, causing a mismatch in delivered data (which the checker caught as missing flit sequence numbers at the receiver). Things realigned after the next SKP interval, but the damage was done in terms of data loss.
- **Debugging:** The simulation logs and waveforms made the issue clear. Traced the DUT's internal signals and found that the state machine responsible for upsizing didn't insert an SDS because of a corner-case bug: it was coded to send SDS only for Gen3 and above speeds, and the test was running at Gen4 (16 GT/s), which is indeed ≥Gen3. The condition was correct, so why was SDS skipped? It turned out an internal flag that indicates "new lanes training completed" was being cleared one cycle too early, so the logic that triggers SDS saw this flag low and erroneously assumed SDS was not needed (as if in Gen1/2 mode where SDS isn't used). Essentially, a race condition caused the SDS to be missed. Confirmed this by re-running the simulation with slight timing tweaks and seeing that occasionally, the SDS would appear if the timing aligned differently. This non-deterministic behavior reinforced that a bug was present.
- **Resolution:** The dut's upsizing sequence generator was fixed to latch the training-complete signal properly and only clear it after issuing the SDS. Also added an explicit check: if the link is in Flit mode and new lanes are being added at Gen3+, force an SDS transmission on those lanes. After the fix, the simulation showed the DUT sending TS1/TS2 on lanes 2-3, then an SDS, and only after that did it begin sending flits on all four lanes. The partner model synchronized perfectly, and no flits were lost. Expanded the tests to cover upsizing at Gen5 and Gen6 speeds; in each case, the corrected design behaved correctly.
- **Impact:** Without this fix, a real hardware link might have experienced data corruption or misalignment when coming out of L0p. In practice, higher-level error detection (like the Data Link Layer's replay mechanism or End-to-End CRC) would likely catch and recover from the dropped flits. However, this would introduce performance penalties and complexity. More seriously, it could complicate compliance testing – a protocol analyzer might flag the missing SDS as a spec violation. Thanks to the simulation, the timing glitch was identified and resolved early. This case underscored the value of cycle-accurate simulation in catching issues that formal methods (which abstract time) might miss [6, 8].

Table 1 Verification Results Snapshot [4, 7]

Aspect	**Metric/Result**
LTSSM State Coverage (L0p)	100% (All L0p entry/exit transitions covered)
L0p Handshake DLLPs	100% of Request/ACK/NAK scenarios covered
Width Change Combinations	15/15 combinations tested (all downsizes/upsizes for x16/x8/x4/x2/x1)
Simultaneous Request Scenario	Formally proven deadlock-free (covered by directed test as well)
L0p Entry Latency (simulated)	1.5 µs worst-case observed (matches spec boundary)
L0p Exit Latency (simulated)	~5–8 µs typical (depending on idle duration, matches L1 exit order)
Data Integrity	0 data errors in 10^6+ transactions (checked end-to-end)
Power State Correctness	No incorrect L0s/L0p overlap (ensured L0s off in FLIT mode)
Post-fix Regression Failures	0 over 500+ random seeds and directed suites

8. Conclusion

Verifying the PCIe 6.0 L0p low-power state demonstrates the critical importance of thorough pre-silicon validation when implementing new protocol capabilities. L0p's ability to scale link width dynamically offers substantial power savings with minimal performance impact but introduces complex state machine behaviors that require flawless implementation. Through combined formal verification and simulation, exhaustively exercised the L0p logic and identified subtle bugs that could have led to link instability or data errors. Formal methods provided comprehensive coverage of corner cases and timing proofs, while simulation enabled detailed inspection of lane-by-lane sequences and integration with real traffic. This complementary approach ensured the controller's LTSSM extension for L0p adhered to specifications across all scenarios. The successful first-pass silicon validation confirms that even for novel features like L0p, rigorous verification can virtually eliminate post-silicon issues. As PCIe technology evolves, features like L0p will remain essential for balancing performance and power, with the methodologies from this work informing verification of future innovations. Thorough verification of power management states is as crucial as primary functional logic since flaws in these secondary states could compromise the reliability and efficiency of PCIe links in deployed systems.

References

[1] PCI-SIG, “PCI Express Base Specification, Revision 6.0,” Jan. Available: https://pcisig.com/pci-express-6.0-specification

[2] D. Das Sharma, “The PCIe 6.0 Specification Webinar Q&A: Leveling Up with L0p,” PCI-SIG Blog, Feb. 2025. Available: https://pcisig.com/pcie%C2%AE-60-specification-webinar-qa-leveling-l0p

[3] Synopsys, “Optimizing PCIe 6.0 Designs at 64GT/s,” Synopsys Technical Bulletin, 2022. Available: https://www.synopsys.com/articles/pcie-6-designs.html

[4] Synopsys, “PCIe 6.0 Power and Latency in HPC SoCs,” Synopsys Whitepaper, 2025. Available: https://www.synopsys.com/articles/pcie-6-hpc-socs.html

[5] PCI-SIG, “PCIe 6.0 Specification: The Interconnect for I/O Needs of the Future (presentation),” June 2020. Available: https://pcisig.com/sites/default/files/files/PCIe%206.0%20Webinar_Final_.pdf

[6] J. Eydoux, “Revolutionizing Power Efficiency in PCIe 6.x: L0p and Flit Mode in Action,” Rambus Blog, Jan. 2025. Available: https://www.rambus.com/blogs/revolutionizing-power-efficiency-in-pcie-6x-l0p-and-flit-mode-in-action/

[7] Gary Ruggles, “Successful PCI Express 6.0 Designs at 64GT/s with IP,” Synopsys, Apr 19, 2021. Available: https://www.synopsys.com/articles/pcie-6-designs.html

[8] Madhumita Sanyal, “PCIe 6.0 Power and Latency Challenges in High-Performance Computing SoCs,” Synopsys, Jan 16, 2024. Available: https://www.synopsys.com/articles/pcie-6-hpc-socs.html