[Public]

Solving Time at Scale: PCIe® Precision Time Measurement on AMD EPYC™ Platforms

Publication # **69420**	Revision: **0.90**
Issue Date: **September 2026**	

Advanced Micro Devices

[Public]

© 2026 Advanced Micro Devices, Inc. All rights reserved.

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD's Standard Terms and Conditions of Sale.

Trademarks

AMD, the AMD Arrow logo, AMD EPYC, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

[Public]

Contents

Chapter 1 Executive Summary 6
Chapter 2 Introduction – The Cost of Uncertain Time 7
2.1 Time as a Hidden Dependency at Scale 7
2.2 AMD's Approach: Hardware-Correlated Time from Root Port to Endpoint 7
Chapter 3 Background: From Network Time to Platform Time 9
3.1 AMD's Approach: Hardware-Correlated Time from Root Port to Endpoint 9
3.2 The "Last Hop" Problem 10
Chapter 4 PCIe Precision Time Measurement 11
4.1 Requester, Responder, and Root 11
4.2 The PTM Dialog 11
Chapter 5 The AMD Platform Time Architecture 13
5.1 The Golden and Leaf Timestamp Counters 13
5.2 From Leaf TSC to PTM Master Time 14
5.3 Time Alignment across Dies and Sockets 14
5.3.1 External Trigger 14
5.3.2 Local Trigger 14
5.3.3 Time Alignment Debug 15
Chapter 6 Spread Spectrum Clocking and the Software Conversion Path 16
Chapter 7 Use Cases for Modern AI and Datacenter Workloads 17
7.1 Distributed AI Training and Inference 17
7.2 Globally Consistent Databases and Transaction Ordering 17
7.3 Observability: Distributed Tracing, Telemetry, and Triggers 17
Chapter 8 Deployment Considerations 18
Appendix A Glossary 19
Appendix B References 20

[Public]

List of Figures

Figure 1. Server Notion of Time Sample Flow 9
Figure 2. PTM Dialog from PCIe Base Specification (PCI-SIG® n.d.) 11
Figure 3. Golden TSC Feeding Leaf Timestamp Counter Sample Flow 13

[Public]

Revision History

Date	Revision	Description
September 2026	0.10	Initial public release.

[Public]

Chapter 1 Executive Summary

Modern artificial intelligence (AI) and datacenter workloads are distributed across thousands of servers, accelerators, and network interface cards (NICs). As these systems scale out, a subtle but pervasive dependency emerges: every node must agree on what time it is. Distributed databases order transactions by time, observability platforms correlate traces across machines by time, and large-scale AI training jobs depend on a coherent time base to detect stalls and imbalance across the data center fabric. When clocks disagree by microseconds, that uncertainty becomes a tax on correctness, performance, and debuggability/telemetry of the workloads running in a distributed environment.

PCI Express® (PCIe) Precision Time Measurement (PTM) addresses this problem at the platform level. PTM is an optional PCIe capability that lets a Root Port and its attached endpoints exchange timestamped messages and compute the propagation delay between them, establishing a common, hardware-correlated time reference across the PCIe link. In effect, PTM does for the platform level PCIe bus hierarchies what the Precision Time Protocol (PTP) does for an Ethernet network – it tightly couples the clock of one device to the clock of another.

AMD EPYC™ processors, available starting with the “Venice” generation, implement PTM in the PCIe Root Port so that the platform can act as the source of time (the PTM Root) for time-sensitive endpoints such as PTM-capable NICs and accelerators. By anchoring PTM to the on-chip always-running timestamp counter and correlating it across the PCIe link, AMD reduces the effective synchronization error between a device clock and the host system clock from the microsecond range typical of software-only methods down to the order of tens of nanoseconds. This whitepaper explains how PTM works, how the AMD platform time architecture delivers that precision, and why tens-of-nanosecond synchronization is becoming a foundational requirement for AI and datacenter infrastructure.

[Public]

Chapter 2 Introduction – The Cost of Uncertain Time

2.1 Time as a Hidden Dependency at Scale

For a single server, software rarely worries about clock accuracy: the operating system reads the processor's timestamp counter and treats it as authoritative. On a datacenter scale, that assumption breaks down. A distributed workload or a database that spans racks must decide which of two transactions happened first. The lesson generalizes: the tighter and more trustworthy the shared clock notion, the simpler and faster distributed coordination becomes.

The same pressure now appears inside AI and datacenter fabrics. Observability systems stitch together traces distributed from many services to reconstruct the path of a single request; if node clocks disagree by microseconds, spans overlap incorrectly and root-cause analysis suffers. Large-scale training and inference jobs coordinate across many accelerators and NICs, where consistent timestamps help operators distinguish a genuinely slow node from one that merely reported time differently. Across finance, telecommunications, industrial automation, a wide variety of workloads already demand sub-microsecond synchronization. As workloads scale, the acceptable error budget keeps shrinking.

PTP solves half of the problem. It can synchronize the clock inside a NIC, its PTP Hardware Clock (PHC), to a network grandmaster with sub-microsecond accuracy. But most application software runs against the host's system clock, not the NIC clock, so the time reference still must cross the PCIe link from the NIC into the processor. Reading the NIC clock over PCIe in software stack is inherently slow and noisy: under load, the round-trip can vary by hundreds of nanoseconds, which becomes the dominant error term.

2.2 AMD’s Approach: Hardware-Correlated Time from Root Port to Endpoint

AMD’s implementation of PCIe Precision Time Measurement closes this "last hop" in hardware. AMD EPYC processors implement the PTM Responder and Root role in the PCIe Root Port, allowing the platform to distribute a common PTM master time to each PTM-capable endpoint. That master time is derived from the processor's always-running Golden Timestamp Counter (GTSC), so the value shared over PCIe is the same coherent time base that the rest of the system uses. The result is a standards-based, scalable mechanism that brings device-to-host synchronization down to the order of tens of nanoseconds.

The remainder of this paper builds the picture in layers: Chapter 3 traces the notion of time from the network grandmaster to the host system clock and shows where PTM fits; Chapter 4 details the PCIe PTM protocol; Chapter 5 describes the AMD platform time architecture that feeds it;

[Public]

Chapter 6 explains the Spread Spectrum Clocking effect on the software conversion path and how AMD enables precise cross-timestamps in the Linux kernel; Chapter 7 maps the capability to concrete AI and datacenter use cases; and Chapter 8 covers the Deployment Considerations.

[Public]

Chapter 3 Background: From Network Time to Platform Time

3.1 AMD's Approach: Hardware-Correlated Time from Root Port to Endpoint

At a datacenter scale, time originates from a single authoritative source and flows outward. The Precision Time Protocol (PTP) defines a hierarchical distribution in which a *grandmaster* clock, often disciplined by a GPS or atomic reference clock, propagates time across the network to every node.

Inside a server, the NIC is the first component to receive network time. A PTP-capable NIC contains a PHC, that is typically a high-precision 64-bit counter, that a daemon such as `ptp4l` from the Linux `linuxptp` package disciplines to the grandmaster. After this step, the NIC knows the time to sub-microsecond precision. The application software that needs the notion of time, however, does not read the NIC clock directly; it reads the operating system's clocks — for example `CLOCK_REALTIME` (adjustable wall time) and `CLOCK_MONOTONIC` (a non-decreasing elapsed-time clock) through *`clock_gettime()`* methods. These OS clocks are ultimately backed by the processor's timestamp counter, and not by the NIC PHC. The notion of time therefore has one more hop to traverse: from the NIC's PHC across the PCIe link into the processor's time domain, as shown in *Figure 1* below.

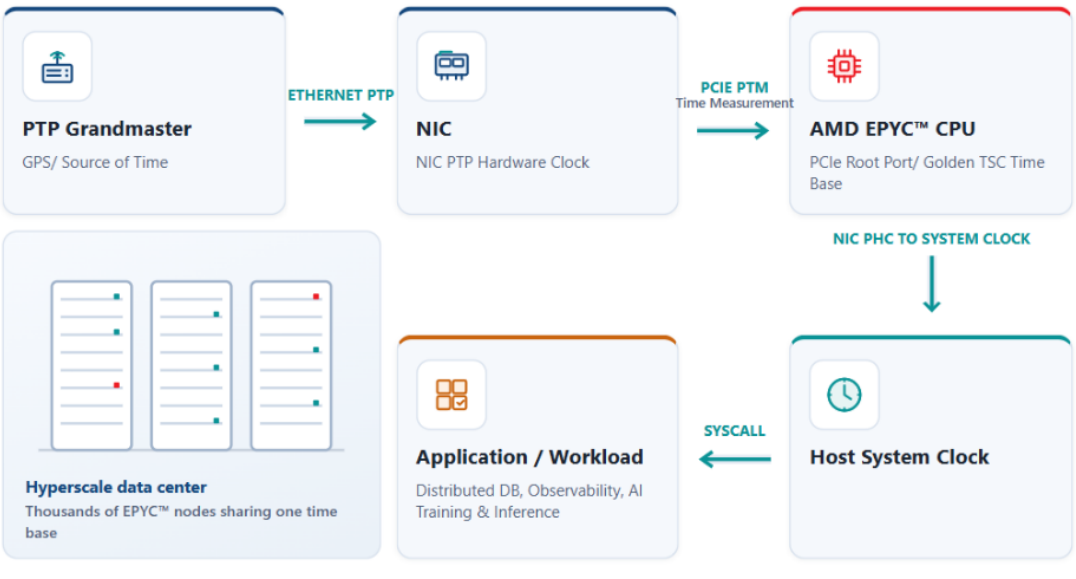

Figure 1. Server Notion of Time Sample Flow

[Public]

Figure 1 illustrates the Notion of Time Propagating from an Authoritative PTP Grandmaster Across Ethernet to the NICs, PHC and then over the PCIe link to the AMD EPYC Processor.

3.2 The “Last Hop” Problem

Bridging the NIC clock to the host clock is the hardest link in the chain. The Linux `linuxptp` stack uses a second daemon `phc2sys`, to discipline `CLOCK_REALTIME` to the NIC PHC, and it may do so by using a brute-force approach: software reads the PHC time and the processor's timestamp counter and continuously corrects the system clock by the measured delta. Because those reads traverse the PCIe link, their latency varies with system load, and the resulting error can reach hundreds of nanoseconds - large enough to dominate the entire synchronization budget.

PTM resolves this tension. Rather than estimating the link crossing in software, PTM measures the link propagation delay directly, in hardware, using the ordinary PCIe main-band path. Any PTM-capable endpoint on the fabric can participate in the exchange, so a single mechanism scales from one device to many endpoints. The next section describes how that measurement works.

[Public]

Chapter 4 PCIe Precision Time Measurement

4.1 Requester, Responder, and Root

PTM is defined as an optional PCIe Extended Capability. A PTM-capable function advertises its abilities through the PTM Capability register, which exposes discrete role bits: *PTM Responder Capable*, *PTM Requester Capable*, and *PTM Root Capable*, along with the local clock granularity. The roles map onto the PCIe topology:

- A **PTM Root** is the ultimate source of PTM master time in a PTM hierarchy. It is a property of a Root Complex — a Root Port that is designated as the time source.
- A **PTM Responder** answers PTM requests. A downstream Root Port acts as the Responder for the endpoints beneath it.
- A **PTM Requester** initiates the dialog. An endpoint, such as a PTM-capable NIC or accelerator, is the Requester that pulls master time from the Root Port.

On AMD EPYC platforms, the PCIe Root Port implements the Responder and Root roles, enabling the processor to serve as the platform time source and advertise master time to attached endpoints. Software can confirm the configuration directly. On Linux, `lspci` reports the capability; for example, showing `PTMCap: Responder+ Root+` and the advertised clock granularity for a Root Port that sources time.

4.2 The PTM Dialog

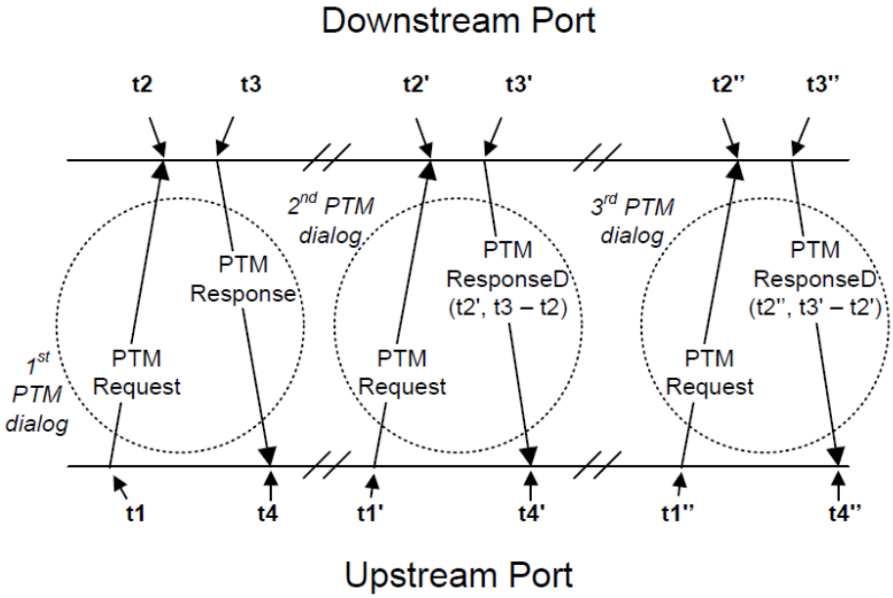

Figure 2. PTM Dialog from PCIe Base Specification (PCI-SIG® n.d.)

[Public]

PTM synchronizes two clocks through a short exchange of messages. The Requester sends a **PTM Request**; the Responder replies with either a **PTM Response (Rsp)** or, when it has valid historical timestamps available, a **PTM ResponseD (RspD)** that carries a 64-bit master-time value and the measured propagation delay. Across the exchange, each port captures local timestamps at the instant when PTM messages are sent/received - conventionally labeled t1 through t4: the Requester records the transmit time (t1) and receive time (t4), while the Responder records the receive time (t2) and transmit time (t3). Because a single dialog cannot convey a port's own transmit timestamp back in time, the Responder reports timestamps from previous exchanges in the ResponseD, and the Requester combines its current and prior timestamps to compute the corrected master time for its own clock domain. The exchange is described in detail in the *PCIe Base Specification Chapter 6.21* titled as *"Precision Time Measurement (PTM) Mechanism"*.

To keep the measurement precise, both sides capture timestamps as close to the physical layer as possible and hold off other transmitting traffic while a PTM message is scheduled, so the interval between deciding to send and transmitting is within a bound. The protocol also bounds the cadence of the dialog: after receiving a response, a Requester must wait at least 1 µs before issuing another PTM Request, and it must not wait more than 100 µs for a response before considering the dialog failed. The implementation may expose the captured t2/t3 (Responder timestamps) and the computed results (Requester) to firmware or software through implementation of specific registers.

AMD's PTM implementation is built for the full range of the PCIe generations the platform supports, spanning PCIe Gen1 through Gen6, including operation in Flit Mode which is introduced with PCIe Gen 6 specification. In Flit Mode, timestamps are aligned to the flit marker that frames the PTM message, whereas in non-flit mode they align to the framing symbol of the transaction-layer packet; in both cases the goal is to timestamp the message as though observed at the port's receiver. PTM operates on native PCIe links; it is not offered in CXL mode of operation.

Chapter 5 The AMD Platform Time Architecture

PTM is only as good as the time it distributes. The precision an endpoint ultimately sees depends on the quality of the clock the Root Port hands out and on how tightly that clock is aligned with the rest of the components on the platform. AMD EPYC processors build PTM on top of purpose-designed, system-wide timestamp architecture.

5.1 The Golden and Leaf Timestamp Counters

At the heart of the architecture is the Golden Timestamp Counter (Golden TSC, or GTSC), an always-running, free-running counter that provides a continuous time reference for the system-on-chip. The Golden TSC is 56 bits wide, starts from zero, and is driven from a fixed reference clock, most commonly 100 MHz, giving a 10 ns tick; at 100 MHz the 56-bit counter provides roughly 23 years before it rolls over. Because it lives in an always-on power domain, it continues to advance time even while other blocks enter low-power states.

Consumers across the chip, CPU cores, accelerators, and I/O blocks such as PCIe Root Ports, do not read the Golden TSC directly. Instead, each block holds a *leaf* timestamp counter that is kept synchronized to its Golden TSC and delivered into the block's own clock domain. Synchronization between the Golden TSC and each leaf uses a controlled hand-off, so that the leaf's count is always aligned with the golden reference. Figure 2 shows this hierarchy feeding the PCIe Root Port.

Figure 3. Golden TSC Feeding Leaf Timestamp Counter Sample Flow

In Figure 3 above, a single always-on Golden TSC feeds per-block leaf timestamp counters in each IP; the PCIe leaf timestamp drives the Root Port PTM engine, which distributes the notion of time.

[Public]

5.2 From Leaf TSC to PTM Master Time

When a Root Port participates in PTM, it draws the t2 and t3 timestamps from the PCIe Leaf TSC. Although the Root Port physically samples the PCIe leaf TSC, firmware keeps that leaf synchronized to the Golden TSC; it therefore carries Golden-TSC time. That counter increments in 10 ns reference-clock units, so the PTM logic multiplies it by ten to express time in nanoseconds before building the PTM message. The net effect is that the time an endpoint receives over PTM is a faithful, nanosecond-scaled view of the same Golden TSC that the rest of the platform runs on.

5.3 Time Alignment Across Dies and Sockets

An EPYC platform spans multiple dies and, in multi-socket systems, multiple processors, each with its own version of Golden TSC. The SOC Firmware acts as a single local coordinator that drives one clean edge, which results in every GTSC latching its value at that common instant, and the firmware then rewrites each counter so that the trigger edge becomes a shared time zero. For a multi-socket system, since both sockets are zeroed against the same physical edge, the two sockets' time bases become aligned to within the platform skew budget. The PTM engine inherits this alignment.

There are two mechanisms to achieve TSC synchronization at boot time and may be chosen through a BIOS option in accordance with the platform configuration.

5.3.1 External Trigger

The external trigger scheme provides customers with an ability to independently trigger and verify TSC synchronization from outside of the EPYC CPU. This scheme is the recommended mode of operation as it allows greater platform debuggability and control from the user’s side. Once PTM has run, a device can realize the host TSC value over PCIe but confirming that the device's realized time tracks the x86 TSC has traditionally required software to read registers over the link, a path that is slow and imprecise.

5.3.2 Local Trigger

In platform configurations where there is no external source to assert the TriggerTSC pin, the SOC Firmware does not depend on an external trigger. Instead, the FW will trigger internally the synchronization process during the power-on flow. For a multi-socket system, the BSP drives the TriggerTSC pin as an output, which is routed to the AP’s TriggerTSC pin as an input. Please note that this routing delay needs to be less than 10ns.

[Public]

5.3.3 Time Alignment Debug

The PTM capable devices that support TGPIO may use a debug scheme for time correlation that can be checked against a hardware edge as opposed to Software polling. EPYC CPU receives the Trigger TSC GPIO in input mode, which is an external edge that is captured against the platform time base. Once received, the Firmware records the captured TSC values in a snapshot register that may be read back through the APML interface. Please refer to the Motherboard Design Guide and Processor Programming Reference for more details. In summary, this scheme works in the following way:

- BMC sends the TSC_ENABLE APML command using the *'Trigger_TSC'* parameter to notify the SOC Firmware to expect a Trigger TSC rising edge that may be driven by an external entity.
- SOC Firmware takes preparation actions for the arrival of the edge.
- The BMC allows the external entity to drive the Trigger TSC edge. Please note that TriggerTSC signal arrival at all sockets (in case of a multi-socket system) must be skew-matched to less than 10 ns.
- BMC sends the TSC_ENABLE APML command with *'Fetch_TSC'* parameter to retrieve the TSC values (from CCDs and IODs) captured on the most recent trigger for synchronization validation.

[Public]

Chapter 6 Spread Spectrum Clocking and the Software Conversion Path

Spread Spectrum Clocking is a widely used electromagnetic-interference mitigation technique that deliberately modulates (spreads) the platform reference clock a fraction of a percent from its nominal frequency. On the AMD platform, the nominal 100 MHz reference clock is spread when SSC is enabled, and both the PTM Engine timestamp counter and the CPU core timestamp counter are ultimately derived from that same modulated reference. Please note that the Spread spectrum is not the only source of deviation: the board-supplied reference clock is itself allowed a small PPM tolerance, so even with SSC disabled some parts run marginally fast and others marginally slow.

The PTM block converts its Leaf timestamp count to nanoseconds using a fixed scale factor (i.e., multiply by ten), that assumes the reference clock runs at exactly the nominal 100 MHz (a 10 ns tick). When SSC has shifted the real reference away from 100 MHz, that fixed ×10 no longer yields true nanoseconds: the value the PTM block reports as "nanoseconds" advances slightly slower than one billion times per real second. The Linux kernel, meanwhile, calibrates the CPU TSC frequency against wall time, so its notion of TSC frequency already reflects the *SSC-averaged* rate.

The insight that makes a robust fix possible is that although the *absolute* rate of each clock wanders with SSC, the *ratio* between the TSC used by the PTM engine and the CPU core TSC does not: both are driven by the same modulated reference, so the modulation cancels in their ratio. That ratio can be obtained from the P0 (base) P-state frequency, which is read from `MSR_AMD_PSTATE_DEF_BASE` as the core frequency ID (FID) in bits [11:0].

The Linux enablement drafted by AMD SW team applies this ratio through the kernel's existing clock source *base clock* mechanism. High level changes in these patches include:

- A new clock source identifier for the AMD TSC root clock proposed as `CSID_X86_GTSC`.
- An *`X86_FEATURE_GTSC`* feature bit and a `detect_gtsc()` routine that reads the FID, programs the PTM-TSC-to-CPU-TSC ratio from the P0 frequency and registers the GTSC as a base clock of the CPU TSC clock source. The base-clock *offset* is set to zero because firmware aligns the per-CCD/IOD master TSC to the Core/PTM Leaf TSC at boot.
- A driver change that reports the captured PTM root-clock timestamp with the new clock source ID and sets use_nsecs = false, when `X86_FEATURE_GTSC` is present. Clearing `use_nsecs` tells the timekeeping core to apply the registered base-clock *ratio* instead of treating the captured value as true nanoseconds.

[Public]

Chapter 7 Use Cases for Modern AI and Datacenter Workloads

Tens-of-nanoseconds synchronization is not an end in itself; it is an enabling capability for a growing set of distributed workloads. This section maps PTM to the scenarios where a common, high-precision platform time changes what infrastructure can do.

7.1 Distributed AI Training and Inference

Large-scale AI training and inference distribute a single job across many accelerators and NICs, often spanning multiple nodes. Operators depend on fine-grained telemetry to keep these jobs efficient: identifying the slowest participant in a collective operation, detecting communication stalls, and attributing lost time to the right stage of the pipeline. When every node tries to stamp its events against a common platform clock accurate to tens of nanoseconds, those cross-node timelines can be compared directly instead of being blurred by clock disagreements. Because PTM anchors the endpoint clock — including a PTM-capable NIC's PHC — to the same platform time the host uses, it also gives the networking and compute devices of an AI system a shared notion of "now." That shared time base is the foundation for correlating network events (such as congestion or packet delays) with computing events (such as stalled workloads, telemetry logs) across the data center fabric.

7.2 Globally Consistent Databases and Transaction Ordering

Distributed databases must agree on the order of events. The tighter the time bound, the shorter a database must wait to guarantee ordering, and the higher its achievable throughput. PTM brings a comparable philosophy inside the server: a precise, hardware-correlated platform clock that transaction logs, data streams, and commit protocols can rely on.

7.3 Observability: Distributed Tracing, Telemetry, and Triggers

Observability is where precise time pays off immediately. Distributed tracing reconstructs the life of a request as it crosses services and machines; if node clocks disagree, the causal ordering of spans becomes ambiguous and latency is misattributed. A platform clock accurate to tens of nanoseconds lets a fleet assemble coherent traces, align telemetry streams from independent nodes, and fire time-based triggers consistently across machines.

[Public]

Chapter 8 Deployment Considerations

Realizing PTM in production requires capable hardware on both ends of the PCIe link and a software stack configured to use it.

- **Hardware support**. PTM requires implementation in both the PCIe Root Port and the endpoint. AMD EPYC processors provide the Root Port Responder and Root roles; PCIe attached devices such as a NIC or other accelerators must implement the PTM Requester role.
- **Platform support**. The recommended configuration is to use External TriggerTSC pin to control the synchronization point on the platform. Please refer to the AMD Motherboard Design Guide for platform routing considerations and the AMD AGESA spec that outlines the BIOS options. Please contact your AMD representative for Open BMC NDA reference code that implements the support for Virtual messaging.
- **Link coverage**. The AMD implementation supports PCIe Gen1 through Gen6, including Flit Mode; PTM is not offered in CXL mode.
- **Software stack**. On Linux, the standard `linuxptp` tools apply directly: `ptp4l` disciplines the NIC PHC to the network grandmaster, and `phc2sys` disciplines the system clock using the precise cross-timestamp exposed by *`PTP_SYS_OFFSET_PRECISE`*. Operators can confirm PTM capability and configuration with `lspci`, which reports the PTM roles and advertised clock granularity for a port. As previous sections explain, achieving the hardware's full precision on AMD depends on converting the PTM timestamp with the SSC-independent, FID-derived PTM-TSC-to-CPU-TSC ratio rather than treating the PTM value as true nanoseconds; deployers should run a kernel that registers the PTM TSC/Golden TSC as a base clock and validate end-to-end `phc2sys` convergence rather than trusting an uncalibrated software reading.

The PTM capability itself is hardware- and platform-driven and therefore OS-agnostic; the software enablement described here is specific to Linux. Other operating systems can implement an equivalent stack to take advantage of the same hardware, but that is outside the scope of this paper. Because PTM builds on standard PCIe capabilities, it integrates into existing time-synchronization deployments without custom sideband wiring or application rewrites.

[Public]

Appendix A Glossary

Term	Definition
PTM	Precision Time Measurement — a PCIe capability for correlating clocks across a link
PTP	Precision Time Protocol (IEEE 1588) — network clock synchronization
PHC	PTP Hardware Clock — the high-precision clock inside a NIC
Golden TSC	The always-running, system-wide timestamp counter reference on AMD EPYC
Leaf TSC	A per-block timestamp counter synchronized to the Golden TSC
Requester / Responder / Root	PTM roles: the endpoint that requests time, the port that answers, and the designated time source
ResponseD (RspD)	A PTM response carrying master time and propagation delay
Flit Mode	Flit-based framing used by the latest PCIe Gen 6 generation
phc2sys / ptp4l	Linux linuxptp daemons that discipline the system clock and the NIC PHC
TriggerTSC	A shared GPIO mechanism that aligns Golden TSCs across dies and sockets
SSC	Spread Spectrum Clocking — deliberate modulation of a reference clock to reduce electromagnetic interference
FID	Frequency ID — P-state frequency identifier; read from MSR_AMD_PSTATE_DEF_BASE bits [11:0]

[Public]

Appendix B References

1) *AMD64 Architecture Programmer's Manual (*PID#40332pub*)*
2) *NCB-PCI_Express_Base_6.4, PCI-SIG®* (login required)