Intel® Trust Domain Extensions (Intel® TDX) Module TD Partitioning Architecture Specification

354807-004US

October 2024

.

Notices and Disclaimers

Intel Corporation (“Intel”) provides these materials as-is, with no express or implied warranties.

All products, dates, and figures specified are preliminary, based on current expectations, and are subject to change without notice. Intel does not guarantee the availability of these interfaces in any future product. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

The products described might contain design defects or errors known as errata, which might cause the product to deviate from published specifications. Current, characterized errata are available on request.

Intel technologies might require enabled hardware, software, or service activation. Some results have been estimated or simulated. Your costs and results might vary.

No product or component can be absolutely secure.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted that includes the subject matter disclosed herein.

No license (express, implied, by estoppel, or otherwise) to any intellectual-property rights is granted by this document.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice.

Copies of documents that have an order number and are referenced in this document or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting http://www.intel.com/design/literature.htm.

Table of Contents

SECTION 1:
INTRODUCTION AND OVERVIEW

10. About this Document

10.1. Scope of this Document

This document describes the architecture and the external Application Binary Interface (ABI) of the Intel® Trust Domain Extensions (Intel® TDX) module's TD Partitioning feature, implemented using the Intel TDX Instruction Set Architecture (ISA) extensions.

This document is part of the **TDX Module Architecture Specification Set**, which includes the following documents:

Table 10.1: TDX Module Architecture Specification Set

Document Name	Reference	Description
TDX Module Base Architecture Specification	[TDX Module Base Spec]	Base TDX module architecture overview and specification, covering key management, TD lifecycle management, memory management, virtualization, measurement and attestation, service TDs, debug aspects etc.
TDX Module TD Migration Architecture Specification	[TD Migration Spec]	Architecture overview and specification for TD migration
→ **TDX Module TD Partitioning Architecture Specification**	[TD Partitioning Spec]	Architecture overview and specification for TD Partitioning
TDX Module TDX Connect Specification	[TDX Connect Spec]	Architecture overview and specification for TDX Connect
TDX Module ABI Reference Specification	[TDX Module ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the entire TDX module architecture
TDX Module TDX Connect ABI Reference Specification	[TDX Connect ABI Spec]	Detailed TDX module Application Binary Interface (ABI) reference specification, covering the TDX connect architecture
TDX Module ABI Reference Tables	[TDX Module ABI Tables]	A set of JSON format files detailing TDX module Application Binary Interface (ABI)
TDX Module ABI Incompatibilities	[TDX Module ABI Incompatibilities]	Description of the incompatibilities between TDX 1.0 and TDX 1.4/1.5 that may impact the host VMM and/or guest TDs

This document is a work in progress and is subject to change based on customer feedback and internal analysis. This document does not imply any product commitment from Intel to anything in terms of features and/or behaviors.

Note: The contents of this document are accurate to the best of Intel's knowledge as of the date of publication, though Intel does not represent that such information will remain as described indefinitely in light of future research and design implementations. Intel does not commit to update this document in real time when such changes occur.

10.2. Document Organization

The document has multiple sections:

- Section 1 contains an introduction to the document, overview of TD Partitioning, scenarios and requirements.
- Section 2 contains the Intel TDX Module TD Partitioning architecture
- Section 3 contains updates to the TD Module ABI. This section will eventually be merged into the [TDX Module ABI Spec].

10.3. Glossary

For a complete TDX module glossary, see the [TDX Module Base Spec].

Table 10.2: Intel TDX Glossary for TD Partitioning

Acronym	Full Name	New for TDX	Description
	L2 Page Alias	Yes	Mapping of a TD private page in an L2 VM's GPA space
L1 VM	**Layer 1 VM**	No	The main virtual machine of the TD, which may also have the role on an L1 VMM
L1 VMM	**Layer 1 VMM**	No	The main virtual machine of the TD, which also serves as a Virtual Machine Monitor (VMM) for nested L2 VM guests
L2 VM	**Layer 2 VM**	No	A virtual machine which is nested within a TD, and is logically a guest of the L1 VMM
VM	**Virtual Machine**	No	A virtual multi-LP processor, contained within a TD

10.4. Notation

See the [TDX Module Base Spec].

10.5. References

See the [TDX Module Base Spec].

11. TDX TD Partitioning Overview

For an overview of TDX, refer to the [TDX Module Base Spec].

11.1. Introduction

TDX TD Partitioning, as described in this document, is designed to provide a minimal environment for supporting the Microsoft VSM and similar architectures. It supports a single unmodified legacy VM and up to two additional supervisory VMs within a guest TD.

TD Partitioning extends the base TDX architecture by allowing TDs to contain multiple virtual machines (VMs). A TD may contain up to 4 VMs. The primary VM (known as the L1 VM) may act as a virtual machine monitor (known as the L1 VMM). Up to 3 nested VMs (known as the L2 VMs) serve as the guest of the L1 VMM.

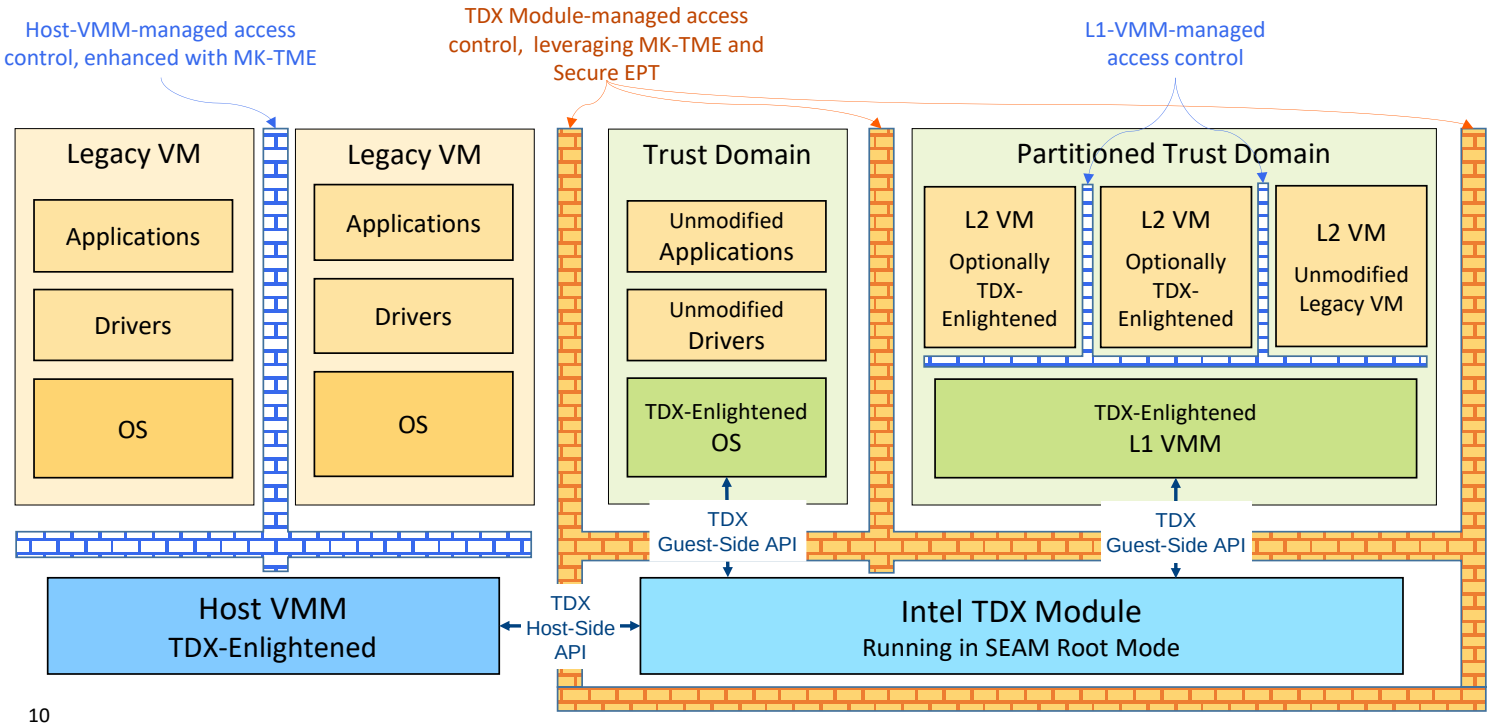

Figure 11.1: TDX Components with TD Partitioning

A TD may have one or more virtual CPUs (VCPUs). With TD partitioning, each VCPU can run in either L1 VMM or in one of the L2 VMs. Transitions between L1 VMM and L2 VMs always occur within the same VCPU.

The diagram below shows an example of how the TD Partitioning architecture can be mapped to Microsoft's VSM architecture.

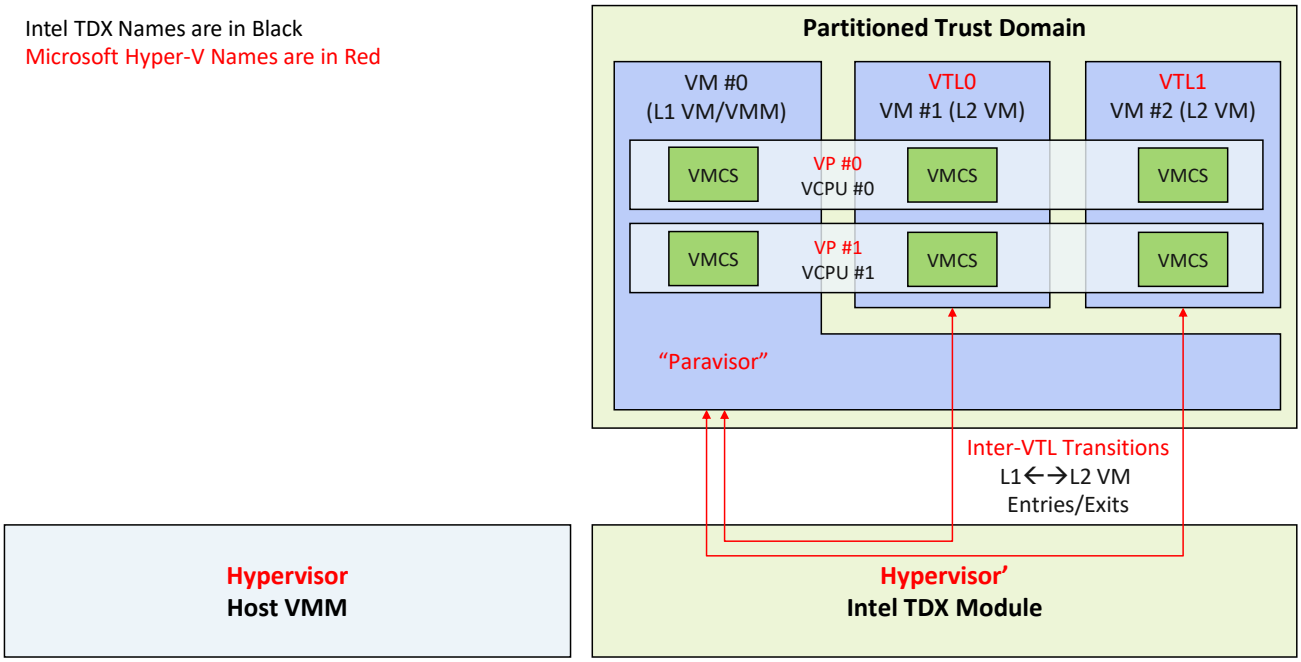

Figure 11.2: Example of TDX TD Partitioning Mapping to Microsoft's VSM Architecture

11.2. Security Properties

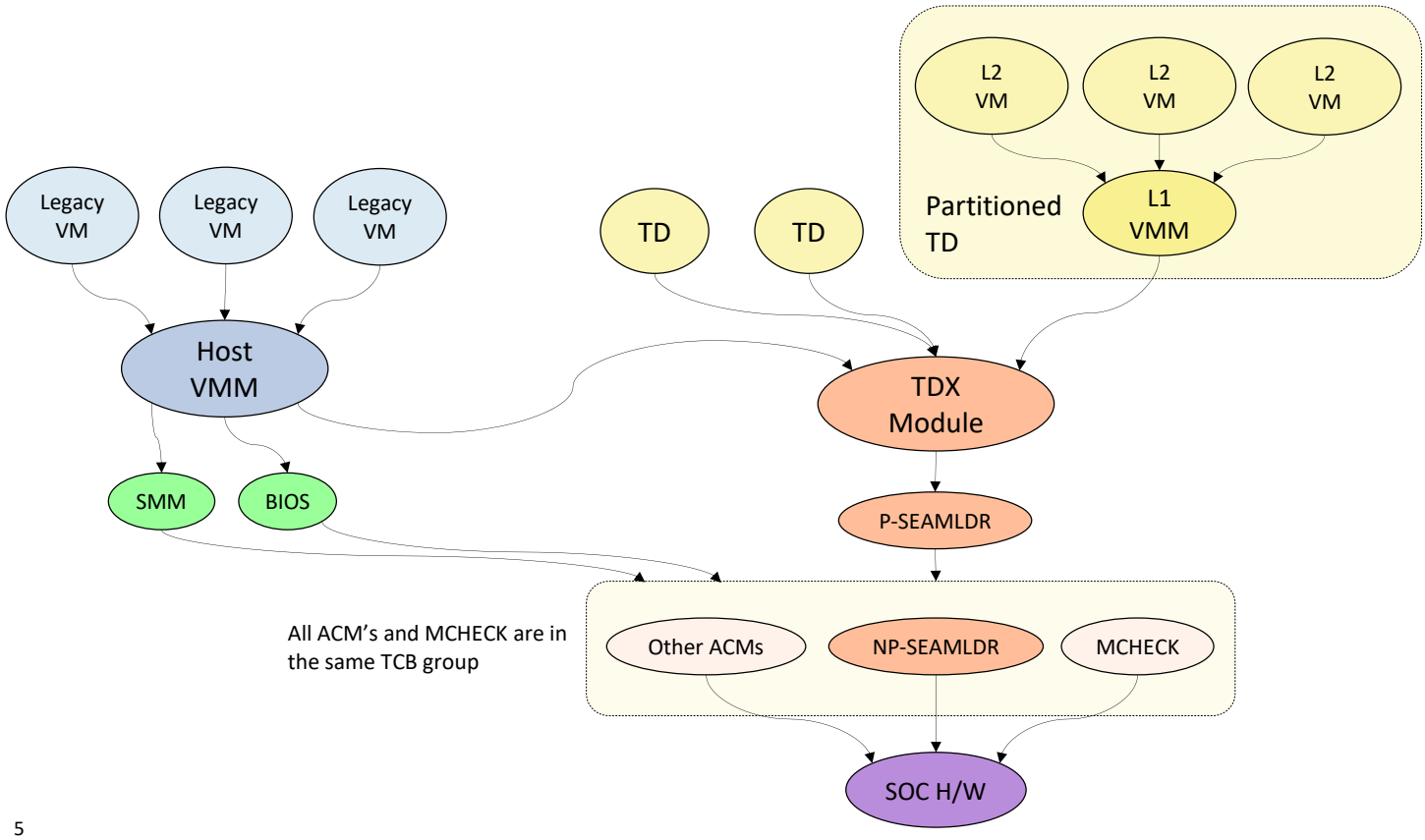

Figure 11.3: TDX TCB Dependency Diagram

Within a TD, the L1 VMM is in the TCB of L2 VMs; the L1 VMM to L2 VM relationship is similar to that of a legacy VMM to legacy VMs. Other than that, TD Partitioning maintains the same security properties of the base TDX architecture.

Note: The L1 VMM is expected to protect itself against untrusted L2 VMs using the mechanisms provided by the TDX module and the underlying CPU's VMX technology. For example, the L1 VMM should properly assign memory access permissions to L2 VMs.

11.3. L2 VM Private Memory Management

TD Partitioning extends the TD private memory model such that each TD private page is managed as a single entity, with a single GPA mapping and state, but a separate set of attributes for each VM within the TD.

Table 11.1: Logical View of a TD Private Page Attributes

Common	L1 VMM	L2 VM #1	L2 VM #2	L2 VM #3
GPA	R=1, W=1, X=1	R, W, Xs, Xu	R, W, Xs, Xu	R, W, Xs, Xu
HPA		SSS, VGP, PWA	SSS, VGP, PWA	SSS, VGP, PWA
State				

- The L1 VMM and all L2 VMs share the same GPA space; however, the access rights and other attributes of pages within this GPA space may be different between each VM.
- All TD private pages are mapped in the L1 VMM GPA space with read, write and execute permissions.
- Individual TD private pages (4KB, 2MB or 1GB) may be mapped by the L1 VMM into the GPA space of an L2 VM. This mapping is called **page alias**. A page alias has the **same GPA** but may have different access permissions and other attributes than the L1 mapping of the same TD private page.

GPA Space	VM #0 L1 VMM	VM #1 L2 VM	VM #2 L2 VM	VM #3 L2 VM
Page A	RWX	R		None
Page B	RWX	R	RWX	R
Page C	RWX	RWX	None	
Page D	RWX	None	RWX	
Page E	RWX	RWX		R
Page F	RWX	RWX		R
Page G	RWX	RW		R
Page H	RWX	RWX		R

Figure 11.4: Example of Page Permissions for L1 VMM and L2 VMs

11.4. L2 VM Transitions

With TD Partitioning, each TD VCPU can run in either L1 VMM or in one of the L2 VMs. Inter-VM transitions always happen within the same VCPU.

- A VCPU running in L1 VMM can initiate an L1→L2 VM entry.
- A VCPU running in L2 VM normally exits back to the to the L1 VMM (L2→L1 VM exit).
- In some cases, TD exit to the host VMM may happen directly from a VCPU running in L2 VM.
- In such cases, the host VMM may re-enter the TD to either resume the exited L2 VM or to resume the L1 VMM.

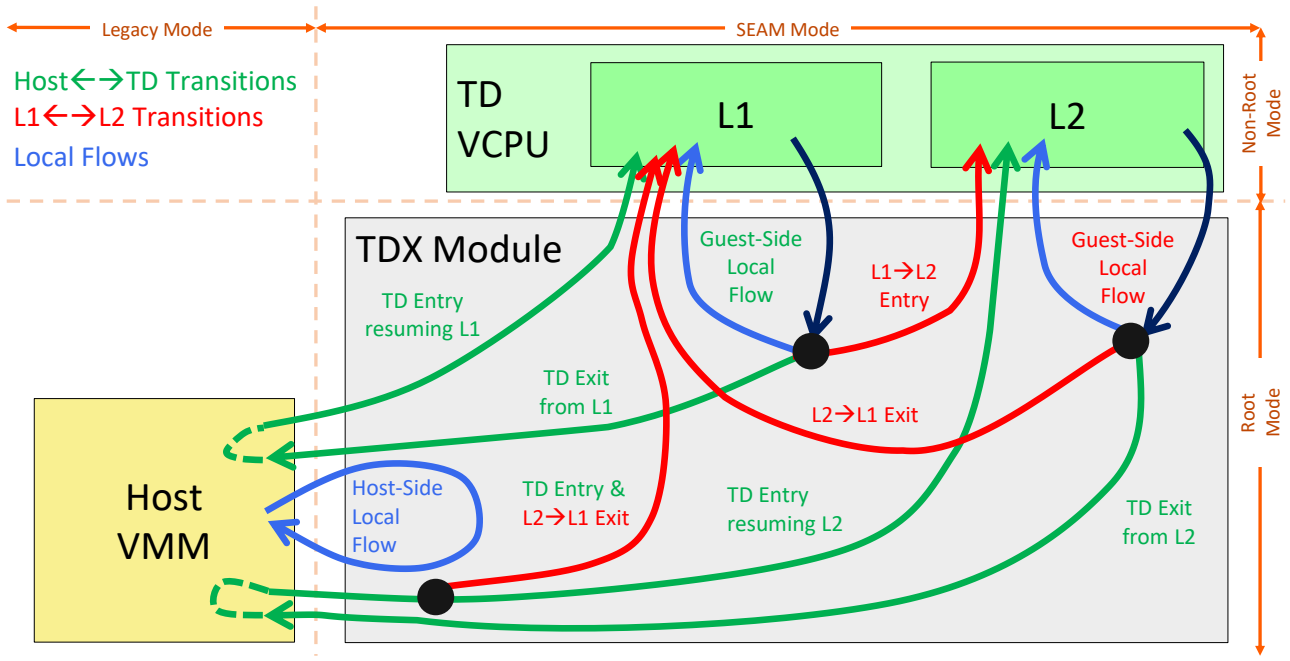

Figure 11.5: TDX Transitions

11.5. L2 VM CPU Virtualization

As a general rule, L2 VM CPU virtualization is controlled by the L1 VMM, within the feature set available to the whole TD and limitation imposed by the TDX architecture. The L1 VMM virtualized CPUID and controls the virtualization of MSRs, thus controlling CPU feature enumeration, for L2 VMs. The L1 VMM configures the L2 VCPUs' VMCS and additional VMX control structures such as MSR exit bitmaps to control L2 VCPU virtualization, within the limitations imposed by the TDX module.

TD Partitioning is designed to support unmodified legacy VMs running as L2 VM. To that end, all CPU modes supported in VMX non-root mode are supported for L2 VMs.

11.6. L2 VM Measurement and Attestation

TD Partitioning does not modify the TDX measurement and attestation architecture. Measurement and attestation of L2 VMs are the responsibility of the L1 VMM.

11.7. L2 VM Debug

The TD Partitioning architecture enhances the off-TD debug facilities to support L2 VMs. In addition, it enables o**n-L2 VM debug** facilities for debugging an L2 VM using software that runs inside that VM, and **L1 VMM debug of L2 VMs**.

11.8. TD Partitioning Interaction with TD Migration

The TD Migration architecture is extended to support live migration of TDs that contains nested L2 VMs.

- The whole TD is always migrated, including any L2 VMs it might have.
- L2 VMs' non-memory state (metadata) is migrated, as well as memory aliasing information.

11.9. Intel TDX TD Partitioning Interface Functions

11.9.1. Host-Side (SEAMCALL Leaf) Interface Functions

Table 11.2: Updated Host-Side (SEAMCALL Leaf) Interface Functions

Class	Interface Function Name	Leaf #	Description
VCPU Scope	TDH.VP.ENTER	0	Enter TDX non-root operation
VCPU Scope	TDH.VP.FLUSH	18	Flush the address translation caches and cached TD VMCS associated with a TD VCPU
Private Memory Management	TDH.MEM.PAGE.DEMOTE	15	Split a 2MB or a 1GB private TD page mapping into 512 4KB or 2MB page mappings respectively
Private Memory Management	TDH.MEM.PAGE.PROMOTE	23	Merge 512 consecutive 4KB or 2MB private TD page mappings into one 2MB or 1GB page mapping respectively
Private Memory Management	TDH.MEM.PAGE.RELOCATE	5	Relocate a 4KB mapped page from its HPA to another
Private Memory Management	TDH.MEM.PAGE.REMOVE	29	Remove a private page from a guest TD
Private Memory Management	TDH.MEM.RANGE.BLOCK	7	Block a TD private GPA range
Private Memory Management	TDH.MEM.RANGE.UNBLOCK	39	Remove the blocking of a TD private GPA range
Private Memory Management	TDH.MEM.SEPT.ADD	3	Add and map a 4KB Secure EPT page to a TD
Private Memory Management	TDH.MEM.SEPT.RD	25	Read a Secure EPT entry
Private Memory Management	TDH.MEM.SEPT.REMOVE	30	Remove a Secure EPT page from a TD

11.9.2. Guest-Side (TDCALL Leaf) Interface Functions

Table 11.3: New and Updated Guest-Side (TDCALL Leaf) Interface Functions

Class	Interface Function Name	Leaf #	Description
VCPU Scope	TDG.VP.ENTER	25	Enter L2 VCPU operation
VCPU Scope	TDG.VP.INVEPT	26	Invalidate cached EPT translations for selected L2 VMs
VCPU Scope	TDG.VP.INVGLA	27	Invalidate cached translations for selected pages in an L2 VM
Private Memory Management	TDG.MEM.PAGE.ACCEPT	6	Accept a pending private page into the TD
Private Memory Management	TDG.MEM.PAGE.ATTR.RD	23	Read the GPA mapping and attributes of a TD private page
Private Memory Management	TDG.MEM.PAGE.ATTR.WR	24	Write the attributes of a private page

SECTION 2:
TD PARTITIONING ARCHITECTURE SPECIFICATION

20. L2 VM Non-Memory State (Metadata) and Control Structures

This chapter discusses the guest TD control structures that hold non-memory state (metadata) and how they are intended to be used during the TD life cycle.

20.1. Overview

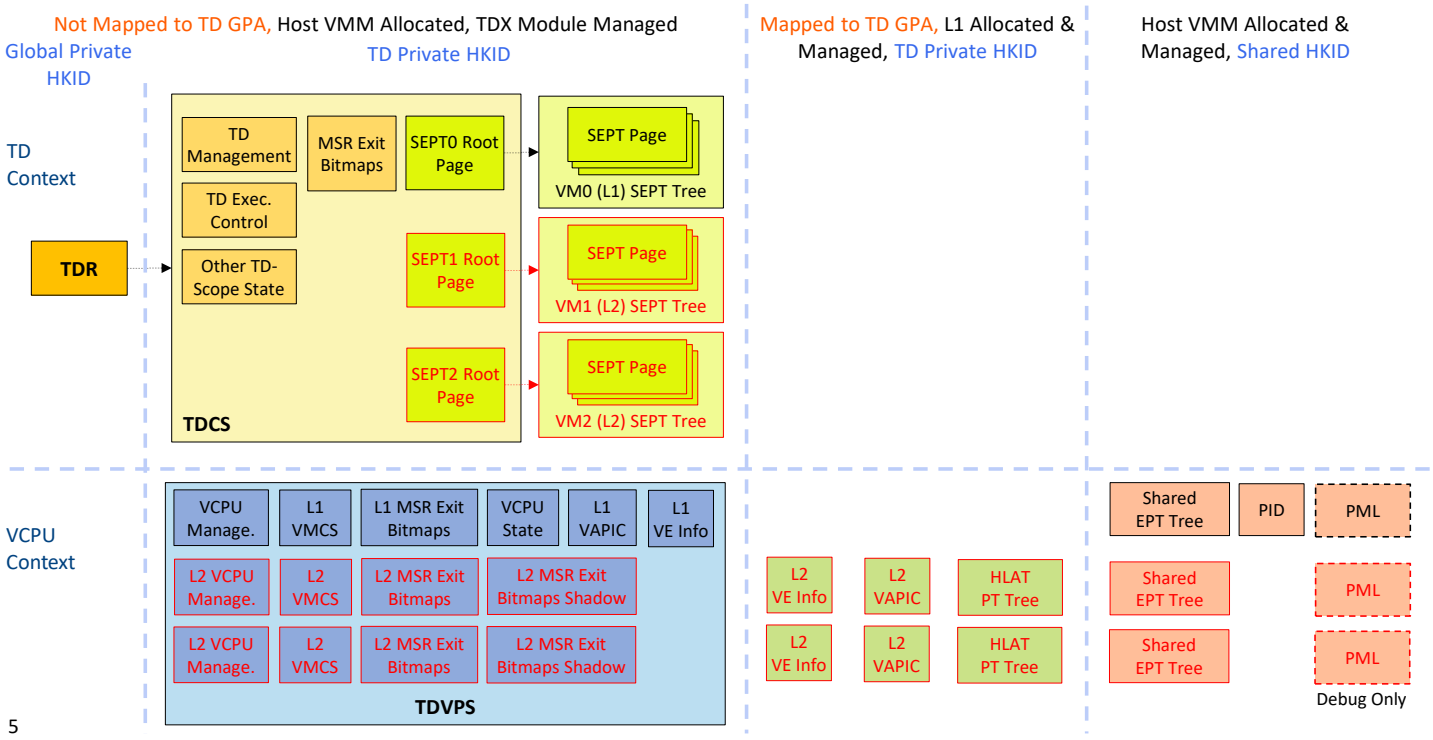

Figure 20.1: Overview of Guest TD Control Structures with TD Partitioning Support

TD Partitioning extends the TD's control structures and adds new ones to control the operation and hold the state of L2 VMs. All guest TD control structures reside in memory pages that are allocated by the host VMM and are addressable by it.

20.1.1. Opaque L2 Control Structures

The following L2 control structure reside in memory pages that are encrypted by the TD's private key, but are not directly accessible to either the L1 VMM or to the host VMM:

L2 SEPT: Secure EPT tree for mapping the L2 VM's GPA space.

20.1.2. Private Control Structures

Multiple standard VMX control structures that are used by the L1 VMM and its L2 VM guests are mapped in the TD's GPA space:

- Virtual APIC page
- VE Info page
- HLAT page table tree

The L1 VMM may map specific control structure pages (e.g., VE Info) in the L2 VM's GPA space by creating page aliases.

20.1.3. Shared Control Structures

Standard VMX control structures that are to be accessible by the host VMM can reside in shared memory:

- Shared EPT
- Page Modification Log (PML)

20.2. Additions to Existing TD Control Structures

20.2.1. TDCS

TDCS holds a set of L2 VM related fields. For each of the L2 VMs, the following fields are maintained:

Table 20.1: TDCS L2 VM Related Fields High Level Definition

Field	Description	Reference
NUM_L2VMS	Number of L2 VMs in this TD	
L2_SEPT_ROOT	Per-L2 VM Secure EPT root page	

For a detailed definition of TDCS, please see the [TDX Module ABI Spec].

20.2.2. TDVPS

20.2.2.1. *New TDVPS Fields*

TDVPS is extended with the following per-VCPU fields:

Table 20.2: New TDVPS Per-VCPU Fields High Level Definition

Field	Description	Reference
CURR_VM	VM index currently used for this VCPU	22.2
ACTIVE_VMCS	VM index of the active VMCS (last VMCS target of VMPTRLD). If there's no active VMCS or the last VMPTRLD was not of one of the guest VMCSes, value is -1.	

20.2.2.2. *New Per-VM TDVPS Fields*

TDVPS is extended with the following fields, per VM:

L2 VM Management Fields

Table 20.3: TDVPS Per-VM (L1 and L2) Management Fields High Level Definition

Field	Description	Reference
VM_LAUNCHED	A Boolean flag, indicating whether the VM (i.e., it's VMCS) has been VMLAUNCH'ed on this LP since it has last been associated with this VCPU. If TRUE, VM entry should use VMRESUME. Else, VM entry should use VMLAUNCH.	
LP_DEPENDENT_HPA_UPDATED	Flags that the LP-dependent HPA fields have been updated. Cleared after new VCPU-to-LP association.	

Table 20.4: TDVPS Per-VM (L2 Only) Management Fields High Level Definition

Field	Description	Reference
L2_ENTER_GUEST_STATE_GPA	GPA of the TDG.VP.ENTER guest state buffer	
L2_ENTER_GUEST_STATE_HPA	HPA of the TDG.VP.ENTER guest stats buffer	
L2_VE_INFO_GPA	GPA of the L2 VE_INFO area	
L2_VE_INFO_HPA	HPA of the L2 VE_INFO area	
L2_VAPIC_PAGE_GPA	GPA of the L2 virtual APIC page (used by the L1 VMM)	

Field	Description	Reference
L2_VAPIC_PAGE_HPA	HPA of the L2 virtual APIC page (used by the L1 VMM)	

L2 VM Execution Control Fields

Table 20.5: TDVPS Per-VM Execution Control Fields High Level Definition

Field	Description	Reference
L2_CTLS.ENABLE_SHARED_EPTP	L2 VCPU policy for using shared memory, set by the L1 VMM	21.7
L2_CTLS.ENABLE_TDVMCALL	Allow the L2 VM to use TDG.VP.VMCALL	22.2.3
SHADOW_MSR_BITMAPS	Shadow MSR exit bitmaps page, defining the L2 VM policy for handling MSR access, set by the L1 VMM	23.8
SHADOW_PROCBASED_EXEC_CTRL2	Shadow of the L2 VMCS "secondary processor-based execution controls". Holds the L2 policy as set by the L1 VMM for the following bits: • Bit 30: Bus-lock detection • Bit 31: Notification exiting	23.12
SHADOW_NOTIFY_WINDOW	Shadow of the L2 VMCS "notify window" execution control. Holds the L2 policy as set by the L1 VMM.	23.12
TSC_DEADLINE	L2 VCPU execution deadline, set by the L1 VMM	23.13

L2 VM VMX Standard Control Structures

Table 20.6: TDVPS Per-VM VMX Standard Control Structures High Level Definition

Field	Description	Reference
VMCS	VMCS used when the VCPU runs in this VM	
MSR_BITMAPS	Exit bitmaps page used by the CPU	

For a detailed definition of TDVPS, please see the [TDX Module ABI Spec].

20.3. Concurrency Restrictions and Enforcement

Currently, TD partitioning adds no additional concurrency restrictions to the TD's non-memory state.

21. L2 VM Memory Management

This chapter described how the Intel TDX module helps manage TD private memory and guest-physical address (GPA) translation.

21.1. Introduction

Each VM in a TD uses its own SEPT tree to translate GPA to HPA. Every TD private page must be mapped in the L1 SEPT tree; in addition, it may be mapped in one or more L2 SEPT trees. The L2 SEPT mapping creates an **alias**.

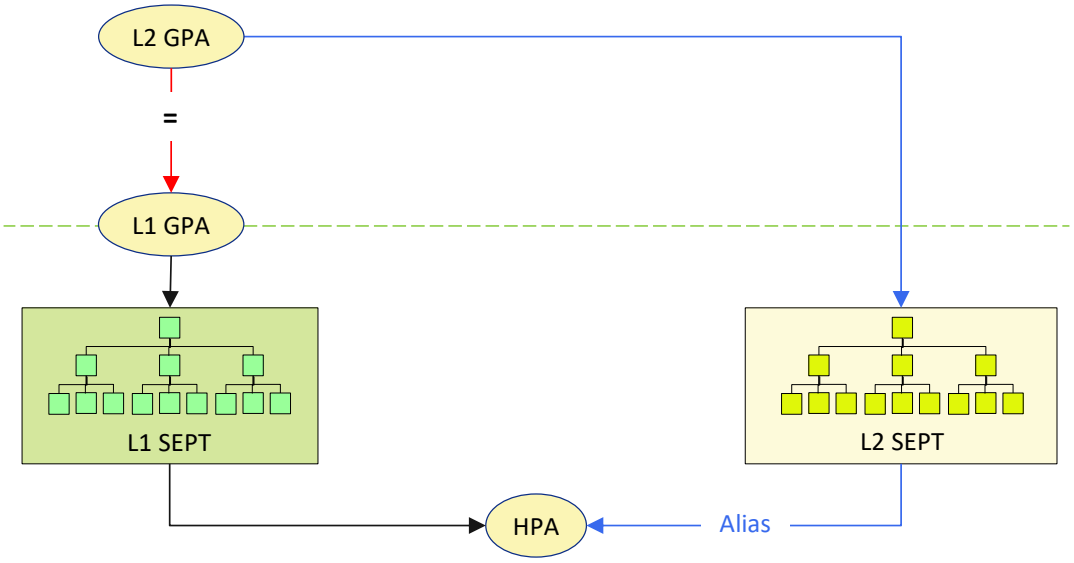

Figure 21.1: GPA→HPA Translation for L1 and L2 VM

The table below summarizes page aliasing features.

Table 21.1: Page Aliasing

Item	Page Aliasing
Mechanism	TD private page is mapped in both L1 SEPT as a leaf entry and L2 SEPT as a leaf entry.
SEPT Leaf Entry	L1 VMM and L2 VM see different SEPT leaf entries.
Mapping Type and Size	Leaf: 4KB, 2MB, 1GB
Mapping Constraints	L2 GPA == L1 GPA L2 mapping size == L1 mapping size
Page Access Permissions	Page access permissions for L2 may be different than its access permissions for L1.
Page Attributes: CET SSS, HLAT, Suppress VE	Page attributes for L2 may be different than its attributes for L1.
Management	Page aliases are managed by the L1 VMM. The host VMM is not directly involved.

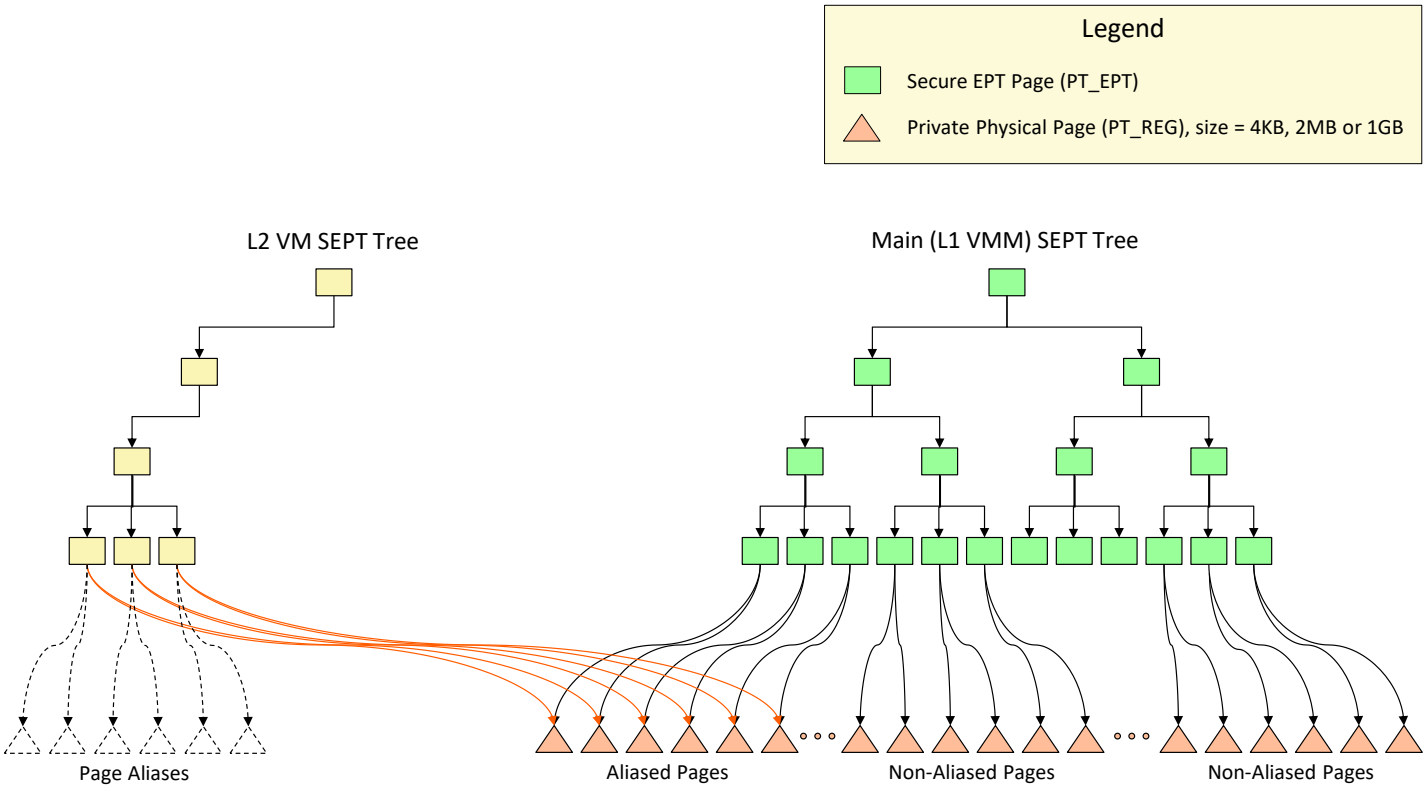

Figure 21.2: Page Aliasing

L2 SEPT Properties

- L2 SEPT trees have the same number of levels (4 or 5) as the L1 SEPT.
- Each non-free L2 SEPT entry has an applicable (same GPA and level) L1 SEPT entry.

21.2. L2 Page Aliasing

21.2.1. Logical View of a TD Private Page with Pages Aliases

Logically, L2 page aliases extend a private TD page attributes to include the attributes as seen by each L2 VM.

Table 21.2: Logical Attributes of a TD Private Page with Page Aliases

Common	GPA			
	HPA			
	Main SEPT entry state			
Per VM	**L1 VMM**	**L2 VM #1**	**L2 VM #2**	**L2 VM #3**
		L2 SEPT entry state	L2 SEPT entry state	L2 SEPT entry state
	R, W, Xs, Xu	R, W, Xs, Xu	R, W, Xs, Xu	R, W, Xs, Xu
		SSS, VGP, PWA	SSS, VGP, PWA	SSS, VGP, PWA
	SVE	SVE	SVE	SVE

The main SEPT entry state, used for memory management and TD migration, is held by the L1 SEPT entry. Memory management interface functions process the L1 SEPT entry and its L2 alias SEPT entries together.

21.2.2. L2 Secure EPT Entry Partial State Diagram

The L2 SEPT entry state machine is relatively simple since the overall page state management is done using the L1 SEPT entry for the page.

The diagram below is a partial state diagram which covers memory management operations for a page alias, not including TD migration. TD migration does not add new states; for details, see [TD Migration Spec].

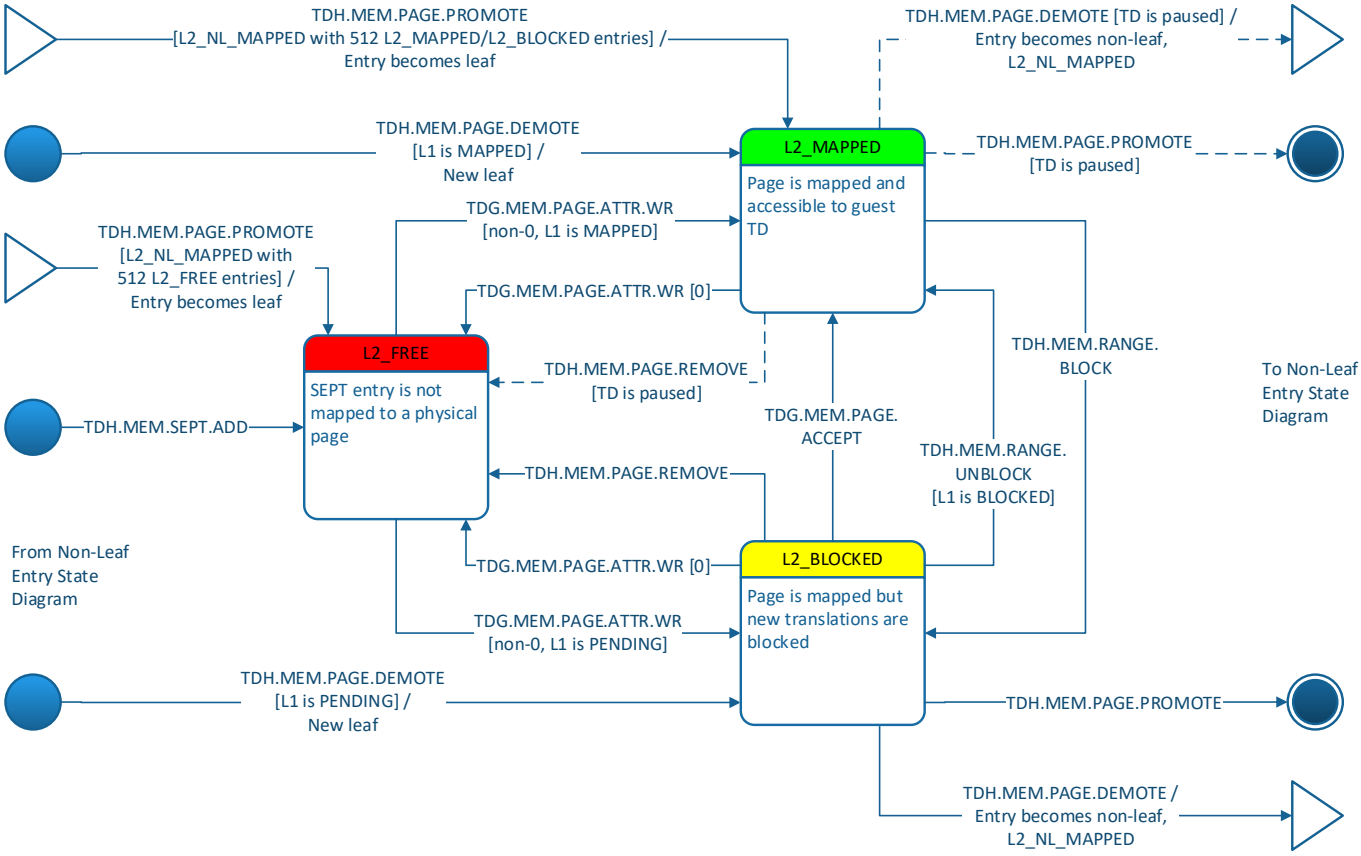

Figure 21.3: L2 Secure EPT Leaf Entry Partial State Diagram

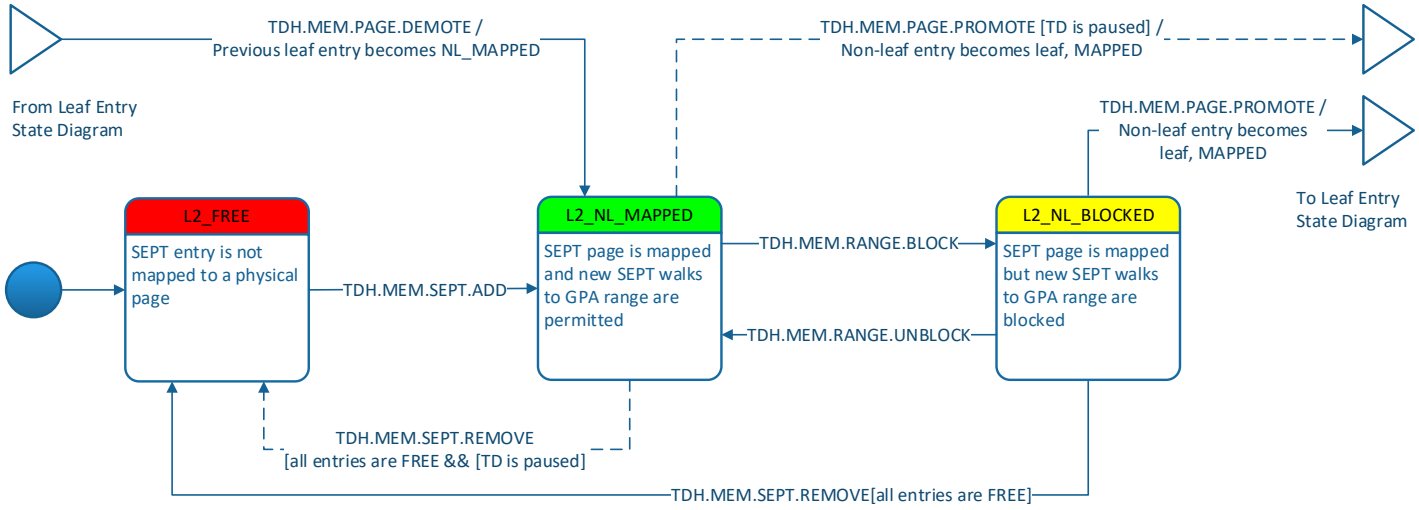

Figure 21.4: L2 Secure EPT Non-Leaf Entry Basic Operation Partial State Diagram

21.2.3. L2 Page Alias Management: TDG.MEM.PAGE.ATTR.RD/WR

21.2.3.1. *Overview*

A TD private page is managed by the L1 VMM as a single entity, including its L1 state and attributes and any L2 page alias state and attributes. L1 VMM operates on page attributes using the GPA mapping size as set by the host VMM (i.e., 4KB,

2MB or 1GB). This requires the L1 VMM to understand the concept of mapping size and to cooperate with the host VMM if there's a need to control page attributes with a different granularity than currently configured.[1]

A single interface function TDG.MEM.PAGE.ATTR.WR enables the L1 VMM to create L2 page aliases, modify their attributes and remove them. The host VMM is not directly involved, except when there's an EPT violation due to the need to add L2 SEPT pages.

L2 aliases may be added, modified and removed from pending pages. When the L1 VMM accepts a pending page (TDG.MEM.PAGE.ACCEPT) it also accepts its L2 page aliases.

TDG.MEM.PAGE.ATTR.RD enables the L1 VMM to read L2 page attributes.

21.2.3.2. *Adding L2 Page Aliases*

The L1 VMM adds new L2 page aliases and sets its attributes using TDG.MEM.PAGE.ATTR.WR with a set of attributes whose R, W, Xs or Xu bits are not all-0. The host VMM is not directly involved. TDG.MEM.PAGE.ATTR.WR may encounter an EPT violation; in this case the TDX module TD-exits to the host VMM to indicate the need to add SEPT pages. This is shown in the picture below.

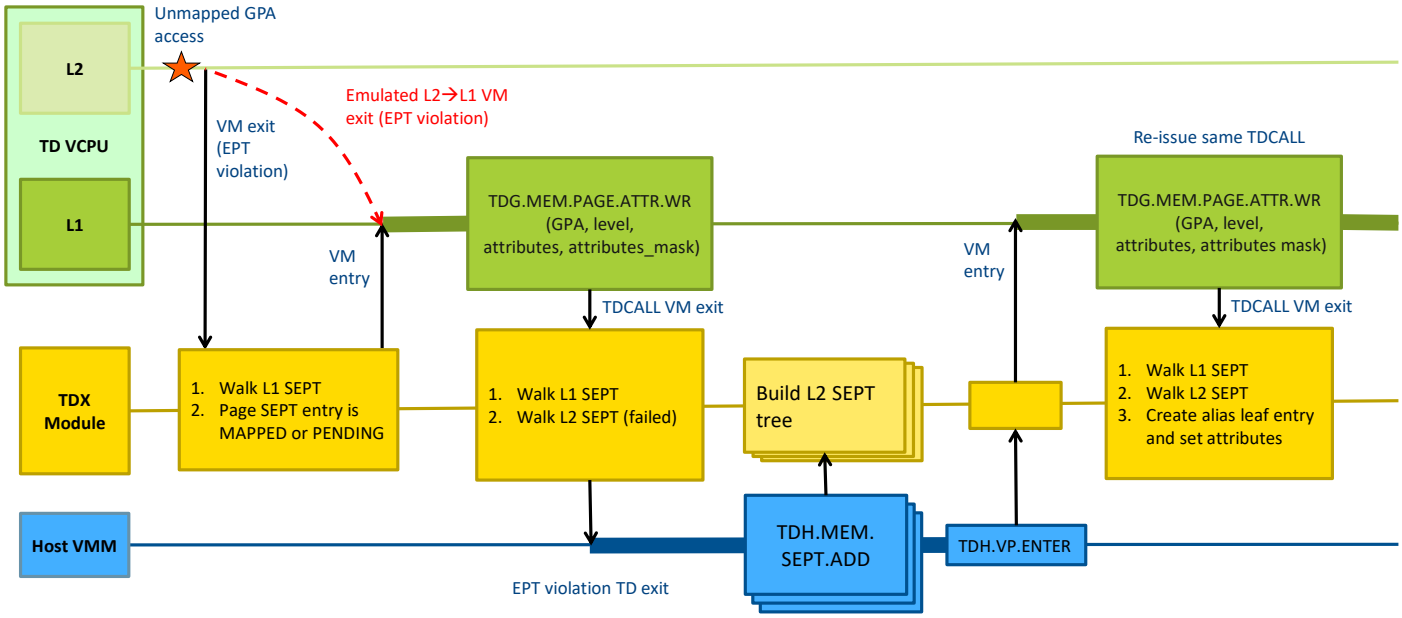

Figure 21.5: Example of Adding an L2 Page Alias where Secure EPT Build is Required

The L1 VMM may add L2 page aliases to a PENDING page. In this case, the page alias is created in an L2_BLOCKED state. When the L1 VMM accepts the page using TDG.MEM.PAGE.ACCEPT, it also accepts any page alias it has; the L2 page alias state becomes L2_MAPPED and its attributes, set by TDG.MEM.PAGE.ATTR.WR, become effective.

The L1 VMM specifies a page size and may fail if the specified size is different than the actual SEPT mapping size.

- If the page is mapped at a lower SEPT level than requested, TDG.MEM.PAGE.ATTR.WR returns an error code and the actual mapping size. The L1 VMM may re-invoke TDG.MEM.PAGE.ATTR.WR specifying the actual mapping size.
- If the page is mapped at a higher SEPT level than requested, this results in an EPT violation TD exit, with information about the guest-requested mapping level. The host VMM typically demotes the page, then re-enter the guest TD so TDG.MEM.PAGE.ATTR.WR is re-invoked.

Typical Use Case

Adding page aliases may be done dynamically, based on an EPT violation L2→L1 VM exit.

1. The L1 VMM invokes TDG.MEM.PAGE.ATTR.WR to set non-0 L2 page attributes to a given GPA.
2. The L2 SEPT tree is missing an EPT page to map the new alias. The TDX module performs a TD exit, indicating an EPT violation with an extended exit qualification providing the L2 VM index and other details.
3. The host VMM invokes TDH.MEM.SEPT.ADD to build the L2 SEPT tree.
4. The host VMM invokes TDH.VP.ENTER to resume the TD.

[1] This is not a new concept in TDX. It already exists for TDG.MEM.PAGE.ACCEPT

5. The guest state, including RIP, hasn't been modified. Thus, TDG.MEM.PAGE.ATTR.WR is executed again. This time the operation succeeds.

21.2.3.3. *Modifying L2 Page Attributes*

The L1 VMM can modify an existing L2 page alias using TDG.MEM.PAGE.ATTR.WR with a set of attributes whose R, W, Xs or Xu bits are not all-0. The L1 VMM is responsible for TLB shootdown of its L2 VM if this is required, e.g., if page access permissions are reduced. TDG.MEM.PAGE.ATTR.WR, TDG.VP.INVEPT and TDG.VP.ENTER allow the L1 VMM to invalidate cached SEPT translations after modifying page attributes.

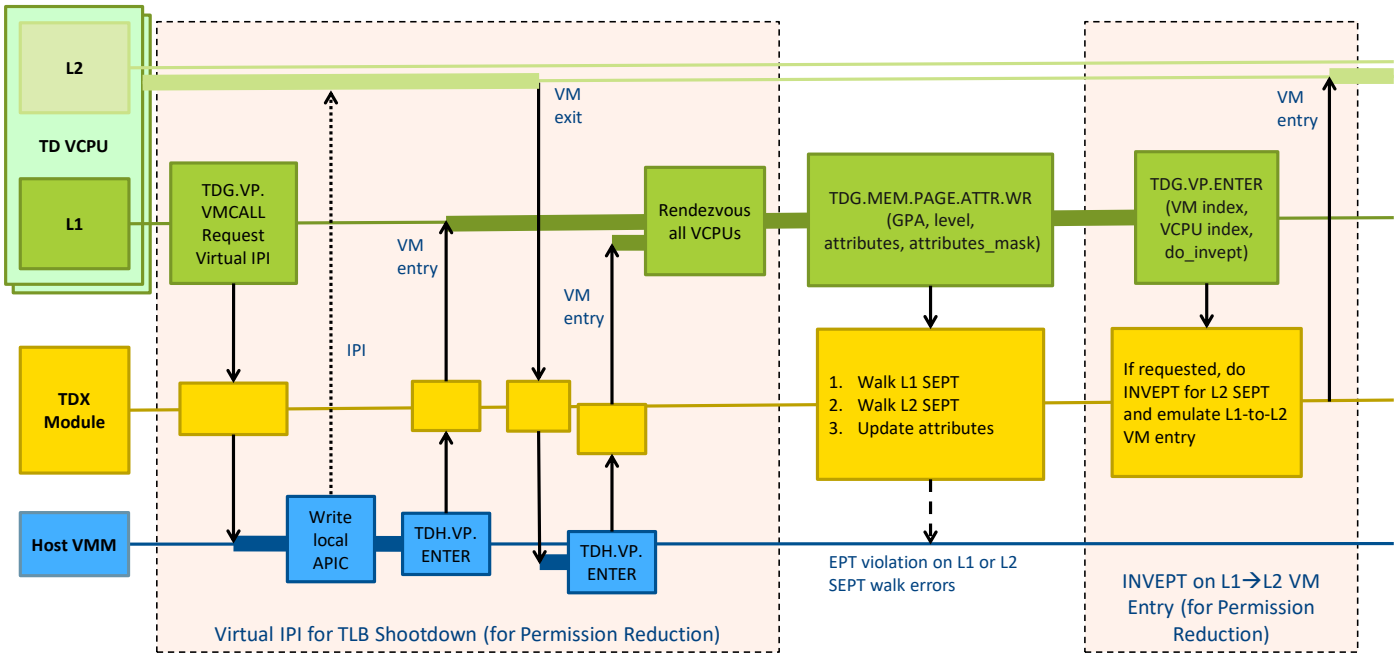

Figure 21.6: Example of L1 VMM Modifying a Page Alias Attributes

The L1 VMM can update the page alias attributes of a PENDING page.

Typical Use Case

1. The L1 VMM rendezvous all VCPUs:
   1.1. To do that, the L1 VMM requests the host VMM to issue IPIs to all the LPs running the current TD (the L1 VMM doesn't directly know if a VCPU is running in L1 or L2 on each LP).
   1.2. Each VCPU running in L2 that receives the IPI TD-exits to the host VMM.
   1.3. The host VMM resumes the L1 VMM by invoking TDH.VP.ENTER.
2. The L1 VMM invokes TDG.MEM.PAGE.ATTR.WR to update the L2 page alias attributes.
3. The L1 VMM resumes L2 by invoking TDG.VP.ENTER, indicating that EPT invalidation should be done before each VM entry.

21.2.3.4. *Removing L2 Page Aliases*

The L1 VMM removes existing L2 page aliases using TDG.MEM.PAGE.ATTR.WR with a set of attributes whose R, W, Xs and Xu bits are all-0. The L1 VMM is responsible for TLB shootdown of its L2 VM. TDG.MEM.PAGE.ATTR.WR, TDG.VP.INVEPT and TDG.VP.ENTER allow the guest TD to invalidate cached SEPT translations.

Note: Since the TD private page is still owned by the TD, there is no need for the host VMM to be involved.

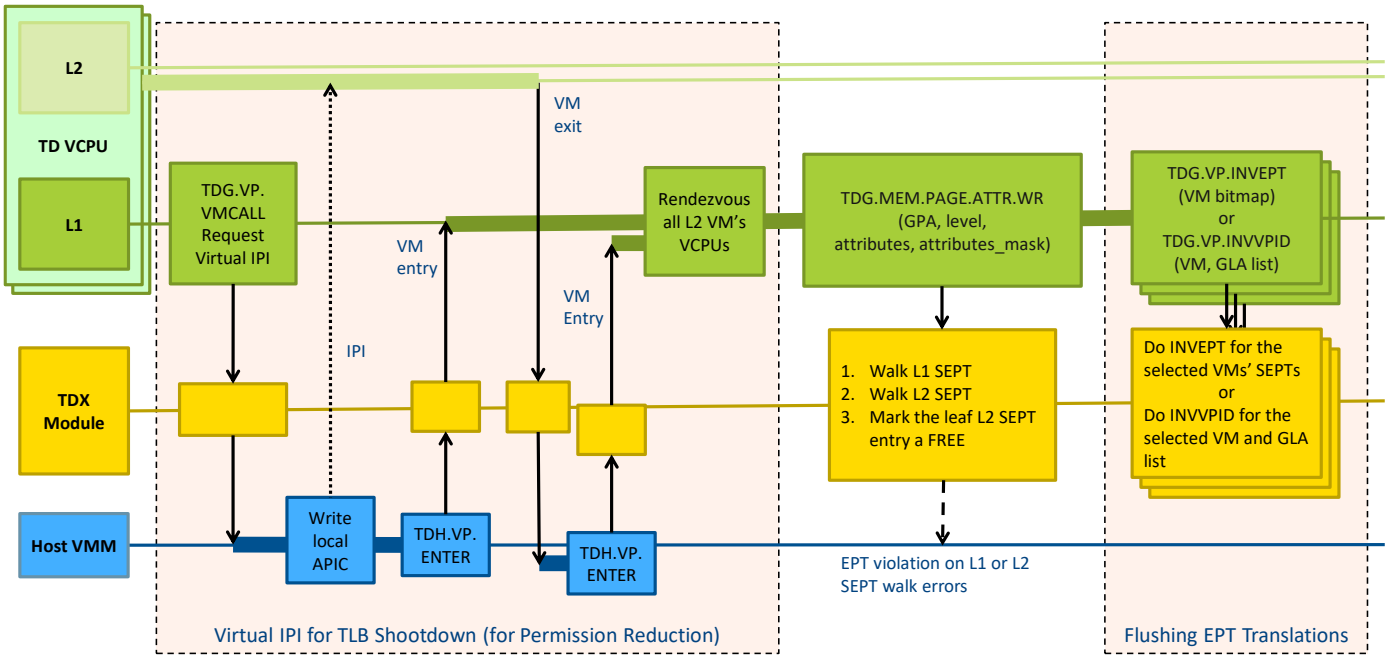

Figure 21.7: Example of L1 VMM Removing a Page Alias

Typical Use Case

1. The L1 VMM rendezvous all VCPUs
   1.1. To do that, the L1 VMM requests the host VMM to issue IPIs to all the LPs running the current TD (the L1 VMM doesn't directly know if a VCPU is running in L1 or L2 on each LP).
   1.2. Each VCPU running in L2 that receives the IPI TD-exits to the host VMM.
   1.3. The host VMM resumes the L1 VMM by invoking TDH.VP.ENTER.
2. The L1 VMM invokes TDG.MEM.PAGE.ATTR.WR to remove the page alias.
3. The L1 VMM invokes TDG.VP.INVEPT or TDG.VP.INVGLA on each VCPU to invalidate EPT for the whole VM or for specific pages.

21.3. Updates to SEPT Tree Management

Secure EPT tree management interface functions are extended to support coordinated management of the L1 and multiple L2 SEPT trees, as described in the following sections.

21.3.1. Host VMM's L2 SEPT Management Strategy

The host VMM may deploy various strategies for populating the L2 SETP tree(2):

Dense L2 SEPT Tree: The host VMM may choose to populate the L2 SEPT tree(s) as densely as the L1 SEPT tree is populated. To do so, it allocates an L2 SEPT page in each L2 SEPT tree for each L1 SEPT page.

Sparse L2 SEPT Tree: The host VMM may choose to populate L2 SEPT pages on demand, i.e., when the L1 VMM creates a page alias and there's a need an L2 SEPT page to map it (indicated by an EPT violation TD exit or by an explicit request from the L1 VMM using TDG.VP.VMCALL).

The host VMM may also choose to maintain L2 SEPT trees only for a subset of the L2 VMs (e.g., if a TD is created with a certain number of L2 VMs, but not all of them are currently in use).

21.3.2. Adding SEPT Pages

TDH.MEM.SEPT.ADD is extended to support adding of L2 SEPT pages:

- The host VMM may add SEPT pages to one or more of the TD's VMs in a single TDH.MEM.SEPT.ADD invocation.
- To add L2 SEPT pages at a certain level, an L1 SEPT page must either already exist at that level or be added during the same TDH.MEM.SEPT.ADD invocation.
- The operation is atomic, in the sense that either all requested SEPT pages are added successfully, or none is added. An exception to this rule is in the case of interrupted operation; see the [TDX Module ABI Spec] for details.

21.3.3. Removing SEPT Pages

TDH.MEM.SEPT.REMOVE is extended to support removal of L2 SEPT pages. When removing an L1 SEPT page at the specified GPA and level, all associated L2 SEPT pages at the same GPA and level are removed.

21.3.4. Page Demotion

TDH.MEM.PAGE.DEMOTE is extended to support adding of L2 SEPT pages, which map the demoted 512 smaller-sized pages. The host VMM may choose whether to add an L2 SEPT page only if there's a page alias for the relevant L2 VM (to support the dense L2 SEPT management policy mentioned above).

21.3.5. Page Promotion

TDH.MEM.PAGE.PROMOTE is extended to remove any L2 SEPT pages which map the promoted 512 smaller-sized pages.

21.4. Other Updates to Memory Management Interface Functions

Existing memory management interface functions are modified to support page aliasing as follows:

Table 21.3: Updates to Memory Management Functions to Support L2 Page Aliasing

Memory Management Function	Description
TDH.MEM.RANGE.BLOCK (of a leaf SEPT entry)	1. New: Block all aliases (sets L2 SEPT leaf entries to L2_BLOCKED) 2. Block L1 mapping: Set L1 SEPT leaf entry to *BLOCKED 3. Record TD_EPOCH in PAMT.BEPOCH
TDH.MEM.PAGE.REMOVE	1. New: Remove all aliases (sets L2 SEPT leaf entries to FREE) 2. Set L1 SEPT leaf entry to FREE 3. Mark PAMT as free
TDH.MEM.PAGE.PROMOTE	1. Check that all 512 small page L1 SEPT entries have the same attributes and contiguous HPA mappings. 2. New: For each L2 SEPT tree, check that either all 512 small page aliases have the same attributes, or none of them exists. 3. Promote L1 SEPT mapping. 4. New: Promote applicable L2 SEPT mappings.
TDH.MEM.PAGE.DEMOTE	1. Demote L1 SEPT mapping. 2. New: For each page alias, demote L2 SEPT mapping. Additional SEPT pages for L2 SEPT are provided by the host VMM as inputs.
TDH.MEM.PAGE.RELOCATE	1. Relocate the physical page. 2. Updates HPA of L1 SEPT entry. 3. New: For each page alias (L2 SEPT entry), update HPA.
TDG.MEM.PAGE.ACCEPT	Extended to accept any aliases that exist to the page.

21.5. L1 VMM Control of L2 EPT Features

The L1 VMM can enable or disable L2 EPT features by writing to the L2 VMCS (e.g., using TDG.VP.WR). Note that these setting impact both secure EPT and shared EPT (if enabled).

Table 21.4: L1 VMM Control of L2 VM EPT Features

SEPT Feature	EPT Bits		L2 VMCS		
	#	Name	Field Name	Bit #	Bit name
CET	60	Supervisor shadow stack (SSS)		7	Enable SSS

SEPT Feature	EPT Bits		L2 VMCS		
	#	Name	Field Name	Bit #	Bit name
Mode-Based Execution Control (MBEC)	10 2	Execute in user mode (Xu) Execute in supervisor mode (Xs)	Secondary processor-based VM-execution controls	22	Mode-based execution control for EPT
HLAT	57	Verify guest paging (VGP)	Tertiary processor-based VM-execution controls	3	Guest-paging verification
	58	Paging-Write Access access(PWA)		2	EPT paging-write control

21.6. L2 VM TLB Invalidation

Two interface functions are provided for L2 TLB invalidation initiated by the L1 VMM:

- TDH.VP.INVEPT allows the L1 VM to request invalidation of all TLB entries that belong the selected L2 VMs.
- TDG.VP.INVGLA allows the L1 VMM to request invalidation TLB of a list of page guest linear addresses that belong to a specific L2 VM.

For details, see the [ABI Spec].

21.7. L2 VM Shared Memory Management

Shared GPA (where SHARED bit is 1) mapping is controlled by the host VMM:

- The host VMM may or may not use the same shared EPT tree for the L1 VMM and for one or more of the L2 VMs.
- The host VMM should write each L2 VMCS “shared EPTP” field with the HPA of the applicable shared EPT root page.

To provide defense-in-depth for unmodified legacy VMs running as L2 VMs, the L1 VMM can enable or disable shared memory access using the TDVPS.L2_CTLS[VM].ENABLE_SHARED_EPTP field.

21.8. Handling EPT Violation VM Exit from L2

An EPT violation L2 VM exit can be the result of multiple reasons and should be handled by either the L1 VMM or the host VMM.

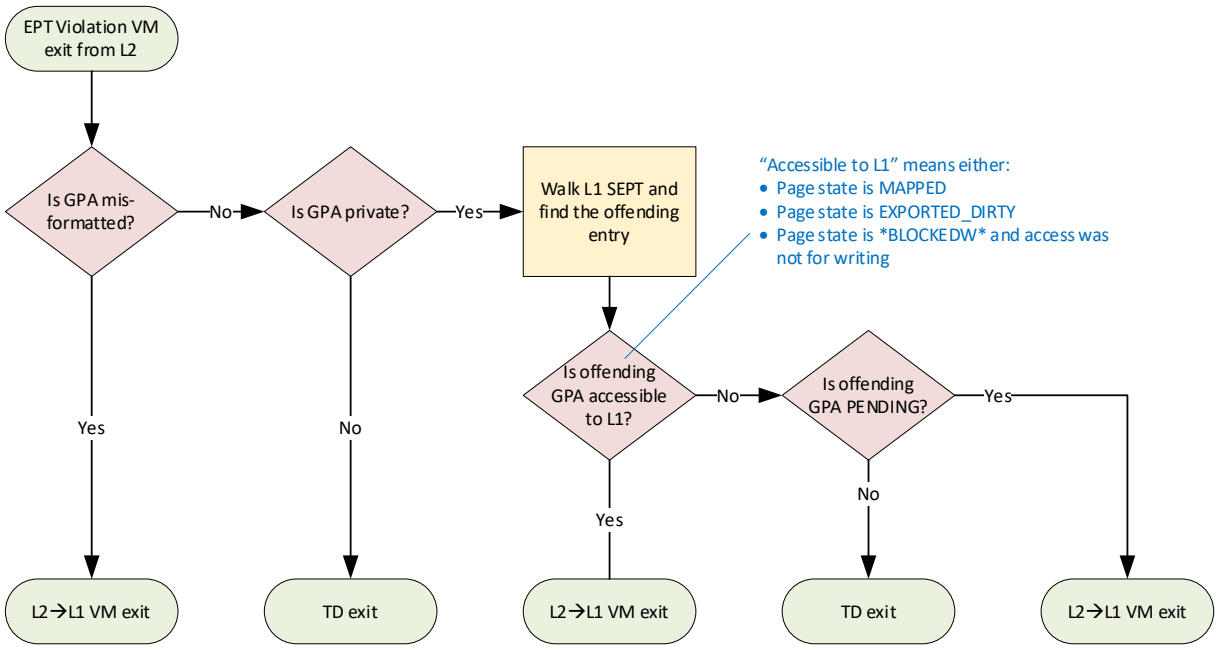

Figure 21.8: Handling EPT Violation VM Exit from L2

The TDX module handles such EPT violations as follows:

1. If the offending GPA is mis-formatted, i.e., GPA bit at position higher than GPAW but lower than MAXPA is set to 1:
   1.1. TDX module completes an L2→L1 VM exit.
   1.2. L1 VMM is expected to check the GPA vs. the virtual MAXPA value it enumerated to the L2 VM and inject a #PF(PFEC.RSVD=1) to the L2 VCPU.
2. Else, if the GPA is private:
   2.1. The TDX module walks the **L1** SEPT tree to find the SEPT entry for the offending GPA.
   2.2. If the offending GPA is not accessible to L1 (i.e., to the TD as a whole TD), i.e., one of the following cases is true:
      2.2.1. Non-leaf L1 SEPT entry state for a containing GPA range is NL_BLOCKED.
      2.2.2. There is no leaf L1 SEPT entry for the page.
      2.2.3. Leaf L1 SEPT entry state is FREE or REMOVED.
      2.2.4. The page has been blocked by the host VMM (L1 SEPT entry state is *BLOCKED).
      2.2.5. The page has been blocked-for-writing by the host VMM (L1 SEPT entry state is *BLOCKEDW*), and access attempt was a write.
      The TDX module complete an asynchronous TD exit, indicating an EPT violation exit from L2.
   2.3. Else, if the offending GPA if of a PENDING or PENDING_EXPORTED_DIRTY:
      2.3.1. The TDX module completes an L2→L1 VM exit.
   2.4. Else, the EPT violation's cause is under L1 VMM control, e.g.:
      2.4.1. The offending GPA has not been added as an alias.
      2.4.2. The alias attributes don't match the attempted memory access.
      In all the above cases, the TDX module completes an L2→L1 VM exit. The L1 VMM is expected to handle the EPT violation.
3. Else (the GPA is shared):
   3.1. Shared GPAs are managed by the host VMM. The TD module complete an asynchronous TD exit, indicating an EPT violation exit from an L2 VCPU.

To handle L2 VM EPT violations, the L1 VMM can query the alias status using TDG.MEM.PAGE.ATTR.RD.

21.9. Handling EPT Misconfiguration VM Exit from L2

This is similar to EPT misconfiguration VM exit from L1, as described in the [TDX Module Base Spec]. An EPT misconfiguration on a private GPA indicates a TDX module bug and is handled as a fatal error. EPT misconfiguration on a shared GPA causes a TD exit and is handled by the host VMM.

21.10. L2 GPA-to-HPA Soft Translation

The GPAs of some structures in TD private memory are specified by the guest TD (using TDG.VP.WR and TDG.VP.ENTER). Such addresses are translated by the TDX module to shadow HPA values and stored in internal variables. Using the TLB tracking mechanism, the TDX model is designed to ensure that the shadow HPA values correspond to the correct private GPAs whenever the CPU or the TDX module itself may use them.

Table 21.5: GPAs Specified by the Guest TD

Address	GPA Stored in	HPA Stored in	Shadow HPA Stored in	Field Name	Applicable When	GPA Mapping in L2
TDG.VP.ENTER output memory operands	TDVPS	N/A	TDVPS	Multiple	VCPU is entering to L2 VCPU is exiting from L2	None
L2 Virtual APIC page	TDVPS (shadow)	L2 VMCS	TDVPS	virtual-APIC address	VCPU is running in L2	None

Notes

- GPA mapping in L2 is not enforced by the TDX module. The TDX module helps ensure that the page is mapped in L1; it is up to the L1 VMM to ensure L2 accessibility for correct L2 VM operation.

22. TD VCPU Enhancements for TD Partitioning

This chapter discusses multiple items related how TD VCPUs are enhanced to support TD Partitioning.

22.1. Overview

Each TD VCPU is a single logical entity with multiple modes of operation, one for each VM. From the CPU's perspective, each of the VCPU's VM modes is controlled by its own VMCS and associated control structures.

22.2. VCPU Transitions

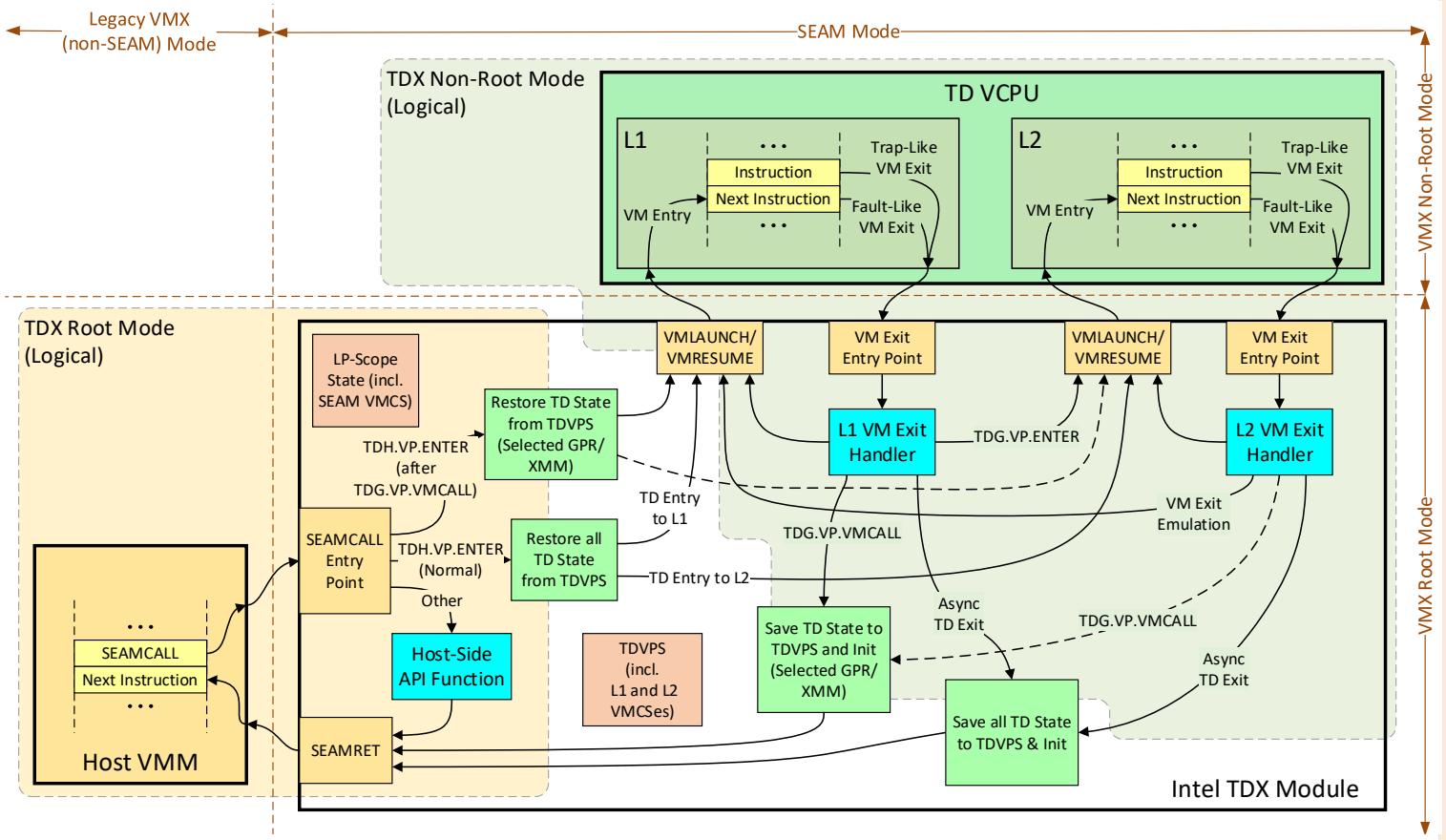

Figure 22.1: TD VCPU Transitions Overview

22.2.1. L1-to-L2 VM Entry and L2-to-L1 VM Exit: TDG.VP.ENTER

The L1 VMM can enter an L2 VCPU by invoking TDG.VP.ENTER, if its state is READY.

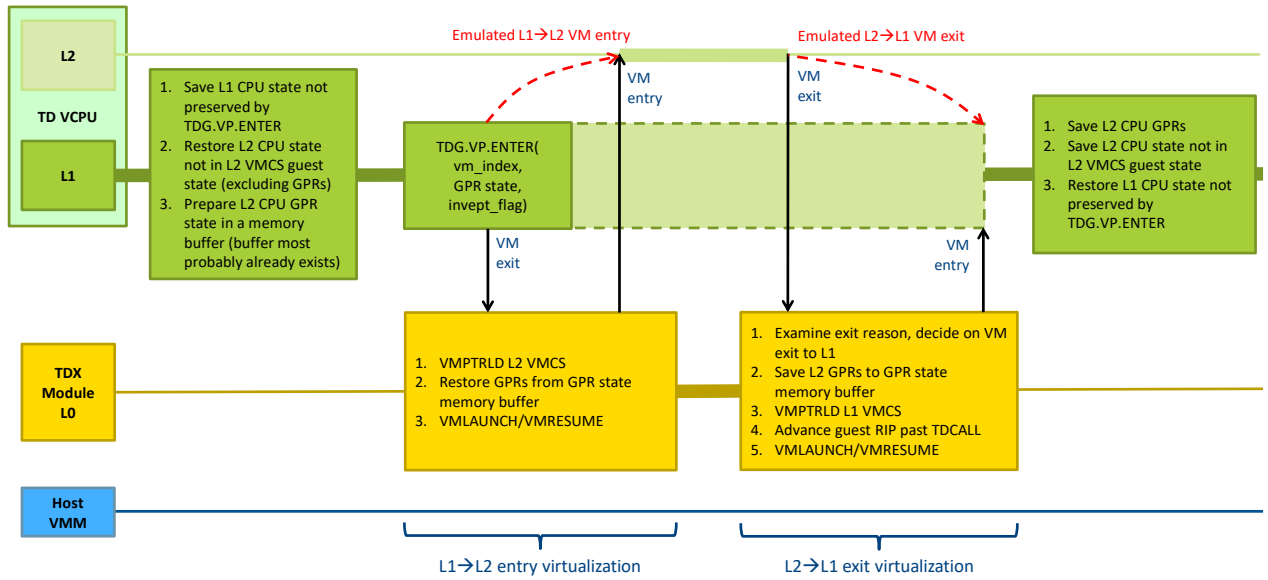

Figure 22.2: Example of L1→L2 VM Entry and Exit

22.2.1.1. *TDG.VP.ENTER Inputs and Outputs*

Intel SDM, Vol. 3, 24.7.2	VM-Exit Controls for MSRs
Intel SDM, Vol. 3, 24.8.2	VM-Entry Controls for MSRs
Intel SDM, Vol. 3, 26.4	Loading MSRs
Intel SDM, Vol. 3, 27.4	Saving MSRs

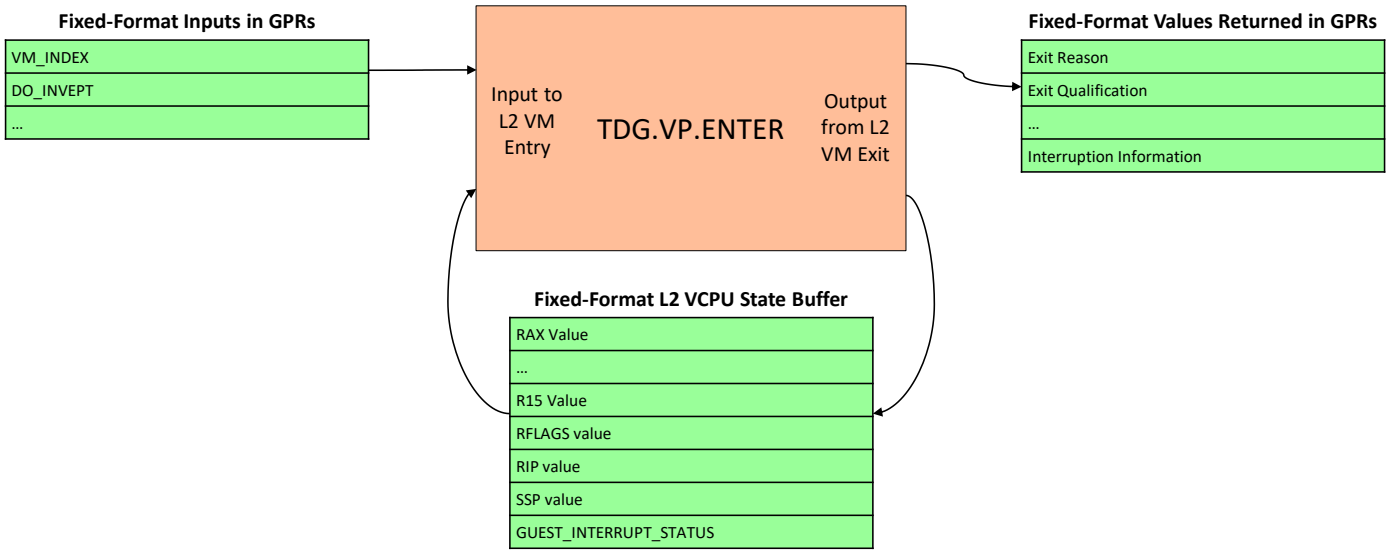

Figure 22.3: TDG.VP.ENTER Inputs and Outputs

22.2.1.1.1. Fixed-Format L2 Entry Guest State Buffer

The state of the L2 GPRs, RFLAGS, RIP and SSP and the L2 guest interrupt state is provided in a fixed-formal L2_ENTER_GUEST_STATE buffer in the L1 VPCU's private memory. Immediately before L2 VM entry, TDG.VP.ENTER reads this buffer and sets the applicable L2 guest state values. After L2 VM exit, if an L2→L1 exit or a TD exit is to be done, the TDX module writes the L2_ENTER_GUEST_STATE buffer with the current guest state values. The L2_ENTER_GUEST_STATE format is defined in the [TDX Module ABI Spec].

22.2.1.1.2. L2 VM Exit Information

On L2→L1 VM exit, the most useful VM exit information fields are provided as GPR output values. These include the following:

- VM exit reason

- VM exit qualification
- Guest linear address
- Guest physical address
- GPA
- VM-exit interruption information
- IDT-vectoring information
- VM-exit instruction information
- Guest CS information (base, selector, limit and access rights)
- Current privilege level (CPL)

For details, see the TDG.VP.ENTER definition in the [TDX Module ABI Spec].

22.2.1.2. *L2 State Preservation Across L1-to-L2-to-L1 Transitions (TDG.VP.ENTER)*

Intel SDM, Vol. 3, 23.4.1 Guest-State Area

A subset of the L2 VCPU state is preserved across L2 VM exit and subsequent L2 VM entry transitions. The preserved state includes the following fields, which are the fields supported by the L2 VMCS guest state area:

- CR0, CR3, CR4, DR7, RSP, RFLAGS
- CS, DS, ES, FS, GS, SS, LDTR, TR, GDTR, IDTR
- MSRs that are stored in VMCS guest state:
  - IA32_DEBUGCTL
  - IA32_SYSENTER_CS
  - IA32_SYSENTER_ESP
  - IA32_SYSENTER_EIP
  - IA32_PERF_GLOBAL_CTRL (if the TD is allowed to use it, i.e., ATTRIBUTES.PERFMON is 1)
  - IA32_PAT
  - IA32_EFER
  - IA32_RTIT_CTL (if the TD is allowed to use it, i.e., XFAM[8] is 1)
  - IA32_S_CET (if the TD is allowed to use it, i.e., XFAM[12:11] are 11)
  - IA32_INTERRUPT_SSP_TABLE_ADDR (if the TD is allowed to use it, i.e., XFAM[12:11] are 11)
  - IA32_PKRS (if the TD is allowed to use it, i.e., ATTRIBUTES.PKS is 1)
  - IA32_SPEC_CTRL

The L1 VMM is responsible for preserving other L2 VCPU state after L2→L1 VM exit (termination of previous TDG.VP.ENTER) and restoring it before invoking TDG.VP.ENTER.

22.2.1.3. *L2 VM Exit Handling*

On VM exit from an L2, the TDX module examines the VM exit information and decides on a proper way to handle the VM exit, which may be one of the following:

Local Flow: Handle the L2 VM exit locally and resume the VCPU in L2 mode. An example of this case is #GP(0) injection on a WRMSR VM exit from L2.

L2→L1 Exit: Handle the L2 VM exit as an L2→L1 VM exit and resume the VCPU in L1 mode. Examples of this case are CPUID handling (23.9) and posted interrupt handling.

TD Exit: Handle the L2 VM exit as a TD exit to the host VMM. An example of this case is physical NMI handling.

22.2.1.4. *L2-to-L1 VM Exit: Returned L2 State*

On L2 VM exit which results in a virtual L2→L1 VM exit, the TDX module completes the TDG.VP.ENTER operation as follows:

- The TDX module dumps the content of GPRs, RFLAGS, RIP and SSP to the fixed-format L2_ENTER_GUEST_STATE provided by the L1 VMM.
- The TDX module reads the fixed-format VM exit information (VM-exit reason etc.) from the L2 VMCS and returns them in GPRs.

22.2.2. Direct Asynchronous TD Exit from L2, and Subsequent TD Entry

22.2.2.1. *Asynchronous TD Exit from L2*

On VM exit from L2, multiple VM exit reasons lead to a TD exit to the host VMM. In such cases, these events have no meaning for the TD. They may be physical in nature (interrupts, NMI, machine check etc.), or they may be logical (e.g., EPT violation) but the TDX module determines they are a in the responsibility of the host VMM.

Before exiting to the host VMM, the TDX module does the following:

- The TDX module saves the CPU state into TDVPS; this would become the VCPU L1 state if a subsequent TD entry causes a virtual L2→L1 VM entry as described below. The identity of the exited L2 VM is recorded in TDVPS.CURR_VM.
- The TDX module writes the TDG.VP.ENTER output memory operands, e.g., L2_ENTER_GUEST_STATE. This is required since memory operands are specified by GPA. GPA→HPA translation may no longer be valid (e.g., the page may be removed by the host VMM) once TD exit is done and TLB tracking variables have been updated. For details of GPA→HPA translation tracking, see the [TDX Module Base Spec].

The outputs of TDH.VP.ENTER indicate the VM and VCPU index to the host VMM. For details, see the [TDX Module ABI Spec].

22.2.2.2. *VCPU Resumption to L2 on TD Entry following a TD Exit from L2*

By default, the TDH.VP.ENTER following a TD exit from an L2 resumes that VCPU in L2.

The TDX module restores the VCPU L2 state from the TDVPS, where it was saved before TD exit.

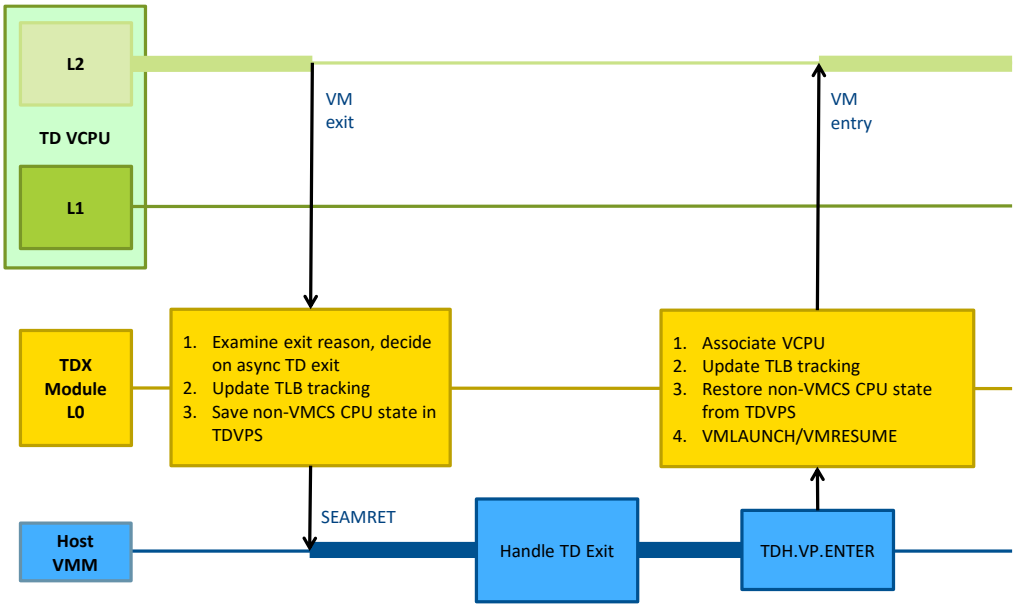

Figure 22.4: Example of Direct TD Exit from L2 and TD Entry Back to L2

22.2.3. TDG.VP.VMCALL: Synchronous TD Exit from L2 and Subsequent TD Entry

TDG.VP.VMCALL may be invoked by an L2 VM, if TDVPS.L2_CTLS[VM].ENABLE_TDVMCALL has been set to 1 by the L1 VMM. The behavior is similar to TDG.VP.VMCALL invoked by L1. The TDX module saves and restores the CPU state to/from TDVPS. TD entry following a TDG.VP.VMCALL resumes the VM that exited, unless routing to L1 has been requested by the host VMM, as described below.

Note: The L1 VMM has no control how the L2 VM uses TDG.VP.VMCALL. L1 should only set TDVPS.L2_CTLS[VM].ENABLE_TDVMCALL if it trusts the L2 VM to work correctly (e.g., to understand that any response from the host VMM is untrusted).

22.2.4. TD Exit from L2 Routed by the Host VMM to L1 on Subsequent TD Entry

When resuming the TD after a TD exit, the host VMM can request that L1 will be resumed, by setting the RESUME_L1 input flag of TDH.VP.ENTER. From the point of view of the L1 VMM, the L2→Host→L1 transition looks as a normal L2→L1 VM exit, except the completion status of TDG.VP.ENTER indicates TDX_L2_EXIT_HOST_ROUTED.

If resuming into L1 after a synchronous (TDG.VP.VMCALL) TD exit, L1 is resumed (i.e., the TDG.VP.ENTER it has invoked is terminated) with a TDX_L2_EXIT_HOST_ROUTED_TDVMCALL status. The L2 VCPU state reflects the successful completion of TDG.VP.VMCALL.

RESUME_L1 is sticky. If resumption of L1 encountered a problem that required a TD exit (e.g., an EPT violation) the following TD entry resumes L1 and provides the same TDX_L2_EXIT_HOST_ROUTED status.

22.2.4.1. *Asynchronous TD Exit from L2 Routed to L1*

A possible use case is TD exit due to an interrupt, where this was requested by the L1 VMM for, e.g., L2 VPUs rendezvous.

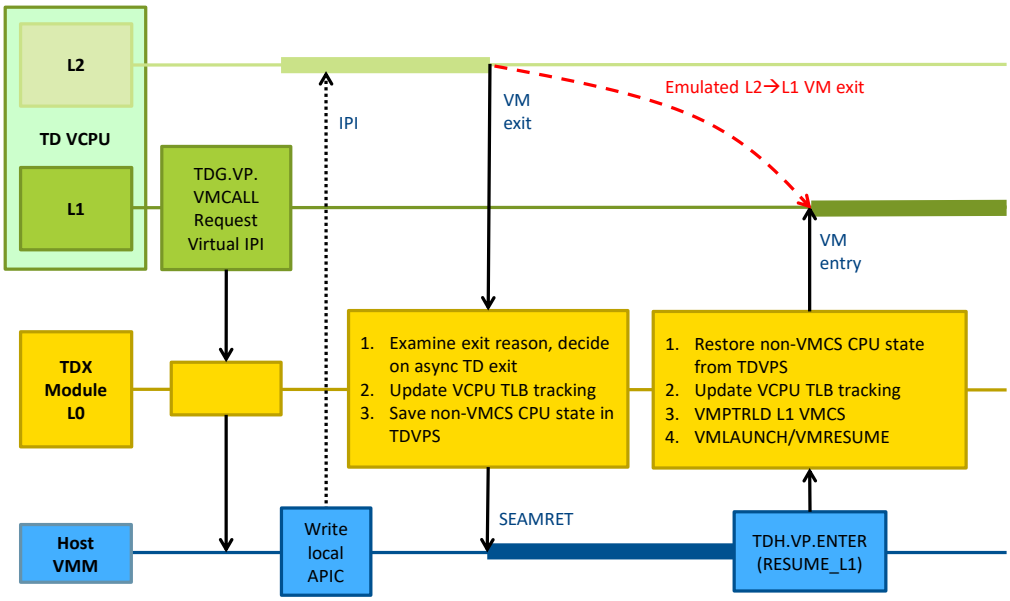

Figure 22.5: Example of Direct TD Exit from L2 and TD Entry Back to L1

In this case, the TDX module emulates the L2→L1 VM exit behavior:

- The TDX module has already written the output memory operands of TDG.VP.ENTER as part of the TD exit from L2, as described above.
- The TDX module updates the output GPR operands (exit information etc.) of TDG.VP.ENTER, based on the L2 VMCS.
- TLB tracking is updated as described in the [TDX Module Base Spec].

22.2.4.2. *Synchronous (TDG.VP.VMCALL) TD Exit from L2 Routed to L1*

A possible use case if for some service requested by the L2 VM from the host VMM, which needs to be handled by the L1 VMM as soon as possible after TD resumption.

On TD entry, if the last TD exit was due to a TDG.VP.VMCALL from L2, and L1 is to be resumed, the TDX module first updates the L2 VCPU state as if TDG.VP.VMCALL was completed, then emulates an L2→L1 exit. L2's GPR and XMM values are as returned by the host VMM, and RIP is advanced to the next L2 instruction.

22.3. L1 Posted Interrupts Handling for TD Partitioning

Intel SDM, Vol. 3, 30.2.1	Evaluation of Pending Virtual Interrupts
Intel SDM, Vol. 3, 30.2.2	Virtual-Interrupt Delivery
Intel SDM, Vol. 3, 30.6	Posted-Interrupt Processing

22.3.1. Overview

TDX supports posting interrupts only to the L1 VMM. When a posted-interrupt notification vector is recognized in TDX non-root mode, there are be three possible cases:

- If a TD VCPU is currently running on the LP in the L1 VMM, the CPU processes the posted-interrupt descriptor as described in the [Intel SDM]. This case is described in the [TDX Module Base Spec].
- During the TDG.VP.ENTER flow, if the TDX module recognizes that there is a pending posted interrupt to the L1 VMM, it aborts the L1→L2 entry and resumes L1. The CPU can then process the interrupt.
- If a TD VCPU is currently running on the LP in an L2 VM, the notification vector is recognized as a normal external interrupt, causing a VM exit to the TDX module. TDX module emulates the CPU's posted interrupt injection to the L1 VMM.

22.3.2. Pending L1 Posted Interrupt during L1-to-L2 Entry

Typically, the L1 VMM will disable interrupts delivery (clear RFLAGS.IF) before invoking TDG.VP.ENTER to start an L1→L2 entry. If a notification interrupt is delivered after RFLAGS.IF is cleared but before TDG.VP.ENTER runs, that interrupt gets processed by the CPU as described in [Intel SDM, Vol. 3, 30.6], but the posted virtual interrupt doesn't get delivered yet. That interrupt will only be delivered once L1 resumes running.

To prevent long interrupt latency, the TDX module detects whether there is a pending virtual interrupt during the TDG.VP.ENTER flow. In that case, it terminates TDG.VP.ENTER and returns to L1, with a status code indicating TDX_PENDING_INTERRUPT. Once running in L1 and RFLAGS.IF is set, the CPU evaluates the posted virtual interrupt vs. the LP's interruptibility state and delivers it. The L1 software doesn't need to directly handle the L2→L1 exit.

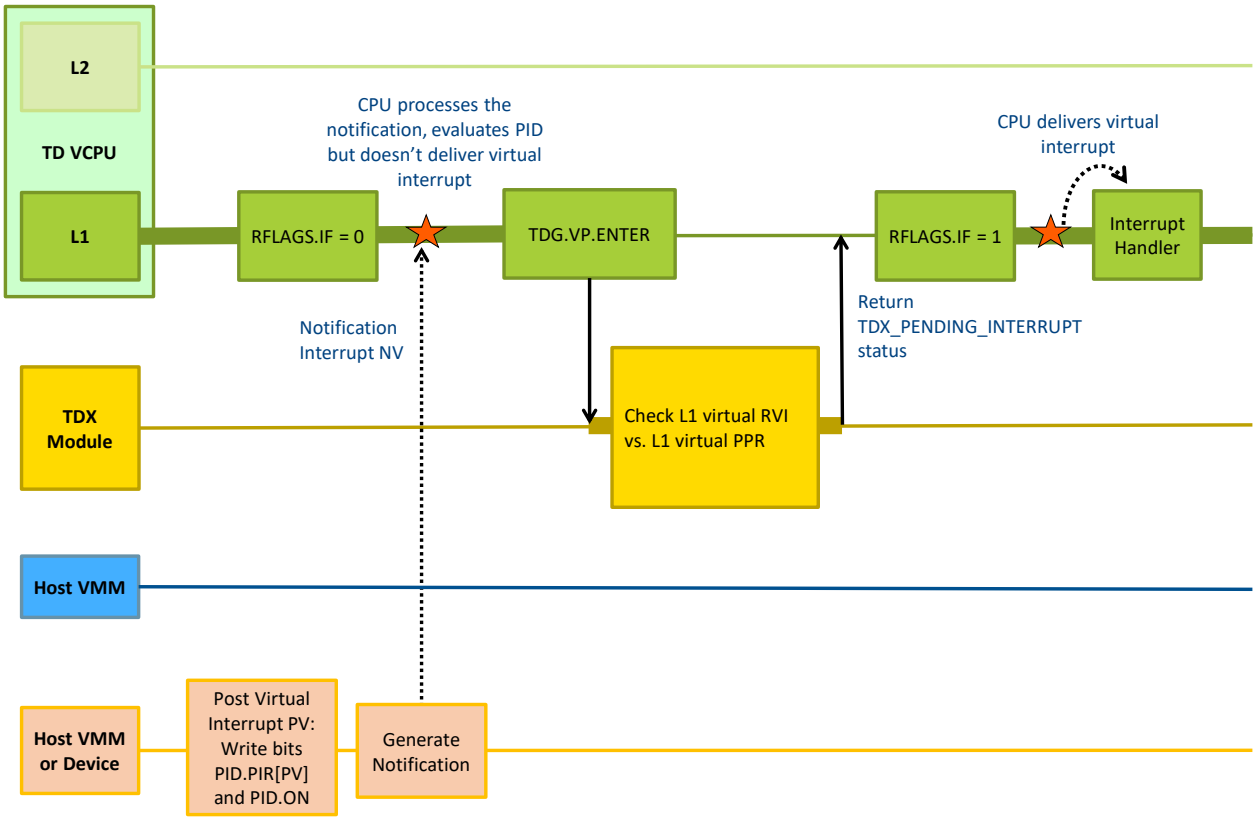

Figure 22.6: Pending L1 Posted Interrupt Detection during L1-to-L2 Entry

22.3.3. L1 Posted Interrupt Received during L2 Run Time

If an interrupt is posted to the L1 VMM while an L2 VM is running, there's a need to resume L1 operation as soon as possible to prevent long latencies. To do that, the TDX module emulates the CPU operation for posted interrupt notification, as described below.

1. The physical notification interrupt received by the LP when a VCPU is running in L2 causes a VM exit to the TDX module.
   1.1. Equivalent to [Intel SDM, Vol. 3, 30.6, step 1]: The LP acknowledges the local APIC
2. On VM exit, the TDX module processes the posted interrupt by emulating the CPU behavior as described in [Intel SDM, Vol. 3, 30.6, steps 2 through 7].
   2.1. If the physical interrupt vector is different than the L1 VMM's notification vector, this is handled as a normal interrupt by a TD exit to the host VMM.
   2.2. If the posted interrupt's priority is not higher than the L1 VMM's current priority (in virtual PPR), then the L2 VM is resumed.
   2.3. The TDX module emulates an L2→L1 exit. The returned completion code of TDG.VP.ENTER indicates TDX_L2_EXIT_PENDING_INTERRUPT.
   2.4. Once running in L1, the CPU evaluates the posted virtual interrupt vs. the LP's interruptibility state and delivers it. The L1 software doesn't need to directly handle the L2→L1 exit.

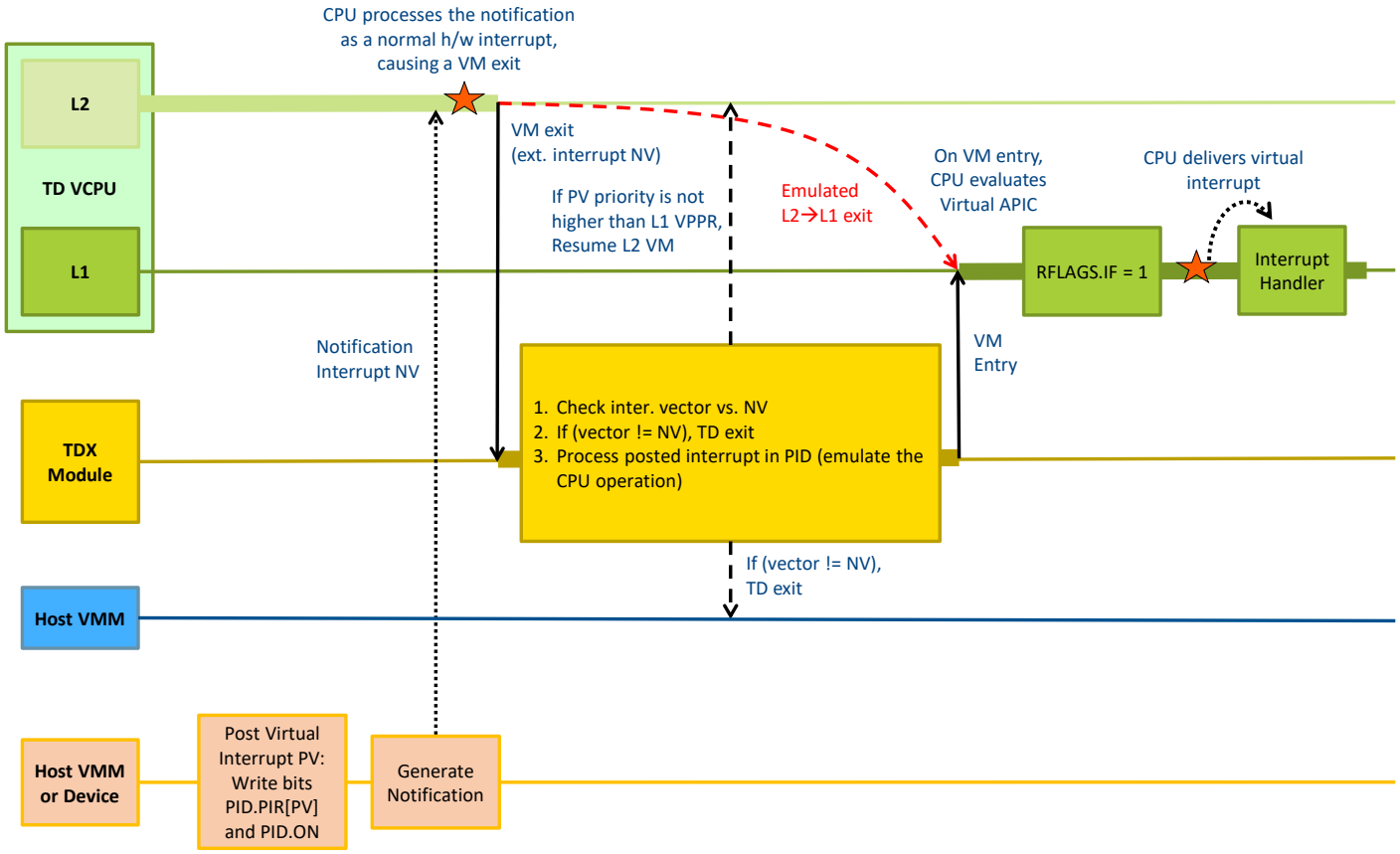

Figure 22.7: L1 Posted Interrupt Received during L2 Run Time

22.3.4. L1 Posted Interrupt Received before Host-to-L2 Entry

Typically, the host VMM runs with interrupts disabled. If a posted interrupt notification is received when the host VMM is running, it is not immediately recognized; it is only recognized after TD entry. If the TD entry is to an L2 VM, the posted interrupt notification is processed as described above, typically ending in an L2→L1 exit where the L1 VMM can handle the posted interrupt.

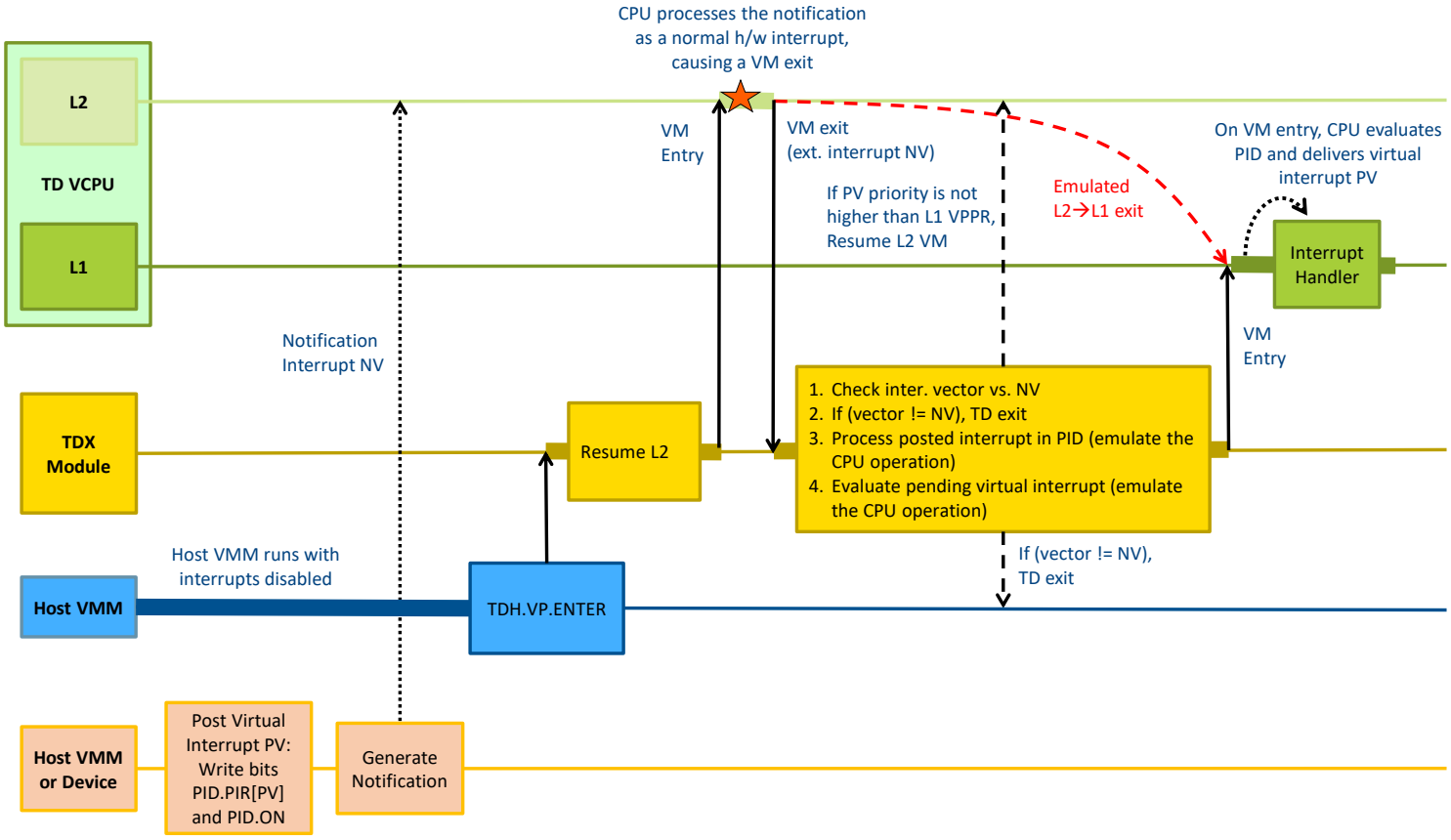

Figure 22.8: L1 Posted Interrupt Received before Host-to-L2 Entry

22.4. L2 VM TLB Address Space Identifier (ASID)

See the [TDX Module Base Spec]. Note the following points:

- Each VM within a TD has its own EPTP, and thus its own ASID.
- INVEPT, used by the TDX module as part of, e.g., TDH.VP.FLUSH, is required to be done for the EPTP of each VM within the TD.

23. L2 VM CPU Virtualization (Non-Root Mode Operation)

This chapter describes how the Intel TDX module virtualizes the CPU to a guest TD.

23.1. General Aspects of L1 Operation as a VMM

23.1.1. L1 VMM Usage of VMX Facilities

VMX Instructions

VMX instructions are not directly available to the L1 VMM. This is indicated by CR4.VMXE (bit 13) value being 0. Instead, it uses interface functions provided by the TDX module.

- To enter into L2 VM operation, the L1 VMM invokes TDG.VP.ENTER.
- To read and write L2 VMCS, the L1 VMM uses TDG.VP.RD and TDG.VP.WR.
- To flush cached L2 EPT translations, the L1 VMM uses TDG.VP.INVEPT or TDG.VP.ENTER.

L2 VMCS

L2 VMCS is accessible to the L1 VMM using the metadata access interface functions TDG.VP.RD and TDG.VP.WR. The TDX module controls which L2 VMCS fields can be written. In addition, selected L2 VMCS fields are provided as TDG.VP.ENTER inputs and outputs.

L2 VMCS fields, as visible to the L1 VMM, are virtual. E.g., addresses are Guest-Physical Addresses (GPAs). The TDX module translates the virtual values to real values stored in the L2 VMCS and used by the CPU.

Other VMX Control Structures

- The L2 MSR bitmaps page is not directly accessible to the L1 VMM. Instead, it used TDG.VP.RD and TDG.VP.WR to access it.
- The L2 Virtual APIC page is mapped in the L1 VMM's GPA space as a private page and is directly accessible to the L1 VMM. The L1 VMM configures the address by writing its GPA to the L2 VMCS' *virtual-APIC address* field.
- The L2 secure EPT is not directly accessible to the L1 VMM. The L1 VMM controls L2 GPA mapping attributes (but not GPA-to-HPA translations) using TDG.MEM.PAGE.ATTR.WR.

23.1.2. Enumeration of VMX Capabilities Available to the L1 VMM

Intel SDM, Vol. 3, Appendix A VMX Capability Reporting Facility

The virtual values of IA32_VMX_* MSRs enumerate VMX capabilities available to the L1 VMM. Not all bits are applicable since, e.g., VMX instructions and EPT management are done indirectly via TDX module functions.

Table 23.1: Enumeration of VMX Capabilities Available to the L1 VMM

MSR Name	MSR Index	Description
IA32_VMX_BASIC	0x0480	For details, see the [TDX Module ABI Spec].
IA32_VMX_MISC	0x0485	For details, see the [TDX Module ABI Spec].
IA32_VMX_PINBASED_CTLS	0x0481	These MSRs are not used; read access results in a #VE. Use the IA32_VMX_TRUE_* MSRs listed below.
IA32_VMX_PROCBASED_CTLS	0x0482	
IA32_VMX_EXIT_CTLS	0x0483	
IA32_VMX_ENTRY_CTLS	0x0484	
IA32_VMX_TRUE_PINBASED_CTLS	0x048D	
IA32_VMX_TRUE_PROCBASED_CTLS	0x048E	
IA32_VMX_PROCBASED_CTLS2	0x048B	
IA32_VMX_PROCBASED_CTLS3	0x0492	

MSR Name	MSR Index	Description
IA32_VMX_TRUE_EXIT_CTLS	0x048F	These MSRs enumerate L2 VMCS fields that are either fixed-0, fixed-1 or that can be modified by the L1 VMM. The virtual values depend on the values supported by the CPU (i.e., the native values of the same MSRs) and on the TD configuration. See the section below describing non-standard behavior of those MSRs.
IA32_VMX_EXIT_CTLS2	0x0493	
IA32_VMX_TRUE_ENTRY_CTLS	0x0490	
IA32_VMX_CR0_FIXED0	0x0486	
IA32_VMX_CR0_FIXED1	0x0487	
IA32_VMX_CR4_FIXED0	0x0488	
IA32_VMX_CR4_FIXED1	0x0489	
IA32_VMX_VMCS_ENUM	0x048A	This MSR is not used; read access results in a #VE.
IA32_VMX_EPT_VPID_CAP	0x048C	For details, see the [TDX Module ABI Spec].
IA32_VMX_VMFUNC	0x0491	Virtualized as 0.

Non-Standard Behavior of Virtual IA32_VMX_* MSRs

In multiple cases, the L2 VMCS bit values are set based on the TD configuration and are not writable by the host VMM. For example, the L2 VMCS' tertiary processor-based execution controls bit 5 (GPAW) is set based on the TD's GPAW configuration; the L1 VMM can't modify the value of this bit. Such bits are enumerated by the applicable IA32_VMX_* MSR as fixed-0, even though their value may be 1. This mean that the L1 VMM must not attempt to modify their value.

When writing to such L2 VMCS fields using TDG.VP.WR, the L1 VMM may use the write mask parameter to avoid modifying any bit that is enumerated as either fixed-0 or fixed-1.

23.1.3. Unit Conversion

As a rule, L1 VMM access to L2 VMCS fields and TD migration of L2 VMCS fields use virtual units (e.g., GPA, TSC ticks etc.). The values are converted to/from native units (e.g., HPA) when the L2 VMCS is written/read.

L2 VMCS Fields Specified as Physical Address

L2 VMCS fields that are specified as physical address are converted to/from GPA when read/written by the L1 VMM. GPA to HPA conversion is discussed in [REF].

L2 VMCS Fields Specified as TSC Ticks

L2 VMCS fields that are specified using TSC tick units are converted to/from virtual TSC units when read/written by the L1 VMM. Those fields are migrated using virtual TSC units so similar conversion is done for export/import. The same fields are not converted when read/written by the host VMM (for debuggable TDs) – the real h/w units are used. These fields include:

- PLE_Gap
- PLE_Window

L2 VMCS Fields Specified as Crystal Clock Ticks

L2 VMCS fields that are specified using crystal clock tick units, as enumerated by CPUID(0x15), are converted to/from virtual crystal clock units when read/written by the L1 VMM. Those fields are migrated using virtual crystal clock units so similar conversion is done for export/import. The same fields are not converted when read/written by the host VMM – the real h/w units are used. These fields include:

- Instruction Timeout Control

23.1.4. L2-to-L1 VM Exit Handling

Intel SDM, Vol. 3, 27.2.3 Information About NMI Unblocking Due to IRET
Intel SDM, Vol. 3, 27.2.4 Information for VM Exits During Event Delivery

The L1 VMM is expected to handle special L2 VM exit conditions that may occur in the Intel VMX architecture, such as:

- Handling of NMI unblocking due to IRET, e.g., re-blocking NMI by updating the L2 VMCS' *guest interruptibility state* field.
- Handling of VM exit during event delivery, e.g., re-injecting the event to L2 based on the L2 VMCS' *IDT-vectoring information* field etc.

Such conditions are handled by the TDX module itself if a VM exit from L2 results in a local flow that ends by VM entry back into L2.

23.2. L2 VM VCPU Initial State

The TDX module initializes the L2 VM state of each VCPU as part of the VCPU initialization (TDH.VP.INIT). The initial state is detailed in the [TDX Module ABI Spec].

The L1 VMM is responsible for properly setting the VCPU L2 state to the desired values, by writing to the TDVPS (including L2 VMCS) using the TDG.VP.WR* interface functions.

23.3. L2 VM Run Time Environment Enumeration

The TDX module does not implement any special behavior. The L1 VMM may virtualize CPUID leaves/sub-leaves and MSRs to enumerate the run time environment for its guest L2 VMs.

23.4. L2 VM CPU Mode Restrictions

Intel SDM, Vol. 3, 2.2 Modes of Operation
Intel SDM, Vol. 3, 9.8.5 Initializing IA-32e Mode
Intel SDM, Vol. 3, 11.5.1 Cache Control Registers and Bits
Intel SDM, Vol. 3, 24.6.6 Guest/Host Masks and Read Shadows for CR0 and CR4

An L2 VM running in TDX non-root mode may use any CPU mode available in VMX non-root mode.

Table 23.2: L2 VM CPU Mode Restrictions

Restriction	L2 VM

CPU and Paging Modes	The CPU is allowed to run in the following modes:
	• Real mode
	• Protected mode (32-bit) with or without paging
	• IA-32e mode with paging, with the sub-modes controlled by CS.L:
	  ○ 64-bit mode
	  ○ Compatibility (32-bit) mode
	Contrary to L1 VMs, CR0.PE and IA32_EFER.LME are not enforced by the TDX module. The L1 VMM may enforce them.

Execute Disable	When running in IA-32e mode, the PT Execute Disable bit (63) is always enabled. To achieve this, IA32_EFER.NXE is enforced to 1, as described in the following sections.
Caching is Always Enabled	The L2 VM runs in Normal Cache Mode. To achieve this, CR0.CD and CR0.NW are enforced to 0, as described in the following sections. The L1 VMM may set the L2 VMCS' CR0 guest/host mask and read shadow execution control such that the L2 VM may attempt to set CR0.CD and CR0.NW and sees their virtual value as if setting was successful. See the following sections for details.

23.5. L2 VM VCPU Instructions Restrictions

TDX rules for CPU instructions restrictions apply to the whole TD, i.e., instructions that are always blocked in the L1 VM are also blocked in the L2 VM. The L1 VMM may configure further restrictions.

Instructions may be blocked depending on VCPU features enabled by the L1 VMM, which may virtualize CRs and MSRs to its L2 VMs. The L1 VMM is responsible for proper feature enumeration to the L2 VM, by virtualized CPUID, CR4 and MSRs. The TDX module restricts this virtualization based on the overall TD restrictions, as described in the following sections.

23.5.1. Mechanisms of Blocking

- In some cases, the CPU will block (#UD) instructions based on VMX controls, CRs, MSRs and XCR0
- In other cases, execution of instructions causes a VM exit to the TDX module. By default, the TDX module emulates an L2 VM exit to the L1 VMM.

23.5.2. Instructions that Cause an L2-to-L1 Exit Unconditionally

Intel SDM, Vol. 3, 25.1.2 Instructions That Cause VM Exits Unconditionally

- Instructions that architecturally cause VM exit unconditionally: CPUID, GETSEC, INVD, XSETBV, INVEPT, INVVPID, VMCALL, VMCLEAR, VMLAUNCH, VMPTRLD, VMPTRST, VMRESUME, VMXOFF, and VMXON
- VMFUNC, VMREAD and VMWRITE
- RSM
- SEAMCALL
- ENCLS, ENCLV

23.5.3. Instructions that Cause a #UD Unconditionally

- SEAMRET

23.5.4. Instructions that Cause an L2-to-L1 Exit

- String I/O (INS*, OUTS*), IN, OUT
- HLT
- MONITOR, MWAIT
- WBINVD, INVD

23.5.5. Other Cases of Unconditionally Blocked Instructions

- Guest TD execution of PCONFIG results in a #UD, #VE or an L2→L1 VM Exit. See 23.15 for details.
- Guest TD execution of ENQCMD results in a #GP(0).
- Guest TD execution of ENQCMDS when CPL is 0 results in an L2 VM exit to the L1 VMM. Otherwise, it results in a #GP(0).

23.6. L2 VM Extended Feature Set

At the whole guest TD scope, **TDCS.XFAM (Extended Features Allowed Mask)** is provided as an input during guest TD build process. The L1 VMM may allow its L2 VMs any **subset** of the TD's extended features allowed by XFAM.

L1 VMM is responsible for context-switching: saving extended state (XSAVES) before L2 VCPU entry and restoring it (XRSTORS) after exit.

CR4 virtualization, MSR virtualization and instructions virtualization are discussed in the following sections. The architecture helps ensure that the L1 VMM can only enable features that are enabled for the whole TD.

23.7. L2 VM CR Handling

23.7.1. CR0 and CR4

Intel SDM, Vol. 3, 2.5	Control Registers
Intel SDM, Vol. 3, 23.8	Restrictions on VMX Operation
Intel SDM, Vol. 3, 24.6.6	Guest/Host Masks and Read Shadows for CR0 and CR4
Intel SDM, Vol. 3, 25.6	Unrestricted Guests

23.7.1.1. *Background: CR0 and CR4 Execution Controls*

The VMCS CR0 and C4 VM-execution control fields include **guest/host masks** and **read shadows**.

Guest/host mask bits set to 1 correspond to bits "owned" by the host:

- A guest attempt to set host-owned CR bit X to a value differing from bit X of the corresponding read shadow causes a VM exit.
- Guest reads return values for host-owned bits from the corresponding read shadow.

Guest/host mask bits cleared to 0 correspond to bits "owned" by the guest:

- A guest attempt to modify guest-owned bit X succeeds.
- Guest reads return values for guest-owned bits from the CR itself.

For TDX, the real host is always the TDX module. For TD Partitioning, the TDX module virtualizes the guest/host behavior so that the L1 VMM can configure itself as the host of L2 VM's CR0/4 bits.

23.7.1.2. *CR0 and CR4 Enumeration to the L1 VMM*

The virtual values of IA32_VMX_CR0/4_FIXED0/1 MSRs, to enumerate to the L1 VMM which L2 guest's CR bits may be freely set, and which bits must be fixed as 0 or as 1. Bits enumerated as fixed-0 or fixed-1 bits are considered to be owned by the TDX module. Other bits are owned by the TD (L1 VMM).

Enumeration is based on the following:

- Platform capabilities, as enumerated by the real values of the same MSRs
- Bits known to the TDX module as reserved
- TDX architecture restrictions
- TD configuration

The table below lists CR0 bits whose enumeration is impacted by the TDX architecture and the TD configuration.

Table 23.3: L2 VM CR0 Enumeration to the L1 VMM (by IA32_VMX_CR0_FIXED0/1)

Bit	Name	Condition
5	NE	Enumerated as fixed-1
29	NW	Enumerated as fixed-0
30	CD	Enumerated as fixed-0
31	PG	Enumerated as non-fixed

The table below lists CR4 bits whose enumeration is impacted by the TDX architecture and the TD configuration.

Table 23.4: L2 VM CR4 Enumeration to the L1 VMM (by IA32_VMX_CR4_FIXED0/1)

Bit	Name	Condition
6	MCE	Enumerated as fixed-1
13	VMXE	Enumerated as fixed-1
14	SMXE	Enumerated as fixed-0
19	KL	If TDCS.ATTRIBUTES.KL is 0, enumerated as fixed-0

Bit	Name	Condition
22	PKE	If TDCS.XFAM[9] is 0, enumerated as fixed-0
23	CET	If TDCS.XFAM[12:11] are 00, enumerated as fixed-0
24	PKS	If TDCS.ATTRIBUTES.PKS is 0, enumerated as fixed-0
25	UINTR	If TDCS.XFAM[14] is 0, enumerated as fixed-0
32	FRED	If virtual CPUID(0x7,1).EAX[17] is 0, enumerated as fixed-0

In addition to the above, CPUID values virtualized by the TDX module enumerate the availability of multiple CR0/CR4 bits, but not all of them.

23.7.1.3. *L2 CR0/4 Initial Values*

For CR0, bits PE (0) and NE (5) are initialized to 1. All the other bits are initialized to 0.

For CR4, bits MCE (6) and VMXE (13) are initialized to 1. All the other bits are initialized to 0.

23.7.2. CR3 and CR8

The L1 VMM can use the L2 VMCS controls to induce L2→L1 exit on CR3 and/or CR8 loads and/or stores. It has access to the guest CR3 and CR8 state. The L1 VMM can also access L2 VMCS CR3-target values and CR3-target count fields. No special TDX module handling is required.

23.8. L2 VM MSR Handling

The L1 VMM configures MSR virtualization policy per VCPU per L2 VM by writing to the L2 MSR Bitmaps page, which is part of TDVPS, page using TDG.VM.WR. The format is as specified in the [Intel SDM]: a bit value of 1 indicates an L2 to L1 VM exit on MSR read or write. A bit value of 0 indicates that no L2 to L1 VM exit is requested.

The value written by the L1 VMM is actually saved in a shadow page. The TDX module combines the L1 VMM's policy with the whole TD's MSR handling policy and sets the value in the real L2 MSR Bitmaps page, linked to the L2 VMCS and used by the CPU.

Depending on the whole TD MSR virtualization policy, a VM exit to the TDX module may still happen. In this case the TDX module may emulate the MSR access. By default, in cases where for L1 it would inject a #VE, the TDX module emulates an L2 to L1 VM exit. This means that the L1 VMM must be ready to accept such VM exit even if it didn't request them.

The default value of the MSR exit bitmaps shadow page is all-1, i.e., L2→L1 exit on all MSR accesses.

Table 23.5: Combining MSR Virtualization Policies (L2 VM and Whole TD)

			Whole TD MSR Virtualization Policy, Configured by the Host VMM		
			Direct CPU Access	**MSR Virtualization**	**#VE**
		TD MSR Exit Bitmap →	**0**	**1**	**1**
L2 VM MSR Virtualization Policy, Configured by the L1 VMM (Stored in the L2 MSR Exit Bitmaps Shadow Page)	**0: No L2→L1 Exit**	**L2 MSR Exit Bitmap →**	0: No VM exit	1: VM exit	1: VM exit
		L2 VCPU RD/WRMSR Handling →	RD/WRMSR executed by CPU	TDX module emulates MSR access. May inject a #GP.	TDX module emulates L2→L1 VM exit
	1: L2→L1 Exit	**L2 MSR Exit Bitmap →**	1: VM exit	1: VM exit	1: VM exit
		L2 VCPU RD/WRMSR Handling →	TDX module emulates L2→L1 VM exit	TDX module emulates L2→L1 VM exit	TDX module emulates L2→L1 VM exit

The TDX module combines the VCPU-scope per-L2 VM's MSR exit bitmaps shadow page with the TD-scope MSR exit bitmaps page by a bitwise-OR, to create an architectural L2 MSR exit bitmaps page, which is used by the CPU. This operation is done on TD initialization and on L1 VMM writes to the L2 MSR exit bitmaps page.

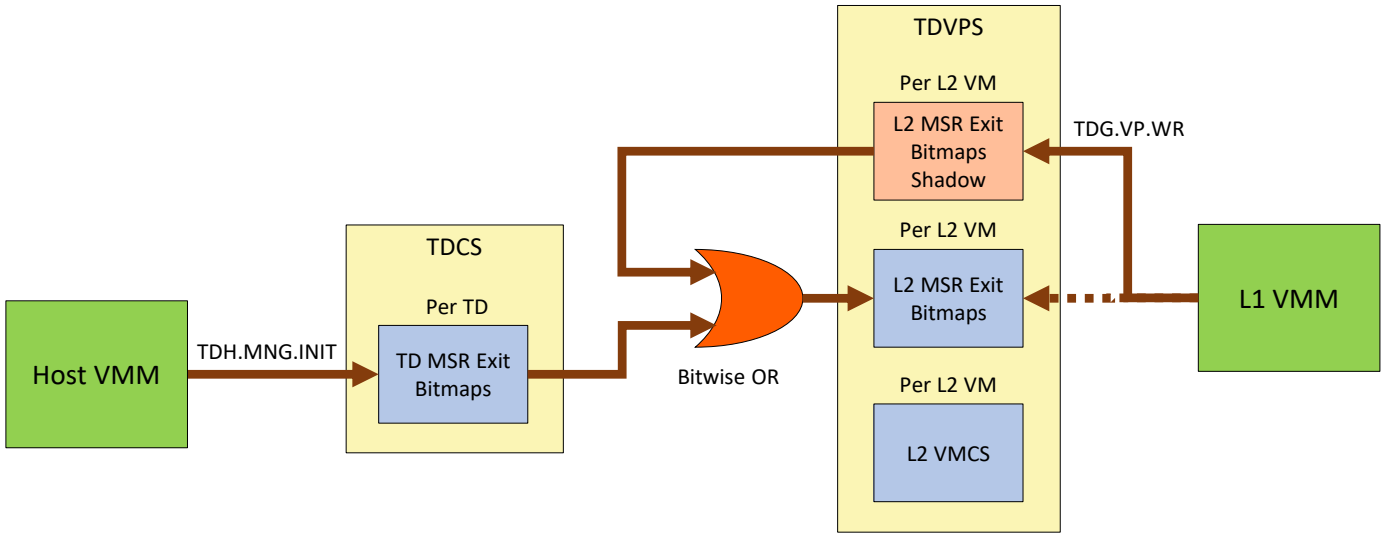

Figure 23.1: MSR Virtualization Policy Configuration

23.9. L2 VM CPUID Virtualization

CPUID execution by an L2 VM results in a VM exit to the TDX module, which then emulates an L2 to L1 VM exit. The L1 VMM may then emulate the CPUID instruction and resume the L2 VM.

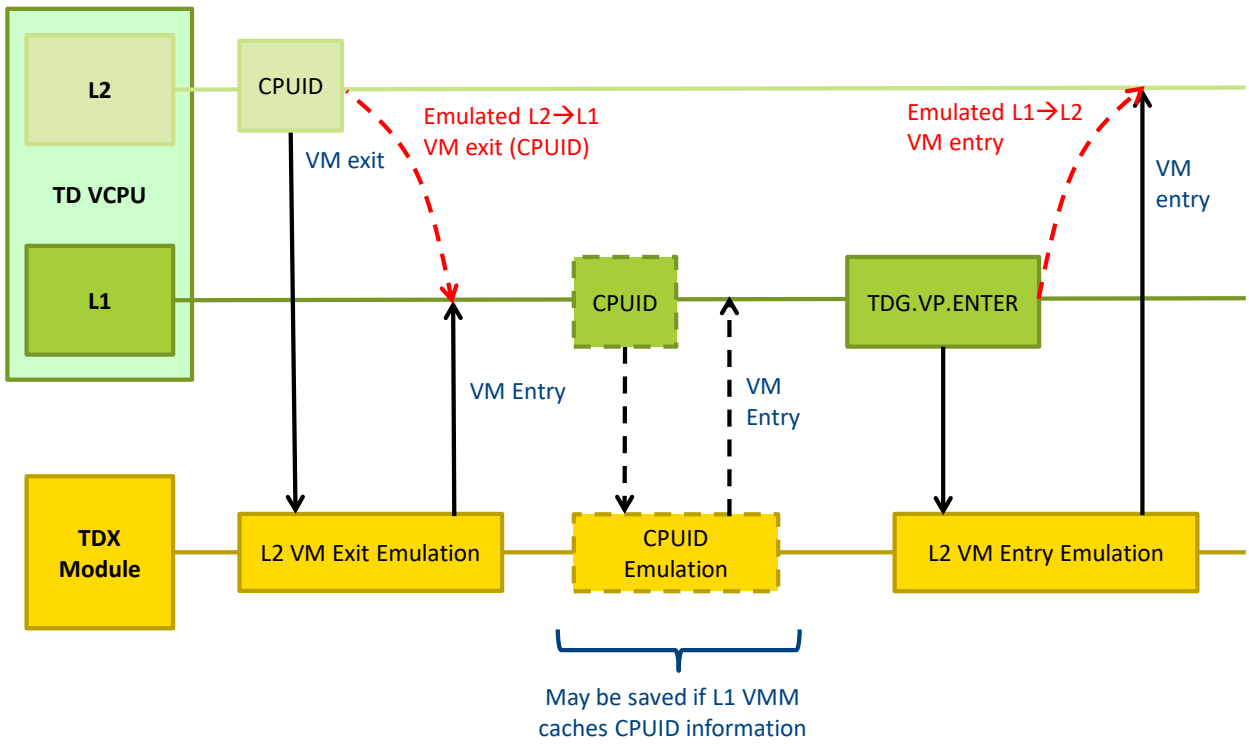

Figure 23.2: L2 VM CPUID Virtualization

23.10. L2 VM Interrupt Handling and APIC Virtualization

Intel SDM, Vol. 3, 24.6.8 Controls for APIC Virtualization
Intel SDM, Vol. 3, 29 APIC Virtualization and Virtual Interrupts

23.10.1. L2 Virtual APIC Page Access by the L1 VMM

Intel SDM, Vol. 3, 29.5 Virtualizing MSR-Based APIC Access

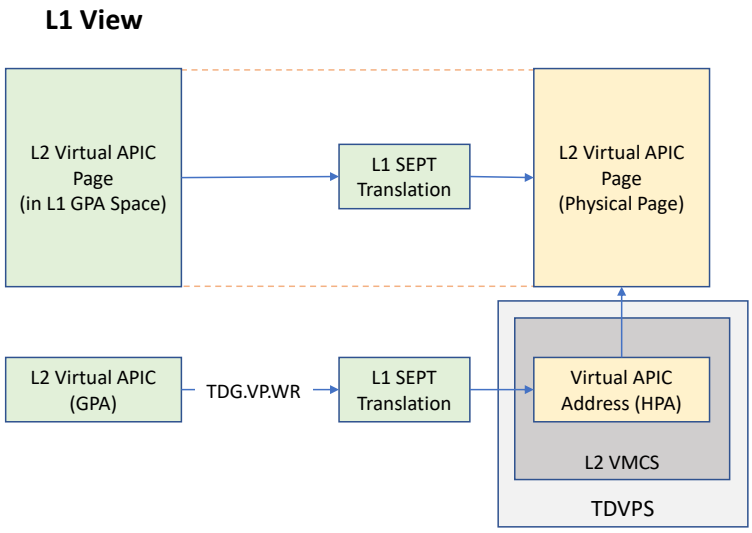

Figure 23.3: L2 Virtual APIC Page Mapping to L1 GPA Space

The L2 virtual APIC page is mapped in the L1 VMM's GPA space and is fully accessible to the L1 VMM.

The L1 VMM specifies the virtual-APIC address, using TDG.VP.WR, as a private GPA in the L1 address space. The GPA must be mapped in the L1 VMM's SEPT. The TDX module translates this GPA to an HPA for configuring the L2 VMCS.

The TDX module helps ensure that the translated address is valid when the L2 VCPU runs, using the mechanism described in the [TDX Module Base Spec].

23.10.2. L2 Virtual Local APIC Mode and Access by the L2 VM

The Local APIC is virtualized to L2 VMs in x2APIC mode (MSR access).

23.10.3. L2 Posted Interrupts (Not Supported)

Posting virtual interrupts to L2 VCPUs, either by the host VMM or I/O device, or by the L1 VMM, is not supported. The L1 VMM can manipulate the L2 VM's Virtual APIC page to inject virtual interrupts.

23.10.4. Virtual NMI Injection to L2 VCPU

There is no provisioning for the host VMM to inject a virtual NMI to L2 VCPUs.

The L1 VMM controls the L2 VMCS and can use the applicable fields (nmi-window exiting, VM-entry interruption information etc.) to inject a virtual NMI to L2 VCPUs.

23.10.5. Handling NMI Unblocking Due to IRET

Intel SDM, Vol. 3, 27.2.3 Information About NMI Unblocking Due to IRET

The L1 VMM is responsible for updating the L2 VM's guest interruptibility state based on the indication of NMI unblocking due to IRET. For VM exits due to faults, NMI unblocking due to IRET is indicated in bit 12 of the VM-exit interruption-information field. For VM exits due to EPT violation, instruction timeout and other reasons not applicable to TDX, it is indicated in bit 12 of the exit qualification. For details, see the [Intel SDM].

Note: VM exits from L2 where no L2→L1 exit happens are handled by the TDX module; the L1 VMM is not involved.

23.11. Vectored Events

23.11.1. Vector-on-Entry (VOE) Injection to L2

Intel SDM, Vol. 3, 26.6.1	Vectored-Event Injection

The L1 VMM can inject vectored event to an L2 VM using the L2 VMCS VM-entry interruption information, VM-entry exception error code and VM-entry instruction-length fields.

The L1 VMM can determine whether the event was injected successfully, based on TDG.VP.ENTER's completion status and exit reason, as follows:

- If L2 VM entry fails for some reason (e.g., bad guest state), VM-entry interruption-information remains valid and a subsequent L2 VM entry (e.g., after fixing bad guest state) will inject the event again. The host VMM can determine this based on TDG.VP.ENTER's completion status and exit reason.
- If L2 VM entry succeeds but triggers a VM exit that occurs before the event is injected, this is the case of VM exit during event delivery via IDT, described below. The L1 VMM typically reinjects the event.
- In other cases, VM entry succeeds, and the event is injected successfully. Guest execution starts at the entry point for the handler of the injected event.

23.11.2. L2-to-L1 Exit during Event Delivery via IDT

Intel SDM, Vol. 3, 24.9.3	Information for VM Exits That Occur During Event Delivery
Intel SDM, Vol. 3, 27.2.4	Information for VM Exits during Event Delivery
Intel SDM, Vol. 3, 26.5.1	Vectored-Event Injection

The L1 VMM is responsible for handling the case where an L2→L1 exit occurred during event delivery via IDT. This is indicated by the L2 VMCS IDT-vectoring information Valid bit (31). Typically, the L1 VMM re-injects the event using the L2 VMCS VM-entry interruption information, VM-entry exception error code and VM-entry instruction-length fields.

Note: VM exits from L2 where no L2→L1 exit happens are handled by the TDX module; the L1 VMM is not involved.

23.12. Prevention of L2 VM-Induced Denial of Service

By default, handling of L2 VM-induced denial of service is done the same way as that of the whole TD, as described in the [TDX Module Base Spec]. Bus lock detection and instruction timeout VM exits cause a TD exit.

23.13. Time Stamp Counter (TSC)

23.13.1. L2 VM TSC Virtualization

The virtual TSC frequency and value as seen by the L2 VM are the same as for the L1 VMM.

The L2 VM has the same access restrictions to TSC MSRs as the L1 VMM. It is not allowed to:

- Write IA32_TIME_STAMP_COUNTER
- Access IA32_TSC_ADJUST
- Access IA32_TSC_DEADLINE

Any access violation results in an L2→L1 VM.

23.13.2. L2 VM TSC Deadline Support

Intel SDM, Vol. 3, 10.5.4.1	TSC-Deadline Mode
Intel SDM, Vol. 3, 25.5.1	VMX-Preemption Timer
Intel SDM, Vol.3, A.6	Miscellaneous Data

The L1 VMM can set an L2 execution deadline, in absolute virtual TSC units. If the L2 VCPU is running when the deadline time arrives, it exits to the L1 VMM. The TSC deadline is set using TDVPS.TSC_DEADLINE[VM] using TDG.VP.WR. This field has similar semantics to the IA32_TSC_DEADLINE MSR. It provides a similar functionality as the MSR but does not emulate its exact MSR behavior.

Internally, the TDX module uses the L2 VMCS' VMX-preemption timer fields; thus, VMX-preemption timer is not available directly to the L1 VMM.

On return from TDG.VP.ENTER, the provided exit reason is VMX-preemption timer expired (52).

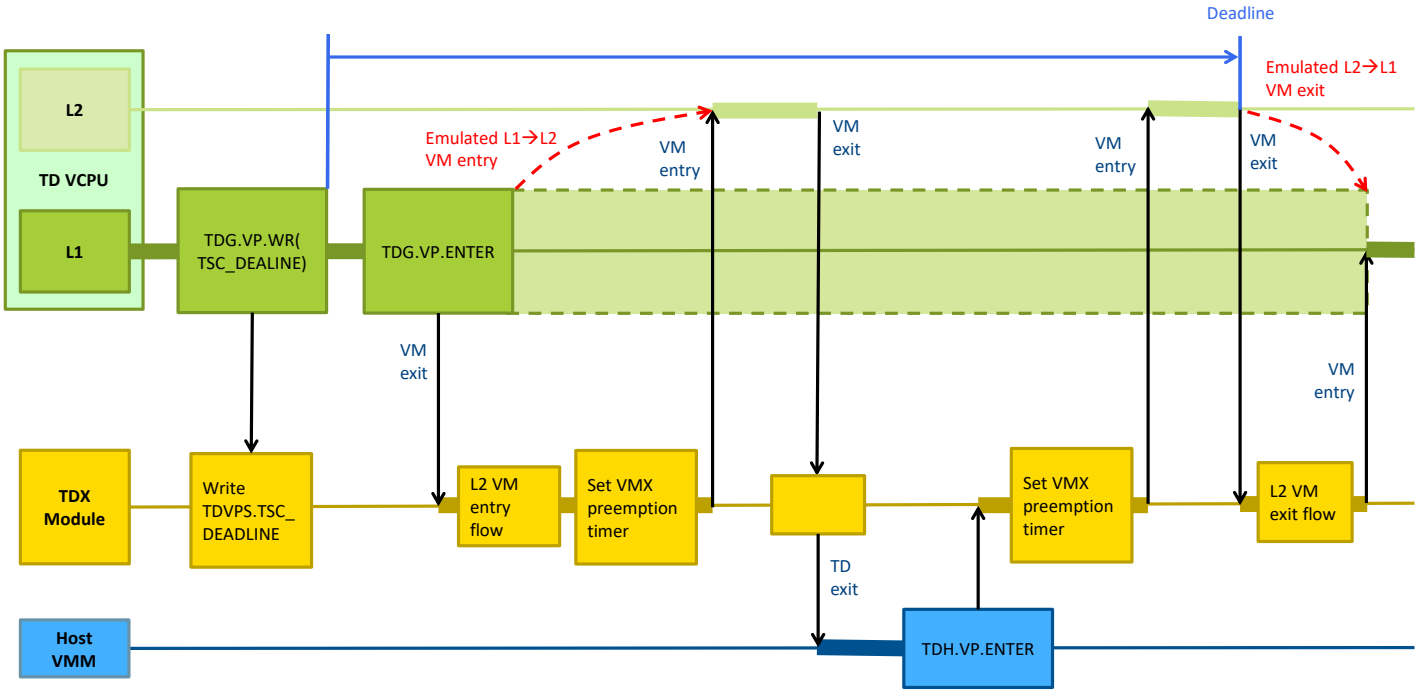

Figure 23.4: TSC Deadline Handling

Details

- Setting TSC_DEADLINE to -1 disables its operation.
- Setting TSC_DEADLINE to a value lower than the current virtual TSC value (e.g., 0) causes immediate L2→L1 VM exit after L1→L2 VM entry.
- If on L2 VM entry the TSC deadline is in the past (e.g., in the example above, the deadline passed while the host VMM was executing), VM entry into L2 will immediately result in a VM exit and the TDX module will emulate an L2→L1 exit.

23.14. Supervisor Protection Keys (PKS)

By design, guest TD usage of Supervisor Protection Keys (PKS) is controlled by the ATTRIBUTES.PKS bit. When PKS is supported by the CPU and ATTRIBUTES.PKS is set to 1, the L1 VMM may enable PKS for its L2 VMs as follows:

- Virtualize CPUID to enumerate PKS availability to the L2 VM.
- Enable L2 VM setting CR4.PKS flag by setting the L2 VMCS' "CR4 guest/host mask" and "CR4 read shadow" fields.
- Allow L2 VM access to the IA32_PKRS MSR by setting the L2 MSR exit bitmaps. The TDX module sets L2 VMCS "load guest PKRS" control to context-switch the IA32_PKRS MSR.

23.15. Intel® Total Memory Encryption (Intel® TME) and Multi-Key Total Memory Encryption (MKTME)

Guest TDs may not directly use the Intel TME and MKTME MSRs and the PCONFIG instruction. The Intel TDX module supports para-virtualization of this ISA for L2 VMs, as described below.

23.15.1. TME Virtualization

TME is enumerated by CPUID(0x7, 0x0).ECX[13]. The host VMM can configure the virtualization of this bit to the whole TD as enabled or disabled on TDH.MNG.INIT.

The L1 VMM can virtualize CPUID(0x7, 0x0).ECX[13] to enumerate availability of TME to an L2 VM. If TME is virtualized as enabled, then L2 VM access to those MSRs causes an L2→L1 VM exit.

23.15.2. MKTME Virtualization

MKTME is enumerated by CPUID(0x7, 0x0).EDX[18]. The host VMM can configure the virtualization of this bit to the whole TD as enabled or disabled on TDH.MNG.INIT. In both cases, MKTME is not actually enabled to the TD; the difference is in enabling para-virtualization of MKTME.

The L1 VMM can virtualize CPUID(0x7, 0x0).EDX[18] to enumerate availability of MKTME to an L2 VM. If MKTME is virtualized as enabled, the TDX module supports para-virtualization as follows. The following operations cause an L2→L1 VM exit:

- L2 VM access to the IA32_MKTME_PARTITIONING MSR (0x87)
- PCONFIG execution by the L2 VM

If MKTME is virtualized as disabled, then:

- Guest TD access to the IA32_MKTME_PARTITIONING MSR (0x87) causes an L2→L1 VM exit.
- PCONFIG execution by the L2 VM causes a #UD.

23.16. Management of Idle and Blocked Conditions

Intel SDM, Vol. 3, 9.10 Management of Idle and Blocked Conditions

23.16.1. HLT Instruction

HLT executed by an L2 VM results in an L2→L1 exit.

23.16.2. PAUSE Instruction and PAUSE-Loop Exiting

Intel SDM, Vol. 3, 25.1.3 Instructions That Cause VM Exits Conditionally

L2 VMs can execute PAUSE. The L1 VMM can set up PAUSE-loop exiting by using the L2 VMCS "PAUSE-loop exiting", "PLE_Gap" and "PLE_Window" VM-execution controls. A PAUSE-loop VM exit from the L2 VM results in an L2→L1 VM exit.

23.16.3. MONITOR and MWAIT Instructions

By default, guest TDs are expected not to use MONITOR/MWAIT. The virtual value of CPUID(1).ECX[3] is, by default, 0. Execution of MONITOR or MWAIT by an L2 VM results in an L2→L1 exit.

However, the host VMM may configure the guest TD to allow MONITOR/MWAIT, using the CPUID configuration table which is part the TD_PARAMS input to TDH.MNG.INIT. Configuring the virtual value of CPUID(1).ECX[3] to 1 also enables the TD (and its L2 VMs) to execute MONITOR and MWAIT.

23.16.4. WAITPKG: TPAUSE, UMONITOR and UMWAIT Instructions

Intel SDM, Vol. 3, 26.1.3 Instructions That Cause VM Exits Conditionally

As described in the [Base Spec], the host VMM may allow guest TDs to use the TPAUSE, UMONITOR and UMWAIT instructions, if the CPU supports them, by configuring the virtual value of CPUID(7,0).ECX[5] (WAITPKG).

If enabled, the L1 VMM has the following control over L2 VM execution:

- The L1 VMM may control L2 VMs' execution of TPAUSE, UMONITOR and UMWAIT instructions by writing to the L2 VMCS' secondary processor-based execution controls "enable user-level wait and pause" bit (26).
- The L1 VMM may also configure an L2→L1 exit on those instructions using the L2 VMCS' primary processor-based execution controls "RDTSC exiting" bit (12).
- The L1 VMM may configure L2→L1 exit on read and/or write of IA32_UMWAIT_CONTROL (MSR 0xE1).

23.17. Other Changes in TDX Non-Root Mode

23.17.1. L2 VM Tasking

Any task switch results in an L2→L1 VM exit. The L1 VMM can handle the task switch as it requires.

23.17.2. L2 VM PAUSE-Loop Exiting

Intel SDM, Vol. 3, 25.1.3 Instructions That Cause VM Exits Conditionally

The L1 VMM can set up PAUSE-loop exiting by using the L2 VMCS “PAUSE-loop exiting”, “PLE_Gap” and “PLE_Window” VM-execution controls. A PAUSE-loop VM exit from the L2 VM results in an L2→L1 VM exit.

24. L2 VM Debug and Profiling Architecture

This chapter discusses how the Intel TDX module debug architecture is extended to support TD Partitioning. The extended debug architecture includes the following debug facilities:

On-L2 VM Debug: Facilities for debugging an L2 VM using software that runs inside that VM

L1 VMM Debug of L2 VM: Facilities for debugging an L2 VM using software that runs inside the L1 VMM

Off-TD Debug of L2 VM: Facilities for debugging an L2 VM, which resides in a TD configured in debug mode, using software that runs outside the TD

24.1. On-L2 VM Debug

Intel SDM, Vol. 3, 17 Debug, Branch Profile, TSC and Intel Resource Director Technology (Intel RDT) Features

24.1.1. Overview

On-VM debug is the normal mode used to debug an L2 VM within a TD. A debug agent resides inside the L2 VM, and it can interact with external entities (e.g., a debugger) via standard I/O interfaces. It can also interact with other entities (e.g., a debugger) running in the L1 VMM or another L2 VM within the TD. The Intel TDX module is designed to virtualize and isolate TD debug capabilities from the host VMM and other guest TDs or legacy VMs. On-VM debug can be used for production or debug TDs – i.e., regardless of the guest TD's ATTRIBUTES.DEBUG state.

As described in the [TDX Module Base FAS], guest TDs are allowed to use almost all architectural debug and tracing features supported by the processor. This applies to L2 VMs as well. The L1 VMM may restrict the enumeration or enabling of debug and trace features to L2 VMs, using the standard VMX capabilities available to it.

24.1.2. Generic Debug Handling

24.1.2.1. *Context Switch*

Context-switch for TD exits directly from an L2 VM and TD entry back to an L2 VM is handled the same way as context switch for the TD as a whole, as described in the [TDX Module Base FAS].

24.1.2.2. *IA32_DEBUGCTL MSR Virtualization*

IA32_DEBUGCTL MSR is virtualized for L2 VMs similarly to how all L2 MSRs are virtualized, as described in 23.8.

24.1.3. Debug Feature-Specific Handling

Table 24.1: Debug Feature-Specific Handling

Debug Feature	How the Feature is Controlled	Handling
Hardware Breakpoints	• DR7, DR0-3 and DR6	By default, no special handling. L1 VMM may configure L2→L1 VM exit on DR access. DR7 is context-switched on L2 VM exit/entry.
General Detect	• DR7 bit 13 (GD)	No special handling: DR7 is context-switched on L2 VM entry/exit.
TSX Debug	• DR7 bit 11 (RTM) • IA32_DEBUGCTL bit 15 (RTM)	No special handling: DR7 and IA32_DEBUGCTL are context-switched on L2 VM entry/exit.
Single Stepping	• RFLAGS bits 18 (Trap Flag) and 16 (Resume Flag) • IA32_DEBUGCTL bit 1 (BTF)	No special handling: RFLAGS and IA32_DEBUGCTL are context-switched on L2 VM entry/exit.
Bus-Lock Detection	• IA32_DEBUGCTL bit 2 (BUS_LOCK_DETECT)	No special handling: IA32_DEBUGCTL is context-switched on L2 VM entry/exit.
Software Breakpoints (INT3)	None	No special handling: software breakpoints are stateless.

Debug Feature	How the Feature is Controlled	Handling
Branch Trace Message (BTM)	• IA32_DEBUGCTL bits 6 (TR) and 7 (BTS)	Not allowed: when an L2 VM attempts to set IA32_DEBUGCTL[7:6] to 0x1, the Intel TDX module causes an L2→L1 VM exit. In debug mode (ATTRIBUTES.DEBUG == 1), the host VMM is allowed to activate BTM by setting the above bits to 0x1.
Branch Trace Store (BTS)	• IA32_DEBUGCTL bits 6 (TR), 7 (BTS), 8 (BTINT), 9 (BTS_OFF_OS) and 10 (BTS_OFF_USR)	IA32_DEBUGCTL is context-switched on L2 VM entry/exit. IA32_DS_AREA needs to be context-switched by the L1 VMM. **Note:** • The L2 VM can configure BTS to raise PMI on buffer overflow (by setting BTINT = 1). However, since PMIs are virtualized by the host VMM, the L2 VM and L1 VMM should be ready to handle spurious, delayed and dropped PMIs. See Perfmon discussion in 24.2 below. • BTS may allow the L2 VM to hang the machine if BTS record generation causes a #PF or a #GP(0), because the act of getting to the exception handler may deliver another BTS. As described in the [TDX Module Base FAS], it is highly recommended that the host VMM enables notification TD exit.
Processor Trace (PT)	• IA32_RTIT_CONTROL • Requires VMM's consent on TD initialization by setting TD_PARAMS.XFAM[8] to 1 • L1 VMM can configure IA32_RTIT_CONTROL virtualization by requesting L2→L1 VM exit on MSR read and/or write	PT state handling as part of the extended feature set state is discussed in 23.6 and in the [TDX Module Base Spec]. The Intel TDX module sets the following control bits the L2 VMCSs. Since the setting is essential to TD and TDX module security, the L1 VMM can't modify them. • "Conceal VMX from PT" exit control, so PIP and VMCS packets are not generated on L2 VM exit • "Conceal VMX from PT" entry control, so PIP and VMCS packets are not generated on L2 VM entry • "Conceal VMX from PT" execution control, so PIP packets do not report "non-root", and PSB+ sequences do not include VMCS packets, during L2 VM execution • "PT2GPA" execution control, so PT addresses are treated as GPAs
Architectural Last Branch Records (LBRs)	• IA32_LBR_CONTROL • Requires VMM's consent on TD initialization by setting TD_PARAMS.XFAM[15] to 1 • L1 VMM can configure IA32_LBR_CONTROL virtualization by requesting L2→L1 VM exit on MSR read and/or write	
Non-Architectural LBRs	• IA32_DEBUGCTL bit 0 (LBR)	L2 VM attempt to set IA32_DEBUGCTL[0] is ignored by the CPU.

24.2. On-L2 VM Performance Monitoring

The host VMM controls whether a guest TD can use the performance monitoring ISA using the TD's ATTRIBUTES.PERFMON bit. If performance monitoring is allowed for the TD, the L1 VMM controls whether performance monitoring is allowed for each L2 VM, by controlling the virtualization of the applicable MSR. For details, see the [TDX Module Base FAS].

24.3. L1 VMM Debug of L2 VMs

L1 VMM is in the TCB of its L2 VMs, and thus have access to the VMX debug features as they are enabled for the whole TD. The L1 VMM has access to some of the L2 VMs' VMCSes control fields. For many execution controls, L1 VMM access is read-only, because their value is essential for TD and TDX module security and correct operation. For details, see the [TDX Module ABI Spec].

24.4. Off-TD Debug of L2 VMs

As described in the [TDX Module Base Spec], a guest TD is defined as **debuggable** if its ATTRIBUTES.DEBUG bit is 1. In this mode, the host VMM can use Intel TDX functions to read and modify TD VCPU state and TD private memory, which is not accessible when the TD is non-debuggable.

This definition directly extends to nested L2 VMs, as described in the table below.

Table 24.2: Off-TD Debug Interface Extensions for TD Partitioning

Intel TDX Function	Extensions for TD Partitioning
TDH.MNG.RD **TDH.MNG.WR**	Access L2 VM metadata
TDH.MEM.SEPT.RD	Read L2 Secure EPT entry
TDH.VP.RD **TDH.VP.WR**	Access TD VCPU's L2 state in TDVPS (including L2 VMCS)

24.4.1. L2 VM Debug Controls Used by the Host VMM

The host VMM can control L2 VM off-TD debug using the TDVPS L2_DEBUG_CTLS field, which is defined below. This field may only be written by the host VMM if the TD is debuggable.

Table 24.3: L2_DEBUG_CTLS: L2 VM Debug Controls

Bit(s)	Name	Description
0	**TD_EXIT_ON_L1_TO_L2**	If set, TDG.VP.ENTER will TD-exit instead of entering an L2 VM. For simplicity, TD exit will happen before the L1-->L2 transition is done. TDH.VP.ENTER's completion status will be TDX_TD_EXIT_BEFORE_L2_ENTRY. This TD exit is fault-like, which allows the debugger to modify whatever it wishes and TD-enter to re-invoke TDG.VP.ENTER.
1	**TD_EXIT_ON_L2_TO_L1**	If set, VM exit from L2 which results in an L2→L1 exit is mutated into a TD exit. TDH.VP.ENTER's completion status will be TDX_TD_EXIT_ON_L2_TO_L1.
2	**TD_EXIT_ON_L2_VM_EXIT**	All L2 VM exits (except fatal errors) will result in a TD exit. TDH.VP.ENTER's completion status will be TDX_TD_EXIT_ON_L2_VM_EXIT.
63:3	**RESERVED**	Reserved, must be 0

24.4.2. Host VMM Access of Debuggable TD's L2 VM State, Controls and Memory

When the TD is debuggable, the off-TD debugger can:

- Read and modify L2 VMCS fields that contain guest state, VM entry load controls, VM exit save controls, and VM execution controls.
- Read and modify TDVPS fields that contain additional TD VCPU's L2 VM state and control.

This may cause the next VM entry into the L2 VM to fail due to bad guest state. It may also generate VM exits that wouldn't have happened otherwise (e.g., VM exit due to a #PF within the L2 VM, even if not configured so by the L1 VMM). By default, such VM exits from L2 are routed to the L1 VMM as L2→L1 VM entries. However, the L1 VMM may not expect such VM entries. To avoid unexpected L2→L1 VM entries, the host VMM can set TDVPS.L2_DEBUG_CTLS.TD_EXIT_ON_L1_TO_L2. With this bit set, any L2→L1 exit is mutated into a TD exit; the host VMM may, on the following TD entry, control whether the L2 VM or the L1 VMM is resumed, using the RESUME_L1 input flag of TDH.VP.ENTER.

25. Guest-Side TDX Functions (TDCALL) for L2 VMs

A subset of the guest-side (TDCALL) TDX functions can be called by L2 VMs, depending on configuration by the L1 VMM. If L2 call a function that is not allowed, it results in an L2→L1 exit.

Currently, the only guest-side TDX function that can be called by L2 VMs is TDG.VP.VMCALL.

25.1. TDG.VP.VMCALL

By default, L2 VMs can't call TDG.VP.VMCALL. The L1 VMM may enable TDG.VP.VMCALL usage by a specific L2 VCPU setting TDVPS.L2_CTLS[VM].ENABLE_TDVMCALL using TDG.VP.WR. The behavior is described in 22.2.3 and 22.2.4.