Intel® TDX Module Interrupt Virtualization Architecture Specification

DRAFT FOR COMMUNITY REVIEW – WORK IN PROGRESS

366830-002US (draft 0.01)

April 2026

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

1. About this Document

1.1. Scope of this Document

This document describes the Intel® Trust Domain Extensions (Intel® TDX) module's architecture for interrupt virtualization.

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

Table 1.1: 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 Interrupt Virtualization Architecture Specification**	[Interrupt Virtualization Spec]	Architecture overview and specification for interrupt virtualization
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 **TD Migration ABI Reference Specification**	[TD Migration ABI Spec]	Detailed TDX Module Application Binary Interface (ABI) reference specification, covering the TD Migration 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 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 updating this document in real time when such changes occur.

1.2. Glossary

See the [TDX Module Base Spec].

1.3. Notation

See the [TDX Module Base Spec].

1.4. References

See the [TDX Module Base Spec].

2. Overview

2.1. Legacy vs. Enhanced Interrupt Virtualization

All TDX Module releases support the following interrupt virtualization functionality:

- Virtual interrupts may only be posted to L1.
- Sending Inter-Processor Interrupts (IPIs) by a guest TD VCPU to other VCPUs is not directly supported.

TDX Modules which support enhanced interrupt virtualization have the following additional functionality:

- Virtual interrupts may be posted to L1 and L2 VMs.
- Virtual interrupts posted by untrusted entities, such as the host VMM or IOMMU, are filtered by a TD-configurable allowed vector mask.
- Guest TD VCPUs may send secure IPIs to other VCPUs. This functionality is supported for both L1 and L2 VM.

Support of enhanced interrupt virtualization is enumerated by TDX_FEATURES0.ENHANCED_INTR_VIRTUALIZATION (bit 45), readable by TDH.SYS.RD*.

Unreleased Feature:	At the time of writing of this document, Enhanced Interrupt Virtualization has not been released yet. Details provided in this document serve as a preview and are subject to change.

2.2. Components Participating in Interrupt Virtualization

2.2.1. CPU

Intel SDM, Vol. 3, Ch. 31 APIC Virtualization and Virtual Interrupts

TDX interrupts virtualization is based on the following x86 ISA features:

Posted Interrupts: Posting virtual interrupts to a guest VM's VCPU by the host VMM or by the IOMMU.

IPI Virtualization: Posting virtual interrupts to a guest VM's VCPU by another VCPU of the same VM.

In most cases, the CPU's ISA support, once configuration is done, works well and results in fast and efficient operation. However, there are multiple edge cases which require the help of the host VMM, of the TD's L1 (to support L2 operation), and of the TDX Module.

2.2.2. Host VMM

The host VMM is responsible for the following:

- Configuring the TD for virtual interrupt handling.
- If enhanced interrupt virtualization is supported, the host VMM is responsible for handling cases where the destination VCPU of a posted interrupt is not running. The host VMM gets notified so it can schedule the destination VCPU to run and handle the interrupt.

2.2.3. L1 VMM

If enhanced interrupt virtualization is supported, then for partitioned TDs, L1 is responsible for the following:

- Configuring L2 for virtual interrupt handling.
- Handling cases where the destination VCPU is running, but not in the L2 VM which is the destination of a posted interrupt. L1 gets notified so it can schedule the destination L2 to run and handle the interrupt.

2.2.4. TDX Module

The TDX Module is responsible for the following:

- Setting up the proper data structures to enable CPU support.
- Handling all the edge cases.
- Emulating any functionality which is not supported by the CPU, such as interrupt vector filtering.

2.3. Data Structures Overview

Intel SDM, Vol. 3, 31.1.6.1 Virtual-Interrupt Posting
VT-d Spec, 9.11 Posted Interrupt Descriptor (PID)

This section provides an overview of data structures used for interrupt virtualization. Many of those data structures are not directly accessible by the host VMM or the guest TD. However, understanding their usage is important for properly supporting interrupt virtualization.

2.3.1. Background: Posted-Interrupt Descriptor (PID)

Posted-Interrupt Descriptor (PID) is a data structure supported by the CPU and IOMMU to enable posting virtual interrupts to guests.

Table 2.1: Format of Posted-Interrupt Descriptor (PID)

Bit Position(s)	Name	Description
255:0	Posted-interrupt requests (PIR)	One bit for each interrupt vector. There is a posted-interrupt request for a vector if the corresponding bit is 1.
256	Outstanding notification (ON)	If this bit is set, there is a notification outstanding for one or more posted interrupts in bits 255:0.
257	Suppress notify (SN)	Setting this bit directs agents not to send notifications.
271:258	Reserved	Reserved.
279:272	Notify vector (NV)	Notifications will use this vector.
287:280	Reserved	Reserved.
319:288	Notify destination (NDST)	Notifications will be directed to this physical APIC ID.
511:320	Reserved	Reserved.

2.3.2. Interrupt Virtualization Data Structures

The following diagram shows the data structures used by the TDX Module for virtualizing interrupts in legacy mode.

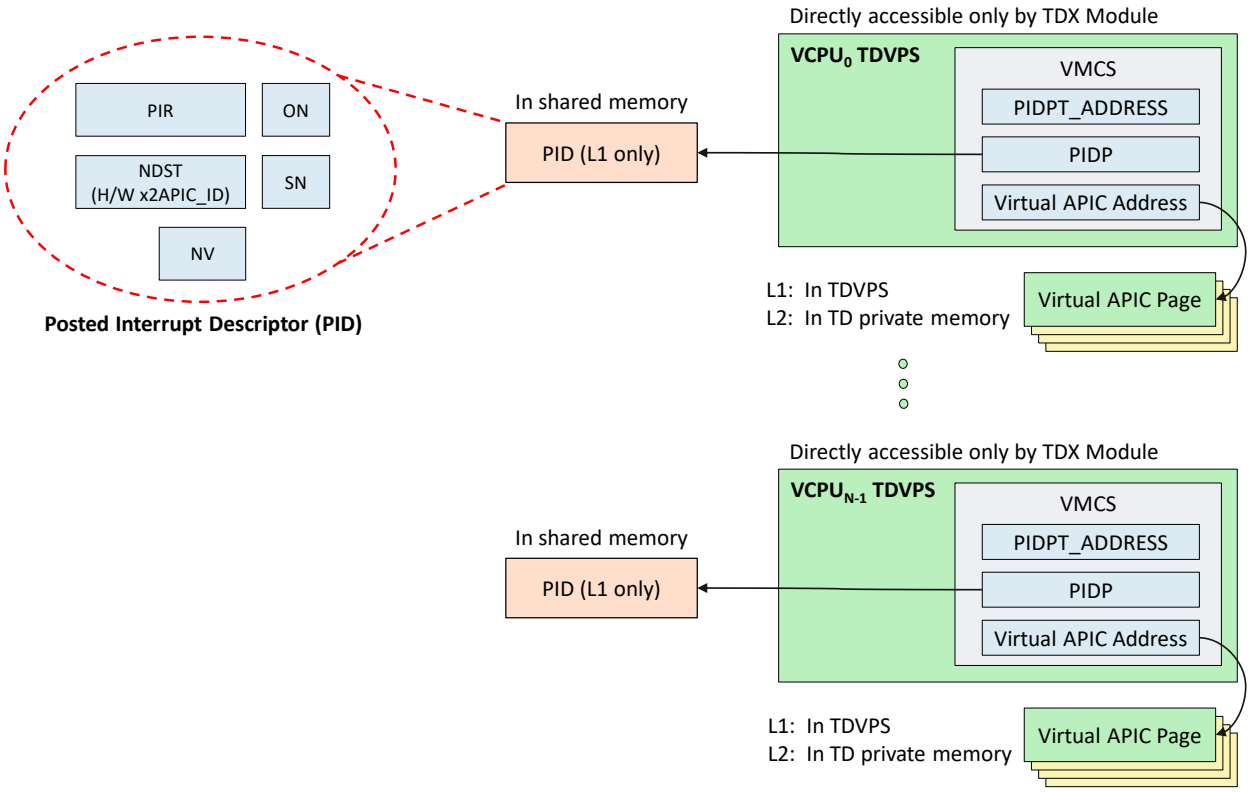

Figure 2.1: Data Structures Used for Legacy Interrupt Virtualization

The following diagram shows the data structures used by the TDX Module for virtualizing interrupts in enhanced mode.

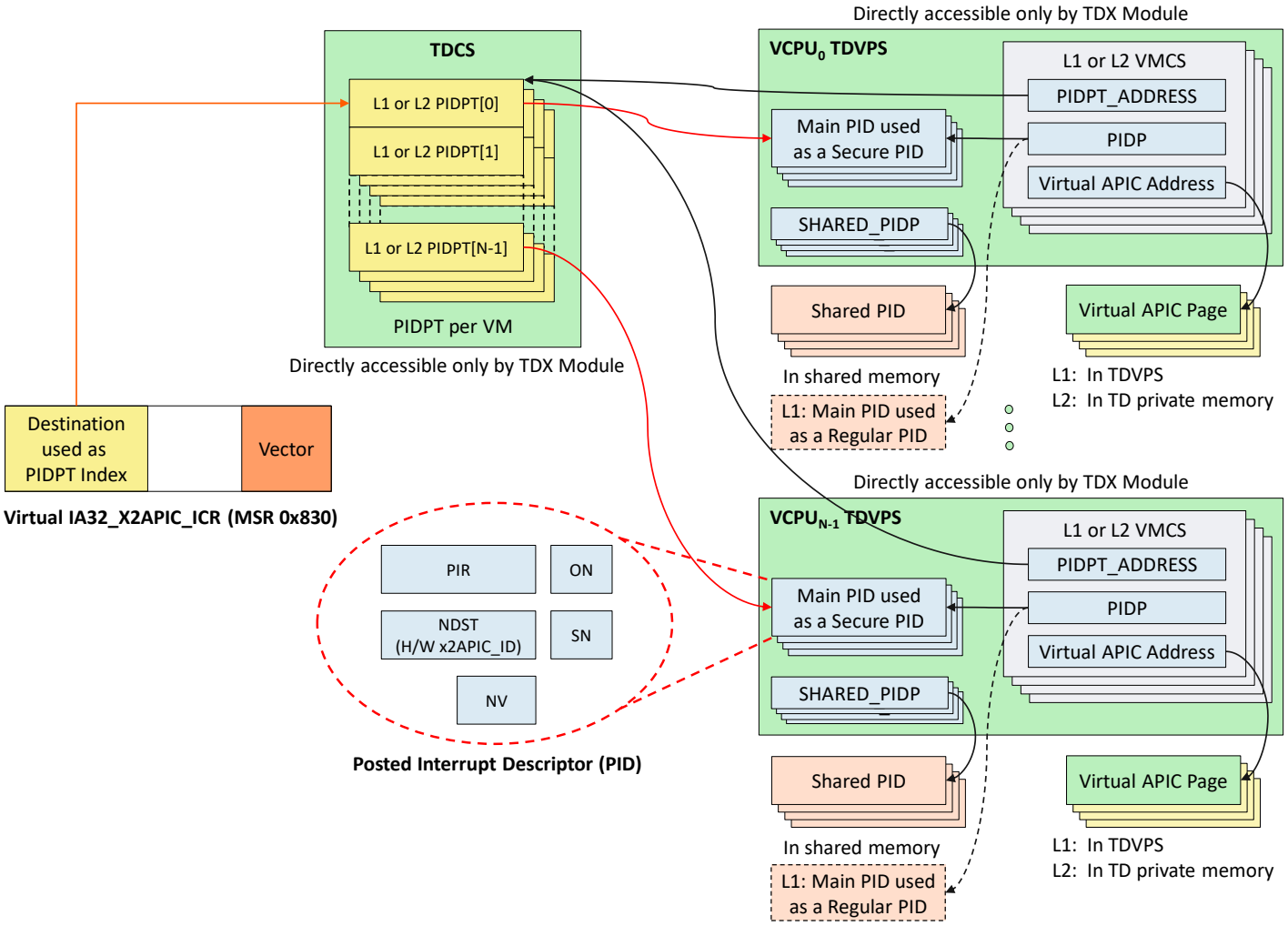

Figure 2.2: Data Structures Used for Enhanced Interrupt Virtualization

5

Table 2.2: Data Structures Used for Interrupt Virtualization

Short Name	Full Name	Support	Description	Direct Access by
VAPIC Page	Virtual APIC Page	Legacy	Holds the virtual APIC state. Used by the CPU, TDX Module and (for L2) the L1 VMM to virtualize APIC operation.	For L1: TDX Module For L2: L1
VAPIC Page Address	Virtual APIC Page Address	Legacy	VMCS field: HPA of VAPIC Page, used by the CPU	TDX Module
PID	Posted Interrupt Descriptor	Legacy	Used by the s/w and h/w to post interrupts to a VM. See PID types used in TDX below.	Depending on configured usage (see below)
Main PID	Main Posted Interrupt Descriptor	Legacy	The architectural PID, used by the CPU. May be configured as a Regular PID (in shared memory) or a Secure PID. See below.	Depending on configured usage (see below)
Regular PID	Regular Posted Interrupt Descriptor	Legacy	The architectural PID, used by the CPU, configured to reside in shared memory	Any s/w, IOMMU
Secure PID	Secure Posted Interrupt Descriptor	Enhanced	The architectural PID, used by the CPU, configured to reside in TDVPS	TDX Module
Shared PID	Shared Posted Interrupt Descriptor	Enhanced	A PID residing in shared memory, where interrupt vectors are filtered using PIR_MASK	Any s/w, IOMMU

Short Name	Full Name	Support	Description	Direct Access by
PIDP	PID Pointer	Enhanced	VMCS field: HPA of the Main PID, used by the CPU	TDX Module
SHARED_PIDP	Shared PID Pointer	Enhanced	HPA of the Shared PID, used by the TDX Module	TDX Module
PIDPT	PID Pointer Table	Enhanced	Contiguous table of PID pointers, used by the CPU to generate an Inter-Processor Interrupt when the ICR is written by s/w	TDX Module
PIDPT_ADDRESS	PID Pointer Table Address	Enhanced	VMCS field: HPA of PIDPT, used by the CPU	TDX Module

2.4. Dual PID

Intel SDM, Vol. 3, 31.1.6.1 Virtual-Interrupt Posting

With enhanced interrupt virtualization, TDX defines the concept of **Dual PID**. The CPU-supported PID is known as the Main PID, while a second PID, called **Shared PID**, is supported by the TDX Module.

2.4.1. Main (Legacy) PID and its Use as Secure PID

The Main PID is the architectural PID, as defined in the [Intel SDM] and directly supported by the CPU.

The Main PID can be configured as the Secure PID. In this case, it resides in memory protected by the TD's private HKID, as part of the TDVPS control structure, and is only directly accessible to the TDX Module.

2.4.2. Shared PID

The Shared PID resides in shared memory and is directly accessible to any component in the platform. It is similar in definition and functionality to the Main PID, except that interrupt vectors are filtered using a bit mask called PIR_MASK. The mask is controlled by the TDX Module.

Shared PID functionality is not directly supported by the CPU; it is implemented by the TDX Module.

2.4.3. Posted Interrupt Processing with Dual PID

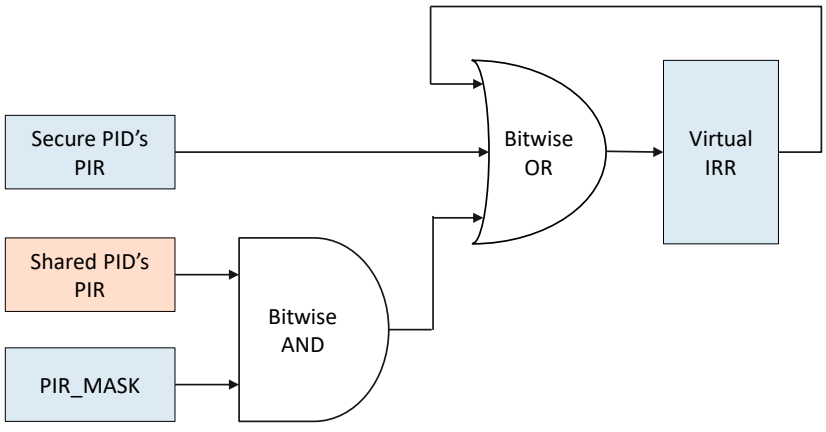

Figure 2.3: Posted Interrupt Processing with Secure and Shared PID

The concept of posted interrupt processing with dual PID extends the architectural definition, as specified in the [Intel SDM, vol. 3, 31.6]. The following text shows the original SDM text (in normal font) and the additional processing (in italics):

…

"3. Clear *Secure*PID.ON. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation)."

3'. If Shared PID is enabled, clear SharedPID.ON. This is done atomically so as to leave the remainder of the descriptor unmodified (e.g., with a locked AND operation).

…

"5. Do a logical-OR of *Secure*PID.PIR into VIRR and clear *Secure*PID.PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared."

5'. If Shared PID is enabled, do a logical-AND of SharedPID.PIR with PIR_MASK, then do a logical-OR of the result into VIRR, and clear ShraredPID.PIR. No other agent can read or write a PIR bit (or group of bits) between the time it is read (to determine what to OR into VIRR) and when it is cleared.

…

3. Guest TD Perspective: Common

This chapter describes common aspects of virtual interrupt handling from the point of view of the guest TD.

3.1. APIC Virtualization

Intel SDM, Vol. 3, 24.6.8	Controls for APIC Virtualization
Intel SDM, Vol. 3, 31	APIC Virtualization and Virtual Interrupts

This section describes aspects of APIC virtualization that are common to L1 and to L2 VM.

3.1.1. Virtual APIC Mode

- Guest TDs must use x2APIC mode. Legacy xAPIC mode (using memory mapped APIC access) is not allowed. This is applicable for both L1 and L2 VMs. If required, L1 may emulate a virtual xAPIC for L2.
- The guest TD cannot disable the APIC. For L1, attempts to RDMSR or WRMSR the IA32_APIC_BASE MSR cause a #VE(NON_CONFIG_PARAVIRT) to L1. For L2, similar attempts result in an L2→L1 exit.

3.1.2. Virtual APIC Access by Guest TD

Intel SDM, Vol. 3, 31.5	Virtualizing MSR-Based APIC Access

Guest TDs are allowed access to a subset of the virtual APIC registers, which are virtualized by the CPU as described in [Intel SDM, Vol. 3, 31.5] and by the TDX Module.

- For L1, access to other APIC registers can cause a #VE; the L1's #VE handler can paravirtualize the register behavior, e.g., by using a software protocol over TDG.VP.VMCALL to request operations from the host VMM.
- For L2, access to other APIC registers can cause an L2→L1 exit. L1 may emulate the required behavior. In addition, L1 may configure the L2 MSR exit bitmaps to cause an L2→L1 exit on any desired MSR.

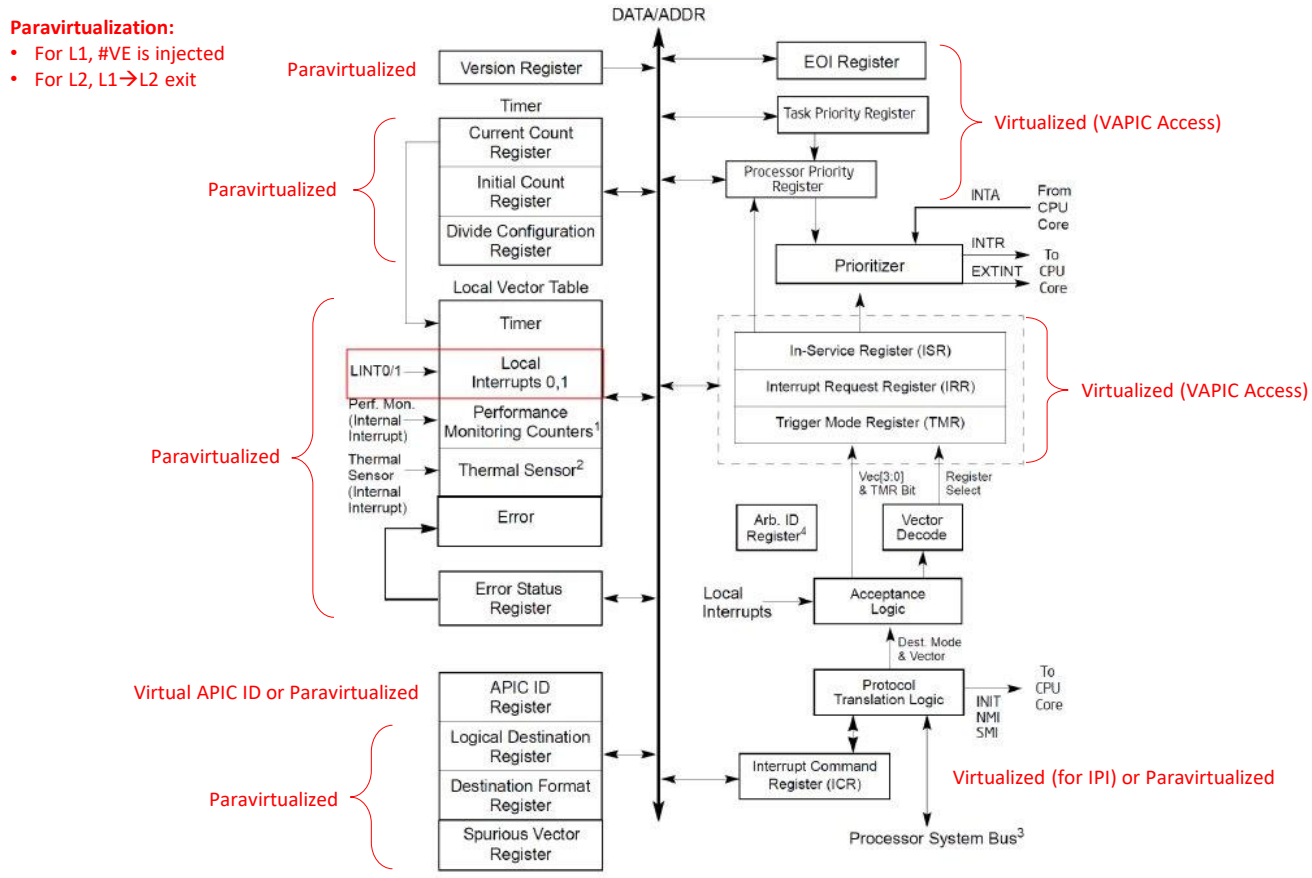

Figure 3.1: Virtual APIC Access by Guest TD

The table below shows how x2APIC MSRs are handled for L1. For L2 VMs, handling is similar except that #VE is replaced by an L2→L1 exit.

Table 3.1: x2APIC MSRs (0x800 – 0x8FF) Processing for L1

MSR Range	MSR Name(s)	Description	Access	On RDMSR	On WRMSR
0x802	IA32_X2APIC_APICID	APIC ID	RO	Virtualized (if TD_CTLS.ENUM_TOPOLOGY is set) or #VE(CONFIG_PARAVIRT)	#GP(0)
0x803	IA32_X2APIC_VERSION	APIC Version	RO	#VE(NON_CONFIG_PARAVIRT)	#GP(0) or #VE[1]
0x808	IA32_X2APIC_TPR	Task Priority	RW	Virtualized (read from VAPIC page)	Virtualized (write to VAPIC page, TPR virtualization)
0x80A	IA32_X2APIC_PPR	Processor Priority	RO	Virtualized (read from VAPIC page)	#GP(0)
0x80B	IA32_X2APIC_EOI	End Of Interrupt	WO	Virtualized (read from VAPIC page)	Virtualized (write to VAPIC page, EOI virtualization)
0x80D	IA32_X2APIC_LDR	Local Destination	RO	#VE(NON_CONFIG_PARAVIRT)	#GP(0) or #VE[1]
0x80F	IA32_X2APIC_SIVR	Spurious Interrupt Vector	RW	#VE(NON_CONFIG_PARAVIRT)	#VE
0x810-0x817	IA32_X2APIC_ISR0-IA32_X2APIC_ISR7	In-Service	RO	Virtualized (read from VAPIC page)	#GP(0)
0x818-0x81F	IA32_X2APIC_TMR0-IA32_X2APIC_TMR7	Trigger Mode	RO	Virtualized (read from VAPIC page)	#GP(0)
0x820-0x827	IA32_X2APIC_IRR0-IA32_X2APIC_IRR7	Interrupt Request	RO	Virtualized (read from VAPIC page)	#GP(0)
0x828	IA32_X2APIC_ESR	Error Status	RW	#VE(NON_CONFIG_PARAVIRT)	#VE(NON_CONFIG_PARAVIRT)
0x830	IA32_X2APIC_ICR	Interrupt Command	RW	Virtualized (IPI, if supported and configured) or #VE(CONFIG_PARAVIRT)	Virtualized (IPI, if supported and configured) or #VE(CONFIG_PARAVIRT)
0x82F, 0x832-0x837, 0x83A	IA32_X2APIC_LVT_*	Local Vector Table	RW	#VE(NON_CONFIG_PARAVIRT)	#VE(NON_CONFIG_PARAVIRT)
0x838	IA32_X2APIC_INIT_COUNT	APIC Timer	RW	#VE(NON_CONFIG_PARAVIRT)	#VE(NON_CONFIG_PARAVIRT)
0x839	IA32_X2APIC_CUR_COUNT		RO	#VE(NON_CONFIG_PARAVIRT)	#GP(0) or #VE[1]
0x83E	IA32_X2APIC_DIV_CONF		RW	#VE(NON_CONFIG_PARAVIRT)	#VE(NON_CONFIG_PARAVIRT)
0x83F	IA32_X2APIC_SELF_IPI	Self IPI	WO	Virtualized (read from VAPIC page)	Virtualized (write to VAPIC page, self-IPI virtualization)
0x83B-0x83D	Reserved	N/A	None	#GP(0) or #VE[1]	#GP(0) or #VE[1]
Other	Reserved	N/A	None	#GP(0)	#GP(0)

[1] If the TDX Module supports #VE reduction, as enumerated by TDX_FEATTURES0.VE_REDUCTION (bit 30), then this MSR access results in a #GP(0). Else, it results in a #VE.

4. Guest TD's L1 Operation as a Guest (Including Non-Partitioned TD)

This chapter describes the non-partitioned TD operation, or a partitioned TD's L1 operation as a guest. L1 operation as a VMM of L2 VMs is described in the following section.

Note: The following text uses the term "L1" to apply both to a non-partitioned TD and to a partitioned TD's L1. They are technically identical.

Unreleased Feature: At the time of writing of this document, Enhanced Interrupt Virtualization has not been released yet. Details provided in this document serve as a preview and are subject to change.

4.1. Feature Enumeration

Detecting TDX Module Support

TDX Module support is enumerated by the following TDX_FEATURES0 bits, readable using TDG.SYS.RD*. For details, see the [ABI Spec].

Table 4.1: Detecting TDX Module Support

Bit(s)	Name	Description
40	ENHANCED_INTR_STATE	The TDX Module supports additional interrupt state indications by TDVPS.VCPU_STATE_DETAILS and the IMM_RESUME_HINT output flag of TDH.VP.ENTER.
45	ENHANCED_INTR_VIRTUALIZATION	The TDX Module supports enhanced interrupt virtualization, including posted interrupt and IPI processing.
46	VEINFO_INTR_STATE	The TDX Module supports returning the guest interruptibility state by TDG.VP.VEINFO.GET.

4.2. L1-to-L1 IPI

If properly configured as discussed below, L1 can post inter-processer interrupts (IPIs) in the architectural way, by writing to its virtual APIC's interrupt command register (ICR), i.e., WRMSR(IA32_X2APIC_ICR).

4.2.1. Legacy Mode

Legacy L1-to-L1 IPI mode is used if one or more of the following is true:

- The TDX Module does not support IPI virtualization, or
- L1 has not been configured by the host VMM to support IPI virtualization, or
- L1 has not configured IPI destination indices (see 4.2.2.3 below), either because L1 is older and is not aware of TDX IPI virtualization, or for some other reason.

In this case, L1-to-L1 IPI can only be done indirectly, with host VMM support. L1 can request the host VMM, using a software-defined protocol over TDG.VP.VMCALL, to post interrupts to other VCPUs.

4.2.2.1.

4.2.2. Enhanced Mode: L1-to-L1 IPI Virtualization Configuration

The following text applies only when the TDX Module supports enhanced interrupt virtualization.

L1 must check and configure the following if it intends to use L1-to-L1 IPI.

Checking L1 IPI Virtualization Support and L1 PIDPT Size

As discussed in 2.3, PIDPT holds multiple entries indexed by an IPI destination index. PIDPT is allocated by the host VMM. L1 should read TDCS.PIDPT_NUM_ENTRIES[0], using TDG.VM.RD, to determine if the size of PIDPT allocated by the host VMM is sufficient for the desired L1 IPI destination index (see below) name space.

L1 posted interrupt configuration can be done on demand. A TDCS.PIDPT_NUM_ENTRIES[0] value of 0 indicates that IPI virtualization has not been configured by the host VMM for L1, and no PIDPT has been allocated. In that case, L1 may

request the host VMM (using a software-defined protocol over TDG.VP.VMCALL) to configure enhanced posted interrupt mode and allocate PIDPT.

Checking L1 Secure PID

Secure PID is used for L1-to-L1 IPIs, since it prevents exposing L1 to untrusted agents that may post interrupts to L1 with the same vectors used by L1 for IPI. See 4.3 below for more information.

If TDCS.PIDPT_NUM_ENTRIES[0] checked above to be non-0, it implies that Secure PID is used for L1. In addition, L1 can directly check whether Secure PID has been configured by reading TDCS.PID_MODE[0], using TDG.VM.RD.

4.2.2.2. *L1 IPI Destination Index Configuration*

For sending IPIs, each VCPU should be assigned an L1 IPI destination index, which is the index of its entry in the PIDPT. The index is used as the destination field in the upper 32 bits when L1 posts an IPI by WRMSR(IA32_X2APIC_ICR). Each VCPU configures its own L1 IPI index by writing to TDVPS.PIDPT_INDEX[0], using TDG.VP.WR. The meaning of the destination index is chosen by L1; it may be the VCPU_INDEX, the L1 virtual x2APIC ID or any other value chosen by L1.

4.2.2.3.

The IPI destination index has the following characteristics:

- The index for each VCPU should be unique. However, the TDX Module does not enforce that.
- The index must fit in the PIDPT; it must be smaller than TDCS.PIDPT_NUM_ENTRIES[0].

Failing to configure a destination index will result in #VE, with an APIC Write exit reason, when L1 attempts to send an IPI using an unconfigured destination index value. See 4.4.2.2 for details.

4.2.3. Enhanced Mode: Posting an L1-to-L1 IPI

If IPI virtualization is supported and has been properly configured as described above, L1 can post inter-processer interrupts (IPIs) in the architectural way, by writing to its virtual APIC's interrupt command register (ICR), i.e., WRMSR(IA32_X2APIC_ICR).

To post an IPI, the following conditions should be met:

- IPI virtualization must have been configured by the host VMM, thus Secure PID may have been configured.
- IPI destination index must have been configured by L1.

To post an interrupt to another VCPU, L1 executes WRMSR(IA32_X2APIC_ICR) with the following data:

- Destination (bits 63:32) is the IPI destination index, which has been configured as described above (e.g., virtual x2APIC ID).
- Bits 7:0 contain the interrupt vector. Allowed vector values are 31 to 255.
- All other bits must be 0.

Only simple unicast IPIs are directly supported by the TDX Module. More sophisticated modes (e.g., post to all excluding self) are not directly supported. If such modes are used by L1 when writing to IA32_XAPIC_API, it will result in a #VE; L1's #VE handler may emulate those modes, if required.

If IPI virtualization has not been enabled for L1, WRMSR(IA32_X2APIC_ICR) results in a #VE. This is the legacy behavior of TDX prior to introducing the enhanced interrupt virtualization feature. See 4.4.2.2 for details.

TDX does not protect against DOS; thus, IPI delivery is not guaranteed (e.g., the host VMM may prevent the destination VCPU from running). L1 is expected to check that the IPI has been delivered.

4.2.4. Enhanced Mode: Posting an L1-to-L1 User IPI

L1 can post user inter-processer interrupts (UIPIs) in the architectural way, by executing SENDUIPI. This feature uses a User Posted Interrupt Descriptor (UPID) table, which resides in the TD private memory and is configured by L1. When properly configured, SENDUIPI results in a virtual notification interrupt, sent as an IPI with the vector and destination values taken from the UPID.

From TDX perspective, this is just another IPI case; see the above section for details. Proper configuration of user interrupts, including UPID, is the responsibility of L1.

4.3. External Posted Interrupts and their Security Implications

4.3.1. Overview

Interrupts posted externally to the guest TD, i.e., by the host VMM or an IOMMU, use a PID in shared memory configured as either a Regular PID or a Shared PID.

A malicious host VMM or an IOMMU may post any virtual interrupt vector in the range 255:31 at any time. The guest TD should be able to process such interrupts without confusing it with a software interrupt that uses the same vector number.

To counter that risk, the following options are available:

- With legacy posted interrupt configuration, L1 needs to implement code to detect such attacks as part of its interrupt handler.
- With enhanced posted interrupt configuration and Shared PID, as described in 2.4.3, L1 can configure interrupt vector filtering for itself and for the L2 VMs.

L1 can determine if a Shared PID has been configured for itself by reading TDCS.PID_MODE[0], using TDG.VM.RD.

4.3.2. Detection of Illegal Posted Interrupt Vectors by the Interrupt Handler

For L1, if configured using the legacy mode, or if configured using the extended mode but with its Main PID as a Regular PID in shared memory (not as Shared PID), the L1 software should implement some protection against confusing posted interrupts with software interrupts. L1's interrupt handler for vector V, which expects a software interrupt, can read the virtual APIC's ISR register by reading the applicable IA32_X2APIC_ISRx MSRs (0x817:0x810). It can check that ISR[V] is indeed 0 for the specific vector.

4.3.3. Enhanced Mode: L1 Posted Interrupt Filtering Configuration: PIR_MASK

Interrupt filtering applies only to Shared PID, if supported and properly configured as indicated by TDCS.PID_MODE[0]. L1 may configure interrupt filtering by writing to TDVPS.PIR_MASK[0] using TDG.VM.WR. The PIR_MASK entry is a 256-bit mask, where each bit N enables the corresponding posted interrupt vector N. By default, all bits are 0 and no interrupt vector is allowed. Bits 30:0 can never be set.

4.4. L1-to-L1 IPI Paravirtualization: Handling IPI-Related #VE

This section describes multiple IPI-related cases where L1 needs to paravirtualize the required behavior.

4.4.1. #VE due to WRMSR(ICR) by L1

#VE with a basic VM exit reason of WRMSR (32) due to WRMSR(IA32_X2APIC_ICR) by L1 happens if IPI virtualization is not supported by the TDX Module or is not enabled for L1. See 4.2.2 above for details. L1 may paravirtualize IPIs with the cooperation of the host VMM.

4.4.2. #VE due to APIC Write by L1

Intel SDM, Vol.3, 31.5	Virtualizing MSR-Based APIC Accesses
Intel SDM, Vol.3, 31.7	Virtualizing SENDUIPI

Overview

#VE due to APIC write is trap-like; it happens after the L1 instruction causing it has executed. L1 can retrieve the information using TDG.VP.VEINFO.GET, which returns the offset of the write access in the VAPIC page in bits 11:0 of the exit qualification, and the data written to the Virtual APIC page at that offset.

#VE due to APIC Write (ICR) by L1

4.4.2.2.1. Legacy Mode

If IPI virtualization is not supported by the TDX Module, a #VE(NON_CONFIG_PARAVIRT) with a basic VM exit reason of APIC Write (56) may happen because of SENDUIPI execution by L1. L1 may request the host VMM, using a software-defined protocol over TDG.VP.VMCALL, to post an interrupt to the destination VCPU.

4.4.2.2.2. Enhanced Mode

If IPI virtualization is supported by the TDX Module, a #VE(CONFIG_PARAVIRT) with a basic VM exit reason of APIC Write (56) may happen because of WRMSR(IA32_X2APIC_ICR) or SENDUIPI execution by L1, due to improper configuration by L1 or due to L1 misbehavior. L1 should handle such #VE as follows:

1. If the exit qualification indicates that the updated VAPIC field is ICR (offset 0x300):
   1.1. Check the 64-bit write value retrieved by TDG.VP.VEINFO.GET:
      1.1.1. Bits 31:8 should be 0. If not, the #VE happened due to L1 WRMSR misbehavior or wrong configuration of UPID.
         Note: On WRMSR(IA32_X2APIC_ICR), a non-0 value of bits 31:20, 17:16, or 13 results in a #GP injection to L1; no APIC write takes place. See [Intel SDM, Vol.3, 31.5].
      1.1.2. Check if the vector value in bits 7:0 is lower than 16. If not, the #VE happened due to L1 WRMSR misbehavior or wrong configuration of UPID.
      1.1.3. Check if the destination value in bits 63:32 is lower than the number of PIDPT entries. If not, the #VE happened either due to improper TDVPS.PIDPT_INDEX[0] configuration by L1, or due to L1 WRMSR misbehavior or wrong configuration of UPID.
   1.2. If L1 supports L1-to-L1 IPI virtualization, check if the PIDPT entry is uninitialized. If so, the #VE happened due to improper TDVPS.PIDPT_INDEX[0] configuration by L1.
   1.3. Else (L1 does not support L1-to-L1 IPI virtualization), L1 may request the host VMM, using a software-defined protocol over TDG.VP.VMCALL, to post an interrupt to the destination VCPU.

4.4.2.3. *#VE due to APIC Write (SELF_IPI) by L1*

#VE(NON_CONFIG_PARAVIRT) with a basic VM exit reason of APIC Write (56) due to WRMSR(IA32_X2APIC_SELF_IPI) by L1 may happen due to L1 misbehavior, if the written interrupt vector value (in EAX[7:0]) is lower than 16.

4.5. Obtaining Guest Interruptibility State on #VE

Intel SDM, Vol. 3, Table 26-3	Format of Interruptibility State
Intel SDM, Vol. 3, 28.7.1	Interruptibility State

If supported by the TDX Module, TDG.VP.VEINFO can return the guest interruptibility state VMCS field to the caller. This may be required for proper paravirtualization.

Enumeration: TDX Module support is enumerated by TDX_FEATURES0.VEINFO_INTR_STATE. For details, see the [ABI Spec].

4.5.1. Typical Use Case: HLT Paravirtualization

Normally, the guest TD's #VE handler enables interrupts once TDG.VP.VEINFO.GET is called, if the code that caused exception runs with interrupts enabled. However, there is a need to emulate "STI shadow" – the CPU behavior where interrupts are enabled only after the instruction following STI.

A typical sequence used by guest TD is "STI; HLT". HLT is a paravirtualized instruction; the TDX Module injects a #VE. The guest TD's #VE handler typically calls TDG.VP.VMCALL to notify the host VMM. It should only enable interrupts immediately before TDCALL. Providing the guest interruptibility state allows the #VE handler to do this.

5. Guest TD's L1 Operation as a VMM of L2 VMs

This section describes a partitioned guest TD's L1 operation as a VMM of L2 VMs.

Unreleased Feature: At the time of writing of this document, Enhanced Interrupt Virtualization has not been released yet. Details provided in this document serve as a preview and are subject to change.

5.1. L2 Virtual Interrupts Configuration and Control

Intel SDM, Vol. 3, 26.6.8 Controls for APIC Virtualization
Intel SDM, Vol. 3, 31 APIC Virtualization and Virtual Interrupts

L1 must check and configure the following for each L2 VM where L2 IPI is intended to be used.

5.1.1. L2 VMCS Interrupt-Related Fields

The following table describes interrupt related fields of the L2 VMCS which are configurable by L1. The following sections describe how those fields are used by L1.

Table 5.1: Interrupt Related L2 VMCS Fields

Field Name	Bit Name	L1 Configurability	Details
Pin-Based VM-Execution Controls	Virtual NMIs	Fixed 1	
	Process posted interrupts	Fixed 0 / Configurable	This bit is configurable by L1 only if enhanced interrupt virtualization is supported by the TDX Module and has been configured by the host VMM.
Processor-Based VM-Execution Controls	Interrupt-window exiting	Configurable	
	Use TPR shadow	Fixed 1	
	NMI-window exiting	Configurable	
	Virtualize APIC accesses	Fixed 0	Virtual xAPIC mode is not supported.
	Virtualize x2APIC mode	Fixed 1	
	APIC-register virtualization	Fixed 1	
	Virtual-interrupt delivery	Configurable	
	IPI virtualization	Fixed 0 / Configurable	This bit is configurable by L1 only if enhanced interrupt virtualization is supported by the TDX Module and has been configured by the host VMM.
APIC-access address		None	Virtual xAPIC mode is not supported.
Virtual-APIC address		Configurable	Set by L1 to the L2 VCPU's VAPIC page GPA.
TPR threshold		Configurable	
EOI-exit bitmap		Configurable	
Posted-interrupt notification vector		None	Set by the host VMM (only if enhanced interrupt virtualization is supported by the TDX Module)
Posted-interrupt descriptor address		None	Set by the TDX Module to the Secure PID address in TDVPS (only if enhanced interrupt virtualization is supported by the TDX Module)

Field Name	Bit Name	L1 Configurability	Details
PID-pointer table address		None	Set by the TDX Module (only if enhanced interrupt virtualization is supported by the TDX Module)
Last PID-pointer index		None	Set by the TDX Module (only if enhanced interrupt virtualization is supported by the TDX Module)

5.1.2. L2 Virtual APIC Page

Intel SDM, Vol. 3, 31.5 Virtualizing MSR-Based APIC Access

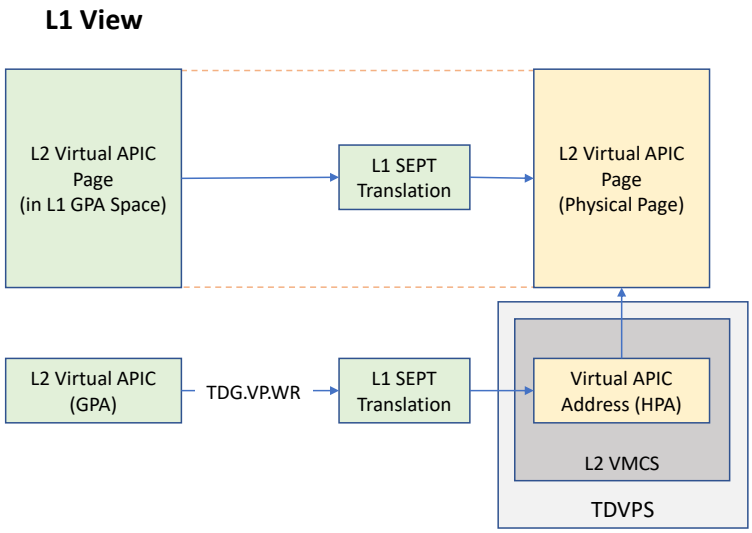

Figure 5.1: L2 Virtual APIC Page Mapping to L1 GPA Space

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].

5.1.3. Enhanced Mode: Checking L2 IPI Virtualization Support and L2 PIDPT Size

L1 can detect IPI virtualization support for L2 in the architectural way, by RDMSR(IA32_VMX_PROCBASED_CTLS3). Bit 4 enumerates IPI support. The TDX Module sets the virtual value of this bit only if both it and the CPU support IPI virtualization.

As discussed in 2.3, PIDPT holds multiple entries indexed by an IPI destination index. PIDPT is allocated by the host VMM. L1 should read TDCS.PIDPT_NUM_ENTRIES[vm], using TDG.VM.RD, to determine if the size of PIDPT allocated by the host VMM is sufficient for the desired L2 destination index (see below – typically virtual x2APIC ID) name space.

L2 IPI configuration can be done on demand. A TDCS.PIDPT_NUM_ENTRIES[vm] value of 0 indicates that IPI virtualization has not been configured by the host VMM for this L2 VM, and no PIDPT has been allocated. In that case, L1 can request the host VMM (using a software-defined protocol) to configure interrupts and allocate PIDPT. The host VMM can configure interrupts for an L2 VM as long as that L2 has not yet been entered.

Note: It is possible for L2 VM posted interrupts to be configured without IPI support. In that case, L1 is still required to handle L2→L1 exits and wakeup interrupts as described in the following sections.

5.1.4. Enhanced Mode: L2 IPI Destination Index Configuration

For sending IPIs, each VCPU should be assigned an L2 IPI destination index, which is the index of its entry in the PIDPT. The index is used as the destination field in the upper 32 bits when L2 posts an IPI by WRMSR(IA32_X2APIC_ICR). Each

VCPU configures its applicable L2 VM indices by writing to TDVPS.PIDPT_INDEX[vm], using TDG.VP.WR. The meaning of the destination index is chosen by L1; typically, it is the L2 virtual x2APIC ID.

The IPI destination index has the following characteristics:

- The index for each VCPU should be unique. However, the TDX Module does not enforce that.
- The index must fit in the PIDPT; it must be smaller than TDCS.PIDPT_NUM_ENTRIES[vm].

Failing to configure a destination index will result in an L2→L1 exit, with an APIC Write exit reason, when L2 attempts to send an IPI to an unconfigured destination index. This is discussed in the following sections.

5.1.5. Enhanced Mode: Enabling L2 IPI Virtualization

By default, L2 IPI virtualization is disabled. L1 should only enable it after the above steps are done. This is done by each VCPU; L1 sets bit 4 (IPI virtualization) of the L2 VMCS tertiary execution control, using TDG.VP.WR, to enable IPI virtualization. If the host VMM did not configure IPI properly, TDG.VP.WR returns a TDX_METADATA_FIELD_VALUE_NOT_VALID status.

By default, L2 execution of WRMSR(IA32_X2APIC_ICR) results in an L2→L1 exit. To enable IPI virtualization, L1 should allow the CPU to process WRMSR(IA32_X2APIC_ICR) by clearing the applicable bit in the L2 VM's WRMSR exit bitmap using TDG.VP.WR.

5.1.6. Enhanced Mode: L2 Posted Interrupt Filtering Configuration: PIR_MASK

L1 can determine if a Shared PID has been configured for an L2 VM by reading TDCS.PID_MODE[vm], using TDG.VM.RD. If so, L1 can configure interrupt filtering by writing to TDCS.PIR_MASK[vm] using TDG.VM.WR. The PIR_MASK entry is a 256-bit mask, where each bit N enables the corresponding posted interrupt vector N. By default, all bits are 0 and no interrupt vector is allowed. Bits 30:0 can never be set.

5.1.7. Enhanced Mode: Wakeup Interrupt Configuration

Wakeup interrupts notify L1 that a posted interrupt is waiting for an L2 VM. They are discussed in 5.6.

L1 configures the wakeup interrupt vector for an L2 VM by writing to TDCS.WAKEUP_VIRT_VECTOR[vm] using TDG.VM.WR. The vector value must be in the range 31 to 255 to enable wakeup interrupts. A vector value of 0 disables wakeup interrupts.

Configuring the wakeup vector is optional. If not configured, L1 is not notified if there's a posted interrupt to L2 while L1 or another L2 is running. In any case, interrupts posted to L2 are processed by the TDX Module on L2 entry.

The same wakeup vector may be configured for multiple L2 VMs; however, that has some performance impact since it requires L1 to look up the wakeup reason once an interrupt is received (see below).

5.2. L1-to-L2 Secure Posted Interrupts

5.2.1. Legacy Mode

5.2.1.1.

The L1 VMM can directly inject virtual interrupts to an L2 VM of the same VCPU by manipulating the L2 VMCS and Virtual APIC page.

Pending Virtual Interrupts Evaluation and Delivery

CPU update of the guest interrupt status (RVI and SVI) is modified in SEAM mode from the legacy VMX behavior. Instead of setting RVI (Requesting Virtual Interrupt) to the highest index of bit set in VIRR, RVI is set to the highest index of bit set in VIRR[31:255], i.e., VIRR[0:30] bits are ignored in the RVI computation.

5.2.2. Enhanced Mode

If supported by the TDX Module and properly configured, L1 can post interrupts to an L2 of any of the TD's VCPUs, using the Secure PID mechanism, by calling **TDG.INTR.POST**, with the following parameters:

- L2 VM number (1, 2 or 3)
- Destination L2 IPI index (as configured by L1)
- Interrupt vector, in the range 31 to 255

As with any posted interrupt, delivery is not guaranteed since TDX does not protect against DOS (e.g., the host VMM may prevent the destination VCPU from running). L1 should check that the IPI has been delivered.

For details on TDG.INTR.POST, see the [ABI Spec].

5.3. Handling L2 Virtual NMI

5.3.1. Virtual NMI Injection to L2 VCPU

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 an L2 VM of the same VCPU. An L2 VM exit with NMI Window exit reason results in an L2→L1 exit.

5.3.2. 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.

5.4. Pending L1 Virtual 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.

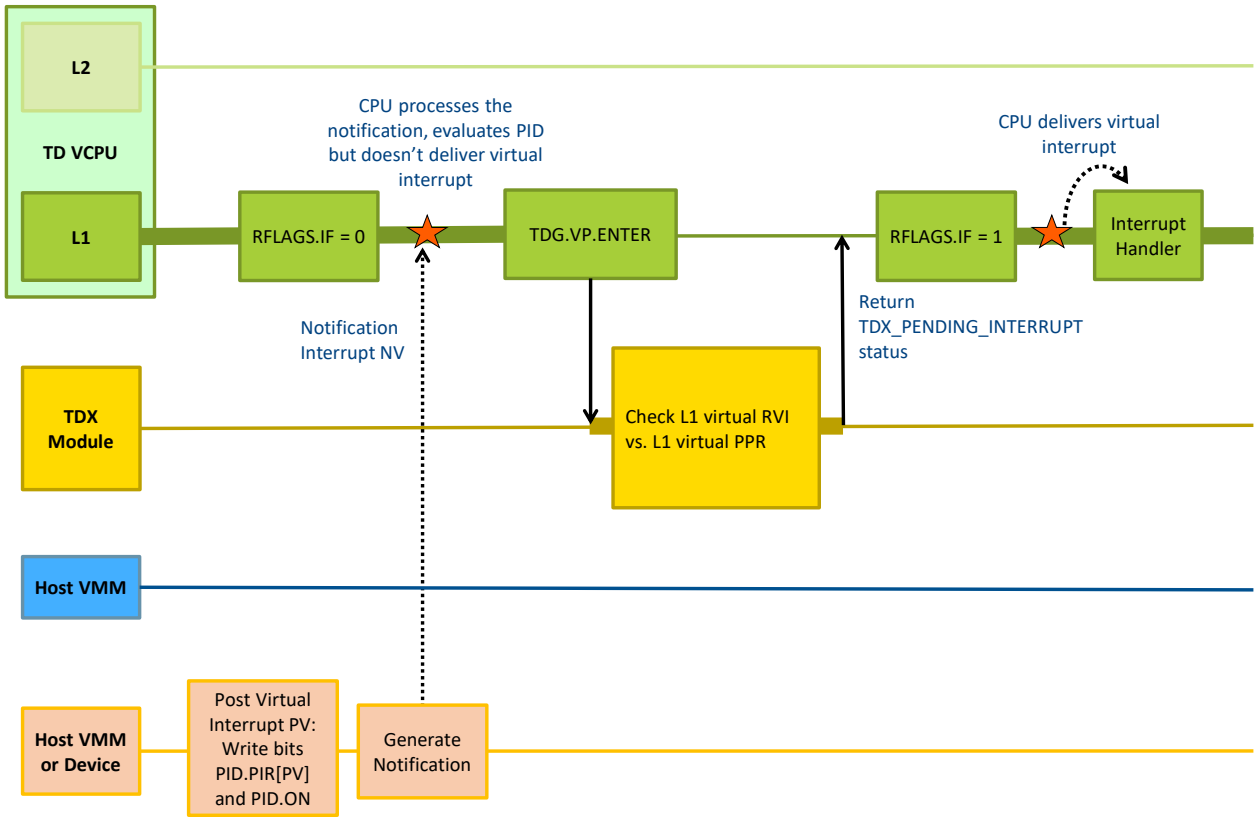

Figure 5.2: Pending L1 Virtual Interrupt Detection during L1-to-L2 Entry

5.5. Handling Interrupt-Related L2-to-L1 Exits

Interrupt-related L2→L1 exits may happen due to multiple reasons, as details in the sections below.

5.5.1. L2-to-L1 Exit due to an Interrupt Posted to L1

There are multiple cases where the VCPU runs in an L2 VM and there is a need to inject a virtual interrupt to be handled by L1:

- An interrupt was posted to L1 – either as an IPI sent by another L1 VCPU, by the host VMM or by IOMMU.
- Enhanced Mode: An interrupt was posted to another L2 VM, and a wakeup interrupt needs to be injected into L1 by the TDX Module, in order to trigger it to enter that L2 VM. Wakeup interrupts are discussed in 5.6 below.

Note: L2→L1 exit may happen immediately on TD entry, following a previous TD exit directly from an L2 VM. From L1's perspective the behavior is the same as with an L2→L1 exit from a running L2 VM.

In the above cases, the running L2 VM exits to L1. The following L2 exit information is provided to L1 as output operands of TDG.VP.ENTER:

- Status (in RAX) indicates TDX_L2_EXIT_PENDING_INTERRUPT or TDX_L2_EXIT_PENDING_INTERRUPT_TDVMACLL.
- Other GPRs indicate the L2 VM exit information. This information may not be useful to L1, since in some cases the VM exit from L2 may not be related to the interrupt – e.g., the VM exit caused a TD exit, and later an interrupt was posted on TD entry.

L1 should not call the interrupt handler directly; the interrupt is injected by the CPU once L1 is interruptible (RFLAGS.IF is 1).

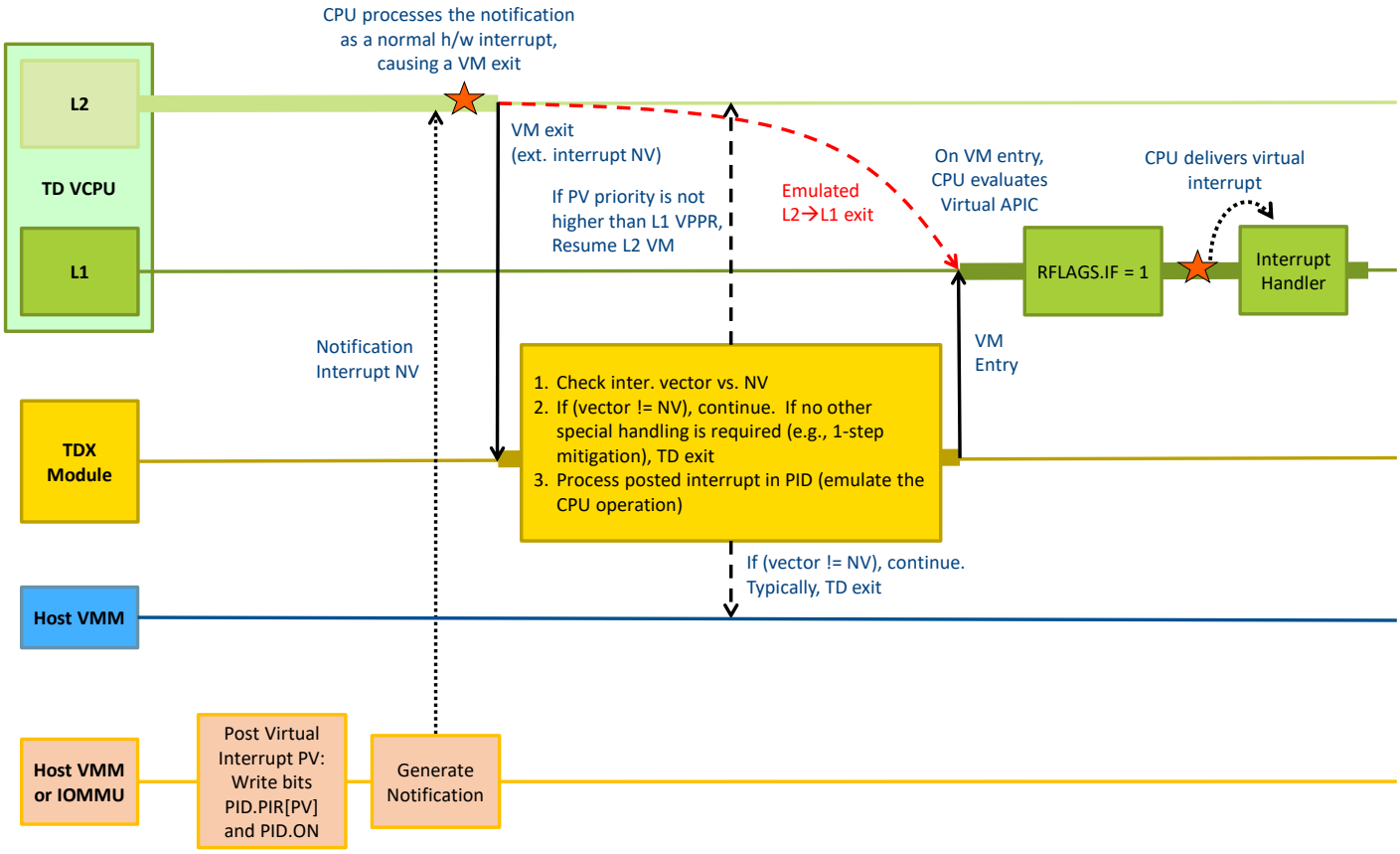

Figure 5.3: Example of an Interrupt Posted to L1 and Received during L2 Run Time

5.5.2. Enhanced Mode: L2-to-L1 Exit due to an Interrupt Posted to L2

There are multiple cases where the VCPU runs in an L2 VM and there is a need to inject a virtual interrupt to be handled by L2:

- An interrupt was posted to another L2 VM, and wakeup interrupt needs to be injected into L1 by the TDX Module, in order to trigger it to enter that L2 VM. Wakeup interrupts are discussed in 5.6 below.
- An interrupt was posted to any L2 VM on TD entry, and wakeup interrupt needs to be injected into L1 by the TDX Module, in order to trigger it process all posted interrupts.

Note: An L2→L1 exit may happen immediately on TD entry, following a previous TD exit directly from an L2 VM. From L1's perspective the behavior is the same as with an L2→L1 exit from a running L2 VM.

In the above cases, the running L2 VM exits to L1. The following L2 exit information is provided to L1 as output operands of TDG.VP.ENTER:

- Status (in RAX) indicates TDX_L2_EXIT_PENDING_INTERRUPT or TDX_L2_EXIT_PENDING_INTERRUPT_TDVMACLL.
- Other GPRs indicate the L2 VM exit information. This information may not be useful to L1, since in some cases the VM exit from L2 may not be related to the interrupt – e.g., the VM exit caused a TD exit, and later an interrupt was posted on TD entry.

If a wakeup interrupt was configured, L1 should not call the interrupt handler directly; the wakeup interrupt is injected by the CPU once L1 is interruptible (RFLAGS.IF is 1). Else, L1 should read TDVPS.WAKEUP_SENT to determine what needs to be done. See the wakeup interrupts discussion in 5.6 below.

5.5.3. L2-to-L1 Exit due to WRMSR(ICR) by L2

An L2→L1 exit with a basic VM exit reason of WRMSR (32) due to WRMSR(IA32_X2APIC_ICR) by L2 happens if IPI virtualization is not supported by the TDX Module or is not enabled for L2. L1 may paravirtualize IPIs with the cooperation of the host VMM.

5.5.4. L2-to-L1 Exit due to APIC Write by L2

Intel SDM, Vol.3, 31.5	Virtualizing MSR-Based APIC Accesses
Intel SDM, Vol.3, 31.7	Virtualizing SENDUIPI

Overview

An L2→L1 exit due to APIC write is trap-like; it happens after the L2 instruction causing it has executed. Bits 11:0 of the exit qualification are set to the offset of the write access in the L2 VAPIC page. The L2 VAPIC page itself is directly accessible by L1.

Enhanced Mode: L2-to-L1 Exit due to APIC Write (ICR) by L2

5.5.4.1.

L2→L1 exits with a basic VM exit reason of APIC Write (56) may happen because of WRMSR(IA32_X2APIC_ICR) or SENDUIPI execution by L2, due to improper configuration by L1 or due to L2 misbehavior. L1 should handle such VM exits as follows:

5.5.4.2.

1. Check the exit qualification field to determine the Virtual APIC page offset.
2. If the exit qualification indicates that the updated VAPIC field is ICR (offset 0x300):
   - 2.1. Read L2 VAPIC's ICR field. The L2 VAPIC resides in a TD private page; ICR is directly accessible to L1 at offset 0x300 (lower 32 bits) and 0x310 (upper 32 bits).
   - 2.2. Bits 31:8 should be 0. If not, the L2→L1 exit happened due to L2 misbehavior.
     Note: On WRMSR(IA32_X2APIC_ICR), a non-0 value of bits 31:20, 17:16, or 13 results in a #GP injection to L2; no APIC write takes place and there is no L2-to-L1 exit. See [Intel SDM, Vol.3, 30.5].
   - 2.3. Check if the vector value in bits 7:0 was lower than 16. If not, the L2→L1 exit happened due to L2 misbehavior.
   - 2.4. Check if the destination value in bits 63:32 is lower than the number of PIDPT entries. If not, the L2→L1 exit happened either due to improper TDVPS.PIDPT_INDEX[vm] configuration by L1, or due to L2 misbehavior.
   - 2.5. Check if the PIDPT entry is uninitialized. If so, the L2→L1 exit happened due to improper TDVPS.PIDPT_INDEX[vm] configuration by L1.

5.5.4.3. *L2-to-L1 Exit due to APIC Write (SELF_IPI) by L2*

An L2→L1 exit with a basic VM exit reason of APIC Write (56) due to WRMSR(IA32_X2APIC_SELF_IPI) by L2 may happen due to L2 misbehavior, if the written interrupt vector value (in EAX[7:0]) was lower than 16.

5.6. Enhanced Mode: L2 Wakeup Interrupt

If configured by L1, an L2 wakeup virtual interrupt is injected to L1 by the TDX Module to notify it that a posted interrupt is waiting for an L2 VM. There are multiple cases:

- An interrupt was posted to an L2 VM, but L1 was running.
- An interrupt was posted to an L2 VM , but another L2 VM was running.
- An interrupt was posted to an L2 VM before or as part of TD entry.

The source of the posted interrupt may be another VCPU, as shown in the diagrams below, or an external source such as the host VMM or IOMMU.

Wakeup interrupt configuration is discussed in 5.1.7 above.

L1 typically handles the wakeup interrupt by entering the L2 VM (TDG.VP.ENTER) so that L2 can handle its posted interrupt.

To determine which L2 VM requires wakeup, there are three options:

Unique Wakeup Interrupt: The simplest option is for L1 to assign a separate wakeup interrupt vector for each applicable L2 VM.

Non-Unique Wakeup Interrupt: However, due to, e.g., virtual interrupt vector namespace considerations L1 may assign the same wakeup interrupt vector for more than one L2 VM. In this case, L1 can read TDVPS.WAKEUP_SENT as a single 32-bit metadata field, using TDG.VP.RD, to determine which L2 VM needs to be entered. For details, see the [TDX Module ABI Spec].

No Wakeup Interrupt: L1 may choose not to configure a wakeup interrupt for an L2 VM. In this case, there is no notification if L1 is already running. L1 will only be notified on L1→L2 exit, with either of the following status codes:

- TDX_L2_EXIT_PENDING_INTERRUPT
- TDX_L2_EXIT_PENDING_INTERRUPT_TDVMCALL

L1 can then read TDVPS.WAKEUP_SENT to determine which L2 VM has a pending interrupt.

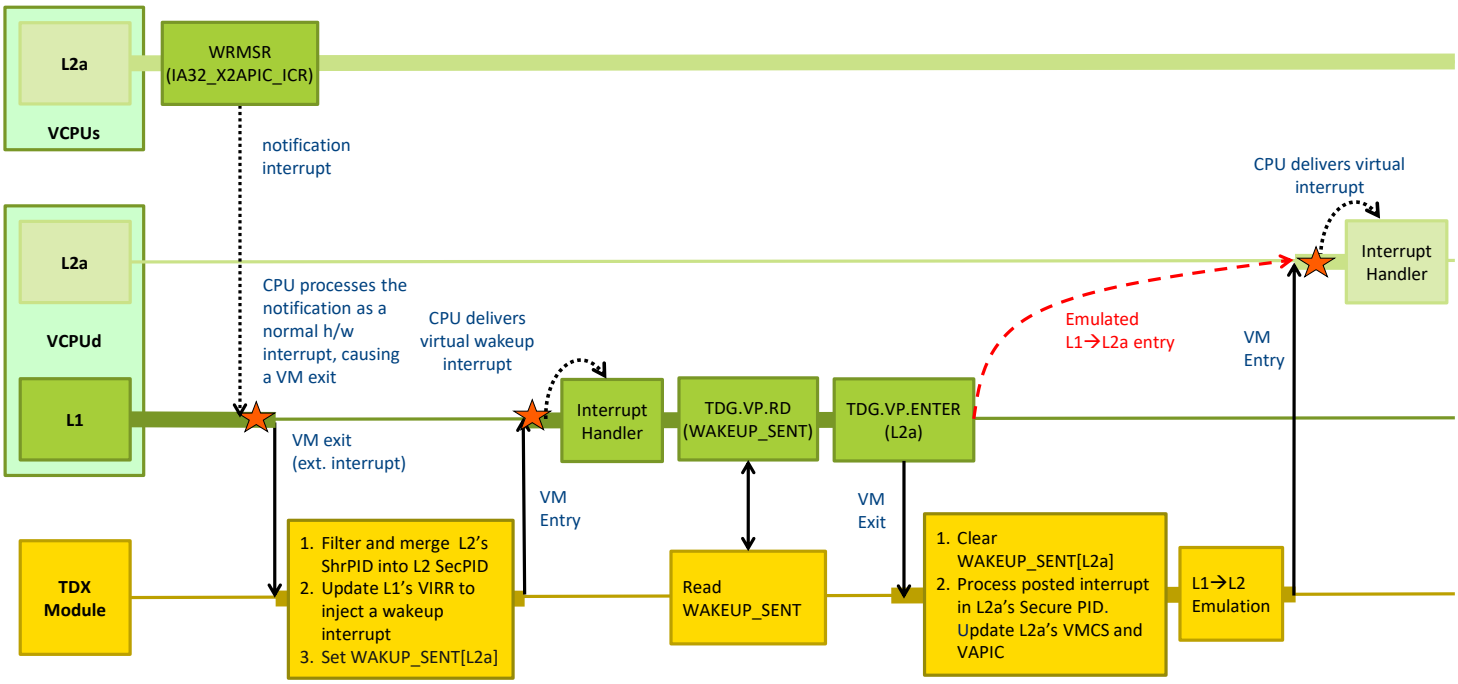

Figure 5.4: Example of a Wakeup Interrupt when the Destination VCPU is Running in L1

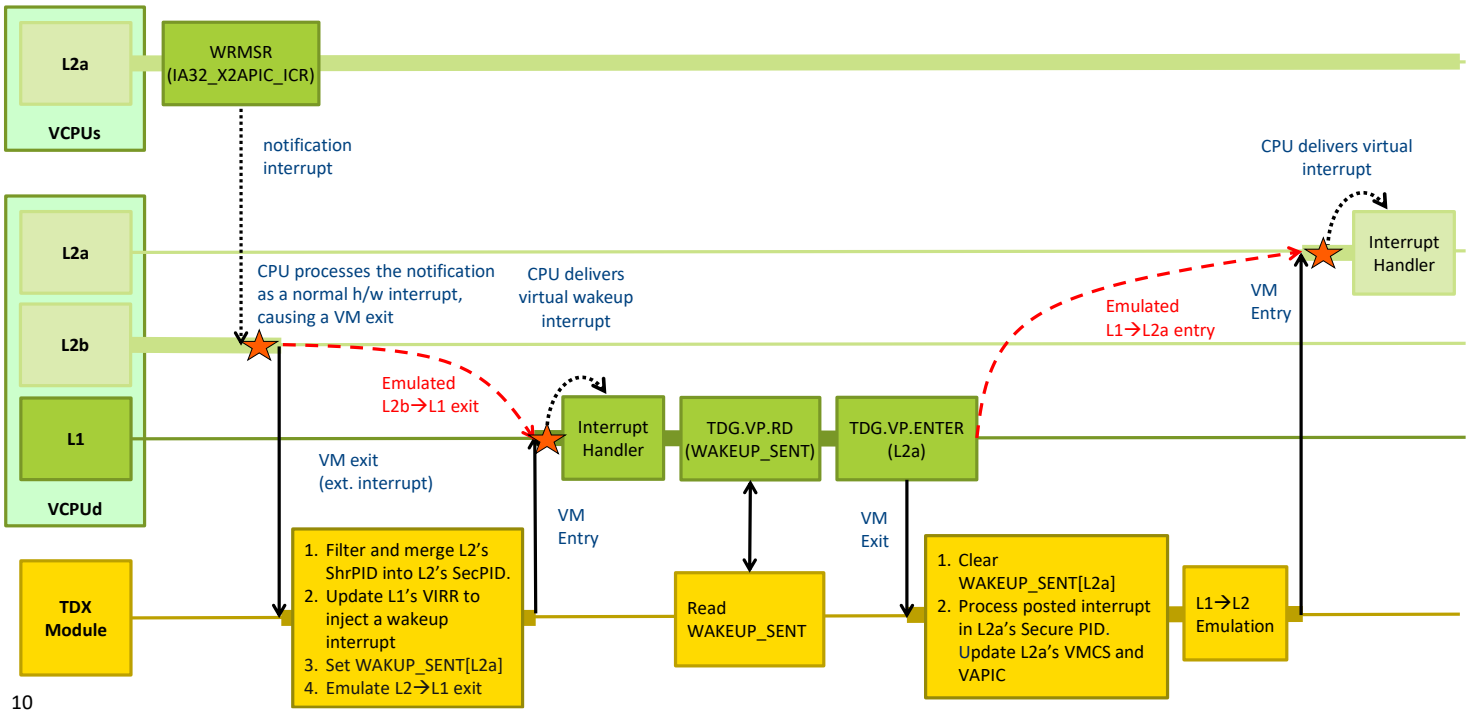

Figure 5.5: Example of a Wakeup Interrupt when the Destination VCPU is Running in Another L2

6. Guest TD Perspective: L2 VM

Intel SDM, Vol. 3, 11.6	Issuing Interprocessor Interrupts
Intel SDM, 11.12.9	ICR Operation in x2APIC Mode
Intel SDM, 30.1.6	IPI Virtualization

From L2's perspective, interrupt support is architectural, and no enabling should be required (subject to the limitations described below).

Unreleased Feature:	At the time of writing of this document, Enhanced Interrupt Virtualization has not been released yet. Details provided in this document serve as a preview and are subject to change.

6.1. Feature Enumeration

The TDX Module provides no IPI feature enumeration for L2. If enumeration is required, it should be emulated by L1.

6.2. Posting an L2-to-L2 IPI

An L2 VM can post inter-processer interrupts (IPIs) in the architectural way, by writing to its virtual APIC's interrupt command register (ICR), i.e., WRMSR(IA32_X2APIC_ICR).

6.2.1. Legacy Mode

If the TDX Module does not support enhanced interrupt virtualization, or it has not been configured, L1 may choose to emulate L2-to-L2 IPI functionality.

6.2.2. Enhanced Mode

To post an IPI, the following conditions must be met:

- PIDPT and Secure PID has been configured by the host VMM.
- IPI has been configured by L1.

To post an interrupt to another VCPU, L2 executes WRMSR(IA32_X2APIC_ICR) with the following data:

- Destination (bits 63:32) is the virtual x2APIC ID.
- Bits 7:0 contain the interrupt vector. Allowed vector values are 31 to 255.
- All other bits must be 0.

Only simple unicast IPIs are directly supported by TDX. More sophisticated modes (e.g., post to all excluding self) are not directly supported; L1 may emulate those modes, if required.

x2APIC ID is enumerated to L2 via either IA32_X2APIC_APICID (MSR 0x802) or CPUID(0x1F).EDX or CPUID(0x0B).EDX.

TDX does not protect against DOS; thus, IPI delivery is not guaranteed (e.g., the host VMM may prevent the destination VCPU from running). L2 is expected to check that the IPI has been delivered.

6.3. Posting an L2-to-L2 User IPI

An L2 VM can post user inter-processer interrupts (UIPIs) in the architectural way, by executing SENDUIPI. This feature uses a User Posted Interrupt Descriptor (UPID) table, which resides in the L2 VM's GPA space and is configured by the L2 VM's OS. When properly configured, SENDUIPI results in a virtual notification interrupt, sent as an IPI with the vector and destination values taken from the UPID.

From TDX perspective, this is just another IPI case. Proper configuration of user interrupts, including UPID, is the responsibility of the L2 VM.

6.4. Enhanced Mode: Posted Interrupt Filtering Configuration

An L2 VM can be configured by the host VMM (see 7.2) and the L1 VMM (see 5.1) to handle interrupts posted by non-secure agents such as the host VMM or IOMMU. However, there is no direct way for L2 to configure posted interrupt vector filtering; this configuration is done by L1. If required, L2 can request L1 using a software-defined protocol to configure the filtering.

7. Host VMM Perspective

This chapter discusses the TDX virtual interrupt architecture from the host VMM's perspective.

Unreleased Feature:	At the time of writing of this document, Enhanced Interrupt Virtualization has not been released yet. Details provided in this document serve as a preview and are subject to change.

7.1. Feature Enumeration

TDX module support for enhanced posted interrupt handling is enumerated by TDX_FEATURES0.ENHANCED_INTR_VIRTUALIZATION (bit 45), readable using TDH.SYS.RD*.

7.2. Configuration

7.2.1. Overview: Enhanced vs. Legacy Posted Interrupt Configuration

By default, posting interrupts to a TD is disabled. The TDX Module allows the host VMM to configure posted interrupts in a legacy mode, which is only supported for L1. This configuration is done per VCPU.

If enhanced interrupt virtualization is supported by the TDX Module, it allows the host VMM to configure posted interrupts in an enhanced way. Enhanced mode is supported for L1 and for L2. This configuration is done per L1 or L2 VM and per VCPU.

The legacy and enhanced mode (for L1) are mutually exclusive.

7.2.2. Legacy L1 Posted Interrupt Configuration (Per VCPU)

Legacy posted interrupt support, for L1, is configured per VCPU. The host VMM configures the PID structure in shared memory, and then configures the following TD VMCS fields:

- PID pointer is set with the HPA of the PID in shared memory.
- Notification vector is set to a value between 31 and 255.
- Pin-based execution controls' process posted interrupt bit (7) is set to 1 to enable posted interrupt processing.

Setting the process posted interrupt bit is only allowed after the PID pointer and the notification vector have been written with legal values.

Since legacy and enhanced mode are mutually exclusive, once TDH.INTR.CONFIG has been called to configure L1 in an enhanced mode, the TD VMCS notification vector field and process posted interrupt bit can't be written by the host VMM.

7.2.3. Enhanced Posted Interrupt Configuration

Enhanced Per-VM (L1 and each L2) Configuration: TDH.INTR.CONFIG

If enhanced interrupt virtualization is supported by the TDX Module, the host VMM can call TDH.INTR.CONFIG to allocate and initialize interrupt-related TD-scope metadata for one VM (L1 or L2) at a time within a TD:

1. Add the VM's PIDPT, as a contiguous memory area.
2. Configure the VM's PID mode: Regular PID (L1 only), Secure PID, Shared PID or Dual PID.
3. Configure the VM's main notification vector (for Regular PID or Secure PID).
4. Configure the VM's Shared PID notification vector (if used).

The host VMM may call TDH.INTR.CONFIG at any time after the TD has been initialized by TDH.MNG.INIT, even after the TD build has been completed (by TDH.MR.FINALIZE), with the following limitations:

- TDH.INTR.CONFIG may only be completed successfully once per L1 and each L2 VM.
- For L1, TDH.INTR.CONFIG may not be called once "enable posted interrupts" control has been set by the host VMM on any of its VCPUs; this action selects legacy interrupt configuration.
- For an L2 VM, TDH.INTR.CONFIG may not be called after that VM was entered for the first time.

L1 configuration for each VCPU becomes effective on the first TD entry to that VCPU following TDH.INIT.CONFIG successful completion. If done after TD build, when L1 may already be running, the host VMM may send IPIs to the TD's VCPUs to help ensure the configuration is effective.

Notification Vector Uniqueness and its Implication on Performance and Functionality

Notification vector values may be configured as unique (among all notification vectors of the same TD) , non-unique or null. This trades performance and functionality vs. physical interrupt vector space consumption, as described in the table below.

- Regular PID (L1 only), Secure PID and Shared PID may be configured with a notification vector value that is unique or non-unique.
- In addition, Regular PID and Shared PID may be configured with a null notification vector value.

7.2.3.2.

Table 7.1: Unique vs. Non-Unique vs. No Notification Vectors

	Unique Notification Vector	Non-Unique Notification Vector	Notification on TD Entry (No Notification Vector)
Usage	+ Normal	+ Normal	- Applicable only for injecting a posted interrupt on TD entry, using PID in shared memory (regular PID or Shared PID)
Performance	+ Posted interrupts (with Regular PIDs in shared memory) or incoming IPIs (with Secure PIDs) are processed by the CPU, with no software involvement, if the VCPU is currently running in the applicable VM.	- Posted interrupts (with Regular PIDs in shared memory) or incoming IPIs (with Secure PIDs) must be processed by the TDX Module if the notification vector is shared. Note that Shared PIDs are always processed by the TDX Module.	- Posted interrupts on TD entry are processed by the TDX Module
	+ When processing a unique notification vector, the TDX Module reads a single PID.	- When processing a non-unique notification vector, the TDX Module must read all applicable PIDs.	+ When processing a notification flag on TD entry, the TDX Module reads a single PID.
Physical Interrupt Vector Space 7.2.3.3.	- Unique notification vectors consume one interrupt vector name space entry (of 225) per PID.	+ Non-unique notification vectors help reduce interrupt vector name space consumption.	+ Does not consume interrupt vector name space.

Enhanced Per-VCPU Configuration

For L1 and each L2 VM, if a PID in shared memory (either as a Regular PID or as Shared PID) has been configured by TDH.INTR.CONFIG, the host VMM should do the following:

- Directly configure the PID structure in shared memory.
- Write the PID's HPA (including HKID) to the L1 or L2 VMCS' PID pointer field, using TDH.VP.WR.

7.3. Interaction with TD Migration

7.3.1. Enhanced Mode: Migrated Metadata

If enhanced interrupt virtualization is supported by the TDX Module, Secure PID is migrated as part of the VCPU metadata.

PID in shared memory (Regular PID or Shared PID) is not migrated. However, as part of the VCPU metadata export (TDH.EXPORT.STATE.VP), the TDX Module processes the PIDs in shared memory (if any) and merges them into the Secure PIDs. Secure PIDs are then processed on the first TD entry to that VCPU on the destination platform.

Note: Secure PIDs are used for this purpose even if not enabled by TDH.INTR.CONFIG for normal IPI usage.

The host VMM should ensure none of the TD's PIDs in shared memory is updated, either by itself or the IOMMU, after the TD is paused by TDH.EXPORT.PAUSE.

7.3.2. Configuration on Import

Overview

Some posted interrupt resources and configurations are not migrated, since resources need to be allocated on the destination platform, and configuration may be different between the source platform and the destination platform. This includes the following:

- Addresses of PIDs in shared memory
- Notification vector values
- PIDPT pages

7.3.2.1.

Thus, as part of TD import, posted interrupts must be configured, per VM (L1 and each L2 VM) and per VPCU, on the destination platform.

Legacy Posted Interrupts Reconfiguration on Import

With legacy configuration of posted interrupt handling, the TD (L1) VMCS posted interrupt execution controls are reset to their initial values when the TD is migrated. The host VMM on the destination platform must configure them in order to use posted interrupts.

7.3.2.2.

Enhanced Posted Interrupts Reconfiguration on Import

7.3.2.3.

If any of the TD's L1 or L2 VMs have been configured for enhanced interrupt virtualization by TDH.INTR.CONFIG before being migrated, they must be reconfigured as part of the TD import.

The host VMM must call TDH.INTR.CONFIG after the TD's mutable state (which includes the interrupt virtualization configuration) has been imported by TDH.IMPORT.STATE.TD, but before the VCPUs' mutable state are imported by TDH.IMPORT.STATE.VP.

The configuration must be compatible with the configuration on the source platform. For example, the allocated PIDPT should be at least as large as the PIDPT on the source platform. Note that there is no requirement for notification vector allocations to be the same as on the source platform.

7.3.2.4.

Initial Posted Interrupts Configuration after Migration

As described above, posted interrupt configuration may happen after the TD build has been completed (by TDH.MR.FINALIZE). Therefore, it is possible for a TD to be migrated before one or more of its VMs (L1 or L2) has been configured for posted interrupts. In this case, posted interrupt configuration may happen on the destination platform.

7.4. Posting Virtual Interrupts

Intel SDM, Vol. 3, 31.6 Posted-Interrupt Processing

The host VMM or the IOMMU can post virtual interrupts to a TD's L1 or L2 VM by writing to the posted-interrupt descriptor (PID) which has been configured, as described above, for the target VCPU. The PID resides in a shared page, directly accessible by the host VMM or the IOMMU. Note that the operation is the same whether this is a Shared PID or a Regular PID.

7.4.1. Posting a Virtual Interrupt when a Notification Vector is Configured

A virtual interrupt is posted as follows:

1. Atomically set the PID's Posted-Interrupt Requests (PIR) bit corresponding to the interrupt vector to be injected.
2. Atomically set the PID's Outstanding Notification (ON) bit.
3. Generate a physical interrupt, targeted at the LP running the target VCPU, as a notification interrupt.

To send a notification interrupt to a VCPU that is about to run on the current LP, the host VMM can generate a self IPI with the notification vector prior to TD entry, by clearing RFLAGS.IF and writing to the IA32_X2APIC_SELF_IPI MSR.

7.4.2. Enhanced Mode: Posting a Virtual Interrupt when no Notification Vector is Configured

As described in 7.2.3.2 above, it is possible to configure posted interrupts with no notification vector. In this case, interrupts are posted on TD entry.

After setting the PID's PIR bit and ON bit, the host VMM can call TDH.VP.ENTER and indicate that an outstanding notification is pending. The TDX Module will then process the PID and inject the virtual interrupt. See the [ABI Spec] for details.

7.5. Virtual NMI Injection (L1 Only)

The host VMM can request the TDX Module to inject a virtual NMI into a guest TD VCPU using the TDH.VP.WR function, by setting the PEND_NMI TDVPS field to 1. NMI injection is only supported for L1. This can be done only when the VCPU is not active (a VCPU can be associated with at most one LP). Following that, the host VMM can call TDH.VP.ENTER to run the VCPU; the Intel TDX Module will attempt to inject the NMI as soon as possible.

The host VMM can use TDH.VP.RD to read PEND_NMI and get the status of virtual NMI injection. A value of 0 indicates that virtual NMI has been injected into the guest TD VCPU. The host VMM also may choose to clear PEND_NMI before it is injected.

If the TDX Module supports enhanced interrupt state indication, an immediate resumption hint may be provided on TD exits, and the host VMM can read VCPU_STATE_DETAILS using TDH.VP.RD. For details, see 7.7.

7.6. Handling a TD Exit due to a Cross-TD-VCPU IPI Request

The host VMM may need to support IPI in some cases, as described below.

7.6.1. Legacy Mode TD Cross-VCPU IPI

If the TD's L1 is not configured for enhanced interrupt virtualization (e.g., because the TDX Module doesn't support that), cross-VCPU IPI requires a software protocol supported by the host VMM. The guest TD's source VCPU should request an operation from the host VMM using TDG.VP.VMCALL. The VMM would then inject the requested virtual interrupt into the guest TD's destination VCPUs using the posted interrupt mechanism, as described in 7.4. This is an untrusted operation; thus, the TD needs to track its completion.

7.6.2. Enhanced Mode TD Cross-VCPU IPI

If a TD L1 or L2 VM is configured for enhanced interrupt virtualization, in most cases there is no need for host VMM involvement in the actual IPI process. However, when a TD VCPU (running in either L1 or L2) sends an IPI to a destination VCPU that is not running (neither in L1 or L2) at that time, the host VMM is notified. The host VMM can then schedule the destination VCPU so it will receive the IPI.

The host VMM is notified by a TD exit from the source VCPU. The TDH.VP.ENTER output contains the following information:

- The status code indicates TDX_IPI_REQUEST and the VCPU_INDEX of the destination VCPU.
- VM index indicates which L1 or L2 VM requested the IPI.

For details, see the [TDX Module ABI Spec].

The host VMM is expected to schedule the destination TD VCPU to run on some LP, and call TDH.VP.ENTER on that LP.

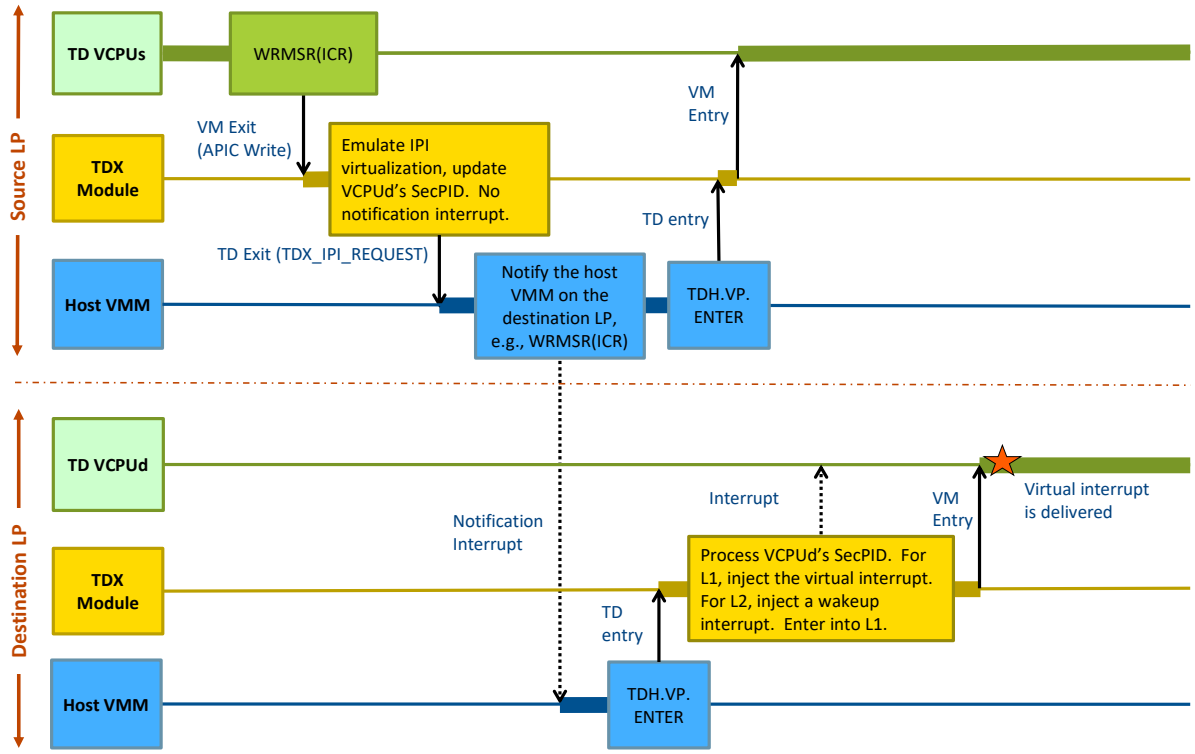

Figure 7.1: Example of Handling a TD Exit due to an IPI Request when the Destination is Not Running

7.7. VCPU Virtual Interrupt Status and the Immediate Resume Hint

On TD exits, the TDX Module provides the host VMM with an immediate resume hint, to help in cases where a TD would typically need to be resumed as soon as possible to handle pending interrupts. It also provides virtual interrupt state information which the host VMM may read, if it needs more details.

Enumeration: Support of the immediate resume hint and most of the interrupt status information bits is enumerated by TDX_FEATURES0.ENHANCED_INTR_STATE (bit 40) and TDX_FEATURES0.ENHANCED_INTR_VIRTUALIZTION (bit 45). For details, see the [TDX Module ABI Spec].

7.7.1. TD Exit with Immediate Resume Hints

If supported, the TDX Module can be configured to provide a hint to the host VMM on TD exits, indicating whether a TD re-entry is requested as soon as possible. This hint, called IMM_RESUME_HINT, is set to 1 if any of the configured bits of TDVPS.VCPU_STATE_DETAILS is 1. Thus, the host VMM can avoid the overhead of explicitly reading TDVPS.VCPU_STATE_DETAILS in the common case where all bits are 0.

The host VMM configures the set of cases when IMM_RESUME_HINT will be set to 1 by writing to the TDCS.VM_CTLS field. For details, see the [TDX Module ABI Spec].

7.7.2. Virtual Interrupt Status Indication

The TDX Module provides, for each VCPU, a TDVPS field called VCPU_STATE_DETAILS. This field holds, for L1 and each L2 VM, a set of state flags that the host VMM can read, using TDH.VP.RD.

Table 7.2: Virtual Interrupt Status Indication

Name	Description
VINTR_PENDING[3:0]	A set of 4 bits which indicate, for L1 and each L2 VM, whether a virtual interrupt is pending delivery to the VCPU in its Virtual APIC page. Indication is based on whether the current RVI (Requesting Virtual Interrupt) has a higher priority than the value of virtual APIC's PPR (Processor Priority Register).

Name	Description
VINTR_IN_SERVICE[3:0]	A set of 4 bits which indicate, for L1 and each L2 VM, whether a virtual interrupt is currently being serviced. The indication is based on whether the current SVI (Servicing Virtual Interrupt) is higher than 30.
VNMI_PENDING	A bit which indicates, for L1, whether a virtual NMI is pending delivery to the VCPU. A virtual NMI is injected by the host VMM by setting TDVPS.PEND_NMI using TDH.VP.WR. **Note:** Injecting a virtual NMI into L2 VMs is not currently supported.
ON_SET[3:0]	A set of 4 bits which indicate, for L1 and each L2 VM, whether there is an outstanding posted interrupt notification. Indication is based on the Secure PID's (if configured) ON bit. No such indication is provided for PIDs in shared memory (whether configured as Regular PID or Shared PID); the host VMM can directly read the ON bit.

For a detailed definition of VCPU_STATE_DETAILS, refer to the [TDX Module ABI Spec]. The spec also defines how the availability of each VCPU_STATE_DETAILS bit, which depends on TDX Module support, is enumerated.

7.7.3. Typical Use Cases

Halt Request from the Guest TD with a Pending Interrupt

7.7.3.1. This use case is standardized by the [GHCI Spec] as Instruction.HLT. The guest TD VCPU indicates to the host VMM, using TDG.VP.VMCALL, that it has no work to do and can be halted. The guest TD is expected to pass an "interrupt blocked" flag, cleared to 0 if and only if either RFLAGS.IF is 1 or the TDCALL instruction that invoked TDG.VP.VMCALL immediately followed an STI instruction (thus interrupt would be enabled when the guest TD VCPU is resume after the TDCALL).

If the "interrupt blocked" flag is 0, it means that the guest TD is expecting an interrupt, which was blocked while TDCALL was executing but would be delivered when the guest TD VCPU is resume after the TDCALL. In such cases, it may be desirable for the host VMM to re-enter the TD as soon as possible to let it handle the interrupt.

The host VMM can determine whether a virtual interrupt is pending to the guest TD VCPU by reading VCPU_STATE_DETAILS.VINTR_PENDING bits using TDH.VP.RD.

If supported by the TDX Module, the host VMM can avoid the need to explicitly read VCPU_STATE_DETAILS after each TD exit due to a TDG.VP.VMCALL(Instruction.HLT) in the common case where no virtual interrupt is pending, by configuring the TD's VM_CTLS such that the TD exit information will indicate a pending interrupt by the IMM_RESUME_HINT flag.

7.7.3.2.

Pending NMI

A similar use case exists for virtual NMI injection. A virtual NMI is injected by the host VMM by writing to TDVPS.PEND_NMI, using TDH.VP.WR, immediately before TD entry. However, the virtual NMI delivery may be delayed until the virtual CPU interruptibility state allows it. Thus, on TD exit the VMM may need to determine if the virtual NMI is still pending to be injected; the host VMM can then re-enter the TD as soon as possible to let it handle the NMI.

7.7.3.3.

The host VMM can determine this by reading TDVPS.PEND_NMI using TDH.VP.RD. If supported by the TDX Module, the host VMM can avoid the need to explicitly read TDVPS.PEND_NMI after each TD exit due to a TDG.VP.VMCALL in the common case where no virtual NMI is pending, by configuring the TD's VM_CTLS such that the TD exit information will indicate a pending NMI by the IMM_RESUME_HINT flag.

Outstanding Posted Interrupt Notification

This is discussed in 7.8.2

7.8. Enhanced Mode: Outstanding Posted Interrupt Notification Detection

7.8.1. Overview

The case of posting an IPI to a VCPU that is not currently running is described in 7.6.2. However, it is also possible for an IPI to be posted to a TD VCPU during the TD exit flow, before the destination VCPU status is marked internally by the TDX Module as not running. In this case, the Secure PID's PIR and ON bits are already updated, but the notification interrupt is not delivered while the destination VCPU is running.

Since the Secure PID is not directly accessible to the host VMM, there are two options for the host VMM to detect that the Secure PID's ON bit was set, as described below.

Table 7.3: Outstanding Posted Interrupt Notification Detection Options

Option	Pros	Cons
Immediate resume hint on TD exit	Almost no overhead on TD entry/exit times	May miss some theoretical, very unlikely concurrency cases
Handling the notification interrupt in the host VMM context	No overhead on TD exit/entry times	
Use TDH.VP.READ to read TDVPS.VCPU_STATE_DETAILS	Get exact outstanding notification information	Adds considerable overhead to TD exit/entry times

7.8.2. Immediate Resume Hint on TD Exit

The host VMM can configure the TD to set the IMM_RESUME_HINT flag on TD exit, if the Secure PID's ON bit is set. This is described in 7.7.2.

The host VMM may choose to immediately resume the TD when an immediate resume is hinted. If required, it may read TDVPS.VCPU_STATE_DETAILS to understand the cause of the immediate resume hint.

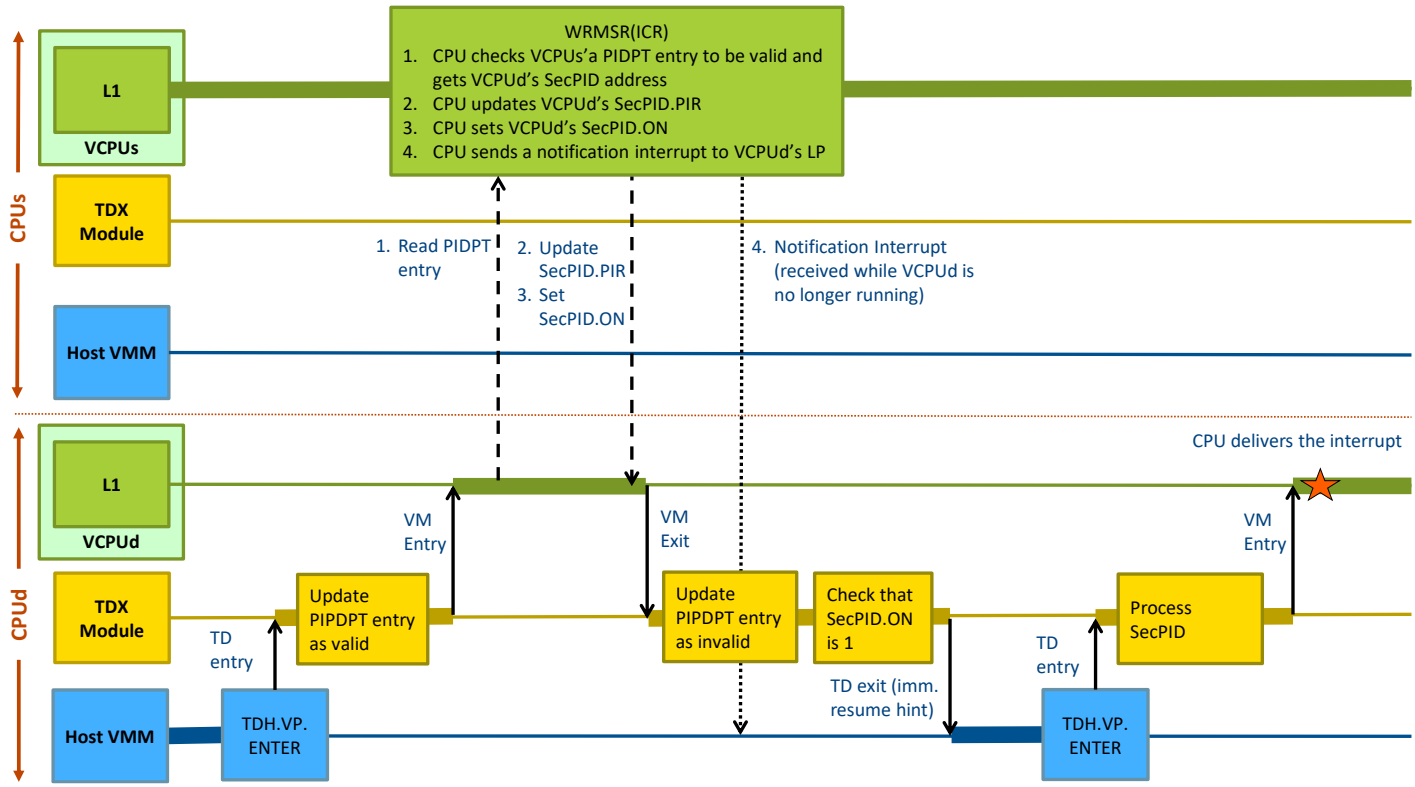

Figure 7.2: Example of an L1 IPI while Destination VCPU Just Exited

Note: The IMM_RESUME_HINT option has a theoretical, very unlikely concurrently case where the setting of the Secure PID's ON bit might not be detected by the TDX Module's TD exit flow. This is depicted in Figure 7.3 below.

7.8.3. Handling the Notification Interrupt in the Host VMM Context

An alternative to the IMM_RESUME_HINT method described above is for the host VMM itself to handle the notification interrupt, which is sent by the source VCPU but missed by the destination VCPU.

The host VMM may choose to immediately resume the TD if the notification interrupt vector was received. If required, it may read TDVPS.VCPU_STATE_DETAILS to understand the cause of the immediate resume hint.

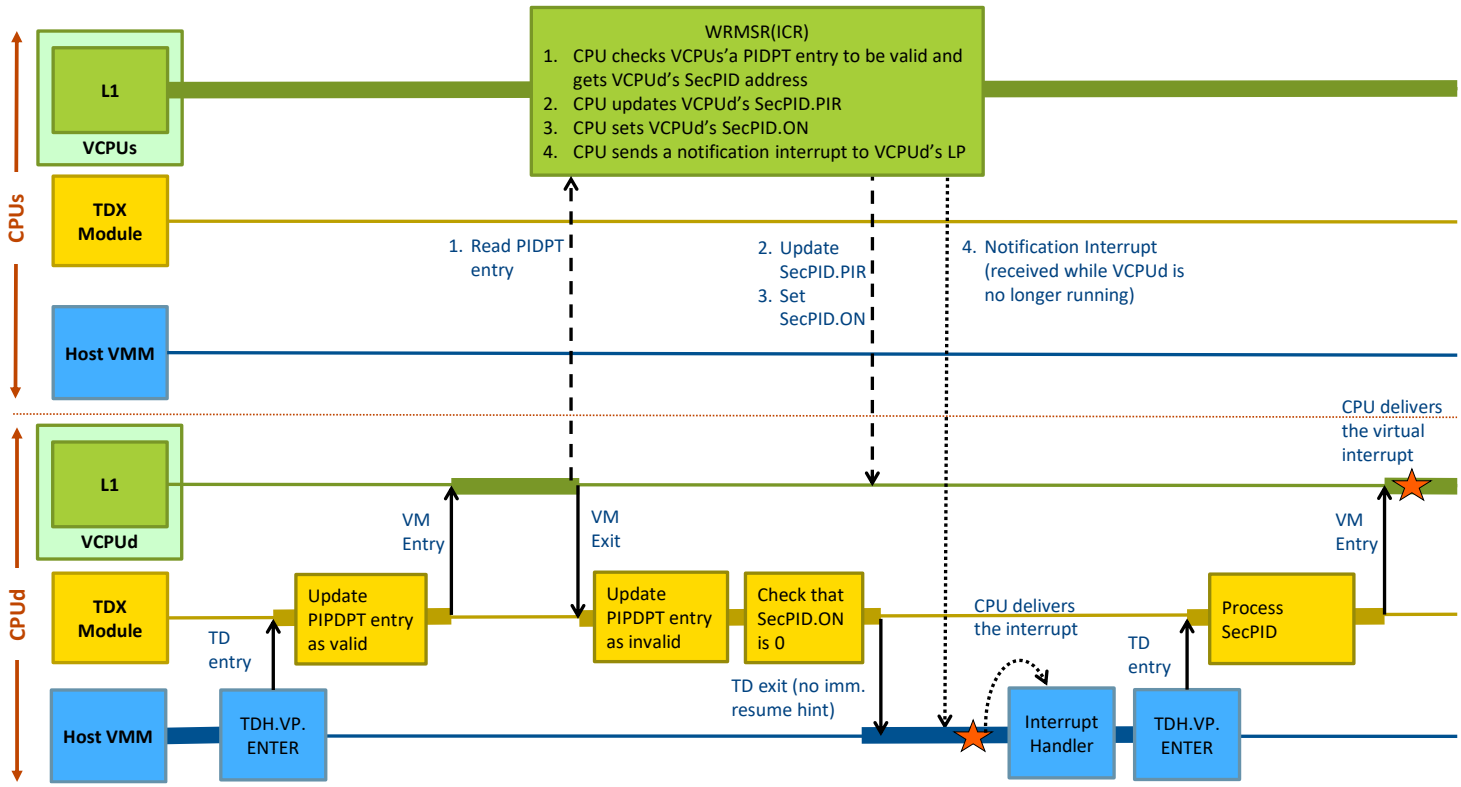

Figure 7.3: Example of an L1 IPI while Destination VCPU Just Exited

7.8.4. Reading TDVPS.VCPU_STATE_DETAILS

The host VMM may choose to use TDH.VP.READ to read TDVPS.VCPU_STATE_DETAILS after a TD exit, regardless of any immediate resume hint or interrupt. This option adds some non-negligible overhead to the TD exit/entry times.