SEV-ES Guest-Hypervisor Communication Block Standardization

Publication #	**56421**	Revision:	**2.00**
Issue Date:	**March 2021**		

Advanced Micro Devices

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

Trademarks

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

Specification Agreement

This Specification Agreement (this "Agreement") is a legal agreement between Advanced Micro Devices, Inc. ("AMD") and "You" as the recipient of the attached AMD Specification (the "Specification"). If you are accessing the Specification as part of your performance of work for another party, you acknowledge that you have authority to bind such party to the terms and conditions of this Agreement. If you accessed the Specification by any means or otherwise use or provide Feedback (defined below) on the Specification, You agree to the terms and conditions set forth in this Agreement. If You do not agree to the terms and conditions set forth in this Agreement, you are not licensed to use the Specification; do not use, access or provide Feedback about the Specification.

In consideration of Your use or access of the Specification (in whole or in part), the receipt and sufficiency of which are acknowledged, You agree as follows:

1. You may review the Specification only (a) as a reference to assist You in planning and designing Your product, service or technology ("Product") to interface with an AMD product in compliance with the requirements as set forth in the Specification and (b) to provide Feedback about the information disclosed in the Specification to AMD.
2. Except as expressly set forth in Paragraph 1, all rights in and to the Specification are retained by AMD. This Agreement does not give You any rights under any AMD patents, copyrights, trademarks or other intellectual property rights. You may not (i) duplicate any part of the Specification; (ii) remove this Agreement or any notices from the Specification, or (iii) give any part of the Specification, or assign or otherwise provide Your rights under this Agreement, to anyone else.
3. The Specification may contain preliminary information, errors, or inaccuracies, or may not include certain necessary information. Additionally, AMD reserves the right to discontinue or make changes to the Specification and its products at any time without notice. The Specification is provided entirely "AS IS." AMD MAKES NO WARRANTY OF ANY KIND AND DISCLAIMS ALL EXPRESS, IMPLIED AND STATUTORY WARRANTIES, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, TITLE OR THOSE WARRANTIES ARISING AS A COURSE OF DEALING OR CUSTOM OF TRADE. AMD SHALL NOT BE LIABLE FOR DIRECT, INDIRECT, CONSEQUENTIAL, SPECIAL, INCIDENTAL, PUNITIVE OR EXEMPLARY DAMAGES OF ANY KIND (INCLUDING LOSS OF BUSINESS, LOSS OF INFORMATION OR DATA, LOST PROFITS, LOSS OF CAPITAL, LOSS OF GOODWILL) REGARDLESS OF THE FORM OF ACTION WHETHER IN CONTRACT, TORT (INCLUDING NEGLIGENCE) AND STRICT PRODUCT LIABILITY OR OTHERWISE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.
4. Furthermore, AMD's products are not designed, intended, authorized or warranted for use as components in systems intended for surgical implant into the body, or in other applications intended to support or sustain life, or in any other application in which the failure of AMD's product could create a situation where personal injury, death, or severe property or environmental damage may occur.

5. You have no obligation to give AMD any suggestions, comments or feedback (“Feedback”) relating to the Specification. However, any Feedback You voluntarily provide may be used by AMD without restriction, fee or obligation of confidentiality. Accordingly, if You do give AMD Feedback on any version of the Specification, You agree AMD may freely use, reproduce, license, distribute, and otherwise commercialize Your Feedback in any product, as well as has the right to sublicense third parties to do the same. Further, You will not give AMD any Feedback that You may have reason to believe is (i) subject to any patent, copyright or other intellectual property claim or right of any third party; or (ii) subject to license terms which seek to require any product or intellectual property incorporating or derived from Feedback or any Product or other AMD intellectual property to be licensed to or otherwise provided to any third party.
6. You shall adhere to all applicable U.S., European, and other export laws, including but not limited to the U.S. Export Administration Regulations (“EAR”), (15 C.F.R. Sections 730 through 774), and E.U. Council Regulation (EC) No 428/2009 of 5 May 2009. Further, pursuant to Section 740.6 of the EAR, You hereby certifies that, except pursuant to a license granted by the United States Department of Commerce Bureau of Industry and Security or as otherwise permitted pursuant to a License Exception under the U.S. Export Administration Regulations ("EAR"), You will not (1) export, re-export or release to a national of a country in Country Groups D:1, E:1 or E:2 any restricted technology, software, or source code You receive hereunder, or (2) export to Country Groups D:1, E:1 or E:2 the direct product of such technology or software, if such foreign produced direct product is subject to national security controls as identified on the Commerce Control List (currently found in Supplement 1 to Part 774 of EAR). For the most current Country Group listings, or for additional information about the EAR or Your obligations under those regulations, please refer to the U.S. Bureau of Industry and Security’s website at *http://www.bis.doc.gov/*.
7. If You are a part of the U.S. Government, then the Specification is provided with “RESTRICTED RIGHTS” as set forth in subparagraphs (c) (1) and (2) of the Commercial Computer Software-Restricted Rights clause at FAR 52.227-14 or subparagraph (c) (1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.277-7013, as applicable.
8. This Agreement is governed by the laws of the State of California without regard to its choice of law principles. Any dispute involving it must be brought in a court having jurisdiction of such dispute in Santa Clara County, California, and You waive any defenses and rights allowing the dispute to be litigated elsewhere. If any part of this agreement is unenforceable, it will be considered modified to the extent necessary to make it enforceable, and the remainder shall continue in effect. The failure of AMD to enforce any rights granted hereunder or to take action against You in the event of any breach hereunder shall not be deemed a waiver by AMD as to subsequent enforcement of rights or subsequent actions in the event of future breaches. This Agreement is the entire agreement between You and AMD concerning the Specification; it may be changed only by a written document signed by both You and an authorized representative of AMD.

Contents

List of Tables

Revision History

Date	Revision	Description
March 2021	2.00	• Updated specification for version 2
March 2021	1.50	• Added SEV_FEATURES input value to SNP AP Creation o Added a statement around the expected use of the same interrupt injection mechanism as set in the BSP when creating APs using SNP AP Creation.
March 2021	1.40	• Removed AP Jump Table and AP Reset Hold deprecation from feature advertisement • Clarified when AP Jump Table and AP Reset Hold can be used • Added two new error codes to SNP Page State Change o Optional error conditions to help debug guest page state tracking o An error code to encompass other unspecified error conditions
February 2021	1.30	• Added feature advertisement support in place of versioning • Added SNP secrets page guest OS usage field definition • Removed GHCB registration option for SEV-ES guests • Clarified VALID_BITMAP requirements
January 2021	1.20	• Update the SNP Extended Guest Request to return multiple certificates • Added support for requesting an IPI with the Restricted Injection feature • Added support for requesting a timer with the Restricted Injection feature
November 2020	1.10	• Added support for AP reset hold using the MSR protocol • Restricted use of SW_SCRATCH location • Added initial support for SEV-SNP
August 2020	1.00	• Added examples of how to obtain the encryption bit position. • Clarifications related to the supported NAE event list in regard to hypervisor and guest expectations. • Clarification of how a VALID_BITMAP bit position for a GHCB quad word is calculated. • Added an example of how to trigger MMIO #NPF using reserved bits. • Added an example of how to set the starting vector (CS:IP) of an AP.
January 2020	0.85	• Added a statement of the CPUID settings that are required to be set for an SEV-ES guest (beyond normal settings) • Updated the SMP Booting documentation and introduced an AP Jump Table set/get functionality to the list of VMGEXIT software definitions.

Date	Revision	Description
June 2019	0.80	• Added a CPUID request / response protocol using the GHCB MSR for use before GHCB page is available. • Updated how NMIs are handled under SEV-ES. • Added a statement that the hypervisor must not intercept read and write access to the GHCB MSR. • Updated guest termination codes. • Added a section regarding hypervisor/VMMCALL exit requirements. • Minor formatting changes and spelling corrections.
March 2019	0.71	• Updated to the GHCB layout for improved hypercall usage. • Added a way for a guest to request termination through VMGEXIT. • Clarified GHCB Negotiation Example section. • Added documentation about ensuring exclusive access to the GHCB during VMGEXIT usage. • Added documentation about GHCB usage in NMI context.
October 2018	0.70	• Initial public release.

1 Introduction

1.1 Overview

The Secure Encrypted Virtualization - Encrypted State (SEV-ES) and the Secure Encrypted Virtualization – Secure Nested Paging (SEV-SNP) features provide protection of the virtual machine, or guest, register state from the hypervisor. The guest's register state is encrypted during world switches and cannot be directly accessed or modified by the hypervisor. SEV-ES and SEV-SNP are documented in the *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.35 and 15.36, respectively.

SEV-ES and SEV-SNP include architectural support for notifying a guest operating system (OS) when certain types of world switches are about to occur, these are called Non-Automatic Exits. This allows the guest OS to selectively share information with the hypervisor through the Guest-Hypervisor Communication Block (GHCB).

When SEV-ES or SEV-SNP is enabled, VMEXITs are classified as either an Automatic Exit (AE) or a Non-Automatic Exit (NAE) as documented in the *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.35.4. AE events are well defined and are events that do not involve or require exposing any guest register state. All other exit events are considered NAE events. For these NAE events, the guest controls what register state to expose in the GHCB.

1.2 Purpose

The purpose of this document is to standardize the GHCB memory area so that a guest OS can interoperate with any hypervisor that supports SEV-ES or SEV-SNP, to standardize on the Non-Automatic Exits that are required to be supported along with the minimum guest state to expose in the GHCB and to standardize on specific actions that might require unique support when running as an SEV-ES or SEV-SNP guest (i.e. NMI handling, SMP booting, etc.).

2 Guest-Hypervisor Communication Block (GHCB)

The GHCB must be mapped decrypted by the guest so that the guest and the hypervisor can communicate. For that reason, the GHCB is defined to be 4,096 bytes (4KB) in size so that it can be contained in a single decrypted page. The format of the GHCB attempts to mirror the SEV-ES VMCB save state area as documented in the *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Appendix B, Table B-4 (this information is represented in ***Table 3*** within this document) through offset 0x3ff. The SEV-ES VMCB save state area extends the traditional VMCB save state area to include additional guest state information. By using this format, hypervisors that support SEV-ES can map the VMCB save state area to the GHCB and limit the amount of changes required to support interacting with an SEV-ES guest. However, the GHCB and the SEV-ES save state area can diverge. Not all the data from the VMCB save state area will be required by the hypervisor, so this document proposes the required VMCB save state information that is to be provided in the GHCB during a VMGEXIT. For brevity, only the fields of the GHCB that are used will be listed. By providing only the information required for the hypervisor to successfully handle the VMGEXIT, the amount of guest state exposed to the hypervisor is limited.

SEV-SNP requires the introduction of a new GHCB protocol version. To allow for enhancements or changes to the SEV-ES GHCB protocol without requiring SEV-SNP support to be implemented, hypervisor feature support has been added.

2.1 Changes for Version 2

Version 2 includes enhancements and additions to the SEV-ES support, along with the initial support for SEV-SNP. Version 2 also introduces hypervisor feature advertisement that allows the hypervisor to advertise its SEV-ES and SEV-SNP support to the guest. This can be useful to allow a hypervisor to support future enhancements without requiring support for all prior features. Not all features are optional, there is a minimum set of features required to be implemented in Version 2, see section ***2.2***.

In addition to hypervisor feature advertisement, version 2 provides:

- SEV-ES enhancements:
    - GHCB Format Version 2:
        - The addition of the XSS MSR value (if supported) when CPUID 0xD is requested.
        - The shared area specified in the GHCB SW_SCRATCH field must reside in the GHCB SharedBuffer area of the GHCB.
    - MSR protocol support for AP reset hold.
- SEV-SNP support:
    - GHCB GPA registration
    - Page state updates

  - Guest firmware commands
  - AP (vCPU) creation from within the guest
  - Restricted interrupt/exception injection
  - Restricted APIC emulation (IPI and timer support)

2.2 Hypervisor Feature Support

Version 2 of the specification introduces advertisement of features that are supported by the hypervisor. Support for this specification in full is mandatory, unless identified as optional through the feature advertisement support. Note, some features may have a dependency on other features.

The feature support will be identified through a bitmap, FEATURES, that can be requested by the guest. Each bit will represent an optional feature. In order to allow for the FEATURES bitmap to be returned before a GHCB has been established, the FEATURES bitmap is limited to 52 bits in size so that it can be returned as part of the GHCB MSR protocol.

The following features are defined:

Table 1: FEATURES Bitmap

Bit Position	Feature Name	Feature Details
0	SEV-SNP	Support provided for base SEV-SNP support: • Preferred GHCB GPA MSR Protocol • Register GHCB GPA MSR Protocol • SNP Page State Change MSR Protocol • SNP Page State Change NAE Event • SNP Guest Request NAE Event • SNP Extended Guest Request NAE Event
1	SEV-SNP AP Creation	Support provided for SEV-SNP guest AP VMSA creation: • SNP AP Create NAE Event Requires SEV-SNP Feature.
2	SEV-SNP Restricted Injection	Support provided for SEV-SNP Restricted Injection: • SNP #HV Doorbell Page • SNP #HV IPI Requires SEV-SNP Requires SEV-SNP AP Creation

3	SEV-SNP Restricted Injection Timer	Support provided for SEV-SNP Restricted Injection Timer • SNP #HV Timer NAE Event Requires SEV-SNP Requires SEV-SNP AP Creation Requires SEV-SNP Restricted Injection

2.3 Establishing the GHCB

The GHCB location in the guest physical address space is specified by the guest. This location is made available to the hypervisor by mapping the memory as decrypted, or shared, allowing the hypervisor direct access to the memory.

The guest physical address of the GHCB is saved and restored by hardware on VMRUN/VMEXIT through the VMCB (offset 0xa0). The guest can read and write the GHCB value through MSR 0xc001_0130. The hypervisor must not intercept access to MSR 0xc001_0130, otherwise the guest will not be able to successfully establish the GHCB. The GHCB address must be 4K (page) aligned, allowing the 12 LSB bits of the GHCB address to be used for providing or requesting information between the hypervisor and the guest related to the GHCB and SEV-ES, referred to as the GHCB MSR protocol.

2.3.1 GHCB MSR Protocol

The GHCB MSR protocol is valid at any time but is most useful when the GHCB page cannot be written by the guest in an unencrypted fashion.

The GHCB MSR protocol uses the lower 12 bits of the GHCB MSR to request an action from the hypervisor. A guest must only use the request values that are documented as guest source values. Using any value that is not a guest request value must be interpreted by the hypervisor as an invalid request. Conversely, a hypervisor must only reply to the guest with response values that are documented as hypervisor source values. Using any value that is not a hypervisor response value must be interpreted by the guest as an invalid response.

Table 2. GHCB Address Destination

Field Name	Bit Position	Definition	Source	Supported Versions
GHCBInfo	11:0	0x000 – GHCB Guest Physical Address	Guest	All
		0x001 – SEV Information	Hypervisor	All
		0x002 – SEV Information Request	Guest	All
		0x004 – CPUID Request	Guest	All

		0x005 – CPUID Response	Hypervisor	All
		0x006 – AP Reset Hold Request	Guest	2+
		0x007 – AP Reset Hold Response	Hypervisor	2+
		0x010 – Preferred GHCB GPA Request	Guest	2+
		0x011 – Preferred GHCB GPA Response	Hypervisor	2+
		0x012 – Register GHCB GPA Request	Guest	2+
		0x013 – Register GHCB GPA Response	Hypervisor	2+
		0x014 – SNP Page State Change Request	Guest	2+
		0x015 – SNP Page State Change Response	Hypervisor	2+
		0x080 – Hypervisor Feature Support Request	Guest	2+
		0x081 – Hypervisor Feature Support Response	Hypervisor	2+
		0x100 – Termination Request	Guest	All
GHCBData	63:12	Value dependent upon GHCBInfo		

- GHCBInfo:
  - 0x000 – GHCB Guest Physical Address
    - GHCBData[63:12] specifies bits [63:12] of the guest physical address of the GHCB (this implies that the GHCB must be 4K aligned).
  - 0x001 – SEV Information
    - GHCBData[63:48] specifies the maximum GHCB protocol version supported
    - GHCBData[47:32] specifies the minimum GHCB protocol version supported
    - GHCBData[31:24] specifies the SEV page table encryption bit number

      Written by the hypervisor before the GHCB address is established (such as on vCPU creation) in order to present the guest with the capabilities of the hypervisor. The guest will choose an appropriate version, within the range supplied by the hypervisor, and set the GHCB Protocol Version field of the GHCB. If the guest cannot support the protocol range supplied by the hypervisor, it should terminate.

      The SEV page table encryption bit number is required by the guest when building the page tables before entering long mode. Normally, the SEV page table encryption bit number is obtained using the CPUID instruction, which will now result in a VMM Communication exception. Without knowing the position of the encryption bit, the GHCB page cannot be marked as decrypted to allow for communication with the hypervisor. Because of this, the hypervisor must supply the page table bit encryption bit number to the guest. This value can be obtained by the hypervisor from CPUID function 0x8000_001f, register EBX[5:0]. Alternatively, for CPUID instructions that are required before the GHCB can be established, the guest can use the CPUID request protocol documented below and is recommended for an SEV-SNP guest.

- 0x002 –SEV Information Request
  - Written by the guest to request the hypervisor provide the SEV information (GHCBInfo = 0x001) needed to perform protocol negotiation.
- 0x004 – CPUID Request
  - GHCBData[63:32] – CPUID function
  - GHCBData[31:30] – Requested CPUID register value
    - 0b00 – EAX
    - 0b01 – EBX
    - 0b10 – ECX
    - 0b11 – EDX
  - GHCBData[29:12] – Reserved, must be zero

    Written by the guest to request a CPUID function register value from the hypervisor. This is useful if CPUID information is required before the GHCB can be established by the guest. Since only a single register value can be returned at a time, multiple VMGEXIT invocations are required to obtain all register values.

    The CPUID request protocol does not support CPUID functions that require non-zero sub-leafs. Additionally, CPUID function 0x0000_000D is not supported as it requires the value of XCR0.
- 0x005 – CPUID Response
  - GHCBData[63:32] – CPUID function register value
  - GHCBData[31:30] – Returned CPUID register value
    - 0b00 – EAX
    - 0b01 – EBX
    - 0b10 – ECX
    - 0b11 – EDX

    - GHCBData[29:12] – Reserved, must be zero

      Written by the hypervisor in response to a CPUID request to return the requested CPUID function register value.

- 0x006 – AP Reset Hold Request
    - GHCBData[63:12] – MBZ

      Written by the guest to request the AP be place in a HLT loop awaiting an INIT-SIPI-SIPI request. This allows an AP to request AP reset hold without having to be in long mode. See section *4.3.1* for a description of the AP reset hold NAE event.

      Not valid for an SEV-SNP guest running with the Restricted Injection feature. A hypervisor should treat this as an unsupported request in this situation.
- 0x007 – AP Reset Hold Response
    - GHCBData[63:12] – Must be non-zero

      Written by the hypervisor after an INIT-SIPI-SIPI sequence has been received for the targeted AP to take it out of HLT. See section *4.3.1* for a description of the AP reset hold NAE event.
- 0x010 – Preferred GHCB GPA Request
    - GHCBData[63:12] – Must be zero

      Written by the guest to request the GHCB guest physical address (GHCB GPA) that the hypervisor prefer to be used for the vCPU invoking the VMGEXIT. See section *2.3.2* for further details and restrictions.
- 0x011 – Preferred GHCB GPA Response
    - GHCBData[63:12] – Preferred GHCB GFN

      Written by the hypervisor in response to a preferred GHCB GPA request. See section *2.3.2* for further details and restrictions.
- 0x012 – Register GHCB GPA Request
    - GHCBData[63:12] – GHCB GFN to register

      Written by the guest to request the GHCB guest physical address (GHCB GPA = GHCB GFN << 12) be registered for the vCPU invoking the VMGEXIT. See section *2.3.2* for further details and restrictions.
- 0x013 – Registger GHCB GPA Response
    - GHCBData[63:12] – Registered GHCB GFN

      Written by the hypervisor in response to a register GHCB GPA request. See section *2.3.2* for further details and restrictions.

- 0x014 – SNP Page State Change Request
  - GHCBData[63:56] – Reserved, must be zero
  - GHCBData[55:52] – Page operation
    - 0x0001 – Page assignment, Private
    - 0x0002 – Page assignment, Shared
  - GHCBData[51:12] – Guest physical frame number

    Written by the guest to request a page state change from the hypervisor based on the operation requested. The page will be acted on as a 4K page. The page state change will be against the system physical address (SPA) that is used to back the guest physical address (GPA) calculated from the supplied guest physical frame number (GFN) for the requested page size. See section ***4.1.6*** for a description of the page operations.
- 0x015 – SNP Page State Change Response
  - GHCBData[63:32] – Error code
  - GHCBData[31:12] – Reserved, must be zero

    Written by the hypervisor in response to a Page State Change request. Any non-zero value for the error code indicates that the page state change was not successful.
- 0x080 – Hypervisor Feature Support Request
  - GHCBData[63:12] – Reserved, must be zero

    Written by the guest to request the hypervisor feature support bitmap.
- 0x081 – Hypervisor Feature Support Response
  - GHCBData[63:12] – FEATURES bitmap

    Written by the hypervisor in response to a Hypervisor Feature Support request. See section ***2.2***.
- 0x100 – Termination Request
  - Written by the guest to communicate to the hypervisor that the guest is requesting termination. The guest should expect the hypervisor to comply with the request for termination. As a safeguard, it is recommended that the guest incorporate a HLT loop or SHUTDOWN following the VMGEXIT. GHCBData contains the termination reason code where GHCBData[15:12] specifies the reason code set and GHCBData[23:16] contains the reason code from that reason code set.

    The reason code set is meant to provide hypervisors with their own termination reason codes. This document defines and owns reason code set 0x0 and the following reason codes (GHCBData[23:16]):

- 0x00 – General termination request
- 0x01 – SEV-ES / GHCB Protocol range is not supported.
- 0x02 – SEV-SNP features not supported

2.3.2 GHCB GPA Registration

GHCB GPA registration is only supported and required for SEV-SNP guests. Some hypervisors may prefer that a guest use a consistent and/or specific GPA for the GHCB associated with a vCPU. Therefore, a guest must use the Register GHCB GPA request MSR protocol function to establish the GHCB GPA before using a GHCB for a VMGEXIT for the first time.

The registration request allows for two actions to be performed.

- Get preferred GHCB GPA

  A guest may request the hypervisor preferred GHCB GPA for the vCPU by setting GHCBData to 0 and GHCBInfo to 0x10. The hypervisor must respond with a GFN or 0xfffffffffffff (the hypervisor does not have a preferred GHCB GPA) in GHCBData and 0x11 in GHCBInfo. If the hypervisor responds with a GFN, the hypervisor must guarantee that the value cannot otherwise be mapped by the guest. For an SEV-SNP guest, the hypervisor must update the RMP table to make the page a hypervisor owned (shared) page. The guest is not required to use the preferred GHCB GPA, but should the guest decide to use it, it should verify that the new GPA is outside of its known memory range.

- Register GHCB GPA

  Before using a GHCB GPA for a vCPU the first time, a guest must register the vCPU GHCB GPA. If the guest is not using the hypervisor preferred GHCB GPA, the guest must make the GHCB page a hypervisor owned (shared) page. Registration is performed by setting GHCBData to the GFN of GHCB GPA (which could be the hypervisor preferred value) and GHCBInfo to 0x12. The hypervisor must respond with the same GFN value or 0xfffffffffffff (the hypervisor does not accept the supplied GHCB GPA) in GHCBData and 0x13 in GHCBInfo.

A hypervisor must terminate the guest if the guest supplies a GHCB GPA that does not match the currently registered version of the GHCB GPA for the vCPU.

2.4 GHCB Negotiation Example

The guest will ultimately provide the GPA of the GHCB page via the GHCB MSR. The hypervisor will obtain this GPA value by reading offset 0x00a0 of the VMCB. Initially, however, the hypervisor can set the GHCB MSR to allow for the GHCB protocol to be negotiated. This example assumes that the hypervisor performs its current steps when preparing to create and start a vCPU and that use of the CPUID instruction is intercepted. The following additional steps document an example for the GHCB negotiation.

2.4.1 SEV-ES (Version 1)

- Hypervisor sets VMCB offset 0x00a0 before launching the vCPU for the first time:
  - The value is used by the guest to negotiate the SEV-ES/GHCB protocol version and establish the page table encryption bit.
  - Given the hypervisor supports only version 1 and the SEV page table encryption bit number is 47 (0x2f). The hypervisor will use GHCBInfo value of 0x001 and set VMCB offset 0x00a0 to:
    - 0x0001_0001_2f00_0001
- Hypervisor launches the guest vCPU (VMRUN).
- Guest determines the encryption bit position in order to properly set up the page tables and mark the GHCB as shared.
  - Guest establishes an exception handler for #VC exceptions
    - Guest will perform a series of CPUID instructions in order to obtain the SEV data. For an SEV-ES guest, these CPUID instructions result in a #VC exception, where the CPUID instructions will be emulated.
  - Guest issues CPUID for leaf 0x80000000:
    - EAX is set to 0x8000001f
    - #VC handler returns
  - Guest issues CPUID for leaf 0x8000001f:
    - Guest #VC exception handler reads MSR 0xC001_0130
    - If GHCBInfo != 0x001:
      - Guest requests termination
    - Guest extracts the maximum GHCB protocol version, GHCBData[63:48], and minimum GHCB protocol version, GHCBData[47:32]. If the guest cannot support a protocol in the range:
      - Guest requests termination
    - Guest extracts the SEV page table encryption bit number, GHCBData[31:24]
    - EAX is set to 0x0000000a
      - SEV and SEV-ES supported
    - EBX is set to the SEV page table encryption bit

        - #VC handler returns
- Guest continues initialization, which, among other things, includes:
    - Ensuring that 64-bit long mode is established
    - Page tables are configured with the encryption bit as required
    - GHCB page is allocated and marked shared in the page tables:
        - Guest writes MSR 0xC001_0130 with the GPA of the allocated GHCB page (GHCBInfo == 0x000)
        - Must be done before a VMGEXIT instruction is issued that uses the GHCB page

The above example is just one way to perform the GHCB negotiation for an SEV-ES guest. For example, you could use the GHCBInfo = 0x004 CPUID Request to obtain the actual values for the CPUID instructions executed by the guest. Or you could use the GHCBInfo = 0x002 Request for SEV Information if MSR 0xC001_0130 does not contain the GHCBInfo = 0x001 SEV Information.

2.4.2 SEV-SNP (Version 2)

The hypervisor may supply the encryption bit position using the SEV Information MSR protocol, but the guest should use the CPUID information supplied in the CPUID Page to determine the encryption bit position. The information in the CPUID Page will have been validated by the SEV firmware.

- Hypervisor sets VMCB offset 0x00a0 before launching the vCPU for the first time:
    - The value is used by the guest to negotiate the SEV-ES/GHCB protocol version and establish the page table encryption bit.
    - Given the hypervisor supports up to version 2 and the SEV page table encryption bit number is 51 (0x33). The hypervisor will use GHCBInfo value of 0x001 and set VMCB offset 0x00a0 to:
        - 0x0002_0001_3300_0001
- Hypervisor launches the guest vCPU (VMRUN).
- Guest determines the encryption bit position in order to properly set up the page tables and mark the GHCB as shared.
    - Guest establishes an exception handler for #VC exceptions
        - Guest will perform a series of CPUID instructions in order to obtain the SEV data. For an SEV-SNP guest, these CPUID instructions result in a #VC exception, where the CPUID instructions will be emulated by setting the register values to the values set in the CPUID Page.

  - Guest issues CPUID for leaf 0x80000000:
    - #VC exception handler sets EAX/EBX/ECX/EDX for leaf 0x80000000
    - #VC exception handler returns
  - Guest verifies that EAX is at least 0x8000001F
  - Guest issues RDMSR for MSR 0xC001_0131 (non-interceptable SEV Status MSR)
    - Guest verifies that it is running as an SEV-SNP guest by verifying that EAX[2:0] == 0x7
  - Guest issues CPUID for leaf 0x8000001F:
    - #VC exception handler sets EAX/EBX/ECX/EDX for leaf 0x8000001F
    - #VC exception handler returns
    - Guest extracts the SEV page table encryption bit position, EBX[5:0]
  - Guest uses the MSR Protocol support to issue an SEV Information Request
    - Guest extracts the maximum GHCB protocol version, GHCBData[63:48], and minimum GHCB protocol version, GHCBData[47:32]. If the guest cannot support a protocol in the range:
      - Guest requests termination

- Guest continues initialization, which, among other things, includes:
  - Ensuring that 64-bit long mode is established
  - Page tables are configured with the encryption bit as required
  - GHCB page is allocated and marked shared in the page tables:
    - Guest uses MSR Protocol to perform Register GHCB GPA Request using the GFN of the allocated GHCB page. If the hypervisor does not respond with the input GFN:
      - Guest requests termination
    - Guest writes MSR 0xC001_0130 with the GPA of the allocated GHCB page (GHCBInfo == 0x000)
    - Must be done before a VMGEXIT instruction is issued that uses the GHCB page

The above example is just one way to perform the GHCB negotiation for an SEV-SNP guest.

2.5 GHCB/VMGEXIT Example

The following shows an example of a guest and hypervisor that conform to version 1 of the GHCB protocol:

- Guest executes an instruction resulting in a #VC exception
    - Guest #VC handler is invoked
    - Guest #VC handler disables preemption and interrupts
    - Guest #VC handler ensures that the physical address of the GHCB is set in MSR 0xC001_0130
    - Guest #VC handler clears any previous GHCB field invocation data
    - Guest #VC handler sets the GHCB fields as required for the instruction
    - Guest #VC handler issues VMGEXIT
- Hypervisor resumes with a VMEXIT code of VMEXIT_VMGEXIT
    - Hypervisor reads VMCB offset 0x00a0 to obtain the guest physical address of the GHCB
    - If GHCBInfo == 0x000
        - Hypervisor translates GHCB guest physical address into a GHCB hypervisor virtual address, handles the exit based on the GHCB SW_EXITCODE, updates the GHCB save state area and resumes the guest.
    - If GHCBInfo == 0x002
        - Hypervisor recreates the GHCB protocol versioning value, sets this value in the VMCB at offset 0x00a0 and resumes the guest.
    - If GHCBInfo == 0x004
        - Hypervisor creates a CPUID response to the CPUID request, sets this value in the VMCB at offset 0x00a0 and resumes the guest.
    - If GHCBInfo == 0x100
        - Hypervisor terminates the guest, optionally displaying the associated GHCBData value.
    - If GHCBInfo is any other value
        - Hypervisor will be unable to process the VMGEXIT and should terminate the guest.
- Guest #VC handler resumes processing
    - Guest copies the GHCB save state information to the guest register state
    - Guest enables interrupts and preemption
    - Guest exits the #VC handler

When a guest is running as an SEV-ES guest, it is important that the guest not do anything that would result in an unplanned NAE event before entering long mode or 32-bit PAE. When not in one of these modes, all memory accesses by the guest are forced to use encryption under the key associated with the guest. As a result, the guest and hypervisor would not be able to communicate through the GHCB since the hypervisor would see encrypted data. The guest should determine the position of encryption bit so that the GHCB can be properly established. One way to perform this would be:

- ○ Issue CPUID for function 0x8000_0000 and verify CPUID function 0x8000_001F is available.
  - ▪ If the CPUID instruction is being intercepted, this will result in a #VC, where the CPUID exchange protocol can be used to obtain the CPUID results.
- ○ Issue CPUID for function 0x8000_001F and obtain the encryption bit position.
  - ▪ If the CPUID instruction is being intercepted, this will result in a #VC, where the CPUID exchange protocol can be used to obtain the CPUID results.

This is not the only way this can be done. If a #VC is encountered, then software would know that it is running as an SEV-ES guest and could use GHCBInfo 0x002 to request the SEV information to obtain the encryption bit position.

2.6 GHCB Layout

Table 3. GHCB Layout

Offset	Size	Contents	Notes	Supported Versions
0x0000	0xcb	RESERVED		
0x00cb	0x01	CPL		All
0x00cc	0x74	RESERVED		
0x0140	0x08	XSS		2+
0x0148	0x18	RESERVED		
0x0160	0x08	DR7		All
0x0168	0x90	RESERVED		
0x01f8	0x08	RAX		All
0x0200	0x100	RESERVED		
0x0300	0x08	RESERVED (RAX already available at 0x01f8)		
0x0308	0x08	RCX		All
0x0310	0x08	RDX		All
0x0318	0x08	RBX		All

Offset	Size	Contents	Notes	Supported Versions
0x0320	0x70	RESERVED		
0x0390	0x08	SW_EXITCODE	Guest controlled exit code	All
0x0398	0x08	SW_EXITINFO1	Guest controlled exit information 1	All
0x03a0	0x08	SW_EXITINFO2	Guest controlled exit information 2	All
0x03a8	0x08	SW_SCRATCH	Guest controlled additional information	All
0x03b0	0x38	RESERVED		
0x03e8	0x08	XCR0		All
0x03f0	0x10	VALID_BITMAP	Bitmap to indicate valid qwords in the save state area starting from offset 0x000 through offset 0x3ef (126 qwords)	All
0x0400	0x08	X87_STATE_GPA	Guest physical address of a page containing X87 related state information conforming to the format produced by the XSAVE instruction.	All
0x0408	0x3f8	RESERVED		
0x0800	0x7f0	RESERVED / Shared Buffer	Can be used as a shared buffer area. Future versions of the GHCB specification will not alter this area definition.	All
0x0ff0	0x0a	RESERVED		
0x0ffa	0x02	SEV-ES/GHCB Protocol Version	Version of the SEV-ES/GHCB layout used by the guest • 0x0001 – SEV-ES/GHCB Protocol Version 1 • 0x0002 – SEV-ES/GHCB Protocol Version 2	All

Offset	Size	Contents	Notes	Supported Versions
0x0ffc	0x04	GHCB Usage	Provides an indicator of the usage and format of the GHCB: • 0x00000000 – The GHCB page follows the format as documented here • Any other value can be used by the hypervisor, which can determine its own format (e.g. for hypercall usage) On VMGEXIT, the hypervisor should check the GHCB Usage field and validate that is a supported value. A hypervisor must support the GHCB Usage value 0x0000 and may support other values. For any unsupported value, the hypervisor can either terminate the guest or resume the guest indicating an exception should be raised. The details of how hypervisors communicate support for additional GHCB Usage values is beyond the scope of this document.	All

2.7 SEV-SNP Secrets Page

The hypervisor should supply an SEV-SNP guest with a secrets page as part of the SEV-SNP launch process (see *SEV Secure Nested Paging Firmware ABI Specification*). Within the SNP secrets page, there is a 96-byte area from offset 0x00a0 to 0x00ff reserved for use by the SEV-SNP guest. To avoid usage conflicts, the format and use of this area is defined as follows:

- Four 4-byte entries, one per VMPL level, beginning at offset 0x00, of the current guest request message sequence number.
    - This allows for communicating the current message sequence number between operating environments, e.g. when transferring control from UEFI to the OS, so that the next environment can successfully issue guest requests.
- One 8-byte entry to hold the physical address of the AP Jump Table beginning at offset 0x10.
    - This allows for communicating the physical address of the AP Jump Table between operating environments without hypervisor involvement.
- 40-bytes reserved for future use beginning at offset 0x18.
- 32-bytes for use by the guest in any manner desired beginning at offset 0x40.

The format of the area is:

```
struct secrets_page_os_area {
        uint32 vmpl0_message_seq_num;
        uint32 vmpl1_message_seq_num;
        uint32 vmpl2_message_seq_num;
        uint32 vmpl3_message_seq_num;

        uint64 ap_jump_table_pa;

        uint8 reserved[40];

        uint8 guest_usage[32];
};
```

3 Guest Exits

3.1 Automatic Exits (AE)

Table 4. List of Automatic Exits

Code	Name	Description
0x52	VMEXIT_MC	Machine check exception
0x60	VMEXIT_INTR	Physical interrupt
0x61	VMEXIT_NMI	Physical NMI
0x63	VMEXIT_INIT	Physical INIT
0x64	VMEXIT_VINTR	Virtual INTR
0x77	VMEXIT_PAUSE	PAUSE instruction
0x78	VMEXIT_HLT	HLT instruction
0x7f	VMEXIT_SHUTDOWN	Shutdown
0x8f	VMEXIT_EFER_WRITE_TRAP	
0x90 – 0x9f	VMEXIT_CR[0-15]_WRITE_TRAP	
0x400	VMEXIT_NPF	Only if PFCODE[3] == 0 (no reserved bit error)
0x403	VMEXIT_VMGEXIT	VMGEXIT instruction
-1	VMEXIT_INVALID	Invalid guest state
-2	VMEXIT_BUSY	Busy bit was set in guest state

Refer to *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.35.4 for information on how the guest RIP is advanced when an AE exit is encountered.

3.2 Guest Non-Automatic Exits (NAE)

NAE events are all exit events that are not AE events. When an NAE event occurs, the VMM Communication Exception (#VC) is always thrown by the hardware when an SEV-ES guest is running. The error code of the #VC exception is equal to the VMEXIT code of the event that caused the NAE.

The guest should inspect the error code to determine the cause of the exception, decide what register state needs to be copied to the GHCB and then invoke the VMGEXIT instruction to generate an AE event. After a subsequent VMRUN instruction by the hypervisor the guest will resume at the next instruction following the VMGEXIT instruction. This provides the guest an opportunity to examine the results provided from the hypervisor in the GHCB and copy them back to its internal state. The #VC handler exits using the IRET instruction, therefore the IRET instruction should not be intercepted (with exception for an NMI which is discussed in a subsequent section).

4 GHCB Protocol

This document will provide the definition for the supported versions of the GHCB protocol that will establish the guest and hypervisor requirements. This will consist of the list of required NAE events that the guest and the hypervisor must support, as well as the required guest state that will be provided by the guest and returned by the hypervisor during a VMGEXIT. In general, the SW_EXITCODE will map to the SVM intercept exit codes. There are some exceptions where a user-defined SW_EXITCODE will be used to provide additional needed information to the hypervisor.

The following table lists the NAE events that are valid for GHCB protocol, along with the version associated. A hypervisor is not required to intercept the instructions that generate all the listed NAE events, but since a guest can invoke VMGEXIT without having taken a #VC, the hypervisor must be able to handle a VMGEXIT from the guest for the listed NAE events. The hypervisor can decide if the VMGEXIT event is valid and respond appropriately. A guest must be able to handle a #VC exception for all the NAE events listed. It is up to the guest to decide how to handle the NAE event. For example, a guest may decide that it should never receive a particular NAE event and, instead of performing VMGEXIT processing, can perform some alternate processing.

The state to and from the hypervisor in the NAE event table is the minimum state information required. Each GHCB field set by the guest and returned by the hypervisor must have the appropriate bit set in the GHCB VALID_BITMAP field. At a minimum:

- The guest is expected to set the bits for SW_EXITCODE, SW_EXITINFO1 and SW_EXITINFO2.
- The hypervisor is expected to set the bits for SW_EXITINFO1 and SW_EXITINFO2.

Before each NAE event, the guest should reset the VALID_BITMAP field, by clearing all bits, before setting the bits associated with the NAE event to be performed. Likewise, the hypervisor should reset the VALID_BITMAP field before setting the bits associated with the results of the NAE event.

The VALID_BITMAP bit position is calculated by taking the offset of the field in bytes and dividing by 8, giving the qword offset. Given the qword offset, the byte offset and bit position within the VALID_BITMAP are calculated. The byte offset is the qword offset divided by 8, while the bit position within the byte offset is the qword offset mod 8, e.g.:

- RAX is offset 0x01f8, 0x01f8 / 8 = 0x3f or 63
- VALID_BITMAP byte offset is 63 / 8 = 7
- VALID_BITMAP bit position within the byte offset is 63 % 8 = 7

The guest and hypervisor can supply additional state if desired but must not rely on that additional state being provided. Unless otherwise specified in the table below, SW_EXITINFO1 and SW_EXITINFO2 must be set to 0.

Table 5. List of Supported Non-Automatic Events

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions
DR7 Read	SW_EXITCODE = 0x27		See section 4.5	All

DR7 Write	RAX
	SW_EXITCODE = 0x37
	SW_EXITINFO1
	SW_EXITINFO2 = 0
			See section 4.5
			
			SW_EXITINFO1 will be set as documented in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.8.1
				All

RDTSC	SW_EXITCODE = 0x6e
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		RAX
		RDX
				All

RDPMC	RCX
	SW_EXITCODE = 0x6f
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		RAX
		RDX
				All

CPUID	RAX
	RCX
	SW_EXITCODE = 0x72
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
	XCR0*
		RAX
		RBX
		RCX
		RDX
			XCR0 is only required to be supplied when a request for CPUID 0000_000D is made.	1

	RAX RCX SW_EXITCODE = 0x72 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0 XCR0* XSS*	RAX RBX RCX RDX	XCR0 is only required to be supplied when a request for CPUID 0000_000D is made. XSS is only required to be supplied when a request for CPUID 0000_000D is made and the guest supports the XSS MSR (0x0000_0DA0).	2+

INVD	SW_EXITCODE = 0x76
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
				All

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions
IOIO_PROT	RAX (for OUT) SW_EXITCODE = 0x7b SW_EXITINFO1 SW_EXITINFO2 SW_SCRATCH = <ADDR>	RAX (for IN)	SW_EXITINFO1 will be set as documented in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.10.2 If string-based port access is indicated in SW_EXITINFO1, SW_EXITINFO2 will contain the REP count, otherwise 0 If string-based port access is indicated in SW_EXITINFO1, SW_SCRATCH will have the SRC (OUTS) or DST (INS) guest physical address of shared memory See section ***4.1.2***	All

MSR_PROT (RDMSR)	RCX
	SW_EXITCODE = 0x7c
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		RAX
		RDX
			See section ***4.1.3***	All

MSR_PROT (WRMSR)	RAX
	RCX
	RDX
	SW_EXITCODE = 0x7c
	SW_EXITINFO1 = 1
	SW_EXITINFO2 = 0
			See section ***4.1.3***	All

VMMCALL	RAX
	CPL
	SW_EXITCODE = 0x81
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		RAX	RAX and CPL are the minimum required state to be provided to the hypervisor.
			The guest can supply additional information as required by the hypercall and indicate that in VALID_BITMAP.
			See section ***4.1.4***
				All

RDTSCP	SW_EXITCODE = 0x87
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		RAX
		RCX
		RDX
				All

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

WBINVD	SW_EXITCODE = 0x89
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
				All

MONITOR/
MONITORX
	RAX
	RCX
	RDX
	SW_EXITCODE = 0x8a
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
			RAX will contain the guest physical address of the MONITOR/MONITORX memory range.	All

MWAIT/
MWAITX
	RAX
	RCX
	SW_EXITCODE = 0x8b
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
				All

#AC			The #VC handler should forward this exception on to the #AC handler.	All

#NPF
MMIO READ
	SW_EXITCODE = 0x8000_0001
	SW_EXITINFO1 = <SRC>
	SW_EXITINFO2 = <LEN>
	SW_SCRATCH = <DST>
			SW_EXITINFO1 will have the SRC guest physical address
			SW_EXITINFO2 must be less than or equal to 0x7fffffff for version 1 and less than or equal to 0x8 for all other versions.
			SW_SCRATCH will have the DST guest physical address of shared memory
			See section ***4.1.5***
				All

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

#NPF/
MMIO WRITE
	SW_EXITCODE = 0x8000_0002
	SW_EXITINFO1 = <DST>
	SW_EXITINFO2 = <LEN>
	SW_SCRATCH = <SRC>
			SW_EXITINFO1 will have the DST guest physical address
			SW_EXITINFO2 must be less than or equal to 0x7fffffff for version 1 and less than or equal to 0x8 for all other versions.
			SW_SCRATCH will have the SRC guest physical address of shared memory
			See section ***4.1.5***
				All

NMI Complete	SW_EXITCODE = 0x8000_0003
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
			See section ***4.4***	All

AP Reset Hold	SW_EXITCODE = 0x8000_0004
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		SW_EXITINFO2	SW_EXITINFO2 =
			State from Hypervisor:
			0 - the vCPU did not receive a SIPI request.
			Non-zero - the vCPU receive a SIPI request.
			Not valid for an SEV-SNP guest running with the Restricted Injection feature. A hypervisor should treat this as an unsupported NAE event in this situation.
			See section ***4.3.1***
				All

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions
AP Jump Table	SW_EXITCODE = 0x8000_0005 SW_EXITINFO1 SW_EXITINFO2	SW_EXITINFO2	SW_EXITINFO1 = 0 (SET) SW_EXITINFO2 = **State to Hypervisor**: the guest physical address to be SET **State from Hypervisor**: 0 SW_EXITINFO1 = 1 (GET) SW_EXITINFO2 = **State to Hypervisor**: 0 **State from Hypervisor**: the guest physical address as previously SET (or zero if not previously SET) Not valid for an SEV-SNP guest. A hypervisor should treat this as an unsupported NAE event in this situation. See section ***4.3.1***	All

SNP Page State Change	SW_EXITCODE = 0x8000_0010
	SW_SCRATCH = <DATA>
		SW_EXITINFO2	SW_SCRATCH will have the guest physical address of a Page State Change structure residing in shared memory.
			
			SW_EXITINFO2 will contain 0 if all entries have been processed successfully or a reason code identifying the why the request has not completed.
			
			See section ***4.1.6***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

SNP Guest Request	SW_EXITCODE = 0x8000_0011
	SW_EXITINFO1 = <REQ GPA>
	SW_EXITINFO2 = <RSP GPA>
		SW_EXITINFO2	SW_EXITINFO1 will have the guest physical address of the request page
			SW_EXITINFO2
			State to Hypervisor: will have the guest physical address of the response page
			State from Hypervisor: Upper 32-bits (63:32) will be 0. Lower 32-bits (31:0) will contain the return code from the firmware call (0 = success)
			Both the request page and response page must be assigned to the hypervisor (shared).
			See section ***4.1.7***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

SNP Extended Guest Request	RAX = <DATA GPA>
	RBX = <DATA PAGE COUNT>
	SW_EXITCODE = 0x8000_0012
	SW_EXITINFO1 = <REQ GPA>
	SW_EXITINFO2 = <RSP GPA>
		RBX
		SW_EXITINFO2
			RAX will have the guest physical address of the page(s) to hold returned data
			RBX
			State to Hypervisor: will contain the number of guest contiguous pages supplied to hold returned data
			State from Hypervisor: on error will contain the number of guest contiguous pages required to hold the data to be returned
			SW_EXITINFO1 will have the guest physical address of the request page
			SW_EXITINFO2
			State to Hypervisor: will have the guest physical address of the response page
			State from Hypervisor: Upper 32-bits (63:32) will contain the return code from the hypervisor. Lower 32-bits (31:0) will contain the return code from the firmware call (0 = success)
			The request page, response page and data page(s) must be assigned to the hypervisor (shared).
			See section ***4.1.8***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

SNP AP Creation	RAX = <SEV_FEATURES>
	SW_EXITCODE = 0x8000_0013
	SW_EXITINFO1
	SW_EXITINFO2
			RAX, on **CREATE/ADD**, will contain the SEV_FEATURES value used in the VMSA specified in SW_EXITINFO2.
			
			SW_EXITINFO1[63:32] = APIC ID
			SW_EXITINFO1[31:0] =
			0 (CREATE/ADD)
			VMSA state to be used with the next INIT-SIPI.
			Not valid for an SEV-SNP guest running with the Restricted Injection feature. A hypervisor should treat this as an unsupported NAE event in this situation.
			1 (CREATE/ADD)
			VMSA state to be used immediately (issue VMRUN).
			2 (DESTROY/REMOVE)
			Remove VMSA state (vCPU is no longer runnnable).
			
			SW_EXITINFO2 is the guest physical address of the VMSA to be for the vCPU associated with the specified APID ID (0 for a destroy/remove request).
			
			See section ***4.1.9*** and ***4.3.2***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions
#HV Doorbell Page	SW_EXITCODE = 0x8000_0014 SW_EXITINFO1 SW_EXITINFO2	SW_EXITINFO2	SW_EXITINFO1 = **0 (GET_PREFERRED)** SW_EXITINFO2 **State from hypervisor**: is the hypervisor preferred guest physical address to use for the doorbell page. **1 (SET)** SW_EXITINFO2 **State to hypervisor**: is the guest physical address to use for the doorbell page. **2 (QUERY)** SW_EXITINFO2 **State from hypervisor**: is the guest physical address that is in use for the doorbell page. **3 (CLEAR)** See section ***4.1.10***	2+

#HV IPI	SW_EXITCODE = 0x8000_0015
	SW_EXITINFO1
	SW_EXITINFO2 = 0
			SW_EXITINFO1 will be set to the x2APIC Interrupt Command Register format.
			See section ***4.1.11***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions
#HV Timer	RAX RBX RCX SW_EXITCODE = 0x8000_0016 SW_EXITINFO1 SW_EXITINFO2 = <MASK>	RAX RBX RCX RDX	SW_EXITINFO1 = **0 (SET)** Set register(s) as identified by the mask in SW_EXITINFO2 (Current Count is not settable) **1 (GET)** Get register(s) as identified by the mask in SW_EXITINFO2 The registers follow the format of the APIC Timer Registers as documented in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 16.4.1 and 10.5.4: MASK[0] RAX = Timer LVT MASK[1] RBX = Divide Configuration MASK[2] RCX = Timer Initial Count MASK[3] RDX = Timer Current Count See section ***4.1.12***	2+

Hypervisor Feature Support	SW_EXITCODE= 0x8000_FFFD
	SW_EXITINFO1 = 0
	SW_EXITINFO2 = 0
		SW_EXITINFO2	SW_EXITINO2
			State from Hypervisor: is the FEATURES bitmap.
			See section ***2.2***
				2+

Termination Request	SW_EXITCODE = 0x8000_FFFE
	SW_EXITINFO1 = <ERR_CODE>
	SW_EXITINFO2 = <ERR_INFO>
			SW_EXITINFO1 will have the termination reason code
			SW_EXITINFO2 may contain an error information code
			See section ***4.1.13***
				2+

NAE Event	State to Hypervisor	State from Hypervisor	Notes	Supported Versions

Unsupported Event	SW_EXITCODE = 0x8000_FFFF
	SW_EXITINFO1 = <ERR_CODE>
	SW_EXITINFO2 = 0
			SW_EXITINFO1 will have the error code on entry to the VMM Communication exception	All

4.1 Invoking VMGEXIT

In general, all NAE events are handled in a standard fashion, except for a few. The standard method is documented in Section 4.1.1. The exceptions are documented following the standard method. The guest has the option of using the #VC handler to trigger VMGEXIT processing or it can para-virtualize the instructions that would cause a #VC and, instead, invoke VMGEXIT processing directly.

Software should ensure that an invocation of VMGEXIT is protected on the vCPU that it will be issued from. For that reason, software should disable interrupts and disable preemption before updating the GHCB and setting the GHCB MSR as well as when accessing the contents of the GHCB following the return from VMGEXIT.

In NMI context, it is recommended to have a separate GHCB for use within NMI context or that the NMI context save on entry and restore on exit the active GHCB information.

The hypervisor can communicate back to the guest in the event of an error during VMGEXIT processing. The SW_EXITINFO1 and SW_EXITINFO2 fields are used for this purpose.

SW_EXITINFO1[31:0] defines the action requested by the hypervisor:

- 0x0000
  - No action requested by the hypervisor.
- 0x0001
  - The hypervisor has requested an exception be issued. The SW_EXITINFO2 field contains the Event Injection (EVENTINJ) value as documented in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.20. The currently supported exceptions that can be requested are:
    - #GP
    - #UD

4.1.1 Standard VMGExit

- Before issuing the VMGEXIT instruction:
  - Disable interrupts and preemption

    - Copy the register contents of the faulting context documented in the "State to Hypervisor" column into the corresponding location in the GHCB.
    - Set the bits in the GHCB VALID_BITMAP field that correspond to the registers documented in the "State to Hypervisor" column.
    - Set the GHCB SW_EXITCODE, SW_EXITINFO1 and SW_EXITINFO2 to the values documented in the "State to Hypervisor" column.
    - Verify or set the GHCB MSR to the guest physical address of the GHCB being used
- Issue the VMGEXIT instruction.
- After return from the VMGEXIT instruction:
    - Advance the RIP over the instruction that generated the #VC
    - GHCB SW_EXITINFO1[31:0] == 0
        - Copy the contents of the GHCB registers documented in the "State from Hypervisor" into the corresponding registers to be made available to the faulting context upon completion of the #VC handler.
    - GHCB SW_EXITINFO1[31:0] == 1
        - Invoke the requested exception handling routine, providing as the error code the value contained in GHCB SW_EXITINFO2.
    - Enable preemption and interrupts

4.1.2 IOIO_PROT (0x7b)

The guest #VC handler will be required to parse and decode the instruction that caused the IOIO_PROT fault (a type of IN/OUT instruction) or it can para-virtualize the instruction to avoid the #VC. In either case, the guest will construct the SW_EXITINFO1 field as defined in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.10.2. If the instruction is a string-based operation, the guest must supply a decrypted buffer for the string operation. The RESERVED shared buffer area within the GHCB (offset 0x800) can be used for this purpose. The guest physical address of the buffer area must be set in the SW_SCRATCH field. The guest can issue multiple VMGEXIT calls to read or write all the string data.

4.1.3 MSR_PROT (0x7c)

The guest #VC handler will be required to parse and decode the instruction that caused the MSR_PROT fault to determine whether the fault is for a RDMSR or WRMSR or the guest can para-virtualize the instruction to avoid the #VC. In either case, the guest must use the appropriate entry in the NAE Event table for determining the state to supply in the GHCB.

4.1.4 VMMCALL (0x81)

Hypercalls are specific to the hypervisor under which the guest is running. It is up to the hypervisor to provide support in the guest OS to supply the registers that are required for that specific hypercall. Should the hypervisor not provide support within the guest OS, then only those registers documented in ***Table 5. List of Supported Non-Automatic Events***, will be provided.

4.1.5 #NPF/MMIO Access

To properly determine an MMIO access, MMIO ranges must have a reserved bit set in the nested page tables such that an #NPF will be generated with the page fault error code RSV bit set to 1. This type of #NPF will cause the #VC handler to execute. This can be accomplished by setting bits 51:n in the nested page table entry, where n is equal to the physical address size (CPUID Fn_8000_0008_EAX[7:0]) minus the reduction in physical address size when memory encryption is enabled (CPUID Fn_8000_001F_EBX[11:6]).

The guest will be required to parse and decode the instruction that caused the #NPF fault or the guest can para-virtualize the MMIO access. If either the destination, for an MMIO read, or the source, for an MMIO write, is a memory location, the guest will need to use either the #NPF/MMIO_READ or #NPF/MMIO_WRITE NAE events. Based on the instruction, the guest will construct the SW_EXITCODE, SW_EXITINFO1, SW_EXITINFO2 fields. The guest must supply a decrypted buffer for the MMIO operation source/destination. The RESERVED shared buffer area within the GHCB (offset 0x800) can be used for this purpose. The guest physical address of the buffer area must be set in the SW_SCRATCH field. The guest can issue multiple VMGEXIT calls to read or write all the data:

- MMIO Read:
    - SW_EXITCODE is set to 0x8000_0001
    - SW_EXITINFO1 is the guest physical address of the MMIO source address
    - SW_EXITINFO2 is the number of bytes to read
    - SW_SCRATCH is the guest physical address of the decrypted buffer area
        - If the number of bytes to read is greater than the size of the decrypted buffer area, the VMGEXIT can be called multiple times with SW_EXITINFO2 adjusted to match the actual amount of data to be transferred in the VMGEXIT.
    - Upon return from the VMGEXIT, the contents of the decrypted buffer area are copied to the true destination address of the MMIO instruction.
- MMIO Write:
    - SW_EXITCODE is set to 0x8000_0002
    - SW_EXITINFO1 is the guest physical address of the MMIO destination address
    - SW_EXITINFO2 is the number of bytes to write
    - SW_SCRATCH is the guest physical address of the decrypted buffer area
        - If the number of bytes to write is greater than the size of the decrypted buffer area, the VMGEXIT can be called multiple times with SW_EXITINFO2 adjusted to match the actual amount of data to be transferred in the VMGEXIT.
    - Before issuing the VMGEXIT, the contents of the true source address of the MMIO instruction are copied to the decrypted buffer area.

4.1.6 SNP Page State Change

The Page State Change NAE event allows for an SEV-SNP guest to request page state changes using the GHCB protocol. The GHCB SW_SCRATCH area must point to a Page State Change structure that resides in GHCB Shared Buffer area. The format of the Page State Change structure is:

```
struct page_state_change {
        struct {
                uint16  cur_entry;
                uint16  end_entry;
                uint32  reserved;
        } page_state_change_header;

        struct {                 // See Table 6. Page State Change Entry
                uint64  cur_page:12;
                uint64  gfn:40;
                uint64  operation:4;
                uint64  pagesize:1;
                uint64  reserved:7;
        } page_state_change_entry[];
};
```

The GHCB Shared Buffer can hold up to 253 Page State Change Entry requests.

Table 6. Page State Change Entry

Bit Position	Description
[63:57]	Reserved, must be zero
[56]	Page size (0 = 4K, 1 = 2MB)
[55:52]	Page operation: • 0x0001 – Page assignment, Private • 0x0002 – Page assignment, Shared • 0x0003 – PSMASH hint • 0x0004 – UNSMASH hint
[51:12]	Guest physical frame number (GFN) to perform page operation against
[11:0]	Current page: **Input:** current page is the offset, in 4K increments, on which to begin the page state change operation. For a page size of 4K, this must be 0. **Output:** current page is the offset, in 4K increments, that have been successfully processed. For a page size of 4K, a value of 1 indicates the page has been successfully processed. For a page size of 2M, a value of 512 indicates all the pages have been successfully processed.

A page state change is performed by the hypervisor based on the operation requested. The page state change will be against the system physical address (SPA) that is used to back the guest physical address (GPA) calculated from the supplied guest physical frame number (GFN) for the requested page size (GPA = GFN << 12).

- 0x0001 – Page assignment, Private
  The hypervisor is requested to update the RMP table to make the GPA private (guest owned). For a 2MB page size, the provided GFN must represent a 2MB aligned GPA.

- 0x0002 – Page assignment, Shared
  The hypervisor is requested to update the RMP table to make the GPA shared (hypervisor owned). For a 2MB page size, the provided GFN must represent a 2MB aligned GPA.

- 0x0003 – PSMASH hint
  The guest is supplying a hint to the hypervisor to update the RMP table to create 512 4K pages from a single 2M page for the GPA. The provided GFN must represent a 2MB aligned GPA. The hypervisor is not required to perform the PSMASH.

- 0x0004 – UNSMASH hint
  The guest is supplying a hint to the hypervisor to update the RMP table to combine 512 4K pages into a single 2M page for the GPA. The provided GFN must represent a 2MB aligned GPA. The hypervisor is not required to perform the UNSMASH.

The page_state_change_header.cur_entry is the entry at which the page state change request is to begin processing. The page_state_change_header.end_entry is the last entry to process. The hypervisor should ensure that cur_entry and end_entry represent values within the limits of the GHCB Shared Buffer.

Checking for errors is a bit different than the standard VMGEXIT method. A value of zero in SW_EXITINFO1 does not guarantee that all operations have all completed or completed successfully. As the page state changes are performed against a Page State Change Entry, the Page State Change structure in shared memory must be updated. For each successful update to the RMP table for a Page State Change Entry, page_state_change_header.cur_entry must be incremented. Additionally, when the page size is 2MB, the page_state_change_entry.cur_page must be incremented for each successful 4K page processed. This allows for the hypervisor to resume the guest at any point during the page state change request and allows the guest to resume the page state change request from the point where it was interrupted. If all entries have been successfully processed, page_state_change_header.cur_entry will be greater than page_state_change_header.end_entry.

If page_state_change_header.cur_entry is not greater than page_state_change_header.end_entry, SW_EXITINFO2 will contain a reason code. The reason codes are defined as follows:

- SW_EXITINFO2 == 0x00000000
  The page state change request was interrupted, retry the request.

- SW_EXITINFO2[63:32] == 0x00000001
  - SW_EXITINFO2[31:0] == 0x00000001
    The page_state_change_header structure is not valid
  - SW_EXITINFO2[31:0] == 0x00000002
    The page_state_change_entry structure, identified by page_state_change_header.cur_entry, is not valid.

- SW_EXITINFO2[63:32] == 0x00000002
  - SW_EXITINFO2[31:0] contains the SNP_PAGE_UNSMASH firmware command error code associated with the failing request as identified by page_state_change_header.cur_entry.

- SW_EXITINFO2[63:32] == 0x00000003
  This error code can be used to help debug guest page state tracking.
  - SW_EXITINFO2[31:0] == 0x00000001
    The page_state_change_entry, identified by page_state_change_header.cur_entry, requested a change to the page state, but the page was already in the requested state. The hypervisor can ignore this condition and not return an error for this page state change request.
  - SW_EXITINFO2[31:0] == 0x00000002
    The page_state_change_entry, identified by page_state_change_header.cur_entry, requested a change to the page state, but a 4KB page and 2MB page RMP overlap was detected. The hypervisor can attempt to correct this condition and not return an error for this page state change request.

- SW_EXITINFO2[63:32] == 0x00000100
  The hypervisor encountered some other error situation and was not able to complete the request identified by page_state_change_header.cur_entry. It is left to the guest to decide how to proceed in this situation.

4.1.7 SNP Guest Request

The SNP Guest Request NAE event allows for an SEV-SNP guest to make requests to the SEV-SNP firmware through the hypervisor using the SNP_GUEST_REQUEST API (see *SEV Secure Nested Paging Firmware ABI Specification* for a list of possible requests).

The Guest Request NAE event requires two unique pages, one page for the request and one page for the response. Both pages must be assigned to the hypervisor (shared). The guest must supply the guest physical address of the pages (i.e. page aligned) as input.

The hypervisor must translate the guest physical address (GPA) of each page into a system physical address (SPA). The SPA is used to verify that the request and response pages are assigned to the hypervisor.

Before invoking the SNP_GUEST_REQUEST API, the hypervisor must assign the response page to the firmware using the RMPUPDATE instruction and the SPA of the response page. After the firmware has completed processing the request, the hypervisor must reclaim the response page by invoking the SNP_PAGE_RECLAIM API and then assign the page to the hypervisor using the RMPUPDATE instruction and the SPA of the response page.

The SNP_GUEST_REQUEST API updates a sequence number, MSG_SEQNO, when successfully processing the request. It is recommended that the hypervisor validate the guest physical address of the response page before invoking the SNP_GUEST_REQUEST API so that the sequence numbers do not get out of sync for the guest, possibly resulting in all successive requests failing.

The hypervisor must save the SNP_GUEST_REQUEST return code in the lower 32-bits of the SW_EXITINFO2 field before completing the Guest Request NAE event.

It is not expected that a guest would issue many Guest Request NAE events. However, access to the SNP firmware is a sequential and synchronous operation. To avoid the possibility of a guest creating a denial of service attack against the SNP firmware, it is recommended that some form of rate limiting be implemented should it be detected that a high number of Guest Request NAE events are being issued.

4.1.8 SNP Extended Guest Request

The SNP Extended Guest Request NAE event is very similar to the SNP Guest Request NAE event. The difference is related to the additional data that can be returned based on the guest request. Any SNP Guest Request that does not support returning additional data must execute as if invoked as an SNP Guest Request.

The details associated with the SNP Guest Request also apply to the SNP Extended Guest Request. This section documents the additional requirements associated with the data page(s).

4.1.8.1 MSG_REPORT_REQ

The SNP Extended Guest Request uses the data page(s) to provide certificates back to the guest. The returned certificates are identified by GUID. This specification defines the following certificate GUIDs:

- Versioned Chip Endorsement Key (VCEK) certificate
  - 63da758d-e664-4564-adc5-f4b93be8accd
- AMD SEV Signing Key (ASK) certificate
  - 4ab7b379-bbac-4fe4-a02f-05aef327c782
- AMD Root Signing Key (ARK) certificate
  - c0b406a4-a803-4952-9743-3fb6014cd0ae

The certificate page(s) must be guest contiguous pages if supplying more than one page and all certificate pages must be assigned to the hypervisor (shared).

Returned certificates are identified in a table starting at offset 0x0 of the DATA GPA. Each entry consists of a 16-byte GUID, a 4-byte offset from DATA GPA to where the certificate data starts and a 4-byte length representing the length of the certificate data. The table is terminated with an entry containing all zeroes for the GUID, offset and length.

```
struct cert_table {
        struct {
                unsigned char guid[16];
                uint32  offset;
                uint32  length;
        } cert_table_entry[];
};
```

The hypervisor must validate that the guest has supplied enough pages to hold the certificates that will be returned before performing the SNP guest request. If there are not enough guest pages to hold the certificate table and certificate data, the hypervisor will return the required number of pages needed to hold the certificate table and certificate data in the RBX register and set the SW_EXITINFO2 field to 0x0000000100000000.

This usage allows the attestation report and the certificates required to verify the report to be returned at the same time. The hypervisor is not required to, or limited to, returning the certificates defined in this specification.

It is beyond the scope of this document as to how the hypervisor obtains the certificates to be returned or how the hypervisor determines the certificates to be returned.

4.1.9 SNP AP Creation

The AP Creation NAE event allows for an SEV-SNP guest to cause the creation/destruction of, or a change to, the register state of an AP, which can provide an alternate method of booting an AP under SEV-SNP (see section ***4.3.2***). An AP Creation request can:

- Create or Add
  Update, or create, a vCPU for the specified APIC ID. The hypervisor should validate that the supplied APIC ID is valid for the SEV-SNP guest.

Before invoking the hypervisor, the guest must issue an RMPADJUST instruction against the target page to indicate it will be used as a VMSA page. There are two forms of Create/Add:

- A VMSA page is provided that is not to be used/installed until an INIT-SIPI sequence is received for the target vCPU.
- A VMSA page is provided that is to be used/installed immediately for the target vCPU (the hypervisor should make the vCPU immediately runnable).

- Destroy or Remove
  Update, or destroy, a vCPU with the specified APIC ID. The hypervisor should validate that the supplied APIC ID is valid for the SEV-SNP guest. After removing the VMSA for the specified vCPU, the vCPU cannot be run until a VMSA is (re)assigned to the vCPU.

When assigning the VMSA address to the vCPU, the hypervisor must use the VMSA page's system physical address (SPA) that is used to back the specified VMSA guest physical address (GPA). Should the hypervisor encounter an error during the request, the vCPU cannot be run until a VMSA is successfully (re)assigned to the vCPU.

The guest is responsible for tracking what pages it creates and uses as a VMSA page.

It is expected that the SEV_FEATURES associated with the VMSA for the AP use the same interrupt injection mechanism as the BSP. The hypervisor can fail the SNP AP Creation request if they do not match.

4.1.10 #HV Doorbell Page

The #HV doorbell page NAE event allows for an SEV-SNP guest to register a doorbell page for use with the hypervisor injection exception (#HV). The doorbell page allows the hypervisor to notify the guest of pending events when the Restricted Injection feature is enabled.

The #HV doorbell page NAE event consists of the following actions:

- GET_PREFERRED
  A guest may use this action to request the hypervisor preferred guest physical address to use for the doorbell page for the vCPU issuing the request. The hypervisor must respond with a page-aligned GPA or 0xffffffffffffffff (the hypervisor does not have a preferred GPA) in the SW_EXITINFO2 field. If the hypervisor responds with a GPA, it must guarantee that the value cannot otherwise be mapped by the guest. The hypervisor must update the RMP table to make the page a hypervisor owned (shared) page. The guest should verify that the new GPA is outside of its known memory range.

- SET
  A guest must use this action to set the doorbell page to be used for the vCPU issuing the request when the Restricted Injection feature is enabled. The guest must supply a page-aligned

GPA in the SW_EXITINFO2 field. The value can be the hypervisor preferred value, or a value determined by the guest. If the value is not the hypervisor preferred value, the guest must make the page a hypervisor owned (shared) page. If the hypervisor accepts the GPA, then it must respond with the input GPA in the SW_EXITINFO2 field. If the hypervisor does not accept the GPA, it should follow the standard VMGEXIT error processing protocol to signal a #GP exception.

- QUERY
  A guest may use this action to determine if a doorbell page has already been set for the vCPU issuing the request. The hypervisor must respond with the previously set doorbell page GPA in SW_EXITINFO2. If a doorbell page was not previously set, the hypervisor must respond with 0 in SW_EXITINFO2.

- CLEAR
  A guest may use this action to clear the doorbell page that is currently assigned to the vCPU. The hypervisor will no longer attempt to deliver events using the hypervisor injection exception (#HV) to the vCPU. If the page will no longer be used as a doorbell page, the guest should make the page a guest owned (private) page.

Refer to section ***5*** for the specification associated with #HV doorbell page layout and restricted injection support.

4.1.11 #HV IPI

The #HV Send IPI NAE event allows for an SEV-SNP guest to send an IPI to other vCPUs in the guest when the Restricted Injection feature is enabled. The SW_EXITINFO1 field contains the information necessary to perform the IPI. The format of the SW_EXITINFO1 field is the same as the x2APIC Interrupt Command Register, allowing hypervisors to possibly re-use existing x2APIC IPI emulation support.

Interrupts are delivered as specified in section ***5***.

If the Restricted Injection feature is not enabled, the hypervisor may request standard VMGEXIT error processing protocol to signal a #GP exception.

4.1.12 #HV Timer

The #HV Timer NAE event allows for an SEV-SNP guest to request timer support from the hypervisor when the Restricted Injection feature is enabled. The hypervisor must provide emulated APIC timer functionality through this NAE event. The APIC timer structure and rules are used for setting and getting timer settings. Specifically, the APIC timer registers (Timer LVT, Divide Configuration, Timer Initial Count and Timer Current Count) and the APIC timer rules for starting and stopping the timer, allowing hypervisors to possibly re-use existing APIC timer emulation support.

The #HV Timer NAE event consists of the following actions:

- SET
  A guest may use this action to set the various #HV Timer values. The SW_EXITINFO2 field contains the timer register MASK that identifies the virtual APIC timer registers to set and the registers containing the values to be used:
  - MASK[0] – RAX: Timer LVT
  - MASK[1] – RBX: Divide Configuration
  - MASK[2] – RCX: Timer Initial Count

  If multiple bits are set in MASK, the order of assignment is the Timer LVT, Divide Configuration then Timer Initial Count.

- GET
  A guest may use this action to get the various #HV Timer values. The SW_EXITINO2 field contains the timer register MASK that identifies the virtual APIC timer register to get and the registers to return the values in.
  - MASK[0] – RAX: Timer LVT
  - MASK[1] – RBX: Divide Configuration
  - MASK[2] – RCX: Timer Initial Count
  - MASK[3] – RDX: Timer Current Count

Interrupts are delivered as defined in section ***5***.

If the Restricted Injection feature is not enabled, the hypervisor may request standard VMGEXIT error processing protocol to signal a #GP exception.

4.1.13 Termination Request

The termination request NAE event allows for the guest to request termination. The guest should expect the hypervisor to comply with the request for termination. As a safeguard, it is recommended that the guest incorporate a HLT loop or SHUTDOWN following the VMGEXIT.

The termination request will follow the same general format as the MSR protocol termination request with the added ability to provide 64 bits of termination information in SW_EXITINFO2. SW_EXITINFO1 contains the termination reason code where SW_EXITINFO1[3:0] specifies the reason code set and SW_EXITINFO1[11:4] contains the reason code from that reason code set.

The reason code set is meant to provide hypervisors with their own termination reason codes. This document defines and owns reason code set 0x0 and the following reason codes (SW_EXITINFO1[11:4]):

- 0x00 – General termination request
- 0x01 – SEV-ES / GHCB Protocol range is not supported
- 0x02 – SEV-SNP feature(s) not supported

  - SW_EXITINFO2 contains a mask of the unsupported features that are specified in the SEV_FEATURES field of the VMSA (read via the SEV_STATUS MSR)

4.1.14 Unsupported Non-Automatic Exits

Should the #VC handler be invoked for a NAE that is not part of the negotiated protocol version, it should perform a VMGEXIT using the "Unsupported Event" exit code.

4.2 Guest Identification of SEV-ES Support

A guest must be able to determine that it is running as an SEV-ES guest. In order to accomplish this, the hypervisor must provide additional CPUID properties to an SEV-ES guest. These properties allow the SEV-ES guest to determine that it is safe to issue the required CPUID and RDMSR instructions, as well as provide required information. The hypervisor must be sure that the following CPUID information is set:

- CPUID leaf 0x0000_0001:
  - ECX[31] must be set to indicate running under a hypervisor
- CPUID leaf 0x8000_001f:
  - EAX[1] must be set to indicate SEV support
  - EBX[5:0] must be the encryption bit position as discovered by the hypervisor
  - EBX[11:6] must be the reduction in physical address space bits for the guest

4.3 SMP Booting

4.3.1 SEV-ES and SEV-SNP

SMP booting under SEV-ES presents new challenges. Traditionally, the INIT-SIPI-SIPI sequence is used to boot an AP. Under virtualization, the SIPI request results in the hypervisor setting the vCPU CS segment register and IP register. The challenge here is that the hypervisor is not allowed to set the vCPU registers once they have been measured and encrypted, which occurs before the guest is started. A new way of booting an AP must be performed. The very first time an AP is started, it must use the register values that were initially set and measured when LAUNCH_UPDATE_VMSA was invoked.

SMP booting under SEV-SNP when the hypervisor does not support the SEV-SNP AP Creation NAE event has the same challenges. Once the registers have been set and measured by SNP_LAUNCH_UPDATE, they cannot be updated. Note, that when the Restricted Injection feature is active in an SEV-SNP guest, the AP Reset Hold NAE event is not available because an APIC is not available, removing the ability of the guest to issue an INIT-SIPI-SIPI sequence. Restricted Injection requires the use of the SEV-SNP AP Creation NAE event (see section ***4.3.2***).

The following are examples of setting the initial CS segment register and IP register for the APs first boot:

- Using the standard reset vector location:
    - Update the code mapped at the reset vector to check a memory location. This memory location, if non-zero, will contain the target address (SIPI vector) for the CPU that is booting.
        1. On initial BSP boot, the value will be zero so normal BSP initialization will be performed.
        2. When the BSP attempts to start an AP, it will place the AP target address into the memory location. The AP will see a non-zero value and jump to that location.
- Using a supplied reset vector location:
    - Provide a pre-determined location to the hypervisor as the initial CS segment register value and IP register value.
        1. For example, the UEFI firmware used to initialize the guest can have a compiled-in location consisting of a CS segment register value and an IP register value that can be discovered by the hypervisor prior to guest execution. These values can be used as the initial values for the guest APs.
        2. When the BSP attempts to start an AP, it will place code into this initial location to direct the AP to the desired target address.

The hypervisor is then required to do the following:

- For the first reset of the AP, the following is required:
    - The hypervisor must not update any register values and, instead, run the vCPU with the initial register values.
- For subsequent resets of the AP, the following is required:
    - When a guest AP reaches its HLT loop (or similar method for parking the AP), it instead can either:
        1. Issue an AP Reset Hold NAE event.
            - This requires the AP to be in PAE or long mode to write decrypted values to the GHCB. The AP does not have to remain in PAE or long mode once the GHCB has been updated.
        2. Issue the AP Reset Hold Request MSR Protocol
    - The hypervisor treats treats either request like the guest issued a HLT instruction and marks the vCPU as halted.
    - When the hypervisor receives a SIPI request for the vCPU, it will not update any register values and, instead, it will either complete the AP Reset Hold NAE event or complete the AP Reset Hold MSR protocol
    - Mark the vCPU as active, allowing the VMGEXIT to complete.
    - Upon return from the VMGEXIT, the AP must transition from its current execution mode into real mode and begin executing at the reset vector supplied in the SIPI request.

1. The AP should verify that the SW_EXITINFO2 field or GHCBData[63:12] field is non-zero
2. The following registers must be set to the Initial Processor State after INIT (see *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Table 14-1):
    - RAX, RBX, RCX, RDX, RSI, RDI, RBP, R8 – R15, RFLAGS
3. The remaining registers are not required to be set to the Initial Processor State after INIT.

4.3.1.1 vCPU Parking

Another challenge that arises is transferring control from one environment to the next, for example from UEFI to an OS. Using the UEFI to OS as an example, before control is handed to the OS, UEFI will park all APs using a HLT loop or similar. This code will be in reserved memory and be running in 32-bit protected mode with paging disabled. This allows the AP HLT loop to execute should a signal bring the AP out of the HLT instruction. However, instead of issuing a HLT instruction, the AP will issue a VMGEXIT with SW_EXITCODE of 0x8000_0004 (this implies that the GHCB was updated prior to leaving 64-bit long mode).

When the OS attempts to boot the AP, the code that will execute will be that of UEFI. At this point, the AP needs to have been told by the OS where to execute. To this end, UEFI needs to supply an AP jump table to the OS. The OS will use this memory to set the address of the AP reset vector:

- Upon return from the VMGEXIT, the AP must transition from its current execution mode into real mode and begin executing at the reset vector supplied by the OS in the AP jump table. The four-byte value from the AP jump table will be in the first 4-bytes of the page and match the following format:

  ```
  struct  Ap_Reset_Address {
          uint16  reset_ip;
          uint16  reset_cs;
  };
  ```

  For example, to begin executing at physical address 0x9f000, the value 0x0000 would be stored at offset 0x00 of the AP jump table and the value 0x9f00 would be store at offset 0x02 of the AP jump table. The UEFI code could push RFLAGS on to the stack, followed by the CS value of 0x9f00 and finally the RIP value of 0x0000 and then issue an IRET to begin executing at 0x9f000. An alternative is to use a far jump to load the new CS / RIP value.
- If the same reset vector is used for all AP's there is no need for serialization of the AP jump table entry. However, if different values are used for different AP's or different situations, then the use of the AP reset address field must be serialized.

The AP jump table must be communicated by UEFI to the OS. The requirements are different, depending on the type of guest:

- SEV-ES guest:
  UEFI must use the AP Jump Table SET software NAE Event to tell the hypervisor to set/save the AP jump table guest physical address.

  The OS must use the AP Jump Table GET software NAE Event to retrieve the location of the AP jump table when starting an AP.
- SEV-SNP guest
  UEFI must supply the AP jump table guest physical address in the SEV-SNP Secrets Page (see section *2.7*).

  The OS must use the AP jump table guest physical address as set in the SEV-SNP Secrets Page when starting an AP.

The AP jump table must be 4K in size, in encrypted memory and it must be 4K (page) aligned. There can only be one AP jump table and it should reside in memory that has been marked as reserved by UEFI.

4.3.1.2 vCPU Hotplug

Because of the requirements to measure and encrypt the VM register state before launching the guest, vCPU hotplug cannot be supported at this time.

4.3.2 SEV-SNP (SNP AP Creation Feature)

Using VMGEXIT SW_EXITCODE 0x8000_0013, an SEV-SNP guest can create or update the vCPU state of an AP, which may allow for a simpler and more secure method of booting an AP.

An SEV-SNP guest must use the RMPADJUST instruction to mark a page as a VM Save Area (VMSA) page. This page can then be used to initialize or change the vCPU state of an AP using VMGEXIT SW_EXITCODE 0x8000_0013. Using this method, a guest no longer is required to issue the AP Reset Hold NAE event.

There are two forms of the AP Create NAE event for creating or adding a VMSA page that can be performed depending upon the value supplied in SW_EXITINFO1:

- 0
  - The VMSA page is not to be used until an INIT request for the target AP is received (this request is not compatible with the Restricted Injection feature).
- 1
  - The VMSA page is to be used immediately (makes the vCPU runnable) for the target AP.

4.3.2.1 vCPU Parking

No special requirements are needed for parking a vCPU. The AP state can be set via SW_EXITCODE 0x8000_0013, allowing the guest to control bringing an AP out of the parked state. Using this method, a guest no longer is required to save the AP Jump Table address in the SEV-SNP Secrets Page.

4.3.2.2 vCPU Hotplug

Because of the ability for a guest to supply VMSA pages to the hypervisor, the AP creation NAE event allows for vCPU hotplug to be supported.

4.4 Non-maskable Interrupts

When injecting an NMI, the hypervisor must not intercept IRET, but must intercept #DB. The hypervisor must use the "NMI Complete" message from the guest as the indicator of when another NMI can be injected. Intercepting #DB (which a hypervisor typically already does today) provides the guest with flexibility in determining when to send the "NMI Complete" message.

The benefit of this method is that the guest processing does not need to be documented in the GHCB specification, just the requirement that the guest only issue the "NMI Complete" message when it can safely handle another NMI. This allows a guest OS to do what is easiest/best for it.

Here is one example of how the guest OS could do this:

- Use the #DB exception
    - NMI handler sets a per-CPU variable to indicate in NMI
    - Just before the actual NMI IRET, the TF flag is set:
        - PUSHF, OR flags on stack to set TF, POPF
        - The IRET must be the next instruction after the POPF
    - Execute IRET
        - #VC is triggered with an error code for a #DB intercept (0x41)
    - #VC handler is invoked and checks for NMI scenario:
        - Checks for error code of #DB intercept (0x41)
        - Checks per-CPU variable to ensure that an NMI was running
        - Clears per-CPU variable
        - Issues "NMI Complete" message using VMGEXIT
        - Exits the #VC handler

4.5 Debug Register Support

Currently, hardware debug traps aren’t supported for an SEV-ES guest. The hypervisor must set the intercept for both read and write of the debug control register (DR7). With the intercepts in place, the #VC handler will be invoked when the guest accesses DR7. For a write to DR7, the #VC handler should perform Standard VMGExit processing. The #VC handler must not update the actual DR7 register, but rather it should cache the DR7 value being written. For a read of DR7, the #VC handler should return the cached value of the DR7 register.

4.6 System Management Mode (SMM)

SMM will not be supported in this version of the specification.

4.7 Nested Virtualization

Nested virtualization is not supported under SEV-ES and SEV-SNP.

5 SNP Restricted Injection

5.1 Hypervisor Doorbells in SNP guests

Safe isolation between an SNP-protected guest and its host environment requires restrictions on the type of exception and interrupt dispatch that can be performed in the guest. Isolated guests are expected to run with the SNP RestrictInjection feature active, limiting the host to ringing a doorbell with a #HV exception. The majority of information communicated by the host is specific to the virtualization architecture (e.g. Virtio or VMBus messages) and will be delivered in a manner that is understood by the specific drivers running within the guest. However, dispatch of these messages is typically driven by interrupt delivery, which is core to the operating system rather than to the drivers themselves. Consequently, it is advantageous to define a common format to describe notifications whose dispatch is integral to OS functionality. At the same time, it is advantageous to avoid a common format for information that is normally only interpreted by drivers, so these drivers have the flexibility to optimize their communication paths as they see fit. This document defines the set of information that provides the common core of host notification, and the format by which it is communicated.

5.2 Essential Data

Virtual machines typically rely on interrupt delivery through an architectural APIC, even for synthetic messages delivered by the host. The common #HV doorbell data is structured to align with APIC behavior, so that existing guest interrupt dispatch and management logic can be used.

Two fields are defined in the #HV doorbell page, a pending event field (“PendingEvent”) and an EOI assist (“NoEoiRequired)”. PendingEvent is a 16-bit field in bytes 0..1 of the page, and NoEoiRequired is byte 2 of the page. Bytes 3..63 are reserved to accommodate future expansion of required functionality. The structure of the remainder of the page is not specified and should be used by the virtualization stack as required for its own message formats.

Because the doorbell page is shared with the host, the physical page that backs it must be a host-owned page (not assigned with RMPUPDATE). All guest accesses must be made with C=0.

PendingEvent is defined as follows:

- PendingEvent[15] (“NoFurtherSignal”) - indicates that the host will not signal #HV due to another non-maskable event until this bit is cleared by the guest.
- PendingEvent[14:10] – reserved for future use
- PendingEvent[[9] - indicates that the host is presenting a virtual #MC to the guest. This is an example of a non-maskable event.
- PendingEvent[8] - indicates that the host is presenting an NMI to the guest. This is an example of a non-maskable event.

- PendingEvent[7:0] - an 8-bit interrupt vector number. When this number is non-zero, it indicates that the host is presenting an interrupt on the specified vector. Once the guest acknowledges receipt of the vector (as described below), the interrupt is placed in service in the host-emulated APIC. When the guest has completed service of the interrupt, it must issue an end-of-interrupt cycle with the host-emulated APIC.

It is desirable to optimize end-of-interrupt indications without requiring exiting the VM. When it is possible to perform an EOI without exiting the guest, NoEoiRequired will be set to a non-zero value by the host, indicating that no explicit EOI is required. When the guest wishes to perform an EOI, it should attempt to atomically change NoEoiRequired from non-zero to zero. If successful, no further processing is necessary; if unsuccessful, the guest must request an explicit EOI by performing VMGEXIT to request a WRMSR to the X2APIC EOI MSR.

Specific treatment of the PendingEvent and NoEoiRequired fields is amplified below.

5.3 Interrupt Shadows

Typical interrupt delivery involves a careful use of interrupt shadows leading up to a HLT instruction to ensure that a guest can never inadvertently dispatch all pending interrupt before executing HLT. Because #HV is delivered without regard to interrupt shadows, guests lose the ability to control interaction between HLT and interrupts. Consequently, a different convention is required. When a guest is ready to enter a halt state, it should not examine pending interrupt state, nor should it enable interrupts; it should execute HLT with interrupts disabled. If the host receives a HLT intercept while the guest has any interrupts or non-maskable events pending, it should immediately re-enter the guest. When a guest resumes following a HLT instruction, it should immediately proceed to examine interrupt state as if it has just received a #HV. This convention ensures that a guest can properly suspend when no interrupts are pending while also ensuring that a guest will neither miss pending interrupts nor suspend before all interrupt processing has properly completed.

5.4 Expected Behaviors

5.4.1 Doorbell configuration

Configuration of the doorbell page should be performed through specific VMGEXIT requests communicated via the GHCB.

Configuration of the #HV doorbell page can be performed using VMGEXIT SW_EXITCODE 0x8000_0014. A brief outline of the available functions is listed below. Refer to section ***4.1.10*** for a full description of the operations.

- GET_PREFERRED
  A guest may use this action to request the hypervisor preferred guest physical address to use

for the doorbell page for the vCPU issuing the request.

- SET
  A guest must use this action to set the doorbell page to be used for the vCPU issuing the request when the Restricted Injection feature is enabled.

- QUERY
  A guest may use this action to determine if a doorbell page has already been set for the vCPU issuing the request.

Whenever the host accepts a mapping of a doorbell page, it is free to execute RMPUPDATE to clear the RMP assignment of the page mapped at that guest physical address. Note that a well-behaved guest will execute PVALIDATE to relinquish access to the page prior to handing it over to the host, as is typically required to ensure security of the RMP (as documented elsewhere). Once the page is mapped, it must be accessed by the guest with C=0 in the guest PTE. If the host is asked to remap the doorbell page at a new location, it is not expected to execute RMPUPDATE to assign a guest page at the old guest physical address; if the guest wants to make use of that guest physical address again, it must use a page assignment protocol (documented elsewhere) to request an assigned page at that location before attempting to use it again with C=1.

5.4.2 Host Behavior

The host is expected to emulate APIC behavior faithfully, at least with respect to interrupt presentation (ready, acknowledge, in-service, and EOI semantics). The manner in which individual interrupt sources are configured is presumed to be specific to the virtualization stack, but the lifecycle of an interrupt vector through the emulated APIC should adhere to the APIC architecture.

Whenever the host determines that an interrupt source is ready to be delivered (i.e. ready, not in service, and at a higher priority than the highest priority interrupt that is presently in service), it must present it to the guest by writing to the vector number to vector field of PendingEvent in the doorbell page. The host does not have the ability to determine the logical TPR in use by the guest, nor the guest value of EFLAGS.IF, so it should simply present the interrupt by writing the APIC vector number. This should occur at a time that the guest vCPU is not running. If PendingEvent.Vector was previously zero AND if PendingEvent.NoFurtherSIgnal was previously zero, the host should schedule delivery of a #HV to indicate to the guest that a new interrupt is available. If PendingEvent.Vector was previously non-zero, because the guest has not yet chosen to acknowledge the interrupt, the previous vector number can be overwritten by the new vector number without sending another #HV (regardless of the value of PendingEvent.NoFurtherSignal), since the guest should already have been informed that a vector was pending. If the host is required to send #HV, it must also set PendingEvent.NoFurtherSignal to ensure that no additional #HV is sent as a result of a non-maskable event (as explained below) until the guest indicates that it is ready to receive it. Note that the host is always expected to present the highest priority ready interrupt, so if a non-zero vector number is overwritten, it should always be overwritten with a higher priority vector. The host must additionally record internally the last vector that was presented.

Whenever the host presents an interrupt, and there are no other interrupts that are ready to be presented (i.e. all lower priority interrupts are already in service), the host should set NoEoiRequired to indicate that no explicit EOI is required. The host should additionally record internally that NoEoiRequired was set. If, on the other hand, multiple interrupts are ready, the host should clear NoEoiRequired, and should record internally that NoEoiRequired was cleared. When multiple interrupts are ready, the host must know precisely when the guest is ready to receive the second interrupt, and this requires an explicit EOI cycle.

Whenever the host wishes to present a non-maskable event such as an NMI or #MC, the host should set the appropriate bit in the PendingEvent field. The host should also set PendingEvent.NoFurtherSignal. If NoFurtherSignal was previously zero, the host should deliver #HV, while if NoFurtherSignal was previously non-zero, then the guest should already be aware that event processing is required.

Whenever the host observes that PendingEvent.Vector is zero but has internally recorded that a non-zero vector was presented, it indicates that the interrupt was acknowledged by the guest. As this is analogous to an interrupt being acknowledged at the APIC by the CPU, the host must mark the interrupt in service so that it can be cancelled by a subsequent EOI.

Whenever the host observes NoEoiRequired is zero and has internally recorded that NoEoiRequired was previously set, it should perform an EOI cycle as if the APIC EOI register had been written. This must occur after examining PendingEvent.Vector, as an EOI may be pending for the interrupt that was just acknowledged by the guest.

Whenever an EOI cycle is completed (either implicitly by clearing NoEoiRequired, or explicitly via an APIC register write), the host should set NoEoiRequired again if any interrupts remain in service and no interrupts are pending (and should additionally record internally that the flag was set). Note that the guest is not obligated to clear NoEoiRequired and may elect to perform a virtual APIC register write to complete the EOI. The host must treat both types of EOI signals identically.

Examination of PendingEvent and NoEoiRequired can be performed by the host at any time the guest vCPU is not running, but must be performed at any time the emulated APIC state is evaluated. Examination of those fields is not required if the host has not recorded that it has placed any data into them.

If a non-cooperative guest chooses to write a non-zero vector into PendingEvent.Vector when no interrupt is pending, or chooses to set PendingEvent.NoFurtherSignal without receiving a #HV signal, it may result in the loss of a #HV notification when the host is prepared to deliver an interrupt. Any bad behavior that results in the guest is a consequence of its non-cooperation, but it will not affect the integrity of the host in any way. Similarly, the host will not be affected if a non-cooperative guest chooses to modify NoEoiRequired when no EOI assist is possible.

5.4.3 Guest Behavior

Whenever a guest receives a #HV notification, it must be prepared to receive an interrupt from the emulated APIC. This can occur at any time; the guest may choose to acknowledge it immediately, or may choose to defer acknowledgement until EFLAGS.IF permits interrupt delivery.

When the guest chooses to acknowledge an interrupt, it must perform an atomic exchange to retrieve the pending event information and to zero the contents of PendingEvent. Since the guest can be interrupted on any instruction boundary, and since the host has the right to exchange one non-zero vector for another without issuing another #HV, the guest must not act on any vector number observed in the doorbell page unless it is atomically exchanged with zero. Once the vector number has been observed, the guest can process the interrupt as it sees fit, either dispatching it immediately, or deferring dispatch until such time that dispatch is safe. Once the guest has acknowledged the interrupt, it must assume that the interrupt has been acknowledged in accordance with the standard APIC architecture, i.e. no additional interrupt of equal or lower priority can be presented until an end of interrupt cycle is performed.

When the guest is ready to perform an EOI, it should perform an atomic exchange of zero with NoEoiRequired. If the previous value was non-zero, then no further action is required, since the act of clearing NoEoiRequired is sufficient to prompt an end of interrupt before delivery of the next interrupt. If the previous value was zero, then the guest must perform an APIC register write (via VMGEXIT to write the X2APIC MSR) to complete the EOI.

Even when EFLAGS.IF=0, a guest should be prepared to dispatch non-maskable events as indicated in PendingEvent. If a guest chooses to dispatch only non-maskable events without clearing the pending vector, it must perform an atomic exchange to clear the non-maskable event flags as well as NoFurtherSignal before proceeding to dispatch any accumulated events. Leaving PendingEvent.Vector unchanged ensures that the guest will not receive an #HV signal due only to changes in maskable interrupt state, while clearing NoFurtherSignal ensures that the guest will receive another #HV if an additional non-maskable event is delivered.

Since a non-cooperative host may write random numbers into the doorbell page, the guest must validate each vector it observes to ensure that it corresponds to a legitimate, expected interrupt. A non-cooperative host may choose not to send #HV, and a non-cooperative host may similarly choose to ignore NoEoiRequired; both of these cases result in denial of service to the guest rather than any corruption of guest state. A non-cooperative host may signal #HV at any time it chooses, possibly resulting in recursive handling by the guest; however, a guest can assume that under normal operation, the host will never inject #HV if PendingEvent.NoFurtherSignal is non-zero, and any receipt of #HV at any other time is grounds for a system panic.

5.5 Pseudocode

5.5.1 Host

```
; when ready to update APIC state (e.g. evaluation of ready interrupts,
; EOI requests)

; check to see whether the guest has acknowledged a previously presented
; interrupt
IF APIC.PendingVector != 0 AND Doorbell.PendingEvent.Vector = 0
    ; acknowledge interrupt
    APIC.Ready[APIC.PendingVector] := 0
    APIC.InService[APIC.PendingVector] := 1
    APIC.PendingVector := 0
FI

; check to see whether the guest has requested an EOI without explicitly
; writing the EOI register
IF APIC.NoEoiRequired AND Doorbell.NoEoiRequired = 0 AND APIC.PendingVector = 0
    APIC.EndOfInterrupt()
    APIC.NoEoiRequired := 0
FI

; calculate ready interrupt state
HV_Required := false
IF NMI is pending
    Doorbell.PendingEvent.NMI := 1
    HV_Required := true
FI
IF #MC is pending
    Doorbell.PendingEvent.MC := 1
    HV_Required := true
FI
IF APIC.Ready is not empty
    ; an explicit EOI is required unless the interrupt being presented is the
    ; only ready interrupt
    IF APIC.NumberOfReadyInterrupts = 1
        APIC.NoEoiRequired := true
        Doorbell.NoEoiRequired := 1
    ELSE
        APIC.NoEoiRequired := false
        Doorbell.NoEoiRequired := 0
    FI

    APIC.PendingVector := next ready interrupt
    IF Doorbell.PendingEvent.Vector = 0
        ; only send #HV if there is no interrupt already pending
        HV_Required := true
    FI
    Doorbell.PendingEvent.Vector := APIC.PendingVector
FI
IF HV_Required
    IF Doorbell.PendingEvent.NoFurtherSignal = 0
        Doorbell.PendingEvent.NoFurtherSignal := 1
    ELSE
        HV_Required := FALSE
    FI
```

```
FI
IF HV_Required
    Schedule #HV
FI
```

5.5.2 Guest

```
; when the guest is ready to dispatch interrupts
IF EFLAGS.IF = 0
    NonMaskableEvents := XCHG(Doorbell.PendingEvent[15..8], 0)
    ; handle NMI or #MC as required
    ; leave handler
FI

PendingEvent := XCHG(Doorbell.PendingEvent, 0)
IF (PendingEvent.NMI OR PendingEvent.MC)
    ; handle NMI or #MC as required
FI

IF PendingEvent.Vector != 0
    ; dispatch interrupt or schedule it internally for future delivery
FI

; upon completion of the active interrupt
NoEoiRequired := XCHG(Doorbell.NoEoiRequired, 0)
IF NoEoiRequired = 0
    WRMSR(X2APIC_EOI)
FI
```