SEV-ES Guest-Hypervisor Communication Block Standardization

Publication #	**56421** Revision: **1.00**
Issue Date:	**August 2020**

Advanced Micro Devices

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
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.
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) feature provides protection of the virtual machine, or guest, register state from the hypervisor. An SEV-ES guest's register state is encrypted during world switches and cannot be directly accessed or modified by the hypervisor. SEV-ES is documented in the *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.35.

SEV-ES includes 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 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, 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 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 must correspond to 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 2 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. The GHCB fields that are not defined in the SEV-ES save state area are mapped at the end of the GCHB. This allows for SEV-ES save state area expansion in the future. 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 SEV-ES save state area that are used in this version of the document will be listed. However, should future versions need to expose new fields, they will correspond to the SEV-ES save area definition. 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.

2.1 Establishing the GHCB

The GHCB location in the guest physical address space is chosen 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.

Table 1. GHCB Address Destination

Field Name	Bit Position	Definition
GHCBInfo	11:0	• 0x000 – GHCB guest physical address (from guest) • 0x001 – SEV Information (from hypervisor) • 0x002 – Request for SEV Information (from guest) • 0x003 – Reserved • 0x004 – CPUID request (from guest) • 0x005 – CPUID response (from hypervisor) • 0x100 – Guest has requested termination
GHCBData	63:12	Value dependent upon GHCBInfo

- GHCBInfo:
  - 0x000
    - 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
    - GHCBData[63:48] specifies the maximum SEV-ES/GHCB protocol version supported
    - GHCBData[47:32] specifies the minimum SEV-ES/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 SEV-ES/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.

- 0x002
  - Written by the guest to request the hypervisor provide the SEV information (GHCBInfo = 0x001) needed to perform protocol negotiation.
- 0x004
  - 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
  - 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.

- 0x100
  - 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.

2.2 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. The following additional steps document an example for the GHCB negotiation.

- 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.
  - Let's say that the hypervisor supports only the current version (1) and that 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 be able 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 SEV-ES/GHCB protocol version, GHCBData[63:48], and minimum SEV-ES/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 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.

2.3 GHCB/VMGEXIT Example

- 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.

2.4 GHCB Layout

Table 2. GHCB Layout

Offset	Size	Contents	Notes
0x0000	0xcb	RESERVED	
0x00cb	0x01	CPL	
0x00cc	0x94	RESERVED	
0x0160	0x08	DR7	
0x0168	0x90	RESERVED	
0x01f8	0x08	RAX	
0x0200	0x100	RESERVED	
0x0300	0x08	RESERVED (RAX already available at 0x01f8)	
0x0308	0x08	RCX	
0x0310	0x08	RDX	
0x0318	0x08	RBX	
0x0320	0x70	RESERVED	
0x0390	0x08	SW_EXITCODE	Guest controlled exit code
0x0398	0x08	SW_EXITINFO1	Guest controlled exit information 1
0x03a0	0x08	SW_EXITINFO2	Guest controlled exit information 2
0x03a8	0x08	SW_SCRATCH	Guest controlled additional information
0x03b0	0x38	RESERVED	
0x03e8	0x08	XCR0	

Offset	Size	Contents	Notes
0x03f0	0x10	VALID_BITMAP	Bitmap to indicate valid qwords in the save state area starting from offset 0x000 through offset 0x3ef (126 qwords)
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.
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.
0x0ff0	0x0a	RESERVED	
0x0ffa	0x02	SEV-ES/GHCB Protocol Version	Version of the SEV-ES/GHCB communication protocol used by the guest • 0x0001 – SEV-ES/GHCB Protocol Version 1
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.

3 Guest Exits

3.1 Automatic Exits (AE)

Table 3. 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

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 SEV-ES/GHCB Protocol Version 1

This document will provide the definition for version 1 of the SEV-ES/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 version 1. 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 is the minimum state information required. Each entry supplied by the guest must set the appropriate bit in the GHCB VALID_BITMAP field. 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.

Table 4. List of Supported Non-Automatic Events

NAE Event	State to Hypervisor	State from Hypervisor	Notes
DR7 Read	SW_EXITCODE = 0x27		See Debug Register Support
DR7 Write	RAX SW_EXITCODE = 0x37 SW_EXITINFO1		See Debug Register Support

NAE Event	State to Hypervisor	State from Hypervisor	Notes
	SW_EXITINFO2 = 0		• SW_EXITINFO1 will be set as documented in *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, Section 15.8.1
RDTSC	SW_EXITCODE = 0x6e SW_EXITINFO1 = 0 SW_EXITINFO2 = 0	RAX RDX	
RDPMC	RCX SW_EXITCODE = 0x6f SW_EXITINFO1 = 0 SW_EXITINFO2 = 0	RAX RDX	
CPUID	RAX RCX XCR0 (for RAX == 0xd) SW_EXITCODE = 0x72 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0	RAX RBX RCX RDX	• XCR0 is only required to be supplied when a request for CPUID 0000_000D is made.
INVD	SW_EXITCODE = 0x76 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		
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

NAE Event	State to Hypervisor	State from Hypervisor	Notes
			• 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.
MSR_PROT (RDMSR)	RCX SW_EXITCODE = 0x7c SW_EXITINFO1 = 0 SW_EXITINFO2 = 0	RAX RDX	
MSR_PROT (WRMSR)	RAX RCX RDX SW_EXITCODE = 0x7c SW_EXITINFO1 = 1 SW_EXITINFO2 = 0		
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.
RDTSCP	SW_EXITCODE = 0x87 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0	RAX RCX RDX	
WBINVD	SW_EXITCODE = 0x89 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		
MONITOR/ MONITORX	RAX RCX RDX SW_EXITCODE = 0x8a		• RAX will contain the guest physical address of the

NAE Event	State to Hypervisor	State from Hypervisor	Notes
	SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		MONITOR/MONITORX memory range.
MWAIT/ MWAITX	RAX RCX SW_EXITCODE = 0x8b SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		
#AC			• The #VC handler should forword this exception on to the #AC handler.
#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 • SW_SCRATCH will have the DST guest physical address of shared memory
#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 • SW_SCRATCH will have the SRC guest physical address of shared memory
NMI Complete	SW_EXITCODE = 0x8000_0003 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		
AP Reset Hold	SW_EXITCODE = 0x8000_0004 SW_EXITINFO1 = 0 SW_EXITINFO2 = 0		

NAE Event	State to Hypervisor	State from Hypervisor	Notes
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)
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

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 4, 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 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

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. 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.
    - On initial BSP boot, the value will be zero so normal BSP initialization will be performed.
    - 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.
    - 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.
    - 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 issues a VMGEXIT with SW_EXITCODE of 0x8000_0004.
    - 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.
  - The hypervisor treats SW_EXITCODE 0x8000_0004 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 set the GHCB SW_EXITINFO2 field to a non-zero value and 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.
    - The AP should verify that the SW_EXITINFO2 field is non-zero

- 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
- The remaining registers are not required to be set to the Initial Processor State after INIT.

4.3.1 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. UEFI must use the AP Jump Table SET software NAE Event to tell the hypervisor to set/save the AP startup jump table guest physical address. 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. The OS must use the AP Jump Table GET software NAE Event to retrieve the location of the AP startup jump table before starting the first AP.

4.3.2 vCPU Hotplug

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

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.