LaGrande Technology

Preliminary Architecture Specification

September 2006

Document Number: 315168 002

Contents

Figures

Tables

Revision History

Revision Number	Description	Revision Date
-001	• Initial release.	May 2006
-002	• Established public document number • Edited throughout for clarity.	August 2006

§

1 LaGrande Technology Overview

Intel Technology for safer computing, code named LaGrande Technology, defines platform level enhancements that provide the building blocks for creating trusted platforms.

Whenever the word trust is used, there must be a definition of who is doing the trust and what is being trusted. This enhanced platform helps to provide the identity of the controlling environment such that those wishing to rely on the platform can make an appropriate trust decision. The enhanced platform determines the identity of the controlling environment by accurately measuring the controlling software (see Measurement and LaGrande Technology).

Another aspect of the trust decision is the ability of the platform to resist attempts to change the controlling environment. The enhanced platform will resist attempts by software processes to change the controlling environment or bypass the bounds set by the controlling environment.

What is the controlling environment for this enhanced platform? The platform is a set of extensions designed to work with Intel® Virtualization Technology for IA-32 Intel Architecture (VT-x). The *IA-32 Intel® Architecture Software Developer's Manual* provides more information on VT-x and guidelines for writing Virtual Machine Monitor software.

The extensions enhance two areas. These are:

- The launching of the Virtual Machine Monitor (VMM)
- The protection of the VMM from potential corruption

As the enhanced platform uses VT-x, Virtual Machine Extensions (VMX) provide the programming interface to manage and interface with the VMM. The enhanced platform provides additional launch and control interfaces using Safer Mode Extensions (SMX).

VMX defines processor-level support for virtual machines on IA-32 processors. VMX enables two classes of software to operate:

- **Virtual Machine Monitor (VMM).** A VMM acts as a host for virtual machines and has full control of the processor and other platform hardware. VMM presents guest software (next bullet) with an abstraction of a virtual processor and allows it to execute directly on the processor. A VMM is able to retain selective control of processor resources, physical memory, interrupt management, and I/O. This control governs access permitted by guest software.
- **Guest Software.** Each virtual machine (VM) is a guest software environment that can support a stack consisting of an operating system (OS) and software applications. A VM operates independently of other virtual machines and can rely on the same interface to processor, memory, storage, graphics, and I/O. The software stack acts as if it is running on a processor and platform with no VMM. An OS executing in a virtual machine operates with reduced privilege because the VMM retains control of processor and platform resources.

The SMX interface includes the following functions:

- Measured launch of the VMM
- Mechanisms to ensure the above measurement is protected and stored in a secure location
- Protection mechanisms that allow the VMM to control attempts to modify the VMM
- Protection mechanisms that allow guest software to assure that no other guest software can modify the guest
- Detection of changes to the VMM by using measured identification of the VMM

1.1 Measurement and LaGrande Technology

LaGrande Technology uses the term *measurement* frequently. Measuring software involves processing the executable such that the result is (a) unique and (b) indicates changes in the executable. A cryptographic hash algorithm meets these needs.

A cryptographic hash algorithm is sensitive to even one-bit changes to the measured entity. A cryptographic hash algorithm also produces outputs that are sufficiently large so the potential of collisions (where two hash values are the same) is small. When the term measurement is used in this specification, the meaning is that the measuring process takes a cryptographic hash of the measured entity.

1.2 LaGrande Technology Partitioned Environment

VMMs can support a variety of guests. The amount of protection a VMM offers to a guest depends on the policy provided by the VMM. For example, a guest can require tight controls on the entire software stack inside of the guest, only require protection of the guest kernel, or require no additional protections. Describing all policy variations that VMMs can support is beyond the scope of this document.

This specification considers two types of protection for the guest software. The first guest, or partition, is defined as a standard partition. The standard partition does not require stringent protections and potentially executes a legacy operating system. Legacy in this context means an operating system that is not aware of VT-x or Intel security enhancements. The second partition is a protected partition. The protected partition cooperates with VT-x and the enhanced platform and provides a set of functions. One possibility for a protected partition would be a partition that executes an Internet firewall.

The controlling environment is provided by a VMM. A VMM launched using the SMX instructions is known as a Measured Virtual Machine Monitor (MVMM). MVMMs provide different launch mechanisms and increased protection (offering protection from possible software corruption).

1.3 Late Launch

A central enhanced platform objective is to provide a measurement of the VMM.

One measurement is made when the platform boots, using techniques defined by the Trusted Computing Group (TCG). The TCG defines a Root of Trust for Measurement (RTM) that executes on each platform reset; it creates a chain of trust from reset to the measured environment. As the measurement always executes at platform reset, the TCG defines this type of RTM as a Static RTM (SRTM).

Maintaining a chain of trust for a length of time may be a challenging for a VMM meant for use in LaGrande Technology; this is because a VMM may operate in an environment that is constantly exposed to unknown software entities. To address this issue, the enhanced platform provides another RTM with SMX instructions. The TCG terminology for this option is Dynamic Root of Trust for Measurement (DRTM). The advantage of a DRTM (also called the 'late launch' option) is that launch can occur at any time without resorting to a platform reset. It is possible to launch a MVMM, execute for a time, terminate the MVMM, execute without virtualization, and then launch the MVMM again. One possible sequence is:

1. During the BIOS load: (a) launch an MVMM for use by the BIOS, (b) terminate the MVMM when its work is done, (c) continue with BIOS processing and hand off to an OS.
2. Then, the OS loads and launches a different MVMM.

In both instances, the platform measures each MVMM and ensures the proper storage of the MVMM measurement value.

When launching a MVMM, the environment must load two code modules into memory. One module is the MVMM. The other is known as an authenticated code (AC) module. The AC module is only in use during the load and measurement process and is chipset-specific. It is digitally signed by the chipset vendor; the launch process must successfully validate the digital signature before continuing the launch process.

1.3.1 Launch Sequence

With the AC module and MVMM in memory, the launching environment can invoke the GETSEC [SENTER] instruction provided by the SMX.

GETSEC [SENTER] broadcasts messages to the chipset and other logical processors in the platform (Intel processors supporting Hyper-Threading Technology with an HT Technology enabled chipset or processors with dual cores). In response, other logical processors perform basic cleanup, signal readiness to proceed, and wait for messages to join the environment created by the MVMM. As this sequence requires synchronization, there is an initiating logical processor (ILP) and a responding logical processor ((RLP) or processors).

After all logical processors signal their readiness to join and are in the wait state; the initiating logical processor loads, authenticates, and executes the AC module. The AC module tests for various chipset and processor configurations and ensures the platform has an acceptable configuration. It then measures and launches the MVMM.

The MVMM initialization routine completes system configuration changes (including redirecting INITs, SMIs, interrupts, etc.); it then issues a new SMX instruction that wakes up responding logical processors (RLPs) and brings them into the protected environment. At this point, all logical processors and the chipset are correctly configured. The MVMM may, at its discretion, take control of the launching environment and create a guest partition that contains the launching environment. It is then possible for the MVMM to exit, and then be launched again (without issuing a system reset).

1.4 Storing the Measurement

SMX operation during the launch provides an accurate measurement of the MVMM. After creating the measurement, the initiating logical processor needs a location to store the measurement. Requirements for the storage location are extensive; they are met by the Trusted Platform Module (TPM), defined by the TCG. An enhanced platform includes mechanisms that ensure that the measurement of the MVMM (completed during the launch process) is properly reported to the TPM.

With the MVMM measurement in the TPM, the MVMM can use the measurement value to protect sensitive information and detect potential unauthorized changes to the MVMM itself.

1.5 Controlled Take-down

Because the MVMM controls the platform, exiting the MVMM is a controlled process. The process includes: (a) shutting down all guest partitions, except for one; (b) and ensuring that memory previously used does not leak sensitive information.

The MVMM cleans up after itself and terminates the MVMM control of the environment. It turns control of the platform over to the one remaining guest partition.

1.6 SMX and VMX Usage

After the MVMM and a protected partition are launched, the MVMM operates using interfaces defined by VMX. Control transfers (VM enters, VM exits) between a protected partition and the MVMM are also governed by VMX.

A VM abort may occur while in SMX operation. This behavior is described in the *IA-32 Intel® Architecture Software Developer's Manual, Volume 3B*. Note that entering authenticated code execution mode or launching of a protected environment affects the behavior and response of the logical processors to certain external pin events.

1.7 Authenticated Code Module

To support the establishment of a protected environment, SMX enables the capability of an authenticated code execution mode. This provides the ability for a special code module, referred to as the authenticated code module (AC module), to be loaded into internal RAM (referred to as authenticated code execution area) within the processor. The AC module is first authenticated and then executed using a tamper resistant method.

Authentication is achieved through the use of a digital signature in the header of the AC module. The processor calculates a hash of the AC module and uses the result to validate the signature. Using SMX, a processor will only initialize processor state or execute the AC code module if it passes authentication. Since the authenticated code module is held within internal RAM of the processor, execution of the module can occur in isolation with respect to the contents of external memory or activities on the external processor bus.

1.8 Chipset Support

One important feature the chipset provides is the MPT table. The MPT table, under control of the VMM, allows the VMM to protect itself and the guest partitions from unauthorized device access to memory. The MPT table blocks access to a specific physical memory page and the enforcement of the block occurs for all DMA access to the protected page.

LaGrande Technology architecture also provides extensions that access certain chipset registers and TPM address space.

Chipset registers that interact with SMX include:

- **Public space registers.** Enhanced public space registers can be accessed by system software using memory read/write protocols. They are mapped to uncacheable (UC) memory type. Chipset enhanced platform public space registers are designated in this document with the convention LT.PUBLIC.REGISTER.
- **Private space registers.** When locked, enhanced platform private space registers are not accessible to system software until they are unlocked by SMX instructions. When unlocked, these registers are accessed by system software (MVMM) or an AC module using regular memory read/write protocols. SMX instructions also provide the ability to lock the private space registers. The private space registers are also mapped as UC. Chipset enhanced platform private space registers are designated with the convention LT.PRIVATE.REGISTER.

The storage spaces accessible within a TPM device are grouped by a locality attribute. The following localities are defined:

- Locality 0 : Non-trusted and Legacy TPM operation
- Locality 1 : An environment for use by the Trusted Operating System
- Locality 2 : Trusted OS
- Locality 3 : Authenticated Code Module
- Locality 4 : LaGrande Technology hardware use only

1.9 TPM Usage

LaGrande Technology makes extensive use of the Trusted Platform Module (TPM) defined by the Trusted Computing Group (TCG) in the *TCG TPM Specification, Version 1.2*. The TPM provides a repository for measurements and the mechanisms to make use of the measurements. The system makes use of the measurements to both report the current platform configuration and to provide long-term protection of sensitive information.

The TPM stores measurements in Platform Configuration Registers (PCRs). PCRs provide a storage area that allows an unlimited number of measurements in a set amount of space. They provide this feature by an inherent property of cryptographic hashes. Outside entities never write directly to a PCR register, they “extend” PCR contents. The extend operation takes the current value of the PCR, appends the new value, performs a cryptographic hash on the combined value, and the hash result is the new PCR value. One of the properties of cryptographic hashes is that they are order dependent. This means hashing A then B produces a different result from hashing B then A. This ordering property allows the PCR contents to indicate the order of measurements.

Sending measurement values from the measuring agent to the TPM is a critical platform task. Dynamic Root of Trust of Measurement (DRTM) requires specific messages to flow from the DRTM to the TPM. The LaGrande Technology DRTM is GETSEC[SENTER] and the system ensures GETSEC[SENTER] has special messages to communicate to the TPM. These special messages take advantage of TPM localities 3 and 4 to protect the messages and inform the TPM that GETSEC[SENTER] is sending the messages.

§

2 Safer Mode Extensions

Safer Mode Extensions (SMX) provide a means for system software to launch an MVMM and establish a protected environment within the platform to support trust decisions by end users.

2.1 Detecting and Enabling SMX

Software can detect support for SMX operation using CPUID instruction. If software executes CPUID with 1 in EAX, a value of 1 in bit 6 of ECX indicates support for SMX operation (GETSEC is available).

See Figure 1 and Table 1 for the definition of feature flag bits of CPUID.01H.ECX. For more information on CPUID, see Chapter 3, "Instruction Set Reference, A-M," in the *IA-32 Intel® Architecture Software Developer's Manual, Volume 2A*.

Figure 1. CPUID Extended Feature Information ECX

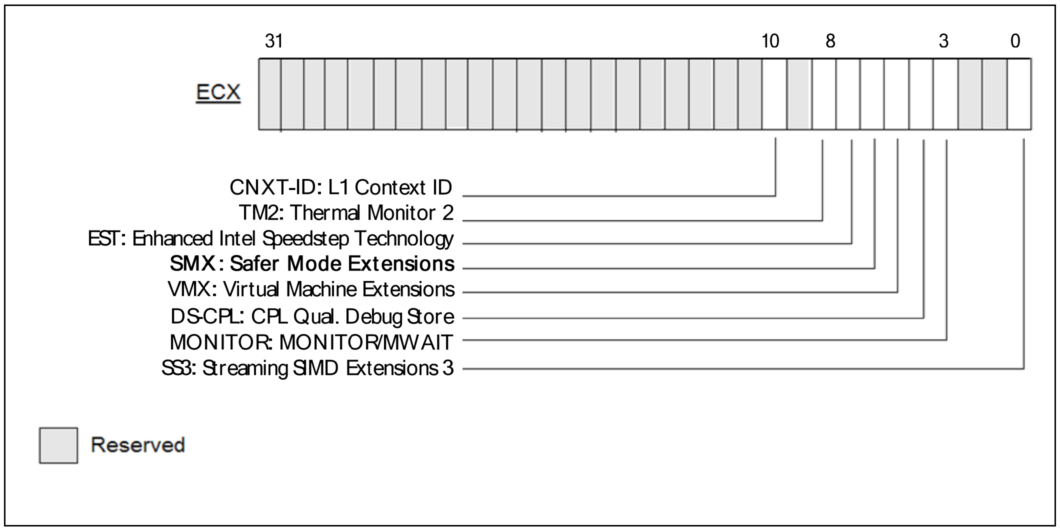

31
10
8
3
0
ECX
CNXT-ID: L1 Context ID
TM2: Thermal Monitor 2
EST: Enhanced Intel Speedstep Technology
SMX: Safer Mode Extensions
VMX: Virtual Machine Extensions
DS-CPL: CPL Qual. Debug Store
MONITOR: MONITOR/MWAIT
SS3: Streaming SIMD Extensions 3
Reserved

Table 1. CPUID Extended Feature Information in ECX

Bit #	Mnemonic	Description
0	SSE3	Streaming SIMD Extensions 3. A value of 1 indicates the processor supports Streaming SIMD Extensions 3.
3	MONITOR	MONITOR/MWAIT. A value of 1 indicates the processor supports this feature.
4	DS-CPL	CPL Qualified Debug Store. A value of 1 indicates the processor supports the extensions to the Debug Store feature to allow for branch message storage qualified by CPL.
5	VMX	Virtual Machine Extensions. A value of 1 indicates the processor supports VMX.
6	SMX	Safer Mode Extensions. A value of 1 indicates the processor supports SMX.
7	EST	Enhanced Intel Speedstep Technology. A value of 1 indicates that the processor supports this technology.
8	TM2	Thermal Monitor 2. A value of 1 indicates whether the processor supports this technology.
10	CNXT-ID	L1 Context ID. A value of 1 indicates the L1 data cache mode can be set to either adaptive mode or shared mode. A value of 0 indicates this feature is not supported. See definition of the IA32_MISC_ENABLES MSR Bit 24 (L1 Data Cache Context Mode) for details.

2.1.1 SMX Functionality

SMX functionality is provided in the processor through the GETSEC instruction. This instruction supports multiple leaf functions. Leaf functions are selected by the value in EAX at the time GETSEC is executed. Each is referred to as a GETSEC leaf function and addressed separately in this document (even though they share the same opcode, 0F 37).

System software must use the capabilities leaf of GETSEC to discover the available leaf functions associated with GETSEC. Table 2 summarizes available GETSEC leaf functions.

Table 2. Currently Defined GETSEC Leaf Functions

Index (EAX)	Leaf function	Description
0	CAPABILITIES	Returns the available leaf functions of the GETSEC instruction
1	Undefined	Reserved
2	ENTERACCS	Enter authenticated code execution mode
3	EXITAC	Exit authenticated code execution mode
4	SENTER	Launch a protected environment
5	SEXIT	Exit the protected environment

Index (EAX)	Leaf function	Description
6	PARAMETERS	Return SMX related parameter information
7	SMCTRL	SMX mode control
8	WAKEUP	Wake up sleeping processors in secured mode
9 - (4G-1)	Undefined	Reserved

2.1.2 Enabling SMX Capabilities

System software enables SMX operation by setting CR4.SMXE[Bit 14] = 1 before attempting to execute GETSEC. Otherwise, execution of GETSEC results in the processor signaling an invalid opcode exception (#UD).

If the CPUID SMX feature flag is clear (CPUID.01H.ECX[Bit 6] = 0), attempting to set CR4.SMXE[Bit 14] results in a general protection exception.

The IA32_FEATURE_CONTROL MSR (at address 03AH) provides feature control bits that configure operation of VMX and SMX. These bits are documented in Table 3.

Table 3. Format of IA32_FEATURE_CONTROL MSR

Bit Position	Content
0	Lock bit (0 = unlocked, 1 = locked)
1	Enable VMXON in SMX operation
2	Enable VMXON outside SMX operation
7:3	Reserved
15:8	SENTER enables (See Table 10 for detail)
31: 16	Reserved

These bullets describe the information in the Table 3:

- Bit 0 is a lock bit. If the lock bit is clear, VMXON will cause a general-protection exception. Attempting to execute GETSEC[SENTER] when the lock bit is clear will also cause a general-protection exception. If the lock bit is set, WRMSR to the IA32_FEATURE_CONTROL MSR will cause a general-protection exception. Once the lock bit is set, the MSR cannot be modified until a power-on reset.

  System BIOS can use this bit to provide a setup option for BIOS to disable support for VMX or both VMX and SMX.

- Bit 1 enables VMXON in SMX operation (between executing the SENTER and SEXIT leaves of GETSEC). If this bit is clear, VMXON will cause a general-protection exception if executed in SMX operation.
- Bit 2 enables VMXON outside SMX operation. If this bit is clear, VMXON will cause a general-protection exception if executed outside SMX operation.
- Bits 8 through 15 specify enabled SENTER leaf functions. Only enabled SENTER leaf functions can be used when executing SENTER. See Table 10 for information.

2.2 SMX Instruction Summary

System software must first query for available GETSEC leaf functions by executing GETSEC[CAPABILITIES]. The CAPABILITIES leaf function returns a bit map of available GETSEC leaves. An attempt to execute an unsupported leaf index results in an undefined opcode (#UD) exception.

2.2.1 GETSEC[PARAMETERS]

If the GETSEC[PARAMETERS] leaf function is present, it is used to report attributes, options and limitations of SMX operation. Software uses this leaf to identify operating limits or additional options.

GETSEC[PARAMETERS] reports data using general-purpose registers. The information reported by GETSEC[PARAMETERS] may require executing the leaf multiple times using EBX as an index. If the GETSEC[PARAMETERS] instruction leaf or specific parameter field is not available, then SMX operation should be interpreted to use the default limits of respective GETSEC leaves or parameter fields defined in the GETSEC[PARAMETERS] leaf.

2.2.2 GETSEC[SMCTRL]

The GETSEC[SMCTRL] instruction is used for providing additional control over specific conditions associated with the SMX architecture. An input register is supported for selecting the control operation to be performed. See the specific leaf description for details on the type of control provided.

2.2.3 GETSEC[ENTERACCS]

The GETSEC[ENTERACCS] leaf enables authenticated code execution mode. The ENTERACCS leaf function performs an authenticated code module load using the chipset public key as the signature reference. ENTERACCS requires the existence of a LaGrande Technology capable chipset since it unlocks the chipset private configuration register space after successful authentication of the loaded module. The physical base address and size of the authenticated code module are specified as input register values in EBX and ECX, respectively.

While in the authenticated code execution mode, certain processor state properties change. For this reason, the time in which the processor operates in authenticated code execution mode should be limited to minimize impact on external system events.

Upon entry into authenticated code execution mode, the previous paging context is disabled (since the authenticated code module image is specified with physical addresses and can no longer rely upon external memory-based page-table structures).

2.2.4 GETSEC[EXITAC]

GETSEC[EXITAC] takes the processor out of authenticated code execution mode. When this instruction leaf is executed, the contents of the authenticated code execution area are scrubbed and control is transferred to the non-authenticated context defined by a near pointer passed with the GETSEC[EXITAC] instruction.

The authenticated code execution area is no longer accessible after completion of GETSEC[EXITAC]. RBX (or EBX) holds the address of the near absolute indirect target to be taken. The locations of all descriptor tables, page tables, and any other memory-based data structures used after exiting authenticated code execution mode must be held outside of the authenticated code module boundaries. This is so they can continue to be accessible after GETSEC[EXITAC].

2.2.5 GETSEC[SENTER]

The GETSEC[SENTER] leaf function is used to launch a protected environment. GETSEC[SENTER] can be considered a superset of the ENTERACCS leaf as it enters authenticated code execution mode as part of the protected environment launch.

Protected environment startup consists of the following steps:

- Rendezvous other logical processors into a controlled mode
- Load and authenticate the specified code (AC) module
- Unlock the private register space of the enhanced-technology enabled chipset
- Enter authenticated code execution mode with an enhanced-technology chipset authenticated code module

The rendezvous process is performed using messages between the logical processor(s) and the chipset. At the completion of this handshake, all logical processors except for the logical processor initiating the protected environment launch (by executing GETSEC[SENTER]) are placed in a newly defined SENTER sleep state. These logical processor(s) are then activated in a controlled manner to join the protected environment, after the initiating logical processor executes GETSEC function WAKEUP.

The purpose of executing an AC module as part of the GETSEC[SENTER] process is to facilitate the accurate measurement of the MVMM and to help ensure a standard starting configuration for the MVMM. AC module execution also operates in a manner that is tamper-resistant. The processor and chipset must both be LaGrande Technology enabled to successfully execute GETSEC[SENTER].

SENTER initializes and extends into TPM PCR 17 the measurement of the AC module measurement and initiating GETSEC[SENTER] parameters.

Completion of the authenticated code module is achieved by execution of the GETSEC[EXITAC] instruction function. Refer to the definitions of GETSEC[SENTER] and GETSEC[ENTERACCS] for details. A LaGrande Technology-capable chipset may also carry out tighter enforcement actions when a secured processor rendezvous is active.

2.2.6 GETSEC[SEXIT]

Exit the protected environment by executing the instruction GETSEC[SEXIT]. This instruction sends a message that rendezvous other logical processors for exiting from the protected environment. External events (if left masked) are unmasked, LaGrande Technology-capable chipset private configuration space is re-locked, and the internal processor SENTER state flag is cleared.

Upon completion of the GETSEC[SEXIT] instruction, processor execution (ILP) continues with the next instruction in the code stream. Responding logical processors, in response to a bus message, also continue execution with the next instruction that was to be executed at the time the original event was recognized. If other logical processor(s) are still in the SENTER sleep state, then they are transitioned to the wait-for-SIPI state, with a state initialization performed comparable to a soft reset (INIT). Then, a conventional APIC based startup inter-processor interrupt can be delivered to reactivate such processors.

2.2.7 GETSEC[WAKEUP]

Responding logical processors (RLPs) are placed in the SENTER sleep state after the initiating logical processor (ILP) executes GETSEC[SENTER]. The ILP can wake up RLPs to join the protected environment by using GETSEC[WAKEUP]. The ILP can execute GETSEC[WAKEUP] under the following conditions:

- the ILP is in the protected environment
- the ILP has exited authenticated code execution mode with GETSEC[EXITAC]

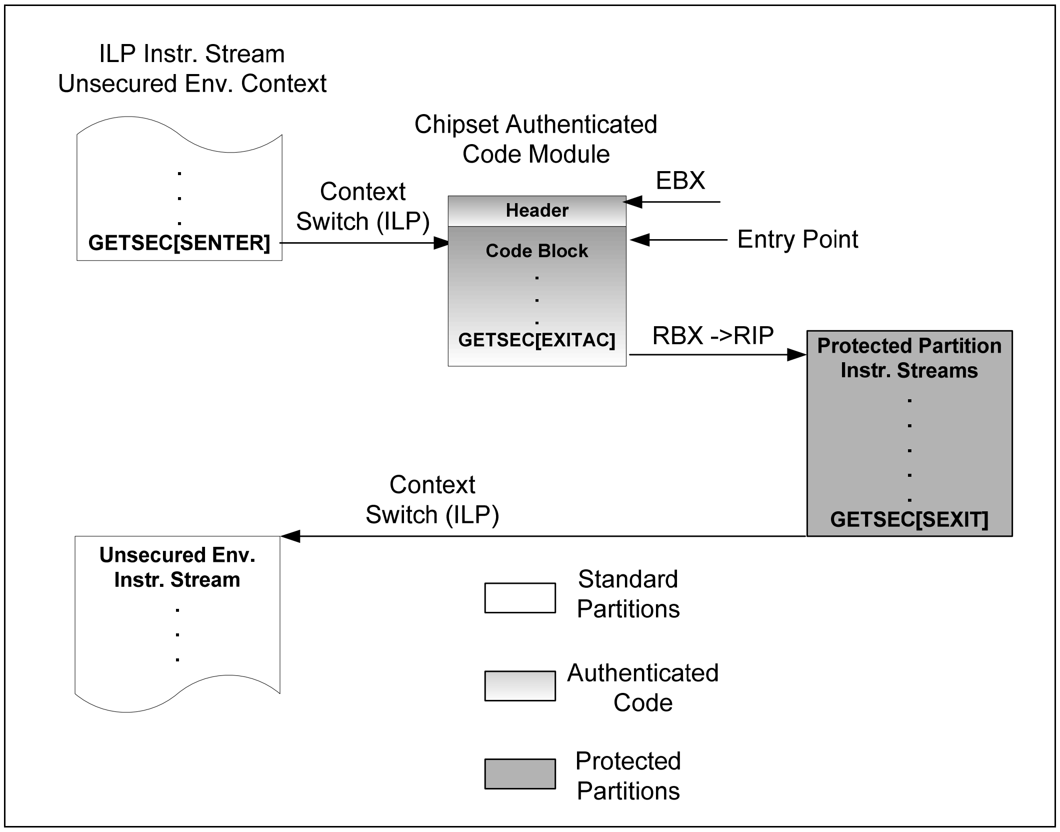
When the RLPs in SENTER sleep state wake up, these logical processors begin execution at the entry point defined in a data structure held in system memory (pointed to by the LaGrande Technology-capable chipset register LT.MVMM.JOIN).

2.2.8 Launching and Shutting Down a Protected Environment

Figure 2 illustrates the life cycle of safer mode operation from a protected environment launch to its shut-down. The life cycle starts with the execution of the GETSEC[SENTER] instruction by system software on the initiating logical processor (ILP). In a multi-threaded or multi-processing environment, this should be done with other logical processors in an idle loop, or asleep (such as after executing HLT).

After the SENTER rendezvous handshake is performed between all logical processors in the platform, the ILP loads the chipset authenticated code module and performs an authentication check. If the check passes, the processor execution context is switched to the authenticated code module at the designated entry point (as defined in the module header). Execution continues within the authenticated code module until the GETSEC[EXITAC] instruction is executed.

Figure 2. Safer Mode Entry and Exit Life Cycle



At this point, the authenticated code execution area within the processor is scrubbed by processor hardware and a near jump to a register-designated location in system memory is performed. If the AC module has performed the necessary functions to configure the platform against unauthorized access from other bus agents, the platform is now considered to be operating in a protected environment.

While executing in a protected environment, the VMM can access the TPM in locality 2. The VMM has complete access to all TPM commands and may use the TPM to report current measurement values or use the measurement values to protect information such that only when the PCR registers contain the same value is the information released from the TPM. This protection mechanism is known as sealing.

A protected environment shutdown is ultimately completed by executing GETSEC[SEXIT]. Prior to this step; system software is responsible for scrubbing sensitive information left in the processor caches, system memory, or I/O state.

2.3 GETSEC Leaf Functions

These sections describe the leaf functions of the GETSEC instruction that are supported by SMX in detail. GETSEC is available only if CPUID.01H.ECX[Bit 6] = 1. This indicates the availability of SMX and the GETSEC instruction. Before GETSEC can be executed, SMX must be enabled by setting CR4.SMXE[Bit 14] = 1.

A GETSEC leaf can only be used if it is shown to be available as reported by the GETSEC[CAPABILITIES] function. Attempts to access a GETSEC leaf index not supported by the processor, or if CR4.SMXE is 0, results in the signaling of an undefined opcode exception.

All GETSEC leafs are available in protected mode, compatibility sub-mode of IA-32e mode, and 64-bit sub-mode of IA-32e mode. Unless otherwise noted, the behavior of all GETSEC functions and interactions related to the authenticated code execution mode or protected environment are independent of IA-32e mode. This also applies to the interpretation of register widths[1] passed as input parameters to GETSEC functions and to register results returned as output parameters.

The GETSEC functions ENTERACCS, SENTER, SEXIT, and WAKEUP require a LaGrande Technology capable-chipset to be present in the platform. The GETSEC[CAPABILITIES] returned bit vector in position 0 indicates if a LaGrande Technology-capable chipset has been sampled present[2] by the processor.

The processor's operating mode also affects the execution of the following GETSEC leaf functions: SMCTRL, ENTERACCS, EXITAC, SENTER, SEXIT, and WAKEUP. These functions are only allowed in protected mode at CPL = 0. They are not allowed while in SMM in order to prevent potential intra-mode conflicts. Further execution qualifications exist to prevent potential architectural conflicts (for example: nesting of the protected environment or authenticated code execution mode). See the definitions of the GETSEC leaf functions for specific requirements.

For the purpose of performance monitor counting, the execution of GETSEC functions is counted as a single instruction with respect to retired instructions. The response by a responding logical processor (RLP) to messages associated with GETSEC[SENTER] or GETSEC[SEXIT] is transparent to the retired instruction count on the ILP.

---

[1]This document uses the 64-bit notation RAX, RIP, RSP, RFLAGS, etc. for processor registers because most processors that support VMX operation also support Intel EM64T. MVMM can be launched in IA-32e mode or outside IA-32e mode. The 64-bit notation of processor registers also refer to its 32-bit forms if SMX operation occurs in 32-bit environment. In some places, notation such as EAX, EBX is used to refer specifically to lower 32 bits of the indicated register.

[2]Sampled present means that the CPU sent a message to the chipset and the chipset responded that it (a) knows about the message and (b) is capable of executing SENTER. This means that the chipset CAN support LaGrande Technology, and is configured and WILLING to support it.

2.3.1 IA32_FEATURE_CONTROL and GETSEC LEAVES

The IA32_FEATURE_CONTROL MSR (at address 03AH) provides additional platform level control over the launch of the protected environment. The properties controlled by IA32_FEATURE_CONTROL must be initialized at power-up by the system BIOS. On power-up reset, the MSR resets to zero, indicating the SENTER function is disabled by default.

The MSR must first be programmed by system BIOS to a configuration consistent with its parameter control usage (see the SENTER leaf description for more details) and locked before system software can execute GETSEC[SENTER]. If SMX functionality is not available (CPUID.01H.ECX[Bit 6] = 0), then the bit fields in the IA32_FEATURE_CONTROL MSR pertaining to SMX control are reserved. The same is true for fields applying to other functions not present for a given processor product. More information about this MSR as it pertains to SMX can be found in the GETSEC[SENTER] instruction description.

GETSEC[CAPABILITIES] – Report the SMX Capabilities

Opcode	Instruction	Description
0F 37 (EAX = 0)	GETSEC[CAPABILITIES]	Report the SMX capabilities. The capabilities index is input in EBX with the result returned in EAX.

Description

The GETSEC[CAPABILITIES] instruction returns a bit vector of supported GETSEC leaf functions. EBX is used as the selector for returning the bit vector field in EAX. GETSEC[CAPABILITIES] may be executed at all privilege levels, but the CR4.SMXE bit must be set, or an undefined opcode exception (#UD) is returned.

With EBX = 0 upon execution of GETSEC[CAPABILITIES], EAX returns the a bit vector representing status on the presence of a LaGrande Technology-capable chipset and the first 30 available GETSEC leaf functions. The format of the returned bit vector is provided in Table 4.

Refer to Table 4. If bit 0 is set to 1, then a LaGrande Technology-capable chipset is present. If bits in the range of 1-30 are set, then the corresponding GETSEC leaf function is available. If the bit vector position is 0, then the GETSEC leaf function corresponding to that index is unsupported and attempted execution results in a #UD.

Bit 31 of EAX indicates if further leaf indexes are supported. If the Extended Leafs bit 31 is set, then additional leaf functions are accessed by repeating GETSEC[CAPABILITIES] with EBX incremented by one. When the most significant bit of EAX is not set, then additional GETSEC leaf functions are not supported; indexing EBX to a higher value results in EAX returning zero.

Table 4. Capabilities Result Encoding (EBX=0)

Field	Bit position	Description
Chipset present	0	LaGrande Technology-capable chipset is present
Undefined	1	Reserved
ENTERACCS	2	GETSEC[ENTERACCS] is available
EXITAC	3	GETSEC[EXITAC] is available
SENTER	4	GETSEC[SENTER] is available
SEXIT	5	GETSEC[SEXIT] is available
PARAMETERS	6	GETSEC[PARAMETERS] is available
SMCTRL	7	GETSEC[SMCTRL] is available
WAKEUP	8	GETSEC[WAKEUP] is available
Undefined	9-30	Reserved
ExtendedLeafs	31	Reserved for Extended Information Reporting

The available leafs as reported in EAX is independent of the processor mode, even though in certain processor contexts some or all of GETSEC leafs may not be accessible.

Flags Affected

None.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
All others	Ignored

Protected Mode Exceptions

#UD	IF CR4.SMXE = 0.

Real-Address Mode Exceptions

#UD	IF CR4.SMXE = 0.

Virtual-8086 Mode Exceptions

#UD	IF CR4.SMXE = 0.

Compatibility Mode Exceptions

#UD	IF CR4.SMXE = 0.

64-bit Mode Exceptions

#UD	IF CR4.SMXE = 0.

GETSEC[ENTERACCS] – Execute Authenticated Chipset Code

Opcode	Instruction	Description
0F 37 (EAX=2)	GETSEC[ENTERACCS]	Enter authenticated execution code mode. EBX holds the authenticated code module physical base address. ECX holds the authenticated code module size (bytes).

Description

The GETSEC[ENTERACCS] instruction loads, authenticates and executes an authenticated code module using an SMX supporting chipset's public key. The ENTERACCS leaf of GETSEC is selected with EAX set to 2 at entry.

There are certain restrictions enforced by the processor for the execution of the GETSEC[ENTERACCS] instruction:

- Execution is not allowed unless the processor is in protected mode with CPL = 0 and EFLAGS.VM = 0.
- Processor cache must be available and not disabled using the CR0.CD and NW bits.
- For processor packages containing more than one logical processor, CR0.CD is checked to ensure consistency between enabled logical processors.
- For enforcing consistency of operation with numeric exception reporting using Interrupt 16, CR0.NE must be set.
- A LaGrande Technology-capable chipset must be present as communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor can not already be in authenticated code execution mode as launched by a previous GETSEC[ENTERACCS] or GETSEC[SENTER] instruction without a subsequent exiting using GETSEC[EXITAC].
- To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or VMX operation.
- To insure consistent handling of SIPI messages, the processor executing the GETSEC[ENTERACCS] instruction must also be designated the BSP (boot-strap processor) as defined by the register bit in the IA32_APIC_BASE MSR.

Failure to conform to the above conditions results in the processor signaling a general protection exception.

Prior to execution of the ENTERACCS leaf, other logical processor(s) in the system must be idle in a wait-for-SIPI state (as initiated by an INIT assertion or through reset for non-BSP designated processors). Alternatively other logical processor(s) may be in the SENTER sleep state as initiated by a GETSEC[SENTER] from the initiating logical processor. If other logical processor(s) in the same package are not idle in one of these states, execution of ENTERACCS signals a general protection exception. The same requirement and action applies if the other logical processor(s) of the same package do not have CR0.CD = 0.

A successful execution of ENTERACCS results in the processor entering an authenticated code execution mode. Prior to reaching this point, the processor performs several checks. These include:

- Establish and check the location and size of the specified authenticated code module to be executed by the processor.
- Broadcast a message to the chipset to enable protection of memory and I/O from activities from other processor agents.
- Load the designated code module into authenticated code execution area.
- Isolate the contents of authenticated code execution area from further state modification by external agents.
- Authenticate the contents of the authenticated code module.
- Initialize the initiating logical processor state based on information contained in the authenticated code module header.
- Unlock the LaGrande Technology-capable chipset private configuration space and TPM locality 3 space.
- Begin execution in the authenticated code module at the defined entry point.
- Inhibit the processor response to the external events: INIT, A20M, NMI and SMI.

The processor masks the response to the assertion of the external signals INIT#, A20M, NMI#,and SMI#. This masking remains active until optionally unmasked by GETSEC[EXITAC] (This defined unmasking behavior assumes GETSEC[ENTERACCS] was not executed by a prior SENTER). The purpose of this masking control is to prevent exposure to existing external event handlers that may not be under the control of the authenticated code module. Once the authenticated code module is launched at the completion of ENTERACCS, it is free to enable interrupts by setting EFLAGS.IF and enable NMI by execution of IRET. This presumes that it has re-established interrupt handling support under the authenticated execution context through initialization of the IDT, GDT, and corresponding interrupt handling code.

SMI# remains masked throughout authenticated code execution mode and can not be unmasked until this mode is exited via GETSEC[EXITAC]. The state of the A20M pin is likewise masked and forced internally to a de-asserted state so that any external assertion is not recognized during authenticated code execution mode. A20M masking stays in effect until exiting authenticated code execution mode.

The GETSEC[ENTERACCS] function requires two additional input parameters input using the general purpose registers EBX and ECX. EBX holds the authenticated code (AC) module physical base address (the AC module must reside below 4 GBytes in physical address space) and ECX holds the programmer specified AC module size (in bytes). The physical base address and size are used to retrieve the code module from system memory and load into the internal authenticated code execution area. The base physical address is checked to verify it is on a modulo-4096 byte boundary. The size is verified to be a multiple of 64, that it does not exceed the internal authenticated code execution area capacity, and that the top address of the AC module does not exceed 32 bits. An error condition results in an abort of the authenticated code execution launch and the signaling of a general protection exception.

As an integrity check for proper processor hardware operation, execution of GETSEC[ENTERACCS] will also check the contents of all the machine check status registers (as reported by the MSRs IA32_MCi_STATUS) for any valid uncorrectable error condition. In addition, the global machine check status register IA32_MCG_STATUS MCIP bit must be cleared and the IERR processor package pin must not be asserted, indicating that no machine check exception processing is currently in progress. These checks are performed prior to initiating the load of the authenticated code module. Any outstanding valid uncorrectable machine check error condition present in these status registers at this point will result in the processor signaling a general protection violation.

To prevent other (logical) processors from interfering with the ILP operating in authenticated code execution mode, the chipset enforces protection on memory (excluding implicit write-back transactions) or I/O activities originating from other processor agents. This protection starts when the ILP enters into authenticated code execution mode. The chipset registers the agent ID of the ILP and only allows memory or I/O transactions initiated from this registered agent. Exiting authenticated code execution mode is done by executing GETSEC[EXITAC]. The protection of bus access remains in effect until the ILP executes GETSEC[EXITAC].

Once the authenticated code module has been loaded into authenticated code execution area, it is protected against further modification from external bus snoops. There is also a requirement that the memory type for the authenticated code module address range be WB (via initialization of the MTRRs prior to execution of this instruction). If this condition is not satisfied, it is a violation of security and the processor may generate an external shutdown condition by writing an error code to the LaGrande Technology-capable chipset ERRORCODE register, and then writing to the LaGrande Technology-capable chipset CMD.SYS-RESET register. This action is referred to as a LaGrande Technology shutdown condition. It is performed when it is considered unreliable to signal an error through the conventional exception reporting mechanism. For details on memory type restrictions associated with authenticated code modules, see Appendix A, “Authenticated-Code Module.”

To conform to the minimum granularity of MTRR MSRs for specifying the memory type, authenticated RAM is allocated to the processor in 4096 byte granular blocks. If an AC module size as specified in ECX is not a multiple of 4096, then the processor will allocate up to the next 4096 byte boundary for mapping as authenticated RAM with indeterminate data. This pad area will not be visible to the authenticated code module as external memory nor can it depend on the value of this data used to fill the pad area.

An authenticated code module is required to conform to a specific format. At the top level the module is composed of three sections: Module header, internal working scratch space, and user code and data. The module header contains critical information necessary for the processor to properly authenticate the entire module, including the encrypted signature and RSA based public key. The processor also uses other fields of the AC module for initializing the remaining processor state after authentication. The definition of the authenticated code module format can be found in Appendix A.

Authentication is performed after loading of the code module into the authenticated code execution area. Information from the authenticated code module header is used to support the authentication process. The RSAPubKey header field contains a public key plus a 32 bit exponent used for decrypting the signature of the authenticated code module. The signature is held in encrypted form in the RSASig header field and it represents the PKCS #1.5 RSA Signature of the module. The RSA Signature signs an area that includes the sum of the module header and all of the USER AREA data field, which represents the body of the module. Those parts of the module header not included are: the RSA Signature, the public key, and the scratch field. An inconsistent authenticated code module format, inconsistent comparison of the public key hash, or mismatch of the decrypted signature against the computed hash of the authenticated module or a corrupted signature padding value results in an abort of the authentication process and signaling of a LaGrande Technology shutdown condition. As part of the authentication step, the processor stores the decrypted signature of the AC module in the first 20 bytes of the 'Scratch' field of the AC module header (see Appendix A).

After authentication has completed successfully, the private configuration space of the LaGrande Technology-capable chipset is unlocked. At this point, only the authenticated code module or system software executing in authenticated code execution mode is allowed to gain access to the restricted chipset state for the purpose of securing the platform.

The architectural state of the processor is partially initialized from contents held in the header of the authenticated code module. The processor GDTR, CS, and DS selectors are initialized from fields within the authenticated code module defined in Table 5. Since the authenticated code module must be relocatable, all address references must be relative to the authenticated code module base address in EBX. The processor GDTR base value is initialized to the AC module header field GDT BasePtr + module base address held in EBX and the GDTR limit is set to the value in the GDT Limit field. The CS selector is initialized to the AC module header SegSel field, while the DS selector is initialized to CS + 8. The segment descriptor fields are implicitly initialized to BASE=0, LIMIT=FFFFFh, G=1, D=1, P=1, S=1, read/write access for DS, and execute/read access for CS. The processor begins the authenticated code module execution with the EIP set to the AC module header EntryPoint field + module base address (EBX). The AC module based fields used for initializing the processor state are checked for consistency and any failure results in a shutdown condition.

A summary of the processor state initialization after successful completion of GETSEC[ENTERACCS] is given for the processor in Table 5. The paging is disabled upon entry into the secure execution. The authenticated code module is loaded and initially executed using physical addresses. It is up to the system software after execution of GETSEC[ENTERACCS] to establish a new trusted paging environment with an appropriate mapping to meet new protection requirements for the authenticated execution. EBP is initialized to the authenticated code module base physical address for initial execution in the authenticated environment. As a result, the authenticated code can reference EBP for relative address based references given that the authenticated code module must be position independent.

Table 5. Processor state initialization after GETSEC[ENTERACCS]

Processor state	Status
CR0	Clear PG, AM, WP
CR4	Clear MCE
EFLAGS	00000002h
IA32_EFER	0
EIP	AC.base (EBX) + [EntryPoint]
[E|R]BX	[E|R]IP of the instruction after GETSEC[ENTERACCS]
[E|R]CX	Pre-ENTERACCS state: [31:16]=GDTR.limit; [15:0]=CS.sel
[E|R]DX	Pre-ENTERACCS state: GDTR.base
EBP	AC.base (EBX)
CS	Sel=[SegSel], base=0, limit=FFFFFh, G=1, D=1, AR=9Bh
DS	Sel=[SegSel] + 8, base=0, limit=FFFFFh, G=1, D=1, AR=93h
GDTR	Base= AC.base (EBX) + [GDTBasePtr], Limit=[GDTLimit]
DR7	00000400h
IA32_DEBUGCTL	0
IA32_MISC_ENABLE MSR	See Table 6

The segmentation related processor state that has not been initialized by GETSEC[ENTERACCS] requires appropriate initialization before use. Since a new GDT context has been established, the previous state of the segment selector values held in ES, SS, FS, GS, TR, and LDTR might not be valid. The IDTR will also require reloading with a new IDT context after entering authenticated code execution mode, before any exceptions or the external interrupts INTR and NMI can be handled. Since external interrupts are re-enabled at the completion of authenticated code execution mode (as terminated with EXITAC), it is recommended that a new IDT context be established before this point. Until such a new IDT context is established, the programmer must take care in not executing an INT n instruction or any other operation that would result in an exception or trap signaling.

Debug exception and trap related signaling is also disabled as part of GETSEC[ENTERACCS]. This is achieved by resetting DR7, TF in EFLAGs, and the MSR IA32_DEBUGCTL. These debug functions are free to be re-enabled once supporting exception handler(s), descriptor tables, and debug registers have been properly initialized following entry into authenticated code execution mode. Also, any pending single-step trap condition will have been cleared upon entry into this mode.

In 64-bit mode, RBX holds the RIP of the next instruction, RCX holds the CS selector value and GDTR limit in the lower 32 bits while zeroing the upper 32 bits, and RDX holds the GDTR base field of the processor state prior to ENTERACCS. All other architectural register state not referenced in Table 5 will be unmodified by execution of GETSEC[ENTERACCS].

Prior to completion of the GETSEC[ENTERACCS] instruction and after successful authentication of the AC module, the LaGrande Technology private space registers are unlocked to be accessible by authenticated code module. This private configuration space can be optionally locked later by software writing to the LaGrande Technology-capable chipset public space location LT.CMD.CLOSE-PRIVATE. The authenticated code module is allowed to access to locality-3 of TPM resources. Locality-3 TPM resources are closed upon successful execution of GETSEC[EXITAC].

The miscellaneous feature control MSR IA32_MISC_ENABLES is initialized as part of the protected environment launch. Certain bits of this MSR are preserved, because preserving these bits may be important to maintain previously established platform settings. See the footnote for Table 6. The remaining bits are cleared for the purpose of establishing a more consistent environment for the execution of authenticated code modules. Among the impact of initializing this MSR, any previous condition established by the MONITOR instruction will be cleared.

Table 6. IA32_MISC_ENABLES Functions Initialized by ENTERACCS/SENTER

Function	Bit #	Initialization action
Fast strings enable	0	Clear to 0
MT thread priority	1	Clear to 0
FOPCODE compatibility mode enable	2	Clear to 0
Thermal monitor enable	3	Set to 1 if other thermal monitor capability is not enabled. [1]
Split-lock disable	4	Clear to 0
Bus lock on cache line splits disable	8	Clear to 0
Hardware prefetch disable	9	Clear to 0
Intel SpeedStep Technology enable	15	Clear to 0
MONITOR/MWAIT s/m enable	18	Clear to 0
Adjacent sector prefetch disable	19	Clear to 0
Context ID bit enable	24	Clear to 0

NOTES:

1. ENTERACCS (and SENTER) initialize the state of processor thermal throttling such that at least a minimum level is enabled. If thermal throttling is already enabled at the time of execution for one of these GETSEC leafs, then no change in the thermal throttling control settings will occur. If thermal throttling is disabled at this time, then execution of ENTERACCS (or SENTER) will enable it via setting of the thermal throttle control bit 3.

To support the possible return to the processor architectural state prior to execution of GETSEC[ENTERACCS], certain critical processor state is captured and stored in the general- purpose registers at instruction completion. EBX holds effective address (EIP) of the instruction following GETSEC[ENTERACCS], ECX[15:0] holds the CS selector value, ECX[31:16] holds the GDTR limit field, and EDX holds the GDTR base field. The subsequent authenticated code can preserve the contents of these registers so that this state can be manually restored if needed, prior to exiting authenticated code execution mode with GETSEC[EXITAC].

Flags Affected

All flags are cleared.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.CD = 1 or CR0.NW = 1 or CR0.NE = 0 or CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1.
	If a LaGrande Technology-capable chipset is not present.
	If VMX mode is currently active as started with VMXON.
	If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP.
	If the processor is already in authenticated code execution mode.
	If the processor is in SMM.
	If a valid uncorrectable machine check error is logged in IA32_MC[I]_STATUS.
	If the authenticated code base is not on a 4096 byte boundary.
	If the authenticated code size > processor internal authenticated code area capacity.
	If the authenticated code size is not modulo 64.
	If other enabled logical processor(s) of the same package CR0.CD = 1.
	If other enabled logical processor(s) of the same package are not in the wait-for-SIPI or SENTER sleep state.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[ENTERACCS] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[ENTERACCS] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[ENTERACCS] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

#GP	IF AC code module does not reside in physical address below 2^{32} -1.

64-bit Mode Exceptions

All protected mode exceptions apply.

#GP	IF AC code module does not reside in physical address below 2^{32} -1

GETSEC[EXITAC] – Exit Authenticated Code Execution Mode

Opcode	Instruction	Description
0F 37 (EAX=3)	GETSEC[EXITAC]	Exit authenticated code execution mode. EBX holds the Near Absolute Indirect jump target and EDX hold the exit parameter flags.

Description

The GETSEC[EXITAC] instruction initiates an exit of authenticated code execution mode established by GETSEC[ENTERACCS] or GETSEC[SENTER]. The EXITAC leaf of GETSEC is selected with EAX set to 3 at entry. EBX (or RBX, if in 64-bit mode) holds the near jump target offset for where the processor execution resumes upon exiting authenticated code execution mode. EDX contains additional parameter control information. Currently only an input value in EDX of 0 is supported. All other EDX settings are considered reserved and result in a general protection violation.

GETSEC[EXITAC] can only be executed if the processor is in protected mode with CPL = 0 and EFLAGS.VM = 0. The processor must also be in authenticated code execution mode. To avoid potential operability conflicts between modes, the processor is not allowed to execute this instruction if it is in SMM or VMX mode. A violation of these conditions results in a general protection violation.

Upon completion of the GETSEC[EXITAC] operation, the processor unmasks responses to external event signals INIT#, NMI#, and SMI#. This unmasking is performed conditionally, based on whether a protected environment had been launched via a previous execution of GETSEC[SENTER]. If a protected environment is active then these external event signals will remain masked if previously established in that state. In this case A20M is kept disabled until the protected environment is exited with GETSEC[SEXIT]. INIT# is unconditionally unmasked by EXITAC. Note that any events that are pending, but have been blocked while in authenticated code execution mode will be recognized at the completion of the GETSEC[EXITAC] instruction if the pin event is unmasked.

The intent of providing the ability to optionally leave the pin events SMI, and NMI masked is to support the completion of a protected environment bring-up that makes use of VMX. In this envisioned security usage scenario, these events will remain masked until an appropriate virtual machine has been established in order to field servicing of these events in a safer manner. Details on when and how events are masked and unmasked in VMX mode are available in the VMXEAS documentation. It should be cautioned that if no VMX environment is to be activated following GETSEC[EXITAC], that these events will remain masked until the protected partition is exited with GETSEC[SEXIT]. If this is not desired then the GETSEC function SMCTRL(0) can be used for unmasking SMI in this context. NMI can be correspondingly unmasked by execution of IRET.

A successful exit of the authenticated code execution mode requires the processor to perform additional steps as outlined below:

- Invalidate the contents of the internal authenticated code execution area..
- Invalidate processor TLBs.
- Clear the internal processor AC Mode indicator flag.
- Re-lock the TPM locality 3 space.
- Unlock the LaGrande Technology-capable chipset memory and I/O protections to allow memory and I/O activity by other processor agents.
- Perform a near absolute indirect jump to the designated instruction location.

The content of authenticated code execution area is invalidated in order to protect it from further use or visibility. This internal processor storage area can no longer be used or relied upon after GETSEC[EXITAC]. Data structures need to be re-established outside of the authenticated code execution area if they are to be referenced after EXITAC. Since addressed memory content formerly mapped to the authenticated code execution area may no longer be coherent with external system memory after EXITAC, processor TLBs in support of linear to physical address translation are also invalidated.

Upon completion of GETSEC[EXITAC], a near absolute indirect transfer is performed with EIP loaded with the contents of EBX (based on the current operating mode size). In 64-bit mode, all 64 bits of RBX are loaded into RIP if REX.W precedes GETSEC[EXITAC]. Otherwise RBX is treated as 32 bits even while in 64-bit mode. Conventional CS limit checking is performed as part of this control transfer. Any exception conditions generated as part of this control transfer will be directed to the existing IDT; thus it an IDTR should also be established prior to execution of the EXITAC function if there is a need for fault handling. In addition, any segmentation related (and paging) data structures to be used after EXITAC should be re-established or validated by the authenticated code prior to EXITAC.

In addition, any segmentation related (and paging) data structures to be used after EXITAC need to be re-established and mapped outside of the authenticated RAM designated area by the authenticated code prior to EXITAC. Any data structure held within the authenticated RAM allocated area will no longer be accessible after completion by EXITAC.

Flags Affected

None.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Causes interpretation of RBX width as 64 bits in 64-bit mode.
All others	Ignored

Protected Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL>0 or EFLAGS.VM =1.
	If VMX mode is currently active as started with VMXON.
	If the processor is not currently in authenticated code execution mode.
	If the processor is in SMM.
	If any reserved bit position is set in the EDX parameter register.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[EXITAC] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[EXITAC] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[EXITAC] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-bit Mode Exceptions

All protected mode exceptions apply.

GETSEC[SENTER] – Enter protected environment

Opcode	Instruction	Description
0F 37 (EAX=4)	GETSEC[SENTER]	Launch protected environment EBX holds the authentication code module physical base address. ECX holds the authenticated code module size (bytes). EDX controls the level of functionality supported by the protected environment launch.

Description

The GETSEC[SENTER] instruction initiates the launch of a protected environment and places the initiating logical processor into the authenticated code execution mode. The SENTER leaf of GETSEC is selected with EAX set to 4 at execution. The physical base address of the code module to be loaded and authenticated is specified in EBX. The size of the module in bytes is specified in ECX. EDX controls the level of functionality supported by the protected environment launch. To enable the full functionality of the protected environment launch, EDX must be initialized to zero.

The launching software must ensure that the TPM.ACCESS_0.activeLocality bit is clear before executing the GETSEC[SENTER] instructions.

There are restrictions enforced by the processor for execution of the GETSEC[SENTER] instruction:

- Execution is not allowed unless the processor is in protected mode or IA-32e mode with CPL = 0 and EFLAGS.VM = 0.
- The processor cache must be available and not disabled using the CR0.CD and NW bits.
- For enforcing consistency of operation with numeric exception reporting using interrupt 16, CR0.NE must be set.
- A LaGrande Technology-capable chipset must be present as communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor can not be in authenticated code execution mode or a protected environment (launched by a previous GETSEC[ENTERACCS] or GETSEC[SENTER] instruction).
- To avoid potential inter-operability conflicts between modes, the processor is not allowed to execute this instruction if currently in SMM or VMX mode.
- To insure consistent handling of SIPI messages, the processor executing the GETSEC[SENTER] instruction must also be designated as the BSP (bootstrap processor) as defined by the register bit in the IA32_APIC_BASE MSR.
- EDX must be initialized to setting supportable by the processor. Unless otherwise enumerated using the GETSEC[PARAMETERS] leaf, only a value of zero is supported.

Failure to abide by the above conditions results in the processor signaling a general protection violation.

This instruction leaf starts the launch of a protected environment by initiating a rendezvous sequence for all logical processors in the platform. The rendezvous sequence involves the Initiating Logical Processor (ILP) sending a message (by executing GETSEC[SENTER]) and other Responding Logical Processors (RLP) acknowledging the message, thus synchronizing the RLP(s) with the ILP.

In response to a message signaling the completion of rendezvous, RLPs clear the bootstrap processor indicator flag (IA32_APIC_BASE.BSP) and enter an SENTER sleep state. In this sleep state, RLPs enter an idle processor condition while waiting to be activated after a protected environment has been established by operating system software. RLPs in the SENTER sleep state are activated by the GETSEC leaf function WAKEUP.

RLP can exit the SENTER sleep state and start execution in response to a WAKUP signal initiated by ILP execution of GETSEC[WAKEUP]. The RLP retrieves a pointer to a data structure that contains information to enable execution from a defined entry point. This data structure is located using a physical address held in the LaGrande Technology-capable chipset configuration register LT.MVMM.JOIN. The register is publicly writable in the chipset by all processors and is not restricted by the LaGrande Technology-capable chipset configuration register lock status. The processor WAKEUP entry point control using the LT.MVMM.JOIN stays in effect while the processor is in protected environment, until successful completion of GETSEC[SEXIT] by the ILP. The format of this data structure is defined in Table 7.

Table 7. RLP Secure Startup Environment Data Structure

Offset from address held in LT.MVMM.JOIN	Field
0	GDT limit
4	GDT base pointer
8	Segment selector initializer
12	Linear IP entry point (physical address)

The JOIN based data structure contains the information necessary to initialize RLP processor state and permit the processor to join the protected environment. The GDTR, LIP, and CS, DS, SS, and ES selector values are initialized using this data structure. The CS selector index is derived directly from the segment selector initializer field; DS, SS, and ES selectors are initialized to CS+8. The segment descriptor fields are initialized implicitly with BASE = 0, LIMIT = FFFFFH, G = 1, D = 1, P = 1, S = 1; read/write/accessed for DS, SS, and ES; and execute/read/accessed for CS. It is the responsibility of external software to establish a GDT pointed to by the JOIN data structure that contains descriptor entries consistent with the implicit settings initialized by the processor. Certain states held in Table 7 are checked for consistency by the processor prior to execution. A failure of any consistency check results in the RLP aborting entry into the protected environment and signaling a LaGrande Technology shutdown condition. The specific checks performed are documented in the SENTER operation description found later in this section. After successful completion of processor consistency checks and subsequent initialization, execution in the protected partition for the RLP begins from the defined Linear IP entry point at offset 12 (as indicated in Table 7).

A successful launch of the protected partition results in the initiating logical processor entering the authenticated code execution mode. Prior to reaching this point, the ILP must perform these steps:

- Inhibit processor response to the external events: INIT, A20M, NMI, and SMI.
- Establish and check the location and size of the authenticated code module to be executed by the ILP.
- Check for the existence of a LaGrande Technology-capable chipset and TPM interface; abort if not present.
- Verify the current power management configuration is acceptable.
- Broadcast a message to the chipset to enable protection of memory and I/O from activities from other processor agents.
- Load the AC module into authenticated code execution area.
- Isolate the content of authenticated code execution area from further state modification by external agents.
- Authenticate the source and contents of the module.
- Updated the LaGrande Technology-capable chipset managed Trusted Platform Module (TPM) with the authenticated code module's hash.
- Initialize processor state based on the authenticated code module header information.
- Unlock the LaGrande Technology-capable chipset private configuration register space and TPM locality 3 space.
- Begin execution in the authenticated code module at the defined entry point.

The ILP and RLP mask the response to the assertion of the external signals INIT#, A20M, NMI#, and SMI#. This masking is held in place until undone via the GETSEC[EXITAC], GETSEC[SEXIT], GETSEC[SMCTRL] or for specific VMX related operations such as a VM entry or the VMXOFF instruction (see the VMX documentation for more details). The purpose of this masking control is to prevent exposure to existing external event handlers until a protected partition has been put in place to directly handle these events. The state of the A20M pin is masked and forced internally to a de-asserted state so that external assertion is not recognized during an active protected partition. A20M masking as set by GETSEC[SENTER] is undone only after taking down the protected partition with the GETSEC[SEXIT] instruction or processor reset. INTR is masked by simply clearing the EFLAGS.IF bit. It is the responsibility of system software to control the processor response to INTR through appropriate management of EFLAGS.

The authenticated code base address and size parameters (in bytes) are passed to the GETSEC[SENTER] instruction using EBX and ECX respectively. The ILP evaluates the contents of these registers according to the rules for the AC module address in GETSEC[ENTERACCS]. AC module execution follows the same rules, as set by GETSEC[ENTERACCS].

Once successful authentication has been completed by the ILP, the computed hash is broadcast for storage in the TPM at PCR17 after this register is implicitly reset (see Appendix B). PCR17 is a dedicated register for holding the computed hash of the authenticated code module, loaded and subsequently executed by the GETSEC[SENTER]. As part of this registration, PCR18-20 are reset so they can be utilized by subsequently loaded external software for registration of code and data modules.

Before loading and authentication of the target code module is performed the processor also checks that the current voltage ID and bus ratio status correspond to known good values supportable by the processor. The MSR IA32_PERF_STATUS values are compared against either the processor supported maximum voltage/bus ration setting, system reset setting of voltage/bus ration, or the thermal throttle operating point. If the current settings do not meet any of these criteria, then the SENTER instruction functionality will attempt to change the voltage ID and bus ratio select processor controls. For a mobile processor, an adjustment will be made to set the voltage and bus ratio to the thermal throttle operating point. For all other processor configurations, the voltage ID will be set to the processor supported maximum voltage operating point. This implies that existing values programmed in MSR_GV3_BRVID_SEL may be overridden by SENTER. The system software environment may need to take responsibility to restoring such settings that are deemed to be safe, but not necessarily recognized by SENTER after this software has been established in the secure context. If an adjustment is not possible when an out of range setting is discovered, then the processor will abort the protected partition launch. This may be the case for chipset controlled settings of these values or if the controllability is not enabled on the processor. In this case it is the responsibility of the external software to program the chipset voltage ID and/or bus ratio select settings to known good values recognized by the processor prior to SENTER.

The computed hash is broadcast to the TPM using the LaGrande Technology-capable chipset in a four part sequence, (including registration of the protected partition launch control parameter held in EDX). This is described below:

1. A single byte write of 0 to the port TPM.HASH.START. This establishes the locality for TPM ownership to the processor (locality 4), triggers a reset of PCR17-20, and clears the PCR input buffer. However, the TPM ignores the data of this write operation.
2. Write all bytes of the computed hash to the TPM PCR input buffer using the direct processor port TPM.HASH.DATA, looping through each of the 20 bytes, starting with the least significant byte of the first 32-bit word of the hash and ending with the most significant byte of the last 32-bit word of the hash.
3. Four single byte writes of the protected partition flags, as indicated by EDX at the time GETSEC[SENTER] is executed, to the TPM PCR input buffer using port TPM.HASH.DATA, starting with the least significant byte first. This is performed to provide traceability to subsequent protected partition code for determining the level of functionality enabled by the protected partition launch.
4. Single byte write of 0 to the port TPM.HASH.END. This causes the data written in steps 2-3 to the PCR input buffer to be hashed and updated in PCR17 of the TPM. However, the TPM ignores the data of this write operation.

After successful execution of the TPM.HASH.END command, the only way to provide additional measurements to PCR17 is to use the TPM_Extend command. PCR17 contains the measurement of AC code and the SENTER launching parameters.

After authentication is completed successfully, the private configuration space of the LaGrande Technology-capable chipset is unlocked so that authenticated code module or system software executing in authenticated code execution mode can gain access to this normally restricted chipset state This is done for the purpose of launching a protected partition in the platform. The LaGrande Technology-capable chipset private configuration space can be locked back later by software writing to the chipset memory-mapped public space location LT.CMD.CLOSE-PRIVATE or unconditionally using the GETSEC[SEXIT] instruction.

Table 8 provides a summary of processor state initialization for the ILP and RLP (after successful completion of the GETSEC[SENTER]). For both ILP and RLP, paging is disabled upon entry to the protected partition. The authenticated code module is loaded and initially executed using physical addresses. It is up to the ILP to establish a trusted paging environment, with appropriate mapping, to meet protection requirements established during the launch of the protected partition.

EBP is initialized to the authenticated code module base address in the ILP upon initial execution in the authenticated code module. As a result, authenticated code can reference EBP for relative address based references, given the authenticated code module may be position independent. RLP state initialization is not completed until a subsequent wakeup has been signaled by execution of the GETSEC[WAKEUP] by the ILP. This is because some of the RLP state initialization is dependent upon the data structure pointed to by the chipset register LT.MVMM.JOIN, which may not be initialized at the time GETSEC[SENTER] is executed.

Table 8. ILP and RLP Processor State Initialization After GETSEC[SENTER]

Processor state	ILP	RLP
CR0	Clear PG, AM, WP	Clear PG, CD, NW, AM, WP. Set PE, NE.
CR4	00004000H	00004000H
EFLAGS	00000002H	00000002H
EIP	AC.base (EBX) + [EntryPoint]	[LT.MVMM.JOIN+12]
EBP	AC.base (EBX)	NC

CS	Sel = [SegSel], base = 0, limit = FFFFFH,
	
	G = 1, D = 1, AR = 9BH
		Sel = [LT.MVMM.JOIN + 8], base = 0, limit = FFFFFH,
		
		G = 1, D = 1, AR = 9BH

DS	Sel = [SegSel] + 8, base = 0, limit = FFFFFH,
	
	G = 1, D = 1, AR = 93H
		Sel = [LT.MVMM.JOIN + 8] + 8, base = 0, limit = FFFFFH,
		
		G = 1, D = 1, AR = 93H

ES	Sel = [SegSel] + 8, base = 0, limit = FFFFFH,
	
	G = 1, D = 1, AR = 93H
		Sel = [LT.MVMM.JOIN + 8] + 8, base = 0, limit = FFFFFH,
		
		G = 1, D = 1, AR = 93H

SS	Sel = [SegSel] + 8, base = 0, limit = FFFFFH,
	
	G = 1, D = 1, AR = 93H
		Sel = [LT.MVMM.JOIN + 8] + 8, base = 0, limit = FFFFFh,
		
		G = 1, D = 1, AR = 93H

GDTR	Base = AC.base (EBX) + [GDTBasePtr],
	Limit = [GDTLimit]
		Base = [LT.MVMM.JOIN + 4],
		Limit = [LT.MVMM.JOIN]

DR7	00000400H	00000400H
IA32_DEBUGCTL	0	0
IA32_EFER	0	0

Processor state	ILP	RLP
IA32_MISC_ENABLE MSR	See Table 6.	See Table 6.
Performance counters and counter control	0	0

Segmentation related processor state that has not been initialized by GETSEC[SENTER] requires appropriate initialization before use. Since a new GDT context has been established, the previous state of the segment selector values held in FS, GS, TR, and LDTR may no longer be valid. The IDTR will also require reloading with a new IDT context after launching the protected partition before exceptions or the external interrupts INTR and NMI can be handled. In the meantime, the programmer must take care in not executing an INT n instruction or any other condition that would result in an exception or trap signaling.

Debug exception and trap related signaling is also disabled as part of execution of GETSEC[SENTER]. This is achieved by clearing DR7, TF in EFLAGs, and the MSRIA32_DEBUGCTL as defined in Table 8. These can be re-enabled once supporting exception handler(s), descriptor tables, and debug registers have been properly re-initialized following SENTER. Also, any pending single-step trap condition will be cleared at the completion of SENTER for both the ILP and RLP.

MVMM Header

Table 9 shows the format of the MVMM Header structure which is stored within the MVMM image. The MVMM Header structure is used by the SINIT module to set up the correct initial MVMM state and to find the MVMM entry point. The header is part of the MVMM hash.

Table 9. MVMM Header structure

Field	Offset	Size	Description
GUID	0	16	Identifies this structure
HeaderLen	16	4	Length of header in bytes
Version	20	4	Version number of this structure
Entry Point	24	4	Linear entry point of MVMM
Reserved	28	8	Reserved

GUID: This field contains a GUID which uniquely identifies this MVMM Header Structure. The GUID is defined as follows:

```
ULONG   GUID0;      // 9082AC5A
ULONG   GUID1;      // 74A7476F
ULONG   GUID2;      // A2555C0F
ULONG   GUID3;      // 42B651CB
```

HeaderLen: this field contains the length in bytes of the MVMM Header Structure.

Version: this field contains the version of the MVMM header. The initial Version will be 10000H where the upper two bytes are the major version and the lower two bytes are the minor version.

All addresses within the MVMM Header Structure are linear address in the MVMM's linear address space.

Prior to launching the protected environment, system software places the linear address of the structure into LT Device memory (see Appendix C). This linear address is the linear address of the structure within the MVMM page table context.

MVMM Page Table

The MVMM is not required to be loaded into physically contiguous memory since the MVMM code will run in protected mode with paging enabled. The pages containing the MVMM image must be pinned in memory, and all these pages must be located in physical memory below 4G bytes and above 1M bytes.

System software creates an MVMM page table structure to map the entire MVMM image. The pages containing the MVMM page tables must be pinned in memory prior to launching the protected environment. The MVMM page table structure must be in the format of the IA-32 Physical Address Extension (PAE) page table structure.

The MVMM page table has several special requirements:

- The MVMM page tables may contain only 4 kbyte pages.
- A breadth-first search of page tables must produce increasing physical addresses.
- No address may overlap with device memory (GART, etc)
- No address between (640k, 1M] or above Top of Memory (TOM)
- The Page Directories must be in a lower physical address than the Page Tables.
- The Page-Directory-Pointer-Table must be in a lower physical address than the Page Directories.

The SINIT code will check that the MVMM page table matches these requirements before calculating the MVMM digest. The second rule above implies that the MVMM must be loaded into physical memory in an ordered fashion: a scan of MVMM virtual addresses must find increasing physical addresses. The system software can order its list of physical pages before loading the MVMM image into memory.

System software writes the physical base address of the MVMM Page Table's page directory to the LT heap. The size in bytes of the MVMM image is also written to the LT heap, see Appendix C.

Interaction with Specific MSRs

The IA32_MISC_ENABLE MSR is initialized as part of the protected partition launch. Certain bits of this MSR are preserved, because preserving these bits may be important for maintaining previously established platform settings. Bits impacted may be platform specific. The remaining bits are re-initialized to specific settings for the purpose of establishing a consistent environment for the execution of authenticated code modules as defined in Table 8.

One of the impacts of initializing IA32_MISC_ENABLE is that any previous condition established by the MONITOR instruction is cleared. This implies that a subsequent system software environment may have to re-establish specific settings for this MSR in order to meet particular system requirements. Control for any function not listed in Table 8 is defined as unchanged. Note that not all the control functions listed in Table 8 may be present in all SMX capable processors. Treatment of bits designated as reserved for a specific processor is to leave them unchanged.

Performance related counters and counter configuration MSRs are cleared as part of execution of SENTER on both the ILP and RLP(s). This implies any active performance counters at the time of SENTER execution are disabled. To activate processor performance counters, this state must be re-initialized and re-enabled.

Since MCE (along with all other state bits, with the exception of SMXE) are cleared in CR4 upon execution of SENTER processing, any enabled machine check error condition that occurs results in the processor performing the LaGrande Technology shutdown. This also applies to an RLP while in the SENTER sleep state. For each logical processor, CR4.MCE must be reestablished with a valid machine check exception handler in order to avoid a LaGrande Technology shutdown.

Effect of MSR IA32_FEATURE_CONTROL

Bits 15:8 of the IA32_FEATURE_CONTROL affect the execution of GETSEC[SENTER]. These bits consist of two fields:

- Bit 15: a global enable control for execution of SENTER
- Bits 14:8: a parameter control field providing the ability to qualify SENTER execution based on the level of functionality specified with corresponding EDX parameter bits 6:0.

The layout of these fields in IA32_FEATURE_CONTROL is shown in Table 10.

The IA32_CR_FEATURE_CONTROL MSR must be initialized prior to execution of SENTER with the lock bit set to affirm the settings to be used. Once the lock bit is set, only a power-up reset condition will clear this MSR. IA32_CR_FEATURE_CONTROL MSR must be configured in accordance to the intended usage at platform initiation. Note that this MSR is only available on SMX or VMX enabled processors. Otherwise, IA32_CR_FEATURE_CONTROL is treated as reserved.

The parameter control field for SENTER enables extensibility of the SENTER functionality, and allows specific functionality to be selectively defeatured when executing SENTER. Each bit of the SENTER parameter control field of IA32_CR_FEATURE_CONTROL MSR corresponds to specific functionality of SENTER. When a bit in the parameter control field is set to 1, the corresponding functionality is enabled in SMX. To enable all functionality when executing SENTER, bits 15:8 must be set to FFH for the MSR while the corresponding EDX bits must be 0 (EDX parameter bits associated with SENTER act as disable controls).

The definition for each bit of the SENTER parameter control field and corresponding bits are currently reserved. In future implementations, enumeration of selective functionality will use parameter type 4 in GETSEC[PARAMETERS]. If no selective functionality for SENTER exists the corresponding bits in the IA32_CR_FEATURE_CONTROL MSR bits 14:8 must be programmed to 1 if the SENTER global enable bit 15 is set. These setting allow for the future extensibility of SENTER selective functionality capability. Attempts to program invalid settings to this MSR will result in the signaling of a general protection violation.

Table 10. IA32_FEATURE_CONTROL definition for SENTER control

Bit Position	Description
0	Lock. When set to '1' further writes to this MSR are blocked.
2:1	Enable VMX (see Table 3)
7:3	Reserved
14:8	SENTER parameter function control. Each bit in the field represents an enable control for a corresponding SENTER function.
15	SENTER global enable. Must be set to '1' to enable operation of GETSEC[SENTER]
63:16	Reserved

Flags Affected

All flags are cleared.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SENTER] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.CD = 1 or CR0.NW = 1 or CR0.NE = 0 or CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1.
	If VMX mode is currently active as started with VMXON.
	If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP.
	If a LaGrande Technology-capable chipset is not present.
	If a LaGrande Technology-capable chipset interface to TPM is not detected as present.

	If a protected partition is already active or the processor is already in authenticated code mode.
	If the processor is in SMM.
	If a valid uncorrectable machine check error is logged in IA32_MC[I]_STATUS.
	If the authenticated code base is not on a 4096 byte boundary.
	If the authenticated code size > processor's authenticated code execution area storage capacity.
	If the authenticated code size is not modulo 64.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SENTER] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[SENTER] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SENTER] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SENTER] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

#GP	IF AC code module does not reside in physical address below 2^{32} -1.

64-bit Mode Exceptions

All protected mode exceptions apply.

#GP	IF AC code module does not reside in physical address below 2^{32} -1.

GETSEC[SEXIT] – Exit protected partition session

Opcode	Instruction	Description
0F 37 (EAX=5)	GETSEC[SEXIT]	Exit protected partition session.

Description

The GETSEC[SEXIT] instruction initiates an exit of a protected partition established by GETSEC[SENTER]. The SEXIT leaf of GETSEC is selected with EAX set to 5 at execution. This instruction leaf sends a message to all logical processors in the platform to signal the protected partition exit.

There are restrictions enforced by the processor for the execution of the GETSEC[SEXIT] instruction:

- Execution is not allowed unless the processor is in protected mode (CR0.PE = 1) with CPL = 0 and EFLAGS.VM = 0.
- A LaGrande Technology-capable chipset must be present and communicated to the processor by sampling of the power-on configuration capability field after reset.
- The processor must be in a protected partition as launched by a previous GETSEC[SENTER] instruction, but not still in authenticated code execution mode.
- To avoid potential inter-operability conflicts between modes, the processor is not allowed to execute this instruction if it currently is in SMM or VMX mode.
- To insure consistent handling of SIPI messages, the processor executing the GETSEC[SEXIT] instruction must also be designated the BSP (bootstrap processor) as defined by the register bit in the IA32_APIC_BASE MSR.

Failure to abide by the above conditions results in the processor signaling a general protection violation.

This instruction initiates a sequence to rendezous the RLPs with the ILP. It then clears the internal processor flag indicating the processor is operating in a protected partition.

In response to a message signaling the completion of rendezvous, all RLPs restart execution with the instruction that was to be executed at the time the message (SEXIT) initiated by GETSEC[SEXIT] was recognized. This applies to all processor conditions, with the following exceptions:

- If an RLP executed HLT and was in this halt state at the time of the message initiated by GETSEC[SEXIT], then execution resumes in the halt state.
- If an RLP was executing MWAIT, then a message initiated by GETSEC[SEXIT] causes an exit of the MWAIT state, falling through to the next instruction.
- If an RLP was executing an intermediate iteration of a string instruction, then the processor resumes execution of the string instruction at the point which the message initiated by GETSEC[SEXIT] was recognized.

- If an RLP is still in the SENTER sleep state (never awakened with GETSEC[WAKEUP]), it will be sent to the wait-for-SIPI state after first clearing the bootstrap processor indicator flag (IA32_APIC_BASE.BSP) and any pending SIPI state. In this case, such RLPs are initialized to an architectural state consistent with having taken a soft reset using the INIT# pin.

Prior to completion of the GETSEC[SEXIT] operation, both the ILP and any active RLP unmask the response of the external event signals INIT#, A20M, NMI#, and SMI#. This unmasking is performed unconditionally to unblock recognition if pin event state as potentially left blocked after a GETSEC[SENTER] or GETSEC[ENTERACCS]. The state of A20M is unmasked to re-enable of a following behavior to the external A20M pin state, as the A20M pin is not recognized while the protected partition is active.

On a successful exit of the protected partition, the ILP re-locks the LaGrande Technology-capable chipset private configuration space.

At completion of GETSEC[SEXIT] by the ILP, execution proceeds to the next instruction. Since EFLAGS and the debug register state are not modified by this instruction, a pending trap condition is free to be signaled if previously enabled.

Flags Affected

None for ILP. All for RLP.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1.
	If VMX mode is currently active as started with VMXON.
	If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP.
	If a LaGrande Technology-capable chipset is not present.
	If a protected partition is not currently active or the processor is currently in authenticated code mode.
	If the processor is in SMM.

Real-Address Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[SEXIT] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	If CR4.SMXE = 0.
	If GETSEC[SEXIT] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SEXIT] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-bit Mode Exceptions

All protected mode exceptions apply.

GETSEC[PARAMETERS] – Report the SMX Parameters

Opcode	Instruction	Description
0F 37 (EAX=6)	GETSEC[PARAMETERS]	Report the SMX Parameters The parameters index is input in EBX with the result returned in EAX, EBX, and ECX.

Description

The GETSEC[PARAMETERS] instruction returns specific parameter information for features supported by the processor. Parameter information is returned in EAX, EBX, and ECX, with the input parameter select using EBX.

Software retrieves parameter information by searching with an input index for EBX starting at 0, and then reading the returned results in EAX, EBX, and ECX. EAX[4:0] is designated to return a parameter type field indicating if a parameter is available and what type it is. If EAX[4:0] is returned with 0, this designates a null parameter and indicates no more parameters are available. Only two valid parameter types are supported at this time (more types can readily be reported as future architectures require).

Table 11 defines the parameter types supported in current and future implementations.

Table 11. Supported Reporting Parameters

Parameter Type EAX[4:0]	Parameter Description	EAX[31:5]	EBX[31:0]	ECX[31:0]
0	Null	Reserved (0 returned)	Reserved (unmodified)	Reserved (unmodified)
1	Supported AC module versions	Reserved (0 returned)	Version comparison mask	Version numbers supported
2	Max size of authenticated code execution area	Multiply by 32 bytes	Reserved (unmodified)	Reserved (unmodified)
3	External memory types supported during AC mode	Memory type bit mask	Reserved (unmodified)	Reserved (unmodified)
4	Selective SENTER functionality control	EAX[14:8] correspond to available SENTER function disable controls	Reserved (unmodified)	Reserved (unmodified)
5-31	Undefined	Reserved (unmodified)	Reserved (unmodified)	Reserved (unmodified)

Supported AC module versions (as defined by the HeaderVersion field) can be determined for a particular SMX capable processor by the type 1 parameter. Using EBX to index through the available parameters reported by GETSEC[PARAMETERS] for each unique parameter set returned for type 1, software can determine the complete list of AC module version(s) supported.

For each parameter set, EBX returns the comparison mask and ECX returns the available HeaderVersion field values supported, after AND'ing the target HeaderVersion with the comparison mask. Software can then determine if a particular AC module version is supported by following the pseudo-code search routine given below:

```
parameter_search_index = 0;
DO
        EBX= parameter_search_index++;
        EAX= 6;
        GETSEC;
        IF (EAX[4:0] == 1)
            IF ((version_query & EBX) == ECX)
                version_is_supported = 1;
                BREAK;
            ENDIF
        ENDIF
WHILE (EAX[4:0] != 0);
```

If only AC modules with a HeaderVersion of 0 are supported by the processor, then only one parameter set of type 1 will be returned, as follows: EAX = 00000001H, EBX = FFFFFFFFH, and ECX = 00000000H.

The maximum capacity for an authenticated code execution area supported by the processor is reported with the parameter type of 2. The maximum supported size in bytes is determined by multiplying the returned size in EAX[31:5] by 32. Thus, for a maximum supported authenticated RAM size of 32KBytes, EAX returns with 00008002H.

Supportable memory types for memory mapped outside of the authenticated code execution area are reported with the parameter type of 3. While authenticated code execution mode is active as initiated by the GETSEC functions SENTER or ENTERACCS and terminated by EXITAC, there are restrictions on what memory types are allowed for the rest of system memory. It is the responsibility of the authenticated code to initialize the memory type range register (MTRR) MSRs and/or the page attribute table (PAT) to only map memory types consistent with the reporting of this parameter. The reporting of supportable memory types of external memory is indicated using a bit map returned in EAX[31:8]. These bit positions correspond to the memory type encodings defined for the MTRR MSR and PAT programming. See Table 12.

The parameter type of 4 is used for enumerating the availability of selective GETSEC[SENTER] function disable controls. If a 1 is reported in bits 14:8 of the returned parameter EAX, then this indicates a disable control capability exists with SENTER for a particular function. The enumerated field in bits 14:8 corresponds to use of the EDX input parameter bits 6:0 for SENTER. If an enumerated field bit is set to 1, then the corresponding EDX input parameter bit of EDX may be set to 1 to disable that designated function. If the enumerated field bit is 0 or this parameter is not reported, then no disable capability exists with the corresponding EDX input parameter for SENTER and EDX bit(s) must be cleared to 0 to enable execution of SENTER. If no selective disable capability for SENTER exists as enumerated, then the corresponding bits in the IA32_CR_FEATURE_CONTROL MSR bits 14:8 must also be programmed to 1 if the SENTER global enable bit 15 of the MSR is set. This is required to enable future extensibility of SENTER selective disable capability with respect to potentially separate software initialization of the MSR.

Table 12. External Memory Types Supported Using Parameter 3

EAX bit position	Memory type for external non-AC module memory
8	Uncacheable (UC)
9	Write combining (WC)
11:10	Reserved
12	Write-through (WT)
13	Write-protected (WP)
14	Write-back (WB)
31:15	Reserved

If the GETSEC[PARAMETERS] leaf or specific parameter is not present for a given SMX capable processor, then default parameter values should be assumed. These are defined in Table 13.

Table 13. Default Parameter Values

Parameter type EAX(4:0)	Parameter description	Default setting
1	Supported AC module versions	0.0 only
2	Authenticated code execution area size	32 KBytes
3	External memory types supported during AC mode	UC only
4	Available SENTER function selective disable controls	None

Flags Affected

None.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Real-Address Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Virtual-8086 Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[PARAMETERS] is not reported as supported by GETSEC[CAPABILITIES].

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-bit Mode Exceptions

All protected mode exceptions apply.

GETSEC[SMCTRL] – SMX mode control

Opcode	Instruction	Description
0F 37 (EAX=7)	GETSEC[SMCTRL]	Perform specified SMX mode control as selected with the input EBX.

Description

The GETSEC[SMCTRL] instruction is available for performing certain SMX specific mode control operations. The operation to be performed is selected through the input register EBX. Currently only an input value in EBX of 0 is supported. All other EBX settings will result in the signaling of a general protection violation instead.

If EBX is set to 0, then the SMCTRL leaf is used to re-enable SMI events after they have been previously masked by the GETSEC[SENTER] instruction for the initiating logical processor (ILP) or the LT bus event initiated by GETSEC[SENTER] for responding logical processors (RLPs). The determination of when this instruction is allowed and the events that are unmasked is dependent on the processor context (See Table 14). For brevity, the usage of SMCTRL where EBX=0 will be referred to as GETSEC[SMCTRL(0)].

As part of support for the protected partition, the events SMI, NMI, and INIT are masked after an SENTER operation, remain masked after EXITAC, until unmasked as managed in VMX mode after a subsequent VMX environment has been established. In this case it is not desired to allow events to be unmasked until a secure means for handling events under a virtual machine environment can be established. In this case the VMX monitor may wish to also enable the handling of SMI events with a separate dedicated SMM transfer monitor (STM). This envisioned usage model does not require the use of GETSEC[SMCTRL(0)] as event re-enabling after the VMX environment launch is handled implicitly and through separate VMX based controls. If a dedicated SMM transfer monitor will not be established after VMX mode is enabled, then GETSEC[SMCTRL(0)] can be used to re-enable SMI that has been masked as a result of SENTER.

To define the processor context in which GETSEC[SMCTRL(0)] can be used and which events that will be unmasked, the following table is defined. Note that the events that are unmasked is dependent upon the currently operating processor context.

Table 14. Supported Reporting Parameters

Processor mode	SMCTRL execution action
not post-SENTER	#GP(0), invalid context
in authenticated code execution mode	#GP(0), invalid context
post-SENTER, not in VMX mode, not in SMM	Unmask SMI
VMX mode, SMM root	#GP(0), invalid context
VMX mode, guest	If (GETSEC-exiting=1) then VM exit else #GP(0)
VMX mode, non-SMM root	If SMM transfer monitor is not configured then unmask SMI else #GP(0)

Flags Affected

None

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	IF CR4.SMXE=0.
	If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE=0 or CPL>0 or EFLAGS.VM=1.
	If a protected partition is not currently active or the processor is currently in authenticated code mode.
	If not in VMX mode.
	If the processor is in SMM.
	If the STM is configured.

Real-Address Mode Exceptions

#UD	IF CR4.SMXE=0.
	If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[SMCTRL] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	IF CR4.SMXE=0.
	If GETSEC[SMCTRL] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[SMCTRL] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-bit Mode Exceptions

All protected mode exceptions apply.

GETSEC[WAKEUP] – Wake up sleeping processors in protected partition session

Opcode	Instruction	Description
0F 37 (EAX=8)	GETSEC[WAKEUP]	Signals a wake-up bus message to all processors in the SENTER sleep state.

Description

The GETSEC[WAKEUP] instruction broadcasts a wake-up message to all logical processors currently in the SENTER sleep state. Responding logical processors (RLPs) enter the SENTER sleep state after completion of a rendezvous sequence. They do this in response to the execution of GETSEC[SENTER] from the initiating logical processor (ILP).

The GETSEC[WAKEUP] instruction may only be executed:

1. in a protected partition session as initiated by execution of GETSEC[SENTER],
2. outside of authenticated code execution mode.
3. in protected mode at privilege level 0, while not in SMM or VMX operation.
4. In addition, the logical processor must be designated as the boot-strap processor as configured by setting IA32_APIC_BASE.BSP = 1.

If these conditions are not met, attempts to execute GETSEC[WAKEUP] result in a general protection violation.

In response to wake up signaling, processors in the SENTER sleep state vector to the entry point defined by the JOIN data structure pointed to by LT.MVMM.JOIN. This register is publicly writable in the chipset by all processors and is not restricted by LaGrande Technology configuration register lock status. The format of the JOIN data structure is defined in Table 7.

The JOIN based data structure contains the information necessary initialize RLP processor state so that the processor may join a protected partition. The GDTR, EIP, and CS, DS, SS, and ES selector values are initialized from the contents in the data structure. The CS selector index is derived directly from the segment selector initializer field; while DS, SS, and ES selectors are initialized to CS+8. The segment descriptor fields are initialized implicitly with BASE = 0, LIMIT = FFFFFH, G = 1, D = 1, P = 1, S =1; read/write/accessed for DS, SS, and ES, while execute/read/accessed for CS.

It is the responsibility of external software to establish a GDT pointed to by the JOIN data structure. The GDT must contain descriptor entries consistent with the implicit settings initialized by the processor. Certain states held in Table 7 are consistency checked by the processor prior to execution. A failure of a consistency check results in the RLP aborting entry to the protected partition and the signaling of a LaGrande Technology shutdown. The checks performed are documented in the SENTER operation description. After successful completion of processor consistency checks and subsequent initialization, execution in the protected partition for the RLP begins from the defined linear IP entry point at offset 12, as indicated in Table 7.

Flags Affected

None.

Use of Prefixes

REP, REPNE	Causes #UD
Operand size	Causes #UD
Lock	Causes #UD
REX	Ignored
All others	Ignored

Protected Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	If CR0.PE = 0 or CPL > 0 or EFLAGS.VM = 1.
	If VMX mode is currently active as started with VMXON.
	If a protected partition is not currently active or the processor is currently in authenticated code mode.
	If the processor is in SMM.
	If the initiating processor is not designated as the bootstrap processor via the MSR bit IA32_APIC_BASE.BSP.
	If a LaGrande Technology-capable chipset is not present.

Real-Address Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP	GETSEC[WAKEUP] is not recognized in real-address mode.

Virtual-8086 Mode Exceptions

#UD	IF CR4.SMXE = 0.
	If GETSEC[WAKEUP] is not reported as supported by GETSEC[CAPABILITIES].
#GP(0)	GETSEC[WAKEUP] is not recognized in virtual-8086 mode.

Compatibility Mode Exceptions

All protected mode exceptions apply.

64-bit Mode Exceptions

All protected mode exceptions apply.

§

3 LaGrande Technology Shut-Down

3.1 Reset Conditions

When a LaGrande Technology shutdown condition occurs, the processor writes an error code indicating the reason for the failure to the LT.ERRORCODE register. It then writes to the LT.CMD.SYS-RESET command register, initiating the message that causes a platform reset. After the write to LT.CMD.SYS-RESET; the processor enters a shutdown sleep state with all external pin events, bus or error events, machine check signaling, and MONITOR/MWAIT event signaling masked. Only the assertion of reset back to the processor takes it out of this sleep state. The LaGrande Technology error code register is not cleared by the platform reset; the error code should be accessible after shutdown has occurred for subsequent diagnostics.

LT-shutdown can be generated by the processor as part of execution of certain GETSEC leaf functions (for example: ENTERACCS, EXITAC, SENTER, SEXIT), where recovery from an error condition is not considered reliable. This situation should be interpreted as an abort of the authenticated execution or protected environment launch.

A legacy IA-32 triple-fault shutdown condition is also converted to a LaGrande Technology shutdown sequence if the triple-fault shutdown occurs during authenticated code execution mode or while the protected environment is active. The same is true for other legacy non-SMX specific fault shutdown error conditions. Legacy shutdown to LaGrande Technology shutdown conversions are defined as the mode of operation between:

- Execution of the GETSEC functions ENTERACCS and EXITAC
- Recognition of the message signaling the beginning of the processor rendezvous after GETSEC[SENTER] and the message signaling the completion of the processor rendezvous

There is a special case. If the processor is in VMX operation while the protected environment is active, a triple-fault shutdown condition that causes a guest exiting event back to the Virtual Machine Monitor (VMM) supersedes conversion to the LaGrande Technology shutdown sequence. In this situation, the VMM remains in control after the error condition that occurred at the guest level and there is no need to abort processor execution.

Given the above situation; if the triple-fault shutdown occurs at the root level of the VMM or a VMX abort is detected, then a LaGrande Technology shutdown sequence is signaled. For more details on a VMX abort, see Chapter 23, “VM Exits,” in the *IA-32 Intel® Architecture Software Developer’s Manual, Volume 3B*.

3.2 ERRORCODE Register

Table 15 lists the format of the LT.ERRORCODE register. The processor uses this format when reporting of the error code in a LaGrande Technology shutdown sequence. The processor always reports the error code in bits 0:15, with bit 31 set to 1 to indicate a valid error code. Bit 30 is cleared to 0, indicating that the error is reported by the processor.

Table 15. LT.ERRORCODE Register Bit Format

Bit	Name	Description
0:15	Type	This is implementation and source specific. Provides details on what the step was being performed when a failure condition was detected.
16:29	Reserved	Reserved. Must be written with zeros.
30	Processor/ External	0 = Error condition reported by processor. 1 = Error condition reported by external software.
31	Valid/Invalid	0 = Register content invalid. 1 = Valid error.

Table 16 defines LaGrande Technology shutdown error types reported by hardware. Some error types are only relevant to certain leafs, based on the operations performed.

For LaGrande Technology shutdown conditions induced by external software, the external software (the AC module or MVMM) must write directly to the LT.ERRORCODE and LT.CMD.SYS-RESET registers. Bit 30 of the LT.ERRORCODE register must be written with a 1. Bits 0:15 should be interpreted as software defined and do not need to comply with the definitions in Table 16.

Table 16.Type Field Encodings for Processor-Initiated LaGrande Technology Shutdowns

Type	Error condition	Mnemonic
0	Legacy shutdown	#LegacyShutdown
1-4	Reserved	Reserved
5	Load memory type error in Authenticated Code Execution Area	#BadACMMType
6	Unrecognized AC module format	#UnsupportedACM
7	Failure to authenticate	#AuthenticateFail
8	Invalid AC module format	#BadACMFormat
9	Unexpected snoop hit detected	#UnexpectedHITM
10	Invalid event	#InvalidEvent
11	Invalid JOIN format	#BadJOINFormat
12	Unrecoverable machine check condition	#UnrecovMCError

Type	Error condition	Mnemonic
13	VMX abort	#VMXAbort
14	Authenticated Code Execution Area corruption	#ACMCorrupt
15	Invalid voltage/bus ratio	#InvalidVIDBRatio
16 – 65535	Reserved	

§

4 DMA Page Protection

This chapter describes the core logic chipset feature that protects data in the memory sub-system of a platform, as memory can be accessed by both the processor(s) and chipset.

4.1 Overview of DMA Page Protection

The processor, using its own protection tables, is responsible for protecting pages in memory from unauthorized access by software. The chipset is responsible for protecting some pages of memory from access by bus master devices. Pages designated as protected must not be read or written by bus masters.

The pages in memory to be protected are indicated in the Memory Protection Table (MPT). This table will include an indication for each page in physical memory supported by the platform.

Theoretically, the memory controller must check the MPT table for each read or write by a bus master. Such an extra check would add significant load latency on the memory I/F and degrade performance. To prevent degradation, the chipset may employ caching mechanisms to reduce the number of memory read cycles required to check MPT table entries. The chipset may include a small cache that tracks pages that have been recently accessed. If I/O transactions by bus masters are linear, then the cache will have a high hit rate.

The exact number of entries required for the cache is specific to the chipset and beyond this specification’s scope. Software is required to assist with the hardware caching by issuing commands to the chipset changing entries in the MPT table. See Requirements for Software support of MPT for details on the software requirements for managing the cache.

The chipset must also protect the MPT. This is necessary because a DMA device could write invalid entries to the MPT when the MPT is initialized. To prevent this type of attack, the chipset must not all bus master writes to the MPT once the LaGrande Technology launch process locks the table.

4.2 Requirements for Software support of MPT

The authenticated code module must load the MPT table initially protecting no physical addresses. As the authenticated code module loads the MPT and loads and measures the VMM, the MPT must ensure that each physical page is appropriately indicated in the MPT.

§

Appendix A Authenticated-Code Module

A.1 Authenticated-Code Module Format

The format of the authenticated-code module is in Table 17. This definition represents Revision 0.0 of the AC module header version (defined in the HeaderVersion field).

Table 17. Authenticated Code Module Format

Field	Offset	Size	Description
ModuleType	0	4	Module type
HeaderLen	4	4	Header length (dwords) (fixed to 161 for version 0.0)
HeaderVersion	8	4	Module format version
ModuleID	12	4	Module release identifier
ModuleVendor	16	4	Module vendor identifier
Date	20	4	Creation date (BCD format: year.month.day)
Size	24	4	Module size (dwords)
Reserved1	28	4	Reserved for future extensions
CodeControl	32	4	Authenticated code control flags
ErrorEntryPoint	36	4	Error response entry point offset (bytes)
GDTLimit	40	4	GDT limit (defines last byte of GDT)
GDTBasePtr	44	4	GDT base pointer offset (bytes)
SegSel	48	4	Segment selector initializer
EntryPoint	52	4	Authenticated code entry point offset (bytes)
Reserved2	56	64	Reserved for future extensions
KeySize	120	4	Module public key size less the exponent (dwords) (KeySize = 64 for HeaderVersion = 0.0

Field	Offset	Size	Description
ScratchSize	124	4	Scratch field size (dwords) (ScratchSize = 2 * KeySize + 15 for HeaderVersion = 0.0)
RSAPubKey	128	KeySize * 4 + 4	Module public key
RSASig	388	256	PKCS #1.5 RSA Signature.
Scratch	644	ScratchSize * 4	Internal scratch area used during initialization (needs to be all 0s)
User Area	1216	N * 64	User code/data (modulo-64 byte increments)

- ModuleType

  Indicates the module type. The following module types are defined:

  2 = Chipset authenticated code module.

  Only ModuleType 2 is supported by GETSEC functions SENTER and ENTERACCS.
- HeaderLen

  Length of the authenticated module header specified in 32-bit quantities. The header spans the beginning of the module to the end of the signature field. This is fixed to 161 for loader version 0.0.
- HeaderVersion

  Specifies the AC module header version. A major and minor vendor field are specified, with bits 15:0 holding the minor value and bits 31:16 holding the major value. This should be initialized to zero for header version 0.0. Unsupported header versions will be rejected by the processor and result in an abort during authentication.
- ModuleID

  Module specific identifier.
- ModuleVendor

  Module creator vendor ID. Use the PCI SIG assignment for vendor IDs to define this field. The following vendor ID is currently recognized:

  00008086H = Intel
- Date

  Creation date of the module. Encode this entry in the BCD format as follows: yr.mo.day with two bytes for the year, one byte for the day, and one byte for the month. For example, a value of 20040328H indicates module creation on March 28, 2004.

- Size

  Total size of module in dwords. This includes the header, scratch area, user code and data.

- Reserved1

  Reserved. This should be initialized to zeros.

- CodeControl

  Authenticated code control word. Defines specific actions or properties for the authenticated code module. The following bits are currently defined:

  - 0 = Valid error entry point defined.
  - 1 = Enable error reporting on detection of a snoop hit to a modified line during the load of an authenticated code module.
  - 2 = Reserved. Setting of reserved bits may result in a LaGrande Technology shutdown condition.
  - 3 = Enable error reporting on detection of a snoop hit to a modified line during authenticated code execution. If this bit is set, the occurrence of a HITM event results in an LaGrande Technology shutdown condition.
  - 4:31 = Reserved. Initialize to 0s. Failure to do so may result in an abort of the authentication process and signaling of a LaGrande Technology shutdown condition.

- ErrorEntryPoint

  Error entry point. If bit 0 of the CodeControl word is 1, the processor will vector to this location if a snoop hit to a modified line was detected during the load of an authenticated code module. If bit 0 is 0, then enabled error reporting via bit 1 of a HITM during ACRAM load will result in an abort of the authentication process and signaling of a LaGrande Technology shutdown condition.

- GDTLimit

  Limit of the GDT in bytes, pointed to by GDTBasePtr. This is loaded into the limit field of the GDTR upon successful authentication of the code module.

- GDTBasePtr

  Pointer to the GDT base. This is an offset from the authenticated code module base address.

- SegSel

  Segment selector for initializing CS, DS, SS, and ES of the processor after successful authentication. CS is initialized to SegSel while DS, SS, and ES are initialized to SegSel + 8.

- EntryPoint

  Entry point into the authenticated code module. This is an offset from the module base address. The processor begins execution from this point after successful authentication.

- Reserved2

  Reserved. Should contain zeros.

- KeySize

  Defines the width the RSA public key in dwords applied for authentication, less the size of the exponent. For version 0.0 of the AC module header, KeySize is defined to 64 (a 2048 bit key). The information in this field is intended to support external software parsing of an AC module independent of the module version. It is the responsibility of the developer to reflect an accurate KeySize. This field is not checked for consistency by the processor.

- ScratchSize

  Defines the width of the scratch field size in dwords. For version 0.0 of the AC module header, ScratchSize is defined by KeySize * 2 + 15. The information in this field is intended to support external software parsing of an AC module independent of the module version. It is the responsibility of software to reflect an accurate ScratchSize. This field is not checked by the processor.

- RSAPubKey

  Contains a public key plus a fixed 32-bit exponent to be used for decrypting the signature of the module. The size of this field is defined by the previously defined AC module field, KeySize + 1 dwords.

- RSASig

  The PKCS #1.5 RSA Signature of the module. The RSA Signature signs an area that includes the some of the module header and the USER AREA data field (which represents the body of the module). Parts of the module header not included are: the RSA Signature, public key, and scratch field.

- Scratch

  Used for temporary scratch storage by the processor during authentication. This area can be used by the user code during execution for data storage needs. The area must be initialized to zero before being loaded by ENTERACCS or SENTER.

  After successful authentication of an AC module, the first 20 bytes of the scratch area (offset bytes 644 - 663) contains the computed hash of the module as represented by the encrypted version held in RSASig field. The contents for other locations of the scratch field after authentication are undefined and should not be relied upon by AC module.

- User Area

  User code and data, represented in modulo-64 byte increments. In addition, the boundary between data and code should be on at least modulo-1024 byte intervals. The user code and data region is allocated from the first byte after the end of the Scratch field to the end of the AC module.

A.1.1 Authenticated code module restrictions

Operation in authenticated code execution mode has restrictions. This mode is defined as being active from the first instruction executed after launch of the authenticated code module with the GETSEC leafs SENTER or ENTERACCS, until exiting of authenticated mode upon completion of GETSEC[EXITAC]. Not all restrictions are enforced by the processor.

It is up to the authenticated code module developer to avoid instructions or operations specified here to prevent conflicts that may interfere with processor operation or the integrity of the isolated execution environment.

A.1.2 Bus snoop restrictions

During the load of an authenticated code module or while authenticated execution is active, processor response to bus snoop hits to modified lines is changed from the legacy processor architecture operation. The behavior in this regard is controlled by the CodeControl field of the AC module header defined in Section A.1 This modification only applies to HITM assertions (bus snoop hit of a modified cache line) by another physical processor for a bus transaction initiated for one's own processor.

Three distinct actions are supported by the processor in response to an HITM event during the loading of the AC module:

1. The processor behaves normally with no change from legacy functionality. This condition is established if bit 1of the CodeControl word is 0.
2. The processor begins execution entry in the AC module after loading and successful authentication using the AC module header defined ErrorEntryPoint field. This condition is enabled if the CodeControl word bit 0 is 1, and bit 1 is 1.
3. The processor performs a LaGrande Technology shutdown action. This condition is enabled if the CodeControl word bit 0 is 0 and bit 1 is 1, implying no valid ErrorEntryPoint exists, yet HITM error reporting is enabled during an AC module load.

Once an AC module has been loaded and successfully authenticated, the code module is capable of performing bus transactions to addresses outside of the AC module address range. For external memory transactions that result in a HITM response from another physical processor, two response options are supported:

1. The processor behaves normally with no change from legacy functionality. This condition is established if bit 3 of the CodeControl word is cleared to 0.
2. The processor performs a LaGrande Technology shutdown action. This condition is enabled if the CodeControl word bit 3 is set to 1.

In the case of a signaled LaGrande Technology shutdown in response to a HITM event, this will take place on the next instruction boundary from when the event is detected on the external bus. Given the speculative nature of the processor execution and pipeline depth, an indeterminate delay may exist from the instruction that has directly or indirectly invoked the bus transaction being snooped to the instruction boundary where the event is recognized.

A.1.3 Memory type cacheability restrictions

Prior to launching the authenticated execution environment using the GETSEC leaf functions ENTERACCS or SENTER, processor MTRRs (Memory Type Range Registers) must first be initialized to map out the authenticated RAM addresses as WB (write-back). Failure to do so may affect the ability for the processor to maintain isolation of the loaded authenticated code module. The processor may signal a LaGrande Technology shutdown condition with error code #BadACMMType during the loading of the authenticated code module if non-WB memory is detected.

Once the authenticated code module has been successfully loaded and executed, it is also the responsibility of the authenticated code module developer to properly map the memory type for all physical address references, including those outside of the module boundaries. While physical addresses within the load module must be mapped as WB, the memory type for locations outside of the module boundaries must be mapped to the more restrictive UC (uncacheable) setting by default. The default setting of UC for non-AC module related addresses is required to support inter-operability across SMX capable processor implementations.

Processor support for other memory types is enumerated through the GETSEC leaf function PARAMETERS (index 6) with the parameter type 3. The supportable memory types are reported through this parameter if available. If this parameter is unavailable for a given SMX capable processor, than only the UC memory type for non-AC module addresses may be assumed.

Given the restrictions on the use of cache disable control operations and flushing instructions in authenticated code execution mode, care must be taken in respect to how memory type settings are changed from within this mode. This is necessary to avoid potential cache coherency issues that could come about due to aliasing of different memory types.

After entering authenticated code execution mode, the AC module should first determine the current memory type settings and determine their conformance to the requirements just given. This may also involve the use of the GETSEC[PARAMETERS] function to verify the memory types mapped for non-AC module addresses is also in compliance. Out-of-compliance memory type settings are best corrected by proper initialization of the MTRRs ahead of the AC module launch, so that such settings do not have to be reprogrammed from within the more restrictive AC module environment. It is recommended that if an out-of-compliance memory type configuration is detected by the AC module, that it should treat such a condition as an error and abort further execution.

§

Appendix B Appendix B: SMX Interaction with Platform

B.1 LaGrande Technology Configuration Registers

LaGrande Technology configuration registers are a subset of chipset registers. These registers are mapped into two regions of memory, representing the public and private configuration spaces. Registers in the private space can only be accessed after a protected environment has been established and before the LT.CMD.CLOSE-PRIVATE command has been given. The private space registers are mapped to the address range starting at FED20000H. The public space registers are mapped to the address range starting at FED30000H. All registers are defined as 64 bits and return 0's for the unimplemented bits. The offsets in the table are from the start of either the public or private spaces. See Table 18.

Table 18. Configuration Registers Relevant to MVMM

Offset	Name	Description
000H	LT.STS	This is the general status register. This read-only register is used by AC modules and the MVMM to get the status of various LT features. The bits in LT.STS are defined in Table 19. Public: RO Private: RO
030H	LT.ERRORCODE	Holds the LaGrande Technology shutdown error code. The encoding for this is documented in Table 16. A system reset does not clear the contents of this register. Public: RO Private: RW
038H	LT.CMD.SYS-RESET	A write to this register causes a system reset. This is performed by the processor as part of a LaGrande Technology shutdown, after writing to the LT.ERRORCODE register. Public: - Private: WO

Offset	Name	Description
040H	LT.CMD.OPEN-PRIVATE	A write to this register causes the LaGrande Technology-capable chipset private configuration space to be unlocked. Once unlocked, conventional memory read/write operations can be used to access these registers. Public: - Private: WO
048H	LT.CMD.CLOSE-PRIVATE	A write to this register causes the LaGrande Technology-capable chipset private configuration space to be locked. Once locked, conventional memory read/write operations can no longer be used to access these registers. Public: - Private: WO
258H	LT.CMD.FLUSH-WB	Writing to this register flushes the chipset write buffers. The MVMM writes to this register as part of the NoDma table update sequence. Public: - Private: WO
260H	LT.NODMA.BASE	This register contains the physical base address of the NoDma table. The value in this register is set by BIOS. Public: RW Private: RW
268H	LT.NODMA.SIZE	This register contains a value which indicates the size of the NoDma Table. The values are 0=128K, 1=256K, 2=512K, etc. Public: RO Private: RO
270H	LT.SINIT.BASE	This register contains the physical base address of the memory region set aside by the BIOS for loading an SINIT AC module. The guest OS reads this register to locate the SINIT module (which may have been loaded by the BIOS) or to find a location to load the SINIT module. Public: RW Private: RW
278H	LT.SINIT.SIZE	This register contains the size in bytes of the memory region set aside by the BIOS for loading an SINIT AC module. This register is initialized by the BIOS. The MVMM may read this register when loading an SINIT module. Public: RW Private: RW

Offset	Name	Description
290H	LT.MVMM.JOIN	Holds a physical address pointer to the base of the join data structure referenced by RLPs in response to a SIPI while operating between SENTER and SEXIT. Public: RW Private: RW
300H	LT.HEAP.BASE	This register contains the size in bytes of the LT Heap memory region. The BIOS initializes this registers. The guest OS and the MVMM read this register to determine the LT heap size. Public: RW Private: RW
308H	LT.HEAP.SIZE	This register contains the size in bytes of the LT Heap memory region. The BIOS initializes this register. The system software and the MVMM read this register to determine the LT Heap size. Public: RW Private: RW
8E0H	LT.CMD.SECRETS	Writing to this register indicates to the chipset that there are secrets in memory. The chipset tracks this fact with a sticky bit. If the platform reboots with this sticky bit set the SCLEAN AC module will scrub memory. The chipset also uses this bit to detect invalid sleep state transitions. If software tries to transition to S3 or S4 while secrets are in memory then the chipset will reset the system. The MVMM issues the LT.CMD.SECRETS prior to placing secrets in memory for the first time. Software should read the LT.STS register after issuing this command. The read of the LT.STS register ensures that any successive chipset accesses will occur with the secrets bit set. Public: - Private: WO
8E8H	LT.CMD.NO-SECRETS	Writing to this register indicates there are no secrets in memory. The MVMM will write to this register after removing all secrets from memory as part of the LT teardown process. Software should read the LT.STS register after issuing this command. The read of the LT.STS register ensures any subsequent chipset accesses dependent on the state of the SECRETS bit actually observe the secrets bit clear. Public: - Private: WO

Offset	Name	Description
8F0H	LT.E2STS	This register is used to read the status associated with various errors that might be detected. The bits in this register are only valid if the LT.WAKE-ERROR.STS bit is set in the LT.ESTS register. Public: RO Private: RW

Table 19. LT.STS Bit Definitions

Bit	Access	Name	Comment
11	RO	LT.MEM-CONFIG-OK.STS	This bit indicates whether the chipset has received and accepted the LT.CMD.MEM-CONFIG-CHECKED LT command. This bit is cleared by PCI reset or by the LT.CMD.UNLOCK-MEM-CONFIG command. 0: Indicates that memory configuration checking has not been performed. 1: Indicates that memory configuration checking has been performed. This bit is set to one when the chipset accepts the LT.CMD.MEM-CONFIG-CHECKED LT command.
10	RO	LT.NODMA-TABLE-PROTECT.STS	Will be set to 1 when the LT.CMD.NODMA-TABLE-PROTECT.EN is performed. Cleared by LT.CMD.NODMA-TABLE-PROTECT.DIS or by a system reset.
9	RO	LT.NODMA-CACHE.STS	Will be set to 1 when the LT.CMD.NODMA-CACHE.EN is performed. This bit is cleared by LT.CMD.NODMA-CACHE.DIS or by a system reset.
8	RO	Reserved	Reserved
7	RO	LT.PRIVATE-OPEN.STS	This bit will be set to 1 when the LT.CMD.OPEN-PRIVATE is performed. This bit cleared by the LT.CMD.CLOSE-PRIVATE or by a system reset.
6	RO	LT.MEM-CONFIG-LOCK.STS	This bit will be set to 1 when the memory configuration has been locked. This bit is cleared by LT.CMD.UNLOCK.MEMCONFIG or by a system reset.
5	RO	LT.NODMA-EN.STS	This bit will be set to 1 when the NoDMA table is protecting main memory from bus master access. This bit is cleared by the LT.CMD.NODMA.DIS or by a system reset.

Bit	Access	Name	Comment
4	RO	LT.MEM. UNLOCK.STS	This bit will be set to 1 when the memory has been unlocked using the LT.CMD.UNLOCK-MEMORY command or after a normal reset when no LT environment was in place prior to the reset. When this bit is '1', memory may be accessed by CPU cycles. This bit will be cleared after a reset when there might have been secrets in memory before the reset. This bit must be set if a read to 0xFED4_0000 returns a '1' in bit 0.
3:2	RO	reserved (0)	
1	RO	SEXIT.DONE. STS	This bit is set when all of the bits in the LT.THREADS.JOIN register are clear 0. Thus, this bit will be set immediately after reset (since the bits are all 0). Once all threads have done the LT.CYC.SEXIT-ACK, the LT.THREAD.JOIN register will be 0, so the chipset will set this bit.
0	RO	SENTER.DONE. STS	The chipset sets this bit when it sees all of the threads have done the LT.CYC.SENTER-ACK. When any of the threads does the LT.CYC.SEXIT-ACK the LT.THREADS.JOIN and LT.THREADS.EXISTS registers will not be equal, so the chipset will clear this bit.

A processor issued hash operation to TPM locality 4 using a write to the TPM.HASH.START port causes an implicit opening of locality 4 for processor hardware based accesses. This stays in effect until the issuing of a subsequent TPM.HASH.END to terminate the hash data write sequence and update the contents of PCR17. No status read check of the TPM is performed by the processor GETSEC[SENTER] instruction ahead of the TPM.HASH write sequence. If the TPM is not in acquiesced state at this time, then the PCR17-20 reset and hash registration to PCR17 may not succeed. To insure reliable system software functionality for TPM support, it is recommended that the GETSEC[SENTER] instruction only be executed once the TPM has acquiesced and ownership has been established in the context of the SENTER initiating process.

B.2 Platform Configuration Registers

The TPM contains Platform Configuration Registers (PCR). The purpose of a PCR is to contain measurements. From a TPM standpoint, the TPM does not care what entity uses a PCR to store a measurement.

The TPM provides two types of PCRs, static and dynamic. Static PCRs only reset on system reset; dynamic PCRs reset upon request. Static PCRs are in use by the static root of trust for measurement (SRTM). In the PC, the RTM is the BIOS boot block. The dynamic PCRs are in use by the dynamic root of trust for measurement (DRTM). In the PC, the DRTM is the process initiated by GETSEC[SENTER].

A PC TPM requires 24 PCRs. The first 16 are static PCRs and the last eight are dynamic PCRs. LaGrande Technology uses four of the dynamic PCRs to keep track of the MVMM environment. The current mapping identifies PCR17, 18, 19, and 20 as the LaGrande Technology PCRs.

All PCRs, static or dynamic, have the same size and same updating mechanism. The size is 160 bits. This size allows the PCRs to contain a hash digest value. Storing a measurement value in the PCRs involves a TPM_Extend operation.

B.3 LT Device Space

There are several memory ranges within LT address space provided to access LT related devices. The first range is 0xFED4_xxxx which is divided up into 16 pages. Each page in the FED4 range has specific access attributes. A page in this region may be accessed by LT cycles only, by LT cycles and via private space, or by LT cycles, private and public space.

Address Range	Device Port
FED4 0xxxh	TPM, Locality 0 (fully public)
FED4 1xxxh	TPM, Locality 1 (reserved for future use)
FED4 2xxxh	TPM, Locality 2 (MVMM access only)
FED4 3xxxh	TPM, Locality 3 (AC modules access only)
FED4 4xxxh	TPM, Locality 4 (Hardware or microcode access only)
All others	Reserved

The first five pages of the 0xFED4_xxxx region are used for TPM access. Each page represents a different 'locality' to the TPM. Locality is an attribute used by the TPM to define how it treats certain transactions. Locality is defined by the address range used for commands sent to the TPM. All Intel chipsets must support all localities. Localities 0 and 6 are considered public and accesses to these localities are accepted by the chipset under all circumstances. Accesses to locality 0 and 6 are sent to the ICH even if LT is disabled, there has been no SENTER, or Private space is closed. Localities 4 and 5 are never open, but may only be accessed with LT cycles on the FSB. Localities 2 and 7 through 15 are always open from a locality perspective, but are in Private space so that LT.CMD.OPEN-PRIVATE must have been done for the cycles to be sent to ICH as LT cycles. There are LT commands that will OPEN localities 1 and 3. Localities 1-3 and 7-F require that both LocalityX.OPEN (locality 2, 7-F are always OPEN) and LT.CMD.OPEN-PRIVATE be done before allowing accesses in that range to be accepted. At reset, localities 1 and 3 are closed.

The MVMM must prevent guest access to certain localities by preventing the guests from creating virtual to physical mappings to these localities. Specifically the MVMM must prevent guest access to locality 2. The hardware will prevent guest accesses to localities 4 and 5 so the MVMM does not need to prevent guest mappings to these regions. The MVMM must also prevent guest accesses to the trusted device ranges, 0xFED4Bxxx and 0xFED4Cxxx.

Table 20. Locality 4 TPM Device Space

Address	Name	Description
FED44020H	TPM.HASH.END	Signals the end of a hash data write sequence to the TPM input buffer written using the TPM.HASH.DATA port. A hash is computed of the contents in the TPM input buffer and updated in PCR17.
FED44024H	TPM.HASH.DATA	Data is written to this port that the TPM is to hash.
FED44028H	TPM.HASH.START	Signals the start of a new hash data write sequence to the TPM written via the TPM.HASH.DATA port. This also causes an implicit reset of PCR17-20 and clearing of the TPM input buffer.

§

Appendix C Appendix C: LT Heap Memory

LT Heap memory is a region of physically contiguous memory which is set aside by firmware for the use of LT hardware and software. The system software that launches the protected environment passes data to both the SINIT module and the launched monitor using LT Heap memory. The system software is responsible for filling in the table contents prior to executing the SENTER instruction. An incorrect format or incorrect content of this table or tables described by this table will result in failure to launch the protected environment.

Table 21. LT Heap

Offset	Length	Name	Description
0x0	0x8	BiosOsDataSize	Size in bytes of the LT specific data passed from the BIOS to the OS for the purposes of launching OS the MVMM. This size includes the number of bytes for this field, so this field cannot be less than a value of 8. Note 1.
0x8	BiosOsDataSize-0x8	BiosOsData	BIOS specific data. This data is not defined in this document. The format of this data must follow that prescribed in the SINIT-AC module specification.
BiosOsDataSize	0x8	OsMvmmDataSize	Size in bytes of the data passed from the launching OS to the MVMM. This size includes the number of bytes for this field, so this field cannot be less than a value of 8. Note 1.
BiosOsDataSize + 0x8	OsMvmmData Size – 0x8	OsMvmmData	OS specific data. Format of data in this field is considered OS vendor specific. Note 2.
BiosOsDataSize + OsMvmmDataSize	0x8	OsSinitDataSize	Size in bytes of the data passed from the launching OS to the SINIT AC module. Note 1.
BiosOsDataSize + OsMvmmDataSize + 0x8	OsSinitDataSize - 0x8	OsSinitData	Os data passed to the SINIT AC module. The format of this data must follow that prescribed in the SINIT-AC module specification.

Offset	Length	Name	Description
BiosOsDataSize + OsMvmmDataSize + OsSinitDataSize	0x8	SinitMvmmDataSize	Size in bytes of the data passed from the launched SINIT AC module to the Mvmm. Note 1.
BiosOsDataSize + OsMvmmDataSize + OsSinitDataSize + 0x8	SinitMvmmDataSize - 0x8	SinitMvmmData	SINIT data passed to the Mvmm. The format of this data must follow that prescribed in the SINIT-AC module specification.

NOTES:

1. For proper data alignment on 64bit processor architectures this field must be a multiple of 8 bytes. OsMvmmDataSize + OsSinitDataSize + SinitMvmmDataSize must be less than or equal to LT.HEAP.SIZE.

C.1 BIOS to OS Data Format

The format of the data passed from the BIOS to the OS for the purposes of launching the protected environment is shown in Table 22. There are currently two fields defined in this area. The first field is a version field. This second field, BiosSinitSize, is used to communicate the size of a SINIT module passed from the BIOS to the OS in the SINIT memory range described by LT.SINIT.BASE/LT.SINIT.SIZE. A value of 0 in BiosSinitSize indicates the BIOS has not placed an SINIT module in the SINIT memory range.

Table 22. BIOS to OS Data Table

Offset	Length	Name	Description
0x0	0x4	Version	Version number of the BIOS to OS data. Current value is 0x1. This field is incremented for any change to the definition of the BiosOsData. The BiosOsData is always backwards compatible with previous versions of BiosOsData.
0x4	0x4	BiosSinitSize	This field indicates the size of the SINIT AC module stored in system firmware. A value of 0 indicates the system firmware is not providing a SINIT module for OS use.

C.2 OS to MVMM Data Format

Each OS vendor may have a different format for this data, and any MVMM being launched by an OS must understand the format of this OS's handoff data.

C.3 OS to SINIT Data Format

Table 23 defines the format of the data passed from the launching system software (possibly an OS) to SINIT in the OsSinitData field.

Table 23. OS to SINIT Data Table

Offset	Length	Name	Description
0x0	0x4	Version	Version number of the Os to SINIT data. Current value is 0x1. This field is incremented for any change to the definition of the OsSinitData. The OsSinitData is always backwards compatible with previous versions of OsSinitData.
0x4	0x4	Reserved	Reserved for future use
0x8	0x4	MVMM PageTableBaseLow	Physical address of MVMM page table (the MVMM page directory address) – low 32 bits
0xC	0x4	MVMM PageTableBaseHigh	Physical address of MVMM page table (the MVMM page directory address) – high 32 bits
0x10	0x4	MVMM SizeLow	Size in bytes of the MVMM image – low 32 bits
0x14	0x4	MVMM SizeHigh	Size in bytes of the MVMM image – high 32 bits
0x18	0x4	MVMM HeaderBaseLow	Linear address of MVMM header (linear address within the MVMM page tables) – low 32 bits
0x1C	0x4	MVMM HeaderBaseHigh	Linear address of MVMM header (linear address within the MVMM page tables) – high 32 bits

C.4 SINIT to MVMM Data Format

Table 24 defines the format of the SINIT data presented to the MVMM.

Table 24. SINIT to MVMM Data Table

Offset	Length	Name	Description
0x0	0x4	Version	Version number of the SINIT to MVMM data. Current value is 0x1. This field is incremented for any change to the definition of the SinitMvmmData. The SinitMvmmData is always backwards compatible with previous versions of the SinitMvmmData.
0x4	0x4	NumberOfSinit MDRs	Number of SINIT Memory Descriptor Records that follow this field.
0x8	0x18 * NumberOf SinitMDRs	SinitMDR	Array of SINIT Memory Descriptor Records as defined below. Each record describes a memory region as defined by the SINIT AC module.
0x8+0x18*Num berOfSinitMDRs	0x14	Reserved	Reserved
Opt-in MSR_Hash + 0x14	0x14	EDX_Hash	Zero extended SENTER control flags
EDX_Hash + 0x14	0x14	MSEG_Valid	Zero extended MSEG.Valid bit value
MSEG_Valid_Has h + 0x08	0x14	SINIT_Hash	SHA-1 hash of SINIT-AC
SINIT_Hash + 0x14	0x14	SVMM_Hash	SHA-1 hash of SVMM
SVMM_Hash + 0x14	0x14	STM_Hash	SHA-1 hash of STM. This is only valid if MSEG_Valid.bit = 1.

Table 255. SINIT Memory Descriptor Record

Offset	Length	Name	Description
0x0	0x4	AddressLow	Low 32 bits of a 64 bit physical address of the memory range described in this record.
0x4	0x4	AddressHigh	High 32 bits of a 64 bit physical address of the memory range described in this record.
0x8	0x4	LengthLow	Low 32 bits of a 64-bit length of the memory range.

0x0C	0x4	LengthHigh	High 32 bits of a 64-bit length of the memory range.
0x10	0x1	Type	Memory range type. Valid values 0x00 Usable Good Memory 0x01 SMRAM– Overlayed 0x02 SMRAM– Non-Overlayed 0x03 PCIE- PCIE Extended Config Region 0x04 Protected Memory 0x05– 0xFF – Reserved
0x11	0x7	Reserved	Reserved for future use

§