TCG PC Client Specific TPM Interface Specification (TIS)

Specification Version 1.3
21 March 2013

Contact: admin@trustedcomputinggroup.org

TCG Published

TCG

Disclaimers, Notices, and License Terms

THIS SPECIFICATION IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.

Without limitation, TCG disclaims all liability, including liability for infringement of any proprietary rights, relating to use of information in this specification and to the implementation of this specification, and TCG disclaims all liability for cost of procurement of substitute goods or services, lost profits, loss of use, loss of data or any incidental, consequential, direct, indirect, or special damages, whether under contract, tort, warranty or otherwise, arising in any way out of use or reliance upon this specification or any information herein.

This document is copyrighted by Trusted Computing Group (TCG), and no license, express or implied, is granted herein other than as follows: You may not copy or reproduce the document or distribute it to others without written permission from TCG, except that you may freely do so for the purposes of (a) examining or implementing TCG specifications or (b) developing, testing, or promoting information technology standards and best practices, so long as you distribute the document with these disclaimers, notices, and license terms.

Contact the Trusted Computing Group at www.trustedcomputinggroup.org for information on specification licensing through membership agreements.

Any marks and brands contained herein are the property of their respective owners.

Change History

Revision	Date	Description
01	9 September 2010	Created from TIS 1.21 v 70
05	5 May 2011	Merged changes from TIS 1.21 ver 74 110428, SPI clock rewrite, capability address modification
15	8 September 2011	Pre-ballot review version
17	21 September 2011	Addressed Intel comments and remaining clean-up
18	22 September 2011	Updates per September 22, 2011 DDWG meeting
19	29 September 2011	Corrected issues found in 6.4.5 by reviewers, typos and references
20	4 January 2012	Corrected Figure 4. Updated text in Table 9, Section 5.4.2, references in 3.7.2, footnote reference in 3.7.1,and Table 17 in Section 5.7, based on IP review feedback.
21	5 January 2012	Corrected Table 28 and fixed typo in previous change
22	9 January 2012	Corrected reference in 4.2.1 Informative text
23	23 January 2012	References fixed in sections 5.6.12.2, 6.5, 6.7, and 6.7.2; Updated informative text in 6.4.1 to reflect changes made to normative. Fixed typos in Table 13.
24	23 January 2012	Addressed issue in version structure in Table 17, additional informative text change in 6.4.1, reverted change section 4.2.1, reference in 5.6.4
25	24 February 2012	Editorial changes
26	7 June 2012	Relaxed requirement in normative 5, section 6.4.5
27	10 January 2013	TCG Board of Directors Approved as Final for Publication

Contents

Figures

Tables

Corrections and Comments

TCG members may send comments to:
techquestions@trustedcomputinggroup.org

TPM Dependency and Requirements

The TPM used for Host Platforms claiming adherence to this specification MUST be compliant with the *TPM Main Specification; Family 1.2; Level 2; Revision 116* or later.

1. TPM Requirements General Introduction

Start of informative comment

The TCG Main specifications define a TPM for use on any generic platform. Platform-specific functionality is defined in platform specifications such as this document.

End of informative comment

This document details the additional features that MUST be implemented by a TPM for a PC Client platform as defined in this specification.

Unless otherwise indicated, the features in this specification are based on the *TPM Main Specification Family 1.2; Level 2; Revision 116* parts 1 through 3. The term TPM Main Specification SHALL reference these documents and the features they specify.

1.1 Terminology

Start of informative comment

The following terms are used as defined below throughout the document. All other terms are defined in the PC Client Implementation Specification.

TPM Reset refers to the assertion of the TPM_Init hardware signal.

Platform Software refers to the source of the command, which may be an operating system driver or an application.

Platform Hardware refers to platform components including chipsets and associated microcode, and microprocessors and associated microcode.

The S-CRTM refers to code supplied by the platform manufacturer, as a subset of platform firmware that initializes and configures platform components. The S-CRTM is defined in the PC Client Implementation Specification. It is the portion of platform firmware that defines the initial trust boundary.

Operating Systems, or OS, refers generically to an operating system and its collection of drivers and services.

The Static OS is the operating system that is loaded during the initial boot sequence of the platform from its platform reset. Typically, when the Static OS is unloaded the platform performs a platform reset.

The Dynamic OS is the operating system that is dynamically loaded sometime after and usually at the initiation of the Static OS. There may be more than one Dynamic OS per Host Platform but only one can be loaded at a time. The Dynamic OS can be unloaded keeping the Static OS resident and operational.

The terms “read” and “write” are used from the perspective of the calling entity accessing the TPM. A Read is the transaction where the calling entity requests and receives data from a specified register or buffer in the TPM. A Write is the transaction where the calling entity sends data to a register or buffer in the TPM.

The PC Client Implementation Specification is the set of specifications which includes the PC Client Implementation Specification for Conventional BIOS, the TCG EFI Platform Specification and the TCG EFI Protocol Specification. In this document any reference to the PC Client Implementation Specification will be understood to refer to either the PC Client

Specific Implementation Specification for Conventional BOIS or the set of the TCG EFI Platform Specification and the TCG EFI Protocol Specification or all of them.

End of informative comment

1.2 Division of Documentation

Start of informative comment

The PC Client Specifications are divided into two documents:

1. This specification, the *PC Client Interface Specification*, discusses the specifics regarding the requirements of the TPM for the PC Client but only the requirements for the TPM itself. This document discusses the details of what interfaces and protocols are used to communicate with the TPM and the platform-specific set of requirements. Items such as the minimum number of PCRs required and NV Storage available are discussed. The target audience for this document is the TPM manufacturers but platform manufacturers should review it as well.
2. The *PC Client Implementation Specification* specifies the requirements for the TPM as it is implemented on the platform. Issues such as PCR mapping, functional interfaces, pre-operating system driver functionality, and interfaces are discussed. The target audience for this document is platform manufacturers.

End of informative comment

2. Summary of TPM Features to Support the PC Client

2.1 Register Definitions

Start of informative comment

This specification identifies the various registers that allow communication between the TPM and platform hardware and software.

End of informative comment

2.2 Locality

Start of informative comment

Within a platform some components may be more trusted than others and some may be trusted implicitly – for example, a Root of Trust for Measurement. A trusted component, therefore, is a component within the platform that performs operations with certain privileges that are not granted to other platform components. Some trusted components may serve as a Root of Trust for Measurement (RTM) while others may be part of a trusted chain established by an RTM. While some trusted components may have a hierarchical relationship from a platform component perspective (i.e., some are more privileged than others), the only hierarchical access control from the TPM's perspective is the PCR reset and extend attributes as defined in Section 3.7PCR Attributes[1].

"Locality" is an assertion to the TPM that a command's source is associated with a particular component. Locality can be thought of as a hardware-based command authorization. The TPM is not actually aware of the nature of the trusted component and, in fact, does no enforcement at the interface level. The protection and separation of the localities (and therefore the association with the associated components) is entirely the responsibility of the platform components. The ability to reset and extend, notwithstanding, it's important to keep in mind that, from a PCR "usage" perspective, there is no hierarchical relationship between different localities. Platform components may include the OS using protection mechanisms such as virtual memory or paging. The TPM simply enforces locality restrictions on TPM assets such as SEALed blobs restricted by PCR attributes. For example, while a component assigned to Locality 4 can reset Locality 2 PCRs if a blob is SEALed to Locality 2, a component executing at Locality 4 cannot UNSEAL the blob.

The assertion of locality is done by interacting with the TPM at specified blocks of address ranges. Each locality is assigned an address range, and, when a command is received at the address range associated with a locality, the TPM sets the TPM's internal *localityModifier* value to the locality value.

Note on convention for using the term locality: When referring to localities in general the term locality will be lower case (i.e., starts with an 'l'.) When discussing a specific locality, the term locality will be capitalized (i.e., Locality 0 does something.) When using a phrase such as: "executes at Locality 0", this means the command is sent to the memory-mapped

[1] The TPM Access register seize bit also has a hierarchal relationship between the localities but that will not expose TPM assets to other localities.

TPM addresses defined for Locality 0, and the platform components that enforce access to the TPM have authorized that command be sent from that component to that address.

There are six Localities defined (Localities 0 – 4 and Locality None). The text below describes each locality and its associated components:

Locality 4: Trusted hardware component. This is used by the D-CRTM to establish the Dynamic RTM.[2]

Locality 3: Auxiliary components. Use of this is optional and, if used, it is implementation dependent.

Locality 2: Dynamically Launched OS (Dynamic OS) "runtime" environment.

Locality 1: An environment for use by the Dynamic OS.

Locality 0: The Static RTM, its chain of trust and its environment.

Locality None: This locality is defined for using TPM 1.1 type I/O-mapped addressing. The TPM behaves as if Locality 0 is selected. ***Note:*** The TPM 1.1 I/O-mapped addressing is deleted for SPI TPMs in this version of the specification.

See Section 5.1 Locality for more details. See also the PC Client Implementation Specification for more information about how locality is expressed and used.

End of informative comment

2.3 Resettable PCRs

Start of informative comment

Resettable PCRs, with the exception of PCR 16 and PCR 23, are a set of PCRs for use by the Dynamic RTM and its chain of trust. Access to these PCRs is controlled by the various locality indicators.

End of informative comment

2.4 Minimum Amount of NV Storage Specified

Start of informative comment

The *TPM Main Specification* provides for a general-purpose area of Non-volatile storage for use by the platforms. The definition of this area is the purview of the various platform specific specifications. This specification will define the minimum amount required for the PC Client.

End of informative comment

2.5 Minimum Number of PCRs

Start of informative comment

[2] Reference the *PC Client Implementation Specification* for the definition of Dynamic RTM.

The *TPM Main Specification* allows the platform specific specifications to require a minimum number of PCRs and to allocate usage for them based on the needs and the environment of the platform.

End of informative comment

3. Part 1: TPM Attributes

3.1 Power Management

Start of informative comment

While allowed by the LPC specification (if implemented by the TPM), the TPM is designed to be either fully functional (device power management state D0) or not functional (device power management state D3). In practical applications of TPM, power management of the TPM has no real meaning. The TCG specifications define TPM behavior and functions to simplify the TPM's interactions with the platform's components including the software. The TPM_SaveState and TPM_Startup commands were created as a mechanism for the platform's software and BIOS to communicate entry into and exit from the D3 Power State. The TPM_SaveState command allows a Static OS to indicate to the TPM that the platform may enter a low power state where the TPM will be required to enter into the D3 power state. The use of the term "may" is significant in that there is no requirement for the platform to actually enter the low power state after sending the TPM_SaveState command. The software may, in fact, send subsequent commands after sending the TPM_SaveState commands. The TPM_SaveState command simply tells the TPM to save the required volatile contents because power to the TPM may be removed at any time. The TPM is responsible for tracking its internal state so that, if a command that alters the TPM's saved state is sent to the TPM after a TPM_SaveState command, the TPM voids the saved internal state so a subsequent TPM_Startup(ST_STATE) will fail. In this case, it is the responsibility of platform software to send a subsequent TPM_SaveState command to preserve the new internal state of the TPM.

It is the responsibility of the S-CRTM to indicate to the TPM using the TPM_Startup command whether the TPM must reset or restore its saved state (e.g., PCR values, etc.). If the S-CRTM commands the TPM to restore the saved state (i.e., ST_STATE), this restores the transitive trust chain. If the S-CRTM commands the TPM to reset the saved state (i.e., ST_CLEAR), this clears and restarts a new transitive trust state. The rationale here is that the S-CRTM is trusted to establish the initial transitive trust chain, so it should also be trusted to determine whether to restore or clear it.

Power management has changed since the original LPC specification and TPM TIS were produced. The LPCPD# pin, as defined in the LPC specification, is a shared pin allowing for a power management protocol for ACPI S3-aware devices on the LPC bus. As TPMs do not know or participate in Suspend to RAM (ACPI S3), this pin has no meaning for a TPM. As such, the implementation of the LPCPD# pin on a TPM is platform and chipset implementation specific. If TPM vendors implement the LPCPD# pin and power management protocol, they should provide documentation indicating the method to disable the function.

In this version of this specification, the concept of a lower power operating mode has been introduced which allows a TPM to enter a lower power state under specific considerations. The TPM, if in the idle state, can reduce its power consumption by shutting down internal functional blocks as long as its SPI or LPC interface and the TPM registers remain active. The intention is to prevent any impact to existing TPM drivers. When the TPM receives a transaction on its interface that would cause it to move from Idle to Ready, the TPM must exit the low power mode within TIMEOUT B. There is no additional signaling or register bits required to transition the TPM into or out of a low power state. Because of the performance

limitations of the pre-boot environment, this specification does not allow the TPM to enter a low power state prior to the receipt of a TPM_Startup command.

End of informative comment

1. After TPM_Init, the TPM MUST behave as if it is in ACPI Device Power State D0 even if it supports ACPI Device Power States D1-D2.
2. The TPM MUST NOT accept commands unless it is in the ACPI Device Power State D0.
3. The TPM MUST NOT exit the ACPI Device Power State D3 unless it receives TPM_Init.
4. The TPM MUST NOT enter an alternative ACPI Device Power State upon receipt of a TPM_SaveState command.
5. If implementing an LPC TPM, the TPM MUST be implemented to allow for the LPC power management protocol to be disabled by strapping LPCPD# pin HIGH.
6. If implementing an SPI TPM, the TPM MAY support lower power states ONLY if the TPM is in the Idle state.
    a. If lower power states are supported, the TPM MUST respond to requests to transition to the Ready state within TIME_OUT B.
    b. The TPM MUST NOT enter any lower power state between receipt of TPM_Init and receipt of a TPM_Startup command.

3.2 Non-volatile Storage

Start of informative comment

The Non-volatile (NV) Storage provides a general-purpose data storage area for persistent data. The TPM provides the ability to add access control to this area for security or privacy. This area is organized and addressed using indices.

While this area provides a general-purpose storage area for interoperability, some index values are predefined and/or reserved. A predefined index value is one which has been defined by a TCG specification and must be implemented by the TPM. The DIR0 index is an example of a predefined index. A reserved index value is an index which has been defined by TCG, but for which there is no requirement to implement the value, e.g. the Endorsement Key Credential index. A reserved index value, if not implemented must not be used for a different purpose than defined. There is, however, no requirement to use or write to the space addressed by the predefined indices.[3] See the PC Client Implementation Specification for PC Client reserved indices.

The TPM will enforce any defined attributes for the NV storage, however, with the exception of the NV Storage used for GPIO, the contents of the NV Storage are opaque and are not in any way interpreted or enforced by the TPM.

[3] Some indices may be registered or pre-allocated. TCG may support a registry of pre-allocated indices.

A Platform manufacturer may choose to define manufacturer specific indices in the NV Storage. In this case, the indices, if defined with the "D"-bit attribute, will be permanent in the NV Storage and cannot be cleared, except in the case where the TPM_RevokeTrust command is executed. Specific information regarding the treatment of NV Storage is contained in the PC Client Implementation Specification.

NV Storage is also used to access the TPM's General Purpose I/O. For details on this usage, see Section 3.4 General Purpose I/O (GPIO). This usage allocates indices and any access control information, not storage, so it consumes no actual storage area for data, however, it may consume storage for permissions and authorization.

End of informative comment

3.3 NV Storage Size

Start of informative comment

Providing an adequate minimum amount of storage space is difficult to predict based on future and unspecified use of the platform. However, it is prudent to provide for some minimum and predictable amount of storage to allow processes to budget their allocation. For this reason, this specification defines the minimum amount of storage and number of indices that a TPM must implement.

This specification does not define how a TPM vendor must organize the TPM's NV Storage. The TPM vendor may organize the TPM's NV Storage in such a way that the total amount of storage, minus the overhead required to implement individual indices, is allocated dynamically.

However the TPM is implemented, it is expected to provide flexibility in allocation of indices and storage allocation to the indices. The TPM is expected to provide a malloc()-style allocation of the NV storage area rather than provide a fixed size for each index. For example, a caller could define 9 indices of 1 byte each and a single index that consumes the remaining available space. Alternatively, a caller could define 10 indices of equal size. A TPM with a flexible implementation would allow either extreme.

End of informative comment

1. THE TPM MUST provide a minimum of 2048(dec) bytes of NV Storage.
2. The TPM MUST support at least 10 indices with variable sizes as specified by the caller (within the limits of the total number of bytes provided as stated above)
3. The TPM MUST support read access to the NV Storage indices with the attributes TPM_NV_PER_AUTHREAD and TPM_NV_PER_OWNERREAD set to FALSE when the TPM is Deactivated or Disabled using the TPM_NV_ReadValue command.
4. The TPM MUST support write access to the NV Storage indices with the attributes TPM_NV_PER_AUTHWRITE and TPM_NV_PER_OWNERWRITE set to FALSE when the TPM is Deactivated or Disabled using the TPM_NV_WriteValue command.

3.4 General Purpose I/O (GPIO)

Start of informative comment

General purpose I/O (GPIO) provides an optional interface between the TPM's command interface and an external device. The actual use and protocol of the signal is implementation specific and is not specified by TCG.

The TPM's command interface accesses the GPIO pins using the NV Storage interface. This is much like a "memory-mapped" I/O in other architectures. The *TPM Main Specification* reserves 256 indices for this purpose tagged TPM_NV_INDEX_GPIO_xx where xx is the range 00-FF. The platform-specific specification may define each index and its association with a specific GPIO pin and that pin's purpose. The PC Client TPM Interface Specification only specifies the routing of the GPIO index to the GPIO pin. It is the purview of the PC Client Implementation Specifications to specify the routing to the specific device. In general, this is done by mapping the data sent in the TPM_NV_WriteValue and TPM_NV_WriteValueAuth commands' data field to the associated GPIO pin(s).

Because GPIO can be used for security or privacy functions, it must not be open, by default, for public access. For this reason, it is required that the NV Storage area that is mapped to the GPIO be "defined" like any other NV Storage area prior to allowing its use. When defining this area, the TPM Owner may elect to assign rights per the normal TPM_NV_ATTRIBUTES definitions. If the TPM Owner is removed, the area returns to undefined and must be defined again before use. The reason for this behavior is that the new TPM Owner may have different security and privacy requirements for this GPIO.

The range reserved for GPIO is not specific to a particular platform. It is, therefore, a requirement that software or other platform processes using GPIOs understand the nature of the platform before using it (i.e., which NV Storage Index is associated with which GPIO and the purpose of the GPIO on that particular platform).

Note to Implementers

Careful examination of the TPM_NV_ATTRIBUTES reveals that if none of the read or write permission fields (1-2 and 16-18) are set, this area is set to public reads and writes. Also, note the use of negative logic as stated above.

Note that the pin-out specified in Section 6.7.1 TPM Packaging is only recommended and is not mandatory. TPMs are allowed to be implemented using any packaging. However, if this packaging is chosen, the pins, including the location of the GPIO pins, are mandatory. If this packaging is not used, the TPM manufacturer must provide documentation to the platform manufacturer indicating which pin is used as a GPIO pin. For ease in documentation, regardless of whether the TPM implements the recommended packages or uses their own, the designation of this pin will be GPIO-Express-00.

End of informative comment

Implementation of this section is optional; but if implemented, it MUST be done in the manner specified in this section.

3.4.1 Reserved NV Storage Indices for GPIO

Start of informative comment

To allow for both standardized and innovative uses of the GPIO feature, the indices are divided into two areas: Defined and Vendor Specified. Within the Defined area, only the specific use is allowed. Indices within this range which are not currently defined and labeled as reserved must not be used. Within the Vendor Specified set of indices, Vendors (either TPM or platform manufacturers) are allowed to use these for any purpose. It must be

understood that multiple vendors may choose to use the same NV Storage index (or set of indices) for different purposes; therefore, any software that uses this area will be TPM or platform manufacturer specific.

End of informative comment

The TPM and platform manufacturers MUST use Table 1 when allocating NV Storage Indices for TPM_NV_INDEX_GPIO_xx.

Table 1: Reserved NV Storage Indices for GPIO

TPM_NV_INDEX_GPIO_xx where xx is:	Destination or Use	Description
00	GPIO-Express-00	MUST only be used as defined in in this section.
01 – 7F	Reserved	MUST NOT be used
80 – FF	Vendor Specified	MAY be used for vendor-specific purposes. Use is not standardized.

Start of informative comment

Setting of the GPIO-Express-00 pin low enables security and privacy features; therefore, it must always default to high and must remain high until explicitly set low using a TPM_NV_WriteValue or TPM_NV_WriteValueAuth command. Special consideration of this behavior is necessary for TPMs that implement non-D0 ACPI device states to be certain that the GPIO-Express-00 pin does not float to a high value during the transition to a lower power state.

The PC Client Implementation Specification describes the optional connection of this pin to platform components. TPM manufacturers implementing this feature, especially those not implementing the standard TPM pin-out, should review the relevant sections in the PC Client Implementation Specification.

End of informative comment

1. The TPM MAY implement the GPIO functionality specified in this section. However, if the TPM implements any part of this, it MUST be implemented as specified.
2. If the TPM does not implement this section, it MUST NOT allow TPM_NV_INDEX_GPIO_00 to be defined. That is, calls to TPM_NV_DefineSpace with this index anywhere in the requested range MUST fail and the return code SHOULD be TPM_AREA_LOCKED.
3. Access to TPM_NV_INDEX_GPIO_00 MUST be treated as undefined until defined using TPM_NV_DefineSpace. While undefined, or if not implemented, the GPIO-Express-00 pin MUST be *high*.

The following text applies if and only if the TPM implements this GPIO feature.

4. If the TPM uses the recommended packaging in Section 6.7.1 TPM Packaging, it MUST assign the GPIO-Express-00 pin to the pin stated in that section. If the TPM does not use the recommended packaging, the TPM manufacturer must provide documentation to the platform manufacturer indicating which pin is assigned to GPIO-Express-00.
5. Upon completion of any TPM_Startup command, the GPIO-Express-00 pin MUST be set *high* and MUST not change until receipt of a TPM_NV_WriteValue or a TPM_NV_WriteValueAuth command with the TPM_NV_INDEX_GPIO_00 index. (Prior to TPM_Startup the state of this pin is not guaranteed.

6. When there is no TPM Owner, the TPM_NV_INDEX_GPIO_00 area MUST NOT be allocated and MUST be deallocated when an Owner is cleared (see main specification part 3 on TPM_OwnerClear).
7. The GPIO-Express-00 field MUST be bit 0 (i.e., the least significant bit) of TPM_NV_INDEX_GPIO_00. Writes to bits 1-7 MUST be ignored. On read, bits 1-7 MUST return 0's.
8. Upon a write to the GPIO-Express-00 field, the TPM MUST set the state of the GPIO-Express-00 pin to the value written to the GPIO-Express-00 field. That is, writing a "1" to GPIO-Express-00 field MUST set the GPIO-Express-00 pin to *high*; writing a "0" to GPIO-Express-00 field MUST set the GPIO-Express-00 pin to *low*.
    a. The TPM MUST NOT change the state of the GPIO-Express-00 pin until a subsequent write to the GPIO-Express-00 field, change of power state, or an operation that causes the TPM_NV_INDEX_GPIO_00 area to be undefined.
9. Upon a read from TPM_NV_INDEX_GPIO_00, the TPM MUST set the GPIO-Express-00 field to the value of the GPIO-Express-00 pin at the time a TPM_NV_ReadValue or TPM_NV_ReadValueAuth command is executed.

3.5 PCR Requirements

Start of informative comment

This section specifies the number and attributes for the set of PCRs required for a PC Client Platform. The purpose for specifying this is to establish common and expected behavior for both platform hardware and software.

There needs to be an indicator that a Dynamic OS is currently invoked regardless of whether the Dynamic OS is currently controlling the platform. This indication is done using the TPM_STANY_FLAGS.TOSPresent flag. The value of this flag upon any TPM_Startup is FALSE. Since the first DRTM (which begins the chain of trust for the Dynamic OS) is signaled using the TPM_HASH_START command, the TPM_HASH_START command signals the presence of the Dynamic OS by setting the TPM_STANY_FLAGS.TOSPresent flag to TRUE. When the Dynamic OS exits (i.e., tears itself down, not just a change in control of the untrusted operating system), it may need to set this flag back to FALSE depending on the platform's architecture.

The TPM_STANY_FLAGS.TOSPresent flag is used to alter the behavior of the resettable PCRs to allow sealing and attestation to distinguish between values extended into the resettable PCRs while the Dynamic OS is launched and present and values extended to the resettable PCRs while there is no Dynamic OS present. In this way, if an entity were able to extend a known good set of values that would indicate the presence of a Dynamic OS, the values would still not be correct because the starting values are different between Dynamic OS present and not present. Thus, if this flag is FALSE (i.e., no Dynamic OS present), the default value of TPM_PCRVALUE is (0xFFFFFFFF).

End of informative comment

3.6 Number of PCRs

A conformant TPM MUST provide a minimum of 24 PCRs.

400 If a TPM is implemented with more than 24 PCRs, the attributes of the additional PCRs are not defined by this specification.

3.7 PCR Attributes

The attributes of the PCRs MUST be implemented as listed in Table 2.

Table 2: PCR Attributes

PCR Index	Alias	pcrReset	pcrResetLocal for Locality 4, 3, 2, 1, 0	pcrExtendLocal for Locality 4, 3 ,2, 1, 0
0 – 15	Static RTM	0	0,0,0,0,0	1,1,1,1,1
16	Debug	1	1,1,1,1,1	1,1,1,1,1
17	Locality 4	1	1,0,0,0,0	1,1,1,0,0[4]
18	Locality 3	1	1,0,0,0,0	1,1,1,0,0
19	Locality 2	1	1,0,0,0,0	0,1,1,0,0
20	Locality 1	1	1,0,1,0,0	0,1,1,1,0
21	Dynamic OS Controlled	1	0,0,1,0,0	0,0,1,0,0
22	Dynamic OS Controlled	1	0,0,1,0,0	0,0,1,0,0
23	Application Specific	1	1,1,1,1,1	1,1,1,1,1

405 1. The TPM MUST enforce the access to pcrResetLocal and pcrExtendLocal to those granted to each locality per Table 2.

Note: The pcrResetLocal attribute describes which Localities can reset a PCR using a TPM_PcrReset command.

3.7.1 PCR Initial and Reset Values

410 The contents of the cells in Table 3 represent the value of the PCRs at the conclusion of the state named in the title of each column.

Note: Within the scope of specifying the Reset Value for PCRs, the value -1 is defined to be 20 bytes with all bits set to the value of '1'.

Table 3: PCR Initial and Reset Values

PCR Index	TPM_Startup	PCRReset TOSPresent=FALSE	TPM_HASH_START	PCRReset TOSPresent=TRUE
0-15	0	NC*	NC*	NC*
16	0	0	NC	0
17-22	-1	-1	0	0
23	0	0	NC	0

[4] See Section 3.7.2for exceptions and restrictions to this behavior.

Table Notes:
NC = No Change caused by the event.
NC* = Though indicated as “No Change”, it is the PCR attributes that prevent action.

3.7.2 PCR Restrictions

Start of informative comment

The Locality 4 PCR contains the first measurement of the Dynamic RTM for the Dynamic OS. Because the security of the Dynamic Launch is dependent solely on the reset and initial measurement in the Locality 4 PCR, access to Locality 4’s extend operations should not have security implications.

It is expected that any PC Client platform is designed such that the platform protects Locality 4 access to the TPM, ensuring access only from platform components operating at Locality 4.

End of informative comment

1. When the Locality 4 PCR is at its reset value of 0 or -1, the entry for the Locality 4 PCR in Section 3.7 Table 2 SHALL be interpreted as if the column labeled “pcrExtendLocal for Locality 4, 3, 2, 1, 0” contains the bit field definitions: 1, 0 ,0 ,0, 0.
2. Once the Locality 4 PCR is no longer at its reset value of 0 or -1, Table 2 in Section 3.7 applies as written.

3.7.3 PCR Behavior in the Dynamic Launch Sequence

Start of informative comment

The following informative diagram describes the logical sequence and dependencies for resetting PCRs as defined in the preceding sections.

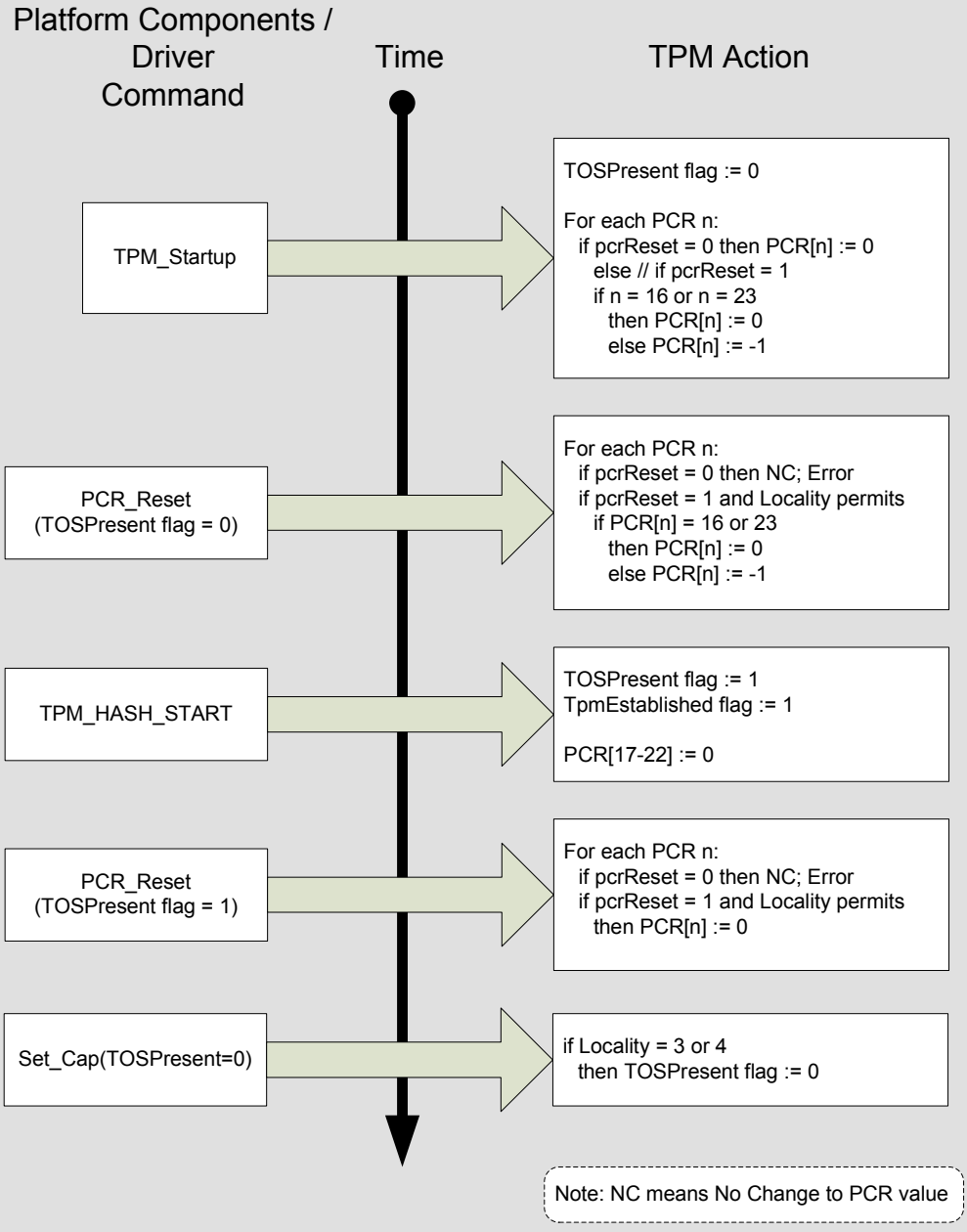


Figure 1 Dynamic Launch Sequence

End of informative comment

3.7.4 TPM Behavior for PCR Structure Values

Start of informative comment

The PCR information structures have several formats. This is mostly caused by the need to expand the information contained within the structure while maintaining backward compatibility with previous versions of TPMs. To provide consistent behavior, the following rules are specified. Software is advised to adhere to the mandatory restriction (i.e., item #2 below) even if the TPMs they are developed on accept values outside the mandatory values.

End of informative comment

1. The TPM MAY accept structures that contain any sizeOfSelect value.
2. The TPM MUST accept structures that contain a sizeOfSelect value of:
   a. A 2 when in a TPM_PCR_SELECTION structure in a TPM_Quote command.
   b. A 3 when in a TPM_PCR_SELECTION structure when used directly as a command operand.
   c. A 2 when in a TPM_PCR_INFO structure.
   d. A 3 when in either a TPM_PCR_INFO_LONG and TPM_PCR_INFO_SHORT structure.

4. Part 2: TPM Commands

4.1 Ordinal Table

Start of informative comment

The *TCG Main Specification* defines all functions needed for all types of platforms in a platform non-specific manner. Some of these defined functions are either not applicable or not appropriate for some types of platforms, and it is left to the platform specific specifications to enumerate which of the TPM commands are to be required, optional, or prohibited for that type of platform.

End of informative comment

To be conformant to this specification, the TPM MUST support ordinals as defined in the following table:

Table 4: Ordinal Table for TPM Commands

Function (by Ordinal Identifier)	M = Mandatory O[5] = Optional X = Deleted in TPM 1.2
TPM_ORD_ActivateIdentity	M
TPM_ORD_AuthorizeMigrationKey	M
TPM_ORD_CertifyKey	M
TPM_ORD_CertifyKey2	M
TPM_ORD_CertifySelfTest	X
TPM_ORD_ChangeAuth	M
TPM_ORD_ChangeAuthAsymFinish	M
TPM_ORD_ChangeAuthAsymStart	M
TPM_ORD_ChangeAuthOwner	M
TPM_ORD_CMK_ApproveMA	M
TPM_ORD_CMK_ConvertMigration	M
TPM_ORD_CMK_CreateBlob	M
TPM_ORD_CMK_CreateKey	M
TPM_ORD_CMK_CreateTicket	M
TPM_ORD_CMK_SetRestrictions	M
TPM_ORD_ContinueSelfTest	M
TPM_ORD_ConvertMigrationBlob	M
TPM_ORD_CreateCounter	M

[5] O1, O2, O3: For each of these flags, if any of these functions are implemented, all those containing the same flag MUST become mandatory.

Function (by Ordinal Identifier)	M = Mandatory O[5] = Optional X = Deleted in TPM 1.2
TPM_ORD_CreateEndorsementKeyPair	M
TPM_ORD_CreateMaintenanceArchive	O1
TPM_ORD_CreateMigrationBlob	M
TPM_ORD_CreateRevocableEK	O3
TPM_ORD_CreateWrapKey	M
TPM_ORD_DAA_JOIN	M
TPM_ORD_DAA_SIGN	M
TPM_ORD_Delegate_CreateKeyDelegation	M
TPM_ORD_Delegate_CreateOwnerDelegation	M
TPM_ORD_Delegate_LoadOwnerDelegation	M
TPM_ORD_Delegate_Manage	M
TPM_ORD_Delegate_ReadTable	M
TPM_ORD_Delegate_UpdateVerification	M
TPM_ORD_Delegate_VerifyDelegation	M
TPM_ORD_DirRead	M
TPM_ORD_DirWriteAuth	M
TPM_ORD_DisableForceClear	M
TPM_ORD_DisableOwnerClear	M
TPM_ORD_DisablePubekRead	M
TPM_ORD_DSAP	M
TPM_ORD_EstablishTransport	M
TPM_ORD_EvictKey	M
TPM_ORD_ExecuteTransport	M
TPM_ORD_Extend	M
TPM_ORD_FieldUpgrade	O
TPM_ORD_FlushSpecific	M
TPM_ORD_ForceClear	M
TPM_ORD_GetAuditDigest	O2
TPM_ORD_GetAuditDigestSigned	O2
TPM_ORD_GetAuditEvent	X
TPM_ORD_GetAuditEventSigned	X
TPM_ORD_GetCapability	M
TPM_ORD_GetCapabilityOwner	M
TPM_ORD_GetCapabilitySigned	X
TPM_ORD_GetOrdinalAuditStatus	X
TPM_ORD_GetPubKey	M
TPM_ORD_GetRandom	M
TPM_ORD_GetTestResult	M

Function (by Ordinal Identifier)	M = Mandatory O[5] = Optional X = Deleted in TPM 1.2
TPM_ORD_GetTick	M
TPM_ORD_IncrementCounter	M
TPM_ORD_Init	M
TPM_ORD_KeyControlOwner	M
TPM_ORD_KillMaintenanceFeature	O1
TPM_ORD_LoadAuthContext	O
TPM_ORD_LoadContext	M
TPM_ORD_LoadKey	M
TPM_ORD_LoadKey2	M
TPM_ORD_LoadKeyContext	O
TPM_ORD_LoadMaintenanceArchive	O1
TPM_ORD_LoadManuMaintPub	O1
TPM_ORD_MakeIdentity	M
TPM_ORD_MigrateKey	M
TPM_ORD_NV_DefineSpace	M
TPM_ORD_NV_ReadValue	M
TPM_ORD_NV_ReadValueAuth	M
TPM_ORD_NV_WriteValue	M
TPM_ORD_NV_WriteValueAuth	M
TPM_ORD_OIAP	M
TPM_ORD_OSAP	M
TPM_ORD_OwnerClear	M
TPM_ORD_OwnerReadInternalPub	M
TPM_ORD_OwnerReadPubek	M
TPM_ORD_OwnerSetDisable	M
TPM_ORD_PCR_Reset	M
TPM_ORD_PcrRead	M
TPM_ORD_PhysicalDisable	M
TPM_ORD_PhysicalEnable	M
TPM_ORD_PhysicalSetDeactivated	M
TPM_ORD_Quote	M
TPM_ORD_Quote2	M
TPM_ORD_ReadCounter	M
TPM_ORD_ReadManuMaintPub	O1
TPM_ORD_ReadPubek	M
TPM_ORD_ReleaseCounter	M
TPM_ORD_ReleaseCounterOwner	M
TPM_ORD_ReleaseTransportSigned	M

Function (by Ordinal Identifier)	M = Mandatory O[5] = Optional X = Deleted in TPM 1.2
TPM_ORD_Reset	M
TPM_ORD_ResetLockValue	M
TPM_ORD_RevokeTrust	O3
TPM_ORD_SaveAuthContext	O
TPM_ORD_SaveContext	M
TPM_ORD_SaveKeyContext	O
TPM_ORD_SaveState	M
TPM_ORD_Seal	M
TPM_ORD_Sealx	O
TPM_ORD_SelfTestFull	M
TPM_ORD_SetCapability	M
TPM_ORD_SetOperatorAuth	M
TPM_ORD_SetOrdinalAuditStatus	O2
TPM_ORD_SetOwnerInstall	M
TPM_ORD_SetOwnerPointer	M
TPM_ORD_SetRedirection	O
TPM_ORD_SetTempDeactivated	M
TPM_ORD_SHA1Complete	M
TPM_ORD_SHA1CompleteExtend	M
TPM_ORD_SHA1Start	M
TPM_ORD_SHA1Update	M
TPM_ORD_Sign	M
TPM_ORD_Startup	M
TPM_ORD_StirRandom	M
TPM_ORD_TakeOwnership	M
TPM_ORD_Terminate_Handle	M
TPM_ORD_TickStampBlob	M
TPM_ORD_UnBind	M
TPM_ORD_Unseal	M

The connection commands manage the TPM's connection to the Trusted Building Block (TBB). See the *PC Client Implementation Specification* for a description of the TBB.

Table 5: Ordinal Table for TPM Connection Commands

Function (by Ordinal Identifier)	Flagged as Optional in the *TCG Main Specification*	M = Mandatory O = Optional
TSC_ORD_PhysicalPresence		M
TSC_ORD_ResetEstablishmentBit		M

4.2 Locality-Controlled Functions

4.2.1 Execution Sequence

Start of informative comment

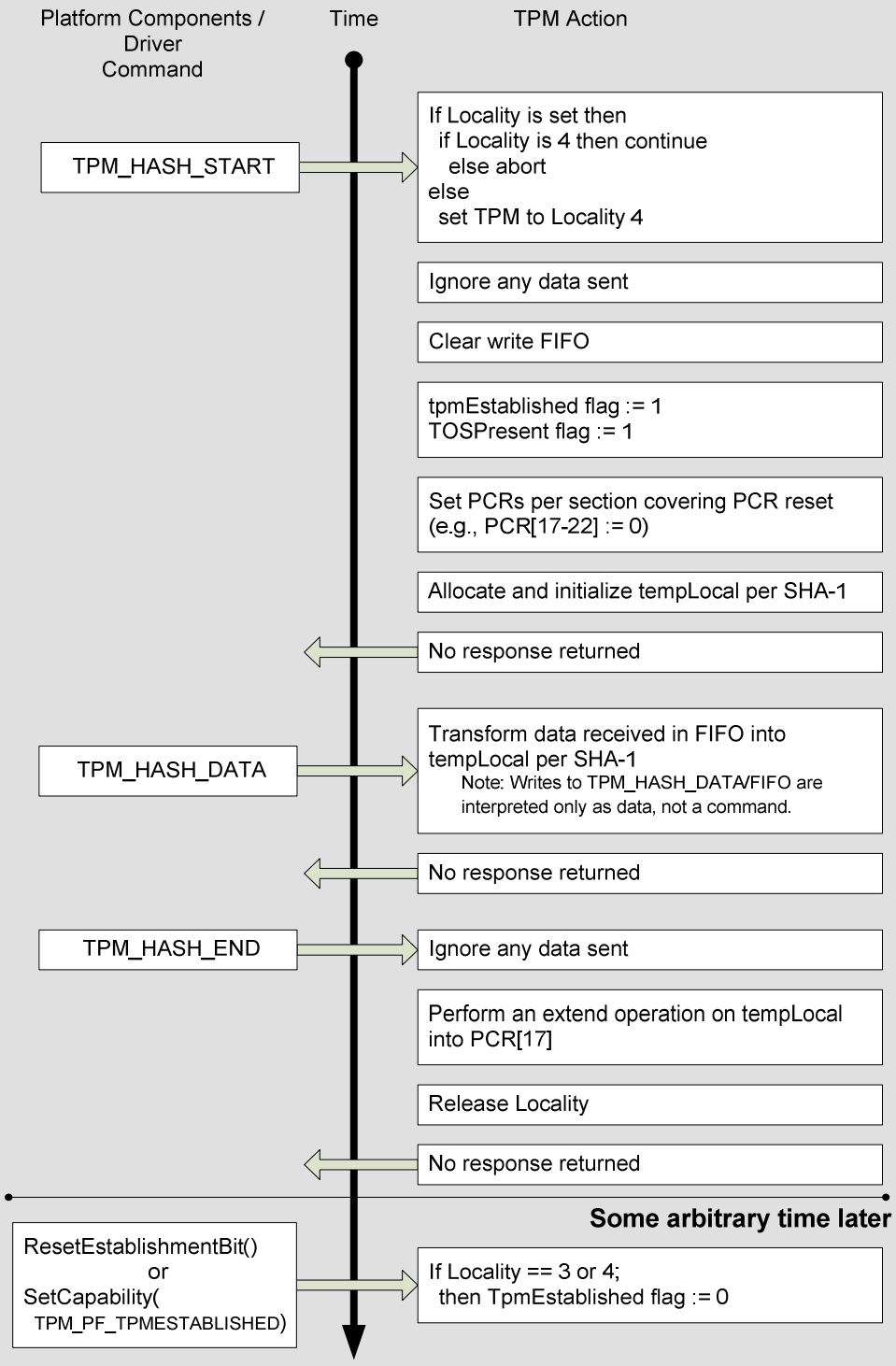
Each RTM is provided with a method to send component measurements. The Static RTM sends its measurements to the TPM using the TPM_SHA1* commands using Locality 0. Because the Dynamic RTM is started while the platform may be in an untrusted state, special and trusted mechanisms must be established to communicate the source of the corresponding commands to the TPM. These commands are indicated and controlled by the appropriate locality.

Locality 4 has the unique ability to send data to the TPM to be hashed and extended to the Locality 4 PCR. This is simply a stream of data sent within the TPM_HASH_DATA interface command. There is no header or other information that accompanies the data. Upon receipt of the TPM_HASH_END interface command, the TPM must complete the Hash and extend the resultant value into the Locality 4 PCR.

The TPM_HASH_START / TPM_HASH_END sequence is also intended to act as a signal to the TPM to reset the resettable PCRs. The goal should be to insert as few WAIT cycles as possible. If the TPM can reset the resettable PCR on TPM_HASH_START and do it quickly, then it can perform the reset upon receipt of the TPM_HASH_START interface command. If the TPM cannot, it should reset the resettable PCRs after the TPM_HASH_END interface command is sent while it is calculating the hash of the received data. Either method is acceptable.

Note: Prior to receiving the TPM_HASH_START command the TPM must have received a TPM_Startup command. If the TPM receives a TPM_HASH_START after a TPM_Init but before a startup command, the TPM treats this as an error, see Section 5.6.10 Errors informative text.

The data sent on the TPM_HASH_START and TPM_HASH_END has no meaning and may be any value.



505 **Figure 2 Overview of D-CRTM Measurement Sequence**

End of informative comment

1. The TPM MUST perform the hash functions using the SHA-1 algorithm as defined in TPM Main Specification Part I, section "4.2.6 SHA-1 Engine".
2. Upon receipt of the TPM_HASH_START interface command, the TPM MUST perform the following operations to affect the resettable PCRs as defined by the following pseudo-code:

   (Note: While the resulting functionality presented by the steps below is normative, the actual operations and their sequence as presented here are informative. There is no requirement to perform the following operations exactly as shown. However implemented, the results MUST be the same as if the TPM were implemented as described below.)

   a. TPM_HASH_START: Upon receipt of the TPM_HASH_START LPC command:

      (1) If no TPM_ACCESS_x.activeLocality field is set, the TPM MUST set the TPM_ACCESS_x.activeLocality field to indicate Locality 4. Any currently executing command MUST be aborted per and subject to Section 5.6.3.1.

      (2) If TPM_ACCESS_x.activeLocality is set, and if the TPM_ACCESS_x.activeLocality field is not 4, the TPM MUST ignore this command.

      (3) The TPM MUST clear the write FIFO.

      (4) If there is an exclusive transport session, it MUST be invalidated.

      (5) Set the TPM_PERMANENT_FLAGS->tpmEstablished flag to TRUE (1). Note: see description of Bit Field: tpmEstablishment in 5.6.11Access Register.

      (6) Set the TPM_STANY_FLAGS->TOSPresent flag to TRUE (1).

      (7) Set PCRs per column labeled TPM_HASH_START in Table 3: PCR Initial and Reset Values.

      (8) Ignore any data component of the TPM_HASH_START interface command.

      (9) Allocate tempLocation of a size required to perform the SHA-1 operation.

      (10) Initialize tempLocation per SHA-1.

   b. TPM_HASH_DATA: Upon receipt of TPM_HASH_DATA interface commands:

      (1) Transform tempLocation per SHA-1 with data received from this command.

      (2) Repeat for each TPM_HASH_DATA interface command received.

   c. TPM_HASH_END: Upon receipt of a TPM_HASH_END interface command (**Note:** all of the following operations MUST be completed before the TPM returns to the Ready state):

      (1) Ignore any data sent with the command.

      (2) Perform finalize operation on tempLocation per SHA-1.

      (3) Perform an "extend" operation, as defined in the TPM_Extend command, of the value within tempLocation into PCR[Locality 4].

      Note:

      - PCR[Locality 4] = SHA-1(PCR[Locality 4] || tempLoc)

- In the previous line above, "PCR[Locality 4]" within and before the SHA-1 function is TPM_PCRVALUE = 0 (i.e., 20 bytes of all zeros).

(4) Clear TPM_ACCESS_x.activeLocality for Locality 4.

Notes and Exceptions:

1) Even though PCR[21:22] are not resettable by Locality 4 processes using a TPM_PcrReset command, PCR[17-22] are all reset by a TPM_HASH_START interface command from a Locality 4 process.
2) There is no response packet returned as a result of any of the TPM_HASH_* interface commands.
3) If no TPM_Startup command is received by the TPM prior to a TPM_HASH_START interface command, the TPM MAY set TPM_ACCESS_x.activeLocality to Locality 4, clear the write FIFO, and invalidate exclusive transport sessions, but the TPM MUST NOT take any other actions.
4) If no TPM_HASH_DATA interface command is sent between the TPM_HASH_START and TPM_HASH_END interface commands, tempLocation will contain the initialization value as defined by SHA-1 with no transformations into it. Thus, in this case, the contents of the Locality 4 PCR SHALL be:
   a) PCR[Locality 4] = SHA-1 (0 || (SHA-1 defined initialization value))
5) If there was not a TPM_HASH_START interface command prior to receipt of the TPM_HASH_END, the TPM MUST NOT change the state of the TPM but MUST clear TPM_ACCESS_x.activeLocality for Locality 4.
6) After successful completion of a TPM_HASH_START interface command:
   a) All data sent to the TPM_DATA_FIFO_4 MUST be treated as data for the hash function.
   b) All cycles and commands other than a write to the TPM_HASH_DATA and TPM_HASH_END interface commands MUST be ignored until a TPM_HASH_END command is received.
7) If a TPM_PCR_Reset command is received with the correct locality, the TPM MUST reset the indicated PCRs as indicated in Section 3.7.1 PCR Initial and Reset Values.
8) Upon any error in the above steps the TPM:
   a) MUST enter Failure Mode.
   b) MUST release locality.

4.2.2 Timing and Protocol

Start of informative comment

The DRTM executes within a resource-restricted environment which is among the reasons the TPM_HASH_* protocol is used rather than the more obvious TPM command ordinals (e.g., TPM_Extend). It is also difficult and unnecessary for this environment to use the register-based TPM_STS_x protocols. Therefore, during the Locality 4 TPM_HASH_* commands, the only method to throttle commands to the TPM uses the bus wait mechanism. (There is no data returning from TPM within this environment.) Specifically, the TPM uses the LPC bus "long wait" sync (using LPC terms) or SPI wait cycles as defined in Section 6.4.5 Flow Control to indicate to the "host" that it is not able to accept more data.

This environment also may not be conducive to "timeouts" and may be very susceptible to delays or hangs. It is important that the TPM be designed to avoid excessive delays and should not cause the bus to hang during this time.

End of informative comment

1. During the TPM_HASH_* commands, the TPM MUST use the appropriate bus wait mechanism (LPC bus "long wait sync" or SPI "wait cycle") to indicate its inability to accept more commands or data. While the TPM MAY set the TPM_STS_x.* status fields they are "undefined" during these commands (i.e. will likely not be read and will not be honored).
2. The TPM MUST respond to the TPM_HASH_START command within TIMEOUT_B.
3. The TPM SHOULD respond to each TPM_HASH_DATA and TPM_HASH_END command within 250 microseconds and MUST respond within TIMEOUT_B.

5. Part 3 TPM Software Interface

5.1 Locality

Start of Informative Comment

Locality Priority is described in Section 2.2Locality.

End of Informative Comment

5.2 TPM Locality Levels

Start of informative comment

TPM 1.2 supports six levels of locality: Locality None and Locality 0-4. PC Client platform usage of locality levels is defined in the PC Client Implementation Specification.

Note that a 1.2 TPM must support Locality 0 - 4. The TPM is allowed to be backwards compatible with TCG 1.1 software and platforms, in which case the TPM must also support Locality None.

Each PCR, during manufacturing of the TPM, has the locality level set for two types of operations: reset and extends. Each PCR has an 8-bit bit-mask, of which 5 bits are defined (one bit for each locality). These bits define the locality which must be present to allow the operation. The bit mask is exclusive, that is, the setting of bit 2 does not imply bit 1, even though Locality 2 is a “higher” locality. If the PCR wants to allow for both Locality 1 and Locality 2, both bits must be set in the mask. If a command attempts an operation on a PCR, it must be received from the correct locality. Locality levels may also be defined for other TPM resources such as NV storage and TPM Keys. For NV Storage, the TPM will only enforce locality controls after the NV Index access attributes have been defined and the NVLock flag has been set.

The usage of PCRs is defined in section 3.7 PCR Attributes.

For the platform, the locality level is indicated by the address used along with the TPM bus Start cycle. For system software, the TPM has a 64 bit address of 0x0000_0000_FED4_xxxx. On LPC, the chipset passes the least significant 16 bits to the TPM. On SPI, the chipset passes the least significant 24 bits to the TPM. The upper bytes will be used by the chipset to select the TPM’s SPI CS# signal. Table 6 shows the locality based on the 16 least significant address bits and assume that either the LPC TPM sync or SPI TPM CS# is used. Note that previous versions of this specification defined an LPC bus cycle to communicate with the TPM. This was done to prevent simple hardware attacks using a device on the LPC bus that decoded I/O or memory cycles using the previously defined START field. Cycles using the normal memory read/write or I/O read/write START field to the following ranges are not decoded by the TPM.

TPM commands, e.g. TSC_ResetEstablishmentBit, may also require locality. The TPM, upon receipt of each command, sets (based on the TPM register Address) the locality for the command. Locality, as presently defined, may be Locality 0 through 4.

End of informative comment

Table 6: Locality Address Definitions

System/Software Address	TPM Address (Using TPM START Cycle)	Locality
FED4_0xxxh	0xxxh	0
FED4_1xxxh	1xxxh	1
FED4_2xxxh	2xxxh	2
FED4_3xxxh	3xxxh	3
FED4_4xxxh	4xxxh	4

Start of informative comment

There are also legacy I/O cycles used by TPM 1.1 devices and software. The TPM 1.2 compliant with earlier versions of this specification may continue to use legacy I/O cycles, but TPMs compliant to this version of the specification which implement SPI will not use legacy I/O cycles. If a TPM implements LPC, the TPM must not respond to I/O cycles when any of the TPM_ACCESS_x.activeLocality fields is set. It is possible that code could be written to provide some software handshake to allow legacy locality and other localities to coexist, but that is not the intended usage. It is expected that a platform may run a particular piece of code that uses legacy locality when there is no other code in the platform that accesses the TPM. Any platform that has multiple pieces of code accessing the TPM should use Locality 0-3 and not use Locality None.

Note: For the following paragraphs, references to legacy locality and locality None apply to LPC TPM's only and have been deleted for SPI TPM's.

If the TPM has been used by Locality 0-4 transactions, and then placed in the "free" state whereby no TPM_ACCESS_x.activeLocality field is set, the TPM can accept legacy I/O cycles. The TPM hardware does not need to "remember" that it was used by locality-based software. It simply accepts legacy I/O cycles whenever no TPM_ACCESS_x.activeLocality field is set.

With regard to an entity authorized by a TPM, there is no difference between Locality None (legacy locality) and Locality 0. The values of LocalityModifier, as defined in Table 7 below, are equivalent, meaning that an entity authorized for use at Locality None can also be used by Locality 0. The LocalityModifier field is not accessible or visible to the caller, except by inclusion in the PCRInfo structure of a TPM entity. The TPM may and can maintain a "free" state by setting LocalityModifier to 00h when no locality has a pending or active transaction, but the caller cannot see that state. Locality access to the Access register is different, in that Locality None looks identical to a TPM in a "free" state.

When the TPM is in a "free" state, it must be prepared to accept a TPM_HASH_START command.

Note on locality priority:

The statements in section 2.2 Locality regarding locality hierarchy notwithstanding, the selection of localities has a priority. If two localities have requested use of the TPM when the current locality relinquishes it, the locality with the highest priority gets access to the TPM. The locality's priority increases as its locality number increases. i.e., Locality 0 has the lowest priority while Locality 4 has the highest.

Note on Locality 4 management:

Locality 4 accesses are controlled by trusted hardware responsible for maintaining the DRTM, and software should not use Locality 4 commands. Trusted hardware should be implemented so that Locality 4 operations are not accessible to software.

If TPM_ACCESS_x.activeLocality is set to a locality other than 4, the TPM_HASH_START operation is ignored. If there is no locality set, TPM_HASH_START makes Locality 4 the active locality. Once the TPM_HASH_DATA sequence is completed at Locality 4, the TPM_HASH_END command releases locality 4 and returns the TPM to a "free" state.

End of informative comment

1. The TPM MUST support five levels of locality (Locality 0, Locality 1, Locality 2, Locality 3, and Locality 4). The TPM MUST maintain the relationship between Locality and LocalityModifier as defined in Table 7 Relationship between Locality and LocalityModifier.
2. For LPC TPMs, the TPM MUST ignore commands sent using the Legacy LPC I/O cycles when TPM_ACCESS_x.activeLocality is set to Locality 0-4.
3. For the purpose of assigning locality, when the host software has requested use of the TPM using either the TPM_ACCESS_x.activeLocality or TPM_ACCESS_x.Seize, the locality with the highest numeric value has the highest priority. i.e., Locality 1 has a higher priority than Locality 0.
4. An LPC TPM MAY continue to use legacy I/O cycles. While responding to legacy I/O cycles the TPM MUST set the LocalityModifier to Locality 0.
5. When the TPM is at the "free" state, it MUST accept any TPM_HASH_START command.

Table 7 Relationship between Locality and LocalityModifier

Locality	Value of LocalityModifer
Locality 0	01h
Locality 1	02h
Locality 2	04h
Locality 3	08h
Locality 4	10h

5.3 Locality Uses

Start of informative comment

Usage of Locality 0 PCRs is determined by the *TCG PC Client Specific Implementation Specification*. Usage of Locality 1-3 PCRs is reserved for uses which are outside the purview of this specification.

The idea behind locality is that certain combinations of software and hardware are allowed more privileges than other combinations. For instance, the highest level of locality might be cycles that only hardware could create.

While higher localities may exist, Locality 4 is the highest locality level defined. These cycles are generated by hardware in support of the DRTM. Cycles which require Locality 4 would include things such as the TPM_HASH_* interface commands.

As an example, assume a platform, including software, has an operating system based on the Static RTM or Static OS, based on either using no PCRs or the set of non-resettable PCRs(0-15), and trusted software, the dynamically launched operating system, or Dynamic OS, which uses the resettable PCRs (17-22). In this case, there is a need to differentiate cycles originating from the two operating systems. Localities 1-3 are used by the Dynamic OS for its transactions. The Dynamic OS has created certain values in the resettable PCRs. Only the Dynamic OS should be able to issue commands based on those PCRs. The Static OS uses Locality 0.

The non-resettable PCRs (i.e., PCR[0-15]) are not part of the Dynamic OS's domain, so Locality 0 transactions can freely use those PCRs, but must be prevented from resetting or extending PCRs used by the Dynamic OS (see section 7.2 PCR Attributes for the PCR's which are resettable by the Dynamic OS).

Note to Platform and OS Implementers:

Each RTM (e.g., the DRTM) has a root PCR associated with it. The fundamental trusted boot requirement is that when the RTM is initiated/reset its associated root PCR must also be reset. Conversely, the root PCR must never be reset unless its associated RTM is also initialized/reset. When launching the Dynamic OS, the root dynamic PCR must only be reset when the DRTM is initiated/reset.

The TPM architecture doesn't provide the TPM with a method for controlling access to any of its localities, therefore, controlling access to the various localities is up to either the platform components or the OS. However, even the Dynamic OS must not be allowed to reset the root Dynamic PCR because the Dynamic OS is what is measured into this PCR. Therefore, the platform components must restrict access to Locality 4 to only the D-CRTM components. This is to protect the resetting of the root Dynamic PCR (i.e., PCR[17]). (Note that because some TPMs have implemented the command FIFO for Locality 4, the platform must protect the entire Locality 4 address range to prevent unauthorized software from executing the TPM_Reset command on PCR[17] at Locality 4.)

While the Dynamic OS is executing, the platform components may provide software access to localities other than 4. In this case, if the Dynamic OS requires protection for these localities it must protect them using methods such as virtual memory management (i.e., paging).

End of informative comment

1. The TPM MUST enforce locality access to each TPM resource that requires locality (e.g. PCRs) or has locality as a definable attribute (e.g. NV Storage or keys). The TPM MUST enforce locality access for TPM commands (e.g. TSC_ResetEstablishmentBit) that require locality.

5.4 TPM Register Space

5.4.1 TPM Register Space Decode

Start of informative comment

Many of the registers in the TPM Register Space are defined using a contiguous address range within a given locality. Most of these registers are accessed with the TPM decoding all addresses within the specified address ranges. For registers with the same function for different localities, the address range for one locality is not contiguous with the address

range for a different locality. Some of these registers which are defined separately with separate address ranges, e.g. the TPM_STS_x register, may be mirrored such that each separate address range (for example 001Ah_0018h for TPM_STS_0 and 101Ah_0118h for TPM_STS_1) points to the same physical register.

Another one of the registers which is defined as having five addresses is the TPM data register (TPM_DATA_FIFO_x). For this register, the addresses within this range may be aliased to one internal register. This version of the specification adds support for an extended size data register (the TPM_XDATA_FIFO_x), which supports transactions from 1B to 64B. The maximum size transaction supported in this register by the TPM is communicated by the DataTransferSizeSupport field in the Interface Capability register (Section 5.7 Interface Capability).

Note that the addresses allocated for the Locality 4 HASH* commands do not exist in Localities 0-3.

The LPC bus transfers a single byte per transaction to any of the registers. The chipset may, for performance reasons, send a DW (4 bytes) for each transaction. This will appear as four distinct LPC bus transactions, with each incrementing the address by 1 byte with each set of transactions starting with the least significant byte – thus being little-endian. Because the TPM can accept only 1 byte per transaction, the TPM must ignore the 2 least significant bits of the address for the data registers thus receiving the data serially. However, it should not require or expect that each address is incremented modulo-4.

The SPI bus does not limit transfers to a single byte. The extended size data register (TPM_XDATA_FIFO_X) allows a single write to offset 0x0080h up to 64B, without requiring software to increment the address. It is technically possible to send a single transaction on SPI that spans more than one register in the TPM's address space. Software should never attempt to access multiple registers in a single transaction. Software should access each register in unique, individual transactions and not attempt to cross register boundaries. Software may access only part of a register, e.g. read or write one byte of a 4B register. Software should not initiate a transaction that is either larger than the register size (2B access to a 1B register), or that extends beyond the defined register boundary (2B transaction to offset 0x3 of a register defined as existing within the address range of 0x0 to 0x3). This is simply good software behavior and these guidelines are not specific to TPM transactions, but apply to all hardware and software.

Because the TPM must be designed to handle cases where software behaves badly, this specification defines behavior for the TPM in the event that a transaction crosses multiple registers. TPM vendors should design their hardware so that bad software does not impact the state or security of the TPM. Specific error behavior is specified in Section 5.6.10 Errors.

End of informative comment

1. For SPI, if a TPM receives an access request with a length that exceeds the size of the register specified in the transaction address:
   a. Reads:
      i. The TPM MUST return the data for the register designated by the start address.
      ii. The TPM MAY return the data for additional, adjacent registers within the targeted address range, or the TPM MAY return dummy data for additional, adjacent registers within the targeted address range.

b. Writes:

i. The TPM MUST update the register designated by the start address.

ii. The TPM MAY update additional, adjacent registers within the targeted address range, or the TPM MAY ABORT (drop) all subsequent data for additional, adjacent registers.

iii. The TPM MUST NOT change the state of adjacent registers if the writes to that register are dropped.

c. The TPM MUST NOT abort an entire transaction that crosses a register boundary.

d. When a transaction cross a register boundary, the TPM MUST NOT allow data from that transaction to corrupt future SPI transactions.

2. TPM_DATA_FIFO_x
The TPM MUST ignore the 2 least significant bits of the address for these registers and accept each byte received to any of the addresses within this register as a single transfer to be the base address.

3. TPM_XDATA_FIFO_x

a. The TPM MUST accept transactions to offset 0x0080h which are of any length from 1B to the maximum supported length (as reported in the Interface Capability register, Section 5.7 Interface Capability).

b. The TPM MAY accept transactions to offset 0x0081h-0x0083h which are of lengths between 1-3B. ***Note***: This behavior mirrors the behavior of the TPM_DATA_FIFO_x.

c. The TPM MAY accept transactions to offset 0x0080h which are of lengths larger than the maximum supported length (as reported in the Interface Capability register 5.7 Interface Capability), based on the existing interface protocol using TPM_STS_x.burstCount and TPM_STS_X.Expect as defined in Section 5.6.12 Status Register.

4. All other registers:

a. The TPM MUST fully decode the address down to the byte level (unless otherwise specified).

b. The TPM MUST interpret registers that have multiple addresses as follows:

i. The lowest address within the set of addresses contains the least significant byte with the bits incrementing to each successive address up to the highest address which contains the most significant byte.

Note: Addresses are implemented as in Table 8 as shown below.

Table 8: Example Bit-to-Address Mapping

Address	Data Bit Position	Example
00000008	7:0	0000 0000 0000 0000 0000 0000 0000 1000
00000900	15:8	0000 0000 0000 0000 0000 1001 0000 0000
000A0000	23:16	0000 0000 0000 1010 0000 0000 0000 0000
0B0000000	31:24	0000 1011 0000 0000 0000 0000 0000 0000

5.4.2 Register Space Addresses

Start of Informative comment

Table 9 lists the addresses decoded by the TPM. The TPM_ACCESS_x register has multiple, separate and unique instances, one per locality. The other register addresses alias to a single register with the locality used to determine if accesses are permitted or Aborted.

End of Informative comment

Table 9: Allocation of Register Space for TPM Access

Offset	Register Name	Description
Locality 0		
0000h	TPM_ACCESS_0	Used to gain ownership of the TPM for this particular Locality.
000Bh-0008h	TPM_INT_ENABLE_0	Interrupt configuration register
000Ch	TPM_INT_VECTOR_0	SIRQ vector to be used by the TPM
0013h-0010h	TPM_INT_STATUS_0	Shows which interrupt has occurred
0017h-0014h	TPM_INTF_CAPABILITY_0	Provides the information about supported interrupts and the characteristic of the burstCount register of the particular TPM.
001Ah-0018h	TPM_STS_0	Status Register. Provides status of the TPM
00027h-0024h	TPM_DATA_FIFO_0	ReadFIFO or WriteFIFO, depending on the current bus cycle (read or write). These four addresses are aliased to one inside the TPM. Note: The TPM is not required to check that the addresses on the LPC bus are incrementing modulo-4, even though platform hardware would most likely send it that way. The read or write data could be performed by accessing 0024h repeatedly without using the other addresses.
0080h-0083h	TPM_XDATA_FIFO_0	Extended ReadFIFO or WriteFIFO, depending on the current bus cycle (read or write). Transactions to this address may be any size from 1B to maxTransferCapability identified in the capability register. The TPM SHOULD alias this address with the TPM_DATA_FIFO at offset 0x0024
00BFh-0084h	Reserved	Reserved. These addresses are reserved by the chipset. The TPM should not respond to accesses to these addresses. Reserving this address range ensures that software which issues writes larger than 1B to offset 0080h doesn't inadvertently encounter a register in the TPM in this space.
0F03h-0F00h	TPM_DID_VID_0	Vendor and device ID
0F04h	TPM_RID_0	Revision ID
0F7Fh-0F05h	Reserved	Reserved for Future Use
0F80h	FIRST_LEGACY_ADDRESS_0	Deprecated. Alias to I/O legacy space. Deleted for SPI TPM's. SPI TPM's MUST abort accesses to this address.
0F84h	FIRST_LEGACY_ADDRESS_EXTENSION_0	Deprecated. Additional 8 bits for I/O legacy space extension. Deleted for SPI TPM's. SPI TPM's MUST abort accesses to this address.
0F88h	SECOND_LEGACY_ADDRESS_0	Deprecated. Alias to second I/O legacy space. Deleted for SPI TPM's. SPI TPM's MUST abort accesses to this address.
0F8Ch	SECOND_LEGACY_ADDRESS_EXTENSION_0	Deprecated. Additional 8 bits for second I/O legacy space extension. Deleted for SPI TPM's. SPI TPM's MUST abort accesses to this address.
0FFFh-0F90h		Vendor-defined configuration registers

Offset	Register Name	Description
Locality 1		
1000h	TPM_ACCESS_1	Used to gain ownership of the TPM for this particular Locality.
100Bh-1008h	TPM_INT_ENABLE_1	Same as TPM_INT_ENABLE_0
100Ch	TPM_INT_VECTOR_1	Same as TPM_INT_VECTOR_0
1013h-1010h	TPM_INT_STATUS_1	Same as TPM_INT_STATUS_0
1017h-1014h	TPM_INTF_CAPABILITY_1	Same as TPM_INTF_CAPABILITY_0
101Ah-1018h	TPM_STS_1	Same as TPM_STS_0
1027h-1024h	TPM_DATA_FIFO_1	Same as TPM_DATA_FIFO_0
1080h-1083h	TPM_XDATA_FIFO_1	Same as TPM_XDATA_FIFO_1
10BF-1084h	Reserved	Reserved. These addresses are reserved by the chipset. The TPM should not respond to accesses to these addresses. Reserving this address range ensures that software which issues writes larger than 1B to offset 0080h doesn't inadverntently encounter a register in the TPM in this space.
1F03h-1F00h	TPM_DID_VID_1	Same as TPM_DID_VID_0
1F04h	TPM_RID_1	Same as TPM_RID_0
1F7Fh-1F05h	Reserved	Reserved for Future Use
1F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
1F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
1F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
1F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
1FFFh-1F90h		Vendor-defined configuration registers
Locality 2		
2000h	TPM_ACCESS_2	Used to gain ownership of the TPM for this particular Locality.
200Bh-2008h	TPM_INT_ENABLE_2	Same as TPM_INT_ENABLE_0
200Ch	TPM_INT_VECTOR_2	Same as TPM_INT_VECTOR_0
2013h-2010h	TPM_INT_STATUS_2	Same as TPM_INT_STATUS_0
2017h-2014h	TPM_INTF_CAPABILITY_2	Same as TPM_INTF_CAPABILITY_0
201Ah-2018h	TPM_STS_2	Same as TPM_STS_0
2027h-2024h	TPM_DATA_FIFO_2	Same as TPM_DATA_FIFO_0
2080h-2083h	TPM_XDATA_FIFO_2	Same as TPM_XDATA_FIFO_0
20BF-2084h	Reserved	Reserved. These addresses are reserved by the chipset. The TPM should not respond to accesses to this addresses. Reserving this address range ensures that software which issues writes larger than 1B to offset 0080h doesn't inadverntently encounter a register in the TPM in this space.
2F03h-2F00h	TPM_DID_VID_2	Same as TPM_DID_VID_0
2F04h	TPM_RID_2	Same as TPM_RID_0
2F7Fh-2F05h	Reserved	Reserved for Future Use
2F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
2F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used

Offset	Register Name	Description
2F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
2F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
2FFFh-2F90h		Vendor-defined configuration registers
Locality 3		
3000h	TPM_ACCESS_3	Used to gain ownership of the TPM for this particular Locality.
300Bh-30008h	TPM_INT_ENABLE_3	Same as TPM_INT_ENABLE_0
300Ch	TPM_INT_VECTOR_3	Same as TPM_INT_VECTOR_0
3013h-3010h	TPM_INT_STATUS_3	Same as TPM_INT_STATUS_0
3017h-3014h	TPM_INTF_CAPABILITY_3	Same as TPM_INTF_CAPABILITY_0
301Ah-3018h	TPM_STS_3	Same as TPM_STS_0
3027h-3024h	TPM_DATA_FIFO_3	Same as TPM_DATA_FIFO_0
3080h-3083h	TPM_XDATA_FIFO_3	Same as TPM_XDATA_FIFO_0
30BFh-3084h	Reserved	Reserved. These addresses are reserved by the chipset. The TPM should not respond to accesses to this addresses. Reserving this address range ensures that software which issues writes larger than 1B to offset 0080h doesn't inadverntently encounter a register in the TPM in this space.
3F03h-3F00h	TPM_DID_VID_3	Same as TPM_DID_VID_0
3F04h	TPM_RID_3	Same as TPM_RID_0
3F7Fh-3F05h	Reserved	Reserved for Future Use
3F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
3F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
3F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
3F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
3FFFh-3F90h		Vendor-defined configuration registers
Locality 4		Lo
4000h	TPM_ACCESS_4	Used to gain ownership of the TPM for this particular Locality.
400Bh-4008h	TPM_INT_ENABLE_4	Same as TPM_INT_ENABLE_0
400Ch	TPM_INT_VECTOR_4	Same as TPM_INT_VECTOR_0
4013h-4010h	TPM_INT_STATUS_4	Same as TPM_INT_STATUS_0
4017h-4014h	TPM_INTF_CAPABILITY_4	Same as TPM_INTF_CAPABILITY_0
401Ah-4018h	TPM_STS_4	Same as TPM_STS_0
4020h	TPM_HASH_END	This signals the end of the hash operation. See Section 8 Locality-Controlled Functions for detailed description This command MUST be done on the LPC bus as a single write to 4020h. Writes to 4021h to 4023h are not decoded by the TPM.
4027h-4024h	TPM_HASH_DATA/ TPM_DATA_FIFO_4	Same as TPM_DATA_FIFO_0 except that this location is also used as the data port for the Locality 4 HASH procedure as defined in Section 8 Locality-Controlled Functions.

Offset	Register Name	Description
4028h	TPM_HASH_START	This signals the start of the hash operation. See Section 8 Locality-Controlled Functions for detailed description This command MUST be done on the LPC bus as a single write to 4028h. Writes to 4029h to 402Bh are not decoded by TPM.
4080h-4083h	TPM_XDATA_FIFO_4	Same as TPM_XDATA_FIFO_0
40BF-4084h	Reserved	Reserved. These addresses are reserved by the chipset. The TPM should not respond to accesses to this addresses. Reserving this address range ensures that software which issues writes larger than 1B to offset 0080h doesn't inadverntently encounter a register in the TPM in this space.
4F03h-4F00h	TPM_DID_VID_4	Same as TPM_DID_VID_0
4F04h	TPM_RID_4	Same as TPM_RID_0
4F7Fh-4F05h	Reserved	Reserved for Future use
4F80h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_0 but MUST NOT be used
4F84h	Reserved	Parallel to FIRST_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
4F88h	Reserved	Parallel to SECOND_LEGACY_ADDRESS_0 but MUST NOT be used
4F8Ch	Reserved	Parallel to SECOND_LEGACY_ADDRESS_EXTENSION_0 but MUST NOT be used
4FFFh-4F90h		Vendor-defined configuration registers

Offset	Register Name	Description
All addresses not defined in the table above		Reserved, reads return FFh; writes are dropped.

The following sections provide implementation details on the defined registers. Note that each register has multiple versions; e.g., TPM_STS_x represents TPM_STS_0, TPM_STS_1, TPM_STS_2, TPM_STS_3, and TPM_STS_4. The DID/VID, RID, and all the TCG and vendor-specific registers MAY have only one physical copy. If so implemented, these registers MUST be accessible from any locality. If implemented as separate physical registers, each copy MUST hold the same data. See Section 6.1 Locality Usage per Register,

Table 22: Register Usage Based on Locality Setting for more information on reading and writing the configuration registers.

5.5 System Interaction and Flows

5.5.1 Configuration Registers

5.5.1.1 DID/VID Register

Table 10: DID/VID Register

Abbreviation:			TPM_DID_VID_x
General Description:			Vendor and Device ID for the TPM
Default			Vendor specific
Bit Descriptions:			
31:16	Read Only	did	Device ID – vendor specific
15:0	Read Only	vid	Vendor ID – Assigned by TCG Administrator. This is represented within the register in big-endian format. For example, a vendor ID of 0x1234 would be represented as: Bits 7:0 = 34 (0011 0100); Bits 15:8 = 12 (0001 0010).

5.5.1.2 RID Register

Table 11: RID Register

Abbreviation:			TPM_RID_x
General Description:			Revision ID for the TPM
Default			Specific to each revision
Bit Descriptions:			
7:0	Read Only	rid	Revision ID – specifies the revision of the component

5.5.1.3 Legacy Configuration Register

Table 12: Legacy Configuration Register

Abbreviation:	

General Description:			Alias of the standard I/O configuration
Default			Specific to each vendor
Bit Descriptions:			
7:0	R/W		Configuration space defined by each vendor. The TPM MUST support one of the options presented here. It may not use the range of 0F80h to 0F8Fh for some other purpose. The TPM MUST either alias the ranges of 0F80h, 0F84h, 0F88h, and 0F8Ch to the same registers defined in I/O space for that vendor's TPM or the TPM MUST abort cycles to these four addresses. SPI TPMs MUST abort cycles to these four addresses. The TPM MUST also decode the range of XF80h to XF8Fh for X = 1 to 4 for the other localities exactly the way it does for Locality 0; however, these cycles MUST be aborted.

5.6 TPM's Software Interaction

Start of informative comment

When a platform is powered on, platform hardware issues a TPM_INIT to the TPM. After each TPM_Init, the platform must issue a TPM_Startup command to the TPM before issuing any other TPM command. The command and startup type informs the TPM how to initialize itself, for example, by informing the TPM to restore or clear the state of the PCRs that may retain their state across an S3 suspend. The platform firmware is required to perform the TPM_Startup command. The TPM cannot respond to any command before the platform firmware communicates the desired initialization state (see *TPM Main Specification* Part 1, section 9.1).

Note: All references below to legacy accesses, legacy locality, and legacy I/O cycles apply to LPC TPM's only and have been deleted for SPI TPM's.

There are two methods software may use to communicate with the TPM. The preferred method is to use the LPC Locality Read and LPC Locality Write cycles. The legacy access method uses the TPM's legacy I/O-mapped port and may be used prior to usage of the TPM locality cycles. A TPM which supports both the legacy I/O cycles and the LPC locality cycles may respond to I/O cycles if there is no currently active locality for the TPM. Even if the preferred access method is never used by software, the TPM may be used by legacy applications.

After software which uses the TPM locality cycles has started, the TPM has two possible users:

1. Applications that do not use the TPM's locality cycles to talk to the TPM.
2. Applications that do use the TPM's locality cycles to talk to the TPM.

Since there are two possible users of the TPM, some synchronization method is required.

Assume that the legacy code uses only the legacy I/O mapped port and that the OS will use the Locality Read and Locality Write cycles. (Note that how the platform maps CPU accesses to these new cycles is chipset/platform dependent.) This example implies a design with the TPM having two logical ports, one I/O mapped and the other TPM mapped.

It is important to understand that the locality using the TPM and its interface is architected to be a non-preemptive use of the TPM. When there are multiple software users spanning multiple localities, the following explains the handshake mechanism.

Each software agent, when it wishes to use the TPM, must set TPM_ACCESS_x.requestUse to a "1" for the locality it wishes to access. In this case, the TPM is idle so the first agent

that sets this field will become the user. The TPM must set TPM_ACCESS_x.activeLocality for the locality that gains access to TPM. All other localities that have written the TPM_ACCESS_x.requestUse field must poll on the TPM_ACCESS_x register and read a "0" for their TPM_ACCESS_x.activeLocality. The winning locality will read a "1" in this field and start issuing commands to the TPM.

When the currently active locality is finished with the TPM, it must set its TPM_ACCESS_x.activeLocality field to a "1". This indicates that it is finished with its series of commands and causes the TPM to clear the TPM_ACCESS_x.requestUse for the locality which wrote "1" to its TPM_ACCESS_x.activeLocality field. The TPM must look at all pending TPM_ACCESS_x.requestUse fields and grant the access to the highest locality with a pending request by setting TPM_ACCESS_x.activeLocality for that locality. The others continue waiting.

If software, for some reason, decides a lesser locality's software is not playing fair or is hung (by exceeding the maximum timeout value as specified by the TSS), then it can seize the TPM from the current user, as long as that user is at a lower locality. To accomplish this, a "1" is written to the TPM_ACCESS_x.Seize field. This forces the TPM to stop honoring cycles from the other locality, and only honor the new locality's requests.

End of informative comment

5.6.1 Handling Command FIFOs

Start of informative comment

Before issuing a command to the TPM, the software reads the TPM_STS_x register to see if the TPM's state allows it to accept commands.

Software sends commands to the TPM and reads results from the TPM using a data FIFO. When the TPM_STS_x.burstCount field is > 0, this indicates to the software that the data FIFO is ready to accept more data of a command (during a command's send phase) or return more data from a command (response from a command completion). Since the TPM is not allowed to drop a cycle because of an internal stall, if the TPM cannot accept a write cycle or respond to a read cycle, it must insert wait states on the bus using the mechanism appropriate to the bus interface, i.e. stall the LPC bus using standard LPC wait syncs or insert one or more SPI wait cycles.. See Section 5.6.12 Status Register, for a detailed description of burstCount.

The FIFO is only a stack of bytes going into and out of the TPM. TPM_STS_x.burstCount indicates only the depth of the command FIFO, not the direction nor whether the TPM expects more data to be sent or received. TPM_STS_x.Expect and TPM_STS_x.DataAvail fields indicate to the software when the TPM expects more data during a command's send phase or has more data to be read during a read results phase.

End of informative comment

1. The TPM MUST maintain the TPM_STS_x register so that software can determine whether the TPM is in a state where it can accept commands. The TPM MUST NOT drop a cycle because of an internal stall. If the TPM cannot accept a write or read cycle, then the TPM MUST stall the bus using the appropriate method for the bus (standard LPC Wait Syncs or SPI wait cycles).

5.6.2 Completion Command Details

5.6.2.1 Command Send

Start of informative comment

To send a command the software must first set TPM_STS_x.commandReady = 1. Upon receipt of TPM_STS_x.commandReady, the TPM may set TPM_STS_x.Expect = 1 indicating it is ready to receive the command. When the data FIFO is ready to begin receiving the command data, the TPM sets TPM_STS_x.burstCount > 0. The TPM uses TPM_STS_x.burstCount field to throttle the data into the data FIFO.

The TPM keeps TPM_STS_x.Expect = 1 until it receives all the expected data for the command. When the TPM receives all the data for the command (the TPM can calculate this using the command size parameter which is within the first 10 bytes of the command) it sets TPM_STS_x.Expect = 0 indicating to the software that all data expected has been received.

The software signals the TPM to begin executing the command by writing a '1' to the TPM_STS_x_tpmGo field. Upon receipt of TPM_STS_x_tpmGo = 1, the TPM begins executing the command.

End of informative comment

1. Upon receipt of TPM_STS_x.commandReady, the TPM MUST prepare to receive a command.
2. When ready, the TPM MUST set TPM_STS_x.commandReady = 1 and the TPM MUST set TPM_STS_x.burstCount > 0 to indicate to software that it can begin writing command data to the data FIFO. The TPM MAY set TPM_STS_x.Expect = 1.
3. When the TPM receives the first Byte of the command data, it MUST set TPM_STS_x.commandReady = 0 and TPM_STS_x.Expect = 1 as an indication to the software that it expects further command data. The TPM MUST use TPM_STS_x.burstCount to indicate to software whether the data FIFO can accept more data.
4. The TPM MUST keep TPM_STS_x.Expect = 1 until it has received all of the data for this command. When the TPM reads the last byte of data from its data FIFO the TPM MUST set TPM_STS_x.Expect = 0.
5. The TPM MAY ignore TPM_STS_x.tpmGo until TPM_STS_x.Expect is set to 0.

5.6.2.2 Data Availability

Start of informative comment

When a command completes, the TPM puts the results into the data FIFO, which is read via the TPM_DATA_FIFO_x register. Once the TPM has data that can be read, the TPM sets TPM_STS_x.dataAvail = 1 and it remains '1' until all data from the command response are read. After the last byte of the response is read, the TPM sets TPM_STS_x.dataAvail = 0. The TPM uses TPM_STS_x.burstCount field to throttle the response out of the data FIFO.

After sending a command, the software reads the TPM_STS_x.dataAvail register to see if the response from the TPM is available, indicating a command has completed. If the TPM_STS_x.dataAvail field is "1", at least 1 byte of the command response data is available.

End of informative comment

1. Upon completing a command the TPM MUST place the command's response data into the data FIFO.
2. The TPM MUST set TPM_STS_x.dataAvail = 1 as an indication to the software that the command has completed and data is available to be read from the data FIFO. When the last byte of the response data is read from the data FIFO the TPM MUST set TPM_STS_x.dataAvail = 0.

   Note: A value of 1 only indicates that there is at least one byte in the data FIFO; it is not an indicator that the data can be read from the data FIFO. I.e., this is not the final indicator to software that it can begin reading from the data FIFO. Software must also wait until TPM_STS_x.burstCount > 0, see below.
3. The TPM MUST set TPM_STS_x.burstCount > 0 to indicate to software that it can begin reading the response data from the data FIFO. Once the software has started to read the response from the data FIFO, the TPM MUST use TPM_STS_x.burstCount as an indicator to software that data is available in the data FIFO.

5.6.3 Aborts

5.6.3.1 Command Aborts

Start of informative comment

There are several ways to cause a TPM to abort an executing command: TPM_ACCESS_x.Seize, TPM_STS_x.commandReady, and TPM_ACCESS_x.activeLocality. Because of implementation differences and the non-deterministic nature of some commands that may be executing, the TPM may not be able to respond to an abort immediately or within a predictable time. This non-deterministic behavior causes driver design difficulties because the driver will not be able to distinguish between a TPM waiting normally and a TPM that has encountered an error and is not responsive. Therefore, a maximum amount of time is specified so TPM manufacturers have a design parameter that drivers can rely upon.

The TPM's internal state after an abort may be set to the state of the TPM prior to the aborted command or to the state it would have entered after completing the aborted command.

The purpose for an Abort when setting TPM_ACCESS_x.SEIZE or x.activeLocality is that the TPM cannot be allowed to "leak" information between localities. In other words, the response to a command sent from one locality cannot be returned to another locality.

Note: Because there is no requirement for a TPM to handle more than one operation at a time, there can be no actual and standardized TPM command to cause an abort. The method for signaling an abort to the TPM is by writing to specific registers.

End of informative comment

1. Upon a successful abort, the TPM MUST stop the currently executing command, clear the FIFOs, and transition to idle state.
2. The following operations MUST cause an abort:
   a. Writing a "1" to TPM_STS_x.commandReady during the execution of a command.
   b. Writing a "1" to TPM_STS_x.commandReady during the receipt of a command but before execution of a command.
   c. Writing a "1" to TPM_ACCESS_x.Seize ,but only when successful.
   d. Writing a "1" to TPM_ACCESS_x.activeLocality.
   e. Successful completion of the TPM_HASH_START per Section 4.2, Locality-Controlled Functions.
3. The TPM internal state MAY either be in the pre-aborted command or post-aborted command state and MUST not be in any intermediate state.
4. For commands indicated as short or medium duration (i.e., those that do not cause key generation), the TPM MUST respond to an abort in less than the medium duration as reported by TPM_GetCapability command. For commands indicated as long duration or those that cause key generation, the TPM MUST respond to a request to abort the command within 2 seconds.

5.6.3.2 Bus Aborts

Start of informative comment

For LPC, an LPC Abort cycle requires that the cycle have no effect on the TPM. There are two possible implementations, either of which is acceptable:

1. The TPM may simply not drive a valid LPC SYNC. This will cause the chipset to return FFh to the CPU for reads. The write data will be dropped.
2. For writes, the TPM may accept them and do nothing with the writes. The TPM will not provide any TPM-Response to these writes, nor does it provide any indication that it has seen a write but dropped it. For reads, the TPM, if it responds with a valid LPC SYNC, must return FFh as the data. This mimics a true LPC or PCI Master Abort from the CPU's perspective.

For SPI, there is no analog to the LPC SYNC signal. To provide a similar mechanism to the LPC Abort cycle, this specification defines a protocol using MISO.

End of informative comment

LPC Aborts

1. An LPC Abort cycle causes the TPM to ignore the current LPC bus cycle.

SPI Aborts

1. For Read Cycles, the TPM MUST abort a cycle by driving '1' on MISO and continue to hold MISO at '1' until its CS# signal is deasserted.
2. For Write Cycles, the TPM MUST abort a cycle by driving a '1' on MISO, then drop all incoming data.
3. The TPM MAY use the standard SPI Wait mechanism, as defined in Section 6.4.5 Flow Control, to insert a wait state while decoding the cycle before issuing an abort.

5.6.4 Failure Mode

Start of informative comment

Several conditions can cause the TPM to enter a specifically defined state called "failure mode". These conditions and the behavior of the TPM are defined in the TPM Main Specification and are not reproduced here. This section defines additional considerations specific to the behavior of the TIS.

Notes on the Normative text below:

Normative 2: This allows the system software to perform the allowed remediation actions on the TPM. The requirement to allow changing locality exists because the TPM's locality when it entered failure mode may not be the appropriate locality for performing the allowed remediation.

End of informative comment

While the TPM is in a failure mode:

1. In addition to the requirements called out in the TPM Main Specification, the TPM_HASH_START, TPM_HASH_DATA and TPM_HASH_END commands MUST appear to the caller as dropped writes (i.e., they are not allowed to hang or suspend the system), but MUST NOT perform any actions on the TPM such as those specified in Section 4.2.1, Execution Sequence.
2. All TIS registers MUST remain fully functional including the ability to change locality.

5.6.5 Command Duration

Start of informative comment

It is important to distinguish between the two terms: duration and timeout. Duration is the amount of time for a TPM to execute a command once the TPM has received the command's complete set of bytes and the software starts the operation by writing a "1" to TPM_STS_x.tpmGo. Duration has no relationship to the timings of the interface protocols. Those are timeouts (see Section 5.6.6).

During a platform's early boot phase, performance is critical and resources are limited. For this reason, constraints are placed on commands that are typically required during this phase to eliminate the need to compensate for implementation differences of different TPMs. Since the same platform requirements drive the reasons for making commands available before a self test has completed, the commands listed in this section are similar to those in the Section 5.6.8. One addition is the command that retrieves the actual command duration: TPM_GetCapability with the capability TPM_CAP_PROP_DURATION, because until this command is called, the software has no way of knowing how long a command will take.

Once the platform has completed its bootstrapping phase to the point where it has sufficient resources to deal with differing command durations (e.g., for proper user experience), it should call GetCapability: TPM_CAP_PROP_DURATION to get the actual values.

End of informative comment

1. Command Duration is defined as the time between the TPM's receipt of a "1" to TPM_STS_x.tpmGo and the TPM setting both TPM_STS_x.dataAvail and TPM_STS_x.stsValid fields to a "1".
2. The duration of the commands listed in Section 5.6.8 and the command TPM_GetCapability, with the capability TPM_CAP_PROP_DURATION, MUST be less than 750 ms.

5.6.6 Timeouts

Start of informative comment

The term timeout applies to timings between various states or transitions within the interface protocol. Timeout values are the purview of this specification and are not related to duration (see Section 5.6.5).

Because of the variations between implementations, it is not practical to specify timeout values that apply to all implementations. Efficient software must have the means to provide optimizations; therefore, software should be able to determine, with some level of granularity, when a state transition is expected to complete and when the software should determine the TPM has failed. These timings are called timeouts. The timeouts have been broken into four categories, each with a label. The amount of time for each timeout is obtained by calling the TPM_GetCapability command. To provide an initial value (e.g., for calling the TPM_GetCapability command) for generally expected behavior, a maximum for all commands that do not cause key generation is also specified.

Until software calls the TPM_GetCapability: TPM_CAP_PROP_TIS_TIMEOUTS, it should consider each of the timeout values as indicated in the "Default Timeouts" column I Table 13: Definition of Timeouts.

End of informative comment

1. There are four timeout values designated: TIMEOUT_A, TIMEOUT_B, TIMEOUT_C, and TIMEOUT_D. The time associated with each is obtained by calling TPM_GetCapability with the capability: TPM_CAP_PROP_TIS_TIMEOUTS. The TPM MUST return an array of UINT32 values each representing the number of microseconds for the associated timeout in response to a TPM_GetCapability command with a capability: TPM_CAP_PROP_TIS_TIMEOUTS.

2. The default timeouts and offsets for the array of timeouts SHALL be as shown in Table 13.

Table 13: Definition of Timeouts

Offset	TIMEOUT Label	Default Timeouts
[0]	TIMEOUT_A	750,000 microseconds
[1]	TIMEOUT_B	2,000,000 microseconds
[2]	TIMEOUT_C	750,000 microseconds
[3]	TIMEOUT_D	750,000 microseconds

5.6.7 TPM_Init

Start of informative comment

The command TPM_Init is not an actual command with a defined ordinal and set of parameters; rather, it is an indication to the TPM that the Static RTM is being reset and that the RTR and RTS should also be reset. On a PC Client, this is performed using a hardware-based signal.[6] For a TPM implementation using the TPM Packaging specified in Section6.7.1 TPM Packaging, TPM_Init is indicated by asserting the LRESET# or SPI_RST# pin. There is no requirement to use this packaging; therefore, it is up to the TPM manufacturer to define the hardware-based signal that performs this function.

End of informative comment

1. The TPM MUST implement a hardware-based signal for TPM_Init.
2. The assertion of the TPM_Init signal is defined to be TPM_Reset.
3. If the TPM uses the TPM Packaging specified in Section 6.7.1 TPM Packaging, this MUST be done by asserting the reset pin (LRESET# for LPC or SPI_RST# for SPI).
4. If the TPM does not use the TPM Packaging specified in Section6.7.1 TPM Packaging, the TPM Manufacturer MUST define the pin used for TPM_Init.

5.6.8 Self-Test and Early Platform Initiation

Start of informative comment

During the time-sensitive phase of a PC Client's startup procedure, only a small subset of the available commands is likely to be necessary. Therefore, this specification requires the TPM to perform any necessary self test on these commands required to make them available upon completion of TPM_Init.

The Design Principles documents require each platform specific specification state the maximum time a TPM can take to continue its self-test time. Due to variations in security requirements and implementations of TPM, it is difficult to mandate this to the satisfaction of all TPMs for all PC Client implementations. However, the generally accepted constraints of this platform's architecture and applications target the 1 to 2 second timeframe. PC

[6] This distinction is made because it is conceivable that other architectures might use other methods for performing this function.

Client platform manufacturers are advised to keep this particular aspect of the TPM's specification in mind when selecting a TPM for applicability to the platform's targeted use.

End of informative comment

1. After TPM_Init, a TPM MUST test all internal functions that are necessary to perform the following commands necessary for early boot operations. The following operations MUST be available after TPM_Init and before a call to TPM_ContinueSelfTest
   1.1. TPM_ORD_SHA1Start
   1.2. TPM_ORD_SHA1Update
   1.3. TPM_ORD_SHA1Complete
   1.4. TPM_ORD_SHA1CompleteExtend
   1.5. TPM_ORD_Extend
   1.6. TPM_ORD_Startup
   1.7. TPM_ORD_ContinueSelfTest
   1.8. TPM_ORD_SelfTestFull
   1.9. TPM_HASH_START / TPM_HASH_DATA / TPM_HASH_END
   1.10. TPM_ORD_NV_ReadValue for indices with the attributes TPM_NV_PER_AUTHREAD and TPM_NV_PER_OWNERREAD set to FALSE.
   1.11. TSC_ORD_PhysicalPresence
   1.12. TSC_ORD_ResetEstablishmentBit
   1.13. TPM_ORD_GetCapability
      1.13.1. The property TPM_CAP_PROPERTY, subcap property TPM_CAP_PROP_TIS_TIMEOUT MUST be supported during the early boot operations.
      1.13.2. Any other property or subcap property MAY be supported provided it does not cause a self test to be started.
2. The following operations MUST NOT cause the TPM to begin a self test:
   2.1. TPM_ORD_SHA1Start
   2.2. TPM_ORD_SHA1Update
   2.3. TPM_ORD_SHA1Complete
   2.4. TPM_ORD_SHA1CompleteExtend
   2.5. TPM_ORD_Extend
   2.6. TPM_ORD_Startup
   2.7. TPM_HASH_START / TPM_HASH_DATA / TPM_HASH_END
   2.8. TPM_ORD_NV_ReadValue for indices with the attributes TPM_NV_PER_AUTHREAD and TPM_NV_PER_OWNERREAD set to FALSE.
   2.9. TSC_ORD_PhysicalPresence
   2.10. TSC_ORD_ResetEstablishmentBit

2.11. TPM_ORD_GetCapability property TPM_CAP_PROPERTY, subcap property TPM_CAP_PROP_TIS_TIMEOUT

3. The maximum time to continue TPM self-test after receipt of TPM_ORD_ContinueSelfTest SHOULD be less than 1 second.

5.6.9 Input Buffer Size

Start of informative comment

Software must be aware of the maximum amount of data it can transfer to the TPM in one command. This is mostly for the NV Storage functions where relatively large amounts of storage area can be defined by the software. This does not prevent larger areas from being defined. It means, however, that if an area is defined that requires more than the input buffer size allowed to be transferred in one command (i.e., as specified in this section), the software must break up the write into smaller pieces. This is possible because the NV Write commands allow for an offset.

Note that there is no specified output buffer size. This is because the TPM will return an error on the command before exceeding its output buffer size.

The input buffer size specified below was arrived at by calculating a reasonably large command contained within a transport session.

This size is mandated as a minimum the TPM must provide to allow software to be written to a common environment. TPMs are allowed to provide larger input buffer size to increase performance. Software that would like to take advantage of this must call TPM_GetCapability with the capability TPM_CAP_PROP_BUFFER_SIZE to get the TPM's actual input buffer size.

End of informative comment

1. The TPM MUST support an input buffer size of at least 1101 bytes.

5.6.10 Errors

Start of informative comment

In general, there are four types of errors for the TPM, as outlined in the following cases:

1. Errors that the TPM detects and understands and that force it into Failure Mode:
   a. In this mode, the TPM responds correctly to all register reads or writes.
   b. The TPM provides a TPM-defined Response to security operations. This response should be one of the error return codes defined in the TPM Main Specification, e.g. TPM_FAILED_SELFTEST.
   c. The TPM allows certain TPM-defined transactions, e.g. TPM_GetTestResult, to return a response that indicates the particular error, or provides other TPM status information.
2. Errors that the TPM detects and that seem to be attacks on the hardware interface:
   a. The TPM completely stops responding and enters Shutdown because of these events.

i. The TPM may not respond to reads or writes of the TPM_STS_x, TPM_ACCESS_x register or any other register, but at minimum does not set TPM_STS_x.stsValid or TPM_ACCESS_x.tpmRegValidSts to "1".

ii. The TPM may not provide any response.

iii. All accesses to the TPM in this state should result in an Abort until a TPM reset has occurred.

3. Transmission or protocol errors:

a. TPM_STS_x fields and software behavior have been defined to both identify and correct overruns or underruns on both reads and writes.

4. Errors that the TPM does not detect, but cause it to hang or shutdown.

For case 1, Failure Mode, there is no need for a status field or interrupt, since the Response contains all the information that software needs to understand the TPM state. Case 1 may include things such as the RNG self-test failed.

For case 2, Shutdown, there is no need for a status field or interrupt since an LPC TPM does not respond to any cycles at all and an SPI TPM aborts all cycles. Reading FFh from TPM_ACCESS_x indicates this state.

For case 3, the interface has been defined with the needed status fields to detect overruns and underruns, and provides a mechanism to recover from those errors if they are transient. Software should time out after some number of retries.

Case 4 obviously needs no status field or other indication. Note that software may be able to determine that the TPM is in a hung or error state, even though the TPM cannot. For instance, if the TPM hangs in a microcode loop, then status fields would never be updated. Software could detect this case if it has sent a command and TPM_STS_x.dataAvail never went to a "1" for longer than the command duration.

The requirement for all errors is that system security or secrets cannot be compromised.

Therefore, this specification lumps all errors relating to the TPM into one of the first three categories. For instance, assume there is a long power glitch. The TPM can treat this like an attack and, for LPC TPM's, simply cease operation, and SPI TPM's abort all cycles, or it could go to the Error Mode and return error responses to all commands. The exact condition that forces the TPM into one error state or the other is vendor specific.

This specification defines the TPM's behavior for protocol or transmission errors. It is beyond the scope of this specification to define every hardware or software attack. It is within the scope of this specification to define the protocol for dealing with the errors.

As long as the TPM is in states 1-3 above, or in the working state, it meets the requirements of this specification. The TPM must not have any point where it allows secrets or a response other than an error to be returned when it knows there has been an error. It may use any of the above sequences to enforce this rule.

Since the transmission errors are taken care of by the protocol, the TPM has only two options for other errors: go into Failure Mode or to Shutdown. Since each vendor's TPM will have different physical implementations, there is no good way to precisely define each error and whether it should go into Failure Mode or Shutdown. Therefore, it is vendor specific whether a particular error causes Failure Mode or Shutdown Mode responses. The TPM Main Specification prescribes what responses the TPM must return when in Failure Mode.

Software has two cases to deal with. If the TPM has shutdown, this is detected by TPM_STS_x.commandReady or TPM_STS_x.dataAvail not being asserted within the appropriate timeout (as defined in sections 11.2.11 Access Register and 11.2.12 Status Register), and the platform should be rebooted. If the TPM returns Error Responses, then the software should already be designed to handle this case.

Since the recovery for Shutdown Mode is system reset and the recovery for Failure Mode is also system reset, there is no need to define in more detail how the TPM should handle each error. In the future, if an error has a more graceful recovery mechanism, then the specification will need to be more precise on how the TPM must handle the error. Today, the software stack will simply detect a timeout in the protocol and reset the system or it will detect that the TPM is only returning Error Responses for TPM commands and reset the system.

Note: Software attacks should never cause the TPM to enter Shutdown Mode. Shutdown Mode is intended to counter hardware attacks and should not persist through a TPM_INIT unless the hardware attack also persists.

End of informative comment

1. In the event of any error, the TPM MUST NOT compromise system security or secrets.
2. If the TPM detects an error:
   a. The TPM MUST respond correctly to Register Reads and Writes; i.e., it must either correctly read/write the register or do a bus abort; but it MUST not hang the bus.
   b. The TPM MAY respond with a defined TPM Error Response to Security Operations.
   c. The TPM MAY respond by entering Failure mode.
3. If the TPM detects a hardware attack, it MUST enter Shutdown Mode until a subsequent TPM_Init. Following TPM_Init, the TPM MUST clear Shutdown Mode unless the attack condition remains.

5.6.11 Access Register

Start of informative comment

The purpose of this register is to allow the processes operating at the various localities to share the TPM. The basic notion is that any locality can request access to the TPM by setting the TPM_ACCESS_x.requestUse field using its assigned TPM_ACCESS_x register address. If there is no currently set locality, the TPM sets current locality to the requesting one and allows operations only from that locality. If the TPM is currently at another locality, the TPM keeps the request pending until the currently executing locality frees the TPM. Software relinquishes the TPM's locality by writing a "1" to the TPM_ACCESS_x.activeLocality field. Upon release, the TPM honors the highest locality request pending. If there is no pending request, the TPM enters the "free" state.

There may be circumstances where the access to the TPM is "held" by either crashed or ill-behaved software. For this reason, the TPM_ACCESS_x.Seize field may be used. It is generally assumed that software executing at higher level localities is more trusted and less prone to crashing and better behaved at relinquishing the TPM. The TPM_ACCESS_x.Seize field allows higher-level localities to gain control of the TPM. This method, however, should be the exception rather than the common method for gaining access to the TPM.

The relationship between the internal flag TPM_PERMANENT_FLAGS->tpmEstablished and the interface field TPM_ACCESS_x.tpmEstablishment is inverted logic. Therefore, when one is FALSE the other is TRUE. This is because software accesses the TPM_PERMANENT_FLAGS->tpmEstablished field and expects "positive" logic, while hardware reads the TPM_ACCESS_x.tpmEstablishment and expects negative logic.

Software writing to the TPM_ACCESS_x register should set only one field to a "1" for each write. If a write to this register contains more than one field set to “1”, the behavior of this register is undefined in this specification and the behavior between TPM implementations may differ.

Software needs to consider that there is no timeout condition defined for the period between the release of one locality until the access to a subsequent locality is granted (i.e. TPM_ACCESS_x.activeLocality is set to "1"), as this process happens practically immediately from a Software point of view.

Note: The TPM_HASH_x sequence is independent of the Access register. Software does not need to use the Access Register to send the TPM_HASH_x commands, because the TPM_HASH_x commands assert Locality 4. As a result, the TPM must always be ready to accept these commands when there is no active locality.

End of informative comment

1. The TPM MUST implement the TPM_ACCESS_x register as documented in Table 14.
2. Any write operation to the TPM_ACCESS_x register with more than one field set to a "1" MAY be treated as vendor specific.
3. For each write, fields containing a "0" MUST be ignored.

Table 14: Access Register

Abbreviation:		TPM_ACCESS_x		
General Description:		Used to gain ownership of the TPM		
Bit Descriptions:				
7	Read Only	tpmRegValidSts	Default: 0	This bit indicates whether all other bits of this register contain valid values, if it is a "1".
6	Read Only	Reserved	Default: 0	MUST return “0”
5	Read/ Write	activeLocality	Default: 0	Read 0 = This locality is not active. Read 1 = This locality is active. Write 1 = Relinquish control of this locality
4	Read/ Write	beenSeized	Default: 0	Read 0 = This locality operates normally or is not active Read 1 = Control of the TPM has been taken from this locality by another higher locality while this locality had its TPM_ACCESS_x.activeLocality bit set. Write 1 = Clear this bit.
3	Write Only	Seize	Reads always return 0	A write to this field forces the TPM to give control of the TPM to the localtiy setting this bit if it is the higher priority locality.
2	Read Only	pendingRequest	Default: 0	Read 1 = some other locality is requesting usage of the TPM Read 0 = no other locality is requesting use of the TPM

Abbreviation:		TPM_ACCESS_x		
General Description:		Used to gain ownership of the TPM		
Bit Descriptions:				
1	Read/ Write	requestUse	Default: 0	Read 0 = This locality is either not requesting to use the TPM or is already the active locality Read 1 = This locality is requesting to use TPM and is not yet the active locality Write 1 = Request that this locality is granted the active locality
0	Read Only	tpmEstablishment	Default: 1	There are some special end cases (e.g., error conditions) where software needs to know if a Dynamic OS has previously been established on this platform. This bit performs this function.

Bit Field: tpmRegValidSts

Start of informative comment

If TPM_ACCESS_x.tpmRegValidSts is set, then all other fields [bits 0:6] of TPM_ACCESS_x are guaranteed to be correct.

If this field remains a "0" for longer than the period specified in section 5.6.6, Timeouts, Software may assume that the TPM is broken and should not use it.

End of informative comment

1. For any read of the TPM_ACCESS_x. register, the TPM MUST assert a wait state (either an LPC Bus Long Wait Sync or SPI wait cycle) until the field TPM_ACCESS_x.tpmRegValidSts contains a valid logical level (i.e., 0 or 1) which represents its true state/value.
2. For all other register fields, which will contain a valid logical level only when TPM_ACCESS_x.tpmRegValidSts = 1, the TPM MUST not return with TPM_ACCESS_x.tpmRegValidSts = 1 in response to a read if at least one of the fields contains an invalid logical level.
3. TPM_ACCESS_x.tpmRegValidSts MUST be set to "1" within the Reset Timing requirements specified in Section 6.6 Reset Timing.

Bit Field: Reserved

Start of informative comment

This field is reserved for future use.

End of informative comment

Writes to this field MUST be ignored. A read from this field MUST return 0.

Bit Field: activeLocality

Start of informative comment

TPM_ACCESS_x.activeLocality has 3 functions:

1. It is used as an indicator to the Software to show whether the locality currently reading the TPM_ACCESS_x register is the active locality

2. It is used by the Software that is currently accessing the TPM at the active locality to relinquish control of the TPM by writing a "1" to TPM_ACCESS_x.activeLocality

3. It can be used by the Software that has a currently pending request (to obtain the active locality) to cancel this pending request by writing a "1" to TPM_ACCESS_x.activeLocality.

The time from the request by a specific locality to become the active locality until the active locality is granted may vary depending on whether there are already other localities active. After the request of a locality to become the active locality, TPM_ACCESS_x.activeLocality will be a "1" within TIMEOUT_A if there is no other locality active at this time, otherwise TPM_ACCESS_x.activeLocality will be a "1" only after the other locality has relinquished control of its locality.

End of informative comment

Read:

1. If the requesting locality is the active locality, the TPM MUST return this TPM_ACCESS_x.activeLocality = 1.
2. If the requesting locality is not the active locality the TPM MUST return this TPM_ACCESS_x.activeLocality = 0.

Write:

1. If a write occurs at the current active locality:
    a. On a write of a "1" to the active locality's TPM_ACCESS_x.activeLocality field the TPM MUST:
        i. Clear TPM_ACCESS_x.activeLocality field to "0" for the current locality.
        ii. Relinquish control of the TPM for the current locality.
    b. If there are pending requests from other localities, the TPM MUST transfer control to the locality with the highest priority and set TPM_ACCESS_x.activeLocality to "1" for the new active locality. Note: This locality becomes the new active locality.
2. If a write of "1" to TPM_ACCESS_x.activeLocality occurs at a locality which is not the current active locality and the locality performing the write has its TPM_ACCESS_x.requestUse set to "1" (e.g., there is a pending request for this locality which has not been granted), the TPM MUST cancel the pending request from this locality.
3. If the requesting locality is not the active locality and its TPM_ACCESS_x.requestUse is "0":
    a. A write to this TPM_ACCESS_x.activeLocality MUST be ignored.
4. TPM_ACCESS_x.activeLocality MUST be set to "1" within TIMEOUT_A after TPM_ACCESS_x.requestUse has been set to "1" if the TPM is in the "free" state.
5. If there is another locality active at the time when a subsequent locality sets its TPM_ACCESS_x.requestUse field to "1", this TPM_ACCESS_x.activeLocality MUST be set to "1" within TIMEOUT_A after the other locality has relinquished control of its locality.

For the handling of changing locality during command execution and aborts see Section 5.6.3 Aborts

Start of informative comment

For example if Locality 2 is the current active locality and Locality 0 sets TPM_ACCESS_0.requestUse = 1, then Locality 0 has a pending request (i.e. TPM_ACCESS_0.requestUse is "1") and all other localities (1 to 4) have TPM_ACCESS_x.pendingRequest set to "1".

If now Locality 3 sets TPM_ACCESS_3.requestUse = 1 now Locality 3 also has a pending request with TPM_ACCESS_0.requestUse and TPM_ACCESS_3.requestUse being "1" and all other localities (0, 1, 2, 3 and 4) having TPM_ACCESS_x.pendingRequest set to "1".

Now Localities 0, 1, 2, 3, and 4 have TPM_ACCESS_x.pendingRequest set to "1" and localities 0 and 3 have their TPM_ACCESS_x.requestUse set to "1".

As soon as Locality 2 relinquishes control of the TPM by setting TPM_ACCESS_x.activeLocality to a “1”, the TPM automatically transfers the control of the TPM to Locality 3 (because of the locality priority rules) and the pending request remains for Locality 0.

Now localities 1 to 4 have their TPM_ACCESS_x.pendingRequest set to "1" and Locality 0 has TPM_ACCESS_0.requestUse set to "1".

If now Locality 0 decides not to maintain the request to use the TPM, it sets TPM_ACCESS_0.activeLocality = 1 and consequently TPM_ACCESS_0.requestUse is cleared to "0" and TPM_ACCESS_x.pendingRequest of the localities 1 to 4 are cleared to "0" as well.

End of informative comment

Bit Field: beenSeized

Start of informative comment

If a locality is the active locality, Software can use this field to determine whether the active locality has been taken away (i.e. seized) by another, higher priority locality and therefore the seized locality needs to abort an entire task and restart it after it has obtained the active locality again. This field can be cleared by the seized locality.

End of informative comment

1. TPM_ACCESS_x.beenSeized MUST be set to a "1" when the active locality gets seized.
2. A write of a "1" to TPM_ACCESS_x.beenSeized MUST clear this field to a "0".

Bit Field: Seize

Start of informative comment

The seize operation is a mechanism as a "last line of defense", if rogue Software does not relinquish control of a TPM and another, higher locality needs to obtain control of the TPM.

In this case, the Software of the higher locality (i.e. seizing locality) sets the TPM_ACCESS_x.Seize field to a “1” and then polls on TPM_ACCESS_x.activeLocality until this field returns a "1" (i.e. successful seize operation). At this point, the Software operating at the lower locality will be informed about the successful seize operation by TPM_ACCESS_x.beenSeized being set to a "1".

After the successful seize operation, the Software of the seizing locality reads the TPM_STS_x.commandReady field:

If the TPM_STS_x.commandReady field is a “0”, software should write a “1” to the field and then poll until it becomes a “1”,

If the TPM_STS_x.commandReady field is set and TPM_STS_x.burstCount > 0, software may immediately write the command to the TPM.

Seize operations from Locality 0 are ignored by the TPM, unless the TPM is in Locality None, since this operation has no real meaning for Locality 0. If the TPM has been accessed using the legacy I/O addressing, a Locality 0 process may perform a Seize in the same manner that any higher locality may Seize the TPM from a lower locality.

For example, if Locality 0 is the currently active locality and Locality 1 writes a "1" to TPM_ACCESS_1.Seize, the TPM must clear the TPM_ACCESS_0.activeLocality field of locality 0, i.e. remove control of the TPM from Locality 0 and set TPM_ACCESS_x.beenSeized to “1”. Consequently, the TPM must abort any currently executing command and stop accepting commands from Locality 0 as Locality 0 no longer has control of the TPM.

End of informative comment

1. If the write to TPM_ACCESS_x.Seize occurs from a locality of higher priority than the current locality:
    a. The TPM MUST clear the TPM_ACCESS_x.activeLocality fields for any active locality of lower priority than the locality seizing the TPM.
    b. The TPM MUST NOT change the state of the TPM_ACCESS_x.requestUse field for any locality except the one writing this field.
    c. The TPM MUST set TPM_ACCESS_x.activeLocality to a "1", clear the TPM_ACCESS_x.requestUse field to a "0" for the locality writing this field, and, if there are no other active requests, clear the TPM_ACCESS_x.pendingRequest field to “0” for all other localities.
    d. The TPM MUST abort any command that is currently in process.
2. If the write occurs from a locality that is equal to or lower than the current locality the TPM MUST ignore the write.
3. A read to TPM_ACCESS_x.Seize MUST return “0”.

Bit Field: pendingRequest

Start of informative comment

This field indicates whether a locality other than the currently active locality has requested to become the active locality. Software can use this field to determine if it should relinquish control of the TPM so that the other locality can use it.

End of informative comment

1. When a locality writes a "1" to its TPM_ACCESS_x.requestUse, the TPM MUST set TPM_ACCESS_x.pendingRequest to a "1" for all other localities.
2. If there are no pending requests, when the locality that has written a "1" to TPM_ACCESS_x.requestUse has obtained the control of the TPM as signified by TPM_ACCESS_x.activeLocality, TPM_ACCESS_x.pendingRequest for all other localities MUST be cleared to "0".
3. When the locality with a pending request writes a "1" to its TPM_ACCESS_x.activeLocality field (i.e. cancels the pending request):
    a. If there are no other pending requests, the TPM MUST clear all TPM_ACCESS_x.pendingRequest field to "0".

b. If there is one other pending request, the TPM MUST clear the TPM_ACCESS_x.pendingRequest field to “0” for the locality with the active request.

c. If there are multiple pending requests, the TPM MUST NOT clear any TPM_ACCESS_x.pendingRequest field to “0”.

4. The TPM MUST ignore writes to this field.

Bit Field: requestUse

Start of informative comment

This field is used to request access to the TPM as the active locality. Software can write a "1" to this field when it needs to get control of the TPM. After the request is issued, Software must wait until its request to become the active locality is granted.

This field may only be cleared by writing a “1” to TPM_ACCESS_x.activeLocality for the requesting locality. Note: Writing a zero to TPM_ACCESS_x.requestuse does not clear the pending request for this locality. See bit field: activeLocality for more information.

End of informative comment

1. When a locality writes a "1" to TPM_ACCESS_x.requestUse, the TPM MUST set this field to "1" for the requesting locality. **Note:** If the TPM_ACCESS_x.requestUse is already set to “1”, the TPM MUST ignore writes of “1” to this field.
2. When the locality that has set its TPM_ACCESS_x.requestUse has been granted control of the TPM as signified by TPM_ACCESS_x.activeLocality set to “1”, the TPM MUST clear the TPM_ACCESS_x.requestUse field to "0" for the requesting locality.
3. When a locality cancels a pending request, signified by writing a “1” to TPM_ACCESS_x.activeLocality, the TPM MUST clear this field to “0” for the requesting locality.
4. The TPM MUST ignore writes of “0” to TPM_ACCESS_x.requestUse.

Bit Field: tpmEstablishment

Start of informative comment

TPM_ACCESS_x.tpmEstablishment is a register field which represents the inverted state of the TPM_PERMANENT_FLAGS->tpmEstablished. This allows low level software (such as primitive drivers and hardware) to read the state of the TPM_PERMANENT_FLAGS->tpmEstablished without issuing a TPM command. The reason this register field is the inverted state of the flag (i.e., negative logic) is to allow systems without TPMs to indicate, using this register, that no Dynamical OS has been launched. Since the default state of the flag is the value ‘0’, the default state of the register field is ‘1’. Reading from a device that doesn’t exist (there is no device to claim the read request) returns a value of all ones, therefore a read access to the TPM’s access register when there is no TPM present returns as if no Dynamic OS has been established.

End of informative comment

1. If the TPM_ACCESS_x.tpmRegValidSts is set to “1”, any read of the TPM_ACCESS_x.tpmEstablishment field MUST reflect the inverted state of the TPM_PERMANENT_FLAGS->tpmEstablished.
2. If TPM_ACCESS_x.tpmRegValidSts is cleared to “0” (i.e., TPM_ACCESS register is not valid):

a. TPM_ACCESS_x.tpmEstablishment MAY be a "0".

b. TPM_ACCESS_x.tpmEstablishment MUST NOT be a "1" unless that is the correct value of the field.

3. TPM_ACCESS_x.tpmEstablishment MUST be a "1" until the first TPM_HASH_START interface command is received by the TPM either from the first power-on of the TPM or after TPM_PERMANENT_FLAGS->tpmEstablished was reset by the corresponding TPM command TSC_ResetEstablishmentBit.

4. TPM_ACCESS_x.tpmEstablishment MUST be duplicated across Localities 0-4.

5.6.12 Status Register

Start of informative comment

The TPM_STS_x commandReady field is functionally overloaded. If there is no command being executed, a write to this field is an indicator to the TPM that it must prepare to receive a command. If there is a command being executed, a write to this field serves as an abort of that command. If a command has completed and the results have been read, a write to this field allows the TPM to free internal resources (including the Read and Write FIFOs) and proceed with background or other processes allowed during idle time. A TPM may be designed in a manner that allows the first write to this field to clear and free the TPM's resources and make it ready to receive a command.

Software must be prepared to send two writes of a "1" to this field: the first to indicate successful read of all the data, thus clearing the data from the ReadFIFO and freeing the TPM's resources, and the second to indicate to the TPM it is about to send a new command. The time between receiving the data from a command and sending the first write to this field should be very short to allow TPMs that perform background processing to proceed. The time between the first write and the second indicates the beginning of a new command is arbitrary.

Software may be written such that the second write to this field is only necessary if the TPM does not respond with a ready after the first write. In this case, the software, after writing a "1" to this field indicating the receipt of the data, may query this field. If the TPM sets this field to a "1" indicating its readiness to receive a command, the software may proceed to send the command without writing a "1" to this field.

End of informative comment

5.6.12.1 TPM Status Register States

For each write to this register, there MUST be only one field set to a "1". If the TPM receives a write with more than one field set, the TPM MUST ignore the entire cycle. For each write, fields containing "0" are ignored.

The TPM is considered to be in one of the following defined states:

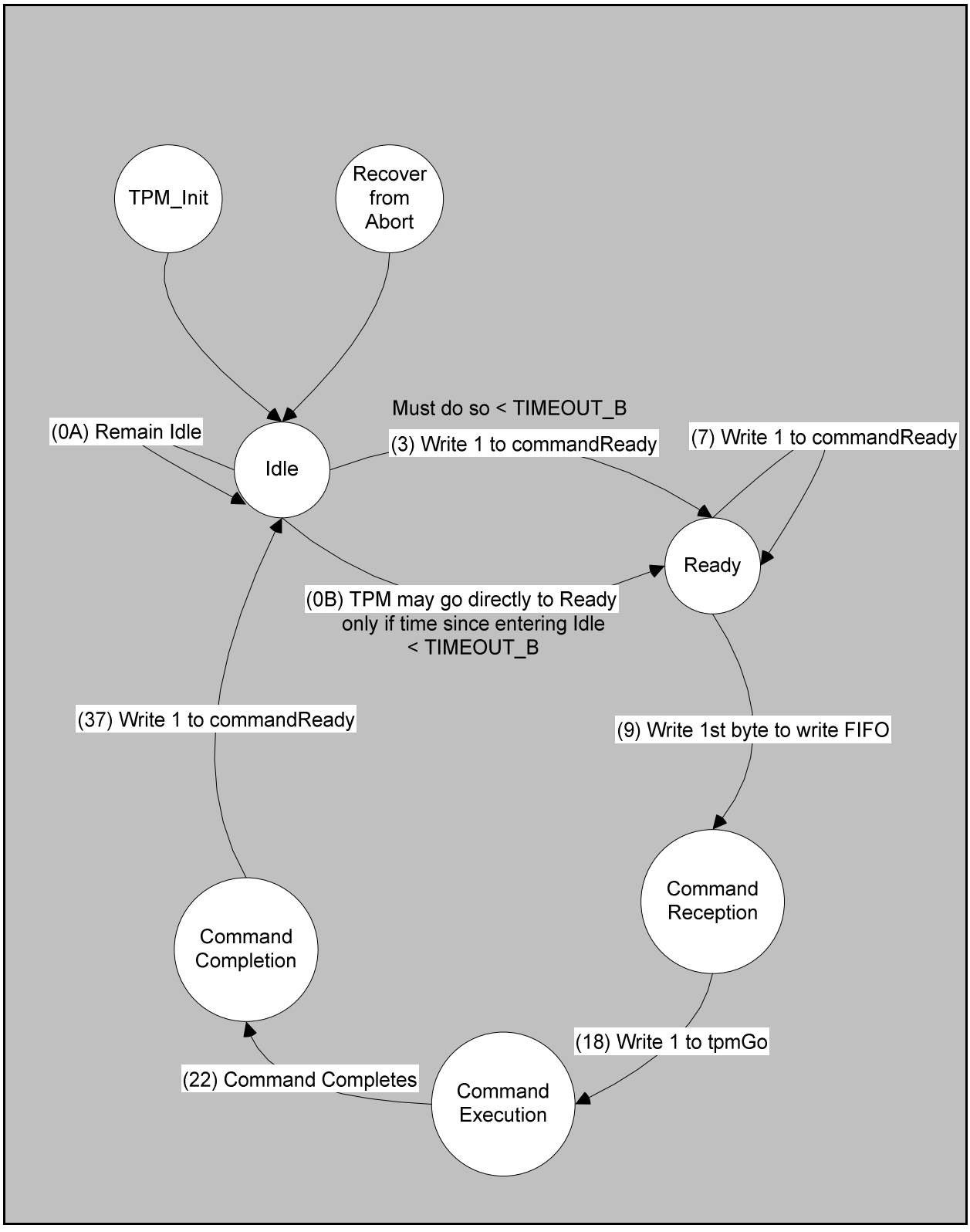
1. *Command Reception* occurs between the write of the first byte of a command to the WriteFIFO following a ready state and the receipt of a write of "1" to TPM_STS_x.tpmGo.

2. *Command Execution* occurs after receipt of a "1" to TPM_STS_x.tpmGo and the TPM setting TPM_STS_x.commandReady:dataAvail to a "1", unless the command is aborted.
3. *Command Completion* occurs after completion of a command (indicated by the TPM setting the TPM_STS_x.commandReady:dataAvail to a "1" and before a write of a "1" by the software to TPM_STS_x.commandReady:commandReady).
4. *Idle* is any time after Command Completion followed by the write of a "1" by the software to TPM_STS_x.commandReady, following locality change, or an abort. Idle is the initial state of TPM upon completion of TPM_Init.
5. *Ready* is any time the TPM is ready to receive a command, as indicated by TPM_STS_x.commandReady being set.

Start of informative comment

The following informative diagram is derived from the above normative statements. It is informative and only for illustrating diagrammatically the above TPM states and their transitions. The numbers in parentheses reference the states represented by row numbers in Table 18: State Transition Table.

End of informative comment



1590

Figure 3 State Transition Diagram

5.6.12.2 Bus Access of the Status Register

For any read from the TPM_STS_x. register, the TPM MUST assert zero or more wait states (either an LPC Bus Long Wait Sync or SPI wait cycles) until all fields in the register, except TPM_STS_x.dataAvail and TPM_STS_x.Expect, contain a valid logical level (i.e., 0 or 1) which represents their true state/value. For the register fields TPM_STS_x.dataAvail and TPM_STS_x.Expect, which will contain a valid logical level only when TPM_STS_x.stsValid=1, the TPM MUST not return with TPM_STS_x.stsValid=1 in response to a read if either TPM_STS_x.dataAvail (while reading results) or TPM_STS_x.Expect (while sending a command) contains an invalid logical level, respectively.

The TPM MUST implement the Status Register as documented in Table 15.

Table 15 describes the bit-map of the fields of the status register.

Table 15: Status Register

Abbreviation:		TPM_STS_x		
General Description:		Contains general status details		
Bit Descriptions:				
23:8	Read Only	burstCount	Default = number of consecutive writes that can be done to the TPM	Indicates the number of bytes that the TPM can return on reads or accept on writes without inserting wait states on the bus. Valid indicator: NA .
7	Read Only	stsValid	Default: 0	This field indicates that TPM_STS_x.dataAvail and TPM_STS_x.Expect are valid. Valid indicator: N/A
6	Read/ Write	commandReady	Default: 0	Read of '1' indicates TPM is ready, Write of '1' causes TPM to transistion its state. Valid indicator: N/A
5	Write Only	tpmGo	Reads always return 0	After software has written a command to the TPM and sees that it was received correctly, software MUST write a "1" to this field to cause the TPM to execute that command. Valid indicator: N/A
4	Read Only	dataAvail	Default: 0	This field indicates that the TPM has data available as a response. When set to "1", software MAY read the ReadFIFO. The TPM MUST clear the field to "0" when it has returned all the data for the response. Valid indicator: TPM_STS_x.stsValid = '1'
3	Read Only	Expect	Default: 0	The TPM sets this field to a value of "1" when it expects another byte of data for a command. It clears this field to a value of "0" when it has received all the data it expects for that command, based on the TPM size field within the packet. Valid indicator: TPM_STS_x.stsValid = '1'
2	Read Only	selfTestDone	Default: 0	This field indicates that the TPM has completed all self-test actions following a TPM_ContinueSelfTest command. Read of '0' indicates self-test is not complete. Read of '1' indicates self-test is complete.
1	Write Only	responseRetry	Reads always return 0	Software writes a "1" to this field to force the TPM to re-send the response. Reads MUST return "0" Valid indicator: N/A
0	Read Only	Reserved (0)	Reads always return 0	

Below is a detailed description of the fields defined above:

Bit Field: BurstCount:

Start of informative comment

It's helpful to understand burstCount by first explaining it in the context of an example implementation. For example, a TPM's firmware processes commands once they are received by the TPM. Software however, does not directly interact with the TPM's firmware. Rather, it sends and receives data via hardware registers called a data FIFO. During a command send phase, software writes command data directly to the data FIFO which the firmware reads from the FIFO. There is no relationship between the size or amount of data sent to the data FIFO (from either side) and the size of the command or the command's response. The data FIFO, therefore, can be thought of as a hardware buffer between the software and the TPM's firmware. The value in the burstCount field is simply the number of

bytes that can be written or read from the data FIFO (i.e., the hardware buffer) at any one time. It will likely require multiple writes (for command send) or multiple reads (for response reads) to and from the data FIFO in order to send a command and read a response.

It is expected that the data FIFO is sufficiently fast so that, provided there is room (during command send) and bytes available (during a read response) in the data FIFO, all writes and reads will occur without any wait states. Therefore, burstCount is defined as the number of bytes that can be written to or read from the data FIFO by the software without incurring a wait state.

Software should read this field and write or read the number of bytes indicated. Once that number of bytes has been written to the TPM, the TPM may set burstCount = 0. Software must wait while burstCount = 0, by polling on burstCount, until the TPM sets burstCount > 0. Once burstCount > 0 software resumes writing or reading data up to the new burstCount value.

Again, there is no relationship between the size of the data FIFO and the value in the burstCount field, versus the size of the command or response data. On a command send, software must not pad to the data FIFO nor must it read more response data than indicated by the command's response size value. For example, if after sending several data FIFO's worth of command data to the TPM, there are seven bytes left to send, even if burstCount = 16, software must still only send seven bytes. Conversely, on response read if there are only three bytes left of the response to read, software must only read three bytes from the data FIFO even if burstCount = 16.

This field may be dynamic or static as indicated by TPM_INTF_CAPABILITY_x.burstCountStatic.

A dynamic burstCount field reports a changing number of bytes that can be read or written. For example, on command send, as data is written to the data FIFO the burstCount field is decremented by the number of bytes written indicating there are fewer bytes available to write into the data FIFO. As the firmware reads the data out of the data FIFO the burstCount field is incremented indicating there are more bytes available in the FIFO. Conversely, on response read, as software reads data from the data FIFO the burstCount field decrements indicating there are fewer bytes in the data FIFO available to read without incurring a wait state. As firmware writes response data to the data FIFO the burstCount field increments indicating there are more bytes available to read without incurring a wait state.

A static burstCount field reports a fixed number of bytes that can be read or written. Software reads burstCount and must keep track of that value. Once software begins to write, for command send, or read, for response read, the TPM sets burstCount = 0 until the fixed value is written or read. Only after the fixed number of bytes have been written or read will the burstCount field contain a nonzero value for software to read. Note that in this case, burstCount is a fixed value from TPM_Init, allowing software to read burstCount once when the software is first initialized and to save the value, and to use the saved value until the next TPM_Init. In this case after the initial read of burstCount, software only needs to look at whether burstCount is a zero or non-zero value.

NOTE: It takes roughly 330 ns per byte transfer on LPC. 256 bytes would take 84 us. Chipsets may not be designed to post this much data to LPC; therefore, the CPU itself is stalled for much of this time. Sending 1 kB would take 350 us. Therefore, even if the TPM_STS_x.burstCount field is a high value, software should be interruptible during this

period. For SPI, assuming 20MHz clock and 64B transfers, it would take about 120 usec to move 256B of data. Sending 1kB would take about 500 usec. If the transactions are done using 4B at a time, then it would take about 1 msec. to transfer 1kB of data.

End of informative comment

1. For dynamic burstCount (I.e., TPM_INTF_CAPABILITY_x.burstCountStatic == 0):
    a. For command send phase
        i. If the data FIFO is not ready to receive data from the software without incurring an LPC wait state, the TPM MUST set burstCount = 0.

        Note: this rule applies not just to the beginning of the command send phase but at any time during the command send phase. E.g., after several writes to the data FIFO have occurred the software may fill the data FIFO completely because the firmware cannot read and process the data as fast as software can write it.

        ii. When the TPM is ready to receive data in the data FIFO, the TPM MUST set burstCount equal to the number of bytes software can write without incurring an LPC wait state.
            1. As data is received from software into the data FIFO, the TPM MUST decrement burstCount.
            2. As the TPM (e.g., firmware) reads data from the data FIFO the TPM MUST increment burstCount.
    b. For response read phase
        i. If the data FIFO is not ready to return data to the software without incurring an LPC wait state, the TPM MUST set burstCount = 0.

        Note: this rule applies not just to the beginning of the response read phase but at any time during the response read phase. E.g., after several reads from the data FIFO have occurred the software may read all the data in the data FIFO (i.e., empty the data FIFO) because the firmware cannot place response data in the FIFO as fast as software can read it.

        ii. When the TPM is ready for software to read data from the data FIFO, the TPM MUST set burstCount equal to the number of bytes software can read without incurring an LPC wait state.
            1. As software reads data from the data FIFO, the TPM MUST decrement burstCount.
            2. As the TPM (e.g., firmware) writes data to the data FIFO the TPM MUST increment burstCount.
2. For static burstCount (I.e., TPM_INTF_CAPABILITY_x.burstCountStatic == 1):
    a. For command send phase
        i. If the data FIFO is not ready to receive data from the software without incurring an LPC wait state, the TPM MUST set burstCount = '0'.
        ii. When the data FIFO is ready to receive data from the software without incurring an LPC wait state, the TPM MUST set burstCount equal to the

maximum number of bytes that can be transferred by software to the TPM without incurring LPC wait sync.

iii. Upon receipt of the first command data byte the TPM MUST set burstCount = 0. The burstCount field MUST remain set = '0' until the indicated number of bytes have been sent by software to the data FIFO.

iv. Once the TPM has received the indicated number of bytes in the data FIFO, the TPM must set burstCount equal to the first value that was sent to the software.

Note: for static burstCount, burstCount is a fixed value and MUST NOT change after TPM_Init until the next TPM_Init.

b. For response read phase

i. If the data FIFO is not ready to return data to the software without incurring an LPC wait state, the TPM MUST set burstCount = '0'.

ii. When the data FIFO is ready for software to read data without incurring an LPC wait sync the TPM MUST set burstCount equal to the maximum number of bytes that can be read by software without incurring LPC wait sync.

iii. Upon software's read of the first response data byte, the TPM MUST set burstCount = 0. The burstCount field MUST remain = 0 until software has read the indicated number of bytes from the data FIFO.

iv. Once the software has read the indicated number of bytes from the data FIFO, the TPM MUST set burstCount equal to the first value that was sent to the software.

Note: for static burstCount, burstCount is a fixed value and MUST NOT change after TPM_Init until the next TPM_Init.

3. Following a write to TPM_STS_x.responseRetry or an Abort operation, any value previously read from the TPM_STS_x.burstCount field is invalid until TPM_STS_x.DataAvail = 1.

4. Timeout:

a. For command send: After TPM_STS_x.commandReady is set to 1, TPM_STS_x.burstCount MUST be non-zero within the time specified by TIMEOUT_D.

b. For response read: After TPM_STS_x.DataAvail == 1, TPM_STS_x.burstCount MUST be non-zero within the time specified by TIMEOUT_D.

5. The TPM MUST be designed to report the correct count even though there is a time delay in returning the 2 bytes on the LPC bus.

Start of informative comment:

There are many ways to ensure the correct count is always reported. A few examples are below:

Return 0x00 for the upper byte in all cases. This limits the field to 255 bytes, but that is a sufficiently large number that polling on this register every 255 bytes is insignificant in terms of performance.

Guarantee that the count does not change between the read of the first byte and the read of the second byte. This could be done if the hardware updates the count field at the time of the previous read or write and does not change it until the next read or write. Alternatively, it could be done if hardware latches both bytes of the count that is returned on the read of the first byte, and returns the latched second byte on the next read. In this case, if there is a read or write of the data before the next read of the TPM_STS_x.burstCount, then the latch is reset.

Use static burstCount.

Note that there could be the case where internally the TPM is processing the FIFO and TPM_STS_x.burstCount is dynamic, whereby the count is changing even though there are no LPC bus transactions. In this case, the read of the low byte might not be synchronized with the high byte – hardware must not allow this condition.

End of informative comment

Bit Field: stsValid

Start of informative comment

TPM_STS_x.stsValid gates reads to the fields TPM_STS_x.dataAvail and TPM_STS_x.Expect. If TPM_STS_x.stsValid is not set, then TPM_STS_x.dataAvail and TPM_STS_x.Expect are not guaranteed to be correct. If the TPM does not support the stsValid Interrupt, software that is using TPM_STS_x.dataAvail or TPM_STS_x.Expect must poll on TPM_STS_x register until TPM_STS_x.stsValid is set. Software should not use the contents of the status register if this field is 0.

End of informative comment

1. The TPM MUST set the TPM_STS_x.stsValid field within TIMEOUT_C after the last data cycle to this register is received.
2. The TPM MUST not set the TPM_STS_x.stsValid field to 1 unless either the TPM_STS_x.Expect field is valid in the command Completion state or TPM_STS_x.dataAvail field is valid in the command Reception state

Bit Field: commandReady

Start of informative comment

TPM_STS_x.commandReady is a dual-function field. It is used by the TPM to indicate readiness to receive a command. Software uses this field to initiate a command sequence with the TPM.

Note on software usage of this field:

Software should be designed such that it checks this field before sending any new data to the TPM data FIFO. This allows software to know the TPM's state. Software should never send data to the TPM when this field is set to 0, indicating the TPM is not ready. After

software has successfully received data from the TPM, it should write a 1 to this field to signal to the TPM that the response was correctly received.

End of informative comment

Read:

1. The TPM is in the *Ready* state when this field is set to 1.
    a. The TPM MUST not enter the *Ready* state unless both the ReadFIFO and WriteFIFO are empty.
    b. The TPM MUST clear this field to 0 when the first byte of data is received into the WriteFIFO.
2. The TPM is not in a *Ready* state when this field is set to 0.

Write:

1. A write of "0" to this field MUST be ignored.
2. Upon a write of a "1" to this field.
    a. When in the Ready state:
        i. The TPM MUST ignore this write and remain in the *Ready* state.
    b. When in the Idle state:
        i. The TPM MUST enter the *Ready* state within the time TIMEOUT_B.
    c. When in the Command *Reception* state:
        i. The TPM MUST treat this as an abort according to Section 5.6.3.
    d. When in the Command *Completion* state:
        i. The TPM MUST clear the ReadFIFO and the WriteFIFO.
        ii. The TPM MUST enter either the *Idle* state or the *Ready* state.
        iii. If the TPM does not enter the *Ready* state within time TIMEOUT_B, it MUST enter the *Idle* state.
    e. When in the Command *Execution* state:
        i. The TPM MUST cause the currently executing command to be aborted according to Section 5.6.3.

Bit Field: tpmGo

Start of informative comment

This field is used by Software to tell the TPM to execute the received command. Execution of the command may take from several seconds to minutes for certain commands, such as key generation. Software should confirm the TPM has received the complete command by reading the TPM_STS_x.stsValid and TPM_STS_x.Expect fields. This field is write-only.

End of informative comment

1. The TPM MUST execute the received command on a write of 1 to this field.
    a. The TPM MUST ignore a write of 0 to this field and MUST NOT return an error.

2. The TPM MUST return 0 on a read request.

Bit Field: dataAvail

Start of informative comment

The TPM sets this field when it is ready to return the response. The validity of this field is determined by TPM_STS_x.stsValid, as defined in normative text for the Bit Field: stsValid.

To detect overruns/under runs, software SHOULD read TPM_STS_x.dataAvail before it reads what it thinks is the last byte of the response. If TPM_STS_x.dataAvail is "1", then there is at least 1 more byte to read. Software reads the last byte and re-reads TPM_STS_x.dataAvail. In this case, TPM_STS_x.dataAvail should be "0"; since, if things are working correctly, there is no more data to return. If TPM_STS_x.dataAvail is still "1", then the TPM has more data to return, and software is out of sync with the hardware; therefore, software should set TPM_STS_x.responseRetry to force the TPM to resend the response.

If a read of TPM_STS_x.dataAvail returns a "0" before software reads the last byte, the TPM thinks it has no more data to return, while software still expects 1 more byte. In this case, software MUST set TPM_STS_x.responseRetry to force the TPM to resend the response.

If the DataAvailInt interrupt is not supported, software must poll on the TPM_STS_x register until TPM_STS_x.dataAvail is set.

End of informative comment

1. The TPM MUST not set this field to a 1 unless it has completed command execution and data is ready to be read.
   a. The TPM MUST set this field when data is present in the ReadFIFO.
   b. The TPM MUST set this field when software writes to TPM_STS_x.responseRetry.
   c. The TPM MUST make data available in the ReadFIFO when this field is set.
   d. The TPM MUST set the DataAvailInt interrupt, if the interrupt is supported, after setting this field.
2. The TPM MUST set this field to zero if it is either not ready to transmit data or has returned the last byte of data.
   a. The TPM MUST clear this field to "0" when the ReadFIFO is empty because either the TPM has sent all the data or is not ready to send data
   b. The TPM MUST clear this field to "0" when software sets TPM_STS_x.commandReady = 1 even though the response data has not been read.
3. This field MUST be valid if TPM_STS_x.stsValid is set.

Bit Field: Expect

Start of informative comment

This field is set by the TPM when it expects to receive data from software. The validity of this field is determined by TPM_STS_x.stsValid.

The TPM will set this field once it starts receiving data. The field will stay set until the TPM receives at least 10 bytes, as this is the smallest valid command length. The TPM will use

the first 10 bytes to determine the actual length of the command. If the length is longer than 10 bytes, this field will stay set until the TPM receives the full command.

Software should examine the Expect field before and after sending the last byte to ensure the TPM has received the full command. If the Expect field is set to 1 before software sends the last byte of the command, there is no error condition. Software should send the last byte and check Expect again. If the Expect field is set to a 0, the command has been successfully received. If the value of the Expect field is 0 prior to software sending the last byte or is 1 after software sends the last byte, an error has occurred and software should restart the command.

End of informative comment

1. The TPM MUST set this field to "1" when in the command Reception state and MAY set this field to "1" when in the Ready state.
2. The TPM MUST NOT set this field to "1" in any other state.
3. The TPM MUST clear this field to "0" when in any other state.
4. This field MUST be valid if TPM_STS_x.stsValid is set.

Bit Field: selfTestDone

Start of informative comment

This field is set by the TPM to indicate the status of the TPM's self-test following receipt of a TPM_ContinueSelfTest command. The field will be at '0' as long as the TPM has capabilities remaining to be tested. Once the TPM completes its self-test, it sets this bit to a '1'. This field is always valid.

End of informative comment

1. The TPM MUST set this field to "1" when all of the actions required to complete the command TPM_ContinueSelfTest are done.
2. The TPM MUST NOT set this field to "1" unless it has completed all of the actions required by TPM_ContinueSelfTest.
3. The TPM MUST ignore writes to this field and MUST NOT return an error.

Bit Field: responseRetry

Start of informative comment

This field is set by Software to force the TPM to resend a response without sending a command. This may occur if a TPM exceeds the timeout defined for the response. Software should implement a retry counter and set this field if the retry counter is less than its threshold. This field is write-only.

End of informative comment

1. The TPM MUST resend the last response on a write of 1 to this field.
   a. The TPM MUST ignore writes of 0 to this field and MUST NOT return an error.
2. The TPM MUST return 0 on a read request

5.6.13 Data FIFO Register

Start of informative comment

This register is the port used by the TPM to return data and status to software. Return packets for commands are multiple bytes but are read by software in 1-byte increments. Software should read the TPM_STS_x.burstCount field to determine how many consecutive bytes it can read. Software should read the TPM_STS_x.burstCount field for better general system performance.

As the TPM_HASH_* Interface commands are independent of the TPM_ACCESS_x register, processes calling these commands should not poll TPM_STS_x.burstCount and should send data to the TPM using only the interface protocols and bus speeds.

End of informative comment

Table 16: Data FIFO Register

Abbreviation:			TPM_DATA_FIFO_x
General Description:			Data port for TPM
Bit Descriptions:			
7:0	Read/Write	Default: undefined	Reads to this register return Command Response data. Writes to this register contain Command Send Data

1. When TPM_STS_x.stsValid is set to "1" and TPM_STS_x.dataAvail is cleared to "0", the TPM SHOULD return FFh to any read request to the Data FIFO register.
2. The TPM MUST NOT drop a write on the bus when it is not able to accept it. Instead, it MUST insert one or more wait states, according to the interface protocol (e.g. wait-sync the LPC bus or an SPI wait cycle according to Section 6.4.5 Flow Control).

5.7 Interface Capability

Start of informative comment:

Starting in this version of this specification, the SPI interface supports an additional data buffer at offset 0x80 to allow for larger data transfers than the existing FIFO at offset 0x24. Bits 9 and 10 have been defined for this version of the specification to allow the TPM to communicate the maximum data transfer size it supports. The TPM may support larger transactions on LPC.

End of informative comment

1. The DataTransferSizeSupport field indicates the maximum transfer size supported by the TPM.
2. If the TPM supports only legacy transfer sizes, the DataTransferSizeSupport field MUST be set to '00'.
   a. Reads and Writes MUST only be accepted at the offset for the DATA_FIFO.
   b. TPM MUST abort transactions to the XDATA_FIFO.
3. If the TPM supports any DataTransferSizeSupport value other than '00':
   a. The TPM MUST support the XDATA_FIFO in addition to the DATA_FIFO.
   b. The TPM MUST accept any data transfer size up to and including the size indicated by DataTransferSizeSupport. **Note:** This includes data transfers from 1-4B, which may be written to either the DATA_FIFO or the XDATA_FIFO.

Table 17: Interface Capability

Abbreviation:		TPM_INTF_CAPABILITY_x		
General Description:		Provides information of which interrupts that particular TPM supports. The TPM MUST implement this register.		
Bit Descriptions:				
31	Read Only	Reserved	Default: Vendor Defined	The value of this field is not specified.
30:28	Read Only	Interface Version	Default: Defined by hardware	000: Interface 1.21 or earlier 001: Reserved 010: Interface 1.3 011-111: Reserved
27:11	Read Only	Reserved	Reads always return 0	
10:9	Read Only	DataTransferSizeSupport	Default: 00 but Defined by hardware	Indicates what transaction size the TPM supports for Data transfers 11 = TPM supports 64B maximum transfer size (note, support for 64B transfer size also indicates support for legacy, 8B and 32B transfers) 10 = TPM supports 32B maximum transfer size (includes support for legacy and 8B transfers) 01 = TPM suports 8B maximum transfer size (includes support for legacy) 00 = TPM supports legacy transfer size only

8	Read only	BurstCountStatic	Default: Defined by hardware	Indicates whether the TPM_STS_x.burstCount field is dynamic or static 1 = TPM_STS_x.burstCount is static 0 = TPM_STS_x.burstCount is dynamic
7	Read only	CommandReadyIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.commandReadyEnable 1 = supported 0 = not supported
6	Read Only	InterruptEdgeFalling	Default: Defined by hardware	Falling edge interrupt support 1 = supported 0 = not supported
5	Read Only	InterruptEdgeRising	Default: Defined by hardware	Rising edge interrupt support 1 = supported 0 = not supported
4	Read Only	InterruptLevelLow	Mandatory: MUST be 1	Low level interrupt support. This interrupt trigger is mandatory. 1 = supported 0 = not allowed
3	Read Only	InterruptLevelHigh	Default: Defined by hardware	High level interrupt support 1 = supported 0 = not supported
2	Read Only	LocalityChangeIntSupport	Mandatory: MUST be 1	Corresponds to TPM_INT_ENABLE_x.localityChangeIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed
1	Read Only	stsValidIntSupport	Default: Defined by hardware	Corresponds to TPM_INT_ENABLE_x.stsValidIntEnable 1 = supported 0 = not supported
0	Read Only	dataAvailIntSupport	Mandatory: MUST be 1	Corresponds to TPM_INT_ENABLE_x.dataAvailIntEnable. This is a mandatory interrupt. 1 = supported 0 = not allowed

5.8 Status Field State Transitions

Table 18 shows the changes in status fields based on the command or action done to the TPM. Notice this is not a state transition table covering the states defined in Section 5.6.12 rather a table describing how the status fields change based on initial condition and action taken. The following rules apply to Table 18.

1. There MAY be intermediate status field states, where a command has finished, but TPM_STS_x.dataAvail is not yet set. Software is expected to poll until the appropriate status field is set.
2. The fields in the table represent values only when TPM_STS_x.stsValid = 1. Transitional states when TPM_STS_x.stsValid = 0 are neither captured nor represented in this table.
3. This table applies only to status field states where a locality has already been selected and no change in locality is performed.
4. The statements in the column labeled “Action Taken” are ***informative*** when shaded and are derived from normative statements contained within the definitions of the TPM_STS_x register in Section 5.6.12, the description of the FIFO handling in section 11.2.1 Handling Command FIFOs, the description of the command transmission in section 11.2.1.1 Command Send and the description of the response reception in section 11.2.1.2 Data Availability. If there is an inconsistency between this column and the statements within normative definitions of the TPM_STS.x registers, the normative statements within definitions of the TPM_STS.x registers take precedence. The statements made in unshaded text and delimited by the phrases: “Start of normative comment” and “End of normative comment” are normative. Note, this is a reversal of the standard notation.
5. Normal transitions are highlighted in yellow and are indexed to the state transitions illustrated in Figure 3 State Transition Diagram.
6. In all cases in Table 18 where Idle is the next state, the TPM is allowed to transition directly to Ready effectively via transition 0.B. This simplifies the table by not having to show two ending states possibilities. When making a transparent transition to the Ready state, the TPM is not required to indicate it is or has been in the Idle state to the software, therefore, making this transition transparent to the software.
7. The following abbreviations are used in Table 18:

Label	Bit Definition
C/R	TPM_STS_x.commandReady
D/A	TPM_STS_x.dataAvail
E	TPM_STS_x.Expect
TG	TPM_STS_x.tpmGo
R/R	TPM_STS_x.responseRetry
N/C	No Change to this field from previous state
—	No TPM access for the corresponding data element
X	Either 0 or 1. The TPM is allowed to maintain this field as either value for this

state. Software MUST be capable of managing the TPM if either case is implemented.

Table 18: State Transition Table

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x					Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E	C/R	TG	R/R	Write Data	Read Data	TPM State	C/R	D/A	E	*Informative*
0.A	Idle	0	0	0	—	—	—	—	—	Idle	0	0	0	***Start of normative comment*** I_T = Time since entering the Idle state If TPM is Idle and I_T >= TIMEOUT_B Then: TPM MUST remain in the Idle state (i.e., continue executing in this in this row/state) until commanded to change by the software (i.e., via state transition in row 3). Else: (i.e., I_T < TIMEOUT_B) TPM MAY transition to the Ready state (i.e., transition to row 0B) ***End of normative comment***
0.B	Idle	0	0	0	—	—	—	—	—	Ready	1	0	X	This specification allows a TPM to be implemented such that the software never sees the Idle state. This transition codifies and enables that behavior. ***Start of normative comment*** I_T = Time since entering the Idle state If TPM is Idle and I_T >= TIMEOUT_B Then: TPM MUST NOT enter the Ready state (i.e., it MUST NOT execute the state transition in this row and MUST remain Idle in Row 0A). Else: (i.e., I_T < TIMEOUT_B) TPM MAY transition to the Ready state (i.e., execute the transition in this row). If the TPM performs this transition, it is not required to indicate that it is, or has been, in the Idle state; rather it is allowed to appear as if it went directly from the state prior to the Idle state to the Ready state transparently to the software. ***End of normative comment***
1	Idle	0	0	0	0	0	1			Idle	0	0	0	There is no response to retry. No state change.
2	Idle	0	0	0	0	1	0			Idle	0	0	0	There is no command to execute. No state change.
3	Idle	0	0	0	1	0	0			Ready	1	0	X	This is the typical state change resulting from the software's request to send a command.
4	Idle	0	0	0				Write data		Idle	0	0	0	TPM drops write, since it is not expecting data. If the software was sending a command and still has data to write, then this is an error in transmission and the software must re-transmit the command.
5	Idle	0	0	0					Read data	Idle	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
6	Ready	1	0	X	0	0	1			Ready	1	0	X	No effect on TPM, since TPM_STS_x.commandReady indicates that the response has been read successfully.
7	Ready	1	0	X	0	1	0			Ready	1	0	X	Causes no change to the TPM, since there is no command to process.
8	Ready	1	0	X	1	0	0			Ready	1	0	X	No effect on TPM, since it is ready to receive commands.
9	Ready	1	0	X				Write first byte		Reception	0	0	1	Clear TPM_STS_x.commandReady on first byte.
10	Ready	1	0	X					Read Data	Ready	1	0	X	No effect on TPM. TPM returns FFh.
11	Reception	0	0	1	0	0	1			Reception	0	0	1	There is no response to retry. No state change.
12	Reception	0	0	1	0	1	0			Reception	0	0	1	TpmGo is not valid at this time. TPM ignores this state transition.

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x					Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E	C/R	TG	R/R	Write Data	Read Data	TPM State	C/R	D/A	E	*Informative*
13	Reception	0	0	1	1	0	0			Idle	0	0	0	The command being sent to the TPM is aborted.
14	Reception	0	0	1				Write more data other than last byte		Reception	0	0	1	
15	Reception	0	0	1				Write last byte		Reception	0	0	0	Good transmission if the software has just sent the last byte. If either software has more than one more byte to send or if expect = 1 and software has not more data to send, this is a transmission error and the software must resend the command.
16	Reception	0	0	1					Read data	Reception	0	0	1	TPM returns FFh.
17	Reception	0	0	0	0	0	1			Reception	0	0	0	No response to retry.
18	Reception	0	0	0	0	1	0			Execution	0	0	0	This is the normal transition from sending the command to the start of execution of the command.
19	Reception	0	0	0	1	0	0			Idle	0	0	0	Aborts the command sent.
20	Reception	0	0	0				Write		Reception	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
21	Reception	0	0	0					Read	Reception	0	0	0	Read 0xFF.

#	Present State of TPM_STS_x				Fields / Data Written to TPM_STS_x or TPM_DATA_FIFO.x					Next State & Result / Reason				Action Taken
	TPM State	C/R	D/A	E	C/R	TG	R/R	Write Data	Read Data	TPM State	C/R	D/A	E	*Informative*
22	Execution	0	0	0	—	—	—	—	—	Completion	0	1	0	Upon command completion, TPM sets TPM_STS_x.dataAvail to a 1.
23	Execution	0	0	0	0	0	1			Execution	0	0	0	There is no response to retry. No state change.
24	Execution	0	0	0	0	1	0			Execution	0	0	0	The TPM is already executing a command. No state change.
25	Execution	0	0	0	1	0	0			Idle	0	0	0	The executing command is aborted.
26	Execution	0	0	0				Write data		Execution	0	0	0	TPM drops write, since it is not expecting data. If the software was sending a command and still has data to write, then this is an error in transmission and the software must re-transmit the command.
27	Execution	0	0	0					Read data	Execution	0	0	0	TPM returns FFh, since TPM_STS_x.dataAvail is not set.
28	Completion	0	1	0	0	0	1			Completion	0	1	0	TPM resets ReadFIFO pointers and start sending the response from the first byte.
29	Completion	0	1	0	0	1	0			Completion	0	1	0	Causes no change to the TPM, no new command to execute.
30	Completion	0	1	0	1	0	0			Idle	0	0	0	Aborts command.
31	Completion	0	1	0				Write data		Completion	0	1	0	No effect on TPM, it is not accepting a command.
32	Completion	0	1	0					Read data other than last byte	Completion	0	1	0	TPM returns data.
33	Completion	0	1	0					Read last byte	Completion	0	0	0	Good transmission, since TPM_STS_x.dataAvail = 0.
35	Completion	0	0	0	0	0	1			Completion	0	1	0	TPM resets ReadFIFO pointers and start sending the response from the first byte.
36	Completion	0	0	0	0	1	0			Completion	0	0	0	TpmGo is not valid at this time. TPM ignores this state transition.
37	Completion	0	0	0	1	0	0			Idle	0	0	0	This is the typical state change resulting from the software's indication that it received the results of the command, and the TPM may proceed to the Idle state and, depending on implementation, proceed directly to the Ready state.
38	Completion	0	0	0				Write		Completion	0	0	0	Write is not expected. Drop write. TPM ignores this state transition.
39	Completion	0	0	0					Read	Completion	0	0	0	Read 0Xff.
40	Any				More than 1 field set on any status write					undefined				If a write to this register contains more than one "1" field, the behavior of the status register is undefined in this specification and the behavior between TPM implementations may differ.

5.9 LPC Interrupts

Start of informative comment

The method for asserting interrupts uses the Serial-IRQ (SIRQ) protocol for interrupts. The protocol emulates the set of individual hardware signals using time division multiplexing between frames. An understanding of the SIRQ protocol is critical to the TPM implementer. The direct assertion of the SIRQ line does not, in itself, signal an interrupt. The assertion of the interrupt is a combination of the change in the SIRQ signal during a time slot designated for that interrupt number. The state (level, edge, high, low) is expressed as the state of the SIRQ line during the assigned time slot over a series of frames.

The TPM must be designed to support asserting any of the IRQ[0:15]. Certain platforms may not support certain IRQs being assigned to the TPM; therefore, the TPM must be capable of asserting any of the 16 possible IRQs. The TPM must not assert PIRQ[A:D] in the SIRQ stream.

There is a capability register that allows each platform to indicate to the TPM which interrupts the platform supports.

Because use of the TPM is non-preemptive (except for the special case of Seize) and the TPM is single threaded, there is no issue with regard to sharing or colliding interrupts across localities. For example, if one locality starts a TPM operation, it cannot release the TPM to another locality until the pending TPM operation completes. However, if one locality (e.g., Locality 0) starts a long TPM operation, then turns control to another locality (e.g., Locality 2) before the long operation completes (and of course does not relinquish the TPM, which would cause an abort), the second locality (e.g., Locality 2) would not know what the interrupt is for. This situation is outside the purview of this specification and negotiation of interrupt handling is done by the software.

The TPM reports all schemes it supports in the Interrupt Capabilities register bits 3-6. The software selects the scheme using the Interrupt Enable register bits 3-4. If the TPM supports only one scheme, bits 3 and 4 may be read only and return the value of the implemented scheme.

When an event occurs that causes the TPM to signal an interrupt, the TPM must set the appropriate fields in the TPM_INT_STATUS_x register. If the TPM has not already sent an interrupt, the “0” to “1” transition of a field in the TPM_INT_STATUS_x register must cause the TPM to assert the appropriate interrupt per the SIRQ protocol. The interrupt service routine will read the TPM_INT_STATUS_x register to determine the cause and take appropriate action. When the interrupt has been serviced, the interrupt service routine must do an End-of-Interrupt (EOI) to the I/O APIC to re-arm the TPM’s interrupt in the I/O APIC. Then the interrupt service routine must also send a TPM_EOI to the TPM to allow it to send new interrupts.

The TPM must not issue another interrupt until it has received its TPM_EOI message (see below), even if new events occur that should cause an interrupt. The TPM should set the appropriate field in TPM_INT_STATUS_x, but the actual assertion of the interrupt will only occur after the TPM_EOI. If the interrupt handler detects multiple fields set in TPM_INT_STATUS_x, it may handle all the causes and clear multiple status fields. This means that an interrupt may be handled without actually causing a new interrupt.

The TPM_EOI to the TPM is writing a "1" to the field in the TPM_INT_STATUS_x register that corresponds to the type of interrupt just handled. Software may write multiple fields if it has handled multiple interrupt causes at one time.

The following informative sections are added for clarification. They should not change functionality.

If there are multiple fields set in the Interrupt register, and software does not clear all the interrupts, then the TPM must issue another interrupt after it sees the TPM_EOI, which is a write to the Interrupt register.

The software must not change the state of the TPM_INT_ENABLE_x globalIntEnable flag while an interrupt is active.

For example: if after the write to the Interrupt register, there are fields still set, the TPM issues another interrupt. If software writes all the fields of the Interrupt register, so that after the write the register = "0", no new interrupt would be generated.

This covers the case that software handles one interrupt at a time and then returns. It also covers the case that software handles all the interrupts it knows about, so writes multiple fields into the Interrupt register, but a new interrupt is flagged between the time software read the Interrupt register and the time it wrote the TPM_EOI.

Application Note:

Many commands respond immediately so during normal operation the driver, after sending a command, should poll the TPM for a response keeping the interrupts masked. If the driver determines that the TPM will not be able to respond immediately, it will stop polling the TPM and unmask the appropriate set of interrupts. If the driver does this, there is a possibility of a race condition between the time the interrupt is unmasked and the state being checked becomes true. Therefore, after unmasking the interrupt(s,) the driver should poll the TPM one more time.

End of informative comment

1. The TPM MUST support the following interrupts:
   a. TPM_INT_STATUS_x.localityChangeIntOccured
   b. TPM_INT_STATUS_x.dataAvailIntOccured
2. The TPM MAY support the following interrupts:
   a. TPM_INT_STATUS_x.stsValidIntOccurred
   b. TPM_INT_STATUS_x.commandReadyIntOccured
3. The TPM MUST support asserting any of the IRQ[0:15]. The TPM MUST NOT assert PIRQ[A:D] in the SIRQ stream.
4. The TPM MUST support low level interrupts, defined in Table 21, and MAY support the other interrupts. The TPM MUST report all schemes it supports in the Interrupt Capabilities register.
5. The TPM MUST set the appropriate field indicating the cause of the interrupt in the TPM_INT_STATUS_x register.
6. Once an interrupt is asserted, the TPM MUST NOT assert another interrupt until it receives a TPM_EOI even if new events occur that should cause an interrupt.

7. If the TPM supports only one scheme, bits 3 and 4 MAY be read-only and return the value of the implemented scheme.
8. The TPM MUST maintain interrupts as inactive during any change to the TPM_INT_ENABLE_x globalIntEnable and while TPM_INT_ENABLE_x globalIntEnable is 0.
9. The TPM has only one interrupt assigned to it, so interrupt settings for one locality MUST be applied to all localities.

5.9.1 Interrupt Enable

Table 19: Interrupt Enable

Abbreviation:				TPM_INT_ENABLE_x
General Description:				Enables specific interrupts and has the global enable. The TPM MUST implement this register.
Bit Descriptions:				
31	Read/ Write	globalIntEnable	Default:0	1 = Interrupts controlled by individual bits 0= All interrupts disabled. Cleared to "0" on reset.
30:8		Reserved	Reads always return 0	
7	Read/ Write	commandReadyEnable	Default: 0	1 = Enabled 0 = Disabled
6:5		reserved	Reads always return 0	Note to readers and future editors: This displacement of the enable fields (i.e. TPM_INT_ENABLE_x.commandReadyEnable not being adjacent to the other enable fields) here and in the tables below was done because the TPM_INT_ENABLE_x.commandReadyEnable field was added late in the draft cycle of this release (1.2). Some TPM manufacturers could not change their implementation in a timely manner if we moved the TPM_INT_ENABLE_x.typePolarity fields.
4:3	Read/ Write	typePolarity	Default: 01	00 = High level 01 = Low level 10 = Rising edge 11 = Falling edge
2	Read/ Write	localityChangeIntEnable	Default: 0	1 = Enabled 0 = Disabled
1	Read/ Write	stsValidIntEnable	Default: 0	1 = Enabled 0 = Disabled
0	Read/ Write	dataAvailIntEnable	Default: 0	1 = Enabled 0 = Disabled

5.9.2 Interrupt Status

Table 20: Interrupt Status

Abbreviation:	TPM_INT_STATUS_x
General Description:	Shows which interrupt has occurred. The TPM MUST implement this register.
Bit Descriptions:	

31:8		Reserved	Reads always return 0	
7	Read / Write 1	commandReadyIntOccured	Default: 0	When “1”, indicates the TPM_STS_x.commandReady field transitioned from 0 to 1. Writing a “1” to this field clears the interrupt. Writing a “0” to this field has no effect.
6:3		reserved	Reads always return 0	
2	Read / Write 1	localityChangeIntOccured	Default: 0	When “1”, indicates the locality change interrupt occurred. This interrupt is caused whenever any locality moves from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality whenever this transition had been delayed due to another locality having TPM_ACCESS_x.activeLocality set. Note that if the TPM has no TPM_ACCESS_x.activeLocality set when TPM_ACCESS_x.requestUse is written, the TPM MUST move directly from TPM_ACCESS_x.requestUse to TPM_ACCESS_x.activeLocality without causing the interrupt. Writing a “1” to this field clears the interrupt (i.e., a TPM_EOI). Writing a “0” to this field has no effect.
1	Read / Write 1	stsValidIntOccurred	Default: 0	This interrupt indicates a “0” to “1” transition on TPM_STS_x.stsValid. Writing a “1” to this field clears the interrupt. Writing a “0” to this field has no effect.
0	Read / Write 1	dataAvailIntOccured	Default: 0	This interrupt indicates that TPM_STS_x.dataAvail transitioned from a “0” to a “1”. This “0” to “1” transition occurs when the command has been completed and there is a Response to be read. This transition MUST only occur when both TPM_STS_x.dataAvail and TPM_STS_x.stsValid fields are “1”. Writing a “1” to this field clears the interrupt. Writing a “0” to this field has no effect.

5.9.3 Interrupt Vector

Table 21: Interrupt Vector

Abbreviation:			TPM_INT_VECTOR_x
General Description:			Contains the SIRQ value. The TPM MUST implement this register.
Bit Descriptions:			
7:4	Reserved (0)	Reads always return 0	Must return “0” on read; writes have no meaning.
3:0	sirqVec	Default: 0	A value of “0” means SIRQ is disabled and SIRQ is tristated. The SIRQ channel used by TPM can be from 1 to 15.

6. Part 4: TPM Hardware Interface

6.1 Locality Usage per Register

Start of informative comment

Table 22 shows which cycles must be accepted by TPM. The following rules apply:

End of informative comment

1. If TPM_ACCESS_x.activeLocality setting changes when a command is executing, the TPM MUST abort the currently executing command.

Table 22: Register Usage Based on Locality Setting

Register	TPM_ACCESS_x.activeLocality Set for ThisLocality		TPM_ACCESS_x.activeLocality Set for Some Other Locality		No TPM_ACCESS_x.activeLocality Set	
	READ	WRITE	READ	WRITE	READ	WRITE
TPM_STS_x	TPM returns correct value	Fields updated	TPM returns FFh	TPM Ignore the write	TPM returns FFh	TPM Ignore the write
TPM_INT_ENABLE_x	TPM returns correct value	Field updated	TPM returns correct value	TPM Ignore the write	TPM returns correct value	TPM Ignore the write
TPM_INT_VECTOR_x	TPM returns correct value	Field updated	TPM returns correct value	TPM Ignore the write	TPM returns correct value	TPM Ignore the write
TPM_INT_STATUS_x	TPM returns correct value	Interrupt cleared	TPM returns correct value	TPM Ignore the write	TPM returns correct value	TPM Ignore the write
TPM_INTF_CAPABILITY_x	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register	TPM returns correct value	Read-only register
TPM_ACCESS_x	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated	TPM returns correct value	Fields updated
TPM_DATA_FIFO_x	TPM returns correct data	TPM accepts data and command	TPM returns FFh	TPM Ignore the write	TPM returns FFh	TPM Ignore the write
Configuration registers– 0F00h to 0FFFh	TPM returns correct value	Fields updated	TPM returns correct value	TPM Ignore the write	TPM returns correct value	TPM Ignore the write
TPM_HASH_START	TPM returns FFh	TPM accepts command	TPM returns FFh	TPM Ignore the write	TPM returns FFh	TPM accepts (and sets TPM_ACCESS_x .activeLocality for Locality 4)
TPM_HASH_DATA	TPM returns FFh	TPM accepts data	TPM returns FFh	TPM Ignore the write	TPM returns FFh	TPM Ignore the write
TPM_HASH_END	TPM returns FFh	TPM accepts command and clears TPM_ACCESS_x .activeLocality for Locality 4	TPM returns FFh	TPM Ignore the write	TPM returns FFh	TPM Ignore the write

6.2 TPM Legacy I/O Space and TPM 1.2 Memory Mapped Space

Start of informative comment

This section applies only to LPC TPMs.

Typically, TPMs designed to work with TPM Main Specification, Version 1.1b used addressing that is described in this section as legacy addressing. In the TPM Main

Specification, Version 1.1b there was no mandate or recommendation regarding addressing. This TIS does mandate a specific addressing scheme, however, it is recognized that some environments may be required to provide legacy addressing to allow backward compatibility. The required addressing scheme takes into account the set of known addresses used. This address mapping scheme allows an LPC TPM to be used in an environment where compatibility with a TPM 1.1b is needed.

The 1.1 TPM has traditionally been logically located in I/O address space. It has used I/O addresses for its index and data register such as e.g. offsets 2Eh&2Fh, 4Eh&4Fh, 7Eh&7Fh and EEh&EFh. Devices that have fixed addresses in the legacy I/O space often conflict with other devices that have fixed addresses at the same offset. Therefore, the following scheme is used to map legacy registers into memory space.

The TPM vendor, if implementing legacy functionality, must implement a BADD pin and thus must provide two legacy I/O address range options, each consisting of an index register at the base address and a data register at the base address plus 1. If the BADD pin is strapped high, the TPM defaults to its lower address range. If the pin is strapped low, the TPM defaults to its higher address range. Selectable address range choices, which may be utilized are e.g. (BADD high / BADD low) 7Eh&7Fh / EEh&EFh or 2Eh&2Fh / 4Eh&4Fh.

The TPM would accept either an I/O access to e.g. 2Eh, 4Eh, 7Eh or EEh (depending on the TPM and its strapping) or a Locality Read or Locality Write cycle to FED4_0F80h as an access to the Legacy 1 register. Note that the registers have been extended from 8 bits to 16 bits with the addition of the Legacy1b and Legacy2b locations. Note that this scheme does not define separate registers (e.g. at I/O address 2Eh and at memory-mapped address FED4_0F80h), but one physical register that may be addressed in two different ways.

Note: In some TPM implementations, the access method may be determined by which type of access occurs following assertion of LPC_RESET#, i.e. if the first access to the TPM is done using the I/O interface, the memory-mapped interface will be disabled until after the TPM receives a subsequent Reset.

End of informative comment

Note: This section applies only to LPC TPMs.

1. An LPC TPM MAY be used in chipsets which do not implement the memory-mapped TPM-Read or TPM-Write cycles, but access the TPM via I/O address space, called “TPM legacy I/O”. If an LPC TPM implements legacy addressing, it MUST support a BADD pin.
2. While all localities are released, any locality can be chosen. When Locality None is the active locality, which by definition is the least privileged locality (see section 9.1 TPM Locality Levels), any locality change to a higher privileged locality (i.e. 0 to 4) MUST cause an abort (see section 5.6.3 Aborts).
3. The TPM MUST NOT accept cycles on the I/O ports if any locality is set.

6.2.1 Legacy LPC Cycles for TPM Interface

Start of informative comment

This section only applies to TPM implementations using the LPC interface. This functionality is deleted for SPI TPMs.

The implementations of the TPM built to Main Specification 1.1b typically use LPC I/O cycles. An LPC TPM may continue to use its legacy port.

The existing port has a locality of 0 regarding its capabilities with respect to the PCRs but has a locality of Locality None with respect to the way the TPM_ACCESS_x register works.

The TPM 1.1 has ports in I/O space for legacy software to use. To support the legacy software, if the TPM supports legacy ports, the TPM must accept cycles on the I/O ports any time that no TPM_ACCESS_x.activeLocality field is set. The TPM must not accept cycles on the I/O ports if any locality is set.

Legacy software may be unaware of the Request and Seize functions. If the TPM supports legacy ports, it must also support the Request and Seize functions as defined in Section 5.6.11 Access Register to insure interoperability.

1. The TPM may continue to use its legacy port.

2. If the TPM uses its legacy port it must accept legacy LPC cycles if and only if the TPM_ACCESS_x.activeLocality field is not set.

3. If the TPM uses its legacy port, it must implement the Request and Seize functions as defined in Section 5.6.11 Access Register.

Table 23 shows the mapping that may be implemented to support legacy addressing.

Table 23: Legacy Port Usage

Name	Use	Access	I/O Address	System Memory Address	TPM 16-bit Address Using the New LPC TPM Cycles	Usage
Legacy1	First legacy address	Read/Write	2Eh, 4Eh, 7Eh, EEh, or other	FED4_0F80h	0F80h	Vendor defined
Legacy1b	Not used today	Read/Write	Additional 8 bits for Legacy1 register	FED4_0F84h	0F84h	Vendor defined
Legacy2	Second legacy address	Read/Write	2Fh,r 4Fh, 7Fh, EFh, or other	FED4_0F88h	0F88h	Vendor defined
Legacy2b	Not used today	Read/Write	Additional 8 bits for Legacy2 register	FED4_0F8Ch	0F8Ch	Vendor defined

Addresses not listed above but that are in the FED4_0F80h to FED4_0F8Fh range are reserved and MAY either return all 0's or ignore the least significant 2 bits of the address and alias to a legitimate address. For example, FED4_0F86h can return all 0's or alias to FED4_0F84h and return the data at that location.

End of informative comment

6.3 TPM LPC Hardware Protocol

Start of informative comment

This specification addresses 1.2 compliant TPM's which make use of the LPC bus for interconnecting to the platform, as there are specific protocol requirements for TPM's using LPC. The definition of specific LPC requirements does not preclude the use of other interfaces on a 1.2 compliant TPM. Future versions of this specification may address other bus interfaces.

End of informative comment

1. If a TPM implements an LPC interface as the method of connecting to the trusted process, it MUST implement the LPC bus per the requirements of the LPC Interface Specification. A link may be found to the specification in Section 15 References.
2. If a TPM implements an LPC interface, the TPM MAY use the LPC CLKRUN# protocol for mobile platforms.
3. If a TPM implements an LPC interface, the TPM MUST be designed such that the LPCPD# pin may be strapped high to disable the LPCPD# protocol.

6.3.1 LPC Locality Cycles for TPM Interface

Start of informative comment

This section only applies to TPM implementations using the LPC interface.

This specification defines two LPC locality cycles, TPM-Write and TPM-Read, which were added for communication between the chipset and the TPM. On the LPC bus, with the exception of the START field, these cycles are identical to I/O cycles. These locality cycles are an additional flag to the TPM (beyond addressing) that the cycles are intended for the TPM as locality commands. These commands can only be generated by a trusted process, e.g. the chipset.

End of informative comment

See Section 5.2 TPM Locality Levels for rules and restrictions on using the standard vs. Locality LPC cycles.

By definition, the Locality None level is lower than Locality 0. If the TPM supports Locality None and Locality None is the active locality, any TPM access request from Locality 0-4 is a higher locality priority. In this case, the TPM MUST respond to Locality 0-4 register writes to TPM_ACCESS_x.requestUse and TPM_ACCESS_x.Seize per the requirements documented in section 11.3.11 Access Register.

6.3.1.1 TPM-Write LPC Locality Cycle

Start of informative comment

Table 24 shows the TPM-Write locality cycle format. It is similar to the existing LPC I/O write.

If the CPU attempts to write more than 1 byte at a time to the TPM, the chipset must break this up into multiple cycles of 1 byte each to consecutive addresses.

End of informative comment

Table 24: LPC Locality Cycle TPM-Write for Accessing the TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read locality cycles.
CYCTYPE + DIR	0010	Same as used for standard LPC I/O Write
ADDR	See Description	Four nibbles. Same as the standard LPC I/O Write.
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR
SYNC		Standard SYNC field for an I/O Write
TAR		Standard LPC TAR

6.3.1.2 TPM-Read LPC Locality Cycle

Start of informative comment

Table 25 shows the TPM-Read locality cycle format. It is similar to the existing LPC I/O read.

If the CPU attempts to read more than 1 byte at a time to the TPM, the chipset must break this up into a series of 1-byte reads to consecutive addresses.

End of informative comment

Table 25: LPC Cycle TPM-Read for Accessing the TPM

Field	Value for Bits [3:0]	Description
START	0101	Previously this was a reserved value. It is now allocated for TPM-Write and TPM-Read.
CYCTYPE + DIR	0000	Same as used for standard LPC I/O Read
ADDR	See Description	Same as for TPM-Write
TAR		Standard LPC TAR
SYNC	Standard	Standard SYNC field for an I/O Read
DATA-Low	DIGEST low nibble	
DATA-High	DIGEST high nibble	
TAR		Standard LPC TAR

6.4 SPI Hardware Protocol

Start of informative comment

There were a number of goals that guided architecture of SPI hardware protocol and flow control for the TPM. These assumptions are as follows:

- The TPM must have a dedicated SPI ChipSelect# (CS#).
- Only the chipset is allowed to assert the TPM CS# signal. This means further that the TPM's CS# can only be connected to the south bridge.
- The SPI protocol should not break existing drivers or software.
- The TPM Interface Specification 1.21 defines all registers as having a size of 4B or less. This register size is maintained for compatibility with software. This applies to the TPM_HASH_X function, which may be generated by HW, because the TPM's data register is only 4B. No additional registers are defined for registers that might be greater than 4B. Future definitions of SW may support 8B or 64B data registers. The SPI flow control and protocol are defined to allow for 8B and 64B data transactions in case they are added later. This allows for future improvements in SPI throughput. An example would be a 64B data register at offset 0x80. The 4B data register is always implemented and available to SW to maintain backwards compatibility.
- In the future there may be uses where large amounts of data, for instance 4kb, need to be passed to the TPM.

End of informative comment

6.4.1 Clocking

Start of informative comment

The LPC interface has a free-running clock, but the clock defined by the SPI interface only runs when an SPI transaction occurs. The TPM, to maintain backwards compatibility, is not required to have an external free-running clock. TPM manufacturers may choose to support an external clock in their implementations. The TPM must, however, generate whatever clock source it needs to support internal command processing, as this command processing will likely take place when the SPI bus is idle and the SPI clock is not running. Additionally, the TPM's timer/tick counter is based on this internal clock and does not require external clock support.

The SPI bus clock frequency may vary based on implementation and/or type of device attached. The TPM default clock frequency for PC Client platforms is defined to be 24MHz, but in future might be higher frequencies.

The SPI bus is a shared-bus architecture. As such, a PC OEM must take care to ensure compatibility between devices on a shared SPI bus. It is likely that there will be BIOS initiated accesses to an SPI ROM containing BIOS code on the same physical bus as an SPI TPM. The PC OEM has two options to deal with this case:

1. The platform and BIOS may be designed to start up at a frequency of 24MHz, as the TPM must support that clock frequency. The BIOS may increase the frequency for transactions to the BIOS ROM at a later point in POST.

2. The platform may be designed with a strap or other hardware method to force operation at a particular frequency, as the PC OEM may select a TPM with support for that particular frequency.

Note 1: The SPI bus may be shared. Therefore when the TPM's CS# is not asserted, the SPI clock may be running at a faster frequency than the TPM supports. Since BIOS knows what TPM is attached to which south bridge, it will need to comprehend the frequency support for both components and will change the SPI clock frequency for the TPM segment while SPI is idle. There is no communication to the TPM of what the SPI frequency is. The frequency can change from command to command, as long as the SPI bus is idle when the frequency changes.

Note 2: TPM's may be used in many types of PC Client platforms. In lower power environments, it is entirely likely that the SPI interface will run at a slower clock frequency, while in high performance environments, the interface will run at a higher frequency. This specification provides a range within which all TPMs will correctly function and allows for TPM vendors to differentiate their parts to allow for a variety of implementations. PC Client systems have multiple standard clock frequencies available which could be used as the source clocks for the SPI interface, such as 14.3MHz, 24MHz, 33MHz, and 66MHz. To enable the widest range of applications, TPM vendors are encouraged to support frequencies between 33MHz and 66MHz in addition to the required clock operating range to allow for higher performance applications.

End of informative comment

1. The TPM MUST support an SPI clock frequency range of 10 - 24 MHz.
2. The TPM MAY support running at lower frequencies.
3. The TPM SHOULD support higher frequencies.

6.4.2 Electrical Specification

Start of informative comment

The SPI interface does not have an industry standard for electrical characteristics that can be referenced for TPM implementations as was done for LPC. This section describes the normative requirements for the TPM as defined at the TPM pins. This does not describe the requirements for the south bridge.

End of informative comment

1. The TPM MUST support a supply and I/O voltage of 1.8V or 3.3V.
2. The TPM MAY support supply and I/O voltages of both 1.8 and 3.3V.
3. The TPM MAY support other supply and I/O voltages.
4. The TPM MUST comply with the electrical specifications in the tables below.

Note: For the electrical specifications in Table 28, the timing characteristics are defined only for the specified clock operating range. For other clock frequencies, the timing characteristics are implementation specific.

Table 26 DC Specifications for 1.8V Supply Voltage

Parameter	Conditions	Min	Max	Units
Vcc power supply		1.65	1.95	V
VIH	Vcc = 1.65V – 1.95V	0.7 * Vcc	0.3 + Vcc	V
VIL	Vcc = 1.65V – 1.95V	-0.3	0.3 * Vcc	V
VOH	Vcc = 1.65V – 1.95V	0.9 * Vcc		Iout = -100μA
VOL	Vcc = 1.65V – 1.95V		0.1 * Vcc	1.5 mA

Table 27 DC Specifications for 3.3V Supply Voltage

Parameter	Conditions	Min	Max	Units
Vcc power supply		3.0	3.6	V
VIH	Vcc= 3.0V – 3.6V	0.7*Vcc	0.5+VCC	V
VIL	Vcc= 3.0V – 3.6V	-0.5V	0.3*VCC	V
VOH	Vcc = 3.0V – 3.60V	0.9 * Vcc		Iout = -100μA
VOL	Vcc = 3.0V – 3.60V		0.1 * Vcc	1.5 mA

1385

Table 28 AC Electrical Specifications

Parameter	Description	Conditions	Min	Max	Units
Clock minimum operating range			10	24	MHz
t_{CLKf}	Clock Period	Rising Edge to Rising Edge	$1/f_{CLK}$-5%	$1/f_{CLK}$+5%	ns
t_{CLKr}	Nominal Clock Period	Nominal Clock Period	$1/f_{CLK}$		ns
t_{CLKL}	Clock Low Time	Clock Low Time (see figure 5)	$0.45t_{CLKr}$	-	ns
t_{CLKH}	Clock High Time	Clock High Time (see figure 5)	$0.45*t_{CLKr}$	-	ns
$t_{CLKslew}$	Clock Slew Rate	$0.2*V_{CC} – 0.6*V_{CC}$	1	4	V/ns
t_{CS}	CS# High Time	Rising Edge to Falling Edge	50		ns
t_{CSS}	CS# Setup to clock	CS Setup time	5		ns
t_{CSH}	CS# Hold to clock	CS Hold time	5		ns
t_{SU}	MOSI Setup to clock	Data Setup time	2		ns
t_{H}	MOSI Hold to clock	Data Hold time	3		ns
t_{HO}	Clock to MISO valid	Output Hold time	0		ns
t_{vmin}	Output valid from clock falling edge minimum	Output Valid Min	0		ns
t_{Vmax}	Output valid from clock falling edge maximum	Output Valid Max		$0.7* t_{CLKL}$	ns
	TPM SPI Pin Capacitance			10	pF

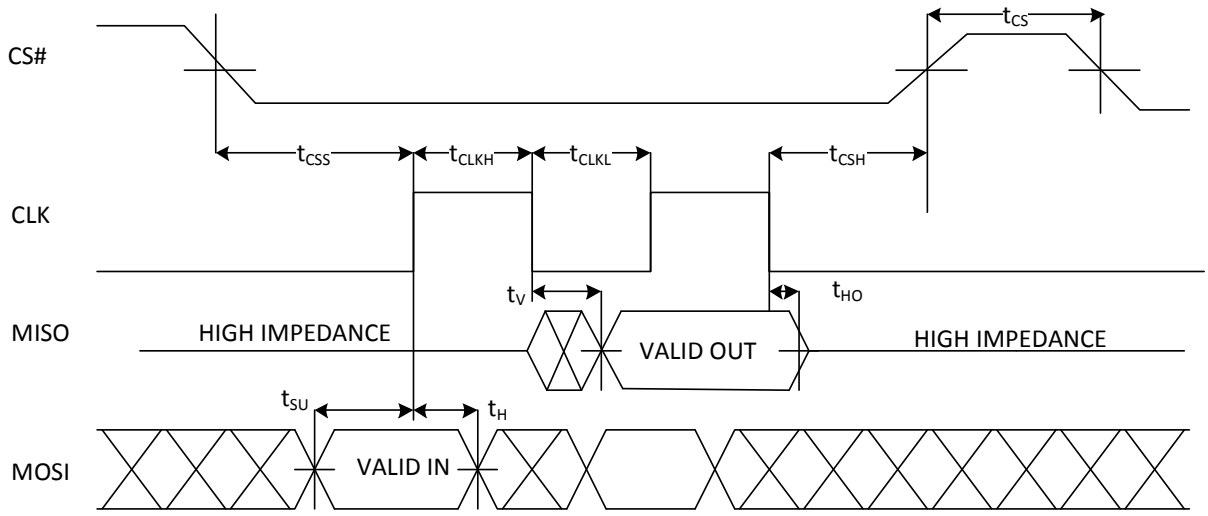


Figure 4 Timing Diagram

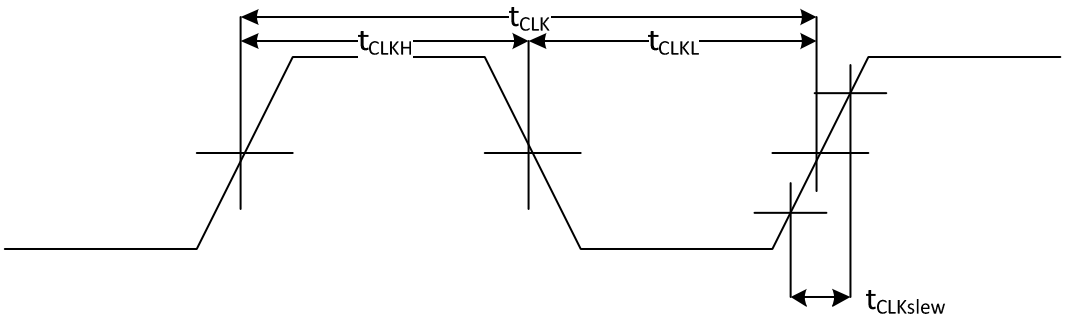


Figure 5 Clock Timing Diagram

6.4.3 SPI Interrupts

Start of informative comment

TPMs compliant with the LPC specification support a serial interrupt, SIRQ. The SPI specification does not have an analog to SIRQ. This specification defines a parallel interrupt which functions in a manner similar to a PCI device's INTx#. The implementation of the pin is active low and open collector such that it is sharable. TPMs might continue to support an SIRQ signal to allow for a common design supporting either interface, but they are not required to do so.

End of informative comment

1. The TPM MUST implement a PIRQ# pin.
   a. PIRQ# MUST be active low
   b. PIRQ# MUST be open collector.
2. The PIRQ# pin MUST be 3.3V tolerant.
3. The TPM MAY implement an SIRQ pin in addition to PIRQ#

6.4.4 Legacy I/O

Start of informative comment

Previous versions of this specification contained support for legacy I/O cycles and addresses supported by TPM1.1b. For the SPI interface, this version of this specification deprecates all support for legacy accesses to the TPM, including the legacy I/O range. If a TPM vendor chooses to continue to support the LPC interface, they may continue to support Legacy addressing on the LPC interface only. It is expected that south bridges supporting SPI as defined in this specification will block any I/O cycles to TPMs connected via an SPI bus. This will be enforced in HW. TPMs compliant with this specification do not need to implement the legacy IOW/IOR access mechanism. The south bridge will route the entire address range from 0xFED4_0000 through 0xFED4_4FFF to the TPM over SPI. This allows the TPM to add new registers and maintain compatibility with the SB

End of informative comment

.

6.4.5 Flow Control

Start of Informative Comment

The SPI interface does not define a flow control mechanism. The TPM, as defined, requires flow control to allow for varying sized data transfers.

This specification defines a method of flow control that operates on a transaction basis. On the LPC interface, transfer occur on a byte by byte basis, regardless of the transaction size. On SPI, transfers may occur in 4B, 8B, 32B or 64B chunks.

The method of flow control specified in this section allows the TPM to insert a wait state to hold off the transfer. Additionally, each transaction will provide, as part of the packet preceding the address, a transaction size. This allows a TPM vendor to implement more advanced mechanisms of flow control. For example, if the south bridge initiates a write of 32B to the TPM_XData_FIFO, the TPM can read the size of the transfer and, if it does not have 32B of open space in its buffer, it would insert a wait state. Alternatively, the TPM could accept the transaction if it had a 64B buffer with only 8B of data present. The TPM vendor isn't required to verify transaction sizes before resorting to the flow control mechanism specified here. The TPM may choose to ignore the flow control method and choose to insert wait states, as defined here, if their buffer is not currently empty. This will have a performance impact on larger transaction sizes, but should pose no issue with 4B or 8B transfers. Byte level flow control was not considered, as the overhead of allowing flow control between each byte is too high with almost no benefit.

To allow flexibility for larger size transactions in the future, south bridges are likely to have limited, if any, HW checking on the size of accesses to the TPM address space. If the south bridge receives a transaction for any size from 1B to 64B that doesn't cross a 64B boundary, it may choose to accept and issue that transaction to the TPM on SPI as received. The TPM, if it doesn't insert a wait state at the designated point must accept the transaction, as long as it doesn't cross a register boundary. If the transaction crosses a register boundary, the TPM may choose to accept all of the data and discard the data that exceeds the size limit for that register as long as doing so does not cause a change to the

state of any adjacent register. The flow control specified in this specification defines a transaction structure for SPI consisting of 1B of command (including direction of transfer and transaction size), 3B of address, followed by the transaction data (either write data from the southbridge to the TPM or read data from the TPM to the southbridge). The TPM may insert wait states following receipt of the address. The southbridge will monitor the MISO line on the rising edge of the clock in the window following transmission of the last bit of the address. The TPM, in order to insert a wait state, drives the MISO line low (0) on the falling edge of the previous clock (clock in which the last bit of address is driven by the southbridge). The TPM would continue to drive MISO line in 8bit increments until it is ready to receive or transmit data. The southbridge polls the MISO line every 8 clocks until it sees a 1, then it either starts to transmit data or expects to receive data on the next falling clock edge.

NOTE: For the purposes of defining the flow control, on SPI the MOSI or MISO signal is driven by the owner on the clock's falling edge, and captured by the receiver on the clock's rising edge.

For a read, the command and address are driven on MOSI and the TPM responds with data on MISO. With no wait states, the TPM would drive data on the next falling clock edge following receipt of the last address bit on the rising edge of the clock. This is illustrated in Figure 6 Example Read transaction with a WAIT state below. The SB monitors the MISO pin in the same clock window that A[0] (the last address bit) is valid. If MISO is captured high on the rising edge of the clock, then the SB will continue to write or read data on the following clock edges. This aligns with standard SPI protocol where there are no wait states. In Figure 6 Example Read transaction with a WAIT state and Figure 7 Example of WRITE transaction with Wait state below, if the TPM drives a '1' in the A[0] window, or in any subsequent wait state window, then MISO is no longer used for wait states, in which case MISO is either providing valid data for reads or is a don't care for writes.

There is no further flow control allowed. Once the data starts coming from the TPM, it must provide the entire transaction's worth of data, which can be from 1 to 64 bytes, depending on the TPM's supported transaction size. In this example, if the SB had latched a '1' on the rising clock in the wait state window, then it would start latching in data starting on the next rising edge, which is the normal behavior without wait states. The TPM may insert any number of wait states that it needs.

Since the wait state is defined as '0' on MISO, if there is no TPM present at all, then the design has a weak pull-up on MISO. If a TPM is not present, a pull-up on MISO means that the SB controller sees a '1', and will latch in 0xFF on the read. This follows standard master abort behavior of 0xFF for read data and matches the behavior when the TPM was on LPC.

For writes, the mechanism is similar. If MISO is '0' to request a wait state, then the data driven during that byte is not valid and the TPM will drop the data. If there is a wait state, the master will drive the first byte of the transaction until the slave stops requesting wait states. The SB will sample MISO on the last data bit of the byte (multiples of 8 clocks after the first wait state window). Again, the TPM must hold MISO as '0' for 8 clocks each time it requests a wait state. If MISO is '1', this indicates the TPM is ready for the entire write, accepts the first byte which the SB has sent in the same clock, and the SB will then drive the 2nd and subsequent bytes on the following clocks. Once the data starts transmitting, the entire write data will be sent with no further flow control. See Figure 7 Example of WRITE transaction with Wait state for an example of a write transaction with wait states inserted .

Wait states are byte based. For reads where MISO is used to return data to the master, the SB will start sampling at byte[-1], which is the window when A[7:0] is transmitted. If the last bit of this window is '0', then the TPM is requesting a wait state and the SB continues reading MISO for 8 clocks (a byte's worth) and will look at the last bit to determine if there should be another wait state or not. From the SB standpoint, it is simply reading a byte and processing it as a byte. The last bit of that byte determines what to do next. If the bit is '1', then the SB knows to start sending the valid data on writes, or receiving data for reads. One option on writes is to send data byte 0 over and over until there is no wait state, and then move to byte 1, etc. For reads, each byte can be sampled, and when the last bit is '1', start moving the next byte into the data buffer and increment the byte count received.

DESIGN NOTE: The TPM interface was architected knowing that some of the actual command processing could take seconds to complete. Therefore registers were provided that SW can poll on to determine when it should read the actual results of the command. Software should never attempt to read the DATA FIFO without verifying that the TPM_STS_x.commandReady and TPM_STS_x.dataAvail fields are set. If software does, the TPM will return FF's as described in section 5.6.13 Data FIFO Register. For writes, the TPM is required to insert wait states if software attempts to write data without waiting for the TPM to transition to the Ready state. The HW will allow flow control until the TPM is ready to provide the data, which could be as long as the applicable timeout. On the other hand the registers used for the SW control must not have excessively long delays or the system performance would be impacted. There are some registers for which wait states would present a problem in the overall operation of the system. The TPM is only allowed 1 wait state to decode the address before returning the contents of the register. The registers with this restriction are:

ACCESS (0x0), once the requirements defined in Section 6.6 Reset Timing are satisfied

INT_ENABLE (0x8)

INT_VECTOR (0xC)

INT_STATUS (0x10)

INTF_CAPABILITY (0x14)

STS (0x18), after the register contains a valid logical level as defined in Section 5.6.12.2 Bus Access of the Status Register

HASH_START (0x20)

DID_VID (0xF00)

RID (0xF04)

NOTE: when inserting wait states on the bus, if that SPI segment is used by other devices, then they will be stalled until the TPM completes the transaction. Adding wait states slows down the system, so should be used sparingly.

Example of a read and write transaction with WAIT state are shown below

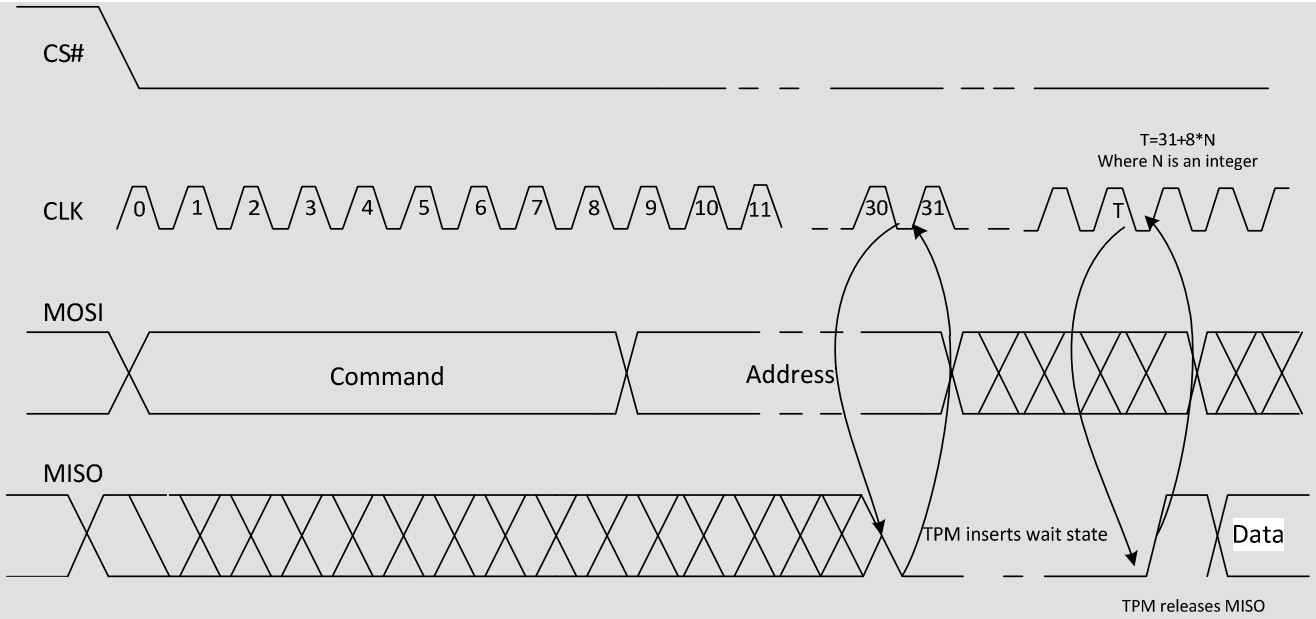


Figure 6 Example Read transaction with a WAIT state

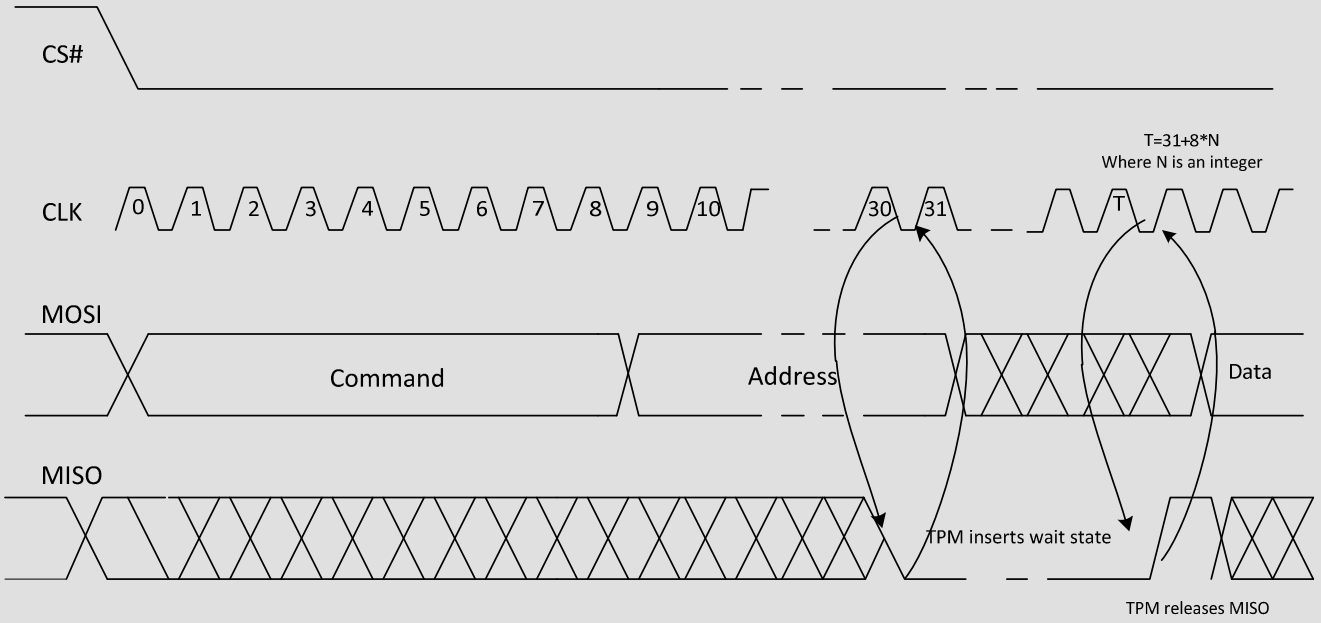


Figure 7 Example of WRITE transaction with Wait state

End of informative comment

1. The wait state window is defined to start on the rising edge of the clock on the transmission of the last bit of address to the rising edge of the subsequent clock.

a. The wait state window repeats every 8 clocks in the same transaction until the TPM no longer inserts wait states.

2. The TPM MUST drive MISO low (0) on the falling edge of the clock in the wait state window to signal insertion of a wait state.

3. The TPM MUST continue to drive MISO low (0) on the falling edge of the clock in subsequent wait state windows for the same transaction until it is ready to receive (Read) or transmit (Write) all of the data for that transaction.

   a. The TPM MUST drive MISO high (1) on the falling edge of the clock in the wait state window to signal no further wait states will be inserted.

4. The TPM MUST drive MISO high (1) on the falling edge of the clock in the first wait state window if it does not intend to insert a wait state.

5. The TPM MUST NOT insert more than 1 wait state on a Read cycle to the following registers:

   a. TPM_ACCESS_X, once the requirements defined in Section 6.6 Reset Timing are satisfied

   b. TPM_STS_X, after the register contains a valid logical level as defined in Section 5.6.12.2 Bus Access of the Status Register

   c. TPM_INTF_CAPABILITY

   d. TPM_INT_ENABLE

   e. TPM_INT_STATUS

   f. TPM_INT_VECTOR

   g. TPM_DID

   h. TPM_VID

6. The TPM MAY insert wait states for accesses to the TPM_HASH_x register, but MUST NOT exceed TIMEOUT_B.

6.4.6 SPI Bit Protocol

Start of informative comment:

The bit protocol defined in this section provides for transactions to follow the following rules:

- Data is transferred most significant bit (msb) first, least significant byte (LSB) first
- Address and command are transferred msb first for the entire field, e.g. the 24-bit address is transferred by sending A23 first, then A22 all the way to A0.
- Master and slave both drive data on the falling edge of the SPI clock.
- Master and slave both sample data on the rising edge of the SPI clock.
- The address presented to the TPM on the SPI bus will always be a 24-bit address that is offset from 0xFE. The chipset will decide the full address and if the cycle is in the 0xFED4_xxxx range, it will assert the TPM's CS# pin.
- Only SPI mode 0 is supported (CPHA=0, CPOL=0).

The bit order defined below is transmitted on the wire starting from the bottom of the table, ending with the top of the table. The first bit in the protocol is the read/write bit, allowing the TPM to determine what type of transaction follows. The last bit is the least significant bit (lsb) of the most significant byte (MSB) of the data packet. As described in Section 6.4.5 Flow Control, it is legal to transmit any number of bytes of data from 1B to 64B. Zero length transactions are not supported or allowed.

There is no status byte built into the protocol. The existing methods using TPM_STS_x.burstCount and TPM_STS_x.Expect govern transfer failures, as defined in Section 5.6.12 Status Register. If the TPM has not received all of the bytes of the transaction, it will set TPM_STS_x.Expect to a 1, to signal to the chipset that it still expects data. On a read, if the chipset does not receive the required number of bytes (or any bytes), the chipset may issue a retry, or it may send all FF's to the driver, signaling a failure.

The SPI interface has evolved to support double data rate transactions. The TPM does not support double data rate transfers.

1. The TPM MUST support the bit protocol defined in Table 29 SPI Bit Protocol.
2. The TPM MUST drive read data on the falling edge of the clock
3. The TPM MUST sample write data on the rising edge of the clock.
4. The TPM MUST decode transactions sent to offset 0xD4_xxxx when its CS# is asserted.

Bit Transfer Order on MISO/MOSI	**BYTE on MISO/MOSI**	**Usage**	**Notes**
	67 for 64B xactions 11 for 8B xactions	future use for larger register sizes	
57-63 – last bits on wire	7	Data[30:24]	
56		Data[31]	msb of 4th LSB
49-55	6	Data[22:16]	
48		Data[23]	msb of 3rd LSB
41-47	5	Data[14:8]	
40		Data[15]	msb of 2nd LSB
33-39	4	Data[6:0]	
32		Data[7]	msb of LSB
Optional flow control can be done in this window. See Flow Control section for details. This is the only place in the bit xfers where flow control can be done.			

31	3	Addr[0]	lsb of address
9-30	1-3	Addr[22] down to Addr[1]	
8	1	Addr[23]	msb of address
2-7	0	bits[5:0] Size of xfer where bit[5] of this field is the 3rd bit transferred on the wire, and bit [0] is the 8th bit on the wire. This field is 0's based count of the bytes. Any byte count from 1 to 64 is legal.	Bit [5:0] decode '11_1111' = 64 bytes ' etc. for 63 down to 6 bytes '00_0100' = 5 bytes '00_0011' = 4 bytes '00_0010' = 3 bytes '00_0001' = 2 bytes '00_0000' = 1 byte
1		rsvd; bit[6]	
0 – first bit on wire		Byte0, bit[7] Read/Write	1=read, 0 = write

Table 29 SPI Bit Protocol

1605

6.5 TPM Byte Ordering

Start of informative comment

The TPM Interface Specification contains definitions for TPM registers which have multi-byte address ranges. Data transmitted on the interface to these registers is transferred from the lowest address or least significant byte (LSB at byte offset 0) to the highest address or most significant byte. For more information on the addressing and address decode of these registers, see Sections 5.4, 5.6.11and 5.6.12.

The TPM Main Specification defines command structures which have multi-byte fields, which are defined as follows: Integer values are expressed as an array of one or more bytes. The byte at offset zero within the array is the most significant byte of the integer, referred to within this section as big-endian. For example, a field designation of UINT-32 is a byte array of 4 bytes with the most significant byte at byte offset 0. These commands are transmitted on the TPM interface as the payload of a TPM register access.

The TPM Interface does not ensure or validate the byte ordering of the payload. It is the responsibility of the TPM software, typically the TPM driver in conjunction with the TSS, to correctly marshal the command payload for any write to a TPM register.

To write to a multi-byte register, e.g. the TPM_DATA_FIFO, the TPM must receive the least significant byte first, as defined in Section 5.4.2. The payload of that transfer will contain the command, which may include multi-byte arrays, having their MSB at offset 0.

The driver is required to handle the byte ordering of TPM command fields vs. the byte ordering of the TPM registers, LPC bridge and LPC bus. Software may do Double Word (DW) (4 bytes) or Word (2 bytes) or Byte accesses to the TPM. Standard PC platforms will have bridges that translate these 4-byte or 2-byte accesses into single-byte accesses on LPC. PC platforms will always break up the request so that they send the least significant address of that request first on the LPC bus.

As an example, the TPM data structure TPM_PROTOCOL_ID is defined in the TPM Main Specification as having a value of 0x0005 for PID_OWNER, where 0x00 is the MSB in the array. To ensure the TPM receives the correct command payload, this structure must be sent to the TPM with 0x00 as the first byte and 0x05 as the second byte. However, the bridge between the CPU and the TPM sends the LSB first and, therefore, a CPU write of 0x0005 would send the 0x05 to the TPM first, then the 0x00 which is not the correct sequence for the TCG_PROTOCOL_ID for PID_OWNER. The driver could perform two 1-byte accesses to the TPM, the first write would be with data 0x00, and the second write would be with data 0x05. Or software could do a Word access and send 0x0500, which would result in the bridge issuing the 0x00 cycle first on the LPC bus followed by an LPC cycle of 0x05.

End of informative comment

The TPM Main Specification has multi-byte fields, expressed as an array where the byte at offset zero within the array is the most significant byte of the array, defined as big-endian. To meet the big-endian requirement for multi-byte fields, the TPM MUST first receive the MSB (most –significant byte) on the LPC bus.

6.6 Reset Timing

Start of informative comment

The operation of the platform's CRTM likely occurs during a very time-sensitive period. Because of this, strict requirements are necessary for the TPM's reset timing. During this time, the platform's CRTM may need to make decisions based on the presence or absence of the TPM's response that affect the rest of the platform's boot cycle. This requires that any return from TPM_ACCESS_x register be valid regardless of the timing – the TPM must not be allowed to return anything but a valid response from this register.

While the TPM_ACCESS_x register is the most critical, the availability of the other registers is important for performance reasons.

This section contains the timing requirements for the TPM's registers. The normative requirements describing the relationship between the individual fields within a register are contained in sections 5.6.11 Access Register and 5.6.12 Status Register.

End of informative comment

1. Within 500 microseconds of the completion of TPM_Init:
   a. All fields within all TPM_ACCESS_x registers MUST be a valid logical level as indicated by the tpmRegValidSts field being set to a '0' or a'1'.
2. Within 30 milliseconds of the completion of TPM_Init:
   a. All fields within the access register and all other registers MUST return with the state of all their fields valid (i.e. TPM_ACCESS_x.tpmRegValidSts is set to '1').
   b. The TPM MUST be ready to receive a command

6.7 TPM Hardware Implementation

Start of informative comment

Hardware implementations of TPM as a device and in a PC Client platform require the careful consideration of some key elements. This section provides guidance for the TPM vendor's hardware implementation of the TPM and for the motherboard manufacturers designing the TPM into a PC Client platform. Section 6.7.1 is targeted at TPM vendors, providing for a standardized package and pin-out that allows for form and fit compatibility across multiple TPM vendors, providing the greatest design flexibility for both TPM vendors and motherboard manufacturers. Section 6.7.2 is targeted at motherboard manufacturers, providing a collection of the critical hardware elements necessary to implement the TPM in a PC Client system.

End of informative comment

6.7.1 TPM Packaging

Start of informative comment

A standard package allows TPM and motherboard manufacturers the convenience and cost savings of not having to define from scratch the packaging and pin-out for a TPM. This packaging and pin-out recommendation is provided as a convenience for either an end product or as a basis for extension or modification. It is recognized that individual environments may dictate other schemes; therefore, implementation of this section is optional and any deviance will not detract from a platform's claim to adherence to this specification.

End of informative comment

The TPM MAY use the packaging and pinout recommendation as defined in this section (per Figure 8 TPM Pinout and Table 30: Pin Assignments).

In order to claim compliance to this section of this specification, the TPM MUST use both the packaging and pinout as defined in this section:

- It SHALL be said to use the "Packaging as specified in Section 6.7.1of the TCG PC Client Specific TPM Interface Specification (TIS) 1.3".
- It MUST be designed as a 28-pin TSSOP using 9.6 mm plastic length (with 0.65 mm lead pitch) by 6.1 mm or 4.4 mm plastic width.

If a TPM does not use either the packaging or pinout specified in this section:

- It MUST NOT claim compliance to this section of this specification.
- The TPM manufacturer MUST provide documentation to the platform manufacturer regarding the package and pinout, including the GPIO-Express-00 pin's electrical characteristics.

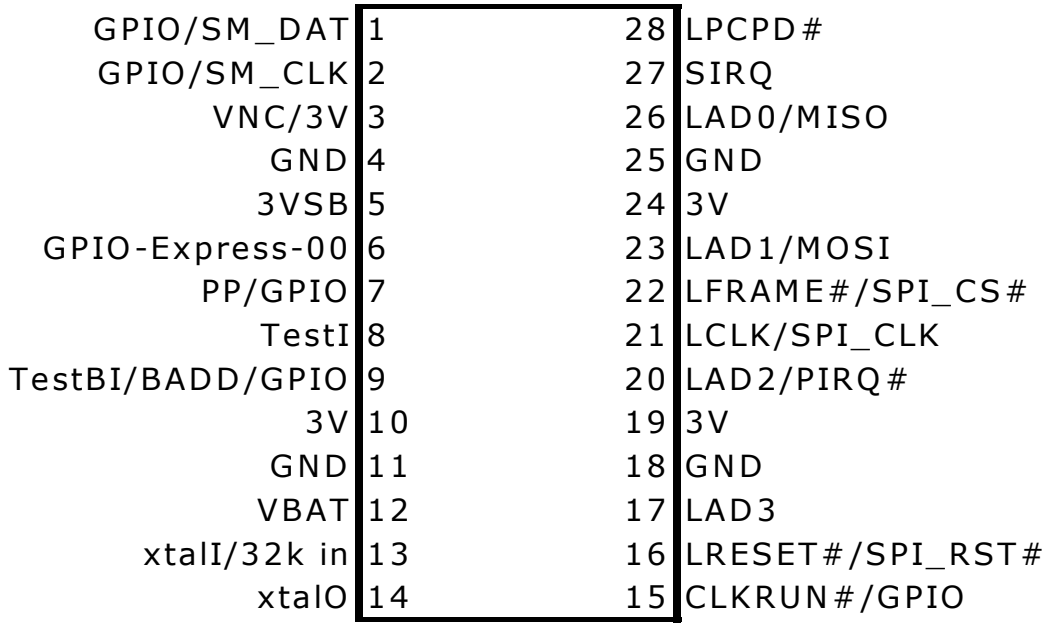


Figure 8 TPM Pinout

Using the pin-out defined in Figure 8 TPM Pinout, the pins SHALL be assigned as defined in Table 30: Pin Assignments.

Start of informative comment

TPMs implementing BADD logic are using various legacy I/O base addresses (index register at base address and data register at base address + 1), e.g. (BADD high / BADD low): 7Eh&7Fh / EEh&EFh, 2Eh&2Fh / 4Eh&4Fh.
See Section 6.2 TPM Legacy I/O Space and TPM 1.2 Memory Mapped Space for details. This functionality is deleted for SPI TPM's.

The pins which are marked "M" for "mandatory" in Table 26 must be implemented. Those which are marked "O" for "optional" are not required for claims of adherence, but if implemented, must be implemented per Table 26.

End of informative comment

Table 30: Pin Assignments

Signal	Pin(s)	Type	Description	LPC pin-assignments	SPI pin-assignments
LAD3	17	BI	As defined in the LPC Interface Specification	M	O
LAD2/PIRQ	20	BI/O	LAD 2: As defined in the LPC Interface Specification PIRQ#: SPI Interrupt, active low, open collector, internal pull-up (?)	LAD2 – M	PIRQ#-M
LAD[1:0]	23, 26	BI		M	O
LPCPD#	28	I	Implementation of this pin MUST allow for the pin to be strapped HIGH.	O	O
LCLK/SPI_CLK	21	I	LCLK: As defined in the LPC Interface Specification. SPI_CLK: As defined in section 6.4.1 and 6.4.2	LCLK-M	SPI_CLK-M
LFRAME#/SPI_CS#	22	I	LFRAME#: As defined in the LPC Interface Specification SPI_CS#: As defined in section 6.4	LFRAME-M	SPI_CS#-M
LRESET#/SPI_RST#	16	I	LRESET#: As defined in the LPC Interface Specification SPI_RST#: Active Low	LRESET#-M	SPI_RST#-M
SERIRQ	27	BI	As defined in the LPC Interface Specification	M	O
CLKRUN#/GPIO	15	BI	Same as PCI CLKRUN#. Active Low, internal pull-down. Only needed by peripherals that need DMA or bus mastering in a system that can stop the PCI bus (generally mobile devices). Implementation of CLKRUN# is TPM and chipset vendor specific GPIO will default to low.	O	O
PP/GPIO	7	I,BI	Physical Presence, active high, internal pull-down. Used to indicate Physical Presence to the TPM. GPIO will default to low	O	O
XTALI/32k in	13	I	32 kHz crystal input or 32 kHz clock input	O	O
XTALO	14	O	32 kHz crystal output	O	O
GPIO/SM_CLK	2	BI	Defaults as a GPIO. GPIO will default high. Also used as System Management Bus (SMB) Clock signal	O	O
GPIO/SM_DAT	1	BI	Defaults as a GPIO. GPIO will default high Also used as System Management Bus (SMB)Data signal.	O	O
GPIO-Express-00	6	BI	GPIO assigned to TPM_NV_INDEX_GPIO_00, internal pull-up	O	O

Signal	Pin(s)	Type	Description	LPC pin-assignments	SPI pin-assignments
			Open-Collector output (when configured as output).		
VNC	3		Vendor-controlled No Connect. This pin will be defined by the TPM vendor or can be a GPIO. There is no defined default state for this signal.	O	O
TESTI	8	I	This pin will be pulled low on the motherboard. Pull high to enable Test mode. Pull low to disable Test mode and enable GPIO/BADD on pin 9(TESTBI).	O	O
TESTBI/ BADD/GPIO	9	8	TESTBI: Test port. Internal pull-up If TESTI is pulled low, TESTBI acts as a GPIO and (optionally) BADD. GPIO will default high. BADD (optional, defaults high, use external pull-down to signal "low) can be used to select the legacy I/O base address. This logic is manufacturer specific, as well as the selected addresses. Setting is read at Startup.	O	O
Power					
3V	10, 19 24	I	This is a 3.3 volt DC power rail supplied by the motherboard to the module. The maximum power for this interface is 250 mA. Available from S0-S2.	M	M
GND	4, 11, 18, 25	I	Zero volts. Expected to be connected to main motherboard ground.	M	M
VBAT	12	I	3.3 V battery input. Available from S0-S5 and in G3 state.	O	O
3VSB	5	I	3.3 V standby DC power rail. Available from S0-S5.	O	O

6.7.2 Hardware Implementation of a TPM in a PC Client Platform

Start of informative comment

The TPM in the PC Client platform serves as the Root of Trust. As such, the hardware implementation of the TPM on the motherboard has to account for how the TPM is connected to the other components of the platform which form the trust chain, such as the CPU. It is important that the TPM reset, clock and power signals support the TPM's function as the RTM and RTR and cannot be easily circumvented. Motherboard manufacturers should take care to ensure that the physical connections and routing minimize the possibility of attacking the S-CRTM and DRTM.

End of informative comment

1. The TPM_Init (LRESET#/SPI_RST#) signal MUST be connected to the platform CPU Reset signal such that it complies with the requirements specified in section 1.2.7 HOST Platform Reset in the PC Client Implementation Specification for Conventional BIOS.
2. The TPM's main power pins (3V) MUST be connected such that the TPM is powered during ACPI states S0-S2 and MAY be powered in S3-S5.
3. If a TPM implements the optional VBAT and/or 3VSB pins, the pins MAY be connected to a battery or auxiliary power source. The motherboard manufacturer SHOULD consult their TPM documentation.

4. If a LPC TPM is implemented using the recommended packaging in Table 30: Pin Assignments, the TPM's LPC bus MUST be connected as defined in the LPC Specification, except as follows:
    a. If the LPCPD# power down protocol is not implemented in both the chipset and the TPM, the LPCPD# pin on the TPM MUST be strapped HIGH.
    b. If the LPCPD# power down protocol is implemented in both the chipset and the TPM, the LPCPD# pin MAY be strapped HIGH.
    c. CLKRUN# MAY be strapped HIGH to disable the TPM's CLKRUN# protocol. **Note:** If the TPM does not implement a pin for CLKRUN#, it is assumed to support the host disabling the LPC clock without changing the TPM state.
5. If an SPI TPM is implemented using the recommended packaging as defined in Figure 9 and Table 30, the TPM's SPI bus MUST be connected so that:
    a. The TPM has a dedicated chip select (SPI_CS#).
    b. The TPM's SPI_RST# is connected directly to the platform's RST#, so that it cannot be controlled independently of the south bridge asserting CPU_RST#.
6. An SPI TPM MUST be implemented in a platform such that is only accessed, via asserting SPI_CS#, if an MMIO access to 0xFED4xxxx is received by the chipset. ***Note***: Locality 4 accesses to the hardware hash registers may be accessed via an implementation specific mechanism other than MMIO, but these accesses still obey this rule by virtue of being in the TPM's address range.
7. The Platform MUST provide a hardware mechanism, e.g. a hardware-based strap, to configure the platform's TPM and chipset to support the SPI interface.

6.7.2.1 SPI Platform Design Notes

Start of informative comment:

This section provides guidelines for platform OEM's and ISV's to aid in design of platforms and software using an SPI TPM. The following sections are informative only, as they describe recommended behavior.

End of informative comment

6.7.2.2 SW Interface to SPI-TPM

Start of informative comment:

The SPI interface has been architected to be transparent to the driver and application layers in a TPM-enabled software stack. There are some SPI properties which will produce different results than LPC in cases where software does not follow good design practice. In these cases, this specification addresses the TPM requirements so that none of the TPM's security is impacted by bad software design, at the risk of a potentially poor user experience. As such, software and drivers should follow these recommendations to ensure a robust implementation.

- SW should continue to use the memory mapped 0xFED4_xxxx address range to access the SPI-TPM. ***Note:*** This is the same address range as for the LPC-TPM.
- All existing LPC-TPM code will continue to work as is with the SPI-TPM

- SW which uses legacy LPC cycles to the TPM will not work with an SPI TPM.
- The SPI TPM has added an extended data FIFO for larger data transfers. Software may or may not use this new register.
- Software should continue to use the existing protocols for accessing the TPM as described in Sections 5.6.11 Access Register and 5.6.12 Status Register.

End of informative comment

6.7.2.3 SW Command Interface to SPI-TPM

Start of informative comment:

Many platforms support FLASH components on SPI by allowing SW to program a command-based interface. SW sets up specific commands such as block write, erase, etc. through some platform dependent set of registers.

NOTE: some platforms may choose to not have a TPM as part of the platform, or continue to use the LPC-TPM. Those platforms may use the SPI-TPM CS# for additional FLASH space or some other usage. In that case the command-based accesses may be allowed. The mechanism to determine whether the CS# is attached to a TPM must not be SW dependent but should be some HW mechanism fixed by the OEM at manufacturing.

End of Informative comment

7. References

1. The Trusted Computing Group: http://www.trustedcomputinggroup.org
2. Low Pin Count (LPC) Interface Specification: http://www.intel.com/design/chipsets/industry/lpc.htm
3. PC Specific Implementation Specification: http://www.trustedcomputinggroup.org/developers/pc_client/specifications
4. System Management Bus (SMBus) Specification Version 2.0: http://www.smbus.org
5. PCI Express Electromechanical Specifications: http://www.pcisig.com
6. The Serial IRQ (SERIRQ) protocol definition: http://www.smsc.com/ftpdocs/papers.html
7. TPM Main Specification: http://www.trustedcomputinggroup.org/developers/trusted_platform_module/specifications