



Automotive, Industrial & Multimarket

Infineon TPM (SLB 9635 TT 1.2 / SLD 9630 TT 1.1)

Subject: Infineon-LPC-TPM-Bios-Driver for 32Bit

Category: Preliminary BIOS-Porting Design Guide

Doc-Class: Project Specific Documents

Doc-Version: 2.00.0000

Date: 10/12/2005

Security Classification: CONFIDENTIAL (Distribution under NDA only)

Dev. / Step Code:	Sales Code:
Status: CONFIDENTIAL (Distribution under NDA only)	Date: 12 October 2005
Document: IfxTPMBiosDrv32_BiosGuide.doc	Created with: Microsoft Word 10.0
Author: AIM CC SW	
Document path:	
Comments:	

REVISION HISTORY

VERSION	DATE	CHANGE MADE BY	SECTION NUMBER	DESCRIPTION OF CHANGE
0.10	2002-06-07	Infineon Technologies	all	First draft
0.20	2002-07-12	Infineon Technologies	all	Extensions in section 5.5
0.30	2002-09-09	Infineon Technologies	all	First review inputs
0.90	2002-09-16	Infineon Technologies	all	Move to v0.90
0.90.0001	2003-02-26	Infineon Technologies	3.2.1 and 6.2	New driver header entry and 16 Bit description.
1.00.0000	2003-05-06	Infineon Technologies	3.2 5.5.2.6 5.5.3.3	Basic requirements Sample BIOS Setup flow added Alternative S3->S0 flow
1.00.0001	2003-06-06	Infineon Technologies	3.2.1	New driver header entry for return operation.
1.00.0002	2003-12-12	Infineon Technologies	3.2	Clarification for ACPI integration
1.00.0003	2004-01-13	Infineon Technologies	all	Editorial changes only
1.00.0004	2004-04-23	Infineon Technologies	5.5.2.6	BIOS Setup flow corrected
2.00.0000	2005-10-12	Infineon Technologies	all 3.2 3.2.1 5.5.2.6 6.1	Editorial changes only Updated ACPI integration. New driver header entry. Updated BIOS Setup Flow Chart. Updated Deployment Package Description.

Important: Further information is confidential and on request. Please contact:
Infineon Technologies AG in Munich, Germany,
AIM CC TC,
Phone +49-89-234-80000
Fax +49-89-234-81000

For support please contact:

E-Mail:
security.chipcard.ics@infineon.com

To learn more about Infineon Technologies TPM please visit our web site at:
<http://www.infineon.com/tpm>

Published by Infineon Technologies AG,
St.-Martin-Strasse, D-81541 München
© Infineon Technologies AG 2005
All Rights Reserved.

Attention please!

The information herein is given to describe certain components and shall not be considered as warranted characteristics. Terms of delivery and rights to technical change reserved.

We hereby disclaim any and all warranties, including but not limited to warranties of non-infringement, regarding circuits, descriptions and charts stated herein.

Infineon Technologies is an approved CECC manufacturer.

Information

For further information on technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies Office in Germany or our Infineon Technologies Representatives world-wide (see address list).

Warnings

Due to technical requirements components may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies Office.

Infineon Technologies Components may only be used in life-support devices or systems with the express written approval of Infineon Technologies, if a failure of such components can reasonably be expected to cause the failure of that life-support device or system, or to affect the safety or effectiveness of that device or system. Life support devices or systems are intended to be implanted in the human body, or to support and/or maintain and sustain and/or protect human life. If they fail, it is reasonable to assume that the health of the user or other persons may be endangered.

Contents

1	Document Management.....	6
1.1	Document was created using the following tools.....	6
1.2	Relationship with other documents.....	6
1.3	Definitions of terms and abbreviations.....	7
2	Introduction.....	8
2.1	Purpose of the document.....	8
2.2	The OSI Layers (communication layers).....	8
3	Product architecture.....	10
3.1	Product structure.....	10
3.1.1	BIOS-Driver System-Software Overview.....	10
3.1.2	BIOS-Driver System Flowing Overview.....	11
3.2	Basic requirements.....	11
3.2.1	Object Format of BIOS Drivers.....	13
3.2.2	BIOS-Driver Object Image Layout (MA and MP).....	15
4	MA-Driver Module Architecture.....	16
4.1	Introduction.....	16
4.1.1	MA Driver Limitations.....	16
4.1.2	MA-Driver Basic requirements.....	16
4.2	MA-Driver Parameters and Structures.....	17
4.2.1	MA-Driver argument structure.....	17
4.2.2	Parameter pblnBuf.....	17
4.2.3	Parameter dwlnPCRLen.....	17
4.2.4	Parameter bMAInitTPMFctld.....	17
4.2.5	Parameter bMAPhyPresenceTPMCmdld.....	17
4.3	MA-Driver Functions.....	18
4.3.1	MA-Driver Function Interface.....	18
4.3.1.1	Function MAInitTPM (Function Number: 01h).....	19
4.3.1.2	Function MAHashAllExtendTPM (Function Number: 02h).....	21
4.3.1.3	Function MAPhysicalPresenceTPM (Function Number: 03h).....	23
5	MP-Driver Module Architecture.....	24
5.1	Introduction.....	24
5.1.1	MP Driver Limitations.....	24
5.1.2	MP-Driver Basic requirements.....	24
5.2	MP-Driver Parameters and Structures.....	25
5.2.1	Parameter pblnBuf.....	25
5.2.2	Parameter pbOutBuf.....	25
5.2.3	Parameter dwlnLen.....	25
5.2.4	Parameter dwOutLen.....	25
5.2.5	Structure TPMTransmitEntry.....	25
5.2.6	Parameter lpTPMTransInfo.....	26
5.3	MP Driver Functions.....	26
5.3.1	MP Driver Function Interface.....	26
5.3.1.1	Function MPInitTPM (Function-Nr-AL-Register: 01h).....	27
5.3.1.2	Function MPCloseTPM (Function-Nr-AL-Register: 02h).....	29
5.3.1.3	Function MPGetTPMStatusInfo (Function-Nr-AL-Register: 03h).....	30
5.3.1.3.1	Return-Values for MPGetTPMStatusInfo (Function: 03h).....	31
5.3.1.4	Function MPTPMTransmit (Function-Nr-AL-Register: 04h).....	32

5.4	Error/Return Codes	40
5.5	Basic BIOS integration hints from TCG perspective.....	41
5.5.1	Basic BIOS integration phases	41
5.5.2	Basic BIOS-Setup commands for TPM from TCG perspective.....	42
5.5.2.1	Enable and activate a TPM device from TCG perspective	42
5.5.2.2	Disable and deactivate a TPM device from TCG perspective.....	43
5.5.2.3	Enabling Ownership for TPM device from TCG perspective (optional)	44
5.5.2.4	Clear TPM device from TCG perspective (ForceClear command)	45
5.5.2.5	TSC_PhysicalPresence operation for TPM device from TCG perspective	46
5.5.2.6	Typical TPM-Admin BIOS Setup flow.....	47
5.5.3	Basic BIOS behavior during the Pre-OS-Boot state from TCG perspective	48
5.5.3.1	S5 ⇒ S0.....	49
5.5.3.2	S4 ⇒ S0.....	51
5.5.3.3	S3 ⇒ S0.....	51
5.5.3.4	S2 ⇒ S0 and S1 ⇒ S0.....	52
5.6	Notes.....	53
5.6.1	Hash-Performance for IFX-TPM device:	53
5.6.2	Endianness of structures and data types	53
5.6.3	Size Estimation for MA- and MP-Driver image files.....	53
6	Appendix	54
6.1	Deployment package description of the IFX-TPM-BIOS-Drivers (32Bit-Versions):.....	54
6.2	16Bit-Driver Remarks/Assumptions	54
6.3	Trademarks	55
7	Open Topics	55

1 Document Management

1.1 Document was created using the following tools

- WINWORD 2002

1.2 Relationship with other documents

	Document	Author	Date
[1]	TCPA Main Specification V1.1b	TCPA Consortium	2002/02/26
[2]	TPM Main Specification v1.2	TCG	
[3]	TCG PC Specific Implementation Spec v1.1	TCG	2002/08/18
[4]	System BIOS for IBM PCs Compatibles	IBM	1991/--/--
[5]	82801BA I/O Controller (ICH2) Datasheet	Intel	2000/10/--
[6]	LPC Interface Specification Revision 1.0	Intel	1997/09/29
[7]	Preliminary TPM SLB9635TT1.2 Databook Rev.0.9	Infineon Technologies	Jan 2005
[8]	TPM SLD9630TT1.1 Databook Rev.2.2	Infineon Technologies	Feb 2004
[9]	TCG PC Client Specific Implementation Specification	TCG	2005/07/15
[10]	SLB 9635 TT 1.2 Databook	Infineon Technologies	

1.3 Definitions of terms and abbreviations

Abbreviation	Explanation
ACRYL	Advanced Cryptographic Library
API	Application Programming Interface
BIOS	Basic Input/Output System
CPU	Central Processing Unit
CRTM	Core Root of Trust for Measurement
DMA	Direct Memory Access
EPP	Enhanced Parallel Port
FPU	Floating-Point Unit
HW	Hardware
I/O	Input and Output
ICH	I/O Controller Hub (e.g. ICH2 82801BA from Intel®)
IPL	Initial Program Load
IRQ	Interrupt Request line
ISA	Industry Standard Architecture (PC bus architecture; Plug and Play)
ISO	International Organization for Standardization
LPC	Low Pin Count Interface
MA	Memory Absent
MMX/3DNow	CPU with extended Multimedia technology (register and instructions)
MP	Memory Present
OS	Operating System
OSI	Open Systems Interconnection
PC	Personal Computer
PCI	Peripheral Connect Interface (PC bus architecture)
PFA	PCI Function Address consists of a function number, a device number and a bus number.
POST	Power-On Self Test; (BIOS routines that initialize and configure the PC-HW)
RTC	The CMOS Real Time Clock
SDK	Software Development Kit
SERIRQ	Serial Interrupt Request for PCI
SW	Software
TCG	Trusted Computing Group
TCPA	Trusted Computing Platform Alliance
TPM	Trusted Platform Module
USB	Universal Serial Bus

2 Introduction

Thank you for your interest in the Infineon Technologies LPC TPM device family. This document describes how to configure the Infineon Technologies TPM-Bios-Driver for 32Bit environments and to communicate with the TPM device from BIOS perspective. It is intended for BIOS writers to assist for developing a TCG-Aware BIOS. The information contained in this document is for reference only, and is subject to change without notice.

2.1 Purpose of the document

This document is intended for a typical BIOS developer. It provides the basic knowledge and understanding required to port a BIOS to support TCG functionality. This guide describes how to use the Infineon Technologies IFX-LPC-TPM-Bios-Driver on 32Bit systems to integrate TCG-Functionality. The main focus of this document is to demonstrate the driver usage and the basic/raw TCG function sequences. Related aspects, like TCG-Event logging, or ACPI handling are not covered by this documentation, these characteristics are part of the TCG-PC-Client-Specific Implementation Specification. After reading this document a typical BIOS developer should be able to develop an IFX-TPM device enabled BIOS.

2.2 The OSI Layers (communication layers)

Physical (Layer 1)

Concerned with transmission of unstructured bit stream over the physical link. It invokes such parameters as signal voltage swing and bit duration. It deals with the mechanical, electrical, procedural characteristics to establish, maintain and deactivate the physical link (Pins, RS232, wires, cards, volts, Repeaters).

Data link (Layer 2)

Provides for the reliable transfer of data across the physical link. It sends blocks of data (frames) with the necessary synchronization, error control and flow control (Hardware Address, Bridges, Intelligent hubs).

Network (Layer 3)

Provides upper layers with independence from the data transmission and switching technologies used to connect systems. It is responsible for establishing, maintaining and terminating connections (Software Address, Routers).

Transport (Layer 4)

Provides reliable, transparent transfer of data between end points. It provides end-to-end error recovery and flow control (Reliability).

Session (Layer 5)

Provides the control structure for communication between applications. It establishes, manages and terminates connections (sessions) between cooperating applications.

Presentation (Layer 6)

Performs generally useful transformations on data to provide a standardized application interface and to provide common communications services. It provides services such as encryption, text compression and reformatting.

Application (Layer 7)

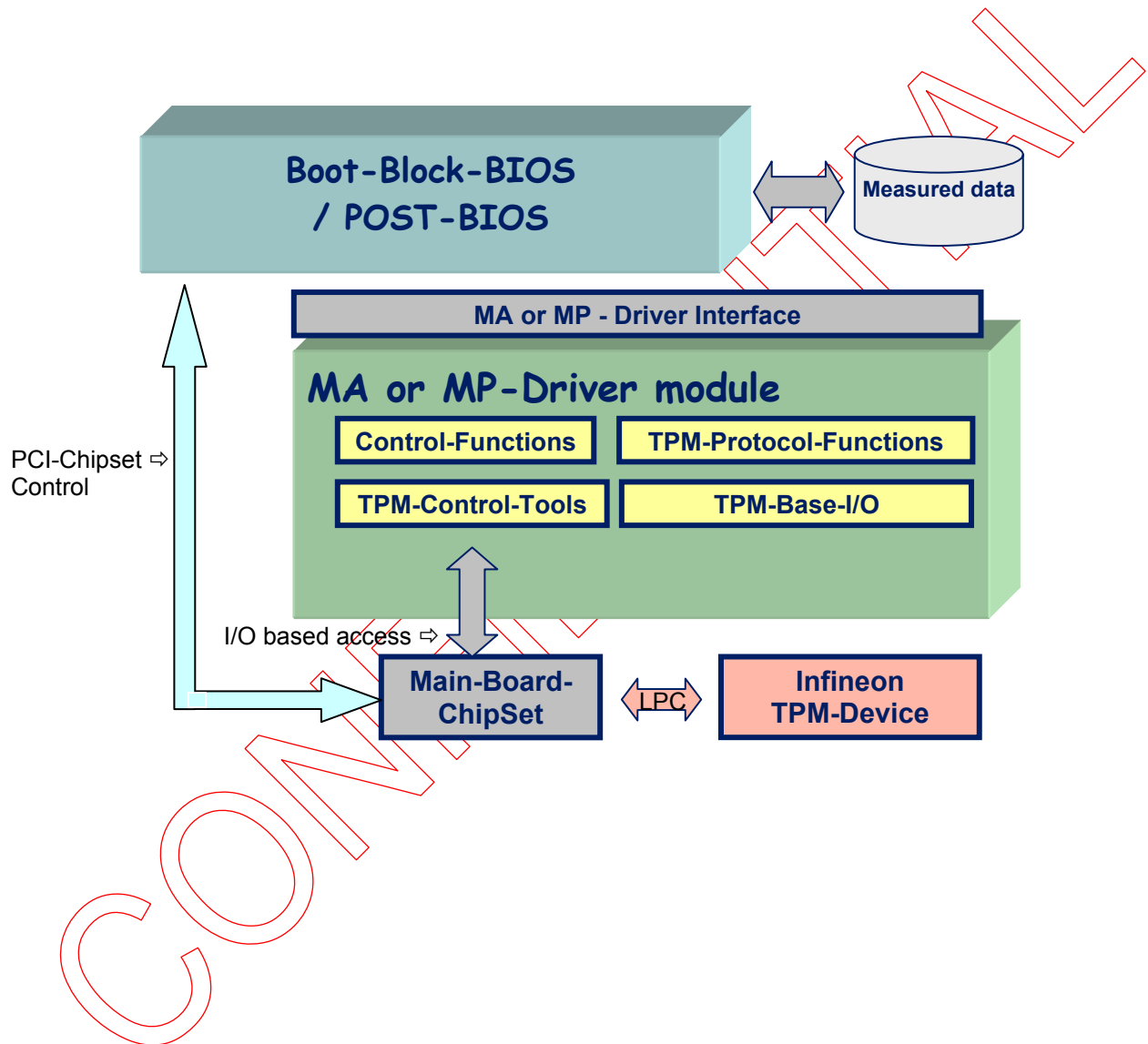
Provides services to the users of the OSI environment. It provides services for FTP, transaction server, network management, etc.

CONFIDENTIAL

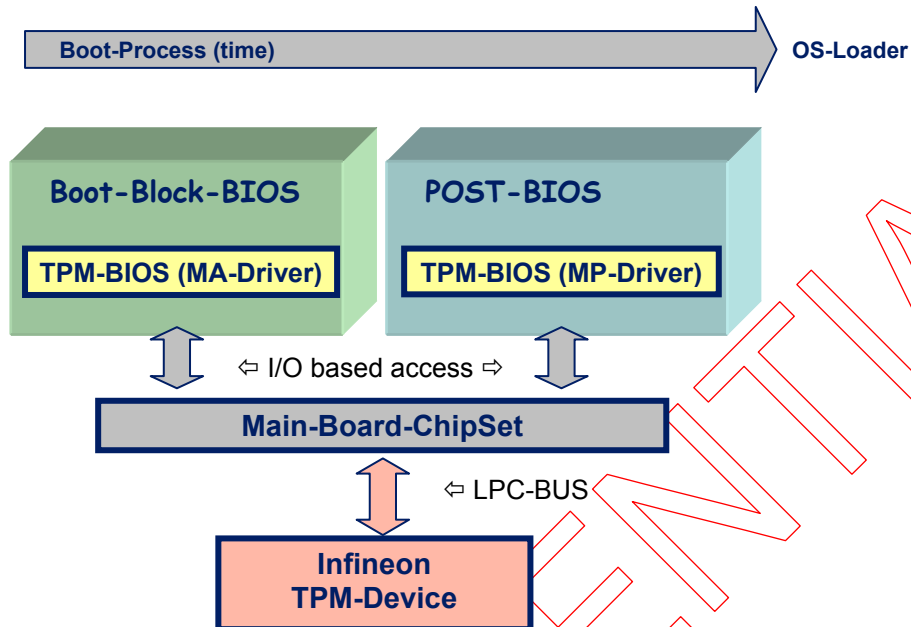
3 Product architecture

3.1 Product structure

3.1.1 BIOS-Driver System-Software Overview



3.1.2 BIOS-Driver System Flowing Overview



3.2 Basic requirements

- **CMOSTimer**

The CMOS Real Time Clock (RTC) will be available for both drivers and initialized by the caller. The RTC will be available by its legacy I/O addresses.

RTC-Index-Register: 0x70 // RTC ram index register

RTC-Data-Register: 0x71 // RTC ram data register

RTC-NMI-Mask: 0x80 // Mask for the NMI flag

RTC-REGA-Offset: 0x0A // Clock periodic and status

RTC-REGB-Offset: 0x0B // RTC interrupt control

RTC-REGC-Offset: 0x0C // RTC interrupt status

RTC-PI-BIT-Mask: 0x06 // Bit position for periodic interrupt flag

Status-Register A (RTC-REGA):

Bit:	7	6	5	4	3	2	1	0
Function:	UIP	Basis			Rate			
Default:		010			0110			

UIP: Update-Flag

Basis: Time-Base ⇒ Standard value 010 (Binary) = 32.768 Hz

Rate: Rate selector ⇒ Standard value 0110 (Binary) = 1.024 Hz

- **Motherboard Initialization**

All Motherboard chipset initialization (concerning the LPC communication channel to the TPM device) will be completed by the CRTM or POST-BIOS prior to calling the BIOS **MA-Driver** or **MP-Driver**. The LPC data transfer is available by its I/O addresses.

Setting up the Configuration Base Address:

The default location of the configuration registers of the TPM (after Reset) is defined by the setting of the BADDR pin strap option, normally this address is set to 0x4E. Alternatively, 0x2E can be set. If neither option can be set, the TPM must be configured so, that an appropriate I/O address can be used for the configuration. To be considered here is that for the initial configuration the TPM can only be addressed at 0x4E or 0x2E. After writing the new address value into the CFGH/CFGH register of the TPM and closing the config mode, the TPM can be accessed at the new address. Note that the used address must be configured in the PCI Chipset by the BIOS itself.

Setting up the Runtime Base Address:

The communication with the TPM is handled via a second I/O address space, the so-called runtime I/O space. Any available address space can be used for this purpose, the length must be 12 Bytes minimum and the base must be set on a 16-Byte boundary. The BIOS itself is responsible for a suitable setup of the PCI Chipset so that the assigned address space is routed to the LPC.

Example of the PCI Chipset setting on an Intel 82801 (ICH)

(LPC bridge functions resides in PCI-Device 31:Function0)

Configuration-Mode:

The default location for the TPM device configuration decoding is 0x4E and 0x4F (for Baddr_i pin of TPM device is set to "1" during reset by HW of the platform see also Preliminary TPM Data-Book). This setting is controlled by the Bit 13 at the LPC_EN-LPC I/F Enables register (Offset address 0xE6-0xE7) on LPC I/F-D31:F0 (e.g. ICH2 or ICH3 and equivalent). Setting this bit to one enables the decoding of the I/O addresses 0x4E and 0x4F to the LPC interface. This range is used by the TPM device for configuration.

RunTime-Mode:

The communication with the TPM device is handled over the address of the generic I/O decoder range 2. The basic location for this is controlled with the GEN2_DEC-LPC I/F Generic Decoder Range 2 register (Offset address 0xEC-0xED) on LPC I/F-D31:F0 (e.g. ICH2). The Bit 0 must be set that the GEN2 I/O range is forwarded to the LPC interface.
(The address is aligned on a 64-byte boundary and must have address lines 31:16 as 0)

Example value for the register with the default TPM-Bios-Driver addresses 0x4700:

Basic address	Bit 0 for LPC access	GEN2 register value
0x4700	OR 0x0001	= 0x4701

SERIRQ-Control:

Bit 6 in the SERIRQ_CNTL register on LPC I/F-D31:F0 (e.g. ICH2) control the mode of the serial IRQ. The preferred mode for the IFX-TPM device is the serial IRQ continuous mode; it is enabled by setting this Bit to one.

- **ACPI integration**

To communicate the resources assigned by the platform BIOS to the OS-level device driver, an appropriate ACPI table entry must be available. All chipset initializations required to provide the routing of the assigned config and runtime address to the LPC bus must be established by the resource management of the main platform BIOS.

See the following example:

```
// TPM device at LPC-Bus
Scope(\_SB.PCI0.LPC)
{
    Device (TPM)
    {
        Name (_HID, EISAID("IFX0102"))
        Name (_CID, EISAID("PNP0C31"))
        Name (_STR, Unicode ("Infineon Trusted Platform Module"))
        // Friendly name displayed at the "New HW Found" dialogue
        Name (_CRS, ResourceTemplate()
        {
            IO (Decode16, 0x004E, 0x004E, 0x01, 0x02)
            //IO config 4Eh-4Fh
            IO (Decode16, 0x4700, 0x4700, 0x01, 0x0C)
            //IO runtime 4700h-470Ch
            // address 0x4700 is just an example
            // and final assignment is up to
            // platform manufacturer
            Memory32Fixed (ReadWrite, 0xFED40000, 0x5000,)
            IRQNoFlags () {5,6,7,8,9,10,11,12,13,14,15}
            // IRQ Config
            // Interrupt listed is just an example
            // and final assignment is up to
            // platform manufacturer
        })
    }
}
```

Note that the PnP ID "IFX0102" is valid for all Infineon SLB 9635 TT 1.2 and SLD 9630 TT 1.1 TPM devices, while both I/O base addresses (0x4E and 0x4700) and interrupt flags listed are only examples and must reflect the real setting.

- **Basic requirements**

The BIOS drivers MUST fulfill the following requirements:

- The drivers MUST be completely self-contained since no BIOS services should be used;
- The drivers MUST check the validity of all the input parameters;
- The drivers are responsible to add and remove all TPM-Vendor specific protocol information to the TCG-Transfer-Data (TCG-Command);
- The drivers MUST handle all and pay attention to all communication and initialization waiting and delay times of the TPM device;

3.2.1 Object Format of BIOS Drivers

Both drivers provide a standard object format to the BIOS vendor as described in this section.

The table below describes what the header of the BIOS drivers will look like and where the driver code should start. The BIOS will move the driver into high memory, and then call the start code of the driver. The driver code MUST be relocate-able and MUST be 32-bit code, capable of running in a flat segment memory model.

During one call (entry and exit of the function) the address location of the loaded driver image must be unchanged.

• **BIOS Driver Header for MA and MP with vendor specific extensions**

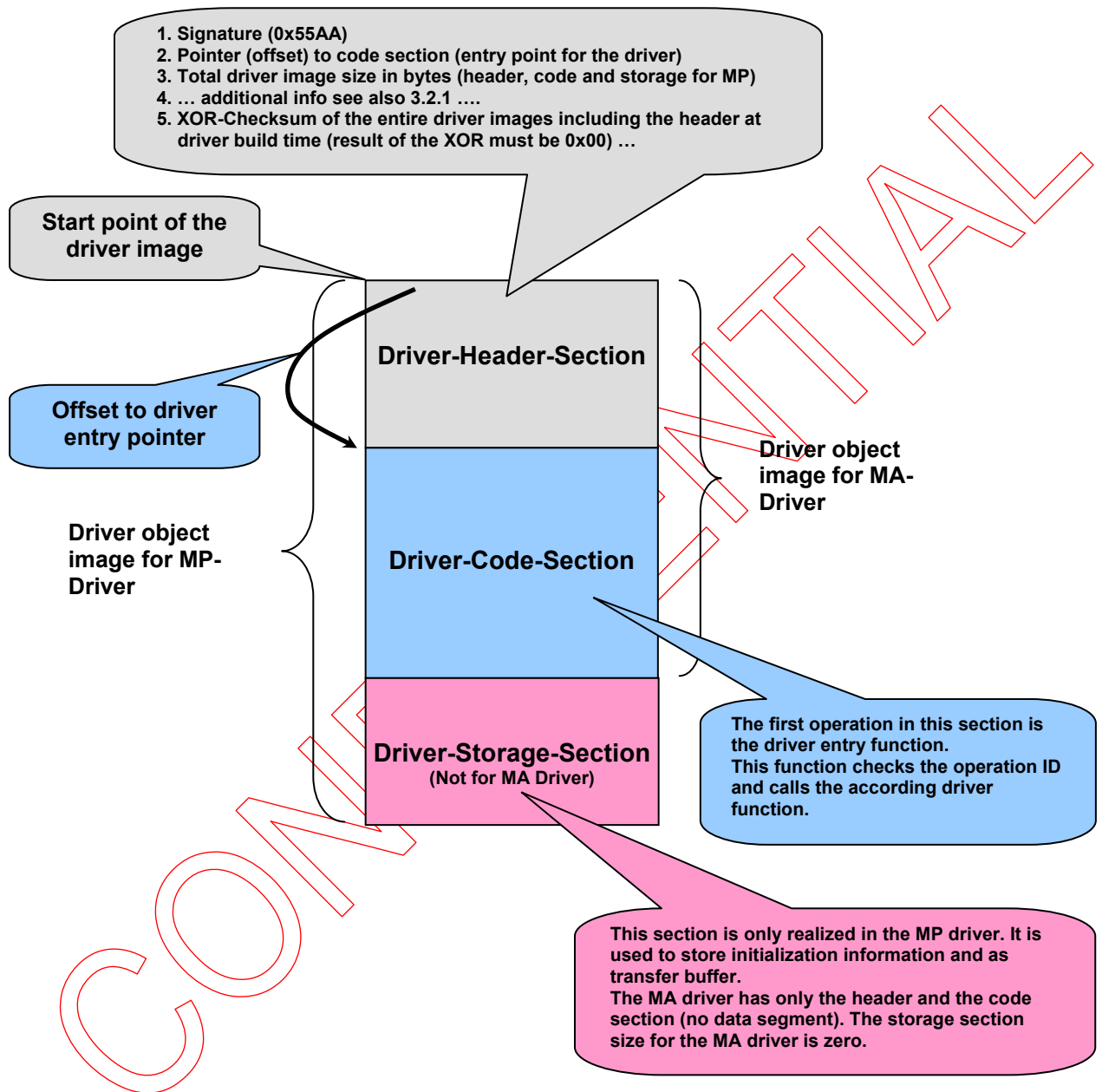
Offset	Size	Default-Value	Description
00h	WORD	55AAh	Signature used to designate the start of the BIOS driver. This is deliberately set different than the Option ROM header.
02h	DWORD		Pointer to beginning of code (Offset to entry point for the driver).
06h	WORD		Total size of the driver in bytes (including the header).
08h	DWORD	00004700h	Base address of the TPM (as set by BIOS). This is set by the resource manager of the BIOS (or OS).
0Ch	DWORD	0000004Eh *	Optional 2nd base address. This is for memory and I/O mapped or decoding I/O location/address (as set by BIOS).
10h	BYTE	FFh	IRQ Level (00h is not assigned FFh is not required) (as set by BIOS and MUST be sharable).
11h	BYTE	FFh *	DMA Channel (FFh in none assigned) (as set by BIOS).
12h	BYTE		XOR-Checksum of entire driver including this header at driver builds time. This is not maintained by the BIOS.
13h	BYTE	00h	Reserved and set to zero.
14h	DWORD	00000000h *	PCI PFA if appropriate (as set by BIOS).
18h	DWORD	00000000h *	USB, CardBus, etc (as set by BIOS).
1Ch	DWORD	0000004Eh	TPM Configuration Address from v1.2 version on. This is for I/O mapped or decoding I/O location/address (as set by BIOS).
↓	↓	↓	↓ Start of IFX-TPM vendor specific header data section. ↓
20h	BYTE	02h	Selects the type of the TPM interrupt (see also TPM Data-Book section about Interrupt configuration register) (vendor specific data area).
21h	BYTE	55h	TPM enable mask(key) for index/data register pair
22h	BYTE	00h	Driver interrupt operation mode. Selects the mode for the STI/CLI operations in the timer functions of the drivers. 0 => enable interrupt control, i.e. use CLI/STI 1 => disable interrupt control, i.e. don't use CLI/STI (The caller has to ensure that Interrupts are disabled)
23h	BYTE	00h	Reserved for TPM vendor and set to zero.
24h	BYTE	00h	Return mode selector for the drivers 0 => use the RETF instruction to return from driver. 1 => use the RET instruction to return from driver.
25h	BYTE	00h	Reserved for TPM vendor and set to zero.
26h	WORD	0000h	Reserved for TPM vendor and set to zero.
↑	↑	↑	↑ End of IFX-TPM vendor specific header data section. ↑
28h			Entry point into driver.

(*) marked header entries are ignored by the MA- and MP-Driver.

The TPM device has two special register, Index-Register and Data-Register (address of Data-Register is Index-Register-Address + 1) this register pair is used by the IFX-TPM-Bios-Drivers to configure the TPM device. The base address (Index-Register) of the device must be mapped to the I/O address specified in this header entry. It must be set by the BIOS prior the first function call to the driver.

This is the generic basic I/O address for the runtime mode communication with the TPM device it is set by the BIOS prior the first function call to the driver. The size of this range must be 16 bytes minimum.

3.2.2 BIOS-Drivers Object Image Layout (MA and MP)



4 MA-Driver Module Architecture

4.1 Introduction

This driver is designed to operate in a very limited environment. Specifically, it operates without memory, using only the CPU registers for data storage. The driver MUST be completely self-contained since no BIOS services will be available.

It is expected to be used in the BIOS Boot Block of Compound BIOS's. It is not required for Motherboards containing an Integrated BIOS to implement this driver; instead they may choose to implement only the Memory Present (MP) Driver.

The purpose of the MA Driver is to hash and extend the first portion of BIOS code before jumping to that code. The MA driver and TPM MUST perform the hash and extend operation in less than two (2) seconds. The integration of the MA-Driver image must be take place in the build process of the BIOS-Boot-Block because there is no memory available in this environment all resources and pointer information and location must be resolved to the fixed value or place.

4.1.1 MA Driver Limitations

- No DMA
- No IRQ
- Use the RET/RETF instruction to return to the caller.
- No Physical Memory
- MA-Driver Register usage table (General-Purpose and Segment register):

Register	Size	In / Out	Description
EAX	32	Not available	Driver must preserve this register.
EBX	32	Not available	Driver must preserve this register.
ECX	32	In / Out	Driver I/O; Set by the caller.
EDX	32	In / Out	Driver I/O; Set by the caller.
ESI	32	Not available	Driver must preserve this register.
EDI	32	Not available	Driver must preserve this register.
ESP	32	In (Offset)	Offset of the pointer to argument packet see Section 4.2.1. Set by the caller.
SS	16	In (Segment)	Segment of the pointer to argument packet see Section 4.2.1. Set by the caller.

- All other registers MAY be used as working registers by the MA driver without preserving them.
- The IA-32 processor (PIII, Athlon or equivalent processor) architecture supports MMX/ 3DNow and FPU. It MAY be negotiated between the BIOS vendor (more specifically the vendor of the Core RTM) and the supplier of the Core-RTM-Driver (typically the TPM vendor) that this Driver can use the MMX/3DNow register MM0 through MM7 as working registers. (Note: The MMX registers are mapped to the physical location of the floating-point registers (R0 through R7). This means when a value is written into an MMX register using an MMX instruction, the value also appears in the corresponding floating-point register.)

4.1.2 MA-Driver Basic requirements

- The MA-Driver uses the MMX/3DNow register MM0 through MM7 as working registers and the contents of this registers are not saved no prior initializations are necessary.

4.2 MA-Driver Parameters and Structers

4.2.1 MA-Driver argument structure

On entry to the MA driver, SS:ESP points to an instance of this structure. The CRTM MAY have one or more of these structures per function to allow multiple calls into a single function from different locations.

```

MADriverArgPacketStruct          STRUC
    ReturnAddr      DD      ?      ; [IN] Return address.Allows driver to return via RET/RETF.
    HeaderPtr       DD      ?      ; [IN] Pointer to the BIOS Driver Header.
    FunctionNum     DB      ?      ; [IN] Function number identifying the function to
                                   perform.
MADriverArgPacketStruct          ENDS
  
```

4.2.2 Parameter pblnBuf

BYTE *pblnBuf	
Description	Pointer to start address of the input data for the data transfers to TPM.

4.2.3 Parameter dwlnPCRlen

DWORD dwlnPCRlen	
Description	Upper 16 bits contains the PCRIndex. The lower 16 bits contain the length of the input data record – 1. (i.e., FFFFh hashes 65536 bytes.)

4.2.4 Parameter bMAInitTPMFctId

BYTE bMAInitTPMFctId	
Description	<p>Selects the TPM-Operation for the CRTM-Driver initialization.</p> <p>0000h = No TPM-Operation is selected.</p> <p>To activate the TPM_Startup command set this parameter with a T CPA_STARTUP_TYPE identifier specified in the Main Specification (see TPM_Startup section in Main Specification).</p>

4.2.5 Parameter bMAPhyPresenceTPMCmdId

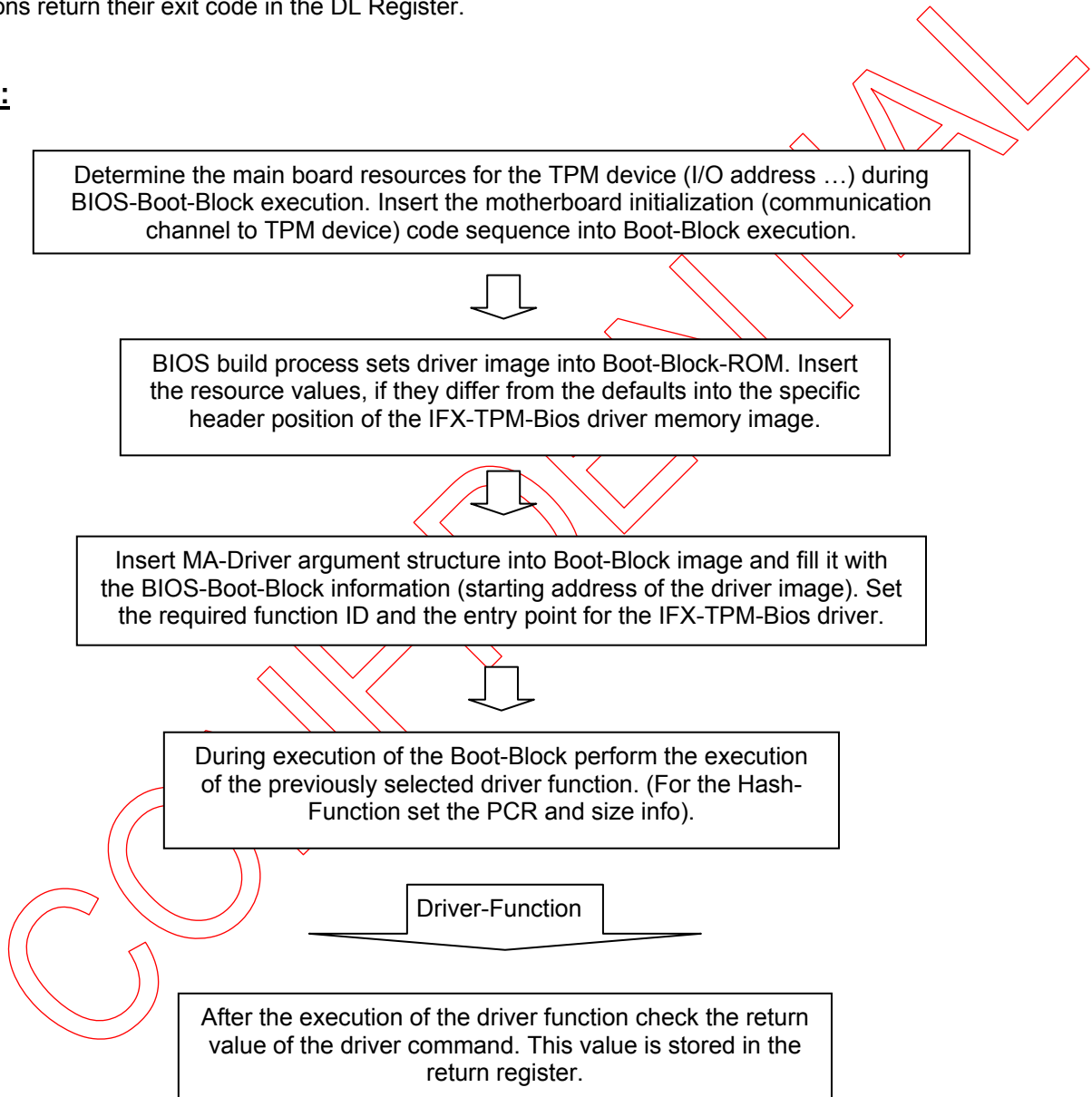
BYTE bMAPhyPresenceTPMCmdId	
Description	<p>Selects the TPM-Operation for the PhysicalPresence command.</p> <p>This value is used in the TPM-Param-Block of the TSC_PhysicalPresence command. For the detailed definition of this identifier please use the T CPA-Main Specification.</p>

4.3 MA-Driver Functions

4.3.1 MA-Driver Function Interface

The function number is contained in the FunctionNum field of the MADriverArgPacketStruct structure. The base for the function numbers is 01h. The offset for vendor specific driver function numbers is 80h. All functions return their exit code in the DL Register.

Flow:



4.3.1.1 Function MAInitTPM (Function Number: 01h)

The first call to the MA Driver must execute this function. This function does the initialization of the TPM and establishes and verifies the communication (with the parameters from the header) between the MA Driver and the TPM. All motherboard initializations (see also section 3.2) must be completed prior calling this function. If a TCG operation is selected by the bTPMInitCRTMFctId parameter this function will send the command string to the TPM.

Keep in mind if you intend to execute additional TCG commands (e. g. Hashing) on the TPM device a successful execution of a TPM_Startup operation is required (for additional information see also the TCGA-Main-Specification).

A TPM device can be opened with the same address only once by one host at a time. If the requested access cannot be granted (e.g., invalid input parameter) or if opening the connection to the TPM ends unsuccessfully, the function returns corresponding errorCode.

BYTE MAInitTPM (BYTE bMAInitTPMFctId);

Input Parameters

DL = bMAInitTPMFctId
Function identifier for the TPM_Startup operation.

Return Value

DL = return value of this function

One of the following values:

TPM_OK
TPM_IS_LOCKED
TPM_NO_RESPONSE
TPM_INVALID_RESPONSE
TPM_RESPONSE_TIMEOUT
TPM_INVALID_ACCESS_REQUEST
TPM_FIRMWARE_ERROR
TPM_GENERAL_ERROR
TPM_TRANSFER_ABORT
TPM_TCG_COMMAND_ERROR

Example:

//// Pseudo code segment to demonstrate the driver function call in 32Bit flat mode
//// environment.

TPM_OFFSET_DRVENTRY EQU 0x02

e.g. CallTPMBiosMADrvFct(BYTE bDrvFctId, BYTE bTPMDrvSubFctId)

```
{
    WORD  wDrvRetVal = RET_SUCCESS;

    // setup the MA-Driver calling structure
    MADriverArgPacket_TYPE  DrvArgPara;
    ZeroMemory(&DrvArgPara, sizeof(MADriverArgPacket_TYPE));

    if ( (bDrvFctId == MAInit_FctNr) || (bDrvFctId == MAPhyPre_FctNr) )
    {
        __asm  // asm block for driver call
        {
            pushad                                // save the registers
            push    ebp
            lea     eax, ReturnPoint0             // load the return jump address
            mov     DrvArgPara.ReturnAddr, eax    // and store it in the argument structure
            mov     eax, dwDrvLoadPoint           // load the driver image pointer
            mov     DrvArgPara.HeaderPtr, eax     // and store it in the argument structure
            //
            // calculate the driver entry point
            //
            mov     ebx, eax                      // and load it into EBX-Register
            add     ebx, TPM_OFFSET_DRVENTRYINFO // set the pointer to the offset info
            add     eax, [ebx]                    // calculate the driver calling address
            mov     ebx, eax                      // and move it to ebx

            mov     al, bDrvFctId                 // load the function id
            mov     DrvArgPara.FunctionNum, al    // and store it in the argument structure
            mov     dl, bTPMDrvSubFctId          // set the Sub-Function number
            mov     edi, esp                      // save the stack pointer
            lss     esp, DrvArgPara              // set the pointer register to the arguments
            jmp     ebx                           // and jump to the driver entry point

ReturnPoint0:
            //
            // ATTENTION: keep in mind all IFX-TPM driver function executes a RETF instruction as default return operation
            //
            mov     esp, edi                      // restore the stack pointer
            pop     ebp
            mov     wDrvRetVal, dl               // read the return info from the driver
            popad                                   // and restore the registers
        }
    }
} // end of CallTPMBiosMADrvFct
```

Sample explanations (the names are totally random):

asmTPMDrvPrg = pointer to MA-Driver entry point (see also driver header description at 3.2.1 and 3.2.2)
bTPMDrvSubFctId = driver function operation selector (DL-Register parameter see 4.3.1.1 and 4.3.1.3)
dwDrvLoadPoint = start address of the driver image (see also driver header description at 3.2.1 and 3.2.2)

4.3.1.2 Function MAHashAllExtendTPM (Function Number: 02h)

This function sends TCG hash operations to the TPM device to hash the specified (low part of dwInPCRLen) memory range. This function performs TPM_SHA1Start, TPM_SHA1Update, and TPM_SHA1CompleteExtend to the PCR specified in *dwInPCRLen* parameter. It transmits the data from the input buffer (**pbInBuf*) to the TPM and reads the response from the TPM. After successful Power-On and opening a TPM connection, the host can use this function to measure the first partition of the POST BIOS. This function is responsible for block chaining and error handling during the interaction with the TPM device over the communication interface. All vendor specific transport protocol information are added and removed by this function. If no open connection to a TPM device is available, if this function receives no valid response from the TPM, if the function calling parameters are invalid or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MAHashAllExtendTPM (DWORD **pbInBuf*, DWORD *dwInPCRLen*);

Input Parameters	<i>EDX</i> = <i>*pbInBuf</i> Pointer to the start address of input buffer containing the data for the TPM device (see 4.2.2). <i>ECX</i> = <i>dwInPCRLen</i> PCRIndex and Length of the input buffer data (see 4.2.3).
Return Value	<i>DL</i> = return value of this function One of the following values: TPM_OK TPM_IS_LOCKED TPM_NO_RESPONSE TPM_INVALID_RESPONSE TPM_RESPONSE_TIMEOUT TPM_INVALID_ACCESS_REQUEST TPM_FIRMWARE_ERROR TPM_GENERAL_ERROR TPM_TRANSFER_ABORT TPM_TCG_COMMAND_ERROR

Example:

//// Pseudo code segment to demonstrate the driver function call in 32Bit flat mode
//// environment.

TPM_OFFSET_DRVENTRY EQU 0x02

e.g. **CallMAHashAllExtendDrvFct**(BYTE *pInBufAddr, WORD wBufSize, WORD wPCRInd)
{
 WORD wDrvRetVal = RET_SUCCESS;

 // setup the MA-Driver calling structure
 MADriverArgPacket_TYPE DrvArgPara;
 ZeroMemory(&DrvArgPara, sizeof(MADriverArgPacket_TYPE));
 wBufSize--; // see sample explanations

```
__asm // asm block for driver call
{
    pushad // save the registers
    push    ebp
    lea     eax, ReturnPoint0 // load the return jump address
    mov     DrvArgPara.ReturnAddr, eax // and store it in the argument structure
    mov     eax, dwDrvLoadPoint // load the driver image pointer
    mov     DrvArgPara.HeaderPtr, eax // and store it in the argument structure
    //
    // calculate the driver entry point
    //
    mov     ebx, eax // and load it into EBX-Register
    add     ebx, TPM_OFFSET_DRVENTRYINFO // set the pointer to the offset info
    add     eax, [ebx] // calculate the driver calling address
    mov     ebx, eax // and move it to ebx
}
//
// set the hash arguments for driver call
//
    mov     edx, pInBufAddr // load the start address of input buffer
    mov     cx, wPCRInd // load the PCR index number
    rol     ecx, 010h // shift it in the high word
    mov     cx, wBufSize // load the length info of the input buffer

    mov     al, bDrvFctId // load the function id
    mov     DrvArgPara.FunctionNum, al // and store it in the argument structure
    mov     edi, esp // save the stack pointer
    lss     esp, DrvArgPara // set ss:esp points to the arguments
    jmp     ebx // and jump to the driver entry point
ReturnPoint0:
//
// ATTENTION: keep in mind all IFX-TPM driver function executes a RETF instruction as default return operation
//
    mov     esp, edi // restore the stack pointer
    pop     ebp
    mov     wDrvRetVal, dl // read the return info from the driver
    popad // and restore the registers
} // end of CallMAHashAllExtendDrvFct
```

Sample explanations (the names are totally random):

asmTPMDrvPrg = pointer to MA-Driver entry point (see also driver header description at 3.2.1 and 3.2.2)
 dwDrvLoadPoint = start address of the driver image (see also driver header description at 3.2.1 and 3.2.2)

Attention: Keep in mind the size definition of the hash data blob see 4.2.3

4.3.1.3 Function MAPHysicalPresenceTPM (Function Number: 03h)

This function sends the TSC_PhysicalPresence operations with the command value specified in the *bMAPhyPresenceTPMCmdId* parameter to the TPM device.

If no open connection to a TPM device is available, if this function receives no valid response from the TPM, if the function calling parameters are invalid or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MAPHysicalPresenceTPM (BYTE *bMAPhyPresenceTPMCmdId*);

Input Parameters	<i>DL = bMAPhyPresenceTPMCmdId</i> Command identifier for the TPM_PhysicalPresence operation (see 4.2.5).
Return Value	<i>DL = return value of this function</i> One of the following values: TPM_OK TPM_IS_LOCKED TPM_NO_RESPONSE TPM_INVALID_RESPONSE TPM_RESPONSE_TIMEOUT TPM_INVALID_ACCESS_REQUEST TPM_FIRMWARE_ERROR TPM_GENERAL_ERROR TPM_TRANSFER_ABORT TPM_TCG_COMMAND_ERROR

Example:

See section 4.3.1.1 (only other function number)

5 MP-Driver Module Architecture

5.1 Introduction

The MP Driver is a module of the TCG software for the TPM device. The main goal for the MP Driver is to support the customer in their BIOS integration of the TPM control and communication software. The driver includes functions for the handling of the data block transmission protocol between the TPM device and the host system.

As discussed above, the MP-Driver will need to be 32-bit relocate-able code. The BIOS code will bring up memory load the MP-Driver-Image and call the start of the driver. Prior to calling the MP Driver, the BIOS will set a base address for the TPM. This base address will be stored as part of the driver header. All of the configuration data (not JUST the base address) will be set by the BIOS prior to calling the MP-Driver. All the data transfers from and to the TPM are done through this module. Through this module a Host system reads, writes and controls the TPM. The MP-Driver communicates through the ChipSet-LPC-Interface with the TPM device.

5.1.1 MP Driver Limitations

- No DMA
- No IRQ
- All registers used as working registers by the MP driver must be restored after the execution of the driver function.
- Use the RET/RETF instruction to return to the caller.
- The IA-32 processor (PIII, Athlon or equivalent processor) architecture supports MMX/ 3DNow and FPU. It MAY be negotiated between the BIOS vendor (more specifically the vendor of the Core RTM) and the supplier of the Core-RTM-Driver (typically the TPM vendor) that this Driver can use the MMX/3DNow register MM0 through MM7 as working registers. (Note: The MMX registers are mapped to the physical location of the floating-point registers (R0 through R7). This means when a value is written into an MMX register using an MMX instruction, the value also appears in the corresponding floating-point register.)

5.1.2 MP-Driver Basic requirements

- The MP driver MAY be relocated after MPInitTPM and at any time between call MP driver functions.
- MP Driver needs to be placed into ACPI non-reclaimable area. The driver MUST support being relocated between calls.
- The resources allocated to the TPM MAY be changed by the BIOS between calling MP driver functions, therefore, the MPInitTPM function MUST be recallable.
- All registers not used for return parameters MUST be preserved.
- MP Driver needs to be built such that it has any data memory it requires is part of the body of the driver image

5.2 MP-Driver Parameters and Structures

5.2.1 Parameter pbInBuf

BYTE *pbInBuf	
Description	Pointer to input data for the data transfers to TPM.

5.2.2 Parameter pbOutBuf

BYTE *pbOutBuf	
Description	Pointer to output buffer for the data transfers from the TPM.

5.2.3 Parameter dwInLen

DWORD dwInLen	
Description	Length of the input data record.

5.2.4 Parameter dwOutLen

DWORD dwOutLen	
Description	DWORD provides the length of the output buffer as input and stores the length info of the return data record as output.

5.2.5 Structure TPMTransmitEntry

This structure is used by the TPMTransmit function to transfer the input and output parameters. The two output parameters (pbOutBuf, dwOutLen) can be NULL-Pointers if no response is necessary or it has no meaning for the caller. This mode can be also helpful in memory less environments (e.g. BIOS-Boot-Block).

```

TPMTransmitEntryStruct  STRUC
    pbInBuf              DD    ?    ; [IN]    Pointer to input data for the data
                                transfers to TPM.
    dwInLen              DD    ?    ; [IN]    Length of the input data record.
    pbOutBuf             DD    0    ; [OUT]    Pointer to output buffer for the data from
                                the TPM.
    dwOutLen             DD    0    ; [IN/OUT]  DWORD to store the length info of the
                                output data record.
TPMTransmitEntryStruct  ENDS

```

The parameter dwOutLen is both an input and output parameter:

As input (entry point of this function) it specifies the maximum number of bytes, which can be read from the TPM device to the output buffer. If the function terminates successfully the value of this variable is adjusted to match with the number of bytes received from the TPM.

5.2.6 Parameter lpTPMTransInfo

TPMTransmitEntryStruct *lpTPMTransInfo	
Description	Pointer to a TPMTransmitEntryStruct , which carries the input and output parameters for data transfer between host system and TPM device.

5.3 MP Driver Functions

5.3.1 MP Driver Function Interface

The AL-Register contains the function selector number for the different functions of this driver (the base for this is **01h**). The offset for vendor specific driver function numbers is 80h. All these functions returns there exit code in AL-Register.

5.3.1.1 Function MPInitTPM (Function-Nr-AL-Register: 01h)

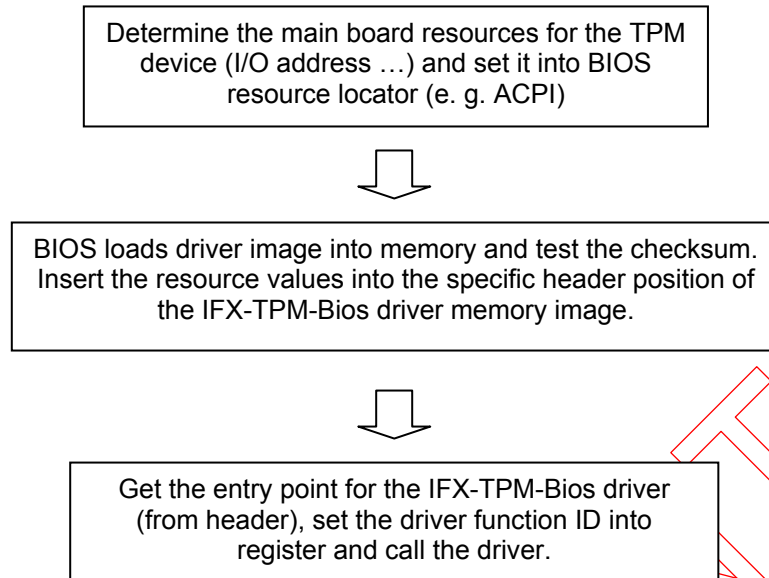
Initializes the MP-Driver on the current parameters from the header and identifies the TPM device. The application (BIOS) must initialize the MP-Driver before they can call MP-Driver-Library functions. If the BIOS unloads the driver image and reloads it, then it is necessary to call the MPInitTPM function for the new instance again. This function is used to initialize the TPM if not already done by the BIOS Boot Block or if there are some differences between the communication parameters for the CRTM and POST-Phase. This function must be also called (recalled) if the BIOS has the requirement to change some communication parameters (e. g. I/O address) used by the TPM (such as if BIOS performs PnP conflict resolution). This function does the initialization of the TPM and the driver and establishes (opens a connection) and verifies the communication (with the parameters from the header) between the MP-Driver and the TPM device.

The MP-Driver always runs in polling mode to handle the communication with the TPM device. But this MPInitTPM function arranges the interrupt number in the TPM configuration space, which is used by the OS-Driver environment. If the interrupt number is set to 0xFF no interrupts are generated. This means the interrupts are disabled in the TPM device and the communication runs in polling mode this is the default mode.

A TPM device can be opened with the same address only once by one host at a time. If the requested access cannot be granted (e. g. invalid input parameter) or if opening the connection to the TPM ends unsuccessfully, the function returns corresponding *errorCode*.

BYTE MPInitTPM (void);	
Input Parameters	<i>All necessary inputs are located in the driver header structure (see 3.2.1).</i>
Output Parameters	<i>None</i>
Return Value	<i>AL = return value of this function</i> One of the following values: TPM_OK TPM_INVALID_ADR_REQUEST TPM_IS_LOCKED TPM_INVALID_DEVICE_ID TPM_INVALID_VENDOR_ID TPM_RESERVED_REG_INVALID TPM_FIRMWARE_ERROR TPM_UNABLE_TO_OPEN TPM_GENERAL_ERROR

Flow:



Example:

//// Pseudo code segment to demonstrate the driver function call in 32Bit flat mode
 //// environment.

```

TPM_OFFSET_DRENTY EQU 0x02
MPInitTPM_FctNr EQU 0x01

int IRCVal = RET_SUCCESS;
;
;call the MPInitTPM function
;
mov eax, dwDrvLoadPoint ; get the start point of the loaded driver memory image
;
; load the offset for the driver entry info
;
mov ebx, eax ; and load it into EBX-Register
add ebx, TPM_OFFSET_DRENTYINFO ; set the pointer to the offset info
add eax, [ebx] ; calculate the driver calling address
mov ebx, eax ; and move it to ebx
mov eax, MPInitTPM_FctNr ; load the function number to AL register
call ebx ; call the IFX-TPM driver function (MPInitTPM)
;
; ATTENTION: keep in mind all IFX-TPM driver function executes a RETF instruction as default return operation
;
if (function number == MPGetTPMStatusInfo_FctNr)
    IRCVal = eax; // load the return value from the driver
else
    IRCVal = al // load the return value from the driver
  
```

Sample explanations (the names are totally random):

dwDrvLoadPoint = start address of the driver image (see also driver header description at 3.2.1 and 3.2.2)

5.3.1.2 Function MPCloseTPM (Function-Nr-AL-Register: 02h)

This operation closes a connection to a TPM device with the specified parameters in the header. All data related to this connection to the device, such as allocated memory, are released. The registers in the configuration space of the TPM device are reinitialized to the reset status and the logical device is deactivated.

If the specified parameters in the header are not valid, or if closing of the connection to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MPCloseTPM (void);	
Input Parameters	<i>All necessary inputs are located in the driver header structure (see 3.2.1).</i>
Output Parameters	<i>None</i>
Return Value	<i>AL = return value of this function</i> One of the following values: TPM_OK TPM_INVALID_ADR_REQUEST TPM_UNABLE_TO_CLOSE TPM_GENERAL_ERROR

Example:

See section 5.3.1.1 (only other function number)

5.3.1.3 Function MPGetTPMStatusInfo (Function-Nr-AL-Register: 03h)

This function reads the current error and status information from the TPM device. All data related to this connection, such as allocated memory, are still valid.

If the specified parameters in the header are not valid, or this device is not yet open, the function fails and returns an error flag.

DWORD MPGetTPMStatusInfo (void);	
Input Parameters	<i>All necessary inputs are located in the driver header structure (see 3.2.1).</i>
Output Parameters	<i>None</i>
Return Value	<i>EAX = return value of this function</i> <i>For the coding of the return value see 5.3.1.3.1.</i>

Example:

See section 5.3.1.1 (only other function number)

5.3.1.3.1 Return-Values for MPGetTPMStatusInfo (Function: 03h)

If the return value is **zero** no error condition is active for this TPM connection this status is the OK-Status of the TPM device.

DWORD-Return-Value	
Bit	Descriptions
0	If set a general error condition is active for this TPM connection. For details evaluate the condition of the following error information (Bit 1:15).
1	Invalid status/error request access.
2	If set a general firmware error occurred during start up of the TPM firmware.
3	Time out occurred during send process of the request sequence to the TPM device.
4	Response time out in TPM communication.
5	Transfer communication abort with the TPM device.
6	Reserved. This bit is read-only and has a value of 0.
7	Reserved. This bit is read-only and has a value of 0.
8	Reserved. This bit is read-only and has a value of 0.
9	Reserved. This bit is read-only and has a value of 0.
10	Reserved. This bit is read-only and has a value of 0.
12	Reserved. This bit is read-only and has a value of 0.
13	Reserved. This bit is read-only and has a value of 0.
14	Reserved. This bit is read-only and has a value of 0.
15	Reserved. This bit is read-only and has a value of 0.
16	If set a general status information is available for this TPM. For details evaluate the condition of the following status information (Bit 17:31).
17	The TPM device is not personalized (e. g. Endorsement key pair is missing).
18	Integrity discrepancy in the TPM initialization.
19	Self-Test of TPM device complete.
20	Data transmission with TPM device active.
21	Reserved. This bit is read-only and has a value of 0.
22	Reserved. This bit is read-only and has a value of 0.
23	Reserved. This bit is read-only and has a value of 0.
24	Reserved. This bit is read-only and has a value of 0.
25	Reserved. This bit is read-only and has a value of 0.
26	Reserved. This bit is read-only and has a value of 0.
27	Reserved. This bit is read-only and has a value of 0.
28	Reserved. This bit is read-only and has a value of 0.
29	Reserved. This bit is read-only and has a value of 0.
30	Reserved. This bit is read-only and has a value of 0.
31	Reserved. This bit is read-only and has a value of 0.

5.3.1.4 Function MPTPMTransmit (Function-Nr-AL-Register: 04h)

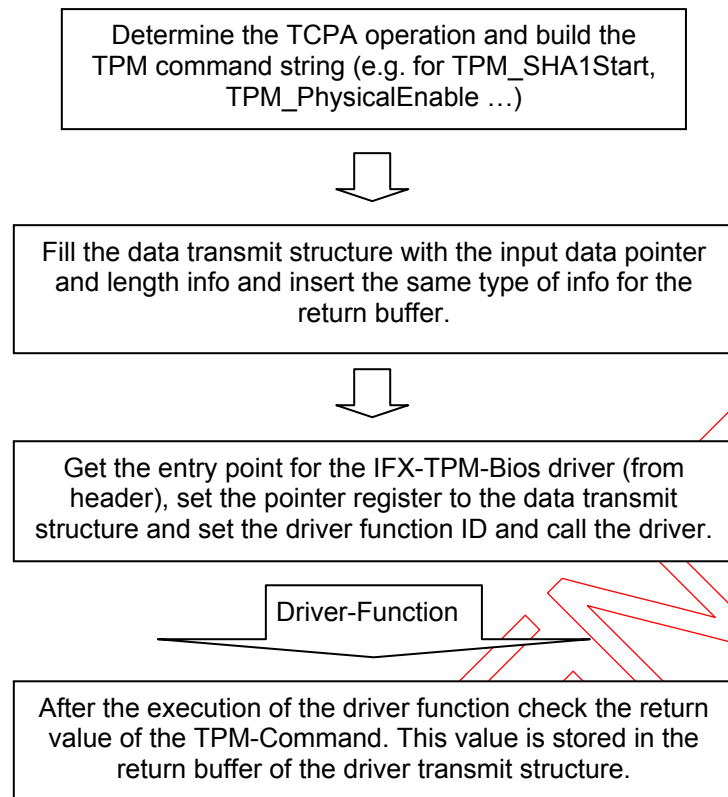
Transmits the data from the input buffer (**pbInBuf*) to the TPM and reads the response from the TPM to the output buffer (**pbOutBuf*). After successful Power-On and opening a TPM connection, the host can send the first request to the TPM by writing the bytes to the TPM device. When the request is processed by the TPM and the response is available the TPM firmware issues a receiver data available flag (and generates an interrupt if it is enabled) the host (MP-Driver) can poll to this signal and then start the data receive process. This function is responsible for error handling during the interaction with the TPM device over the communication interface.

All vendor specific transport protocol information are added and removed by this function. The input and output buffer contains only TCG-Command-Param-Lists, this data streams are opaque to this function. This means that the TCG-Command-Param-Lists in these buffers will not be interpreted or reorganized by this function.

If no open connection to a TPM device is available, if it returns no response, if the function calling parameters are invalid or the transmission of the data block to the TPM ends unsuccessfully, the function fails and returns corresponding *errorCode*.

BYTE MPTPMTransmit (MPTPMTransmitEntryStruct *lpTPMTransInfo);	
Input Parameters	<i>ESI</i> = pointer to a TPMTransmitEntryStruct (see 5.2.5). <i>pbInBuf</i> Pointer to the input buffer containing the data (TCG command string) for the TPM device (see 5.2.1). <i>dwInLen</i> Length of the input buffer data (see 5.2.3).
Input/Output Parameters	<i>dwOutLen</i> Pointer to store the length info of the received data (see 5.2.4). It also carries the size (input) of the OutBuf to store the response of the TPM device.
Output Parameters	<i>pbOutBuf</i> Pointer to the output buffer to store the data from the TPM device (see 5.2.2).
Return Value	<i>AL</i> = return value of this function One of the following values: TPM_OK TPM_IS_LOCKED TPM_NO_RESPONSE TPM_INVALID_RESPONSE TPM_RESPONSE_TIMEOUT TPM_INVALID_ACCESS_REQUEST TPM_FIRMWARE_ERROR TPM_GENERAL_ERROR TPM_TRANSFER_ABORT

Flow:



Example:

//// Pseudo code segment to demonstrate the driver function call

```

OFFSET_DRVENTRYINFO EQU 0x02
MP_DRV_TRANSMIT_FID EQU 0x04

;
; call the MPTPMTransmit function
;
mov esi, pTransStruct ; load the pointer to the driver transmit structure
mov eax, dwDrvLoadPoint ; get the start point of the loaded driver memory image
;
; load the offset for the driver entry info
;
mov ebx, eax ; and load it into EBX-Register
add ebx, OFFSET_DRVENTRYINFO ; set the pointer to the offset info
add eax, [ebx] ; calculate the driver calling address
mov ebx, eax ; and move it to ebx
mov eax, MP_DRV_TRANSMIT_FID ; load the function number to AL register
call ebx ; call the IFX-TPM driver function (MPTPMTransmit)
;
; ATTENTION: keep in mind all IFX-TPM driver function executes a RETF instruction as default return operation
;
test al ; test the return value of the driver function
  
```

Example for MPTPMTransmitEntryStruct handling (e.g. Hash-Sequence):

(see also T CPA-Main-Specification command TPM_SHA1Start, TPM_SHA1Update, TPM_SHA1CompleteExtend)

Definitions for Command-String:

```
const DWORD SHA1StartCmdLen = 10;
const DWORD SHA1UpdateCmdLen = 14;
const DWORD SHA1CompleteExtendCmdLen = 18;
const DWORD RetBufSize = 128;
const DWORD HashBlockBoundary = 64;

const DWORD TCPAPRAMSIZE_POS = 2;
const DWORD TCPARESULT_POS = 6;

const DWORD HASHDATASIZEEXT_POS = 14;
const DWORD HASHDATAEXT_POS = HASHDATASIZEEXT_POS + 4;

const DWORD HASHDATASIZEUPDATE_POS = 10;
const DWORD HASHDATAUPDATE_POS = HASHDATASIZEUPDATE_POS + 4;

const DWORD SHA1StartHashLen_Pos = 10;
```

Command-String (see also T CPA Main Specification):

```
- TPM_SHA1Start:
BYTE acSHA1Start[SHA1StartCmdLen] = {
{0x00}, {0xC1},           // TCPA_TAG = TPM_TAG_RQU_COMMAND
{0x00}, {0x00}, {0x00}, {0x0A}, // ParamSize
{0x00}, {0x00}, {0x00}, {0xA0} // TCPA_COMMAND_CODE = TPM_ORD_SHA1Start
};

- TPM_SHA1Update:
BYTE acSHA1Update[SHA1UpdateCmdLen] = {
{0x00}, {0xC1},           // TCPA_TAG = TPM_TAG_RQU_COMMAND
{0x00}, {0x00}, {0x00}, {0x00}, // placeholder for paramSize
{0x00}, {0x00}, {0x00}, {0xA1}, // TCPA_COMMAND_CODE = TPM_ORD_SHA1Update
{0x00}, {0x00}, {0x00}, {0x00} // placeholder for hashDataSize
};

-TPM_SHA1CompleteExtend:
BYTE acSHA1CompleteEx[SHA1CompleteExtendCmdLen] = {
{0x00}, {0xC1},           // TCPA_TAG = TPM_TAG_RQU_COMMAND
{0x00}, {0x00}, {0x00}, {0x00}, // placeholder for paramSize
{0x00}, {0x00}, {0x00}, {0xA3}, // TCPA_COMMAND_CODE = TPM_ORD_SHA1CompleteExtend
{0x00}, {0x00}, {0x00}, {0x01}, // TCPA_PCRINDEX = e.g. PCR #1
{0x00}, {0x00}, {0x00}, {0x00} // placeholder for hashDataSize
};
```

Inputs:

```
BYTE      *pbDataToHashBuf; // pointer to the data to be hashed
DWORD     dwHashSizeByte;   // number of byte to hash
BYTE      acRetBuf[RetBufSize]; // buffer to store the T CPA_RESULT of the T CPA command
```

Start pseudo code sequence (CallSHA1Sequenz):

```

BYTE      *pbInTransBuf = NULL;
DWORD     dwHashComExSize = 0;
DWORD     dwCurPackSize = 0;
DWORD     dwTpmRetCode = 0;
DWORD     dwLastPackSize = 0;
BYTE      *pbTmpHashBuf = NULL;

TPMTransmitEntryType TPMTransInfo;          // the MPTPMTransmitEntryStruct
TPMTransmitEntryType *pTransStruct = NULL;

ZeroMemory(acRetBuf, RetBufSize);             // clear the return buffer
ZeroMemory(&TPMTransInfo, sizeof(TPMTransmitEntryType));
// set pointer to transfer structure
pTransStruct = &TPMTransInfo;

do
{
    // insert the info to execute the SHA1Start command
    TPMTransInfo.dwInLen = sizeof(acSHA1Start);    // size of command
    TPMTransInfo.pbInBuf = acSHA1Start;           // pointer to command info
    TPMTransInfo.dwOutLen = RetBufSize;           // size of output buffer
    TPMTransInfo.pbOutBuf = acRetBuf;             // pointer to output buffer

    __asm // asm block for driver call (used for all calls in this example)
    {
        pushad // save the registers
        mov esi, pTransStruct // load the pointer to the driver transmit structure
        mov eax, dwDrvLoadPoint // get the start point of the loaded driver memory image
        mov ebx, eax // and load it into EBX-Register
        add ebx, OFFSET_DRVENTRYINFO // set the pointer to the offset info
        add eax, [ebx] // calculate the driver calling address
        mov ebx, eax // and move it to ebx
        mov eax, MP_DRV_TRANSMIT_FID // load the function number to AL register
        call ebx // call the IFX-TPM driver function (MPTPMTransmit)
        mov wDrvRetVal, ax // read the return info from the driver
        popad // and restore the registers
    }

    // read the TCPA_RESULT info from the return buffer (size is DWORD = 4 Bytes)
    memcpy(&dwTpmRetCode, &acRetBuf[TCPARESULT_POS], 4);
    // and switch the byte order
    // all TCPA structures MUST be packed on a byte boundary
    // each TCPA structure MUST use big endian bit/byte ordering
    dwTpmRetCode = Swap(dwTpmRetCode); // swap to little endian byte order

    // read the maximum number of bytes that can be send to TPM_SHA1Update
    memcpy(&dwMaxHashLoopBytes, &acRetBuf[SHA1StartHashLen_Pos], 4);
    dwMaxHashLoopBytes = Swap(dwMaxHashLoopBytes); // swap to little endian byte order

    // test the return info
    if ((dwTpmRetCode != 0) || (dwMaxHashLoopBytes == 0))
    {
        MessageBox(NULL, "Error Hash-StartUp", MB_OK, 0);
        dwRetVal = dwTpmRetCode;
        goto FinaleComp;
    }
}

```

```

pbTmpHashBuf = pbDataToHashBuf;

// for large block sizes start the SHA1Update loop
if (dwHashSizeByte > HashBlockBoundary)
{
    if (dwHashSizeByte > dwMaxHashLoopBytes)
    {
        // calculate the transfer packets size
        dwCurPackSize = sizeof(acSHA1Update) + dwMaxHashLoopBytes;
        // get memory for this packets
        pbInTransBuf = new BYTE[dwCurPackSize];
        // set the command string
        memcpy(pbInTransBuf, acSHA1Update, sizeof(acSHA1Update));

        // insert the paramSize info into command string
        dwCurPackSize = Swap(dwCurPackSize);
        memcpy(pbInTransBuf + TCPAPRAMSIZE_POS, &dwCurPackSize, 4);
        dwCurPackSize = Swap(dwCurPackSize);

        // insert the HashBlockSize info into command string
        dwMaxHashLoopBytes = Swap(dwMaxHashLoopBytes);
        memcpy(pbInTransBuf + HASHDATASIZEUPDATE_POS, &dwMaxHashLoopBytes, 4);
        dwMaxHashLoopBytes = Swap(dwMaxHashLoopBytes);

        // calculate the necessary loops for SHA1Update
        DWORD dwHashLoop = dwHashSizeByte / dwMaxHashLoopBytes;

        // set the parameter into the transfer struct
        TPMTransInfo.dwInLen = dwCurPackSize;
        TPMTransInfo.pbInBuf = pbInTransBuf;
        TPMTransInfo.dwOutLen = RetBufSize;
        TPMTransInfo.pbOutBuf = acRetBuf;

        // start the update loop
        for (int iLoopCount = dwHashLoop; iLoopCount > 0; iLoopCount--)
        {
            // add the hash data into the command buffer
            memcpy(pbInTransBuf + HASHDATAUPDATE_POS, pbTmpHashBuf,
                dwMaxHashLoopBytes);

            pbTmpHashBuf += dwMaxHashLoopBytes;
            ZeroMemory(acRetBuf, RetBufSize);

// call the driver see asm block above!!!

            // read the TCPA_RESULT info from the return buffer
            memcpy(&dwTpmRetCode, &acRetBuf[TCPARESULT_POS], 4);
            dwTpmRetCode = Swap(dwTpmRetCode);

            if (dwTpmRetCode != 0)
            {
                MessageBox(NULL, "Error Hash-Update", MB_OK, 0);
                dwRetVal = dwTpmRetCode;
                goto FinaleComp;
            }
        } // end of for loop
    }
}

```

```
        delete [] pbInTransBuf;
        pbInTransBuf = NULL;
    } // end of if (dwHashSizeByte > dwMaxHashLoopBytes)

    if (dwTpmRetCode != 0)
    {
        MessageBox(NULL, "Error Hash-Update2", MB_OK, 0);
        dwRetVal = dwTpmRetCode;
        goto FinaleComp;
    }

    // start last packets transfer
    if (((pbDataToHashBuf + dwHashSizeByte) - pbTmpHashBuf) > HashBlockBoundary)
    {
        dwCurPackSize = (((pbDataToHashBuf + dwHashSizeByte) - pbTmpHashBuf) /
                          HashBlockBoundary);

        dwLastPackSize = (dwCurPackSize * HashBlockBoundary);
        dwCurPackSize = sizeof(acSHA1Update) + dwLastPackSize;
        pbInTransBuf = new BYTE[dwCurPackSize];
        memcpy(pbInTransBuf, acSHA1Update, sizeof(acSHA1Update));

        dwCurPackSize = Swap(dwCurPackSize);
        memcpy(pbInTransBuf + TCPAPRAMSIZE_POS, &dwCurPackSize, 4);
        dwCurPackSize = Swap(dwCurPackSize);

        dwLastPackSize = Swap(dwLastPackSize);
        memcpy(pbInTransBuf + HASHDATASIZEUPDATE_POS, &dwLastPackSize, 4);
        dwLastPackSize = Swap(dwLastPackSize);

        TPMTransInfo.dwInLen = dwCurPackSize;
        TPMTransInfo.pbInBuf = pbInTransBuf;
        TPMTransInfo.dwOutLen = RetBufSize;
        TPMTransInfo.pbOutBuf = acRetBuf;

        memcpy(pbInTransBuf + HASHDATAUPDATE_POS, pbTmpHashBuf,
               dwLastPackSize);

        pbTmpHashBuf += dwLastPackSize;
        ZeroMemory(acRetBuf, RetBufSize);

        // call the driver see asm block above!!!

        // read the TCPA_RESULT info from the return buffer
        memcpy(&dwTpmRetCode, &acRetBuf[TCPARESULT_POS], 4);
        dwTpmRetCode = Swap(dwTpmRetCode);
```

```

        if (dwTpmRetCode != 0)
        {
            MessageBox(NULL, "Error Hash-Update", MB_OK, 0);
            dwRetVal = dwTpmRetCode;
            goto FinaleComp;
        }

        delete [] pblnTransBuf;
        pblnTransBuf = NULL;
    } // end of last packet transfer

    dwComExtHashLoopBytes = (pbDataToHashBuf + dwHashSizeByte) - pbTmpHashBuf;
} // end of if for large block sizes start the SHA1Update loop
else
{
    dwComExtHashLoopBytes = dwHashSizeByte;
}

// start preparation for SHA1CompleteExtend command
FinaleComp:
dwCurPackSize = sizeof(acSHA1CompleteEx) + dwComExtHashLoopBytes;
pblnTransBuf = new BYTE[dwCurPackSize];
memcpy(pblnTransBuf, acSHA1CompleteEx, sizeof(acSHA1CompleteEx));
dwCurPackSize = Swap(dwCurPackSize);
memcpy(pblnTransBuf + TCPAPRAMSIZE_POS, &dwCurPackSize, 4);
dwCurPackSize = Swap(dwCurPackSize);

dwComExtHashLoopBytes = Swap(dwComExtHashLoopBytes);
memcpy(pblnTransBuf + HASHDATASIZEEXT_POS, &dwComExtHashLoopBytes, 4);
dwComExtHashLoopBytes = Swap(dwComExtHashLoopBytes);

if (dwComExtHashLoopBytes > 0)
    memcpy(pblnTransBuf + HASHDATAEXT_POS, pbTmpHashBuf,
        dwComExtHashLoopBytes);

TPMTransInfo.dwInLen = dwCurPackSize;
TPMTransInfo.pbInBuf = pblnTransBuf;
TPMTransInfo.dwOutLen = RetBufSize;
TPMTransInfo.pbOutBuf = acRetBuf;

ZeroMemory(acRetBuf, RetBufSize);

// call the driver see asm block above!!!

// read the TPCA_RESULT info from the return buffer
memcpy(&dwTpmRetCode, &acRetBuf[TCPARESULT_POS], 4);
dwTpmRetCode = Swap(dwTpmRetCode);

If (dwTpmRetCode != 0)
{
    MessageBox(NULL, "Error Hash-CompleteExtend", MB_OK, 0);
    dwRetVal = dwTpmRetCode;
    break;
}

delete [] pblnTransBuf;
pblnTransBuf = NULL;

```

```
} while(FALSE);  
  
if (pbDataToHashBuf)  
    delete [] pbDataToHashBuf;  
  
if (pbInTransBuf)  
    delete [] pbInTransBuf;
```

End of CallSHA1Sequenz

CONFIDENTIAL

5.4 Error/Return Codes

The base number for the return codes is **TPM_RET_BASE = 01h**. The catalog of error and return codes can be extended to include TPM vendor specific return codes at the end of this list.

If either driver fails to communicate with the TPM device the caller (BIOS) MUST do one of the following:

- Permanently disable the connection to the TPM,
- Boat-anchors the platform,
- Perform a Platform Reset, or
- Force transfer control of the platform to a manufacturer approved environment.

Error Code	Value (Dec)	Description
TPM_OK	00	Indicator of successful execution of the function.
TPM_GENERAL_ERROR	TPM_RET_BASE + 00	A general unidentified error occurred.
TPM_IS_LOCKED	TPM_RET_BASE + 01	The access cannot be granted the device is open.
TPM_NO_RESPONSE	TPM_RET_BASE + 02	No response from the TPM device.
TPM_INVALID_RESPONSE	TPM_RET_BASE + 03	The response from the TPM was invalid.
TPM_INVALID_ACCESS_REQUEST	TPM_RET_BASE + 04	The access parameters for this function are invalid.
TPM_FIRMWARE_ERROR	TPM_RET_BASE + 05	Firmware error during start up.
TPM_INTEGRITY_CHECK_FAILED	TPM_RET_BASE + 06	Integrity checks of TPM parameter failed.
TPM_INVALID_DEVICE_ID	TPM_RET_BASE + 07	The device ID for the TPM is invalid.
TPM_INVALID_VENDOR_ID	TPM_RET_BASE + 08	The vendor ID for the TPM is invalid.
TPM_UNABLE_TO_OPEN	TPM_RET_BASE + 09	Unable to open a connection to the TPM device.
TPM_UNABLE_TO_CLOSE	TPM_RET_BASE + 10	Unable to close a connection to the TPM device.
TPM_RESPONSE_TIMEOUT	TPM_RET_BASE + 11	Time out for TPM response.
TPM_INVALID_COM_REQUEST	TPM_RET_BASE + 12	The parameters for the communication are invalid.
TPM_INVALID_ADR_REQUEST	TPM_RET_BASE + 13	The address parameter for the access is invalid.
TPM_WRITE_BYTE_ERROR	TPM_RET_BASE + 14	Bytes write error on the interface.
TPM_READ_BYTE_ERROR	TPM_RET_BASE + 15	Bytes read error on the interface.
TPM_BLOCK_WRITE_TIMEOUT	TPM_RET_BASE + 16	Blocks write error on the interface.
TPM_CHAR_WRITE_TIMEOUT	TPM_RET_BASE + 17	Bytes write time out on the interface.
TPM_CHAR_READ_TIMEOUT	TPM_RET_BASE + 18	Bytes read time out on the interface.
TPM_BLOCK_READ_TIMEOUT	TPM_RET_BASE + 19	Blocks read error on the interface.
TPM_TRANSFER_ABORT	TPM_RET_BASE + 20	Transfer abort in communication with TPM device.
TPM_INVALID_DRV_FUNCTION	TPM_RET_BASE + 21	Function number (AL-Register) invalid for this driver.
TPM_OUTPUT_BUFFER_TOO_SHORT	TPM_RET_BASE + 22	Output buffer for the TPM response too short.
TPM_FATAL_COM_ERROR	TPM_RET_BASE + 23	Fatal error in TPM communication.
TPM_INVALID_INPUT_PARA	TPM_RET_BASE + 24	Input parameter for the function invalid.
TPM_TCG_COMMAND_ERROR	TPM_RET_BASE + 25	Error during execution of a TCG command.
TPM_VENDOR_BASE_RET	128	Start point for return codes are reserved for use by TPM vendors.
TPM_LDI_ERROR	TPM_VENDOR_BASE_RET + 00	Invalid logical device ID for extended TPM info.
TPM_EXTENDED_INFO_ERROR	TPM_VENDOR_BASE_RET + 01	Wrong response from the TPM device by reading the extended TPM info.
TPM_FIFO_ERROR	TPM_VENDOR_BASE_RET + 02	Tx or Rx FIFO status of the TPM device not OK.
TPM_RESERVED_REG_INVALID	TPM_VENDOR_BASE_RET + 03	TPM specific registers are invalid.

5.5 Basic BIOS integration hints from TCG perspective

5.5.1 Basic BIOS integration phases

1. Phase

Integration of the MP-Driver image into the POST-BIOS build process that the driver functionality is available in POST-BIOS time of the Boot-Phase. Establish the information about the resources used by the TPM device in the resource tables (e.g. ACPI, PnP ...) used by the operating system to recognize the new TPM-device properly and offer this information to the OS-Level driver for the TPM device communication. Hash and extend one part of the BIOS to the specified PCR number (e.g. #2).

2. Phase

Integration of the MA-Driver image into the BIOS-Boot-Block build process, that the driver functionality is available for the BB-Code.

Executions of the MAInitTPM function with the, depending on the power transition, correct value for the StartUp (see also the section "*Power States, Transitions, and TPM initialization*" in the TCG-PC Specific Specification) command to initialize the TPM device. Hash and extend the first partition of the BIOS to the specified PCR number (e.g. #0).

Select the physical presence method HW (see TPM-Data-Book) (e.g. special pushbutton, dipswitch, etc) or SW (e.g. command from BIOS-Setup) depending on the platform vendor design philosophy and or this selection flag with the lifetime flag (see also TCGA-MainSpec).

3. Phase

The TCGA Main Specification includes commands which require "local" or "physical" presence at the platform before the command will operate.

Some of this fundamental TPM commands are used for enabling, activating and control the ownership process of a TPM device. The integration of these commands into the BIOS-Setup is recommended to fulfill this TCG requirement for "physical presence".

Basic-TCG-Command-List for BIOS-Setup:

- TSC_PhysicalPresence (to **Set** and **Reset** physical presence)
- TPM_PhysicalEnable
- TPM_PhysicalDisable
- TPM_PhysicalSetDeactivated (set the state of deactivated flag in the persistent area)
- TPM_SetOwnerInstall (set the state of the ownership flag)
- TPM_ForceClear (performs the clear operation under physical access)

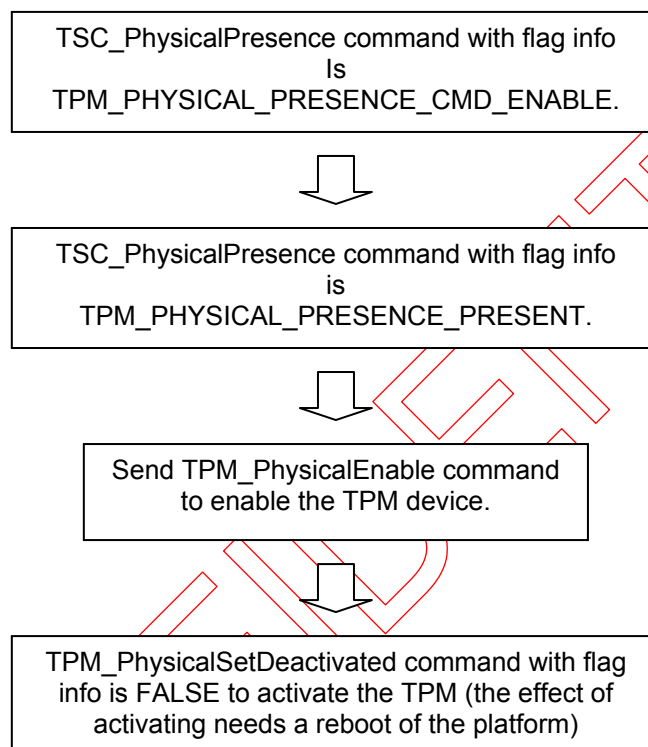
4. Phase

Step by step integration of the platform vendor supported TCG functionality. For this see the TCG PC Specific Implementation Spec. Integrate the interrupt interface for the TCG to offer the TCG utility to other BIOS extensions (e.g. SCSI-OpRoms). Write the PCR-Event-Log-Entries into the ACPI table and so on to developing a TCG-Aware BIOS.

5.5.2 Basic BIOS-Setup commands for TPM from TCG perspective

5.5.2.1 Enable and activate a TPM device from TCG perspective

The following sequence shows the enabling and activating process for a TPM device with the physical presence is realized via software solution (e.g. entering BIOS-Setup and press a special key to go into to a platform vendor individual TPM-Admin-BIOS-Setup-Page, which includes the TPM settings).

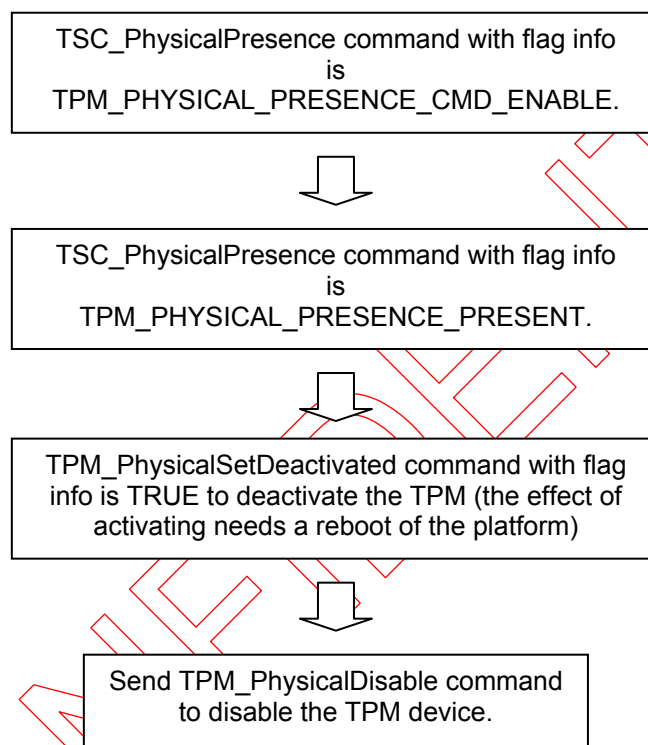


Notes:

- If the platform uses the hardware method to realize the physical presence then the first two steps are obsolete for all operation with physical presence.
- The flags for the TSC_PhysicalPresence command can be combined and then set with one call.

5.5.2.2 Disable and deactivate a TPM device from TCG perspective

The following sequence shows the disabling and deactivating process for a TPM device with the physical presence is realized via software solution (e.g. entering BIOS-Setup and press a special key to go into to a platform vendor individual TPM-Admin-BIOS-Setup-Page, which includes the TPM settings).



Notes:

- If the platform uses the hardware method to realize the physical presence then the first two steps are obsolete for all operation with physical presence.
- The flags for the TSC_PhysicalPresence command can be combined and then set with one call.

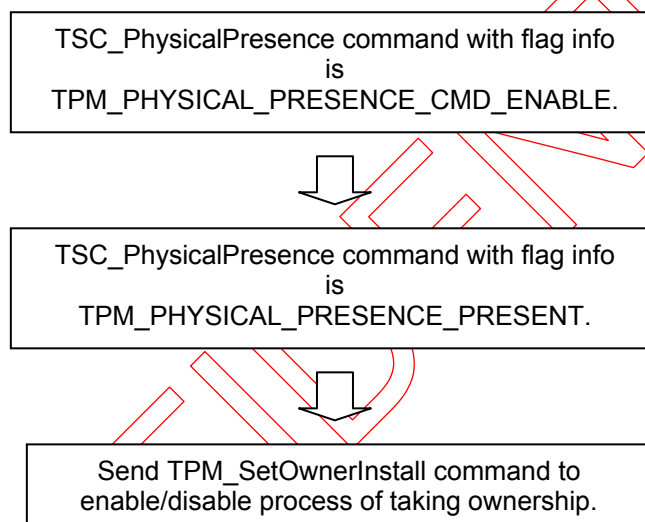
5.5.2.3 Enabling Ownership for TPM device from TCG perspective (optional)

The purpose of this capability is to enable or disable the process of taking ownership of a TPM device and respectively for the specific platform.

The process of enabling and disabling ownership needs physical presence also. The default state is that the process of taking ownership is enabled.

(e.g. entering BIOS-Setup and press a special key to go into to a platform vendor individual TPM-Admin-BIOS-Setup-Page, which includes the TPM settings).

Flow to modify the ownership process:



Notes:

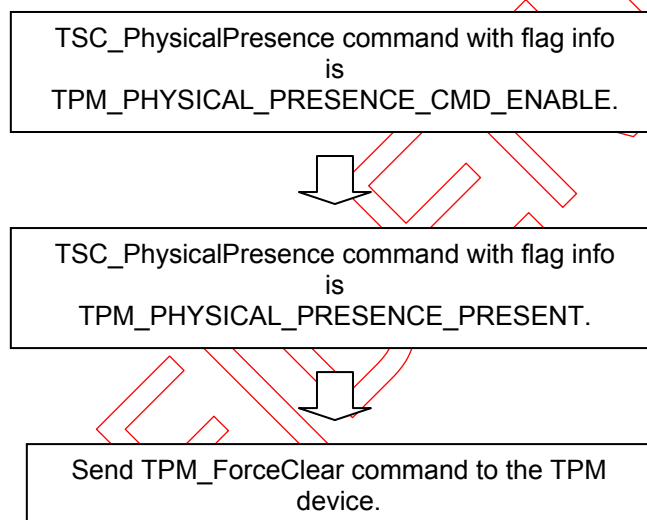
- Keep in mind the special behavior of this command and associated flag if an owner is active for this TPM device.
- If the platform uses the hardware method to realize the physical presence then the first two steps are obsolete for all operation with physical presence.

5.5.2.4 Clear TPM device from TCG perspective (ForceClear command)

The ForceClear command performs the clear operation under physical access. After execution the result of this command is exactly like the TPM_OwnerClear operation.

(e.g. entering BIOS-Setup and press a special key to go into to a platform vendor individual TPM-Admin-BIOS-Setup-Page, which includes the TPM settings).

Flow for TPM ForceClear process:



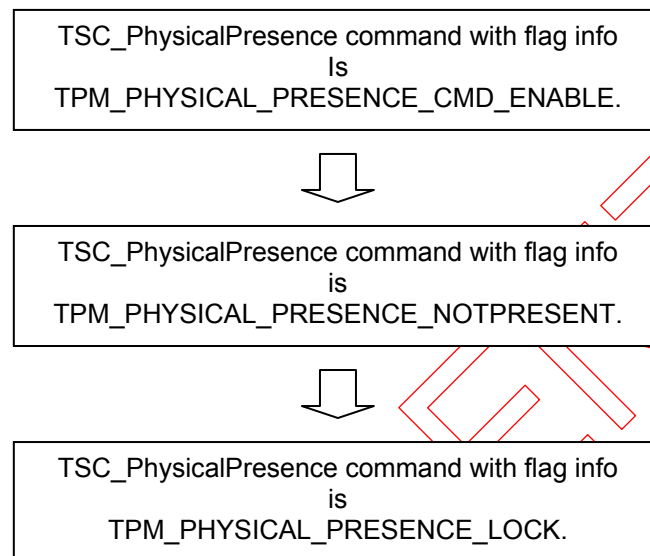
Notes:

- If the platform uses the hardware method to realize the physical presence then the first two steps are obsolete for all operation with physical presence.
- The flags for the TSC_PhysicalPresence command can be combined and then set with one call.

5.5.2.5 TSC PhysicalPresence operation for TPM device from TCG perspective

(Indication: This is normally a hidden command for the Setup-Users)

The physical presence should be disabled if the administration/configuration process of the TPM is finished (e.g. leaving a platform vendor individual TPM-Admin-BIOS-Setup-Page, which includes the TPM settings or by exiting the BIOS-Setup).

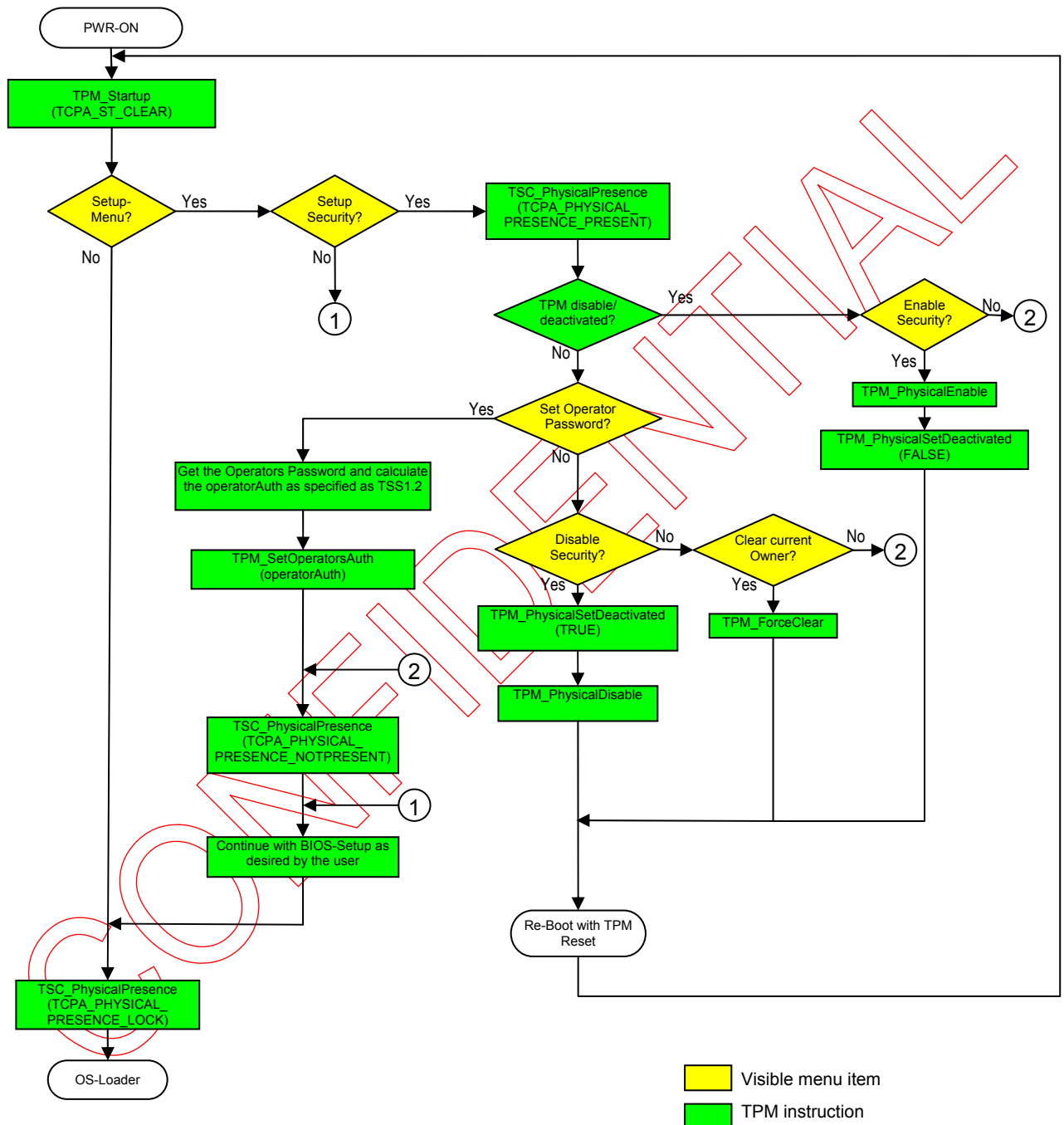


Notes:

- Keep in mind at the end of the BIOS-POST process (before transition to OS loader environment) the BIOS MUST lock the physical presence state also.
- If the platform uses the hardware method to realize the physical presence then the first two steps are obsolete for all operation with physical presence.

5.5.2.6 Typical TPM-Admin BIOS Setup flow

The following sequence shows the typical implementation of a BIOS setup page for enabling, disabling or clearing the owner of a TPM.



Notes:

- It is assumed that the TPM has been set to TPM_PHYSICAL_PRESENCE_CMD_ENABLE during the initialization process of the platform and that this state has been locked with TPM_PHYSICAL_PRESENCE_LIFETIME_LOCK.

5.5.3 Basic BIOS behavior during the Pre-OS-Boot state from TCG perspective

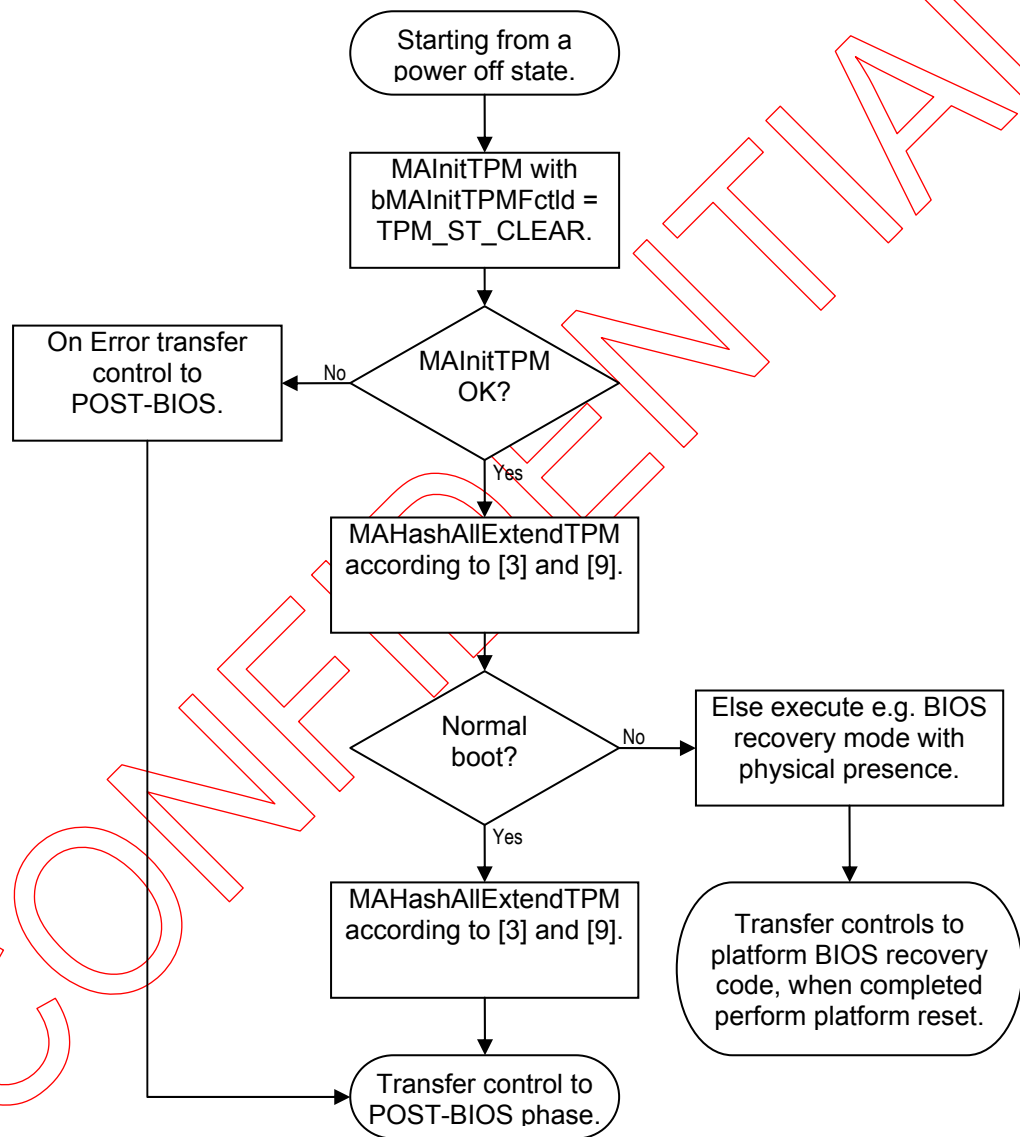
The following pseudo code and flowchart sections are only suggestions of implementation that generalises the control flow of the motherboard initialization during the Pre-Boot state. Not all conditions and error states are included in the main focus of the TCG perspective. This intended only as a raw guide.

CONFIDENTIAL

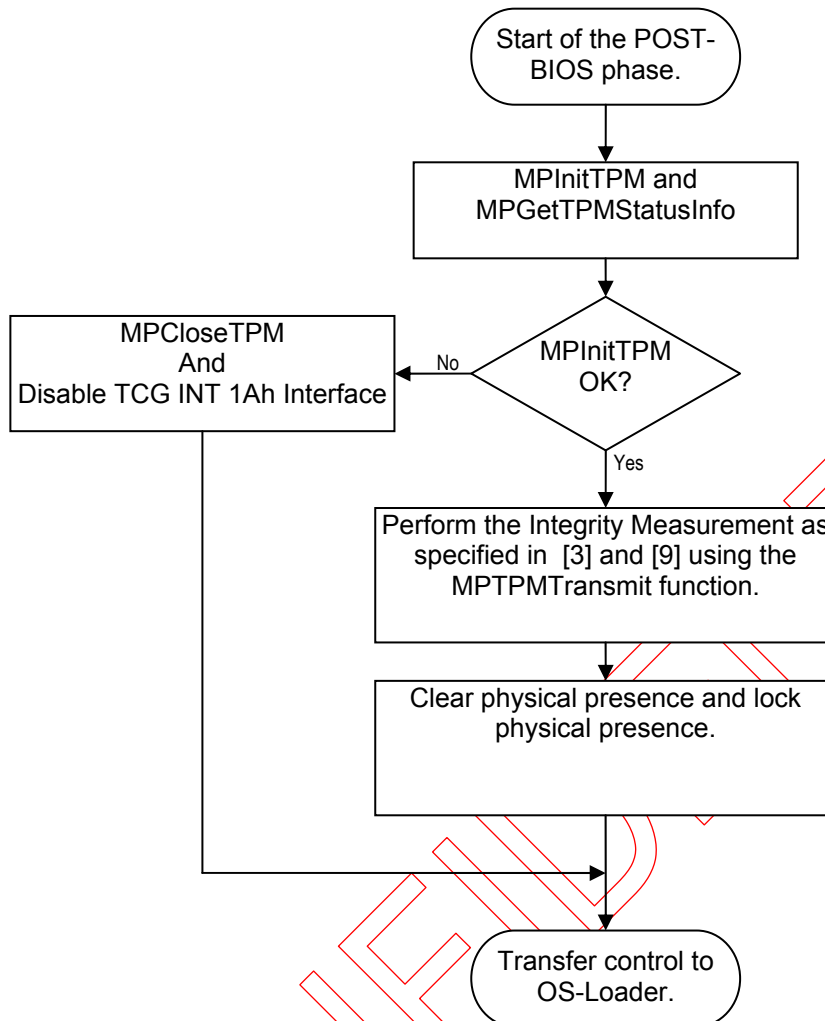
5.5.3.1 S5 ⇌ S0

This is the standard transition from the “Power-Off” state to a “Power-On” state. Platform reset is asserted and the full BIOS initialization sequence is executed.

BOOT-BLOCK-BIOS



POST-BIOS

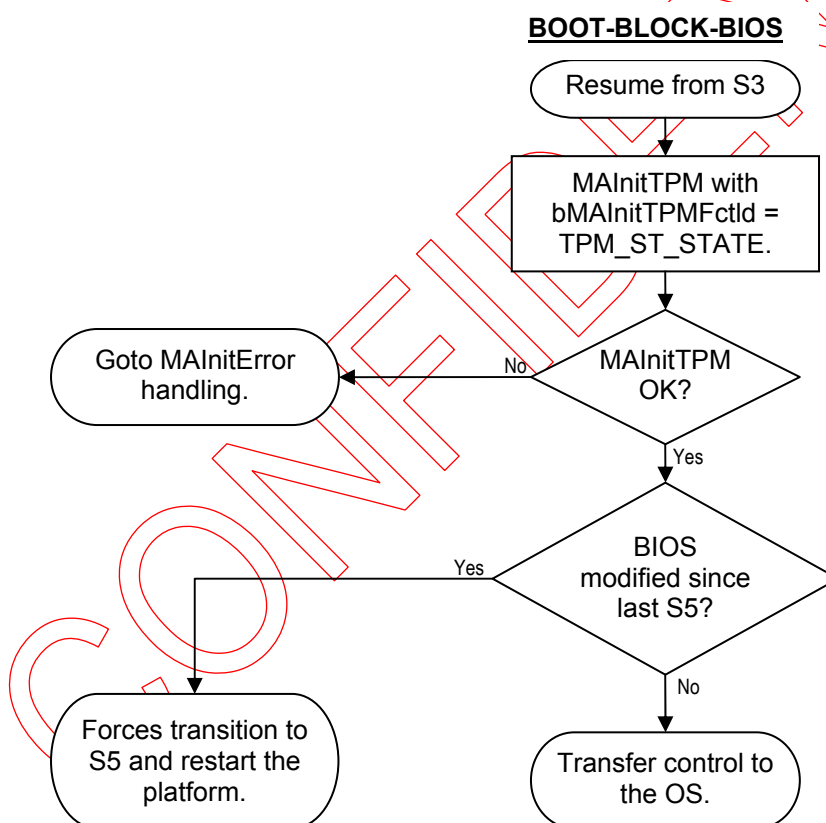


5.5.3.2 S4 ⇨ S0

This is the situation where the IPL instead the OS-Loader will load the memory from a hard disk. Platform reset is asserted. The full BIOS initialization sequence is executed just like S5 ⇨ S0. If there are any changes to the platform's components or configuration, measuring these changes is the responsibility of both the BIOS-Boot-Block and the POST-BIOS. Same flow as S5 ⇨ S0 except IPL loads the saved memory image.

5.5.3.3 S3 ⇨ S0

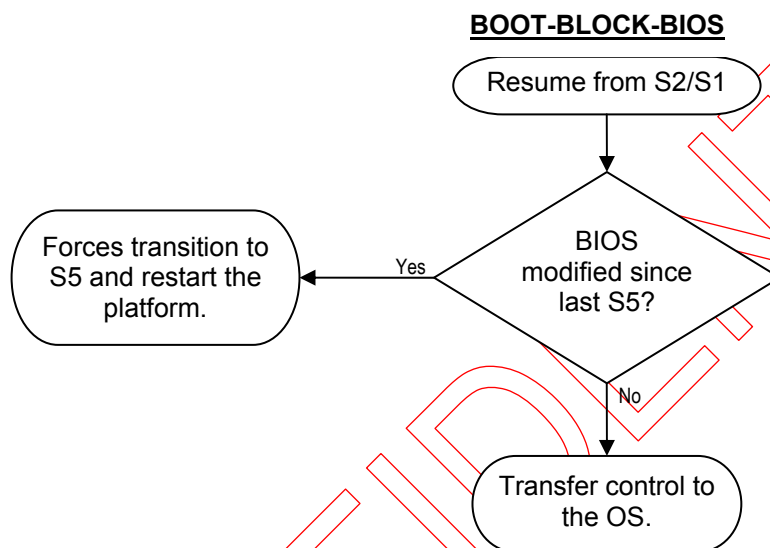
S3 is the most complex mode to handle and the behavior depends strongly on the platform design. The major aspect is whether the TPM device is powered or not during S3. The Post-Boot driver (OS-Level) must issue the TPM_SaveState command that the TPM can save the PCR values. The command to restore the PCR's is issued by the CRTM. Resume from S3 state a platform reset is asserted. The CRTM executes code perform resume without re-measuring the components. CRTM passes control directly the OS. If there are any changes to the platform, measuring these changes is the responsibility of the OS.



Alternatively there is a possibility in conjunction with the OS device driver for Windows2000 and WindowsXP, to perform only the BIOS modification check, as the Startup_State handling is done by this driver.

5.5.3.4 S2 ⇔ S0 and S1 ⇔ S0

Resume from S2/S1 suspend state. Platform reset has never been asserted so the MAInitTPM function cannot be called. CRTM executes the code to perform resume without re-measuring Pre-OS-Boot components the control is directly transferred to the OS. If there are any changes to the Platform's components or configuration, measuring these changes is the responsibility of the OS.



5.6 Notes

5.6.1 Hash-Performance for IFX-TPM device:

Assumptions:

Hash-Block-Size returned by TPM_SHA1Start: SLD 9630 TT 1.1: 960 Byte
SLB 9635 TT 1.2: 1216 Byte

Size of Input data block: 1024 Byte (1kByte)

Sequence:

TPM_SHA1Start ⇒ TPM_SHA1Update ⇒ TPM_SHA1CompleteExtend

SLD 9630 TT 1.1:
≈1ms ≈11ms ≈2ms Total: ≈ 14 ms

SLB 9635 TT 1.2:
≈ 2 ms ≈ 5 ms ≈ 2.5 ms Total: ≈ 10 ms

To hash larger data blocks the caller must implement and execute a loop with the TPM_SHA1Update operation (see also sample in 5.3.1.4).

5.6.2 Endianess of structures and data types

Each TCG structure and data type must use big endian bit and/or byte ordering. All structures must be packed on a byte boundary. The "Byte" is the unit of length when the length of a parameter is specified.

5.6.3 Size Estimation for MA- and MP-Driver image files

The size estimation is based on the currently known and defined functions (TCG-PC-Client-Specific Implementation Specification) are:

MA-Driver: Object file including Code and Const-Data ≈6Kbyte
(No Stack and memory; but uses MMX-Functionality and Register)

MP-Driver: Object file including Code and Data ≈4Kbyte
(Plus approximately 128 Byte Stack)

6 Appendix

6.1 Deployment package description of the IFX-TPM-BIOS-Drivers (32Bit-Versions):

- **MA-Driver:**

- Readme.txt	Short description file for the driver deployment.
Seg32Drv:	
- TpmMaDrv.rom	TPM-Memory-Absent-Driver-Image-File 32Bit segment version
BigRealMode:	
- TpmMaDrv.rom	TPM-Memory-Absent-Driver-Image-File BigReal Mode 32Bit segment version
Seg16Drv:	
- TpmMaDrv.rom	TPM-Memory-Absent-Driver-Image-File 16Bit segment version
BigRealMode:	
- TpmMaDrv.rom	TPM-Memory-Absent-Driver-Image-File BigReal Mode 16Bit segment version

- **IncDrv:**

- DrvInOut.inc	Global interface constants/structures for the TPMMDrv and TPMMPDrv driver library.
- RetCodes.inc	Return and error codes of the TPMMDrv and TPMMPDrv driver library.

- **MP-Driver:**

- Readme.txt	Short description file for the driver deployment.
Seg32Drv:	
- TpmMpDrv.rom	TPM-Memory-Present-Driver-Image-File 32Bit segment version
BigRealMode:	
- TpmMpDrv.rom	TPM-Memory-Present-Driver-Image -File BigReal Mode 32Bit segment version
Seg16Drv:	
- TpmMpDrv.rom	TPM-Memory-Present-Driver-Image-File 16Bit segment version
BigRealMode:	
- TpmMpDrv.rom	TPM-Memory-Present-Driver-Image-File BigReal Mode 16Bit segment version

6.2 16Bit-Driver Remarks/Assumptions

- The pointer in the TPMTransmitEntryStructure (e.g. pbInBuf, pbOutBuf) must hold the Segment value in the high word and the Offset in the low word of the parameter (e.g. pbInBuf segment is 100h and offset is 500h -> parameter value: 0100h:0500h).
- The 16-Bit-Drivers assume, that the TPMTransmitEntryStructure is located in the same segment as the driver image.
- At the call the ESI register should then only carry the offset value for the parameter structure (e.g. 5000h). The usage of the ESI register is based on the current version of TCG-PC-Client-Specific-Specification but for the 16 bit version it is not fully used.

6.3 Trademarks

- AMD, the AMD logo, AMD Athlon, K6, 3DNow!, and combinations thereof, and K86 are trademarks, and AMD-K6 is a registered trademark of Advanced Micro Devices, Inc.
- Microsoft and Windows are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.
- MMX is a trademark and Pentium (e. g. PIII) is a registered trademark of Intel Corporation.
- Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

7 Open Topics

This section is empty, but will be used in the further as new definition issues are raised.

CONFIDENTIAL