DOCUMENT CHANGE HISTORY

DG-06034-002_v03.7

Version	Date	Authors	Description of Change
01.00	November 1, 2011	MY, NG, VS	Initial release
02.00	November 23, 2011	NG	Introduce Slave-Command Support, CPU Performance Control, GPU System Control, and System Performance Control
02.01	January 11, 2012	NG	Introduce a Get Status subcommand to GPU Performance Control for retrieving dynamic GPU performance state (such as the current vPstate).
02.02	February 24, 2012	NG	Make a correction to the *Master Command-Submission Protocol* diagram.
02.03	March 12, 2012	DM	Made corrections to Table A.1 on page 56.
02.04	May 23, 2012	CC	Added Arg1 (GPU information) values to Opcode 05h. (See Section “05h - Get GPU Information” on page 23) Section “Using the Command Register ” on page 8 - Added Copy-bit description.
02.05	July 31, 2012	CC	Added note on direct polling to section “01h - Get Capabilities” on page 16.
02.05a	August 22, 2012	CC	Changed the security policy only. No content change.
02.06	December 6, 2012	CC	Added Opcode 0Ch: ECC statistics data, format version 3, to support geometry changes in Kepler GPUs. See “0Ch - Query ECC Statistics - Format v3” on page 28 for more information.
02.07	March 28, 2013	MY/CC	Added Opcodes 0Dh-11h: Scratch memory and asynchronous request commands.
02.08	May 8, 2013	MY/CC	Added Opcode 12h: Check external power. Removed asynchronous request 0x03 (Read the status of auxiliary power connector).
02.09	August 6, 2013	MY/CC	Added Opcode 13h: Read Dynamic Page Retirement Statistics
02.10	November 14, 2013	SH/CC	Updated product list in Appendix A.

Version	Date	Authors	Description of Change
02.11	April 30, 2014	SH/CC	Added Opcodes F0h—F8h.
02.12	October 13, 2015	MY/CC	Updated Table 3.7 on page 22 (07h size). **Note**: This table is now provided as an attachment. Added Table 3.1, “Status Return Values,” on page 13. Edited Table A.1 and Table A.2.
2.13	March 21, 2016	JK/CC	Added Example Code attachment and explanation.
3.0	June 3, 2016	RA/CC	Added Table 3.7, “Capability Support for Select Tesla and Quadro Products,” on page 18. **Note**: This table is now provided as an attachment.
3.1	June 23, 2016	RA/CC	Doc reorganization. Added Tesla M10, K20, K40 to Table Table 3.7 on page 18. **Note**: This table is now provided as an attachment. Added opcode 15h: Read thermal parameters.
3.2	August 10, 2016	MY/JK/CC	Added opcode 14h: Query ECC Data - Format V4. Added opcode F9h: Assert Power Brake. Added M6 capabilities to Table 3.7 on page 18. **Note**: This table is now provided as an attachment. Updated the SDK readme file and included a pre-complied 64-bit binary.
3.3	December 1, 2016	RA/CC	Opcode 10h: Table 3.9 on page 36: Added Asynchronous requests (Arg1 = 0x06, 0x07) to read the clock limit and set the clock limit, respectively Updated *Capability Support for Tesla and Quadro products* to include Pascal GPUs, and converted to an attachment.
3.4	March 1, 2017	RA/CC	Opcode 10h: Table 3.9 on page 36 arg1 0x07: Added text stating that the clock limit set is persistent when reloading the driver or restarting the system.

Version	Date	Authors	Description of Change
3.5	March 7, 2017	MY/CC	Added Table 3.7, "GPU Capability DWord(4)," on page 19. Added Arg1 (0x09) to Table 3.9, "Asynchronous Requests," on page 36.
3.6	October 11, 2017	RA/CC	Removed "GPU request functionality" feature from Table 3.5 on page 18. Added GPU and memory maximum operating temperature to opcode 0x15 section. Added that NVIDIA driver must be loaded in persistence mode to make the opcode 0x15 query. Added maximum SMBPBI processing time of 100 ms on NVIDIA GPUs. Added column in Table 3.2 on page 14 toindicate which opcodes require the driver to be loaded. Added opcode 0x16: Memory ECC statistics for Volta.
3.7	October 18, 2017	RA/CC	Added opcode 0x16 to get capabilities opcode 0x01. Added opcode 0x16 to Table 3.2 on page 14.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

This document defines an Application Programming Interface between a microcontroller, typically the System Embedded Controller (EC), and the NVIDIA GPU. This interface is provided to give the EC access to GPU thermal, power, and performance data and perform a number of control actions. With additional (optional) hardware interfacing, it is also possible for the GPU to use this interface to request information from, and trigger action on, the SMBPBI Master.

This document contains the following chapters:

- "Using the SMBus Post-Box Interface" on page 2

  Explains how to use the SMBPBI.
- "Interface Commands" on page 12

  Provides a list of the command opcodes and their description.
- "SMBus Interface Additional Details" on page 53

  Provides additional technical details about the SMBPBI.
- "Implementation of Specific Parameters" on page 56

  Lists product-specific features and register information.

CHAPTER 2 USING THE SMBUS POST-BOX INTERFACE

2.1 FILE ATTACHMENTS

2.1.1 Provided Attachments

This PDF contains the following attachments:

- OOB_Example_Code.nvzip

  This file contains example code for the purposes of demonstrating the SMPBI protocol. It is not provided as an SDK, but can be used in customer implementation or as a test suite for validation, and is provided without Warranty.

- oobtest.nvzip

  This file contains a binary file for testing the source code. The binary can be run for test purposes without any compilation required.

- NVIDIA_products_SMBPBI_capabilties.xlsx

  This document shows the SMBPBI capabilities of different NVIDIA Tesla and Quadro products.

2.1.2 Accessing the Attachments

To access the attached files, click the **Attachments** tab from the left-hand toolbar of this PDF, then select the file and click the **Save** option to retrieve it.

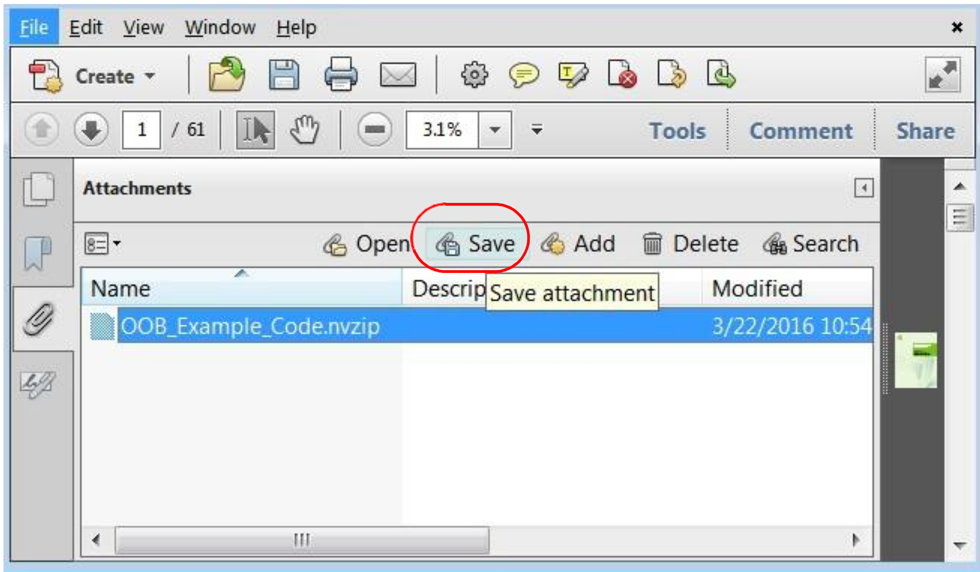

Figure 2.1 Accessing the Attachments

> **Note:** The zip file attachment has been renamed with a .nvzip extension so that it can be embedded in this document. You must save the attached .nvzip file and rename it to .zip before opening/extracting the file. Do not attempt to open the file from the PDF attachments tab directly.

2.2 SYSTEM INTERCONNECT

The SMBus Post-Box is exposed via the SMBus interface that is typically (but not always) connected to the system Embedded Controller (EC). It may be connected through the SMCLK and SMDAT lines of the standard PCI Express edge connector. This is the minimum interfacing required to utilize the GPU's SMBus Post-Box interface.

Regardless of exact configuration, the GPU will always serve as a slave-device on the SMBus. In the configuration below, the EC serves as SMBus master. From here on in this document, the agent responsible for driving the bus will simply be referred to as the *SMBPBI Master* or even the *Master* device.

For enhanced capabilities, the NVIDIA SMBPBI interface supports an optional communication mechanism which allows the GPU to make requests to the SMBPBI master. These requests are known as Slave Commands and require additional hardware interfacing between the GPU and the master control logic. Due to the dependency on system software and hardware, the ability to support slave commands is considered a system-dependent capability. As a result, the GPU driver and GPU itself must be made aware of the capability when it is available[1] . Refer to Section 2.4 for more information on system configuration.

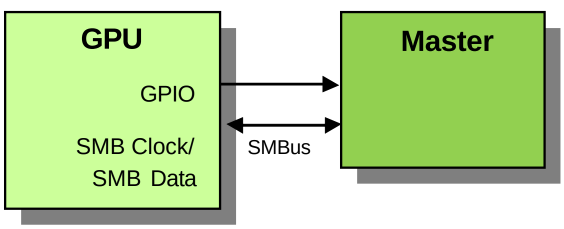

Figure 2.2 SMBus Post-Box Interface

2.3 INITIALIZING THE SMBPBI

At boot, the GPU Driver is responsible for initializing the GPU's SMBus Post-Box Interface. Once initialized, this interface will be supported. On some systems, it is also possible for the interface to be initialized earlier (pre-driver load) by the VBIOS. See Section 2.5.4 for additional information.

1. Support for slave commands is also GPU-dependent. Refer to Appendix A for a listing of the GPUs supporting the feature.

2.4 SYSTEM CONFIGURATION

The following sections describe system configuration steps needed to both announce and enable system-dependent capabilities at system-design time.[2]

2.4.1 Enabling Slave Commands

As mentioned previously, slave commands constitute an optional type of communication whereby the GPU (SMBPBI Slave) (SMBPBI) can send commands/requests to the SMBPBI Master. This capability depends on both the presence of a GPU GPIO and the GPU itself. When available, a GPU GPIO is used as a notification mechanism to the master that the slave has a pending request. It is recommended that this GPIO be wired into the master's interrupt logic to avoid polling. The GPU and GPU Driver must be made aware of this GPIO through GPIO reservation in either the GPIO Video BIOS (VBIOS) or the MXM-SIS. To declare the GPIO in VBIOS, add an entry to the GPIO Assignment Table (in the DCB) with GPIO-function 80h (*SMBPBI_SLAVE_COMMAND*). Similarly, for MXM-SIS, add a GPIO Pin Entry Structure to the *MXM GPIO Device Structure* with Function 80h. Refer to Section 5.6 in the *MXM Graphics Module Software Specification Version 3.0* for details on MXM configuration.

Absence of this assignment indicates that the system is not capable of supporting slave commands (or that the facility is undesired/not-required).

2.4.2 Slave Addressing

2.4.2.1 Default Slave Address

The client acts as the SMBus master and the GPU acts as the SMBus slave device. The GPU slave SMBus address is 4Fh (b' x) by default.

2.4.2.2 Slave Addressing in Dual-GPU Systems

The system can often be configured to respond to address 4Eh (b' x) using board straps (*SMB_ALT_ADDR*). This is the preferred mechanism to use when resolving address conflicts in dual-GPU systems.

2. Most system-dependent capabilites are announced and enabled at runtime through exposure of master and slave capability bits.

2.4.2.3 Slave Addressing in Systems with 3 or More GPUs

In configurations where more than two GPU slave devices are present, NVIDIA recommends that OEMs include a multiplexer component in their designs (such as, for example, the Pericom PI3B3253 Dual 4:1 Multiplexer). Such as component would allow the single SMBus master to select between multiple GPU SMBus slaves that have the same SMBus address.

2.4.2.4 Resolving Address Conflicts Dynamically

In addition to the mechanisms described above, address conflicts may also be resolved dynamically in some GPUs using the SMBus Address Resolution Protocol (ARP). Refer to Appendix A for list of GPUs that support the ARP protocol and Section 5.6 of the *System Management Bus (SMBus) Specification Version 2.0* for details on the protocol itself.

2.4.2.5 Maximum SMBPBI Processing Time

The maximum SMBPBI request processing time on NVIDIA GPUs is 100 ms. This implies the master can poll the SMBPBI request on the GPU up to 10 times per second.

2.5 COMMUNICATING OVER THE SMBUS

2.5.1 Post-Box Registers

The SMBPBI Master and GPU communicate over the SMBus using a shared set of GPU SMBus registers (known as the Post-Box Registers). Minimally, two 32-bit registers in the GPU SMBus register space are involved in this interface for all Post-Box operations. These include the:

- **Command Register** - Register used by the SMBPBI Master to submit requests to the GPU and for receiving completion status.
- **Data Register** - Register that contains the requested data upon the completion of a request. It may also carry additional information for the GPU at the request submission time. The encoding of this register is opcode-specific. Refer to Section 3.1 for more information.

Both of these registers (Command and Data) are owned by the Master. The GPU may never write them outside of:

- Device initialization (before the interface is made available).
- When responding to a command received from the Master.

On systems and GPUs supporting slave commands, the use of a third Post-Box register called Data-Out is required.

- **Data-Out Register** - Register used by the GPU to submit commands to the SMBPBI Master. Unlike The Command-In register, this register is not generally used for receiving completion status. The master should never write to this register unless responding to an explicit request to do so from the GPU.

> **Note:** The SMBus command-codes needed to access these registers in the GPU SMBPBI register space can be found in Appendix A.

The layout of these registers is described in the following sections.

2.5.2 Using the Command Register

GPU Commands are written to the *Command Register* in the following format:

Table 2.1 GPU Command Format

Bit	Access	Default	Description
31:31	R/W	0	Command Execute Synchronization/execution bit. By writing "1" into this field the SMBPBI Master submits the request for execution. Having received the request, the GPU clears this bit. This is the indication for the master that the request has been accepted. Upon the completion of the request processing the GPU will fill the status field with a status code that is guaranteed to be non-zero. This is the indication for the master that it may collect the results of the request.
30:30		0	Copy If this bit is set, then upon a successful execution of this request, bits [23..0] of the Data Register (see Section 2.5.3) will be copied into bits [23..0] of the Command and Status Register.
29:29	R/W	0	Reserved Must be written with zero by the master
28:24	R/W	0	Status The value in this field characterizes the result of the request execution by the GPU. When submitting a request, the master must clear this field. Refer to Section 3.1.1 for a listing of possible values.
23:16	R/W	0	Arg2 Optional opcode-specific argument to pass to the GPU along with the command.
15:8	R/W	0	Arg1 Optional opcode-specific argument to pass to the GPU along with the command.
7:0	R/W	0	Opcode The request operation code.

Note: Before submitting the first request to the GPU, the master must check the Status field of this register and should not issue a request if the value is INACTIVE or NULL.

2.5.3 Using the Data Registers

The *Data Register* contains the requested data upon the completion of a request. It may also carry additional information for the GPU at the request submission time. The encoding of this register is opcode-specific. Refer to “Interface Commands” on page 12 for more information.

Table 2.2 Data Register Format

Bit	Access	Default	Description
31:0	R/W	0	Data 32-bit input/output data.

Subsequent sections in this document refer to this register by the names *Data-In* and *Data-Out.* The appended modifier is to designate the context in which the register is being used. When used to pass additional command information to the GPU at request submission time, the data stored in this register is known as *Data-In*. Likewise, when holding the output (or result) of a request, the data in the register is known as *Data-Out*.

Since this API interface uses a single data register, NVIDIA recommends that the SMBPBI Master write data to Data Register (Data-In) only when the command has specified an opcode that requires additional input data. Likewise, the master should avoid reading the Data Register (Data-Out) upon completion of a command that does not store a result in the Data Register.

2.5.4 READY Status Code & Implementation Phases

2.5.4.1 About the Implementation Phases

There are two distinct pieces of software that implement the protocol described in this document. One is operating in the environment where the GPU driver has not been loaded, or is inactive. This software is included in the NVIDIA board's BIOS (or integrated into the System BIOS). The other functions in cooperation with the GPU driver and can be viewed as an integral part of the driver (or incorporated in the GPU driver).

These two phases of the protocol implementation may be somewhat different in the functionality they implement. This difference is reflected in capability dwords (see Section 3.3). It is important for the SMBPBI Master implementation to be aware of the capabilities that are currently available. Therefore, the protocol provides a method to notify the master when the Implementation Phase changes (and thus the capabilities associated with that phase).

2.5.4.2 Phase Change Notification through the Status Field

During a transitional period, when the software comes through an initialization sequence, an INACTIVE status will be posted in the Status field. No requests should be submitted at this time. Once the initialization is complete, the status will change to READY and requests can be submitted. However, having received the very first request after the phase change, the software will not execute this request, but instead will post the READY status once again. This is done so that the master will notice the change in the implementation phase.

At this point the master must discard all the internal state that may have been previously cached from the interface (and that can be linked to the implementation phase that has just terminated). This especially concerns the capability DWORDs that may have been cached. The master should then proceed to query the currently available capabilities. Having done this, it may then resume its operation by resubmitting the query that has previously completed with the READY status.

The mechanism described above precludes the possibility that the implementation phase change (and the associated capabilities change) will occur unnoticed by the SMBPBI Master.

> **Note:** Due to potential changes in the Implementation Phase, a status value of READY may be returned for any GPU command.

2.5.5 Master Command Submission Protocol

The diagram below shows the process for issuing commands.

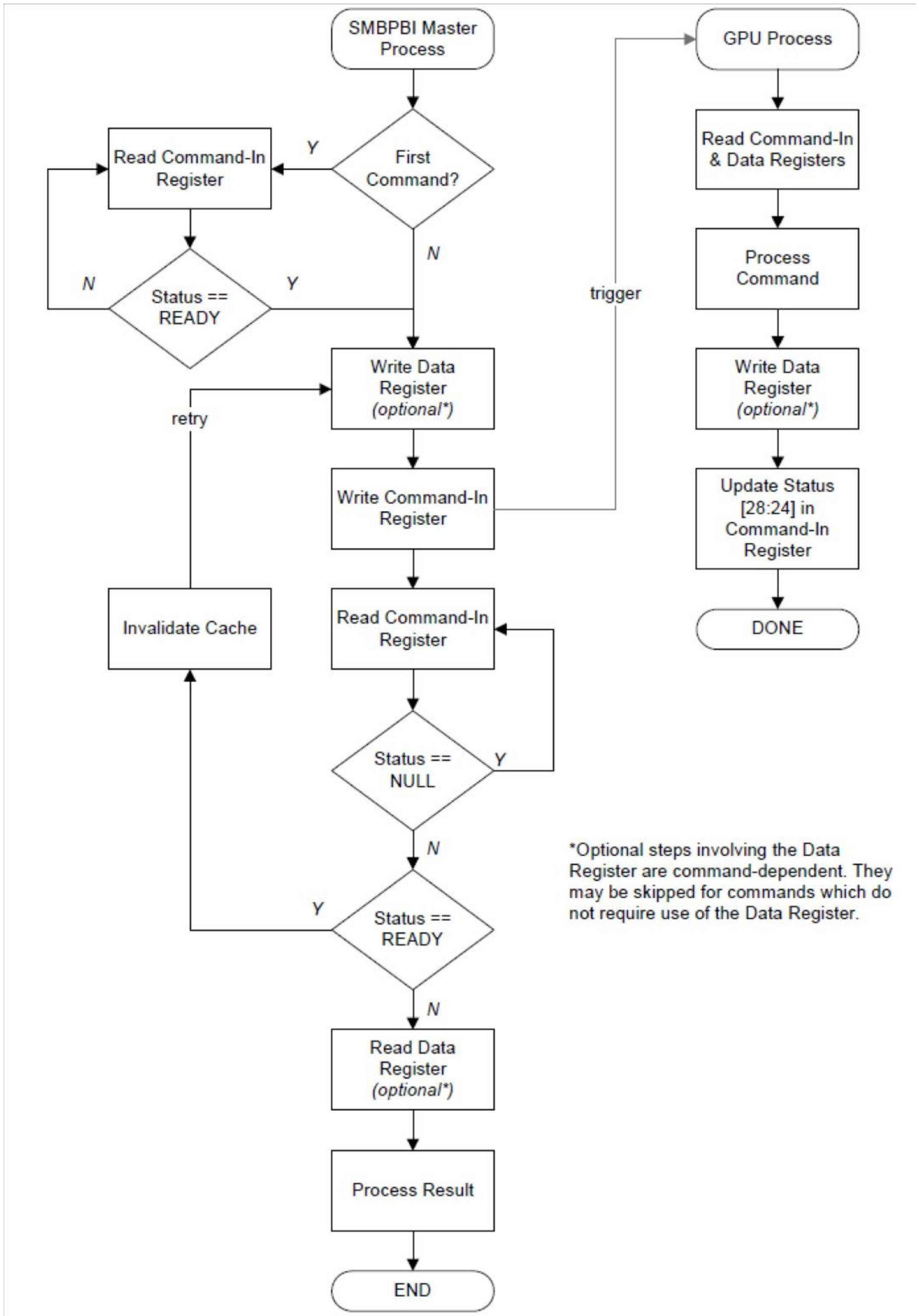

Figure 2.3 Master Command-Submission Protocol