NVIDIA SMBus Post-Box Interface (SMBPBI) for GPUs

Software Design Guide

DG-06034-002_v05.1 | July 2021

Document History

DG-06034-002_v05.1

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 110.
02.04	May 23, 2012	CC	Added Arg1 (GPU information) values to Opcode 05h. (See Section "05h - Get GPU Information" on page 28) Section "Using the Command and Status Register " on page 7 - Added Copy-bit description.
02.05	July 31, 2012	CC	Added note on direct polling to section "01h - Get Capabilities" on page 18.
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 34 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 14. 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 42: 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 42 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 22. Added Arg1 (0x09) to Table 3-9, "Asynchronous Requests," on page 42.
3.6	October 11, 2017	RA/CC	Removed "GPU request functionality" feature from Table 3-5 on page 21. 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 16 to indicate 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 16.
3.8	March 1, 2018	RA/CC	Added minimum cycle time to the Set GPU Power Limit, opcode 0x10
3.9	April 5, 2018	RC/CC	Added Opcode 17h: Get/Set Write-Protect Mode. Added Get Capabilities DWORD(1) Bit 22 (Get/Set Write-Protect mode is supported). Added Tesla Pascal family to Table A-1 and Table A-2 .
3.10	July 31, 2018	RC/CC	Added a note for Opcode 02h regarding retrieving memory temperature without a driver loaded. Updated Table 3-2 for Opcodes 17h & 18h Added Opcode "18h - Query GPU State Flags" on page 61 Updated NVIDIA product capabilities attachment.

Version	Date	Authors	Description of Change
3.11	December 20, 2018	RC/CC	Added SMBus Direct details Updated Table 3.1 with ERR_SENSOR_DATA (0x0C) return status code Updated GPU capability DWORD(0), DWORD(4) Updated Opcode 10H with Get Current SM and Memory Utilization (Arg = 0x0A) Updated Data-Out for Opcode 15H Added Opcode "19h - Get Accumulated GPU Context/SM Utilization Time" on page 63 Added Opcode "1Ah - Query NVLink Information" on page 64 Added Chapter 4 back into the document. Updated Tables A.1 and A.2. Updated NVIDIA product capabilities and oobtest attachment.
3.12	July 31, 2019	RC/CC	Updated capability DWORD(1) & DWORD(3). Updated Arg1 details for Opcode 05h - Get GPU Information. Added Opcode 1Bh - Query Clock Frequency Info. Update Opcode details for F4H, F5H, F6H, and F9H. Added OpcodeFAh - Get/Set MCU FW Write-Protect. Updated NVIDIA product capabilities attachment. Updated Table A-1 and Table A-2 to add NVIDIA T4 and Quadro RTX family.

Version	Date	Authors	Description of Change
4.0	January 27, 2020	RC/CC	Added new SMBPBI features for Volta-Next GPU generation Updated Section 2.5.2: Using the Command and Status Register details Updated Section 2.5.3: Using the Data Registers with details regarding the Extended Data Register Added new ERR_DISPOSITION and PARTIAL_FAILURE Status Return Values to Table 3-1, "Status Return Values Updated GPU Capability DWORD(2) & DWORD(4) Added new product dimension, PCIe capability, and TGP limit Arg1 encodings for 05h - Get GPU Information Updated set total GPU power limit asynchronous request API details (Op 0x10, Arg 1 = 0x01) Added new set/clear clock limit asynchronous request API details (Op 0x10, Arg1 = 0x0B) Updated opcode 11h with Arg2 values for events pending register and event mask register. Updated 16h - Query ECC Statistics - Format V5 query output details and provided a usage example Added memory clock Arg2 encoding to 1Bh - Query Clock Frequency Info Added new opcode 1Ch - Kick off a Request Bundle with usage example Added Section 4.4.3 "Events Pending Register". Added Section 4.4.4 "Event Mask Register". Updated NVIDIA product capabilities attachment. Updated Table A-1 and Table A-2 to add Tesla Volta-Next (SXM4).

Version	Date	Authors	Description of Change
4.1	March 19, 2020	RC/CC	Updated document title to "NVIDIA SMBus Post-Box Interface (SMBPBI) for GPU" Updated oobtest.nvzip and OOB_Example_Code.nvzip attachments Added Result Size Encoding Updated GPU Capability DWORD(1) DWORD(2), & DWORD(4) Added Set/Reset GPU mode asynchronous request 0xC Added Get Clock Limits superseding in-band configurations asynchronous request 0xD Added asynchronous request status code ASYNC_REQ_STATUS_REQUEST_DEFERRED Updated Opcode 18H with MIG and drain/ reset state flags Updated 1AH - QUERY NVLINK INFORMATION with new queries for NVLink status (Format V2), NVLink bandwidth, NVLink sublink width, and NVLink data throughput Added new Opcode 1DH - Request the Oldest Driver Event Message (DEM) from the Server Buffer Added new Opcode 1Eh - Request ECC Statistics (Format V6) Added new Opcode 20h - Request Row-remapping Statistics Added new Opcode 21h - Query PCIe Link Status and Error Counts Updated NVIDIA product capabilities attachment.

Version	Date	Authors	Description of Change
4.2	October 12, 2020	RC/CC	Added new capability bits 27:19 for DWORD[2] & capability bit 11 to DWORD[3] Updated note in 02H - GET TEMPERATURE (SINGLE-PRECISION) to specifically call out V100. Added note for policyMask on Op 0x10, Arg1 0x09 Added NVLink availability (Arg1 0x0F) for Opcode 1AH - QUERY NVLINK INFORMATION Added performance state query (Arg1 0x03) for Opcode 1BH - QUERY CLOCK FREQUENCY INFO Added bank remapping availability histogram (Arg1 0x02) & row-remapping is pending state flag (Arg1 0x01, Arg2 0x00, Bit 1:1) for Opcode 20H - REQUEST ROW-REMAPPING STATISTICS Added target link speed (Arg 1 0x03) for Opcode 21H - QUERY PCIE LINK STATUS AND ERROR COUNTS Added new Opcode 22H - QUERY ENERGY COUNTER Added new Opcode FBH - ACCESS MCU SCRATCH REGISTERS Added new MIG toggle & clock limit set success event bits to Events Pending Register and Event Mask Register Updated Tables A.1 and A.2 Updated NVIDIA product capabilities and oobtest attachments.

Version	Date	Authors	Description of Change
4.3	December 2, 2020	RC/CC	Added SMBPBI support for GeForce Ampere architecture products Added note regarding persistence mode in section 3.1.2 Added note regarding Events Pending bit support with Opcodes F0h - FBh Added new capability bit in DWORD[4] Bit 11 for Get/Set fan curve points asynchronous request Corrected fanProperties Bit[0] in asynchronous request Opcode 10H, Arg1 04h) Added new asynchronous request (Arg1 0Fh) Get/Set fan curve points for Opcode 10H - SUBMIT/POLL ASYNCHRONOUS REQUEST Added NVIDIA RTX A6000 and GeForce RTX Ampere architecture products to Table A.1, Table A.2, and the NVIDIA product capabilities attachment.
5.0	May 25, 2021	RC/CC	Updated Opcode 12h - Check External Power & F8h - Get Board Power Supply Status descriptions. Added NVIDIA A30, NVIDIA A10, and NVIDIA A100-Next SXM for Delta-Next and Redstone-Next to Table A-1 , Table A-2 , and the NVIDIA product capabilities attachment.
5.1	July 6, 2021	RC/CC	Added “Future SMBPBI API Previews” on page 97 No change to document attachment.

Table of Contents

1 Introduction 1

2 Using the SMBus Post-Box Interface 2

2.1 File Attachments 2

2.1.1 Provided Attachments 2

2.1.2 Accessing the Attachments 3

2.2 System Interconnect 4

2.3 Initializing the SMBPBI 4

2.4 System Configuration 5

2.4.1 Slave Addressing 5

2.5 Communicating Over the SMBus 6

2.5.1 Post-Box Registers 6

2.5.2 Using the Command and Status Register 7

2.5.3 Using the Data Registers 9

2.5.4 READY Status Code & Implementation Phases 10

2.5.5 Master Command Submission Protocol 10

2.5.6 SMBus Direct 12

3 Interface Commands 13

3.1 Overview 13

3.1.1 Status Return Values 14

3.1.2 Command Listing 16

3.2 00h - No-op (no action) Request 18

3.3 01h - Get Capabilities 18

3.4 02h - Get Temperature (Single-Precision) 24

3.5 03h - Get Temperature (Extended-Precision) 25

3.6 04h - Get Power 27

3.7 05h - Get GPU Information 28

3.8 07h - Query ECC Statistics - Format v2 32

3.9 0Ch - Query ECC Statistics - Format v3 34

3.10 0Dh - Read From Scratch Memory 37

3.11 0Eh - Write Into Scratch Memory 38

3.12 0Fh - Copy Block Into Scratch Memory 39

3.13 10h - Submit/Poll Asynchronous Request 40

3.14 11h - Access Internal State Registers 50

3.15 12h - Check External Power 51

3.16 13h - Read Dynamic Page Retirement Statistics 51

3.17 14h - Query ECC Statistics - Format V4 52

3.18 15h - Read Thermal Parameters 54
3.19 16h - Query ECC Statistics - Format V5 56
3.20 17h - Get/Set Write-Protect Mode 60
3.21 18h - Query GPU State Flags 61
3.22 19h - Get Accumulated GPU Context/SM Utilization Time 63
3.23 1Ah - Query NVLink Information 64
3.24 1Bh - Query Clock Frequency Info 68
3.25 1Ch - Kick off a Request Bundle 70
3.26 1DH - Request the Oldest Driver Event Message (DEM) from the Server Buffer 74
3.27 1Eh - Request ECC Statistics (Format V6) 75
3.28 20h - Request Row-remapping Statistics 76
3.29 21h - Query PCIe Link Status and Error Counts 79
3.30 22h - Query Energy Counter 81
3.31 F0h - Enable/Disable Power Supply 82
3.32 F1h - Get Power Supply Status 83
3.33 F2h - Assert/Deassert PCIe Fundamental Reset State 83
3.34 F3h - Get PCIe Fundamental Reset State 84
3.35 F4h - Set/Release Thermal Alert 85
3.36 F5h - Get Power Brake State 85
3.37 F6h - Get Thermal Alert State 87
3.38 F7h - Set Error LED State 87
3.39 F8h - Get Board Power Supply Status 88
3.40 F9h - Assert Thermal Alert 88
3.41 FAh - Get/Set MCU FW Write-Protect 89
3.42 FBh - Access MCU Scratch Registers 90

4 SMBus Interface Additional Details 92
4.1 GPU Slave Transaction Types 92
4.2 Byte Ordering 93
4.2.1 Binary Values 93
4.2.2 String Values 93
4.3 Scratch Memory 93
4.4 Internal State Registers 94
4.4.1 Internal State Registers Overview 94
4.4.2 Scratch Memory Bank Register 94
4.4.3 Events Pending Register 94
4.4.4 Event Mask Register 96

5 Future SMBPBI API Previews 97
5.1 04h – Get Power (Additions) 97
5.2 10h – Submit/Poll Asynchronous Request (Additions) 98
5.3 18h – Query GPU State Flags (Additions) 99

5.4 1Ah – Query NVLink Information (Additions) 100
5.5 1Bh – Query Clock Frequency Info (Additions) 102
5.6 TBD - Get Dynamic System Information 104
5.7 TBD – Get Voltage Information 105
5.8 TBD – GPU Performance Monitoring 106
A Implementation of Specific Parameters 110

List of Tables

Table 2-1. GPU Command Register Format 7
Table 2-2. GPU Command Status Register Format 7
Table 2-3. Data Register Format 9
Table 2-4. Extended Data Register Format 9
Table 2-5. SMBUS Direct Data Registers 12
Table 3-1. Status Return Values 14
Table 3-2. List of Opcodes 16
Table 3-3. GPU Capability DWord(0) 19
Table 3-4. GPU Capability DWord(1) 19
Table 3-5. GPU Capability DWord(2) 21
Table 3-6. GPU Capability DWord(3) 22
Table 3-7. GPU Capability DWord(4) 22
Table 3-8. Get GPU Information Arg1 Encoding 28
Table 3-9. Asynchronous Requests 42
Table 3-10. Asynchronous Request Status Codes 48
Table 4-1. Scratch Memory Register Layout 94
Table 4-2. Events Pending Register Layout 95
Table 4-3. Event Mask Register Layout 96
Table 5-1. Get Dynamic System Information Arg1 Encoding 105
Table A-1. GPU SMBus Post Box Register Addresses 110
Table A-2. GPU-Specific Features and Capabilities 111

List of Figures

Figure 2.1 Accessing the Attachments 3
Figure 2.2 SMBus Post-Box Interface 4
Figure 2.3 Master Command-Submission Protocol 11

Chapter 1. Introduction

This document defines an Application Programming Interface between a microcontroller, typically the System Embedded Controller (EC) or Baseboard Management Controller (BMC), 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.

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 13

  Provides a list of the command opcodes and their description.
- "SMBus Interface Additional Details" on page 92

  Provides additional technical details about the SMBPBI.
- "Implementation of Specific Parameters" on page 110

  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 SMBPBI 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.

  Note: The command "./oobtest -h" provides the binary input argument options.

- NVIDIA_products_SMBPBI_capabilties.xlsx

  This document describes the SMBPBI capabilities for different NVIDIA GPU 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.

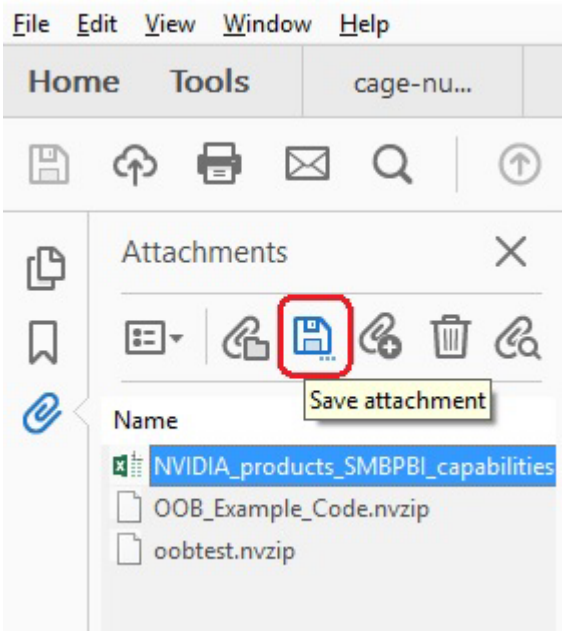


Figure 2.1 Accessing the Attachments

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) or Baseboard Management Controller (BMC). 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. Throughout 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 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.

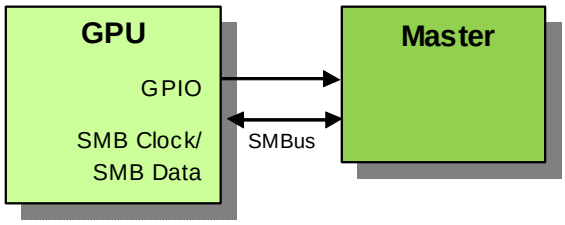


Figure 2.2 SMBus Post-Box Interface

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 Slave Addressing

2.4.1.1 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'1001111x) by default.

2.4.1.2 Slave Addressing in Dual-GPU Systems

The system can often be configured to respond to address 4Eh (b'1001110x) using board straps (*SMB_ALT_ADDR*). This is the preferred mechanism to use when resolving address conflicts in dual-GPU systems.

2.4.1.3 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.

2.4.1.4 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.

2.4.1.5 Maximum SMBPBI Command 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. Most system-dependent capabilites are announced and enabled at runtime through exposure of master and slave capability bits.

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 most Post-Box operations. These include the:

- Command and Status Register - Register used by the SMBPBI Master to submit requests to the GPU and for receiving command completion status. The same physical register is used for submitting commands and receiving command 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.

There are opcodes which support providing more than 32-bits of data at a time and use the following additional register:

- Extended Data Register - Register that contains additional requested data upon completion of a request. It may also carry additional information for the GPU at the request submission time. The encoding and usage of this register is opcode-specific.

All of these registers (Command/Status, Data, and Extended 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.

Note: The respective Post-Box register addresses 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 and Status Register

GPU commands are written to the Command and Status Register in the following format:

Table 2-1. GPU Command Register Format

Bit	Access	Default	Description
31:31	R/W	0	Command Execute Synchronization/execution bit. By writing "1" into this bit 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.
29:24	R/W	0	Reserved Must be written with zero by the master
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.

The GPU command status is returned by the Command and Status Register in the following format.

Table 2-2. GPU Command Status Register Format

Bit	Access	Description
31:31	R/W	Command Execution Status Must be zero after setting this to "1" for command execution. If this bit is not zero, the register is still holding a request and should not be interpreted as the Status Register.

Table 2-2. GPU Command Status Register Format (Continued)

30:30		Event(s) Pending If this bit is set, there are pending events for the SMBPBI master to handle.
29:29	R/W	Reserved Value is undefined
28:24	R/W	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.
23:0	R/W	Data Copy / Additional Status and Data / Undefined If the 'Copy' bit was set in the Command Register and the request has completed successfully, this field contains a copy of the Data Register[23:0] bits. Some requests may leave additional status and/or data in this field in case of a request failure. If a request doesn't leave additional status/data and if copy_bit is unset, then this value is undefined.

.

Note: The *GPU Command Status Register* has been updated with Bit[30] called 'Event(s) Pending' which coincides with the 5-bit command status provided in the MSB. The transition during a driver load is tracked as an event and will assert the Event pending flag during runtime. Note that the Event(s) pending flag must be cleared, otherwise the flag will persist.
For example, the successful command status return value expected is 0x1F but with the Events pending bit asserted, the MSB will now be returned as 0x5F.

The Event(s) Pending Bit[30] is not supported for Opcodes F0h - FBh and should be ignored when these respective opcodes are called.

Result Size Encoding

Certain commands as documented with the respective opcodes will support an encoding format which allows reporting of the result size. These commands require both the data-out and extended-data registers to store their results.

The data-copy bit-field is a part of the Status Register (23:0) and this encoding format is only supported when the 'Copy' bit is set in the request. The encoding method allows the system BMC to reduce register reads by communicating whether the result size overflows into the other registers.

Bit 23:2	Bit 1	Bit 0
Lower 22 bits of requested data	Read extended data register	Read data register

Read data register: Indicates whether the result size exceeds the 23:2 bits space in the status register when 'Copy' bit is set in the request. This means that the system BMC needs to read the data register to obtain an accurate value.

Read extended data register: Indicates the result is larger than the size of the data register and the system BMC needs to also read the extended data register to obtain an accurate value.

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 13 for more information.

Table 2-3. 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.

The *Extended Data Register* is utilized for some request types that require returning more than 32 bits of data. These requests leave additional data in the *Extended Data Register*.

Table 2-4. Extended Data Register Format

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

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.

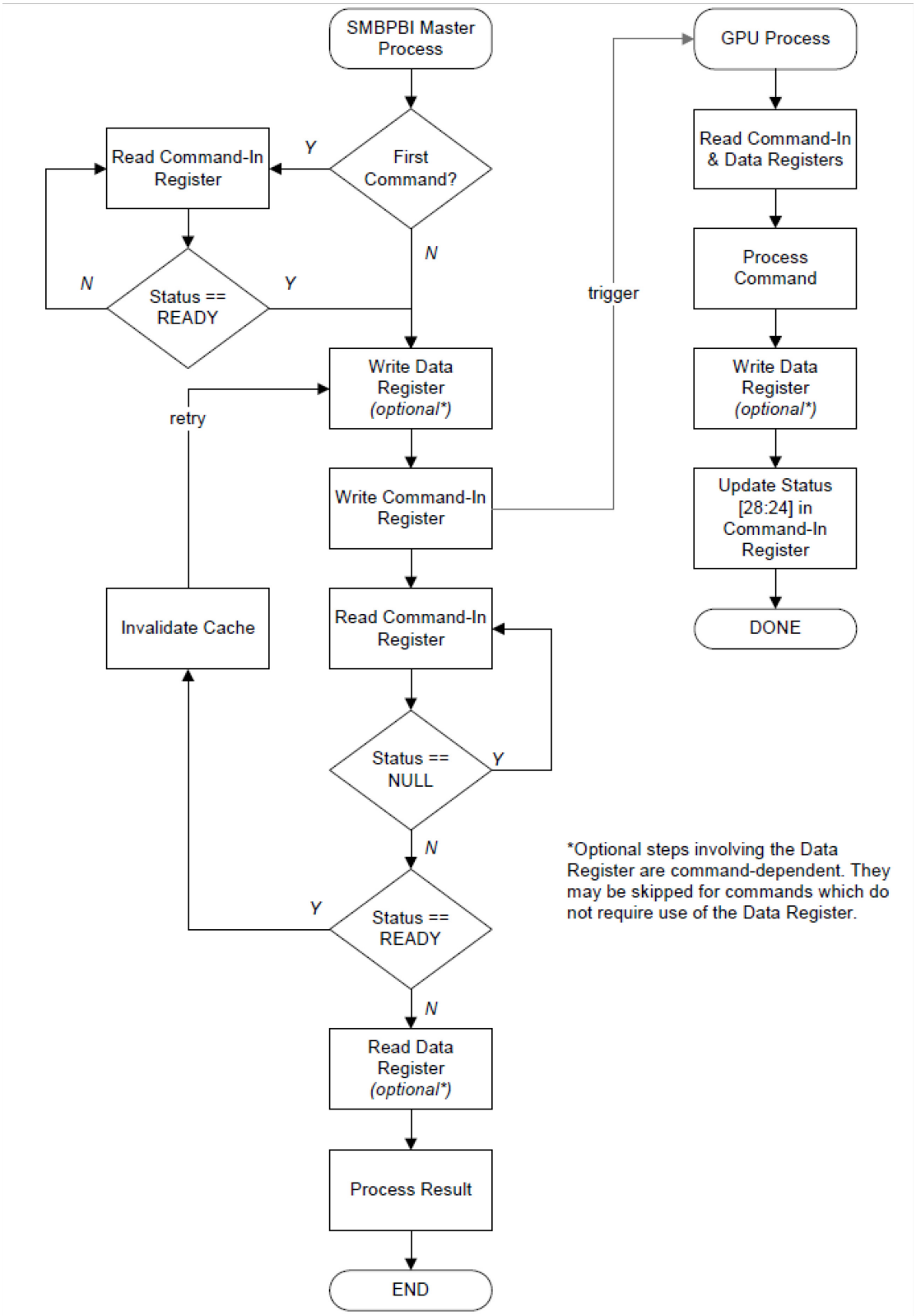
The mechanism described above precludes the possibility that the implementation phase change (and the associated capabilities change) will occur unnoticed by the SMBPBI Master.

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



2.5.6 SMBus Direct

SMBus direct allows access to a subset of GPU telemetry data via SMBus direct data registers. SMBus direct data registers are listed in the table below.

Note: The SMBPBI protocol must be utilized for full GPU telemetry data.

Table 2-5. SMBUS Direct Data Registers

Address Offset	Register Name	Access	Description
0x00	rlts (temp_h)	Read-only	GPU temperature in degrees Celsius. The format is hex integer.
0x62	vendor_id[7:0]	Read-only	Vendor ID - Lower 8 bits
0x63	vendor_id[15:8]	Read-only	Vendor ID - Upper 8 bits
0x64	device_id[7:0]	Read-only	Device ID - Lower 8 bits
0x65	device_id[15:8]	Read-only	Device ID - Upper 8 bits
0x66	sub_vendor_id[7:0]	Read-only	Sub Vendor ID - Lower 8 bits
0x67	sub_vendor_id[15:8]	Read-only	Sub Vendor ID - Upper 8 bits
0x68	sub_device_id[7:0]	Read-only	Sub Device ID - Lower 8 bits
0x69	sub_device_id[15:8]	Read-only	Sub Device ID - Upper 8 bits

Chapter 3. Interface Commands

3.1 Overview

GPU commands are defined as commands originating from the SMBPBI Master, sent to the GPU, over the SMBus interface. Commands of this type may also be referred to as Master commands. Each command-type is designated with a unique opcode to identify the command, two optional 8-bit arguments, one optional 32-bit argument, and a return status as defined in the following sections.

Slave commands are defined as requests originating from the GPU and sent to the master over the SMBus interface. Support for this type of command is optional and currently used only on select notebook platforms.

All commands, regardless of where they originate from, share the same set of completion status values.

3.1.1 Status Return Values

Table 3-1 contains a description of the status return values.

Table 3-1. Status Return Values

Return Value	Status	Description
00h	NULL	Invalid status. Can indicate the GPU is not accessible via SMBus. Also, the SMBPBI Master should use this value to clear the command status prior to command submission to determine if the command status has changed during command execution.
01h	ERR_REQUEST	Indicates SMBUS error while receiving the request.
02h	ERR_OPCODE	Request is using invalid opcode value.
03h	ERR_ARG1	Request is using invalid arg1 value (for given opcode).
04h	ERR_ARG2	Request is using invalid arg2 value (for given opcode).
05h	ERR_DATA	Invalid data are passed within current request (for given opcode).
06h	ERR_MISC	Unknown/miscellaneous error has occurred. No further details are provided.
07h	ERR_I2C_ACCESS	Internal I2C error.
08h	ERR_NOT_SUPPORTED	Requested parameter is not supported on given configuration.
09h	ERR_NOT_AVAILABLE	The system is unable to handle the request at this time, but will be able to handle it later.
0Ah	ERR_BUSY	A request is already in process.
0Bh	ERR_AGAIN	Insufficient resources to process the request. Try again.
0Ch	ERR_SENSOR_DATA	The sensor was unable to complete the data transaction.
0Dh	ERR_DISPOSITION	An error was found in the Result Disposition Rules. The Status Register additional status field contains the index of the failing rule.
1Bh	PARTIAL_FAILURE	Some of the individual requests in the bundle have failed.
1Ch	ACCEPTED	Asynchronous request is accepted. Poll request is still being processed.
1Dh	INACTIVE	The GPU slave is inactive and client should not submit requests.

Table 3-1. Status Return Values (Continued)

Return Value	Status	Description
1Eh	READY	The request has not been executed and needs to be resubmitted. The GPU slave is active and ready to accept requests from the SMBPBI Master. This status code is also indicating that there was a change in the Implementation Phase. See Section 3.2 for a detailed discussion of this status code.
1Fh	SUCCESS	Request was successfully processed and data register holds output value/result (if defined for given opcode).

3.1.2 Command Listing

The GPU will accept the commands listed in Table 3-2 via the Post Box interface. Some commands will work only when the NVIDIA driver is loaded. The capability command can be used to query the supported commands of the device, and can also be used to determine if the driver is loaded or unloaded. If a supported command is sent to a GPU when the driver is not loaded, the command will fail with ERR_NOT_SUPPORTED. .

Note: On Linux platforms, the NVIDIA driver is automatically unloaded if the GPU is idle. Persistence mode can be enabled via 'nvidia-smi -pm 1' to always keep the driver loaded and ensure SMBPBI commands requiring the driver loaded can be executed successfully.

Table 3-2. List of Opcodes

Opcode	Description	Driver Load Needed
00h	00h - No-op (no action) Request	No
01h	01h - Get Capabilities	No
02h	02h - Get Temperature (Single-Precision)	No
03h	03h - Get Temperature (Extended-Precision)	Yes[a]
04h	04h - Get Power	Yes
05h	05h - Get GPU Information	No
07h	07h - Query ECC Statistics - Format v2; format V2	No
0Ch	0Ch - Query ECC Statistics - Format v3; format V3	No
0Dh	0Dh - Read From Scratch Memory	Yes
0Eh	0Eh - Write Into Scratch Memory	Yes
0Fh	0Fh - Copy Block Into Scratch Memory	Yes
10h	10h - Submit/Poll Asynchronous Request	Yes
11h	11h - Access Internal State Registers	Yes
12h	12h - Check External Power	Yes
13h	13h - Read Dynamic Page Retirement Statistics	No
14h	14h - Query ECC Statistics - Format V4	No
15h	15h - Read Thermal Parameters	Yes
16h	16h - Query ECC Statistics - Format V5	No
17h	17h - Get/Set Write-Protect Mode	No
18h	18h - Query GPU State Flags	No
19h	19h - Get Accumulated GPU Context/SM Utilization Time	No
1Ah	1Ah - Query NVLink Information	No

Table 3-2. List of Opcodes (Continued)

Opcode	Description	Driver Load Needed
1Bh	1Bh - Query Clock Frequency Info	No
1Ch	1Ch - Kick off a Request Bundle	No
1Dh	1DH - Request the Oldest Driver Event Message (DEM) from the Server Buffer	No
1Eh	1Eh - Request ECC Statistics (Format V6)	No
20h	20h - Request Row-remapping Statistics	No
21h	21h - Query PCIe Link Status and Error Counts	No
22h	22h - Query Energy Counter	No
F0h	F0h - Enable/Disable Power Supply[b]	No
F1h	F1h - Get Power Supply Status[b]	No
F2h	F2h - Assert/Deassert PCIe Fundamental Reset State[b]	No
F3h	F3h - Get PCIe Fundamental Reset State[b]	No
F4h	F4h - Set/Release Thermal Alert[b]	No
F5h	F5h - Get Power Brake State[b]	No
F6h	F6h - Get Thermal Alert State[b]	No
F7h	F7h - Set Error LED State[b]	No
F8h	F8h - Get Board Power Supply Status[b]	No
F9h	F9h - Assert Thermal Alert[b]	No
FAh	FAh - Get/Set MCU FW Write-Protect[b]	No
FBh	FBh - Access MCU Scratch Registers[b]	No

a. This opcode can function without the driver loaded for NVIDIA Volta GPUs and later.

b..Opcodes F0h - FBh are serviced by the MCU/FPGA which does not support the 'Events Pending' Command/Status Register Bit[30]. The Events Pending (Bit[30]) should be ignored when executing any of these respective opcodes

All other values are reserved.

3.2 00h - No-op (no action) Request

Used by the SMBPBI Master to verify that the SMBus Post-Box Interface is available. Always returns SUCCESS status.

Command

Opcode	00h – No-op
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.3 01h - Get Capabilities

Retrieve the capabilities of the SMBus Post-Box Interface. Capabilities are requested and reported at bit-fields in sets (or groups) of 32 to match the width of the Data Register. A '1' indicates the sub-function or capability is supported. Capabilities are grouped by related functionality. See Arg1 and Data encoding for details.

Note: Users can directly poll for temperature and product ID information on the SMBus without using the Post-Box Interface. See *SMBus Interface for NVIDIA GPUs* (DA-06344-001) for more information.

Command

Opcode	01h – Get Capabilities
Arg1	Requested capability set (zero-based dword #i)
Arg2	Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 cap_dword(i)

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Command completed successfully.

Table 3-3. GPU Capability DWord(0)

Bits	Description
0:0	Sensor measuring temperature of primary GPU
1:1	Sensor measuring temperature of secondary GPU (on dual GPU boards)
3:2	Reserved
4:4	Sensor measuring temperature of the GPU board
5:5	Sensor measuring temperature of the GPU board memory (feature is supported if corresponding bit is set)
7:6	Reserved
11:8	Specifies how many fractional bits are used by extended precision of temperature reading (valid values are {0..8})
15:12	Reserved
16:16	Total board power consumption (feature is supported if corresponding bit is set)
23:17	Reserved
24:24	Reading GPU target temperature (Op 0x15, Arg1 0x00) supported
25:25	Reading GPU slowdown temperature (Op 0x15, Arg1 0x01) supported
26:26	Reading GPU shutdown temperature (Op 0x15, Arg1 0x02) supported
27:27	Reading memory target temperature (Op 0x15, Arg1 0x03) supported
28:28	Reading GPU maximum operating temperature (0p 0x15, Arg1 0x04) supported
31:29	Reserved

Table 3-4. GPU Capability DWord(1)

Bits	Description
0:0	Board part number version 1 supported (Op 0x05, Arg1 0x00)

Table 3-4. GPU Capability DWord(1) (Continued)

1:1	OEM information, format version 1 supported (Op 0x05, Arg1 0x01)
2:2	Serial number, format version 1 supported (Op 0x05, Arg1 0x02)
3:3	Marketing name, format version 1 supported (Op 0x05, Arg1 0x03)
4:4	GPU silicon revision, format version 1 supported (Op 0x05, Arg1 0x04)
5:5	Memory vendor, format version 1 supported (Op 0x05, Arg1 0x05)
6:6	Memory part number, format version 1 supported (Op 0x05, Arg1 0x06)
7:7	Build date, format version 1 supported (Op 0x05, Arg1 0x07)
8:8	Firmware version, format version 1 supported (Op 0x05, Arg1 0x08)
9:9	PCI configuration vendor ID, format version 1 supported (Op 0x05, Arg1 0x09)
10:10	PCI configuration device ID, format version 1 supported (Op 0x05, Arg1 0x0a)
11:11	PCI configuration subsystem vendor ID, format version 1 supported (Op 0x05, Arg1 0x0b)
12:12	PCI configuration subsystem device ID, format version 1 supported (Op 0x05, Arg1 0x0c)
13:13	GPU GUID, format V1 is available (Op 0x05, Arg1 0x0d)
14:14	InfoROM version, format V1 is available (Op 0x05, Arg1 0x0e)
15:15	Reserved
16:16	ECC statistics data format V1 is available (Op 0x06)
17:17	ECC statistics data format V2 is available (Op 0x07)
18:18	ECC statistics data format V3 is available (Op 0x0C)
19:19	Retired page count query is available (Op 0x13)
20:20	ECC statistics data format V4 is available (Op 0x14)
21:21	ECC statistics data format V5 is available (Op 0x16)
22:22	Get/Set Write-Protect Mode is supported (Op 0x17)
23:23	Get ECC enabled state is supported (Op 0x18, Arg1 0x0)
24:24	Get GPU reset required state flag is supported (Op 0x18, Arg1 0x01)
25:25	Get GPU accumulated context/SM utilization time is supported (Op 0x19)
26:26	Query NVLink Information (number of NVLinks, NVLink link status, NVLink speed) is supported (Op 0x1A, Arg1 0x00 - 0x02)
27:27	Query NVLink Information (error counts) is supported (Op 0x1A, Arg1 0x03 - 0x06)
28:28	Query Clock Frequency Information is supported (Op 0x1B)
29:29	Get MIG Enabled State is supported (Op 0x18, Arg1 0x00)
30:30	ECC statistics data format v6 is available (Op 0x1E)
31:31	Reserved

Table 3-5. GPU Capability DWord(2)

Bits	Description
0:0	NVIDIA GPU driver is not loaded. See Section 3.2 for a discussion of the protocol implementation phases.
1:1	Reserved
4:2	Size of the scratch space (as a power of 2): 000 - scratch space not available 001 - 4 banks 010 - 8 banks ... 111 - 256 banks
5:5	Fan queries version V1 are available. (See "List of Asynchronous Requests" on page 42)
6:6	Product length version 1 supported (Op 0x05, Arg1 0x0F)
7:7	Product width version 1 supported (Op 0x05, Arg1 0x10)
8:8	Product height version 1 supported (Op 0x05, Arg1 0x11)
9:9	PCIe link speed version 1 supported (Op 0x05, Arg1 0x12)
10:10	PCIe link width version 1 supported (Op 0x05, Arg1 0x13)
11:11	TGP limit version 1 supported (Op 0x05, Arg1 0x14)
12:12	Scratch space bank size (0 for 1k, 1 for 256B)
13:13	Row-mapping statistics supported (Op 0x20)
14:14	PCIe link status and error counts query supported (Op 0x21)
15:15	Drain and reset recommended reporting supported (Op 0x18, Arg1 0x1)
16:16	NVLink throughput counter reporting supported (Op 0x1A; Arg1 0x09, 0x0A, 0x0C)
17:17	NVLink status V2 reporting is supported (Op 0x1A; Arg1 0x07)
18:18	NVLink sublink width reporting is supported (Op 0x1A; Arg1 0x08)
19:19	Query energy counter request supported (Op 0x22)
20:20	Row-remapping pending query request supported (Op 0x20, Arg1 0x1, Arg2 0x0 - 0x2)
21:21	Row-remapping histogram query request supported (Op 0x20, Arg1 0x2)
24:22	Reserved
25:25	Requested PCIe link speed query is supported (Op 0x21, Arg1 0x3)
26:26	Current performance state query is supported (Op 0x1B, Arg1 0x3)
27:27	NVLink availability query is supported (Op 0x1A, Arg1 0x3)
31:28	Reserved

Table 3-6. GPU Capability DWord(3)

Bits	Description
0:0	Enable/Disable Power Supply request supported (Op 0xF0)
1:1	Get Power Supply Status request supported (Op 0xF1)
2:2	Assert/Deassert PCIe Fundamental Reset request supported (Op 0xF2)
3:3	Get PCIe Fundamental Reset State request supported (Op 0xF3)
4:4	Set/Release Thermal Alert request supported (Op 0xF4)
5:5	Get Power Brake State request supported (Op 0xF5)
6:6	Get Thermal Alert State request supported (Op 0xF6)
7:7	Set Error LED State request supported (Op 0xF7)
8:8	Get Board Power Supply Status request supported (Op 0xF8)
9:9	Set Thermal Alert request supported (Op 0xF9)
10:10	Get/Set MCU FW Write-Protect request supported (Op 0xFA)
11:11	Access MCU Scratch Register supported (Op 0xFB)
31:12	Reserved

Table 3-7. GPU Capability DWord(4)

Bits	Description
0:0	Get HW violation time values since GPU reset/reload request supported (Op 0x10 Arg1 0x09)
1:1	Get global SW violation time values since GPU reset/reload request supported (Op 0x10 Arg1 0x09)
2:2	Get power policy violation time values since GPU reset/reload request supported (Op 0x10 Arg1 0x09)
3:3	Get thermal policy violation time values since GPU reset/reload request supported (Op 0x10 Arg1 0x09)
4:4	Get current SM and memory utilization values request supported (Op 0x10 Arg1 0x0A)
5:5	Get Driver Event Message request supported (Op 0x1D)
6:6	Request bundling is supported (Op 0x1C)
7:7	Reserved
8:8	Set ECC mode is supported (Op 0x10, Arg1 0x0C)
9:9	Reserved
10:10	Set MIG mode is supported (Op 0x10, Arg1 0x0C)
11:11	Get/Set fan curve points is supported (Op 0x10, Arg1 0x0F)

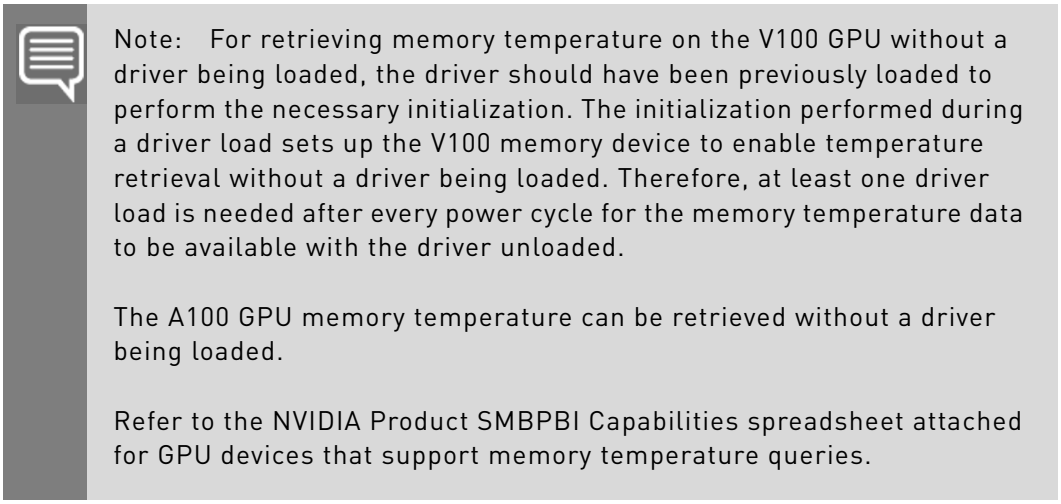
Table 3-7. GPU Capability DWord(4) (Continued)

31:12	Reserved

See the attached document *NVIDIA products SMBPBI capabilities.xlsx* to view SMBPBI capability support for NVIDIA GPU products.

3.4 02h - Get Temperature (Single-Precision)

Retrieves single-precision temperature reading (in Celsius) for a specific temperature source (lowest 8 bits are zero).



Note: For retrieving memory temperature on the V100 GPU without a driver being loaded, the driver should have been previously loaded to perform the necessary initialization. The initialization performed during a driver load sets up the V100 memory device to enable temperature retrieval without a driver being loaded. Therefore, at least one driver load is needed after every power cycle for the memory temperature data to be available with the driver unloaded.

The A100 GPU memory temperature can be retrieved without a driver being loaded.

Refer to the NVIDIA Product SMBPBI Capabilities spreadsheet attached for GPU devices that support memory temperature queries.

Command

Opcode	02h – Get Temperature (Single-Precision)
Arg1	Temperature source (required sensor)
	00h – sensor on primary GPU (GPU_0)
	01h – sensor on secondary GPU (GPU_1, available only on dual-GPU boards)
	04h – sensor monitoring board temperature (if supported)
	05h – sensor monitoring memory temperature (if supported)
	(all other values are reserved)
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Fixed-point signed integer with 8 fractional bits (7:0 all zero) representing the source temperature in Celsius. The temperature is stored in 31:8 and must be right-shifted by 8 bits by the master to get the correct representation.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Temperature was successfully retrieved.
ERR_NOT_SUPPORTED	Requested sensor is not supported on given configuration.
ERR_ARG1	Invalid sensor ID.
ERR_SENSOR_DATA	Sensor was not able to complete data transaction.

Note: Before making this call check cap_dword(i) for all supported temperature sources.

3.5 03h - Get Temperature (Extended-Precision)

Retrieves extended-precision temperature reading (in Celsius) for a specific temperature source.

Note: This opcode can function without the driver loaded for Volta GPUs and later, though a prior driver load to retrieve memory temperature is still required.

Command

Opcode	03h – Get Temperature (Extended-Precision)
Arg1	Desired temperature source. Refer to the temperature sources listed in the Single-Precision version of this command above for the full list of available sources.
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0 Fixed-point signed integer with 8 fractional bits representing the source temperature in Celsius. The number of non-zero fractional bits depends on the thermal source used and may vary from 2 to 5.

31:31 – Sign bit

30:8 – Integer bits

7:0 – Fractional bits

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Temperature was successfully retrieved.
ERR_NOT_SUPPORTED	Requested sensor is not supported on given configuration.
ERR_ARG1	Invalid sensor ID.
ERR_SENSOR_DATA	Sensor was not able to complete data transaction.

Note: Before making this call check cap_dword(i) for all supported temperature sources. Additionally, use cap_dword(i) to retrieve the precision supported by given configuration.

3.6 04h - Get Power

Read power consumption (in mW) from a specific source.

Command

Opcode	04h – Get Power
Arg1	Desired power source. 00h – Total board power consumption. *(all other values are reserved)*
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Power in mW (100mW resolution)

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Power was successfully retrieved.
ERR_NOT_SUPPORTED	Requested power-source is not supported on given configuration.
ERR_ARG1	Invalid power-source.

Note: Before making this call check cap_dword(i) for all supported power sources

3.7 05h - Get GPU Information

Read static GPU and board identification data, as specified by "type" and "offset". Availability of individual types in a given implementation is reflected in capability dwords.

Command

Opcode	05h – Get GPU Information
Arg1	Type of information desired. *See Table 3-8 on page 28.*
Arg2	The offset into the requested data in the increments of 4 bytes. Available types of information with their respective sizes, locations, and format versions are listed below.

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Requested data. Encoding is request-specific; see table below for details.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully retrieved.
ERR_ARG1	Arg1 value does not correspond to a supported information-type.
ERR_ARG2	Offset stored in Arg2 is invalid for the requested information-type.

Table 3-8. Get GPU Information Arg1 Encoding

Arg1 Value	Size (Bytes)	Type	Example	Format Version	Capability Location (dword/bit)	Note
00h	24	Board part number	900-21228-3850-100	1	1/0	Top_Level - PCB - SKU – Revision 900 is the Top Level, customer orderable 21228 is the PCB 3850 is the SKU 100 is the Revision

Table 3-8. Get GPU Information Arg1 Encoding (Continued)

Arg1 Value	Size (Bytes)	Type	Example	Format Version	Capability Location (dword/bit)	Note
01h	Up to 504	OEM information	N/A	1	1/1	Bytes 0-7 reserved for header Bytes 8-504 for OEM use
02h	16	Serial number	0322411000001	1	1/2	NVIDIA Reserved, Week, Year, Serial ID In the example, "032" are NVIDIA reserved bytes, 24 is the numerical work week of year, 11 is the last 2 digits of the year, and 000001 is the incremental serial ID for the week.
03h	24	Marketing name	Tesla X2090	1	1/3	
04h	16[a]	GPU part number	1091-890-A2	1	1/4	GPU_DeviceID - GPU_SKU - GPU_Revision 1091 is the GPU DeviceID of P1228 890 is the GPU SKU A2 is the GPU Revision
05h	1	Memory vendor	H	1	1/5	H = Hynix S= Samsung
06h	20	Memory part number	161-0107-100	1	1/6	
07h	4	Build date	20101221	1	1/7	Represents December 21, 2010
08h	14	Firmware version	70.10.40.00.09	1	1/8	
09h	2	PCIe configuration vendor ID	0x17DB	1	1/9	

Table 3-8. Get GPU Information Arg1 Encoding (Continued)

Arg1 Value	Size (Bytes)	Type	Example	Format Version	Capability Location (dword/bit)	Note
0Ah	2	PCIe configuration device ID	0x1091	1	1/10	
0Bh	2	PCIe configuration subsystem vendor ID	0x10DE	1	1/11	
0Ch	2	PCIe configuration subsystem device ID	0x088E	1	1/12	
0Dh	16	GPU GUID	GPU-782cbd6a-4a41-5e9b-a3c2-4cc625ec18b0	1	1/13	
0Eh	16	InfoROM version	G500.0200.00.03	1	1/14	
0Fh	4	Product length		1	2/6	
10h	4	Product width		1	2/6	
11h	4	Product height		1	2/6	
12h	1	PCIe link speed	0x4	1	2/6	This field represents the maximum PCIe link speed capabilities of the respective product. This field does not provide the current PCIe link speed status. 4 - PCIe Gen 4.0 3 - PCIe Gen 3.0 2 - PCIe Gen 2.0 1 - PCIe Gen 1.0

Table 3-8. Get GPU Information Arg1 Encoding (Continued)

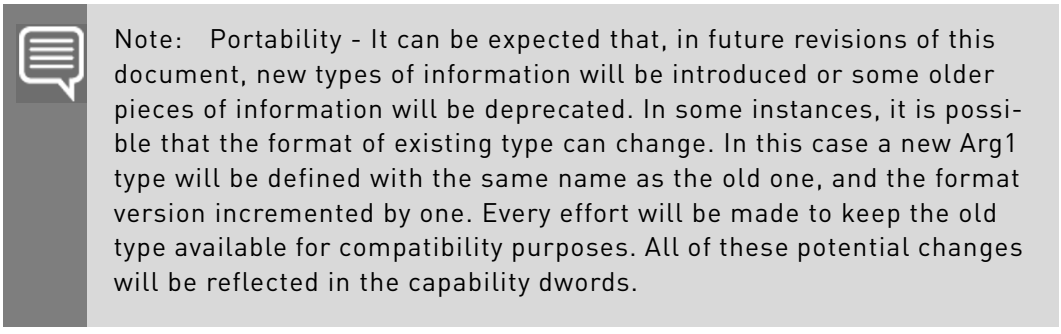
Arg1 Value	Size (Bytes)	Type	Example	Format Version	Capability Location (dword/bit)	Note
13h	1	PCIe link width	0x10	1	2/6	This field represents the maximum PCIe link width capabilities of the respective product. This field does not provide the current PCIe link width status. 0x10 - x16 0x8 - x8 0x4 - x4 0x1 - x1
14h	4	TGP limit	0x61a80 = 400,000 mW	1	2/6	Returns TGP limit in hexadecimal milliwatts

a. The 16-byte field for the GPU part number is a null-terminated string.

Example Usage

Execute(opcode=05h, arg1=02h, arg2=00h) - Delivers bytes 0..3 of the 16-character long serial number into the Data register.

Execute(opcode=05h, arg1=00h, arg2=01h) - Delivers bytes 4..7 of the 24-character long Board part number into the Data register.



Note: Portability - It can be expected that, in future revisions of this document, new types of information will be introduced or some older pieces of information will be deprecated. In some instances, it is possible that the format of existing type can change. In this case a new Arg1 type will be defined with the same name as the old one, and the format version incremented by one. Every effort will be made to keep the old type available for compatibility purposes. All of these potential changes will be reflected in the capability dwords.

3.8 07h - Query ECC Statistics - Format v2

Read memory ECC statistics data, format version V2.

Command

Opcode	07h – Query ECC Statistics
Arg1	Encoding specific to query-type. See tables below for usage.
Arg2	Encoding specific to query-type. See tables below for usage.

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Encoding specific to request-type. See tables below for usage.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

Note: Before making this call check appropriate cap_dword(i) for support.

Query-Type 1: Querying Aggregate Counters

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:6	00h selects aggregate counters
DEVICE	5:4	00h selects the frame buffer (FB) 01h selects the graphics device (GR)
SOURCE	3:2	00h, device being FB, selects level 2 cache (LTC) 01h, device being FB, selects the frame buffer (FB) 00h, device being GR, selects level 1 cache (L1) 01h, device being GR, selects SM register file (RF)
TYPE	1:0	00h selects the single-bit error counter (SBE) 01h selects the double-bit error counter (DBE)

Arg2 Encoding (Device-Specific):

DEVICE == FB

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-5
SLICE	3:0	Slice number in the range 0-1

DEVICE == GR

Field Name	Position	Value & Description
GPC	7:4	GPC number in the range 0-1
TPC	3:0	TPC number in the range 0-1

Query-Type 2: Querying Frame Buffer Individual Addresses Statistics

The driver gathers ECC data for up to 14 individual addresses and cumulative SBE/DBE counters per partition/slice pair. Below are the Arg1/Arg2 layouts that facilitate access to this information.

Arg2 Encoding:

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-5
SUBPARTITION	3:0	Subpartition number in the range 0-1

Arg1 Encoding for cumulative SBE/DBE counters per partition/slice pair:

Field Name	Position	Value & Description
SEL	7:6	01h selects per partition/slice frame buffer statistics
TARGET	5:5	01h selects cumulative SBE/DBE counters
unused	4:0	X

The query result appears in the Data register and has the following layout:

Field Name	Position	Value & Description
DBE	31:16	cumulative double-bit errors
SBE	15:0	cumulative single-bit errors

Arg1 Encoding for individual addresses in the slice/partition pair:

Field Name	Position	Value & Description
SEL	7:6	01h selects per partition/slice frame buffer statistics
TARGET	5:5	00h selects individual address data
ADDR/CNT	4:4	00h selects DBE/SBE counters 01h selects the address value itself
INDEX	3:0	index onto the array of addresses in the range 0-13

The query result appears in the Data register and has the following layout:

Field Name	Position	Value & Description
DBE	23:16	cumulative double-bit errors
SBE	7:0	cumulative single-bit errors

3.9 0Ch - Query ECC Statistics - Format v3

Read memory ECC statistics data, format version V3 for NVIDIA Kepler and Maxwell GPUS.

Format version 3 adds support for a new type of error count, and updates the structure to support geometry changes between Fermi and Kepler based chips, preserving unit-specific counters for each type of error. Supporting changes include the following:

- Slices per partition changed from 2 to 4 for LTC.
- Number of GPCs increased from 4 to 5.
- Number of TPCs per GPC decreased from 4 to 3.
- Added texture parity counters.
- Removed cumulative counters, which were redundant with aggregate counters.

Command

Opcode 0Ch – Query ECC Statistics

Arg1 Encoding specific to query-type. See tables below for usage.

Arg2 Encoding specific to query-type. See tables below for usage.

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 Encoding specific to request-type. See tables below for usage.

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Command completed successfully.

Note: Before making this call check appropriate cap_dword(i) for support.

Query-Type 1: Querying Aggregate Counters

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:6	00h selects aggregate counters
DEVICE	5:4	00h selects the frame buffer (FB) 01h selects the graphics device (GR)
SOURCE	3:2	00h, device being FB, selects level 2 cache (LTC) 01h, device being FB, selects the frame buffer (FB) 00h, device being GR, selects level 1 cache (L1) 01h, device being GR, selects SM register file (RF) 02h, device being GR, select texture units (TEX)
TYPE	1:0	00h selects the single-bit error counter (SBE) 01h selects the double-bit error counter (DBE)

Arg2 Encoding (Device-Specific):

DEVICE == FB

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-5
SLICE/ SUBPARTITION	3:0	Slice number in the range 0-3 (when SOURCE selected is LTC (00h)) Subpartition number in the range 0-1 (when SOURCE selected is FB (01h))

DEVICE == GR

Field Name	Position	Value & Description
TEX	7:6	Texture unit number in the range 0-3 (when SOURCE selected is TEX (02h))
GPC	5:3	GPC number in the range 0-4
TPC	2:0	TPC number in the range 0-2

Query-Type 2: Querying Frame Buffer Individual Addresses Statistics

The driver gathers ECC data for up to 16 individual addresses per partition/subpartition pair. Below are the Arg1/Arg2 layouts that facilitate access to this information.

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:6	01h selects per partition/slice frame buffer statistics
MBZ	5:5	00h must be zero
ADDR/CNT	4:4	00h selects DBE/SBE counters 01h selects the address value itself
INDEX	3:0	Index into the array of addresses in the range 0-15

Arg2 Encoding:

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-5
SUBPARTITION	3:0	Subpartition number in the range 0-1

The query result appears in the Data register and has the following layout:

Field Name	Position	Value & Description
DBE	23:16	Double-bit errors
SBE	7:0	Single-bit errors

3.10 0Dh - Read From Scratch Memory

The opcode reads a word from the scratch memory into the data register.

The effective byte address of the word in memory is evaluated according to the following formula:

`effAddr = (readBank * 0x400) + (src * 4)`, where

- `effAddr` : effective byte address
- `readBank` : contents of the read bank internal state register (Section 4.4.1)
- `src` : offset value specified in Arg1

See Section 4.3 for more information about scratch memory.

Command

Opcode	0Dh – Read scratch memory
Arg1	Address offset where the scratch memory word is located
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Requested word from scratch memory

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully retrieved.
ERR_NOT_SUPPORTED	Scratch memory is not supported in this configuration.
ERR_MISC	DMA controller error.

3.11 0Eh - Write Into Scratch Memory

The opcode writes data from the data register into the scratch memory.

The effective byte address of the initial memory destination is evaluated according to the following formula:

effAddr = (writeBank * 0x400) + (dst * 4), where

effAddr : effective byte address

writeBank : contents of the write bank internal state register (Section 4.4.1)

dst : offset value specified in Arg1

If during this operation the destination address reaches the end of the entire scratch memory block (see Appendix A.2), it wraps around and resumes writing at the start of the scratch memory block (address 0).

See Section 4.3 for more information about scratch memory.

Command

Opcode	0Eh – Write into scratch memory
Arg1	Address offset where the data is to be written
Arg2	The number of consecutive words to write into scratch memory, decremented by 1.

Data-In

Bit 31:0	Data to write into scratch memory

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully written.
ERR_NOT_SUPPORTED	Scratch memory is not supported in this configuration. (See capability word 2, bits 4:2)
ERR_MISC	DMA controller error.

3.12 0Fh - Copy Block Into Scratch Memory

The opcode copies the contents of a memory block in scratch memory (Section 4.4.1).

- The effective byte address of the memory block to copy from is evaluated according to the following formula:

  effAddr = (readBank * 0x400) + (src * 4), where

  effAddr : effective byte address

  readBank : contents of the read bank internal state register (Section 4.4.1)

  src : offset value specified in the least significant byte (bits 7:0) of the data register

- The effective byte address of the memory destination is evaluated according to the following formula:

  effAddr = (writeBank * 0x400) + (dst * 4), where

  effAddr : effective byte address

  writeBank : contents of the write bank internal state register (Section 4.4.1)

  dst : offset value specified in Arg1

- The size in words of the block to copy, decremented by 1, is specified in Arg2.
- The source and destination blocks may not overlap or wrap around the end of the scratch memory block.

See Section 4.3 for information about scratch memory.

Command

Opcode	0Fh – Copy scratch memory
Arg1	Address offset where the data is to be copied
Arg2	The number of consecutive words to copy, decremented by 1.

Data-In

Bit 31:0	Address offset of the data to be copied

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully copied.
ERROR_NOT_SUPPORTED	Scratch memory is not supported in this configuration. (See capability word 2, bits 4:2)

ERR_MISC	DMA controller error.
ERR_DATA	Source buffer size extends beyond the end of the scratch memory block.
ERR_ARG1	Destination buffer extends beyond the end of the scratch memory block.
ERR_ARG2	Source and destination buffers overlap.

3.13 10h - Submit/Poll Asynchronous Request

A certain subset of supported functionality requires assistance from the GPU driver running on the host system. Because the GPU driver is a part of the OS that does not necessarily conform to real time requirements, the response time for such a request is potentially unbounded.

If this type of request was implemented in a synchronous way, like other requests, a long response time would make the SMBus interface unavailable for any other urgent request for the whole duration of the original request execution. For this reason, requests that demand participation of the host OS GPU driver are implemented asynchronously.

This opcode is used either to issue an asynchronous request to the GPU driver, or to poll the status of an asynchronous request.

Some of the asynchronous requests are able to be started without the driver loaded; however, the driver must be loaded to complete the operation on a write (for example Arg1 = 0x0B).

Command

Opcode	10h – Submit/poll asynchronous request
Arg1	*0x00 to 0xFE*: Indicates which asynchronous request to issue See Table 3-9 for the list of supported requests.
	0xFF: Indicates an asynchronous request is to be polled
Arg2	(Arg1=0x00 to 0xFE): The address offset of the parameters for the asynchronous request
	(Arg1=0xFF): The ID of the asynchronous request to poll

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	(Arg1=0x00 to 0xFE): The request ID assigned to the asynchronous request
	(Arg1=0xFF): The status of the polled asynchronous request

Extended Data-Out

Bit 31:0 Unused

Status

ERR_ARG2	The request ID supplied in Arg2 does not correspond to an active request.
ERR_AGAIN	The system lacks necessary resources to accept the request.
ERR_BUSY	An attempt was made to issue an asynchronous request while another one is already issued.
ACCEPTED	The request is still being executed by the GPU driver.

Submitting a Request

First, fill the parameters appropriate for the request into scratch memory (Section 4.3).

Then issue the request (opcode 0x10) with the request type encoded in Arg1 according to the list below ("List of Asynchronous Requests" on page 42), and Arg2 containing the pointer to the parameter block in the scratch memory.

The effective byte address of the parameter block is evaluated according to the following formula:

effAddr = (readBank * 0x400) + (src * 4), where

effAddr : effective byte address

readBank : contents of the read bank internal state register (Section 4.4.1)

src : offset value specified in Arg2

Unless another asynchronous request is already pending, the new request gets accepted immediately and passed to the host OS GPU driver for execution. The data register contains an ID assigned to the request.

In the current implementation only a single asynchronous request may be issued at a time. In case an attempt is made to issue an asynchronous request while another one is already in process, a ERR_BUSY (0x0A) status code is returned. The data register will contain the request ID of the request in process. This restriction may be removed in a future implementation.

Polling a Submitted Request

Upon a successful request submission, the client will need to poll the status of the submitted request.

Issue the poll request (opcode 0x10) with Arg1 containing the code 0xFF and Arg2 containing the request ID that was previous assigned to the request.

If the asynchronous request has been processed, the status code returned will be either one of the error codes, if the values supplied in Arg1/Arg2 of the original asynchronous request were incorrect, or SUCCESS (0x1F) will be returned with the data register filled with the status code returned by the GPU driver (see Table 3-10, "Asynchronous Request Status Codes," on page 48).

Also, in case of a successful completion, the parameter block in the scratch memory may be filled with requested values, depending on the type of the original request.

List of Asynchronous Requests

The request parameters are exchanged with the client through the scratch memory (See Section 4.3). The location in the scratch memory is pointed to by the Arg2 of the instruction. The layout of these parameters in the scratch memory is explained in this section through the C language structure definitions.

The same parameter structure also accommodates the information returned to the client by the system. The structure members in the definitions below are tagged with the labels [in] for the members, that pass the information from the client, or [out] for the opposite direction.

These definitions use the following C language types, when defining the structure members:

- **NvU8** 8 bit long unsigned integer
- **NvU16** 16 bit long unsigned integer
- **NvU32** 32 bit long unsigned integer
- **NvS8** 8 bit long signed integer
- **NvS16** 16 bit long signed integer
- **NvS32** 32 bit long signed integer

Table 3-9. Asynchronous Requests

Arg1	Requested Action
0x00	Read total GPU power limit control data. Parameters: NvU32 flags - [unused] NvU32 limitCurrInput - [out] Current total GPU power limit value to enforce, requested by the SMBPBI client, measured in milliwatts. 0xffffffff is returned if the limit has not been set by the SMBPBI client. NvU32 limitCurrOutput - [out] Currently arbitrated total GPU power limit value after taking into account limits, requested by all clients, measured in milliwatts.

Table 3-9. Asynchronous Requests

Arg1	Requested Action
0x01	Set the total GPU power limit. Parameters: NvU32 flags - [in] A collection of single bit flags affecting the request handling Bit 0 - If set, directs the driver to set/clear the total GPU power limit value, supplied in "limitCurrInput", which becomes persistent across reloading the driver or restarting the system. This is achieved by storing the new limit value in the GPU board's persistent memory - InfoROM. If not set, the set/clear action will be effective only until a driver re-load or system restart. Bit 1 - - If set, it will clear total GPU power limit if it exists. In this case the values of "limitCurrInput" are ignored. If not set, directs the driver to establish the total GPU power limit value, supplied in "limitCurrInput". NvU32 limitCurrInput - [in] Current total GPU power limit value to enforce, requested by the SMBPBI client, measured in milliwatts. Must always be within range imposed by the current policy. NvU32 limitCurrOutput - [unused] Note: The minimum cycle time for the Set total GPU power limit command is 10ms.
0x02	Read the total GPU power limit policy information. Parameters: NvU32 limitMin - [out] The lower bound for the total GPU power limit. NvU32 limitMax - [out] The upper bound for the total GPU power limit. NvU32 limitDefault - [out] The default value for the total GPU power limit.
0x03	Request the number of connected fans (format version V1). This arg1 is available if the capability bit 5 is set in the capability word 2 (Table 3-5 on page 21). Parameters: NvU32 fanCount - [out] The number of fans, present on the board.

Table 3-9. Asynchronous Requests

Arg1	Requested Action
0x04	Read the fan properties (format version V1). This arg1 is available if the capability bit 5 is set in the capability word 2 (Table 3-5 on page 21). Parameters: NvU32 fanIndex - [in] Index of the fan, whose properties are requested NvU32 fanProperties - [out] Bit mask of the properties detailed below. Bit Property 0 Measurement units of the fan level: 0 - PWM (Pulse Width Modulation) percent. 1 - RPM (Revolutions Per Minute) 1 Tachometer availability 0 - tachometer not available 1 - tachometer available 2 Tachometer bounds (min/max) availability 0 - tachometer bounds not available 1 - tachometer bounds available NvU16 fanLevelMin - [out] Minimum fan level in the units, specified by the property bit 0 above. NvU16 fanLevelMax - [out] Maximum fan level in the units, specified by the property bit 0 above. NvU16 fanTachMin - [out] Minimum tachometer reading, if available (see property bit 2 above). NvU16 fanTachMax - [out] Maximum tachometer reading, if available (see property bit 2 above).
0x05	Read current fan speed and level (format version V1). This arg1 is available if the capability bit 5 is set in the capability word 2 (Table 3-5 on page 21). Parameters: NvU32 fanIndex - [in] Index of the fan, whose properties are requested NvU8 levelCurr - [out] Current fan level (0, if not available) NvU16 speedCurr - [out] Current fan speed in RPM (0, if the tachometer is not available)
0x06	Read the clock limit. This arg1 allows reading back the clock limit value enforced on the GPU. Parameters: NvU32 limitType - [in] GPU clock limit type value enforced as requested by the SMBPBI client. The supported limitType values are: Maximum customer boost clock - 0x01 NvU32 maxClkLimitMHz - [out] Current GPU clock limit value enforced, as requested by the SMBPBI client, measured in MHz (in hex).

Table 3-9. Asynchronous Requests

Arg1	Requested Action
0x07	Set the clock limit. This arg1 allows setting the clock limit value to enforce on the GPU. The clock limit set is persistent when reloading the driver or restarting the system. Parameters: NvU32 limitType - [in] GPU clock limit type value enforced as requested by the SMBPBI client. The supported limitType values are: Maximum customer boost clock - 0x01 NvU32 maxClkLimitMHz - [in] Current GPU clock limit value to enforce, as requested by the SMBPBI client, measured in MHz (in hex). Must be within the range of supported clocks.
0x08	Get current energy counter value since GPU reset/reload. Parameters: NvU64 energyCounter - [out]
0x09	Get current violation time values since GPU reset/reload. A violation is an event that forced GPU to run below optimum performance clocks. It includes - HW violation duration - Global SW violation that forced GPU below base clocks - Per-policy violation (power, therm, etc) that forced GPU below base clocks Parameters: NvU32 policyMask - [in/out] Client will populate this mask with policies whose violation times are being requested. The driver then populates this mask with policies whose violation times are being provided. Supported policies can be queried using cap_dword(4) Bit Property 0 Duration of HW violation 1 Global SW violation that forced GPU below base clocks 2 Power violation that forced GPU below base clocks 3 Thermal violation that forced GPU below base clocks NOTE: Setting policyMask to 0xFFFFFFFF will result in a query for all supported policies NvU64 violationTime[32] - [out] array of 64 bit violation durations Driver will populate violation durations at indices corresponding to the bit offset that's set in the policyMask.

Table 3-9. Asynchronous Requests

Arg1	Requested Action
0x0A	Get current SM and memory utilization value (percentage). Retrieve the current SM and memory utilization values in percentage. Parameters: NvU32 gpuUtil - [out] Current SM utilization (in percentage) NvU32 memUtil - [out] Current memory utilization (in percentage)
0x0B	Set/Clear clock limits which supersede all in-band clock configurations Parameters: NvU32 flags - [in] Bit flags affecting the request handling Bit 0 - If set, the set/clear becomes persistent across the driver re-load and/or the system restart. This is achieved through storing the new limit values in the GPU board's persistent memory (InfoROM). If not set, the set/clear will be effective only until the next driver re-loads or the system restarts. Bit 1 - If set, the driver will clear the clock limits and the values will be ignored. If not set, the driver will set the range supplied as the new clock limits which will supersede all in-band clock configurations. NvU32 clkMinFreqMHz - [in] Lower bound value of clock limit in MHz NvU32 clkMaxFreqMHz - [in] Upper bound value of clock limit in MHz NOTE: If set when a driver is not loaded, a GPU reset will cause the pending set to be dropped.
0x0C	Set/Reset the device mode Parameters: NvU16 modeType - [in] Select the device mode type: 0x00 - ECC mode 0x01 - MIG mode NvU16 flags - [in] Action for selected device mode type Bit 0 - If set, the selected device mode type will be enabled. If not set, the selected device mode type will be disabled.
0x0D	Get the Clock Limits, which supersede all in-band clock configurations Parameters: NvU16 oobClientMin - [out] The lower bound of clock limits client input NvU16 oobClientMax - [out] The upper bound of clock limits client input NvU16 enforcedMin - [out] The lower bound of current enforced clock limits NvU16 enforcedMax - [out] The upper bound of current enforced clock limits

Table 3-9. Asynchronous Requests

Arg1	Requested Action

0x0F	Get/Set fan curve points, maximum number of fan curve points is 5
	Parameters:
	NvU8 action - [in] Select request type
	0x00 - Get current fan curve points
	0x01 - Get VBIOS default fan curve points
	0x02 - Set new fan curve points
	NvU8 fanIndex - [in] Index of fan to Get/Set
	NvU8 numEffectivePoints - [out] for Get query. This is the number of points implemented and filled in the array below [in] for fanSetCurvePoints. This is how many points the caller is sending to the API. Must not exceed what was returned by fanGetCurvePoints.
	curvePoints Array[maximum number of fan curve points]
	{
	All parameters in curvePoints array are [out] for Get fan curve points or [in] for Set new fan curve points
	NvS16 temperature - [out/in] Degrees Celsius, signed 16-bit value
	Fan properties (Arg1 = 0x04) must be queried first to determine tachometer availability. If tachometer is available, rpmValue is utilized. Otherwise, level is utilized.
	NvU16 rpmValue - [out/in] Tachometer speed in RPM
	
	<OR>
	
	NvU16 level - [out/in] Fan PWM setting in the range of 0 - 100. 0 = no rotation, 100 = full speed.
	}

Asynchronous Request Status Codes

The following table lists the status codes returned by the GPU for asynchronous requests.

Table 3-10. Asynchronous Request Status Codes

Code	Status
0x00	ASYNC_REQ_STATUS_SUCCESS
0x01	ASYNC_REQ_STATUS_ERROR_CARD_NOT_PRESENT
0x02	ASYNC_REQ_STATUS_ERROR_DUAL_LINK_INUSE
0x03	ASYNC_REQ_STATUS_ERROR_GENERIC
0x04	ASYNC_REQ_STATUS_ERROR_GPU_NOT_FULL_POWER
0x05	ASYNC_REQ_STATUS_ERROR_IN_USE
0x06	ASYNC_REQ_STATUS_ERROR_INSUFFICIENT_RESOURCES
0x07	ASYNC_REQ_STATUS_ERROR_INVALID_ACCESS_TYPE
0x08	ASYNC_REQ_STATUS_ERROR_INVALID_ARGUMENT
0x09	ASYNC_REQ_STATUS_ERROR_INVALID_BASE
0x0A	ASYNC_REQ_STATUS_ERROR_INVALID_CHANNEL
0x0B	ASYNC_REQ_STATUS_ERROR_INVALID_CLASS
0x0C	ASYNC_REQ_STATUS_ERROR_INVALID_CLIENT
0x0D	ASYNC_REQ_STATUS_ERROR_INVALID_COMMAND
0x0E	ASYNC_REQ_STATUS_ERROR_INVALID_DATA
0x0F	ASYNC_REQ_STATUS_ERROR_INVALID_DEVICE
0x10	ASYNC_REQ_STATUS_ERROR_INVALID_DMA_SPECIFIER
0x11	ASYNC_REQ_STATUS_ERROR_INVALID_EVENT
0x12	ASYNC_REQ_STATUS_ERROR_INVALID_FLAGS
0x13	ASYNC_REQ_STATUS_ERROR_INVALID_FUNCTION
0x14	ASYNC_REQ_STATUS_ERROR_INVALID_HEAP
0x15	ASYNC_REQ_STATUS_ERROR_INVALID_INDEX
0x16	ASYNC_REQ_STATUS_ERROR_INVALID_LIMIT
0x17	ASYNC_REQ_STATUS_ERROR_INVALID_METHOD
0x18	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_BUFFER
0x19	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_ERROR
0x1A	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_HANDLE
0x1B	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_NEW
0x1C	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_OLD
0x1D	ASYNC_REQ_STATUS_ERROR_INVALID_OBJECT_PARENT
0x1E	ASYNC_REQ_STATUS_ERROR_INVALID_OFFSET
0x1F	ASYNC_REQ_STATUS_ERROR_INVALID_OWNER

Table 3-10. Asynchronous Request Status Codes

Code	Status
0x20	ASYNC_REQ_STATUS_ERROR_INVALID_PARAM_STRUCT
0x21	ASYNC_REQ_STATUS_ERROR_INVALID_PARAMETER
0x22	ASYNC_REQ_STATUS_ERROR_INVALID_POINTER
0x23	ASYNC_REQ_STATUS_ERROR_INVALID_REGISTRY_KEY
0x24	ASYNC_REQ_STATUS_ERROR_INVALID_STATE
0x25	ASYNC_REQ_STATUS_ERROR_INVALID_STRING_LENGTH
0x26	ASYNC_REQ_STATUS_ERROR_INVALID_XLATE
0x27	ASYNC_REQ_STATUS_ERROR_IRQ_NOT_FIRING
0x28	ASYNC_REQ_STATUS_ERROR_MULTIPLE_MEMORY_TYPES
0x29	ASYNC_REQ_STATUS_ERROR_NOT_SUPPORTED
0x2A	ASYNC_REQ_STATUS_ERROR_OPERATING_SYSTEM
0x2B	ASYNC_REQ_STATUS_ERROR_PROTECTION_FAULT
0x2C	ASYNC_REQ_STATUS_ERROR_TIMEOUT
0x2D	ASYNC_REQ_STATUS_ERROR_TOO_MANY_PRIMARIES
0x2E	ASYNC_REQ_STATUS_ERROR_IRQ_EDGE_TRIGGERED
0x2F	ASYNC_REQ_STATUS_ERROR_INVALID_OPERATION
0x30	ASYNC_REQ_STATUS_ERROR_NOT_COMPATIBLE
0x31	ASYNC_REQ_STATUS_ERROR_MORE_PROCESSING_REQUIRED
0x32	ASYNC_REQ_STATUS_ERROR_INSUFFICIENT_PERMISSIONS
0x33	ASYNC_REQ_STATUS_ERROR_TIMEOUT_RETRY
0x34	ASYNC_REQ_STATUS_ERROR_NOT_READY
0x35	ASYNC_REQ_STATUS_ERROR_GPU_IS_LOST
0x36	ASYNC_REQ_STATUS_ERROR_IN_FULLCHIP_RESET
0x37	ASYNC_REQ_STATUS_ERROR_INVALID_LOCK_STATE
0x38	ASYNC_REQ_STATUS_ERROR_INVALID_ADDRESS
0x39	ASYNC_REQ_STATUS_ERROR_INVALID_IRQ_LEVEL
0x40	ASYNC_REQ_STATUS_ERROR_MEMORY_TRAINING_FAILED
0x41	ASYNC_REQ_STATUS_ERROR_BUSY_RETRY
0x42	ASYNC_REQ_STATUS_ERROR_INSUFFICIENT_POWER
0x43	ASYNC_REQ_STATUS_ERROR_OBJECT_NOT_FOUND
0x44	ASYNC_REQ_STATUS_ERROR_BUFFER_TOO_SMALL
0x45	ASYNC_REQ_STATUS_ERROR_RESET_REQUIRED
0x47	ASYNC_REQ_STATUS_REQUEST_DEFERRED

3.14 11h - Access Internal State Registers

The action performed with this opcode is dependent on Arg1 as follows:

- (Arg1 == 0x1) reads into the data register, or
- (Arg1 == 0x0) writes from the data register the contents of the internal state register according to the index specified in Arg2. (See Section 4.3 "Scratch Memory".)

Command

Opcode	11h – Access internal state registers
Arg1	0 - Write into the internal state register from the data register
	1 - Read internal state register into the data register
Arg2	The state register index
	0 - Scratch banks register 1 - Events pending register 2 - Event mask register

Data-In

Bit 31:0	(Arg1 = 0): Data to write
	(Arg1 = 1): Not defined

Data-Out

Bit 31:0	(Arg1 = 0): Not defined
	(Arg1 = 1): Data to read

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully read or written.
ERROR_NOT_SUPPORTED	Not supported in this configuration. (See capability word 2, bits 4:2)
ERR_ARG2	No such register.
ERR_MISC	DMA controller error.

3.15 12h - Check External Power

This opcode determines if a sufficient external power supply is connected to the board. This opcode is intended to report the presence status for the PCIe power connector on GPU PCIe card products without an onboard MCU.

Command

Opcode	12h – Check external power
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	0: Sufficient external power is connected
	1: Insufficient external power

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully retrieved.
ERROR_NOT_SUPPORTED	Not supported in this configuration.

3.16 13h - Read Dynamic Page Retirement Statistics

This opcode reads the count of pages that have been retired dynamically.

Command

Opcode	13h – Read dynamic page retirement statistics
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 7:0	The number of pages dynamically retired because of single-bit errors (SBE)
Bit 15:8	The number of pages dynamically retired because of double bit errors (DBE)

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Information was successfully retrieved.
ERROR_NOT_SUPPORTED	Information not present in the InfoROM.

3.17 14h - Query ECC Statistics - Format V4

Reads memory ECC statistics data, format version V4 for NVIDIA Pascal GPUs.

Command

Opcode	14h – Query ECC Statistics
Arg1	Encoding specific to query-type. See tables below for usage.
Arg2	Encoding specific to query-type. See tables below for usage.

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Encoding specific to request-type. See tables below for usage.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

Note: Before making this call check appropriate cap_dword(i) for support.

Query-Type 1: Querying Aggregate Counters

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:6	00h selects aggregate counters
DEVICE	5:4	00h selects the frame buffer (FB) 01h selects the graphics device (GR)
SOURCE	3:2	00h, device being FB, selects level 2 cache (LTC) 01h, device being FB, selects the frame buffer (FB) 00h, device being GR, selects shared memory (SHM) 01h, device being GR, selects SM register file (RF) 02h, device being GR, select texture units (TEX)
TYPE	1:0	00h selects the single-bit error counter 01h selects the double-bit error counter)

Arg2 Encoding (Device-Specific):

DEVICE == FB

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-15
SLICE/ SUBPARTITION	3:0	Slice number in the range 0-1 (when SOURCE selected is LTC (00h)) Subpartition number in the range 0-1 (when SOURCE selected is FB (01h))

DEVICE == GR

Field Name	Position	Value & Description
TEX	7:6	Texture unit number in the range 0-1 (when SOURCE selected is TEX (02h))
GPC	5:3	GPC number in the range 0-5
TPC	2:0	TPC number in the range 0-4

Query-Type 2: Querying Frame Buffer Individual Addresses Statistics

The driver gathers ECC data for up to 16 individual addresses per partition/subpartition pair. Below are the Arg1/Arg2 layouts that facilitate access to this information.

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:6	01h selects per partition/slice frame buffer statistics
MBZ	5:5	00h must be zero
ADDR/CNT	4:4	00h selects DBE/SBE counters 01h selects the address value itself
INDEX	3:0	Index into the array of addresses in the range 0-15

Arg2 Encoding:

Field Name	Position	Value & Description
PARTITION	7:4	Partition number in the range 0-15
SUBPARTITION	3:0	Subpartition number in the range 0-1

Result

The query result appears in the Data register and has the following layout:

Field Name	Position	Value & Description
DBE	23:16	cumulative double-bit errors
SBE	7:0	cumulative single-bit errors

3.18 15h - Read Thermal Parameters

Reads the thermal parameters specified for the GPU. The NVIDIA driver must be loaded in persistence mode in order to make this query.

Note: Before making this call check appropriate cap_dword(0) for support.

Note: This opcode can function without the driver loaded for Volta GPUs and later. The driver does not need to be previously loaded or loaded at the time of the query.

Command

Opcode 15h - Read thermal parameters

Arg1 00h - GPU target temperature

01h - Minimum GPU hardware slowdown temperature

02h - GPU shutdown temperature

03h - Maximum memory operating temperature

04h - Maximum GPU operating temperature

Arg2 Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 (Arg1 = 00h): GPU target temperature (in degrees Celsius)
(Arg1 = 01h): Minimum GPU hardware slowdown temperature
(Arg1 = 02h): GPU shutdown temperature
(Arg1 = 03h): Maximum memory operating temperature
(Arg1 = 04h): Maximum GPU operating temperature

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	GPU target temperature was successfully retrieved.
ERROR_NOT_SUPPORTED	Requested feature is not supported on given configuration.

3.19 16h - Query ECC Statistics - Format V5

Reads memory ECC statistics data, format version V5 for Volta and Turing GPUs. There is a maximum of 600 entries in the InfoROM which can be individually queried according to their index as specified in arg1 and arg2. Each of the entries are filled in sequentially with no gaps between any entries.

ELEMENT_TYPE = 0 should be used initially in order to determine whether an entry is valid and whether it is an address or region count before running further queries.

Note: There is a maximum of 600 entries in the InfoROM with no roll-over.

Command

Opcode	16h – Query ECC Statistics - Format V5
Arg1	Encoding specific to query-type. See tables below for usage.
Arg2	Encoding specific to query-type. See tables below for usage.

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Encoding specific to request-type. See tables below for usage.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

Query-Type 1: Querying Address and Region Counts

The entry index is evaluated as a 10-bit concatenation of 3-bit **INDEX_HI** and 7-bit **INDEX_LO**. Its value must be less than 600.

Arg1 Encoding:

Field Name	Position	Value & Description
SEL	7:7	00h selects aggregate counters
INDEX_LO	6:0	7 least significant index entry bits

Arg2 Encoding for address counts:

Field Name	Position	Value & Description
INDEX_HI	7:5	3 most significant index entry bits
ELEMENT_TYPE	2:0	00h header and metadata - used to decode the location the entry corresponds to. 01h address where the error occurred 02h uncorrected counts (DBE) Output is a combination of unique and total counts [15:0] Total counts [31:16] Unique counts 03h corrected total (SBE) 04h corrected unique

Arg2 Encoding for region counts:

Field Name	Position	Value & Description
INDEX_HI	7:5	3 most significant index entry bits
ELEMENT_TYPE	2:0	0x0 header and metadata - used to decode the location the entry corresponds to. 0x1 corrected total 0x2 corrected unique 0x3 uncorrected total 0x4 uncorrected unique

Query-Type 2: Querying SRAM Error Buffer

Arg1 Encoding:

The entry index must be less than 16,

Field Name	Position	Value & Description
SEL	7:7	0x1h
INDEX_LO	3:0	7 entry index

Arg2 Encoding:

Field Name	Position	Value & Description
ELEMENT_TYPE	1:0	0x0: error type and metadata 0x1: time stamp 0x2: address

Result for Query 1

The query 1 result appears in the Data register and has the following layout:

For ELEMENT_TYPE=0:

Field Name	Position	Value & Description
METADATA	31:16	Data out; split as follows: [31:27] Holds sub-location information Subpartition number for DRAM Slice number for LTC TPC number for LRF, L1DATA, L1TAG, CBU [26:22] Holds location information Partition number for DRAM/LTC GPC number for LRF, L1DATA, L1TAG, CBU [21:16] Holds the values between 0 to 5 for Volta 0 - LRF (Register file) 1 - L1DATA 2- L1TAG 3 - CBU 4 - L2 Cache 5 - DRAM
HEADER	1:0	Header Identifier 0x0: Invalid Entry 0x1: Address Entry 0x2: Region Entry

For all other ELEMENT_TYPE values:

Field Name	Position	Value & Description
Region Count	31:0	Counter value

Result for Query 2

The query 2 result appears in the Data register and has the following layout:

For ELEMENT_TYPE=0:

Field Name	Position	Value & Description

METADATA	31:16	Data out; split as follows: [31:27] Holds sub-location information Subpartition number for DRAM Slice number for LTC TPC number for LRF, L1DATA, L1TAG, CBU [26:22] Holds location information Partition number for DRAM/LTC GPC number for LRF, L1DATA, L1TAG, CBU [21:16] Holds the values between 0 to 5 for Volta 0 - LRF (Register file) 1 - L1DATA 2- L1TAG 3 - CBU 4 - L2 Cache 5 - DRAM
HEADER	1:0	Header Identifier 0x0: Invalid Entry 0x1: Address Entry 0x2: Region Entry

For all other ELEMENT_TYPE values:

Field Name	Position	Value & Description
Count	31:0	Counter value

Example Usage Case

This example usage case is for querying aggregate ECC errors (uncorrected and corrected) counts in device memory.

1. Determine the last valid error index.

   Use the ELEMENT_TYPE = 0x0 query to find the last valid index. An invalid entry will return ERR_TYPE = 0x0. All valid entries proceed the first invalid index.
2. Query all indexes from 0 to the last valid entry index discovered in (1) to find all DRAM entries.

   Use the ELEMENT_TYPE = 0x0 query to find the DRAM indexes. A DRAM index will return METADATA[21:16] = 0x5.

   Note the entry type (address vs region) present in the returned ERR_TYPE field:

   ERR_TYPE = 0x1 -> address entry

   ERR_TYPE = 0x2 -> region entry
3. For each DRAM index from (2), query the corrected total counts.

   For address entries use:

ELEMENT_TYPE = 0x3

For region entries use:

ELEMENT_TYPE = 0x1

4. For each DRAM index from (2), query the uncorrected total counts.

   For address entries use:

   ELEMENT_TYPE = 0x2

   Total counts will be in bits [15:0].

3.20 17h - Get/Set Write-Protect Mode

This opcode gets or sets write-protect mode for the GPU firmware. If the NVIDIA GPU has write-protect mode enabled, then this command is used in conjunction with in-band GPU firmware update. During normal operation, the write-protect mode should be enabled to prevent tampering or corruption of the GPU firmware. Write-protect mode is required to be disabled during an in-band GPU firmware update process to allow write access to the NVIDIA GPU EEPROM.

Note: The driver must be unloaded to perform this command, otherwise ERR_NOT_SUPPORTED will be returned.
The command is intended to disable write-protect mode when using NVFlash. Write-protect mode should be disabled prior to executing NVFlash and enabled after NVFlash has completed.

When using this command to enable a GPU in-band firmware update using NVFlash, the following is the recommended process:

1. Ensure that the Get/Set Write-Protect opcode is supported by the NVIDIA GPU by checking the GPU capabilities.

   Execute Get Capabilities (01h) and check DWORD[1] Bit 22, indicating whether the Get/Set Write protect feature is supported.
2. Ensure that the driver is unloaded, and disable write-protect mode (Arg1 = 1, Arg2 = 0x5A) to enable EEPROM write access.
3. Perform the in-band GPU firmware update using the NVFlash application.
4. When the in-band GPU firmware update has completed successfully, restore the write-protect mode (Arg1 = 1, Arg2 = 0xA5).

Command

Opcode	17h – Get/Set Write-Protect Mode
Arg1	0 – Get Write-Protect Mode 1 – Set Write-Protect Mode
Arg2	Unused if Arg1 = 0 If Arg1 = 1,

0x5A – Disable Write-Protect Mode
0xA5 – Enable Write-Protect Mode

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 (Arg1 = 0):
0x5A – Write-Protect Mode Disabled
0xA5 – Write-Protect Mode Enabled

(Arg1 = 1): Unused

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Action requested is invalid or undefined.
ERR_ARG2	Invalid value given.
ERR_NOT_SUPPORTED	Set action not currently supported because the driver is loaded.

3.21 18h - Query GPU State Flags

This opcode is used to query miscellaneous GPU state flags such as ECC status and GPU reset required. Before executing this opcode, check the respective capability DWORD to ensure this command is supported.

Command

Opcode 18h – Query GPU state flags

Arg1 Device state flags page

00h – Device state flags page 0
01h – Device state flags page 1

Arg2 Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 If Arg1 = 00h: Device state flags page 0

Bit 0: 0 - ECC cannot be enabled/disabled on this board

1 - ECC can be enabled/disabled on this board

Bit 1: Current ECC State

0 - Disabled

1 - Enabled

Bit 2: Pending ECC State After Reset

0 - Disabled

1 - Enabled

Bit 3: Multi-Instance GPU (MIG) Capability

0 - MIG cannot be enabled/disabled on this board

1 - MIG can be enabled/disabled on this board

Bit 4: Current MIG state

0 - Disabled

1 - Enabled

Bit 5: MIG state change pending after reset

0 - Disabled

1 - Enabled

Bit [31:6]: Reserved

If Arg1 = 01h: Device state flags page 1

Bit 0: 0 - GPU reset is not required

1 - GPU reset is required

Note: the Event Flag will also be asserted if a GPU reset is required.

Bit 1: 0 - Drain and reset is not required

1 - Drain and reset is recommended

Bit [31:2]: Reserved

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Action requested is invalid or undefined.
ERR_NOT_SUPPORTED	Request not supported by the SMBPBI server

3.22 19h - Get Accumulated GPU Context/ SM Utilization Time

This opcode is used to get the accumulated GPU context utilization or SM utilization time in milliseconds. Before executing this opcode, check the respective capability DWORD to ensure this command is supported.

Note: GPU context or SM utilization time can be retrieved with or without the driver loaded. GPU context and SM utilization time counters can accumulate only with the driver loaded.

Utilization counters are reset when an SBR or fundamental reset is received by the GPU. Accumulation registers are 32-bits in size and can cover approximately 49 days of activity. There is no overflow prevention.

Command

Opcode	19h – Get accumulated GPU context or SM utilization time
Arg1	00h – Get accumulated context utilization time (ms) 01h – Get accumulated SM utilization time (ms) FFh – Clear accumulated context and SM utilization counters to zero.
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	If Arg1 = 00h: Accumulated context utilization time (ms) If Arg1 = 01h: Accumulated SM utilization time (ms) If Arg1 = FFh: Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Action requested is invalid or undefined.
ERR_NOT_SUPPORTED	Request not supported.

3.23 1Ah - Query NVLink Information

This opcode is used to query various NVLink related information such as NVLink status, speed, and error counts. Before executing this opcode, check the respective capability DWORDs to ensure this command is supported.

Note: Arg1 = 0Fh (Query availability of NVLinks) can be used first to determine if the specific product supports the NVLink interface prior to querying other NVLink related information.

Command

Opcode 1Ah - Query NVLink Information

Arg1
00h - Number of NVLinks
01h - NVLink status (Format V1)
02h - NVLink bandwidth
03h - NVLink replay error count
04h - NVLink recovery error count
05h - NVLink flit CRC error count
06h - NVLink data CRC error count
07h - NVLink status (Format V2)
08h - Query NVLink sublink width
09h - Query transmitted NVLink data throughput
0Ah - Query received NVLink data throughput
0Bh - Query transmitted NVLink data throughput with protocol overhead
0Ch - Query received NVLink data throughput with protocol overhead
0Fh - Query availability of NVLinks

Arg2
If Arg1 = 00h: Unused
If Arg1 = 01h: NVLink page index.
Use the page index to select which links to report. Each page will contain 32 links. The first 32 links will be outputted in the data register. The next 32 links will be outputted in the extended data register.

NVLink page index = 0xFF will provide aggregate state of all links. The output will be the UP/DOWN state if all link states are the same. If the link states are not all the same, ERR_NOT_AVAILABLE will be returned.

If Arg1 = 02h: NVLink page index.
Use the page index to select which links to report. Each page will contain 2 links. Links (Arg2)*2 will be outputted in the data-out register. Links (Arg2)*2+1 will be outputted in the extended Data register.

NVLink page index = 0xFF will provide aggregate bandwidth of all links. Output will be the bandwidth if all link bandwidths are the same. If not all link bandwidths are the same, ERR_NOT_AVAILABLE will be returned.

If Arg1 = 03h - 06h: NVLink Index.
NVLink Index = 0xFF will provide aggregate error count of all links

If Arg1 = 07h: NVLink page index.
Use the page index to select which links to report. Links (Arg2*16) to (Arg2*16)+7 will be encoded in the data-out register. Links (Arg2*16)+8 to (Arg2*16)+15 will be encoded in the extData register.

Invalid link indices (beyond the number of links) will be in the OFF_STATE.

NVLink page index = 0xFF will provide aggregate status of all links. The output will be the status if all link statuses are the same. If not all link statuses are the same, ERR_NOT_AVAILABLE will be returned.

If Arg1 = 08h: NVLink page index.
Use the page index to select which links to report. Links (Arg2*10) to (Arg2*10)+4 will be encoded in the data-out register. Links (Arg2*10)+5 to (Arg2*10)+9 will be encoded in the extData register.

Invalid link indices (beyond the number of links) will contain sublink widths of 0.

NVLink page index = 0xFF will provide the aggregate sublink state of all links. Output will be the sublink width if all widths are the same. If not all link status are the same, ERR_NOT_AVAILABLE will be returned.

If Arg1 = 09h - 0Ch: NVLink index for per link query.
NVLink Index = 0xFF will provide aggregate values across all links

If Arg1 = 0Fh: NVLink page index.

Use the page index to select which links to report. Each page will contain 32 links. The first 32 links will be outputted in the data register. The next 32 links will be outputted in the extended data register.

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0

If Arg1 = 00h: Total number of NVLinks
If Arg1 = 01h: Bitmap of the first 32 NVLink states, where each bit encodes the state of the corresponding NVLink
0 - NVLink is down
1 - NVLink is up; or the state of all links, if the aggregate command is used
If Arg1 = 02h: NVLink bandwidth. A100 (and beyond) will report bandwidth in MB/s (Megabytes per second). Volta and Turing will report bandwidth in Mbps (Megabits per second).

A100 example:

Return value = 0x61A8 = 25,000 MB/s

Expected return value:
50Gbps * (4 lanes width) * (1 Byte / 8 bits) = 25 GB/s = 25,000 MB/s

If Arg1 = 03h: Replay error count for the provided NVLink Index in Arg2
If Arg1 = 04h: Recovery error count for the provided NVLink Index in Arg2
If Arg1 = 05h: Flit CRC error count for the provided NVLink Index in Arg2
If Arg1 = 06h: Data CRC error count for the provided NVLink Index in Arg2
If Arg1 = 07h: Report up to a total of 8 NVLink states, each encoded in 4 bits. This can be indexed by this bit mapping: (3+(i)*4):(0+(i)*4)

0x0 OFF_STATE - NVLink is disabled
0x1 SAFE_STATE - NVLink in training state
0x2 ACTIVE_STATE - NVLink is enabled
0x3 ERROR_STATE - NVLink has a fatal error

The data will be returned by the bit mapping above for links (Arg2*16) to (Arg2*16)+7, or as the aggregate state of all links.

NOTE: Reported NVLink states are dynamic and can change with each query.

If Arg1 = 08h: Report up to a total of 5 NVLink sublink widths, RX and TX, encoded in 6-bits. The sub links can be indexed by this bit mapping: (5+(i)*6):(0+(i)*6).

Each group of 6 bits have the following format:

[2:0] TX Sublink Width
[5:3] RX Sublink Width

Values:

0x0 WIDTH_0
0x1 WIDTH_1
0x2 WIDTH_2
0x3 WIDTH_4
0x4 WIDTH_8
0x5 Reserved

0x6 Reserved
0x7 WIDTH_INVALID

The data will be returned by the bit mapping above for links (Arg2*10) to (Arg2*10)+4, or as the aggregate sublink widths of all links.

If Arg1 = 09h - 0Ch: Lower 32 bits of throughput counter value in KiB

If Arg1 = 0Fh: Bitmap of the first 32 NVLinks of selected NVLink page index where each bit represents the availability.

0 - Not available

1 - Available

Extended Data-Out

Bit 31:0

If Arg1 = 00h: Unused

If Arg1 = 01h: Bitmap of the next 32 NVLink states
0 - NVLink is down
1 - NVLink is up

If Arg1 = 02h: NVLink bandwidth for NVLinks Arg2*2+1.
A100 (and beyond) will report bandwidth in MB/s (Megabytes per second). Volta and Turing will report bandwidth in Mbps (Megabits per second).

If Arg1 = 03h - 06h: Unused

If Arg1 = 07h: NVLink states for links (Arg2*16)+8 to (Arg2*16)+15

If Arg1 = 08h: NVLink sublink width for links (Arg2*10)+5 to (Arg2*10)+9.

If Arg1 = 09h - 0Ch: Upper 32 bits of throughput counter value in KiB

If Arg1 = 0Fh: Bitmap of the next 32 NVLinks of selected NVLink page index where each bit represents the availability.
0 - Not available
1 - Available

Additional Status and Data (in Status Register):

If Arg1 = 00h - 08h: Unused

If Arg1 = 09h - 0Ch: Traffic counted since last query

Format:

Bit 23:4 CounterDelta - The amount of traffic since last query in units specified in Granularity bits.
This value is set to 0 if DeltaInvalid is set to 1.

Bit 3:3 DeltaInvalid

0 - Indicates counter delta in bits 23:4 is valid

1 - Indicates counter delta in bits 23:4 is invalid[1]

Bit 2:2 NewSample

0 - Indicates counter value was read from an existing sample
1 - Indicates counter value was read from a newly created sample

Bit 1:0 Granularity

0 - Indicates data traffic since last query in KiB
1 - Indicates data traffic since last query in MiB
2 - Indicates data traffic since last query in GiB
3 - Indicates data traffic since last query in TiB

- If DeltaInvalid field is set, Granularity, NewSample and CounterDelta fields are undefined.
- Data-out and Ext-Data will still hold absolute counter values in KiB.
- The CounterDelta field is not supported when the driver is unloaded. Counter delta is a future enhancement for the driver loaded case. For these cases, DeltaInvalid field will be set to 1 to indicate that CounterDelta is invalid.

Status

SUCCESS	Command completed successfully.
ERR_NOT_AVAILABLE	Requested data has mismatching status between links.
ERR_NOT_SUPPORTED	Request not supported.
ERR_MISC	Something internal went wrong

3.24 1Bh - Query Clock Frequency Info

This opcode is used to query various clock frequency related information. Before executing this opcode, check the respective capability DWORD to ensure this command is supported.

Command

Opcode	1Bh - Query Clock Frequency Information
Arg1	Clock parameter
	00h - Current Clock Frequency
	01h - Minimum Programmable Clock

1. Counter delta support will be added at a later date and 'DeltaInvalid[3:3]' bit will always return '1'.

02h - Maximum Programmable Clock

03h - Performance State

Arg2 Clock type

If Arg1 = 00h - 02h:

00h - Graphics Clock

01h - Memory Clock

If Arg1 = 03h: Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0

If Arg1 = 00h: Current clock frequency (in KHz)

If Arg1 = 01h: Minimum programmable clock (in KHz)

If Arg1 = 02h: Maximum programmable clock (in KHz)

If Arg1 = 03h: Bits [3:0] - Performance state

Bits [31:4] - Reserved

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Command completed successfully.
ERR_MISC	Internal error trying to fetch clock values
ERR_ARG1	Arg1 input value is invalid
ERR_ARG2	Arg2 input value is invalid
ERR_NOT_SUPPORTED	Request not supported

3.25 1Ch - Kick off a Request Bundle

This opcode allows the SMBPBI master to kick off multiple requests, called bundles, at once. Before executing this opcode, check the respective capability DWORD to ensure this command is supported for your respective GPU product.

Note: Before executing this opcode for the first time, the request bundle must first be defined.

A bundle is defined in scratch memory and consists of 2 elements:

- An array of structures defining the individual requests
- An optional array of Result Disposition Rules

A bundle kick-off request must specify the following arguments:

- A pointer to the bundle definition in scratch memory
- Number of individual requests in the bundle
- Number of Result Disposition Rules, which can be zero (i.e. "optional").

Note: The maximum number of individual requests in a bundle is 4 and the maximum number of result disposition rules is 10.

Each individual SMBPBI request has the following structure:

```
struct {
          NvU32    cmdStatus;
          NvU32    dataIn;
          NvU32    dataOut;
          NvU32    extDataOut;
}
```

NvU32 cmdStatus plays two roles:

1. Provides Command Register encoding of the individual request.

   Bit[31]: Stop bit - a conditional stop bit: do not execute the subsequent individual requests in the bundle, unless the current individual request completes successfully.

   Bit[30:29]: MBZ - must be set to zero by SMBPBI master

   Bit[28:24]: Unused - field will be cleared by SMBPBI slave at bundle kick off

   Bit[23:16]: Arg2 - second argument, usage dependent on opcode

   Bit[15:8]: Arg1 - first argument, usage dependent on opcode

   Bit[7:0]: Opcode - request operation opcode
2. Provides Status Register holding the status of the completed individual request

Bit[31:29]: Retained - these fields are not modified by the SMBPBI slave and the values of the corresponding bit fields in the Command Register are kept.

Bit[28:24]: Status - the value in this field characterizes the result of the request execution by the server. If an individual request has not been executed because of the stop bit, this field will retain the NULL status value.

Bit[23:0]: Retained - these fields are not modified by the SMBPBI slave and the values of the corresponding bit fields in the Command Register are kept.

NvU32 dataIn holds any additional request arguments which is opcode dependent

Bit[31:0]: Data in

NvU32 dataOut contains the requested data upon the completion of the individual request

Bit[31:0]: Data out bits [31:0]

NvU32 extDataOut contains the additional requested data upon the completion of the individual request that returns more than 32 bits of data.

Bit[31:0]: Data out bits [63:32]

Result Disposition Rules

Result Disposition Rules form an array of 32-bit words up to a maximum of 10, which is also implementation dependent. Each word contains a rule requesting a copy of one of the individual request's output data registers (DATA, EXT_DATA) into one of the SMBPBI output registers (STATUS, DATA, EXT_DATA).

A rule word has the following layout:

- Bit[21:17]: dst_right_bit - the rightmost bit index (0-31) of the destination register bit range
- Bit[16:15]: dst_reg_id - destination register index
  - 0: STATUS (bits 0-23 only are available)
  - 1: DATA
  - 2: EXT_DATA
- Bit [14:10]: bit_range_width - the width of the bit range that is being copied (n-1 i.e. range = 0-31)
- Bit[9:5]: src_right_bit - the rightmost bit index (0-31) of the source bit range that is being copied
- Bit[4:3] src_reg_id - source register index
  - 0: Not valid (i.e. reserved)
  - 1: DATA
  - 2: EXT_DATA
- Bit[2:0] req_index - the index of the individual request in the bundle, whose register is being copied.

Rules will be checked for correctness before any individual requests are executed. In case that any rule is failing the check, the entire bundle will be failed with the ERR_DISPOSITION, and the index of the failing rule will be left in the "additional status" field of the Status Register.

Bundle Completion Status

Upon the successful bundle completion, the SMBPBI server will:

- Post SUCCESS status into the SMBPBI Command/Status register, if and only if all the bundle elements have completed successfully.
- Post PARTIAL_FAILURE status into the SMBPBI Command/Status register in case at least one of the bundle elements has failed. The SMBPBI master will have to examine the individual statuses in the scratch memory.
- Fill the Data Out and the Extended Data registers in the scratch space for those requests that have completed successfully.
- If the Result Disposition Rules have not been provided, a standard disposition is performed in the following way:

  The server copies the Data Out register values of the initial 3 elementary requests into the following SMBPBI registers:

  - Command/Status[23:0], the lowest Data Out bytes from each of the 3 requests.
  - Data[31:0] 2 bytes from the 1st request, one each from the 2nd and the 3rd.
  - Ext Data[31:0] one byte from the first request, 2 bytes from the second request, 1 byte from the 3rd request.

Command

Opcode	1Ch - Kick off a Request Bundle
Arg1	Bit 3:0 - Number of individual requests in a bundle Bit 7:4 - Number of Result Disposition Rules
Arg2	Bit 7:0 - Pointer to the bundle definition in scratch memory

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Combined output of individual requests, as defined in the Result Disposition Rules

Extended Data-Out

Bit 31:0	Combined output of individual requests, as defined in the Result Disposition Rules

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Arg1 input value is invalid
ERR_ARG2	Arg2 input value is invalid

ERR_NOT_SUPPORTED	Request not supported

Bundling Usage Example

The following details provide a bundling usage example of the following SMBPBI commands:

- Get GPU temperature (Opcode = 02h, Arg1/Arg2 = 00h)
- Get HBM temperature (Opcode = 02h, Arg1 = 05h, Arg2 = 00h)
- Get GPU power (Opcode = 04h, Arg1 = 00h, Arg2 = 00h)
- Query current clock frequency (Opcode = 1Bh, Arg1 = 00h, Arg2 = 00h)

1. First, set up scratch memory before kicking off the bundling request.
   a. Perform scratch memory write of 0x80000002 at offset 0x00 - SMBPBI Op 0x0E, Arg1 = 0x00, Arg2 = 0x00, DataIn = 0x80000002 (0x80 | Arg2 = 00 | Arg1 = 00 | Opcode = 02)
   b. Perform scratch memory write of 0x80000502 at offset 0x04 - SMBPBI Op 0x0E, Arg1 = 0x04, Arg2 = 0x00, DataIn = 0x80000502
   c. Perform scratch memory write of 0x80000004 at offset 0x08 - SMBPBI Op 0x0E, Arg1 = 0x08, Arg2 = 0x00, DataIn = 0x80000004
   d. Perform scratch memory write of 0x00001908 at offset 0x10 - SMBPBI Op 0x0E, Arg1 = 0x10, Arg2 = 0x00, DataIn = 0x00001908
   e. Perform scratch memory write of 0x000e1909 at offset 0x11 - SMBPBI Op 0x0E, Arg1 = 0x11, Arg2 = 0x00, DataIn = 0x000e1909
   f. Perform scratch memory write of 0x0000ac0a at offset 0x12 - SMBPBI Op 0x0E, Arg1 = 0x12, Arg2 = 0x00, DataIn = 0x0000ac0a
   g. Perform scratch memory write of 0x0018cc0b at offset 0x13 - SMBPBI Op 0x0E, Arg1 = 0x13, Arg2 = 0x00, DataIn = 0x0018cc0b

   These opcode calls will set up scratch memory with the following layout:

Location	Value
0x00	0x80000002
0x04	0x80000502
0x08	0x80000004
0x0c	0x8000001b
0x10	0x00001908 (*)
0x11	0x000e1909 (**)
0x12	0x0000ac0a (***)
0x13	0x0018cc0b (****)

2. Kick off the bundle request - SMBPBI Opcode 0x1C, Arg1 = 0x04, Arg2 = 0x04

   This opcode call means that the bundle pointer is located at 0x00 and has 4 individual requests (at 0x00) with 4 disposition rules (at 0x10).

   Upon completion:

- The Status Register will have 7 bits for each of the 2 temperatures.
- The Data Register will have 12 bits for the power in mW and 20 bits for the clock in kHz.

Disposition Rules

```
(*) location 0x10 = 0x00001908
 +----------+-------+---------+----------+-------+-------+
 |dst_rt_bit|dst_reg|src_width|src_rt_bit|src_reg|req_idx|
 +----------+-------+---------+----------+-------+-------+
 | 0 0 0 0 0|  0 0  |0 0 1 1 0| 0 1 0 0 0|  0 1  | 0 0 0 |
 +----+---------+---------+--------+---------+-----------+

(**) location 0x11 = 0x000e1909
 +----------+-------+---------+----------+-------+-------+
 |dst_rt_bit|dst_reg|src_width|src_rt_bit|src_reg|req_idx|
 +----------+-------+---------+----------+-------+-------+
 | 0 0 1 1 1|  0 0  |0 0 1 1 0| 0 1 0 0 0|  0 1  | 0 0 1 |
 +----+---------+---------+--------+---------+-----------+

(***) location 0x12 = 0x0000ac0a
 +----------+-------+---------+----------+-------+-------+
 |dst_rt_bit|dst_reg|src_width|src_rt_bit|src_reg|req_idx|
 +----------+-------+---------+----------+-------+-------+
 | 0 0 0 0 0|  0 1  |0 1 0 1 1| 0 0 0 0 0|  0 1  | 0 1 0 |
 +----+---------+---------+--------+---------+-----------+

(****) location 0x13 = 0x0018cc0b
 +----------+-------+---------+----------+-------+-------+
 |dst_rt_bit|dst_reg|src_width|src_rt_bit|src_reg|req_idx|
 +----------+-------+---------+----------+-------+-------+
 | 0 1 1 0 0|  0 1  |1 0 0 1 1| 0 0 0 0 0|  0 1  | 0 1 1 |
 +----+---------+---------+--------+---------+-----------+
```

3.26 1DH - Request the Oldest Driver Event Message (DEM) from the Server Buffer

Retrieves the oldest driver event message (DEM) from the server buffer to place into a scratch memory location.

When executed, scratch memory at the location pointed to by Arg2 will be filled with the DEM recover of the following structure:

```
struct {
        NvU8    recordSize; // in 32-bit words
        NvU8    xidId;
        NvU8    flags;  // bit 0: some DEMs after this one have been lost
                        // bit 1: the text message has been truncated
        NvU32   seqNumber;
        NvU32   timeStamp;  // seconds since the epoch UTC
```

```
        NvU8    textMessage[0]; // NULL terminated
}
```

The maximum length of the text message is 80 including the terminating NULL character.

The associated Event Flag bit will be set upon receipt of a pending DEM. Since this opcode has an associated Event Flag bit, there is no need to poll for execution status.

Command

Opcode	1Dh - Request the oldest driver event message (DEM) from the server buffer
Arg1	0x00
Arg2	Pointer to the scratch memory location where the DEM will be written

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Unused. Result is provided in the scratch memory location pointed by Arg2.

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Arg1 input value is invalid
ERR_ARG2	Indicates an incorrect scratch memory pointer
ERR_NOT_SUPPORTED	Request not supported
ERR_NOT_AVAILABLE	Indicates there are no DEMs in the buffer

3.27 1Eh - Request ECC Statistics (Format V6)

Retrieve ECC statistics in Format V6.

Command

Opcode	1Eh - Request ECC statistics (Format V6)
Arg1	Error Type
00h	Correctable Errors
01h	Uncorrectable Errors
Arg2	Counter Type
00h	SRAM Counter (internal GPU memory)
01h	DRAM Counter (frame buffer memory)

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Lower 32 bits of requested error counter value

Extended Data-Out

Bit 31:0	Upper 32 bits of requested error counter value

Status Register

Refer to the "Result Size Encoding" section when 'Copy' bit is set with the request

Status

SUCCESS	Command completed successfully.
ERR_MISC	Internal error
ERR_ARG1	Arg1 input value is invalid
ERR_NOT_SUPPORTED	Request not supported

3.28 20h - Request Row-remapping Statistics

Retrieve row-remapping statistics such as raw counts, state flags and bank remapping availability histogram.

Over a GPU's lifetime, the bank remapping availability histogram will update based on the availability of reserved rows and transitions in the following manner:

- Max -> High -> Partial -> Low -> None
- Max: entire set of reserved rows are available for remapping
- None: there are no remaining reserved rows available for remapping in a bank

Command

Opcode	20h - Request row-remapping statistics
Arg1	Query type 00h - Raw Counts 01h - State Flags 02h[2] - Bank remapping availability histogram
Arg2	If Arg1 = 00h 00h - Combined counts (correctable and uncorrectable remappings) 01h - Remapping count due to uncorrectable errors 02h - Remapping count due to correctable errors

2. Capability DWORD[2] Bit 21 indicates if the 02h query is supported. Copy-bit should not be set for this query.

If Arg1 = 01h
00h - Page 0
If Arg1 = 02h: Unused

Data-In

Bit 31:0 Unused

Data-Out

If Arg1 = 00h & Arg2 = 00h: Combined counts
Bit 10:0 Number of uncorrectable remappings
Bit 11:11 Bit indicating that number of uncorrectable remappings exceeds bits[10:0] and client should request Arg2 = 0x01 for an accurate count
Bit 22:12 Number of correctable remappings
Bit 23:23 Bit indicating that number of correctable remappings exceeds bits[22:12] and client should request Arg2 = 0x02 for an accurate count

If Arg1 = 00h & Arg1 = 01h:
Bit 31:0 Number of remappings due to uncorrectable errors

If Arg1 = 00h & Arg1 = 02h:
Bit 31:0 Number of remappings due to correctable errors

If Arg1 = 01h & Arg2 = 00h: State Flags Page 0
Bit 0:0 - 0: row-remapping has never failed in the past
1: row-remapping has failed in the past
Bit 1:1[3] - 0: no row-remapping is pending
1: row-remapping is pending

If Arg1 = 02h: Bank remapping availability histogram
Bit[15:0] – Number of banks with Low remapping availability
Bit[31:16] – Number of banks with Partial remapping availability

Extended Data-Out

If Arg1 = 02h: Bank remapping availability histogram.
Bit[15:0] – Number of banks with High remapping availability
Bit[31:16] – Number of banks with Max remapping availability
Else Bit[31:0] – Unused

Status

If Arg1 = 02h: Bank remapping availability histogram
Bit[15:0] - Number of banks with No remapping availability

SUCCESS Command completed successfully.

ERR_MISC Internal error

3. Capability bit DWORD[2] Bit 20 indicates if Bit 1 is supported.

ERR_ARG1	Arg1 input value is invalid
ERR_ARG2	Arg2 input value is invalid
ERR_NOT_SUPPORTED	Request not supported

3.29 21h - Query PCIe Link Status and Error Counts

Retrieve the current PCI Express link status and error count information.

Command

Opcode	21h - Query PCIe link status and error counts
Arg1	Page Index
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

If Arg1 = 00h: Page 0

Bit 2:0 - Link speed
- 0x0: unknown
- 0x1: 2500 MTPS (PCIe Gen 1.0)
- 0x2: 5000 MTPS (PCIe Gen 2.0)
- 0x3: 8000 MTPS (PCIe Gen 3.0)
- 0x4: 16000 MTPS (PCIe Gen 4.0)

Bit 6:4 - Link width
- 0x0: unknown
- 0x1: x1
- 0x2: x2
- 0x3: x4
- 0x4: x8
- 0x5: x16

Bit 15:8 - Non-fatal error count
The following errors get reported as non-fatal depending on the severity in AER registers:
- - Data link layer protocol error
- -Received poisoned TLP
- -Flow control protocol error
- -Completion timeout
- -Completion abort
- -Unexpected completion
- -Receiver overflow
- -Malformed TLP
- -ECRC error
- -Unsupported request

Bit 23:16 - Fatal error count
Following errors get reported as fatal depending on the severity in AER registers:
- -Data link layer protocol error

-Received poisoned TLP
-Flow control protocol error
-Completion timeout
-Completion abort
-Unexpected completion
-Receiver overflow
-Malformed TLP
-ECRC error
-Unsupported request

Bit 31:24 - Unsupported request count

If Arg1 = 01h: Page 1

Bit 31:0 - L0 to recovery count. This count reflects the number of times the link went into the recovery state from the L0 state.

If Arg1 = 02h: Page 2

Bit 15:0 - replay rollover count.
This count reflects the number of replay rollovers issued on the PCIe link by the GPU. A replay rollover occurs when four consecutive replays fail to resolve the errors on the link. It means that the GPU forced the link into recovery.

Bit 31:16 - NAKs received count.
This count reflects the number of NAKS issued on the PCIe link by the host root complex. A NAK is issued by the host root complex when it detects that a TLP from the GPU side was missed. This could be because the GPU did not transmit it, or because the host root complex receiver could not properly decode the packet.

If Arg1 = 03h: Page 3

Bit 2:0 - Requested link speed
0x0: unknown
0x1: 2500 MTPS (PCIe Gen 1.0)
0x2: 5000 MTPS (PCIe Gen 2.0)
0x3: 8000 MTPS (PCIe Gen 3.0)
0x4: 16000 MTPS (PCIe Gen 4.0)

Extended Data-Out

If Arg1 = 00h: Page 0
Bit 15:0 - Correctable error count.
The following errors get reported as correctable errors:
- Receiver error
- Bad TLP
- Bad DLLP
- Replay Rollover
- Replay timeout
- Advisory non-fatal error

If Arg1 = 01h: Page 1

Bit 31:0 - Replay Count.

This count reflects the number of replays issued on the PCIe link by the GPU device. A replay is a retransmission of a TLP and occurs because the ACK timer expired. This means that the host root complex did not send the ACK or the GPU receiver did not properly decode the ACK.

If Arg1 = 02h: Page 2

Bit 15:0 - NAKs sent count.

This count reflects the number of NAKS issued on the PCIe link by the GPU. A NAK is issued by the GPU when it detects that a TLP from the host root complex was missed. This could be because the host root complex did not transmit it, or because the GPU receiver could not properly decode the packet.

Status Register

Refer to the "Result Size Encoding" section when 'Copy' bit is set with the request

Status

SUCCESS	Command completed successfully.
ERR_MISC	Internal error
ERR_ARG1	Arg1 input value is invalid
ERR_NOT_SUPPORTED	Request not supported

3.30 22h - Query Energy Counter

Retrieve the energy counter value in Joules. This synchronous query is equivalent to the asynchronous query (Op 0x10, Arg1 0x08). Ensure the respective GPU device supports this query via capability DWORD[2] Bit 19.

Command

Opcode	22h - Query energy counter
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Lower 32-bits of energy counter value in Joules

Extended Data-Out

Bit 31:0	Upper 32-bits of energy counter value in Joules

Status

SUCCESS	Command completed successfully.
ERR_MISC	Internal error
ERR_NOT_SUPPORTED	Request not supported

3.31 F0h - Enable/Disable Power Supply

This opcode enables or disables the power supply, such as NVVDD, FBVDD, etc.

Note: This command cannot be sent ad hoc. This is a unique, zero-power command, and requires in-band software coordination. More detail can be found in the application note: *NVIDIA Zero Power*, DA-07143-001 (v02 and later).

Command

Opcode	F0h – Enable/disable the power supply
Arg1	0 – Disable 1 – Enable
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.32 F1h - Get Power Supply Status

This opcode gets the power supply status, such as NVVDD, FBVDD, etc.

Command

Opcode	F1 – Get the power supply status
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	00h – Power supply is disabled 01h – Power supply is enabled

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.33 F2h - Assert/Deassert PCIe Fundamental Reset State

This opcode asserts or deasserts the PCIe fundamental reset state.

Note: This command cannot be sent ad hoc. This is a unique, zero-power command, and requires in-band software coordination. More detail can be found in the application note: *NVIDIA Zero Power,* DA-07143-001 (v02 and later).

Command

Opcode	F2h – Assert/deassert PCIe fundamental reset state
Arg1	0 – deassert 1 – assert
Arg2	Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 Unused

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Command completed successfully.

3.34 F3h - Get PCIe Fundamental Reset State

This opcode gets the PCIe fundamental reset state – that is, whether it is asserted or deasserted.

Command

Opcode F3h – Get the PCIe fundamental reset state

Arg1 Unused

Arg2 Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 00h –the reset is deasserted
01h –the reset is asserted

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Command completed successfully.

3.35 F4h - Set/Release Thermal Alert

This opcode sets or releases the thermal alert for all GPUs controlled by the MCU or FPGA. This is the same action that occurs when power brake is asserted from the input pin to the MCU or FPGA. This opcode does not change the state of the power brake input.

Command

Opcode	F4h – Set/release the thermal alert
Arg1	0 – Release the thermal alert 1 – Set the thermal alert
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.36 F5h - Get Power Brake State

This opcode gets the power brake state–that is, whether it is released or set through the input pin to the MCU or FPGA. When the power brake is set, it results in the thermal alert being asserted to all GPUs.

Note: This command does not retrieve the thermal alert status of using opcode 0xF4 Set/Release Thermal Alert. Opcode 0xF6 Get Thermal Alert State should be used to check thermal alert status.

Command

Opcode	F5h – Get the power brake state
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	00h – Brake is released 01h – Brake is set (edge or header connector)

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.37 F6h - Get Thermal Alert State

This opcode gets the thermal alert state – that is, whether a thermal alert is pending or not. Thermal alert status for a specific GPU is selected via the SMBus slave address.

Command

Opcode	F6h – Get the thermal alert state
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	00h – there is no pending thermal alert 01h – a thermal alert is pending

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.38 F7h - Set Error LED State

This opcode sets the Error LED state.

Command

Opcode	F7h – Set the Error LED state.
Arg1	0 – LED off 1 – LED on
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Unused

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.39 F8h - Get Board Power Supply Status

This opcode gets the status of the board power supply. This opcode is intended to report the presence status for the PCIe power connector on GPU PCIe card products with an onboard MCU.

Command

Opcode	F8h – Get board power supply status
Arg1	Unused
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	00h – the board is under-powered 01h –the board has sufficient power

Extended Data-Out

Bit 31:0	Unused

Status

SUCCESS	Command completed successfully.

3.40 F9h - Assert Thermal Alert

Assert/de-assert the thermal alert of a single selected GPU via the SMBus slave address.

Command

Opcode	F9h – Assert thermal alert
Arg1	00h - De-assert thermal alert
	01h - Assert thermal alert
Arg2	Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 Unused

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Command completed successfully.

3.41 FAh - Get/Set MCU FW Write-Protect

This opcode gets or sets the MCU firmware write protection. Write protection is controlled by the Boot Mode Override Lock (BMOL) status in the MCU. This command must be used in conjunction to allow for firmware updates to the MCU via NVFlash.

Note: This command is intended to disable MCU firmware write protection when using NVFlash. Write protection should be disabled prior to executing NVFlash and enabled after NVFlash has completed.

When using this command to enable an MCU in-band firmware update using NVFlash, the following is the recommended process:

1. Ensure that the Get/Set MCU Firmware Write-Protect opcode is supported by the MCU by checking the SMBPBI capabilities.

   Execute Get Capabilities (01h) and check DWORD[3] Bit 10, indicating whether the Get/Set MCU Firmware Write-Protect feature is supported.
2. Disable MCU Firmware Write-Protect (Arg1 = 1, Arg2 = 0x5A) to enable MCU firmware updates.
3. Perform the in-band MCU firmware update using the NVFlash application.
4. When the in-band MCU firmware update has completed successfully, restore the MCU Firmware Write-Protect (Arg1 = 1, Arg2 = 0xA5).

Command

Opcode	FAh - Get/Set MCU Firmware Write-Protect	
Arg1	0h - Get MCU Firmware Write-Protect	
	1h - Set MCU Firmware Write-Protect	
Arg2	If Arg1 = 0:	Unused
	If Arg1 = 1:	

0x5A - Disable MCU Firmware Write-Protect

0xA5 - Enable MCU Firmware Write-Protect

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 If Arg1 = 0:

0x5A - MCU Firmware Write-Protect Disabled

0xA5 - MCU Firmware Write-Protect Enabled

If Arg1 = 1: Unused

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Command completed successfully.
ERR_ARG1	Arg1 input value is invalid
ERR_ARG2	Arg2 input value is invalid
ERR_NOT_SUPPORTED	Request not supported

3.42 FBh - Access MCU Scratch Registers

This opcode provides write or read access to MCU scratch registers where supported.

Command

Opcode FBh – Access MCU scratch registers

Arg1 00h - Write

01h - Read

Arg2 Register index (0x00 - 0x0F)

Data-In

If Arg1 = 00h:

Bit 31:0 Data to write to MCU scratch register

Else,

Bit 31:0 Unused

Data-Out

Bit 31:0 If Arg1 = 0: Unused

If Arg1 = 1:

Bit 31:0 Data read from MCU scratch register

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS	Command completed successfully.
ERR_NOT_SUPPORTED	Request not supported

Chapter 4. SMBus Interface Additional Details

4.1 GPU Slave Transaction Types

In the current implementation only two transaction types are supported by the GPU slave as defined in sec. 5.5.7 of the System Management Bus (SMBus) Specification Version 2.0:

- Block read
- Block write

For block reads the Byte Count returned by the GPU slave always equals 4. For block writes the client must always specify the Byte Count as 4. This, however, does not preclude the possibility that other transaction types will be used by future implementations of this interface.

4.2 Byte Ordering

4.2.1 Binary Values

Binary values, that are longer than one byte, are transmitted across the I2C bus in a little-endian order, that is, the least significant byte (LSB) first.

SMBus byte:	0	1	2	3
	[7:0]	[15:8]	[23:16]	[31:24]

Examples:

- Contents of the Command and Status Register.
- Contents of the Data Register, when a binary value is transmitted. For example, the Build date.

4.2.2 String Values

String values (sequences of ASCII characters) are transmitted across the I2C bus in their natural order.

Example:

- Contents of the Data Register, when a string value is transmitted. For example, the Marketing name.

4.3 Scratch Memory

The API implements a block of scratch memory available to the client for read (opcode 0x0D), write (opcode 0x0E) and copy (opcode 0x0F) operations. While arbitrary data can be stored in this memory, its main purposes are to hold the parameter structures for asynchronous requests to the GPU driver (request opcode 0x10) and request bundle definitions (request opcode 0x1C).

The address granularity of the scratch memory is 4 bytes; that is, the address "x" specified in Arg1 or Arg2 maps to the offset (4 * x). Because there is limited room available to encode the scratch memory addresses in a 32 bit wide request (Arg1, Arg2), the scratch memory block is organized in banks.

The size of a bank is 0x100 * 4 = 0x400, or 1 KByte. As currently implemented, the size of a block is four banks, for a total size of 4 KBytes.

4.4 Internal State Registers

4.4.1 Internal State Registers Overview

The API implements a set of registers that hold certain internal state values created as a result of the client operation. These registers are made available to the client for read and/or write operations through the request opcode 0x11.

The internal state held in these registers is subject to behavior detailed in Section 2.5.4; that is, upon an implementation phase change it would revert to a default value.

Currently implemented registers include the scratch memory bank register, events pending register, and event mask register.

4.4.2 Scratch Memory Bank Register

The Scratch memory bank register specifies the bank index used to evaluate the effective memory address. The total number of available banks is implementation dependent and is specified in the capabilities cap_dword[2], bits 3:1 (See Table 3-5 on page 21).

There are two separate bank sub-registers - one for read and another for write operations. The copy operation uses the read bank sub-register for its source data and the write bank sub-register for its destination. The asynchronous driver request operation always uses the read bank sub-register to compute its effective scratch memory addresses.

The contents of the scratch memory is subject to behavior detailed in Section 2.5.4; that is, upon an implementation phase change it would be cleared.

- Register address (Arg2 value in request opcode 0x11): 0x0
- Register layout:

Table 4-1. Scratch Memory Register Layout

Bit	Value	Default
31:16	0 (must be zero)	0
15:8	Read bank	0
7:0	Write bank	0

- Read bank: Index of the scratch memory bank to be used by read operations.
- Write bank: Index of the scratch memory bank to be used by write operations.

4.4.3 Events Pending Register

The events pending register specifies events that have occurred in the SMBPBI server domain that is bound to the SMBus slave. This register requires handling from the SMBus master. Each

bit in this register represents an individual event type. These bits get set when an event of the associated type occurs.

Another related internal state register is the event mask register (Section 4.4.4), that is laid out in a similar fashion - for every bit in the events pending register there is a corresponding bit in the mask register.

Whenever there is at least one bit set in the events pending register that is not masked by the corresponding bit in the mask register, the event gets propagated to the Status Register, meaning that as soon as the GPU needs to post the status of any completed request, it would simultaneously set the "event" field of the Status Register.

Whenever the SMBus master detects that the "event" field is set in the Status Register, it is supposed to read the event register and inspect the set bits in order to learn which event types have occurred.

Some of the event type bits in the event register can be cleared by the SMBus master by writing zero into the corresponding bit of this register, while other bits cannot be cleared, as explained below.

There are 2 types of the event triggering logic:

- Edge triggered events get asserted as soon the associated condition occurs. Events of this type can be cleared in the event register by writing zero into the corresponding bit of the event register.
- Level triggered events become asserted and stay asserted as long as the associated condition persists. These events cannot be cleared by writing zero into the corresponding bit of the event register, they however will self clear as soon as the associated condition disappears.

- Register address (Arg2 value in request opcode 0x11): 0x1
- Register layout:

Table 4-2. Events Pending Register Layout

Bit	Value	Default Value	Event Type
31:7	Reserved	0	N/A
6	MIG Toggle Command Success	0	Edge
5	Reserved	0	N/A
4	Clock Limit Set Success	0	Edge
3	TGP Limit Set Success	0	Edge triggered
2	Driver Error Messages waiting	0	Level triggered
1	GPU reset required	0	Level triggered
0	Server has restarted	0	Edge triggered

4.4.4 Event Mask Register

The event mask register specifies which events are propagated to the Status Register's "event bit". By default, all of the defined event bits are enabled (i.e. set to 0).

Setting any particular bits to '1' will mask the corresponding event from being propagated to the Status Register's "event" bit.

- Register address (Arg2 value in request opcode 0x11): 0x2
- Register layout:

Table 4-3. Event Mask Register Layout

Bit	Value	Default Value
31:7	Reserved	0
6	MIG Toggle Command Success	0
5	Reserved	0
4	Clock Limit Set Success	0
3	TGP Limit Set Success	0
2	Driver Error Messages waiting	0
1	GPU reset required	0
0	Server has restarted	0

Chapter 5. Future SMBPBI API Previews

The following sections provide a preview into future SMBPBI APIs planned for A100-Next. These details are provided early to help partners plan for necessary BMC development efforts. NVIDIA will continue to provide updates on the APIs noted here as they become available. Once API details are solidified and implemented, they will be merged into Chapter 3. "Interface Commands" on page 13 as appropriate.

These details are preliminary and likely to change.

5.1 04h – Get Power (Additions)

Feature additions for existing Opcode 04h. Read average power consumption (in mW) from a specific source with a 1 second average.

Note: This query is only supported when the driver is loaded.

Command

Opcode 04h – Get Power

Arg1 Desired power source

01h – Reserved

02h – Total HBM power consumption

Arg2 Unused

Data-In

Bit 31:0 Unused

Data-Out

Bit 31:0 Power in mW (100 mW resolution)

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Information was successfully retrieved.

5.2 10h – Submit/Poll Asynchronous Request (Additions)

Feature additions for existing Opcode 10h.

Command

Opcode 10h – Submit/Poll Asynchronous Request

Arg1 TBD – Query Compute Instance Information (MIG)

Parameters

```
#define SHARED_ENGINE_COUNT_WORDS 2

// Engine count word 0
#define SM_ENGINE_COUNT_FIELD                 9:0
#define COPY_ENGINE_COUNT_FIELD               15:10
#define DECODER_COUNT_FIELD                   21:16
#define ENCODER_COUNT_FIELD                   26:22
#define NVJPEG_COUNT_FIELD                    31:27

// Engine count word 1
#define OFA_COUNT_FIELD                       4:0

struct {
NvU16 migDeviceIndex;// [in] Index of the MIG device
NvU32 instanceMemoryUsedPct;// [out] Percentage of instance
memory used (rounded down)
NvU32 instanceMemTotal;                     // [out] Total
memory available for compute instances under the GI  (MiB)
NvU32 sharedEngineCounts[SHARED_ENGINE_COUNT_WORDS];[out]  //
Shared engine counts for the MIG instance
```

TBD – Query GPU and Compute Instance IDs (MIG)

Parameters

```
struct {
NvU32 migDeviceIndexMask;// [out] Bitmask representing all MIG
devices present when the query is issued
}
```

```
migDeviceIndexMask is a bitmask with the set bits representing
IDs associated with MIG devices.
IDs from reading the bitmask can be used with other queries to
collect telemetry for the given MIG device
Event bits will indicate if the instance mask is no longer valid
and needs to be re-queried.
```

Arg2 Location of the requested parameters in scratch memory

5.3 18h – Query GPU State Flags (Additions)

Feature additions for existing Opcode 18h.

Command

Opcode 18h – Query GPU State Flags

Arg1 Device state flags page

00h – Device state flags page 0 01h – Device state flags page 1

Arg2 Unused

Data-In

Bit 31:0 Unused

Data-Out

If Arg1 = 0x00 Device state flags page 0

Bit 0:6 <unchanged>

Bit 7 Device state: This means that driver has prevented the GPU from running with in-band tools.

0 - Enabled

1 - Excluded

If Arg1 = 0x01<unchanged>

Extended Data-Out

Bit 31:0 Unused

Status

Bit 28:24 SUCCESS

ERR_ARG1

NOT_SUPPORTED

5.4 1Ah – Query NVLink Information (Additions)

Feature additions for existing Opcode 1Ah.

Note: Copy bit should not be set for this query.

Command

Opcode 1Ah – Query NVLink Information

Arg1 07h – NVLink status (format V2)

0Dh – NVLink training errors

0Eh – NVLink runtime errors

TBD – NVLink version

TBD – NVLink global capabilities

Arg2 If Arg1 = 07h: NVLink page index

Use the page index to select which links to report. Links (Arg2*16) to (Arg2*16)+7 will be encoded in the data-out register. Links (Arg2*16)+8 to (Arg2*16)+15 will be encoded in the extData register. Invalid link indices (beyond the number of links) will be in the OFF_STATE.

0xFF - Aggregate status of all links. Output will be the status if all link statuses are the same. If not all link statuses are the same, ERR_STATE will be returned.

If Arg1 = 0Dh – 0Eh: NVLink page index.

Use the page index to select which links to report. Each page will contain 32 links. The first 32 links will be outputted in the data register. The next 32 links will be outputted in the extended data register.

NVLink page index = 0xFF will provide aggregate count of all links.

Else,

Unused

Data-In

Bit 31:0Unused

Data-Out

If Arg1 = 07h: Report up to a total of 8 NVLink states, each encoded in 4 bits. This can be indexed by this bit mapping: (3+(i)*4):(0+(i)*4)

0x0 OFF_STATE

0x1 SAFE_STATE

0x2 ACTIVE_STATE

0x3 ERROR_STATE

0x4 L1_LOW_POWER_STATE

0x5 NVLINK_DISABLED

The data will be returned by the bit mapping above for links (Arg2*16) to (Arg2*16)+7, or as the aggregate state of all links.

If Arg1 = 0Dh: Bit of first 32 links in page indicated

0x0 - No training error

0x1 - Training error occurred

Aggregate count returned if arg2 is 0xFF

If Arg1 = 0Eh: Bit of first 32 links in page indicated

0x0 - No runtime error

0x1 - Runtime error occurred

Aggregate count returned if arg2 is 0xFF

If Arg1 = TBD – NVLink version: NVLink version is returned

If Arg1 = TBD – NVLink global capabilities:

0:00x1: P2P over NVLink is supported

0x0: P2P over NVLink is not supported

1:1 0x1: sysmem can be accessed over NVLink

0x0: sysmem cannot be accessed over NVLink

2:2 0x1: P2P atomics are supported over NVLink

0x0: P2P atomics are not supported over NVLink

3:3 0x1: sysmem atomic transactions are supported over NVLink

0x0: sysmem atomic transactions are not supported over NVLink

4:4 0x1: PEX tunnelling is supported over NVLink

0x0: PEX tunnelling is not supported over NVLink

5:5 0x1: SLI over NVLink is supported

	0x0: SLI over NVLink is not supported
6:6	0x1: subdevice is capable of sensing SLI bridge
	0x0: subdevice is not capable of sensing SLI bridge

Extended Data-Out

If Arg1 = 07h: NVLink states for links (Arg2*16)+8 to (Arg2*16)+15

If Arg1 = 0Dh: Bit mask of next 32 links in page indicated

0x0 - No training error

0x1 - Training error occurred

If Arg1 = 0Eh: Bit mask of next 32 links in page indicated

0x0 - No runtime error

0x1 - Runtime error occurred

Status

If Arg1 = 0Dh-0Eh:

Bit 28:24	SUCCESS
	ERR_ARG2
	ERR_NOT_SUPPORTED
Bit 23:12	Total runtime errors
Bit 11:0	Total training errors

Else,

SUCCESS	Information was successfully retrieved.

5.5 1Bh – Query Clock Frequency Info (Additions)

Feature additions for existing Opcode 1Bh.

Command

Opcode	1Bh – Query Clock Frequency Info
Arg1	Clock parameter
	04h – Clock Throttle Reasons
	05h – Supported Clocks
	06h – Lock or Reset Memory Clocks

Data-In

If Arg1 = 05h:

clock index idx, 0 <= idx < Nclk

If Arg1 = 06h:

Data-in[31:16]: Max clock value, MHz

Data-in[15:0]: Min clock value, MHz

Alternatively:

Data-in: 0 - reset previously locked memory clocks

Else,

Bit 31:0 Unused

Data-Out

Bit 31:0

If Arg1 = 04h: Bitmask of throttle reasons:

Bit	Reason
0	GPU idle
1	User Defined Clocks
2	Applications Clocks Setting
3	SW Power Cap
4	HW Slowdown
5	HW Thermal Slowdown
6	HW Power Brake Slowdown
7	Sync Boost
8	SW Thermal Slowdown Tavg
9	SW Thermal Slowdown Tlimit
10	Display Clock Setting

If Arg1 = 05h: Supported Clocks

Data-out[15:0]: clock[idx], MHz

Data-out[31:16]: clock[idx+1], MHz or zero if idx+1 >= Nclk

Extended Data-Out

If Arg1 = 05h:

Ext-data[15:0]:clock[idx+2], MHz or zero if idx+2 >= Nclk

Ext-data[31:16]:clock[idx+3], MHz or zero if idx+3 >= Nclk

Else,

Bit 31:0Unused

Status

If Arg1 = 05h:

Status[28:24]: additionally NV_MSGBOX_CMD_STATUS_ERR_DATA in case idx >= Nclk

Status[23..0]: Nclk: total number of supported clocks in the requested domain

SUCCESS Information was successfully retrieved.

5.6 TBD - Get Dynamic System Information

Used to retrieve dynamic system information, as specified by the type and offset

Command

Opcode	TBD - Get Dynamic System Information
Arg1	Type of information desired
Arg2	Offset to requested data in increments of 8 bytes.

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Requested data (Offset:Offset+3 bytes)

Extended Data-Out

Bit 31:0	Requested data (Offset+4:Offset+7 bytes)

Status

SUCCESS	Information was successfully retrieved.
ERR_ARG1	Arg1 value does not correspond to a supported information-type.
ERR_ARG2	Offset stored in Arg2 is invalid for the requested information-type.

Table 5-1. Get Dynamic System Information Arg1 Encoding

Arg1 Value	Size (Bytes)	Type	Example	Format Version	Capability Location (dword/bit)	Note
TBD	16	Driver version	450.60.03	1	TBD	Used to retrieve the version of the driver loaded. If no driver is loaded since device reset, this query will return ERR_NOT_AVAILABLE. If the driver is unloaded, this query will return the last driver loaded version.

5.7 TBD – Get Voltage Information

Read voltage values (in µV) from a specific source

Command

Opcode	TBD – Get Voltage Information
Arg1	Desired power source
	00h – Reserved
	01h – GPU core voltage
	02h – Reserved
Arg2	Unused

Data-In

Bit 31:0	Unused

Data-Out

Bit 31:0	Voltage in µV

Extended Data-Out

Bit 31:0 Unused

Status

SUCCESS Information was successfully retrieved.

5.8 TBD – GPU Performance Monitoring

Used to query various GPU performance metrics

Examples:

- To get real time device level graphics engine activity:

  arg1= Bit: (7:6) 0x00 (5:0) 0x00, arg2=0xFF

  Data_out will return real time data.

- To get a real time device level graphics engine activity and a snapshot for all device level metrics:

  arg1=Bit: (7:6) 0x02 (5:0) 0x00, arg2=0xFF

  Data_out will return graphics engine activity from this snapshot

- To get SM activity from last snapshot:

  arg1=Bit: (7:6) 0x01 (5:0) 0x01, arg2 0xFF

  Data_out will return SM activity from this snapshot

Command

Opcode	TBD – GPU Performance Monitoring	
Arg1	Bit 7:6	
	0x00	Return instantaneous metric value
	0x01	Return metric value from last snapshot
	0x02	Take a new snapshot, return metric value from the snapshot
	Bits 5:0	Choose metric
	0x00	Graphics Engine
	0x01	SM Activity
	0x02	SM Occupancy
	0x03	Tensor Core Activity
	0x04	DRAM usage
	0x05	FP64 Activity

0x06	FP32 Activity
0x07	FP16 Activity
0x08	PCIe bandwidth
0x09	NvLink bandwidth
0x0a	NVDEC Utilization
0x0b	NVJPG Utilization
0x0c	FECS Utilization
0x0d	CE read/write bandwidth
0x0e	CE Utilization
0x0f	Compute CWD Stall Rates
0x10	NvOFA Utilization

Arg2 Select device-level or partition-level

0xff	Device level
0x00-0xfe	Partition index

Data-In

Bit 31:0 Unused

Data-Out

Arg1=0x00 Graphic Engine: Active cycles/Total cycles in percentage

Bit 7:0	Fractional part, granularity .01%
Bit 15:8	Integer part, granularity 1%

Arg1=0x01 SM Activity: Active cycles/Total cycles in percentage

Bit 7:0	Fractional part, granularity .01%
Bit 15:8	Integer part, granularity 1%

Arg1=0x02 SM Occupancy: Warps/maximum of warps in percentage

Bit 7:0	Fractional part, granularity .01%
Bit 15:8	Integer part, granularity 1%

Arg1=0x03 Tensor Core Activity: Active cycles/Total cycles in percentage

Bit 7:0	Fractional part, granularity .01%
Bit 15:8	Integer part, granularity 1%

Arg1=0x04 DRAM usage

Data-out + Data-ext	Total B/KB/MB/GB transferred on DRAM per sec, bits in status register will show the granularity

Arg1=0x05 FP64 Activity: Active cycles/Total cycles in percentage

Bit 7:0	Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x06 FP32 Activity: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x07 FP16 Activity: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x08 PCIe bandwidth

Data-out + Data-ext Total B/KB/MB/GB transferred on PCIE per sec, bits in status register will show the granularity

Arg1=0x09 NvLink bandwidth

Data-out + Data-ext Total B/KB/MB/GB transferred on NvLink per sec, bits in status register will show the granularity

Arg1=0x0A NVDec utilization: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x0B NVJPG utilization: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x0C FECS utilization: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x0D CE Read/write bandwidthActive cycles/Total cycles in percentage

Data-out + Data-ext Total B/KB/MB/GB transferred per sec, bits in status register will show the granularity

Arg1=0x0E CE Utilization: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x0F Compute CWD Stall Rates: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Arg1=0x10 NvOFA Utilization: Active cycles/Total cycles in percentage

Bit 7:0 Fractional part, granularity .01%

Bit 15:8 Integer part, granularity 1%

Extended Data-Out

Bit 31:0 Unused

Status

Bits 2:0 Indicate granularity of the data returned for Arg1=0x4,0x8,0x9,0x0A,0x0D

0x00 - Bytes

0x01 - KB

0x02 - MB

0x03 - GB

Bit 28:24 SUCCESS

ERR_ARG1

ERR_ARG2

ERR_NOT_SUPPORTED: CC mode will prevent metrics query and cap bits will be turned off

Appendix A. Implementation of Specific Parameters

Address Offsets

This section lists the GPU-specific address offsets of interface registers in the GPU SMBus register space.

Table A-1. GPU SMBus Post Box Register Addresses

Product	Command and Status	Data	Extended Data
Tesla M2050	5Ch	58h	(not supported)
Tesla M2070	5Ch	58h	(not supported)
Tesla M2070Q	5Ch	58h	(not supported)
Tesla X2070	5Ch	58h	(not supported)
Tesla M2075	5Ch	58h	(not supported)
Tesla M2090	5Ch	58h	(not supported)
Tesla X2090	5Ch	58h	(not supported)
Tesla K10	5Ch	5Dh	5Eh
Tesla K20	5Ch	5Dh	5Eh
Tesla K20X/s/m/Xm/c	5Ch	5Dh	5Eh
Tesla K40m/c/t/st	5Ch	5Dh	5Eh
Tesla K80	5Ch	5Dh	5Eh
Tesla M6	5Ch	5Dh	5Eh
Tesla P100	5Ch	5Dh	5Eh
Tesla P40	5Ch	5Dh	5Eh
Tesla P4	5Ch	5Dh	5Eh
Tesla P6	5Ch	5Dh	5Eh
Tesla V100 (PCIe/SXM2/SXM3)	5Ch	5Dh	5Eh

Table A-1. GPU SMBus Post Box Register Addresses (Continued)

Product	Command and Status	Data	Extended Data
NVIDIA T4	5Ch	5Dh	5Eh
Quadro M5000/M6000	5Ch	5Dh	5Eh
Quadro P400/P600/P1000	5Ch	5Dh	5Eh
Quadro P2000/P2200/ P4000	5Ch	5Dh	5Eh
Quadro P5000/P6000	5Ch	5Dh	5Eh
Quadro GV100	5Ch	5Dh	5Eh
Quadro GP100	5Ch	5Dh	5Eh
Quadro RTX8000/ RTX6000/RTX5000/ RTX4000	5Ch	5Dh	5Eh
NVIDIA A100 (SXM4)	5Ch	5Dh	5Eh
NVIDIA A100/A40/A30/A10 (PCIe)	5Ch	5Dh	5Eh
NVIDIA RTX A6000	5Ch	5Dh	5Eh
GeForce RTX Series	5Ch	5Dh	5Eh
NVIDIA A100-Next (SXM)	5Ch	5Dh	5Eh

Features and Capabilities

Table A-2. GPU-Specific Features and Capabilities

Product	SMBus ARP	SMBPBI Slave Commands
Tesla M2050	Not Supported	Not Supported
Tesla M2070	Not Supported	Not Supported
Tesla M2070Q	Not Supported	Not Supported
Tesla X2070	Not Supported	Not Supported
Tesla M2075	Not Supported	Not Supported
Tesla M2090	Not Supported	Not Supported
Tesla X2090	Not Supported	Not Supported
Tesla K10	Supported	Supported
Tesla K20	Supported	Supported
Tesla K20X/s/m/Xm/c	Supported	Supported
Tesla K40m/c/t/st	Supported	Supported

Table A-2. GPU-Specific Features and Capabilities (Continued)

Tesla K80	Not supported	Supported
Tesla M6	Not supported	Not supported
Tesla P100	Not supported	Not supported
Tesla P100 (SXM2)	Supported	Not supported
Tesla P40	Not supported	Not supported
Tesla P4	Not supported	Not supported
Tesla P6 (MXM)	Supported	Not supported
Tesla V100 (SXM2)	Supported	Not supported
Tesla V100 (SXM3)	Supported	Not supported
Tesla V100 (PCIe)	Not supported	Not supported
NVIDIA T4	Not supported	Not supported
Quadro M5000/M6000	Not supported	Not supported
Quadro P400/P600/P1000	Not supported	Not supported
Quadro P2000/P2200/P4000	Not supported	Not supported
Quadro P5000/P6000	Not supported	Not supported
Quadro GV100	Not supported	Not supported
Quadro GP100	Not supported	Not supported
Quadro RTX8000/RTX6000/ RTX5000/RTX4000	Not supported	Not supported
NVIDIA A100 (SXM4)	Supported	Not supported
NVIDIA A100/A40/A30/A10 (PCIe)	Supported	Not supported
NVIDIA RTX A6000	Supported	Not supported
GeForce RTX Series	Supported	Not supported
NVIDIA A100-Next (SXM)	Supported	Not supported

Note: SXM GPUs support SMBus ARP at the SXM module level, but for SXM GPUs on baseboard level products ARP is disabled due to the onboard MCU/FPGA.

Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

HDMI

HDMI, the HDMI logo, and High-Definition Multimedia Interface are trademarks or registered trademarks of HDMI Licensing LLC.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
www.nvidia.com