SANBlaZe

White Paper

CMB and HMB Verification with SANBlaze

Controller Memory Buffer (CMB) and Host Memory Buffer (HMB) Verification Using the SANBlaze SBExpress NVMe Drive Test System

Author:

Haiyan Lin, Sr. Software Engineer, SANBlaze

Table of Contents

Introduction of CMB 2
CMB Related Registers and Settings 3
CMB Configuration with SANBlaze SW 5
CMB Configuration with SANBlaze Command Line Tool 5
CMB Configuration with SANBlaze Python APIs 6
Introduction of HMB 7
HMB Related Parameters and Feature ID 0Dh Settings 7
HMB Configuration with SANBlaze SBExpress Software 9
HMB Configuration with SANBlaze Command Line Tool 9
HMB Configuration with SANBlaze Python APIs 11
Summary 12

This white paper provides an overview of CMB/HMB, status check, setup and configuration using the SANBlaze system with several examples.

Introduction of CMB

The CMB (Controller Memory Buffer) is a region of general purpose read/write memory on the controller. During PCIe initialization the controller will request a certain amount of memory it needs. The host memory manager will assign the memory size and base address by writing into registers on the controller. All NVMe controllers have their own BAR0 and BAR1 for the host to write where that controller's memory begins. Every NVMe controller will have a different base address in BAR0/BAR1. The controller memory addresses are above any the system would normally use.

Figure 1 below shows the 64-bit memory address range available to the host.

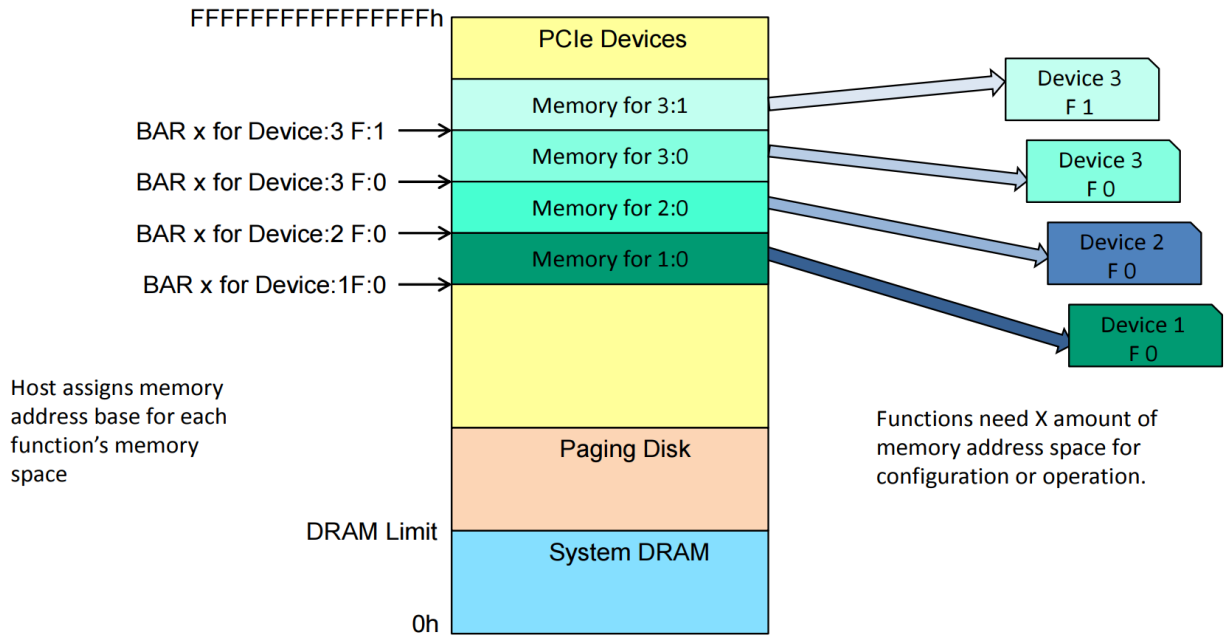

Figure 1. PCIe Memory Locations

The system physical memory limit is labeled "DRAM Limit", above the system DRAM it is the system paging memory, then the system memory available for different PCIe devices/functions. The CMB is physically on the device but addressed and treated like host memory. That way, by adding many more functions, each function supplies the memory it needs and does not take any of the system memory.

Before NVMe 1.2, the data and metadata for read/write commands, SQ, CQ, PRP and SGL lists were stored in host physical memory. Since NVMe 1.2 the optional CMB feature allows the host to define an area buffer in the controller to hold some or all of the mentioned data above. If the host puts this data in a buffer that is on the device, the device does not have to use PCIe facilities to fetch the data or send it to the host. The addressing of the storage facilities is part of the host address range and identified by the BARs (BAR0-BAR5) in the device's configuration header space.

CMB Related Registers and Settings

There are 6 CMB related NVMe controller registers defined in the latest NVMe specification (“NVMe Base 1.4_NEXT 2020.10.12a.docx”) so far as follows. The NVMe drive FW supporting NVMe v1.4 (“2019.06.10-Ratified”) only defined the top 4 CMB related registers below. The NVMe v1.3 or earlier version only defined the top 2 CMB related registers below.

1. CMBLOC – Controller Memory Buffer Location, Offset 38h
2. CMBSZ – Controller Memory Buffer Size, Offset 3Ch
3. CMBMSC – Controller Memory Buffer Memory Space Control, Offset 50h
4. CMBSTS – Controller Memory Buffer Status, Offset 58h
5. CMBEBS – Controller Memory Buffer Elasticity Buffer Size, Offset 5Ch
6. CMBSWTP – Controller Memory Buffer Sustained Write Throughput, Offset 60h

In NVMe v1.4 the NVMe controller CAP (Controller Capabilities) register has RO (read-only) bit 57 for CMBS (Controller Memory Buffer Supported). If set CAP.CMBS to ‘1’ then the controller supports for the CMB. The host indicates intent to use the CMB by setting CMBMSC.CRE to ‘1’. Once this bit is set to ‘1’, the controller indicates the properties of the CMB via the CMBLOC and CMBSZ registers. In NVMe 1.3 or earlier versions there is no CAP.CMBS bit defined. The SANBlaze driver can support different NVMe versions, and it supports CMB as well based on the NVMe version the SSD supports. For example, if SSD supports NVMe 1.4 then CAP.CMBS will be checked, but for SSD drives supporting an NVMe version earlier than 1.4, it will not check CAP.CMBS.

Following is an example of an NVMe v1.2 drive with the CMB related registers dump from the SANBlaze software:

```
ssh <ipaddr of system e.g. 192.168.100.104>
User: root
PW: SANBlaze (your admin may have changed this)
cd /virtualun/apis/
python3
>>> from sanblaze_apis import *


>>> t103 = XML_API("192.168.100.104", 0, 103, 1, raw_return=2)
>>> temp = t103.dump_nvme_controller_registers()
<result>
  <controller>
    <id>103</id>
. . .
    <cap>00000030320307ff</cap>
    <vs>00010200</vs>
. . .
    <cmbloc>00000004</cmbloc>
    <cmbsz>00080001</cmbsz>
. . .
  </controller>
  <status>0</status>
</result>

                    VS - Version:
                     Tertiary_Version_Number = 0x00
                        Minor_Version_Number = 0x02
                       Major_Version_Number  = 0x0001
. .
    CAP - Controller Capabilities:
```

```
              Maximum_Queue_Entries_Supported = 0x07FF
                   Contiguous_Queues_Required = 0x01
              Arbitration_Mechanism_Supported = 0x01
                                      Timeout = 0x32
                                Doorbell_Stride = 0x00
                                 NSSR_Supported = 0x01
                           Command_Sets_Supported = 0x0001
                           Boot_Partition_Support = 0x00
                         Memory_Page_Size_Minimum = 0x00
                         Memory_Page_Size_Maximum = 0x00
          Persistent_Memory_Region_Supported = 0x00
          Controller_Memory_Buffer_Supported = 0x00
. . .
CMBLOC - Controller Memory Buffer Location:
                        Base_Indicator_Register = 0x04
               CMB_Queue_Mixed_Memory_Support = 0x00
   CMB_Queue_Physically_Discontiguous_Support = 0x00
     CMB_Data_Pointer_Mixed_Locations_Support = 0x00
CMB_Data_Pointer_and_Command_Independent_Locations_Support = 0x00
       CMB_Data_Metadata_Mixed_Memory_Support = 0x00
                     CMB_Queue_Dword_Alignment = 0x00
                                        Offset = 0x000000
CMBSZ - Controller Memory Buffer Size:
                       Submission_Queue_Support = 0x01
                       Completion_Queue_Support = 0x00
                           PRP_SGL_List_Support = 0x00
                              Read_Data_Support = 0x00
                             Write_Data_Support = 0x00
                                    Size_Units = 0x00
                                          Size = 0x000080
>>> temp
{'result': {'controller': {'id': '103', . . ., 'cap':
{'Maximum_Queue_Entries_Supported': 2047, 'Contiguous_Queues_Required': 1,
'Arbitration_Mechanism_Supported': 1, 'Timeout': 50, 'Doorbell_Stride': 0,
'NSSR_Supported': 1, 'Command_Sets_Supported': 1, 'Boot_Partition_Support': 0,
'Memory_Page_Size_Minimum': 0, 'Memory_Page_Size_Maximum': 0,
'Persistent_Memory_Region_Supported': 0, 'Controller_Memory_Buffer_Supported': 0},
'vs': {'Tertiary_Version_Number': 0, 'Minor_Version_Number': 2, 'Major_Version_Number
': 1}, . . ., 'cmbloc': {'Base_Indicator_Register': 4,
'CMB_Queue_Mixed_Memory_Support': 0, 'CMB_Queue_Physically_Discontiguous_Support': 0,
'CMB_Data_Pointer_Mixed_Locations_Support': 0,
'CMB_Data_Pointer_and_Command_Independent_Locations_Support': 0,
'CMB_Data_Metadata_Mixed_Memory_Support': 0, 'CMB_Queue_Dword_Alignment': 0, 'Offset':
0}, 'cmbsz': {'Submission_Queue_Support': 1, 'Completion_Queue_Support': 0,
'PRP_SGL_List_Support': 0, 'Read_Data_Support': 0, 'Write_Data_Support': 0,
'Size_Units': 0, 'Size': 128}, . . ., 'status': '0'}}
```

You can see that the controller supports CMB, and CMBLOC.BIR (Base Indicator Register in Controller Memory Buffer Location) = 0x04 which indicates the BAR for the lower 32-bits of the address is 4, so the BAR4 and BAR5 contain the Controller Memory Buffer address. NVMe v1.2 only defines CMBLOC.OFST besides the CMBLOC.BIR, and the drive has CMBLOC.OFST = 0x00000, so the offset of the Controller Memory Buffer from the contained address in BAR4 and BAR5 is 0.

The CMBSZ.SQS (Submission Queue Support bit in Controller Memory Buffer Size) = 1 so the controller supports ASQ and IOSQ (Admin and I/O Submission Queues) in the Controller Memory Buffer. The CMBSZ.CQS (Completion Queue Support) = 0 so all Completion Queues shall be placed in host memory. The CMBSZ.LISTS (PRP SGL List Support) = 0 so all PRP Lists and SGLs shall be placed in host memory as well. The CMBSZ.RDS (Read Data Support) = 0 so the controller does not support data and metadata in the Controller Memory Buffer for commands that transfer data from the controller to the host. The

CMBSZ.WDS (Write Data Support) = 0 so the controller does not support data and metadata in the Controller Memory Buffer for commands that transfer data from the host to the controller. The CMBSZ.SZU (Size Units) = 0 indicates the granularity of the Size field is 4KB. The CMBSZ.SZ (Size) = 0x00080 indicates the size of the Controller Memory Buffer available for use by the host is 128 * 4 = 512 KB.

CMB Configuration with SANBlaze SW

You can configure CMB using the SANBlaze software through the SBExpress GUI, command line tool (CLI), or Python APIs. In this white paper we only show the CMB configuration through the command line and Python APIs.

CMB Configuration with SANBlaze Command Line Tool

You can find the CMB configuration through the controller proc file like /iport0/target103:

```
[root@sbexpr104 ~]# grep UseCMB /iport0/target103
UseCMB=0/0/0/524288/1/0
```

There are six values in the output above separated by "/". These six values are defined as follows:

- 1st value: The CMB size currently in use. 0 means 0 bytes of CMB is currently in use.
- 2nd value: The set of options that are using the CMB such as bit 0 is SQs, bit 1 is CQs, bit 2 is LISTS, bit 3 is RDS, bit 4 is WDS. A value of 0 means nothing is using the CMB.
- 3rd value: The set of options that use the CMB such as bit 0 is SQs, bit 1 is CQs, bit 2 is LISTS, bit 3 is RDS, bit 4 is WDS. A value of 0 means nothing is using the CMB.
- 4th value: The size of CMB. The value 524288 means the size of the CMB is 512 KB for the controller above.
- 5th value: What options the controller supports for using CMB. 1 means bit 0 is 1 so SQs can use CMB for the controller above.
- 6th value: The chunk size when doing RD (read data) or WD (write data). It is defined in the driver. There is a routine memcpy() in the driver to copy to/from CMB. The chunk size is the size passing to memcpy() if not 0. If it is 0 then there is no limit and the size passing to memcpy() is the amount left in the SGL.

Among these 6 values above only 2 of them are configurable. They are the 3rd value (the set of options that are desired to use the CMB) and the 6th value (the chunk size when doing RD or WD). You can change these 2 values through the command line such as "echo UseCMB=%x, %d > /iportX/targetY". For example, you can change the 3rd value to 1 and the 6th value to 1024 as follows:

```
[root@sbexpr104 ~]# echo UseCMB=1,1024 > /iport0/target103
[root@sbexpr104 ~]# grep UseCMB /iport0/target103
UseCMB=0/0/1/524288/1/1024
```

Some changes require a restart to the controller to take effect such as bit 0 or/and bit 1 of the 3rd value. You can restart the controller with the command shown below:

```
[root@sbexpr104 ~]# echo ControllerRestart > /iport0/target103
[root@sbexpr104 ~]# grep UseCMB /iport0/target103
UseCMB=524288/1/1/524288/1/1024
```

Changes to the other bits (except bit 0 or/and bit 1) of the 3rd value and the 6th value do not require a controller restart to take effect.

CMB Configuration with SANBlaze Python APIs

You can also use the SANBlaze Python APIs to check CMB status and configure the CMB as follows:

```
>>> temp = t103.get_proc_target_info()
{
  "read": [{
    "iport": 0,
    "target": 103,
    . . .
    "UseCMB": "524288/1/1/524288/1/1024",
    . . .
  }],
  "status": 0
}

>>> temp
{'read': [{'iport': 0, 'target': 103, . . ., 'UseCMB': '524288/1/1/524288/1/1024', . .
.}], 'status': 0}

>>> temp = t103.get_vlun_nvme_update_controller("SetCmb", options_set_using_CMB=0x0,
chunk_size_RD_WD=0)
<result>
  <controller>
    <id>103</id>
    <options_set_using_CMB>0x0</options_set_using_CMB>
    <chunk_size_RD_WD>0</chunk_size_RD_WD>
    <set_cmb_status>
    </set_cmb_status>
  </controller>
  <status>0</status>
</result>

>>> temp
{'result': {'controller': {'id': '103', 'options_set_using_CMB': '0x0',
'chunk_size_RD_WD': '0', 'set_cmb_status': ''}, 'status': '0'}}

>>> temp = t103.get_vlun_nvme_update_controller("ResetController")
<result>
  <status>0</status>
</result>

>>> temp
{'result': {'status': '0'}}

>>> temp = t103.get_proc_target_info()
{
  "read": [{
    "iport": 0,
    "target": 103,
    . . .
    "UseCMB": "0/0/0/524288/1/0",
    . . .
    }],
  "status": 0
}

>>> temp
```

{'read': [{'iport': 0, 'target': 103, . . ., 'UseCMB': '0/0/0/524288/1/0', . . .}],
'status': 0}

Introduction of HMB

The Host Memory Buffer (HMB) feature allows the controller to utilize an assigned portion of host memory exclusively. The use of the host memory resources is vendor specific and is not defined until NVMe v1.2. Host software may not be able to provide any or a limited amount of the host memory resources requested by the controller. The controller shall function properly without host memory resources.

The controller may indicate limitations for the minimum usable descriptor entry size and the maximum number of descriptor entries. If the host does not create the Host Memory Buffer within the indicated limits, then the host memory allocated for use by the controller may not be fully utilized.

During initialization, the host software may provide a descriptor list that describes a set of host memory address ranges for exclusive use by the controller. The host memory resources assigned are for the exclusive use of the controller (host software should not modify the ranges) until host software requests that the controller release the ranges and the controller completes the Set Features command. The controller is responsible for initializing the host memory resources. The host software should request that the controller release the assigned ranges prior to a shutdown event, a Runtime D3 event, or any other event that requires the host software to reclaim the assigned ranges. After the controller acknowledges that the ranges are no longer in use, the host software may reclaim the host memory resources. In the case of Runtime D3, the host software should provide the host memory resources to the controller again and inform the controller that the ranges were in use prior to the RTD3 event and have not been modified.

The host memory resources are not persistent in the controller across a reset event. The host software should provide the previously allocated host memory resources to the controller after the reset completes. If the host software is providing previously allocated host memory resources (with the same contents) to the controller, the Memory Return bit is set to ‘1’ in the Set Feature Identifier 0Dh command.

The controller shall ensure that there is no data loss or data corruption in the event of a surprise removal while the Host Memory Buffer feature is being utilized. During shutdown, if there is an active host memory buffer, inform the controller that it should stop using the buffer.

HMB Related Parameters and Feature ID 0Dh Settings

There are some parameters in identify controller output related to HMB as follows:

- Host Memory Buffer Preferred Size (HMPRE): This field indicates the preferred size that the host is requested to allocate for the Host Memory Buffer feature in 4 KiB units. This value shall be greater than or equal to the Host Memory Buffer Minimum Size (HMMIN). If this field is non-zero, then the Host Memory Buffer feature is supported. If this field is cleared to 0h, then the Host Memory Buffer feature is not supported.
- Host Memory Buffer Minimum Size (HMMIN): This field indicates the minimum size that the host is requested to allocate for the Host Memory Buffer feature in 4 KiB units. If this field is

cleared to 0h, then the host is requested to allocate any amount of host memory possible up to the HMPRE value.

- Host Memory Buffer Minimum Descriptor Entry Size (HMMINDS): This field indicates the minimum usable size of a Host Memory Buffer Descriptor Entry in 4 KB units. If this field is cleared to 0h, then the controller does not indicate any limitations on the Host Memory Buffer Descriptor Entry size.
- Host Memory Maximum Descriptors Entries (HMMAXD): This field indicates the number of usable Host Memory Buffer Descriptor Entries. If this field is cleared to 0h, then the controller does not indicate a maximum number of Host Memory Buffer Descriptor Entries.

The following example is from an NVMe v1.4 drive consisting of an HMB-related parameters dump using the SANBlaze software:

```
>>> t309 = XML_API("192.168.100.148", 0, 309, 1, raw_return=2)
>>> temp = t309.identify_controller()
Command IdentifyController passed on port 0 target 309 in tester 192.168.100.148
Command output is decoded as follows:
                                        VID = 0x15B7
                                      SSVID = 0x1414
. . .
                                       MFTA = 0x0032
                                      HMPRE = 0x00008000
                                      HMMIN = 0x00001000
. . .
                                    HMMINDS = 0x00000000
                                     HMMAXD = 0x0008
                                  NSETIDMAX = 0x0000
>>> temp
(1, {'VID': 5559, 'SSVID': 5140, . . ., 'MFTA': 50, 'HMPRE': 32768, 'HMMIN': 4096, . .
., 'HMMINDS': 0, 'HMMAXD': 8, 'NSETIDMAX': 0, . . .})
```

You can see the drive has HMPRE = 0x8000 which indicates the SSD supports HMB feature and the preferred size that the host is requested to allocate for Host Memory Buffer is 0x8000 * 4KB = 131072 KB = 128 MB. The HMMIN = 0x1000 so the minimum size that the host is requested to allocate for the Host Memory Buffer is 0x1000 * 4KB = 16 MB. The HMMINDS = 0 so the controller does not indicate any limitations on the Host Memory Buffer Descriptor Entry size. The HMMAXD = 0x0008 which indicates the number of usable Host Memory Buffer Descriptor Entries is 8.

The optional feature ID 0Dh is used to control the host memory buffer. The Host Memory Buffer feature provides a mechanism for the host to allocate a portion of host memory for the controller to use exclusively. After a successful completion of a Set Features enabling the host memory buffer, the host shall not write to the associated host memory region, buffer size, or descriptor list until the host memory buffer has been disabled.

After a successful completion of a Set Features command that disables the host memory buffer, the controller shall not access any data in the host memory buffer until the host memory buffer has been enabled. The controller should retrieve any necessary data from the host memory buffer in use before posting the completion queue entry for the Set Features command. Posting of the completion queue entry for the Set Features command acknowledges that it is safe for the host software to modify the host memory buffer contents.

For example, you can get the feature ID 0Dh from the drive above with the SANBlaze software as follows:

```
>>> temp = t309.get_feature(0x0D, 0)
Command GetFeatures passed on port 0 target 309 in tester 192.168.100.148
Command output is decoded as follows:
                      Host_Memory_Buffer_Size = 0x00000000
    Host_Memory_Descriptor_List_Address_Lower = 0x00000000
    Host_Memory_Descriptor_List_Address_Upper = 0x00000000
      Host_Memory_Descriptor_List_Entry_Count = 0x00000000

Command Completion Queue Status is decoded as follows:
                               CommandSpecific = 0x00000000
                                     Reserved0 = 0x00000000
                               SQ_Head_Pointer = 0x000F
                                 SQ_Identifier = 0x0000
                             Command_Identifier = 0x00E1
                    Status_Field:
                                       PhaseBit = 0x01
                                     StatusCode = 0x0000
                                  StatusCodeType = 0x00
                                       Reserved = 0x00
                               MoreInformation = 0x00
                                    DoNotRetry = 0x00
>>> temp
(1, {'Host_Memory_Buffer_Size': 0, 'Host_Memory_Descriptor_List_Address_Lower': 0,
'Host_Memory_Descriptor_List_Address_Upper': 0,
'Host_Memory_Descriptor_List_Entry_Count': 0})
```

From the CQ (completion queue) entry DWord 0 (CommandSpecific) you can see it is 0 so the host memory buffer is disabled, and the controller is not using the host memory buffer. The Host Memory Buffer Size (HSIZE), Host Memory Descriptor List Address Lower (HMDLAL), Host Memory Descriptor List Address Upper (HMDLAU) and Host Memory Descriptor List Entry Count (HMDLEC) are all 0 as well.

HMB Configuration with SANBlaze SBExpress Software

You can configure HMB with the SANBlaze software either through SBExpress GUI, command line interface (CLI) or Python APIs. In this white paper we only show the HMB configuration through command line and Python APIs.

HMB Configuration with SANBlaze Command Line Tool

You can find the HMB configuration through the controller proc file like /iport0/target309:

```
[root@VLUN-148-IPMI-100-122 ~]# grep UseHMB /iport0/target309
UseHMB=0,0,0/0,0,0/0,0,0/32768,4096,0,8
```

There are 4 groups of values in the output above separated by "/". These 4 groups of values are defined as follows:

- Group 1: Last set of HMB configuration inputs. The three values in this group are defined as follows:
    - 1st value: HMB enable/disable flag (1 for enable, 0 for disable)
    - 2nd value: The maximum size that should be allocated, in units of 4 KB; 4096 means 16 MB. If it is 0, then there is no limit.

    - 3rd value: The number of chunks ("buffer descriptor entries") to use. If it is 0, the buffer to be allocated is divided into chunks of at most 4 MB. If it is not 0, the buffer to be allocated is divided into that many chunks.
- Group 2: Current configuration of HMB. The three values in this group have the same definition as the 3 values in Group 1.
- Group 3: The number of chunks, max size and page size allocated for the controller. The 3 values in this group are defined as follows:
    - 1st value: The number of chunks.
    - 2nd value: The maximum size allocated, in units of 4 KB; 4096 means 16 MB. If it is 0, then there is no limit.
    - 3rd value: memory page size
- Group 4: HMB related parameters from identify controller outputs. The 4 values in this group are defined as follows:
    - HMPRE (Host Memory Buffer Preferred Size)
    - HMMIN (Host Memory Buffer Minimum Size)
    - HMMINDS (Host Memory Buffer Minimum Descriptor Entry Size)
    - HMMAXD (Host Memory Maximum Descriptors Entries)

You can modify the HMB configuration (enable/disable, allocation size and number of chunks) with a command line such as "`echo UseHMB=%d,%d,%d > /iportX/targetY`". The input parameters are defined as follows:

- The first %d is HMB enable/disable. 1 means enable and 0 means disable.
- The second %d is HMB allocation size in units of 4 KB. If 0 then use HMPRE instead.
- The third %d is number of chunks. If 0 then the driver will set chunks = 1024.
- X: Port number.
- Y: Controller number.

After issuing the command above, it will require a controller reset to take effect as shown below.

```
[root@VLUN-148-IPMI-100-122 ~]# echo UseHMB=1,4096,8 > /iport0/target309
[root@VLUN-148-IPMI-100-122 ~]# echo reset_ctlr=309 > /proc/vlun/nvme
[root@VLUN-148-IPMI-100-122 ~]# grep UseHMB /iport0/target309
UseHMB=1,4096,8/1,4096,8/8,4096,4096/32768,4096,0,8
```

Now dump feature ID 0Dh and it outputs as follows:

```
>>> temp = t309.get_feature(0xD, 0)
Command GetFeatures passed on port 0 target 309 in tester 192.168.100.148
Command output is decoded as follows:
                      Host_Memory_Buffer_Size = 0x00001000
    Host_Memory_Descriptor_List_Address_Lower = 0x31AB7380
    Host_Memory_Descriptor_List_Address_Upper = 0x00000020
      Host_Memory_Descriptor_List_Entry_Count = 0x00000008

Command Completion Queue Status is decoded as follows:
                              CommandSpecific = 0x00000001
                                    Reserved0 = 0x00000000
                              SQ_Head_Pointer = 0x002E
                                SQ_Identifier = 0x0000
                            Command_Identifier = 0x009A
                    Status_Field:
                                     PhaseBit = 0x01
                                   StatusCode = 0x0000
```

```
                                    StatusCodeType = 0x00
                                          Reserved = 0x00
                                    MoreInformation = 0x00
                                         DoNotRetry = 0x00
>>> temp
(1, {'Host_Memory_Buffer_Size': 4096, 'Host_Memory_Descriptor_List_Address_Lower':
833319808, 'Host_Memory_Descriptor_List_Address_Upper': 32,
'Host_Memory_Descriptor_List_Entry_Count': 8})
```

From the CQ (completion queue) entry DWord 0 (CommandSpecific) you can see it is 1 so the host memory buffer is enabled, and the controller is using the host memory buffer now. The Host Memory Buffer Size (HSIZE) is 0x1000 * 4KB = 16 MB, Host Memory Descriptor List Address Lower (HMDLAL) and Upper (HMDLAU) have been set, and Host Memory Descriptor List Entry Count (HMDLEC) is 8 as we set.

HMB Configuration with SANBlaze Python APIs

You can also use the SANBlaze Python APIs to check HMB status and configure HMB as in the following example to disable HMB, and set HMB allocation size to 0 and number of chunks to 0.

```
>>> temp = t309.get_proc_target_info()
{
  "read": [{
    "iport": 0,
    "target": 309,
. . .
    "UseHMB": [ "1", "4096", "8/1", "4096", "8/8", "4096", "4096/32768", "4096", "0", "8" ],
. . .
  }],
  "status": 0
}

>>> temp
{'read': [{'iport': 0, 'target': 309, . . ., 'UseHMB': ['1', '4096', '8/1', '4096',
'8/8', '4096', '4096/32768', '4096', '0', '8'], . . .}], 'status': 0}

>>> temp = t309.get_vlun_nvme_update_controller(valid_command="SetHmb", enable_hmb=0,
hmb_allocation_size=0, num_hmb_chunks=0)
<result>
  <controller>
    <id>309</id>
    <enable_hmb>0</enable_hmb>
    <hmb_allocation_size>0</hmb_allocation_size>
    <num_hmb_chunks>0</num_hmb_chunks>
    <set_hmb_status>
    </set_hmb_status>
  </controller>
  <status>0</status>
</result>

>>> temp
{'result': {'controller': {'id': '309', 'enable_hmb': '0', 'hmb_allocation_size': '0',
'num_hmb_chunks': '0', 'set_hmb_status': ''}, 'status': '0'}}

>>> temp = t309.get_vlun_nvme_update_controller("ResetController")
<result>
  <status>0</status>
</result>

>>> temp
{'result': {'status': '0'}}
```

```
>>> temp = t309.get_proc_target_info()
{
  "read": [{
    "iport": 0,
    "target": 309,
. . .
    "UseHMB": [ "0", "0", "0/0", "0", "0/0", "0", "0/32768", "4096", "0", "8" ],
. . .
  }],
  "status": 0
}

>>> temp
{'read': [{'iport': 0, 'target': 309, . . ., 'UseHMB': ['0', '0', '0/0', '0', '0/0',
'0', '0/32768', '4096', '0', '8'], . . .}], 'status': 0}

>>> temp = t309.get_feature(0x0D, 0)
Command GetFeatures passed on port 0 target 309 in tester 192.168.100.148
Command output is decoded as follows:
                      Host_Memory_Buffer_Size = 0x00000000
    Host_Memory_Descriptor_List_Address_Lower = 0x00000000
    Host_Memory_Descriptor_List_Address_Upper = 0x00000000
      Host_Memory_Descriptor_List_Entry_Count = 0x00000000

Command Completion Queue Status is decoded as follows:
                              CommandSpecific = 0x00000000
                                    Reserved0 = 0x00000000
                              SQ_Head_Pointer = 0x002D
                                SQ_Identifier = 0x0000
                            Command_Identifier = 0x0065
                    Status_Field:
                                     PhaseBit = 0x01
                                   StatusCode = 0x0000
                                StatusCodeType = 0x00
                                     Reserved = 0x00
                              MoreInformation = 0x00
                                   DoNotRetry = 0x00
>>> temp
(1, {'Host_Memory_Buffer_Size': 0, 'Host_Memory_Descriptor_List_Address_Lower': 0,
'Host_Memory_Descriptor_List_Address_Upper': 0,
'Host_Memory_Descriptor_List_Entry_Count': 0})
```

For a comprehensive list of API commands, please contact SANBlaze by emailing us at info@sanblaze.com.

Summary

SANBlaze now provides – in addition to an easy-to-use GUI interface and its extensive SBExpress automated test scripts for CMB/HMB verification – a way to programmatically control these interfaces and low-level commands through a common Python API interface for ease of integration into your existing Python test infrastructure. With the ability to run alongside your current test infrastructure, this will allow easy migration to the SANBlaze test platform and allow you to greatly enhance your CMB/HMB test coverage and results.