



# Intel® Virtual RAID on CPU (Intel® VROC) Private UEFI Device Info Protocol

Intel Confidential

## Legal Notices and Disclaimers

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined". Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Benchmark results were obtained prior to implementation of recent software patches and firmware updates intended to address exploits referred to as "Spectre" and "Meltdown". Implementation of these updates may make these results inapplicable to your device or system.

Tests document performance of components on a particular test, in specific systems. Differences in hardware, software, or configuration will affect actual performance. Consult other sources of information to evaluate performance as you consider your purchase. Results have been estimated based on internal Intel analysis and are provided for informational purposes only. Any difference in system hardware or software design or configuration may affect actual performance.

All documented performance test results are obtained in compliance with JESD218 Standards; refer to individual sub-sections within this document for specific methodologies. See [www.jedec.org](http://www.jedec.org) for detailed definitions of JESD218 Standards. Intel does not control or audit the design or implementation of third party benchmark data or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmark data are reported and confirm whether the referenced benchmark data are accurate and reflect performance of systems available for purchase.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

For copies of this document, documents that are referenced within, or other Intel literature please contact your Intel representative.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

\*Other names and brands may be claimed as the property of others. Copyright © 2018 Intel Corporation. All rights reserved.

Contents

Contents.....3

**1 Summary .....4**

    1.1 Protocol GUID.....4

    1.2 Protocol Interface structure.....4

        1.2.1 Parameters .....5

    1.3 Description .....6

    1.4 Example .....6

        1.4.1 Example sample code output .....10

# 1 Summary

Intel® Virtual RAID on CPU (Intel® VROC) version 5.5 PV adds more parameters to Intel VROC Private Device Info Protocol from previous released versions. This document's purpose is to enable easy creation of a custom EFI application to retrieve information about NVMe devices behind the VMD controllers in the EFI environment.

This Intel VROC protocol can be used to retrieve information for non-RAID and RAID member NVMe devices connected to Intel VMD domains.

## 1.1 Protocol GUID

```
// {EF104449-A5CD-44FF-9A65-150DFDA527EE}
#define EFI_VROC_DEVICE_INFO_PROTOCOL_GUID \
    { 0xef104449, 0xa5cd, 0x44ff, { 0x9a, 0x65, 0x15, 0xd, 0xfd, 0xa5, 0x27, 0xee } }
```

## 1.2 Protocol Interface structure

```
#pragma pack(1)
typedef struct _VROC_DEVICE_INFO_PROTOCOL
{
    UINTN      SocketNumber;
    UINTN      VmdControllerNumber;
    UINTN      RootPortNumber;
    UINT32     SlotNumber;
    UINTN      BusNumber;
    UINTN      DeviceNumber;
    UINTN      FunctionNumber;
    UINT16     VendorId;
    EFI_STRING SerialNumber;
    EFI_STRING ModelNumber;
    UINT64     TotalBlocks;
    UINT32     BlockSize;
    BOOLEAN    RaidDeviceMember;
    UINT16     RootPortOffset;
    EFI_HANDLE BlockDeviceHandle;
    UINT16     DeviceId;
    UINT16     SubsystemVendorId;
    UINT16     SubsystemId;
    UINT8      ClassCode[3];
    UINT8      RevisionId;
    UINT8      FirmwareRev[8];
    BOOLEAN    OptionROMBar;
    UINT8      RootPortBusNum;
    UINT8      RootPortDeviceNum;
    UINT8      RootPortFunctionNum;
    UINT8      SegmentNum;
} VROC_DEVICE_INFO_PROTOCOL;
#pragma pack()
```

### 1.2.1 Parameters

<i>SocketNumber</i>	CPU socket number where NVMe device is connected to.
<i>VmdControllerNumber</i>	Identifier of the VMD controller associated to NVMe device per each CPU. This value is enumerated for each CPU separately.
<i>RootPortNumber</i>	Number of the PCI root port where NVMe device is connected.
<i>SlotNumber</i>	NVMe device slot number.
<i>BusNumber</i>	NVMe device's PCI bus number.
<i>DeviceNumber</i>	NVMe device's PCI device number.
<i>FunctionNumber</i>	NVMe device's PCI function number.
<i>VendorId</i>	NVMe device's vendor ID, i.e. for Intel it is 0x8086
<i>SerialNumber</i>	Null terminated string with NVMe device's serial number.
<i>ModelNumber</i>	Null terminated string with NVMe device's model number.
<i>TotalBlocks</i>	Total size of NVMe device, in blocks.
<i>BlockSize</i>	Logical block size in bytes.
<i>RaidDeviceMember</i>	If TRUE, this disk is a member of RAID device, otherwise it is a pass-thru disk.
<i>RootPortOffset</i>	In case of direct attached NVMe drives, offset is a 0-based number of the slot, where the drive is attached, within a given VMD domain. In case of switch attached drives, this field will be equal to switch slot number and a <i>SlotNumber</i> field has to be used to identify the drives instead.
<i>BlockDeviceHandle</i>	EFI handle on which EFI_BLOCK_IO_PROTOCOL was installed for this device (available only for pass thru devices, for RAID members this field is NULL).
<i>DeviceId</i>	NVMe device's PCI device ID
<i>SubsystemVendorId</i>	NVMe device's Subsystem vendor ID
<i>Class Code</i>	NVMe Class Code
<i>RevisionId</i>	NVMe Firmware Revision ID
<i>Firmware Revision</i>	NVMe Firmware Version ID
<i>Option ROM BAR</i>	False when Intel VMD is enabled. This is because the Option ROM on a single NVMe SSD controller firmware will not be exposed on devices on VMD-enabled domains.
<i>RootPortBusNum</i>	Bus Number of the Intel VMD Root Port

RootPortDeviceNum	Device Number of the Intel VMD Root Port
RootPortFunctionNum	Function Number of the Intel VMD Root Port
SegmentNum	Segment Number of the Intel VMD Root Port

### 1.3 Description

This protocol is installed on each handle of NVMe device connected to VMD controller that is managed by VROC driver.

Caller should not try to deallocate or change values of the memory under returned pointer as this can lead to undefined behavior. This is VROC driver responsibility to allocate and free this memory area whenever it is necessary.

If caller service intends to keep device information i.e. for a life time of a global structure instance, a copy of interface memory should be made and protocol should be closed. This is dictated by fact, that protocol data can change in time if device enumeration is done for i.e. create or delete volume.

To calculate device capacity, TotalBlocks and BlockSize parameters can be used. Following example shows how to get disk size in bytes:

```
UINT64 sizeInBytes = result->TotalBlocks * result->BlockSize;
```

### 1.4 Example

Below code is just a sample code, it shows usage of VROC device info protocol from UEFI Shell application.

```
#include <Uefi.h>
#include <Library/UefiApplicationEntryPoint.h>
#include <Library/UefiLib.h>

/**
 VROC Private Device Protocol GUID.
 {EF104449-A5CD-44FF-9A65-150DFDA527EE}
**/
#define EFI_VROC_DEVICE_INFO_PROTOCOL_GUID \
    { 0xef104449, 0xa5cd, 0x44ff, { 0x9a, 0x65, 0x15, 0xd, 0xfd, 0xa5, 0x27, 0xee } }

/**
 VROC Private Device Protocol GUID global variable.
**/
EFI_GUID gEfiVROCDeviceInfoProtocolGuid = EFI_VROC_DEVICE_INFO_PROTOCOL_GUID;

/**
 VROC Private Device Protocol interface.
**/
#pragma pack(1)
typedef struct _VROC_DEVICE_INFO_PROTOCOL
{
    UINTN      SocketNumber;
    UINTN      VmdControllerNumber;
    UINTN      RootPortNumber;
```

```

UINT32      SlotNumber;
UINTN       BusNumber;
UINTN       DeviceNumber;
UINTN       FunctionNumber;
UINT16      VendorId;
EFI_STRING  SerialNumber;
EFI_STRING  ModelNumber;
UINT64      TotalBlocks;
UINT32      BlockSize;
BOOLEAN     RaidDeviceMember;
UINT16      RootPortOffset;
EFI_HANDLE  BlockDeviceHandle;
UINT16      DeviceId;
UINT16      SubsystemVendorId;
UINT16      SubsystemId;
UINT8       ClassCode[3];
UINT8       RevisionId;
UINT8       FirmwareRev[8];
BOOLEAN     OptionROMBar;
UINT8       RootPortBusNum;
UINT8       RootPortDeviceNum;
UINT8       RootPortFunctionNum;
UINT8       SegmentNum;

```

```

} VROC_DEVICE_INFO_PROTOCOL;
#pragma pack()

```

```
/**
```

```
Application entry point.
```

```
Arguments:
```

```
ImageHandle  The image handle.
```

```
SystemTable  The system table.
```

```
Retruns:
```

```
EFI_OUT_OF_RESOURCES  There is not enough pool memory to allocate.
```

```
EFI_INVALID_PARAMETER One of the parameters passed to EFI
                        functions has invalid value.
```

```
EFI_NOT_FOUND         No instance of VROC device info protocol was found.
```

```
EFI_SUCCESS           NVMe devices data was printed to console.
```

```
*/
```

```
EFI_STATUS EFIAPI UefiMain(
```

```
IN EFI_HANDLE ImageHandle,
```

```
IN EFI_SYSTEM_TABLE *SystemTable
```

```
)
```

```
{
```

```
EFI_HANDLE *pHandleBuf = NULL;
```

```
UINTN nHandles = 0;
```

```
UINTN nIdx = 0;
```

```
EFI_STATUS nStatus = EFI_SUCCESS;
```

```
VROC_DEVICE_INFO_PROTOCOL *result = NULL;
```

```
EFI_BOOT_SERVICES *gBS = SystemTable->BootServices;
```

```
Print(L"Searching for VROC_DEVICE_INFO_PROTOCOL instances...\n");
```

```
nStatus = gBS->LocateHandleBuffer(
```

```
ByProtocol,
```

```
&gEfiVROCDeviceInfoProtocolGuid,
```

```
NULL,
```

```

    &nHandles,
    &pHandleBuf
);

if (nStatus == EFI_OUT_OF_RESOURCES)
{
    Print(L"ERROR: There is not enough pool memory to store the matching results.\n");
    return nStatus;
}

if (nStatus == EFI_INVALID_PARAMETER)
{
    Print(L"ERROR: One of the parameters has an invalid value.\n");
    return nStatus;
}

if (nStatus == EFI_NOT_FOUND)
{
    Print(L"No instance of VROC device info protocol was found.\n");
    return nStatus;
}

Print(L"Found %d instances.\n\n", nHandles);

for (nIdx = 0; nIdx < nHandles; nIdx++)
{
    if (pHandleBuf[nIdx])
    {
        Print(L "[%d] Opening VROC_DEVICE_INFO_PROTOCOL.\n", nIdx);

        nStatus = gBS->OpenProtocol(
            pHandleBuf[nIdx],
            &gEfiVROCDeviceInfoProtocolGuid,
            (void**)&result,
            ImageHandle,
            NULL,
            EFI_OPEN_PROTOCOL_BY_HANDLE_PROTOCOL
        );

        if (nStatus == EFI_INVALID_PARAMETER)
        {
            Print(L "[%d] ERROR: One of the parameters has an invalid value.\n", nIdx);
            continue;
        }

        if (nStatus == EFI_ACCESS_DENIED || nStatus == EFI_ALREADY_STARTED)
        {
            Print(L "[%d] ERROR: Already opened with BY_DRIVER or EXCLUSIVE attribute.\n", nIdx);
            continue;
        }

        if (result)
        {
            UINT64 sizeInBytes = result->TotalBlocks * result->BlockSize;

            Print(L "[%d] %s - %s\n", nIdx, result->ModelNumber, result->SerialNumber);

            Print(L"    VMD Controller: %d\n", result->VmdControllerNumber);
            Print(L"    Socket Number: %d\n", result->SocketNumber);
            Print(L"    Slot Number: %d\n", result->SlotNumber);
            Print(L"    Root Port: %d\n", result->RootPortNumber);
            Print(L"    Root Port Offset: %d\n", result->RootPortOffset);
        }
    }
}

```



```

Print(L"    PCI Bus:Device.Function: %02x:%02x:%02x\n",
      result->BusNumber,
      result->DeviceNumber,
      result->FunctionNumber);

Print(L"    Total Blocks: %ld\n", result->TotalBlocks);
Print(L"    Block Size: %d\n", result->BlockSize);
Print(L"    Size in Bytes: %ld\n", sizeInBytes);
Print(L"    RAID Member: %s\n", result->RaidDeviceMember ? "TRUE" : "FALSE");
Print(L"    Block Device Handle: %x\n", result->BlockDeviceHandle);
Print(L"    Vendor ID: 0x%x\n", result->VendorId);
Print(L"    Device ID: 0x%x\n", result->DeviceId);
Print(L"    Subsystem Vendor ID: 0x%x\n", result->SubsystemVendorId);
Print(L"    Subsystem ID: 0x%x\n", result->SubsystemId);
Print(L"    Device Class Codes: %02x:%02x:%02x\n",
      result->ClassCode[0],
      result->ClassCode[1],
      result->ClassCode[2]);

Print(L"    Revision ID: 0x%x\n", result->RevisionId);
Print(L"    Firmware Revision: %a\n", result->FirmwareRev);
Print(L"    Option ROM Bar: %s\n", result->OptionROMBar ? "TRUE" : "FALSE");
Print(L"    RootPortBusNum: %d\n", result->RootPortBusNum);
Print(L"    RootPortDeviceNum: %d\n", result->RootPortDeviceNum);
Print(L"    RootPortFunctionNum: %d\n", result->RootPortFunctionNum);
Print(L"    SegmentNum: %d\n", result->SegmentNum);
Print(L"\n[%d] Getting device path from block device handle.\n", nIdx);
if (result->BlockDeviceHandle == NULL)
{
    Print(L "[%d] No block IO protocol associated to given handle.\n", nIdx);
}
else
{
    EFI_DEVICE_PATH_PROTOCOL * pdpDev = DevicePathFromHandle(result->BlockDeviceHandle);
    if (!pdpDev)
    {
        Print(L"ERROR: Could not get device path from handle!\n");
    }
    else
    {
        Print(L "[%d] Device path text: %s\n",
              nIdx,
              ConvertDevicePathToText(pdpDev, TRUE, TRUE)
        );
    }
}
}
else
{
    Print(L "[%d] ERROR: No interface returned.\n", nIdx);
}

Print(L "[%d] Closing VROC_DEVICE_INFO_PROTOCOL.\n\n", nIdx);

gBS->CloseProtocol(
    pHandleBuf[nIdx],
    &gEfiVROCDeviceInfoProtocolGuid,
    ImageHandle,
    NULL
);
}
}

```

```
gBS->FreePool(pHandleBuf);  
  
return EFI_SUCCESS;  
}
```

### 1.4.1 Example sample code output

```
|Searching for VROC_DEVICE_INFO_PROTOCOL instances...  
Found 4 instances.
```

```
[0] Opening VROC_DEVICE_INFO_PROTOCOL.  
[0] INTEL SSDPE2MX400G4 - CVPD514300HH400NGN  
      VMD Controller: 1  
      Socket Number: 0  
      Slot Number: 64  
      Root Port: 2  
      Root Port Offset: 0  
PCI Bus:Device.Function: 03:00.0  
      Total Blocks: 781422768  
      Block Size: 512  
      Size in Bytes: 400088457216  
      RAID Member: FALSE  
      Block Device Handle: 60D76AD8  
  
[0] Getting device path from block device handle.  
[0] Device path text: NVMe(0x1,00-00-00-00-00-00-00-01)  
      DeviceId : 0x953  
      SubsystemVendorId : 0x8086  
      SubsystemId : 0x3705  
      Device Class codes[0-3] = 0x2 : 0x8 : 0x1  
      RevisionId : 0x1  
      Firmware Revision : 8EV10174  
      OptionROMBar : 0  
      RootPortBusNum : 0  
      RootPortDeviceNum : 0  
      RootPortFunctionNum : 0  
      SegmentNum : 0  
[0] Closing VROC_DEVICE_INFO_PROTOCOL.
```