

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

Specification

Revision 1.7

November 2022

Intel Confidential



You may not use or facilitate the use of this document in connection with any infringement or other legal analysis. You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All product plans and roadmaps are subject to change without notice.

The products described 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.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

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.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries.

*Other names and brands may be claimed as the property of others.

Copyright© 2022, Intel Corporation. All rights reserved.



Contents

1 Summary 4
1.1 1.1 Protocol GUID 4
1.1.1 Protocol Interface Structure 4
1.1.2 Parameters 5
1.2 Description 6
1.3 Example 7
1.3.1 Example Sample Code and Output 14

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 } }
```

1.1.1 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;

  UINT8           VmdControllerType;

  UINT8           ServiceLabel [RSTE_OEM_SERVICE_LABEL_SZ];  / / / < service label

  UINT8           LocationType;              / / / < location type

  UINT32          LocationOrdinalValue;      / / / < ordinal value connected with location type

} VROC_DEVICE_INFO_PROTOCOL;

#pragma pack()

#define RSTE_OEM_SERVICE_LABEL_SZ 31          / / / < Size of service label including null
```

1.1.2 Parameters

SocketNumber	CPU socket number where NVMe device is connected to.
VmdControllerNumber	Identifier of the VMD controller associated to NVMe device per each CPU. This value is enumerated for each CPU separately.
RootPortNumber	Number of the PCI root port where NVMe device is connected.
SlotNumber	NVMe device slot number.
BusNumber	NVMe device's PCI bus number.
DeviceNumber	NVMe device's PCI device number.
FunctionNumber	NVMe device's PCI function number.
VendorId	NVMe device's vendor ID, i.e. for Intel it is 0x8086

SerialNumber	Null terminated string with NVMe device's serial number.
ModelNumber	Null terminated string with NVMe device's model number.
TotalBlocks	Total size of NVMe device, in blocks.
BlockSize	Logical block size in bytes.
RaidDeviceMember	If TRUE, this disk is a member of RAID device, otherwise it is a pass-thru disk.
RootPortOffset	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 *SlotNumber* field has to be used to identify the drives instead.
BlockDeviceHandle	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.
DeviceId	NVMe device's PCI device ID
SubsystemVendorId	NVMe device's Subsystem vendor ID
Class Code	NVMe Class Code
RevisionId	NVMe Firmware Revision ID
Firmware Revision	NVMe Firmware Version ID
Option ROM BAR	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.
RootPortBusNum	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

1.2 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;
```

1.3 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.%x\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;
}
```

1.3.1 Example Sample Code and 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.
```

§§