Intel® Data Streaming Accelerator User Guide

Ref#: 353216-004US
December 2025

Notices & Disclaimers

Table of Contents

Table of Contents 3
Tables 4
Figures 4
Examples 4
Revision History 5
Chapter 1 Introduction 7
1.1 Definitions 7
1.2 References 9
Chapter 2 Platform Configuration 10
2.1 BIOS Configuration 10
2.2 Linux Kernel Configuration 10
2.2.1 Intel® IOMMU Driver 10
2.2.2 Intel® DSA Driver 10
2.2.3 Generic Drivers (vfio-pci and uio_pci_generic) 12
Chapter 3 Intel® DSA Configuration 14
3.1 Intel® DSA Device Enumeration 14
3.1.1 PCI Information 14
3.1.2 sysfs Directories 17
3.2 Device Configuration and Control Interfaces 18
3.2.1 WQs, Engines, and Groups 18
3.2.2 Bandwidth Control 19
3.2.3 Read Buffer Controls 19
3.2.4 Traffic Class Configuration 20
3.2.5 Linux Interfaces 21
3.2.6 accel-config 21
3.2.7 WQ Device File Permissions 22
Chapter 4 Intel® DSA Programming 22
4.1 Sample Linux Application 23
4.1.1 Descriptor Preparation 24
4.1.2 Descriptor Submission Portal Mapping 24
4.1.3 Descriptor Submission 25
4.1.4 Completion Polling 25
4.1.5 Partial Completion Handling 26
4.2 Programming Considerations 28
4.2.1 Ordering/Fencing 28
4.2.2 Caching and Persistent Memory Controls 28
4.3 Library Support for Intel® DSA 30
Chapter 5 Intel® DSA Performance Micros 31
Chapter 6 Intel® DSA Performance Counter References 32
Chapter 7 Helper Functions for Legacy GCC Versions 33
Chapter 8 Sample C Program 35
8.1 Steps 35
8.2 The Program 35
Chapter 9 Debug Aids for Configuration Errors 40

Tables

Table 1-1 Acronym Definition 7
Table 1-2 References 9
Table 2-1 Linux Operating System Vendors Supporting Intel® DSA Drivers 12
Table 3-1 Dedicated and Shared WQ Comparison 18
Table 4-1 Software Actions for Continuation After a Page Fault 27
Table 4-2 Libraries with Support for Intel® DSA 30
Table 5-1 Intel® DSA Performance Micro (dsa-perf-micros) Links 31

Figures

Figure 3-1 Logical Organization of an Intel® DSA and Cores on an Intel® Xeon® Server Processor 14
Figure 3-2 Example Intel® DSA BW Range & Read Buffer Controls 19
Figure 3-3 Intel® DSA Device/Group/Engine/WQ Configuration and Control sysfs Entries 21
Figure 4-1 Descriptor Processing Sequence 23

Examples

Example 2-1 Linux Kernel Configuration Options for the Intel® IOMMU Driver 10
Example 2-2 Linux Kernel Configuration Options for the Intel® DSA Driver 10
Example 2-3 IDXD Driver Initialization Messages 11
Example 3-1 Listing All Intel® DSA Devices 15
Example 3-2 lspci Output for an Intel® DSA Device 15
Example 3-3 SVM Capabilities and Status 17
Example 3-4 sysfs SVM Capability 17
Example 3-5 Intel® DSA sysfs Directories 17
Example 3-6 Profiles Included in accel-config 21
Example 3-7 accel-config Command Line with WQ Configuration File 21
Example 3-8 Using accel-config to Verify Device Configuration 22
Example 3-9 Using accel-config to create a custom configuration 22
Example 3-10 Using accel-config to load a configuration 22
Example 3-11 WQ Device Files 22
Example 4-1 Descriptor Initialization 24
Example 4-2 Descriptor Submission 25
Example 4-3 Descriptor Completion Check 25
Example 4-4 Descriptor Completion Check with Pause 25
Example 4-5 UMONITOR/UMWAIT Sequence to Reduce Power Consumption While Polling 26
Example 7-1 MOVDIR64B 33
Example 7-2 ENQCMD 33
Example 7-3 UMWAIT 34
Example 7-4 UMONITOR 34
Example 8-1 Intel® DSA Shared WQ Sample Application 35
Example 9-1 Linux Kernel ACPI Subsystem Messages When VT-d is Enabled 40

Revision History

Date	Revision	Description
November 2022	001	The initial release of the document.
January 2023	002	• Reference to GitHub added (Section 3.2.3). • Replaced section of incorrect code (Appendix B Example 2).
July 2024	003	• Added Section 3.2.2 Intel® DSA Read Buffer Controls. • Updated Figure 3-2. • Changed appendices into chapters. • Updated Code in Chapter 8.
December 2025	004	• Added information about DSA Gen 3 features. • Added information about using generic drivers (section 2.2.3).

CHAPTER 1 INTRODUCTION

Intel® DSA is a high-performance data copy and transformation accelerator integrated into Intel® processors, starting with 4th Generation and later Intel® Xeon® processors. It is targeted for optimizing streaming data movement and transformation operations common with applications for high-performance storage, networking, persistent memory, and various data processing applications.

This document is for system administrators who may need to configure Intel DSA devices and developers who want to enable Intel DSA support in applications and use libraries that provide interfaces to Intel DSA. It should be read alongside the Intel® DSA Architecture Specification and documentation for software utilities and libraries that support Intel DSA, such as accel-config/libaccel-config, Libfabric, and Intel® MPI.

This document describes Intel DSA setup and configuration and includes sample code for Intel DSA. It does not include configuration mechanisms and settings specific to domain-specific software stacks such as DPDK and SPDK. For additional information, please refer to the appropriate documentation.

1.1 Definitions

Table 1-1 Acronym Definition

Acronym	Term	Definition
BIOS	Basic Input Output System	
ATS	Address Translation Service	A protocol defined by the PCI Express* specification to support address translations by a device.
IDPT	Inter-Domain Permissions Table	Table to control inter-domain operations.
IDPTE	Inter-Domain Permissions Table Entry	An entry in the IDPT.
IOMMU	I/O Memory Management Unit	A DMA Remapping Hardware Unit defined by Intel® Virtualization Technology for Directed I/O.
PCI	Peripheral Component Interconnect	A local computer bus for interconnecting peripheral devices with the processor/memory subsystems. PCI Express is a serial computer bus expansion standard designed to replace the PCI.
PRS	Page Request Service	When an endpoint determines that it requires access to a page for which the ATS translation is unavailable, it sends a Page Request message requesting that the page be mapped into system memory.
PASID	Process Address Space Identifier	A value used in memory transactions to convey the address space on the host of an address used by the device.
OS	Operating System	Allocates hardware resources (for example, processor cores, memory, and devices) to ensure optimal usage by multiple concurrent applications.

Acronym	Term	Definition
SVM	Shared Virtual Memory	An ability for an accelerator I/O device to operate in the same virtual memory space of applications on host processors. It also implies the ability to operate from pageable memory, avoiding functional requirements to pin memory for DMA operations.
QoS	Quality of Service	The ability of a system to provide predictable latency and bandwidth.
SKU	Stock-Keeping Unit	The part number or product number that identifies an item.
SR-IOV	Single-Root I/O Virtualization	A PCI Express standard for virtualizing PCI Express endpoint device interfaces.
WQ	Work queue	A queue in the device used to store descriptors.

1.2 References

Table 1-2 References

Description	URL
Intel® DSA Architecture Specification	https://software.intel.com/content/www/us/en/develop/articles/intel-data-streaming-accelerator-architecture-specification.html
Intel® DSA Perf Micros	https://github.com/intel/dsa-perf-micros
Intel® DSA Perfmon Support	https://github.com/intel/dsa-perf-micros/wiki/DSA-Perfmon
Intel® Data Movement Library	https://intel.github.io/DML
PCI Express* Base Specification 4.0	http://www.pcisig.com/specifications/pciexpress
Intel® Virtualization Technology for Directed I/O (Intel® VT-d or Intel® IOMMU)	https://software.intel.com/content/www/us/en/develop/download/intel-virtualization-technology-for-directed-io-architecture-specification.html
Intel® MPI	https://www.intel.com/content/www/us/en/developer/tools/oneapi/mpi-library.html
DPDK IDXD DMADEV driver	https://doc.dpdk.org/guides/dmadevs/idxd.html
SPDK IDXD driver	https://spdk.io/doc/idxd.html
accel-config and libaccel-config	https://github.com/intel/idxd-config
Libfabric	https://github.com/ofiwg/libfabric/blob/main/man/fi_shm.7.md

CHAPTER 2 PLATFORM CONFIGURATION

2.1 BIOS Configuration

This document primarily focuses on Intel® DSA usage with user space memory, requiring Intel® Virtualization Technology for Directed I/O (VT-d) to be enabled. As described in the Address Translation section in the Intel® DSA Architecture Specification, DSA may also be used with physical or other virtual addresses (for example, I/O virtual address, IOVA). While a detailed discussion of this is beyond the scope of this document, Section 2.2.3 provides a brief description and some related pointers.

2.2 Linux Kernel Configuration

2.2.1 Intel® IOMMU Driver

The Intel® IOMMU driver with scalable mode support (CONFIG_INTEL_IOMMU_SVM) must be enabled in the kernel configuration, as shown in Example 2-1. If either the CONFIG_INTEL_IOMMU_DEFAULT_ON or the CONFIG_INTEL_IOMMU_SCALABLE_MODE_DEFAULT_ON options are not enabled, then “intel_iommu=on,sm_on" must be added to the kernel boot parameters.

Example 2-1 Linux Kernel Configuration Options for the Intel® IOMMU Driver

```
CONFIG_INTEL_IOMMU=y CONFIG_INTEL_IOMMU_SVM=y

CONFIG_INTEL_IOMMU_DEFAULT_ON=y

CONFIG_INTEL_IOMMU_SCALABLE_MODE_DEFAULT_ON=y
```

2.2.2 Intel® DSA Driver

When building/installing the Linux kernel, enable the kernel configuration options shown in Example 2-2.

Example 2-2 Linux Kernel Configuration Options for the Intel® DSA Driver

```
CONFIG_INTEL_IDXD=m

CONFIG_INTEL_IDXD_SVM=y

CONFIG_INTEL_IDXD_PERFMON=y
```

Work queues (WQs) are on-device storage to contain descriptors submitted to the device and can be configured to run in either of two modes:

- Dedicated (DWQ), or
- Shared (SWQ).

A SWQ allows multiple clients to submit descriptors simultaneously without the software overhead of synchronization needed to track WQ occupancy. SWQ is the preferred WQ mode since it offers better device utilization versus hard partitioning with DWQs, which may result in underutilization. The Intel® DSA Driver (IDXD) with DWQ support was introduced in kernel version 5.6. The IDXD driver with SWQ support is available in Linux upstream Kernel versions 5.18 and beyond.

IDXD driver initialization can be checked using the **dmesg** command, which prints the kernel message buffer, as shown in Example 2-3.

Example 2-3 IDXD Driver Initialization Messages

```
$ dmesg -t | grep "idxd"
idxd 0000:6a:01.0: enabling device (0144 -> 0146)
idxd 0000:6a:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:6a:02.0: enabling device (0140 -> 0142)
idxd 0000:6a:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:6f:01.0: enabling device (0144 -> 0146)
idxd 0000:6f:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:6f:02.0: enabling device (0140 -> 0142)
idxd 0000:6f:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:74:01.0: enabling device (0144 -> 0146)
idxd 0000:74:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:74:02.0: enabling device (0140 -> 0142)
idxd 0000:74:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:79:01.0: enabling device (0144 -> 0146)
idxd 0000:79:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:79:02.0: enabling device (0140 -> 0142)
idxd 0000:79:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:e7:01.0: enabling device (0144 -> 0146)
idxd 0000:e7:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:e7:02.0: enabling device (0140 -> 0142)
idxd 0000:e7:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:ec:01.0: enabling device (0144 -> 0146)
idxd 0000:ec:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:ec:02.0: enabling device (0140 -> 0142)
idxd 0000:ec:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:f1:01.0: enabling device (0144 -> 0146)
idxd 0000:f1:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:f1:02.0: enabling device (0140 -> 0142)
idxd 0000:f1:02.0: Intel(R) Accelerator Device (v100)
idxd 0000:f6:01.0: enabling device (0144 -> 0146)
idxd 0000:f6:01.0: Intel(R) Accelerator Device (v100)
idxd 0000:f6:02.0: enabling device (0140 -> 0142)
idxd 0000:f6:02.0: Intel(R) Accelerator Device (v100)
```

Table 2-1 shows the distribution kernel versions with complete IDXD driver support. For the latest information, please refer to vendor documentation.

Table 2-1 Linux Operating System Vendors Supporting Intel® DSA Drivers

Vendor	Intel® DSA Driver
SUSE Linux Enterprise Server	SLES 15 SP4
Red Hat Enterprise Linux	RHEL 8.7 & 9.1
Ubuntu	Ubuntu 22.10

2.2.3 Generic Drivers (vfio-pci and uio_pci_generic)

Intel DSA can be configured for use with a generic driver such as vfio-pci or uio_pci_generic. This allows using Intel DSA without SVM.

When operating in this mode, it is the application's responsibility to use appropriate driver APIs to register and pin memory regions before initiating any DMA operations. Details of the APIs are beyond the scope of this document. The documentation for DPDK and SPDK contain examples of such uses.

To configure DSA for use with vfio or uio_pci_generic, use the following steps:

1. The DSA device must be detached from the idxd driver. To do this, either boot the system with the command line parameter “modprobe.blacklist=idxd” or use the command
   ```
   # rmmod idxd
   ```
2. For uio_pci_generic: Boot the system with the command line parameter “intel_iommu=off”.
3. List the DSA devices to get their pci addresses. The list may look like this:
   ```
   $ lspci | grep 0b25
   6a:01.0 System peripheral: Intel Corporation Device 0b25
   6f:01.0 System peripheral: Intel Corporation Device 0b25
   74:01.0 System peripheral: Intel Corporation Device 0b25
   79:01.0 System peripheral: Intel Corporation Device 0b25
   e7:01.0 System peripheral: Intel Corporation Device 0b25
   ec:01.0 System peripheral: Intel Corporation Device 0b25
   f1:01.0 System peripheral: Intel Corporation Device 0b25
   f6:01.0 System peripheral: Intel Corporation Device 0b25
   ```
4. Install the driver you want to use with one of these commands:*
   ```
   # modprobe vfio-pci
   # modprobe uio_pci_generic
   ```
5. Attach the DSA devices to the desired driver with one of these commands:
   ```
   # echo "8086 0b25" >  /sys/bus/pci/drivers/vfio-pci/new_id
   # echo "8086 0b25" >  /sys/bus/pci/drivers/uio_pci_generic/new_id
   ```
6. Now DSA is ready to be used with the selected driver. Check that the devices are attached to the desired driver with the following command, using any of the PCI device ids from step 3. The devices will not be listed under /dev/dsa.
   ```
   $ ls -l /sys/bus/pci/devices/*/driver | grep 6a:01
   ```
7. Install librte_eal, which comes with DPDK.

* Some versions of Intel DSA must not be used with untrusted software. In this case, the DSA device appears in the the vfio-pci driver denylist to prevent assignment of the Intel DSA device to the vfio-pci driver. If you intend to run trusted software, you can use the parameter ‘disable_denylist=1’ on the modprobe command for vfio-pci.

8. To use the DSA Performance Micros, export the following variable. Change the librte_eal version in this command to match the one present in your system.
   $ export DSA_PERF_MICROS_EAL_PATH=/usr/lib64/librte_eal.so.23

CHAPTER 3 INTEL® DSA CONFIGURATION

This section describes how Intel® DSA devices and WQs can be configured and enabled by a superuser before running an application that uses Intel DSA. Before describing the configuration process, Linux OS structures for Intel DSA are described to help debug configuration issues.

3.1 Intel® DSA Device Enumeration

3.1.1 PCI Information

Figure 3-1 shows the logical organization of an Intel DSA on Intel® Xeon® server processor. Depending on the processor SKU, between one and four Intel DSA devices exist per socket. A system with two sockets can have up to eight Intel DSA devices.

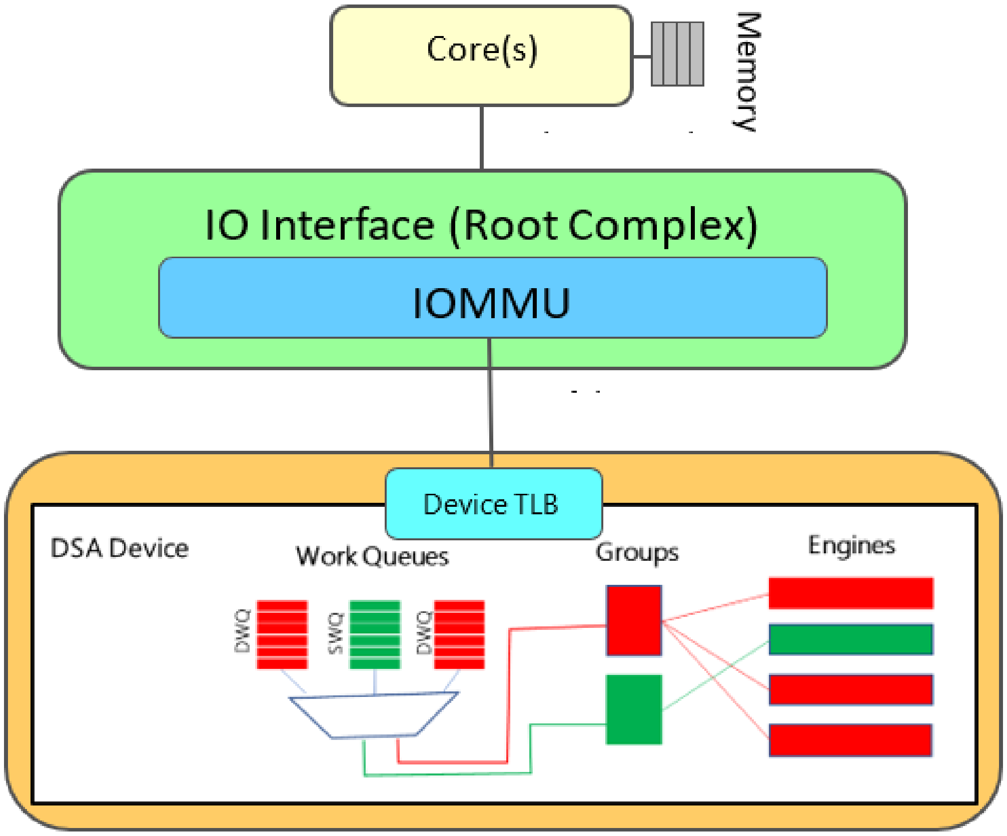

Figure 3-1 Logical Organization of an Intel® DSA and Cores on an Intel® Xeon® Server Processor

Intel DSA PCI device ID is **0x0b25** or **0x1212**. The following command lists the Intel DSA devices on the system:

Example 3-1 Listing All Intel® DSA Devices

```
$ lspci | egrep '0b25|1212'
6a:01.0 System peripheral: Intel Corporation Device 0b25
6f:01.0 System peripheral: Intel Corporation Device 0b25
74:01.0 System peripheral: Intel Corporation Device 0b25
79:01.0 System peripheral: Intel Corporation Device 0b25
e7:01.0 System peripheral: Intel Corporation Device 0b25
ec:01.0 System peripheral: Intel Corporation Device 0b25
f1:01.0 System peripheral: Intel Corporation Device 0b25
f6:01.0 System peripheral: Intel Corporation Device 0b25
```

The complete lspci output for an Intel DSA device can be obtained, as shown in Example 3-2. If the **Kernel driver in use** field within the lspci output is blank, use the **modprobe idxd** command to load the driver.

Example 3-2 lspci Output for an Intel® DSA Device

```
# lspci -vvv -s 6a:01.0
6a:01.0 System peripheral: Intel Corporation Device 0b25
        Subsystem: Intel Corporation Device 0000
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr+ Stepping- SERR+
FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort-
>SERR- <PERR- INTx-
        Latency: 0
        NUMA node: 0
        Region 0: Memory at 206ffff60000 (64-bit, prefetchable) [size=64K]
        Region 2: Memory at 206ffff00000 (64-bit, prefetchable) [size=128K]
        Capabilities: [40] Express (v2) Root Complex Integrated Endpoint, MSI 00
                DevCap: MaxPayload 512 bytes, PhantFunc 0
                        ExtTag+ RBE+ FLReset+
                DevCtl: CorrErr+ NonFatalErr+ FatalErr+ UnsupReq-
                        RlxdOrd+ ExtTag+ PhantFunc- AuxPwr- NoSnoop+ FLReset-
                        MaxPayload 128 bytes, MaxReadReq 4096 bytes
                DevSta: CorrErr- NonFatalErr- FatalErr- UnsupReq- AuxPwr- TransPend-
                DevCap2: Completion Timeout: Not Supported, TimeoutDis+, NROPrPrP-, LTR+
                        10BitTagComp+, 10BitTagReq+, OBFF Not Supported, ExtFmt+,
                        EETLPPrefix+, MaxEETLPPrefixes 1 EmergencyPowerReduction Not
                        Supported, EmergencyPowerReductionInit-
        Capabilities: [80] MSI-X: Enable+ Count=9 Masked-
                Vector table: BAR=0 offset=00002000
                PBA: BAR=0 offset=00003000
        Capabilities: [90] Power Management version 3
                Flags: PMEClk- DSI- D1- D2- AuxCurrent=0mA PME(D0-,D1-,D2-,D3hot-,D3cold-)
                Status: D0 NoSoftRst+ PME-Enable- DSel=0 DScale=0 PME-
        Capabilities: [100 v2] Advanced Error Reporting
                UESta: DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF-
        MalfTLP- ECRC- UnsupReq- ACSViol-
                UEMsk: DLP- SDES- TLP- FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF-
        MalfTLP- ECRC- UnsupReq+ ACSViol-
                UESvrt: DLP- SDES- TLP+ FCP- CmpltTO- CmpltAbrt- UnxCmplt- RxOF-
        MalfTLP+ ECRC- UnsupReq- ACSViol-
                CESta: RxErr- BadTLP- BadDLLP- Rollover- Timeout- AdvNonFatalErr-
                CEMsk: RxErr- BadTLP- BadDLLP- Rollover- Timeout- AdvNonFatalErr-
                AERCap: First Error Pointer: 00, ECRCGenCap- ECRCGenEn- ECRCChkCap-
                ECRCChkEn- MultHdrRecCap- MultHdrRecEn- TLPPfxPres- HdrLogCap-
                HeaderLog: 00000000 00000000 00000000 00000000
        Capabilities: [150 v1] Latency Tolerance Reporting
                Max snoop latency: 0ns
                Max no snoop latency: 0ns
        Capabilities: [160 v1] Transaction Processing Hints
                Device-specific mode supported
                Steering table in TPH capability structure
        Capabilities: [170 v1] Virtual Channel
                Caps:   LPEVC=1 RefClk=100ns PATEntryBits=1
                Arb:    Fixed+ WRR32- WRR64- WRR128-
                Ctrl:   ArbSelect=Fixed Status: InProgress-
                VC0:    Caps:  PATOffset=00 MaxTimeSlots=1 RejSnoopTrans-
                        Arb:   Fixed- WRR32- WRR64- WRR128- TWRR128- WRR256-
                        Ctrl:  Enable+ ID=0 ArbSelect=Fixed TC/VC=fd
                        Status: NegoPending- InProgress-
                VC1:    Caps:  PATOffset=00 MaxTimeSlots=1 RejSnoopTrans-
                        Arb:   Fixed- WRR32- WRR64- WRR128- TWRR128- WRR256-
                        Ctrl:  Enable+ ID=1 ArbSelect=Fixed TC/VC=02
                        Status: NegoPending- InProgress-
        Capabilities: [200 v1] Designated Vendor-Specific <?>
        Capabilities: [220 v1] Address Translation Service (ATS)
                ATSCap: Invalidate Queue Depth: 00
                ATSCtl: Enable+, Smallest Translation Unit: 00
        Capabilities: [230 v1] Process Address Space ID (PASID)
                PASIDCap: Exec- Priv+, Max PASID Width: 14
                PASIDCtl: Enable+ Exec- Priv+
        Capabilities: [240 v1] Page Request Interface (PRI)
                PRICtl: Enable+ Reset-
                PRISta: RF- UPRGI- Stopped+
                Page Request Capacity: 00000200, Page Request Allocation: 00000020
         Kernel driver in use: idxd
         Kernel modules: idxd
```

Shared Virtual Memory (SVM) is a usage where a device operates in the CPU virtual address space of the application accessing the device. Devices supporting SVM do not require pages accessed by the device to be pinned. Instead, they use the PCI Express Address Translation Services (ATS) and Page Request Services (PRS) capabilities to implement recoverable device page faults. Devices supporting SVM use PASIDs to distinguish different application virtual address spaces.

PCIe capabilities and status related to SVM – ATSCtl, PASIDCtl, and PRICtl are enabled, as shown in Example 3-3. Refer to the Address Translation section in the Intel® DSA Architecture Specification for further details on how Intel DSA utilizes the PASID, PCIe, ATS, and PRS capabilities to support SVM.

Example 3-3 SVM Capabilities and Status

```
Capabilities: [220 v1] Address Translation Service (ATS)
              ATSCap: Invalidate Queue Depth: 00
              ATSCtl: Enable+, Smallest Translation Unit: 00
Capabilities: [230 v1] Process Address Space ID (PASID)
              PASIDCap: Exec- Priv+, Max PASID Width: 14
              PASIDCtl: Enable+ Exec- Priv+
Capabilities: [240 v1] Page Request Interface (PRI)
              PRICtl: Enable+ Reset-
              PRISta: RF- UPRGI- Stopped+
              Page Request Capacity: 00000200, Page Request Allocation:
              00000020
```

SVM capability is also available in sysfs.

Example 3-4 sysfs SVM Capability

```
$ cat /sys/bus/dsa/devices/dsa0/pasid_enabled
1
```

3.1.2 sysfs Directories

The Linux sysfs file system is a pseudo-file system that provides an interface to kernel data structures. The files under sysfs provide information about devices, kernel modules, file systems, and other kernel components.

The Linux driver generates the sysfs directories shown in Example 3-5 for an example dual-socket system with eight total Intel DSA devices (four devices per processor). The Intel DSA and Intel® In-Memory Analytics Accelerator (Intel® IAA) devices are both managed by the IDXD device driver. The numbering of Intel DSA and Intel IAA devices depends on the number of each device in the CPU SKU. In the dual-socket example below, four Intel IAA devices are present per socket. They are named iax{1,3,5,7,9,11,13,15}. Correspondingly, the Intel DSA devices are named dsa{0,2,4,6,8,10,12,14}.

Example 3-5 Intel® DSA sysfs Directories

```
$ ls -df /sys/bus/dsa/devices/dsa*
/sys/bus/dsa/devices/dsa0 /sys/bus/dsa/devices/dsa2
/sys/bus/dsa/devices/dsa10  /sys/bus/dsa/devices/dsa4
/sys/bus/dsa/devices/dsa12  /sys/bus/dsa/devices/dsa6
/sys/bus/dsa/devices/dsa14  /sys/bus/dsa/devices/dsa8
```

3.2 Device Configuration and Control Interfaces

The Intel DSA device is configured through entries in the sysfs filesystem. After configuration, the device and WQ can be enabled. The driver creates a **/dev/dsa/wqD.Q** device file for every enabled WQ, where D is the Intel DSA ID, and Q is the WQ ID. The application uses the WQ device file to submit work to the hardware, as described in Section 4.1.

3.2.1 WQs, Engines, and Groups

Software specifies work for the device by constructing descriptors in memory and submitting them to the WQ. Shared WQs allow multiple clients to submit descriptors concurrently and are recommended for application use.

All descriptors submitted to a DWQ use a single address space, so a DWQ can be used when a single OS-level process uses the WQ. Dedicated WQs require SW to manage flow control by keeping track of descriptors submitted and completed to ensure the WQ capacity is not overrun.

An engine is an operational unit within an Intel DSA device. A group is a logical organization of a set of WQs and engines to achieve a specific QoS objective. Multiple groups can provide performance isolation between applications sharing the device. Refer to sections 3.3 and 3.4 in the Intel® DSA Architecture Specification for more details on Intel DSA WQ, Engine, and Group capabilities and controls.

With current implementations, the memory bandwidth achievable by a given Intel DSA instance for data sizes of 4KB or larger may be realized with only a single engine within the device. In such cases, provisioning additional engines in the same Intel DSA instance may not offer additional performance benefits. However, using additional engines may be useful to improve the quality of service (QoS) within the Intel DSA instance. A user may achieve higher aggregate performance as compared to a single Intel DSA instance using multiple Intel DSA instances.

Table 3-1 Dedicated and Shared WQ Comparison

	Shared WQ	**Dedicated WQ**
Client Scaling	Can scale to a large number of clients (e.g., process, VM, container) with simultaneous SW-lock-free work submitted to the same WQ. This is the preferred interface for all work submissions, especially when sharing the device across multiple clients/users.	A limited number of DWQs intended only for specific vertical uses (e.g., networking and storage infrastructure).
WQ Sharing	Software does not need to keep track of outstanding descriptors. SW can use the ENQCMD result to identify successful vs. unsuccessful descriptor submissions.	Each DWQ uses a single address space. SW must track the number of outstanding descriptors to avoid queue overflow.
Submission Rate	Rate of descriptor submission limited by ENQCMD round-trip latency (approx. 200-250 ns on 4th Generation Intel® Xeon®).	Software can stream descriptors to the device at a high rate using MOVDIR64B. If descriptor ordering is not a requirement, SW can use different portal addresses to avoid serializing writes to the same address. Refer to the Ordering and Fencing section in the Intel® DSA Architecture Specification.

3.2.2 Bandwidth Control

Some Intel® DSA implementations provide software controls to limit the maximum read and write bandwidth per group. When supported, the Read Bandwidth Limit and Write Bandwidth Limit fields in GRPFLAGS limit the maximum read and write bandwidth for that group. The limits are expressed as a fraction of the maximum value supported by the device implementation. Hardware throttles an engine when the bandwidth utilization of the group equals or exceeds the configured maximum.

This can be used by system software to ensure that applications or guests submitting work to that group don't use more bandwidth than has been allocated to them.

3.2.3 Read Buffer Controls

Intel DSA uses read buffers to manage memory read latency. For implementations that do not implement Bandwidth Control (section 3.2.2), system software may utilize the read buffer controls to limit read bandwidth. Read buffer controls can also be used to reserve a minimum amount of read bandwidth for a group.

The group configuration registers in the device provide two fields to control the number of Read Buffers that are available to be used by each group.

- **Read Buffers Allowed:** The value in this field indicates the maximum number of Read Buffers that may be in use at one time by all engines in the group. The default value is the total number of read buffers, meaning that there is no limitation. A smaller value in this field can limit the read bandwidth usable by this group and also by the device as a whole.

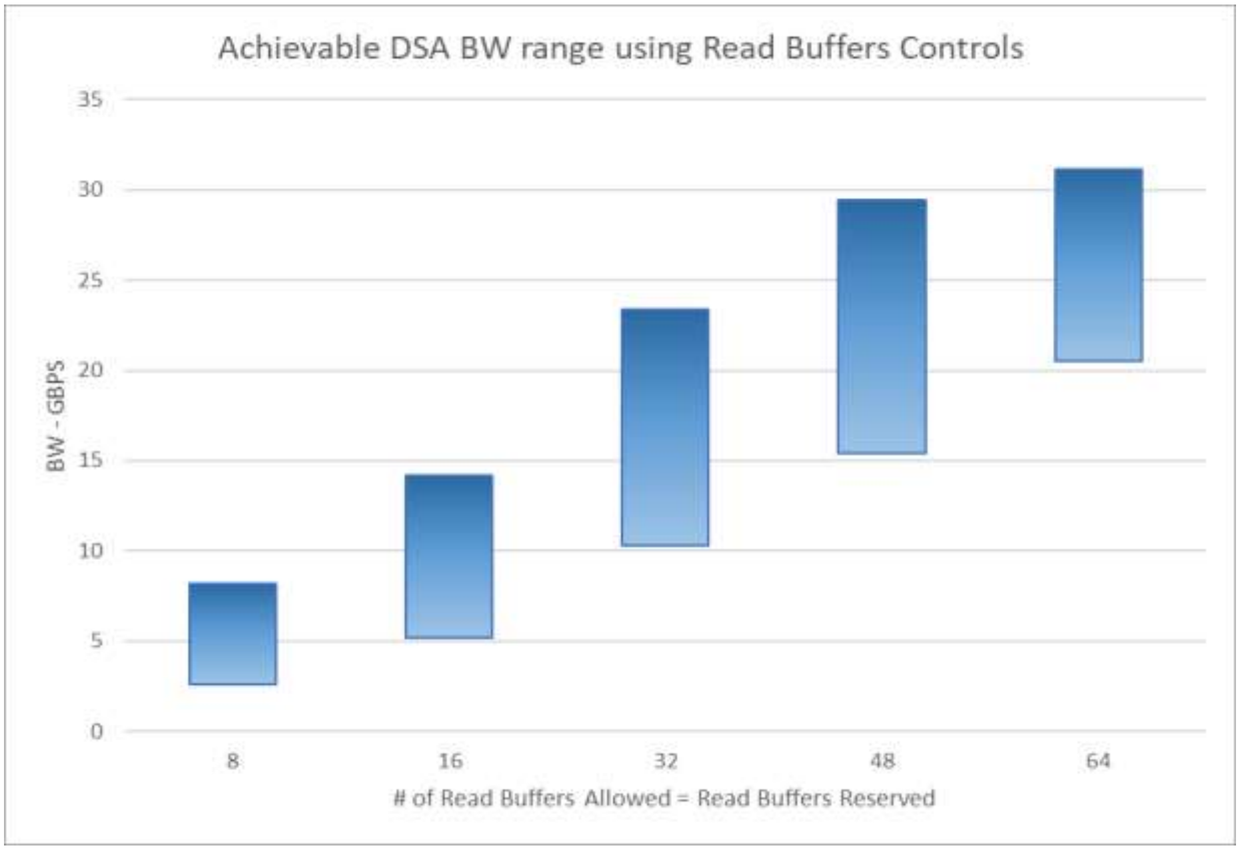

Figure 3-2 Example Intel® DSA BW Range & Read Buffer Controls

- **Read Buffers Reserved:** A non-zero value for the Read Buffers Reserved field ensures a minimum number of read buffers available to the engines in the group and allows a guaranteed read bandwidth, but it may restrict the bandwidth usable by other groups, even when the engines in this group are idle. The default value for this field is 0.

The amount of memory read latency associated with each read buffer varies depending on the system load. As a result, the read bandwidth achieved with specific values of Read Buffers Allowed and Read Buffers Reserved falls within a range of values. An example with equal values for Read Buffers Reserved and Read Buffers Allowed for a first generation Intel DSA implementation is shown in Figure 3-2.

See Section 4.3 in the Intel® DSA Architecture Specification for more information on programming Read Buffers Allowed and Read Buffers Reserved controls.

3.2.4 Traffic Class Configuration

Traffic classes are designed to segregate traffic with different QoS characteristics. For example, data being written to memory with lower bandwidth and/or higher latency should be segregated from data being written to main memory, to prevent the higher performance data from being blocked behind the lower performance data. Sharing a single virtual channel for DMA traffic to both high-bandwidth and low-bandwidth memory targets may impact throughput.

Intel® DSA devices support two Virtual Channels, VC0 and VC1, and eight Traffic Classes, 0 through 7. Traffic Class 0 is always mapped to VC0. The other traffic classes can be mapped to either VC0 or VC1. The mapping of TCs to VCs is controlled by system software by programming the TC/VC Map field in the PCIe Virtual Channel capability.

System software controls the TC values available in a group by programming the TC-A (Bits 2:0) and TC-B (Bits 5:3) fields in the device Group Configuration Table GRPFLAGS register. Application software selects which of the two TCs to use by setting flags in the descriptor.

First-generation Intel® DSA devices on 4th and 5th Generation Intel® Xeon® Scalable Processors are limited to using a single virtual channel for DMA traffic. For these implementations, DMA traffic should always use Virtual Channel 1 (VC1), because VC0 is used for descriptor submissions to the device. System software should program both the TC-A and TC-B fields to non-zero TC values that are mapped to VC1. Intel® DSA devices later than the first generation do not have this restriction, so software can use Traffic Classes mapped to either Virtual Channel 0 or Virtual Channel 1 as described above.

The Linux driver by default follows the guidelines above for the respective device versions. This default behavior can be overridden by passing the **tc_override** parameter to the driver, and the user can customize traffic class configuration via the **traffic_class_a** and **traffic_class_b** entries within the per-group sysfs directory.

3.2.5 Linux Interfaces

Intel DSA Groups, Engines, and WQs are configured using sysfs entries created by the IDXD driver.

Figure 3-3 shows the configuration and control sysfs entries for the dsa0 device. The DSA device is configured by a superuser. An active group must have at least one WQ and one Engine assigned to it. (A group is inactive if it is assigned no WQs or Engines.) The configuration using the accel-config utility is described in Section 3.2.6. An Intel DSA device and associated WQs may only be enabled after configuring the Groups, Engines, and at least the size of the WQs.

```
$ ls /sys/bus/dsa/devices/dsa0
cdev_major       group0.0            max_work_queues       uevent
clients          group0.1            max_work_queues_size  version
cmd_status       group0.2            numa_node             wq0.0
configurable     group0.3            op_cap                wq0.1
engine0.0        max_batch_size      pasid_enabled         wq0.2
engine0.1        max_engines         power                 wq0.3
engine0.2        max_groups          read_buffer_limit     wq0.4
engine0.3        max_read_buffers    state                 wq0.5
errors           max_tokens          subsystem             wq0.6
gen_cap          max_transfer_size   token_limit           wq0.7
```

Figure 3-3 Intel® DSA Device/Group/Engine/WQ Configuration and Control sysfs Entries

3.2.6 accel-config

The program **accel-config** is a Linux application that provides a command line interface for Intel DSA configuration. The program comes with a shared library (libaccel-config.so) that applications can use to query and modify Intel DSA configuration. **accel-config** can be used with text-based configuration files. Recommended configurations for a few use cases are included in the accel-config installation. **app_profile.conf** is a configuration intended for user space applications and provides two groups with one SWQ and one engine each.

Example 3-6 Profiles Included in accel-config

```
$ cd idxd-config/contrib/configs/ && ls *.conf
app_profile.conf net_profile.conf os_profile.conf storage_profile.conf
```

The WQs are configured so that applications desiring to use Intel DSA for operations with a relatively small memory footprint can submit descriptors to the WQ with a smaller value of maximum transfer size configured for that WQ. This avoids head-of-line blocking, i.e., prevents these operations from queuing. Example 3-7 shows how to configure and enable WQs using **app_profile.conf**. A super-user must execute this command since only a super-user can modify sysfs entries.

Example 3-7 accel-config Command Line with WQ Configuration File

```
$ accel-config load-config -c contrib/configs/app_profile.conf -e
Enabling device dsa0
Enabling wq wq0.1
Enabling wq wq0.0
```

accel-config can show the current configuration using the list command, as shown in Example 3-8.

Example 3-8 Using accel-config to Verify Device Configuration

```
$ accel-config list
```

Example 3-9 Using accel-config to create a custom configuration

This example shows using accel-config with command-line parameters to enable Intel DSA with a custom configuration and save the configuration to a file. The configuration values used are examples and may vary depending on device and/or driver capabilities.

1. Configure the attributes of the device. This example sets all attributes to default values.

```
$ accel-config config-device dsa0
```

2. Configure a group by configuring an engine and WQ.

```
$ accel-config config-engine dsa0/engine0.0 --group-id=0
$ accel-config config-wq dsa0/wq0.0 --group-id=0 --wq-size=32 --priority=1 \
--block-on-fault=0 --threshold=4 --type=user --name=swq --mode=shared \
--max-batch-size=32 --max-transfer-size=2097152
```

3. Enable the device and WQ.

```
$ accel-config enable-device dsa0
$ accel-config enable-wq dsa0/wq0.0
```

4. Save the configuration to a file. This saves the configuration of all device instances in the system.

```
$ accel-config save-config -s save_config.conf
```

Example 3-10 Using accel-config to load a configuration

```
$ accel-config load-config -c save_config.conf
```

3.2.7 WQ Device File Permissions

The idxd driver creates WQ device files, as seen in Example 3-11.

Example 3-11 WQ Device Files

```
$ ls -la /dev/dsa/wq0.0 /dev/dsa/wq0.1
crw------- 1 root root 240, 1 Oct 5 11:58 /dev/dsa/wq0.0
crw------- 1 root root 240, 0 Oct 5 11:58 /dev/dsa/wq0.1
```

The super-user must grant read-write permissions to the device file to the user/group under which the process runs.

CHAPTER 4 INTEL® DSA PROGRAMMING

A user can initiate an application that uses Intel® DSA, but only after the superuser configures an Intel DSA device with at least one associated WQ. Additionally, the superuser must enable the user's access to the WQ character device file. The commands to configure the device and a shared WQ are described in Section 3.2.

This section provides walk-through C program snippets illustrating the steps needed to use Intel DSA. Chapter 8 provides a complete source code listing for a C program that uses Intel DSA.

4.1 Sample Linux Application

Figure 4-1 shows the steps from descriptor preparation to descriptor completion. Each step is discussed in further detail within respective sub-sections.

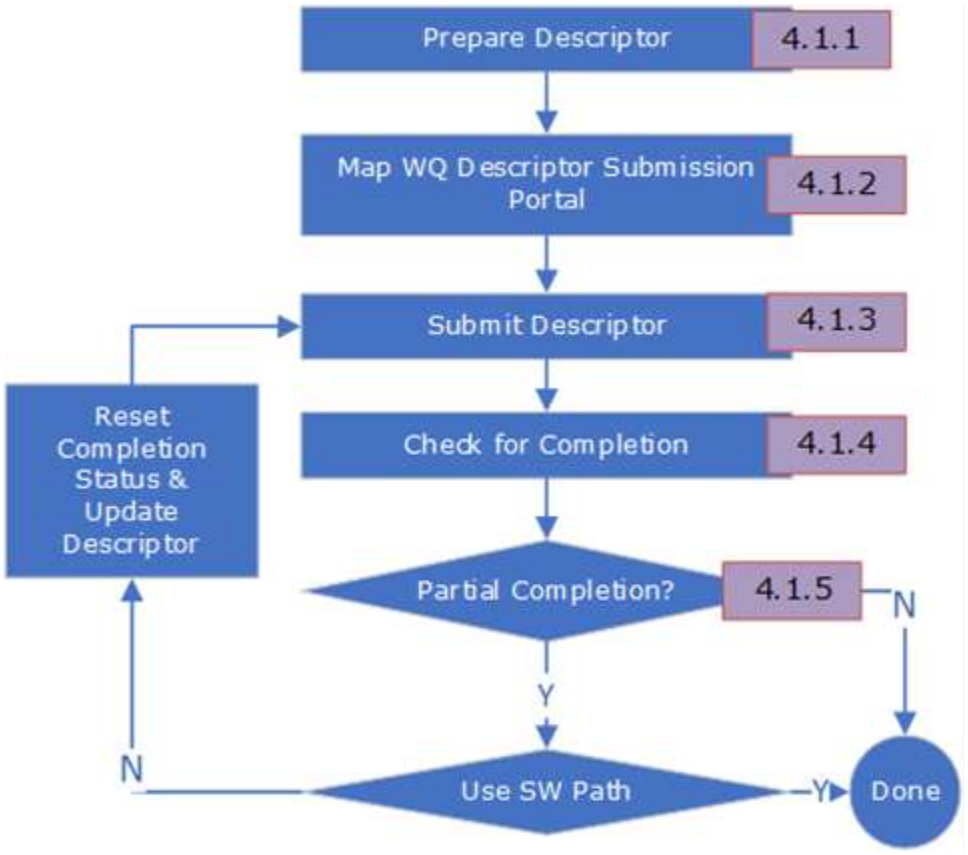

Figure 4-1 Descriptor Processing Sequence

4.1.1 Descriptor Preparation

Sample code to prepare a Memory Move descriptor is shown in Example 4-1. Since the Block On Fault flag is zero, if a descriptor incurs a page fault on either source or destination addresses, the operation status code indicates that the operation has completed with a page fault. The number of bytes transferred for the memmove operation is provided in the completion record. For details on the Block on Fault flag, please refer to Section 4.1.5 .

Example 4-1 Descriptor Initialization

```
struct dsa_completion_record comp __attribute__((aligned(32)));
struct dsa_hw_desc desc = { };
desc.opcode = DSA_OPCODE_MEMMOVE;

/*
 * Request a completion – since we poll on status, this flag
 * must be 1 for status to be updated on successful
 * completion
 */
desc.flags = IDXD_OP_FLAG_RCR;     /* Request Completion Record */

/* CRAV should be 1 since RCR = 1 */
desc.flags |= IDXD_OP_FLAG_CRAV;  /* Completion Record Address Valid */

/* Hint to direct data writes to CPU cache */
desc.flags |= IDXD_OP_FLAG_CC;    /* Cache Control 1 */

desc.xfer_size = BLEN;            /* Buffer length */
desc.src_addr = (uintptr_t)src;
desc.dst_addr = (uintptr_t)dst;
comp.status = 0;
desc.completion_addr = (uintptr_t)&comp;
```

4.1.2 Descriptor Submission Portal Mapping

Before submitting a descriptor to the device, an application must open a previously configured WQ device file (for example, **/dev/dsa/wq0.0**) and map into its address space (the work submission portal on that WQ). The portal may now be used to submit descriptors to the device.

A shared WQ device file can be opened by multiple processes concurrently, whereas only a single process can open a dedicated WQ device file at any given time. The map_wq() function in Chapter 8 shows the use of accel-config library functions to enumerate WQs and select an enabled WQ of the desired type.

As a mitigation for a potential security vulnerability in Intel® Data Streaming Accelerator (Intel® DSA) and Intel® Analytics Accelerator (Intel® IAA) in some Intel® Xeon® processors, Intel has published guidance and software updates.[1] The OS may disallow an unprivileged process from mapping the work submission portal into its address space. Specifically, Linux divides superuser privileges into “capabilities” that can be independently controlled. Only a process with the **CAP_SYS_RAW_IO** capability can map the work submission portal into its address space.

For unprivileged applications, the Linux device driver provides the write system call to submit descriptors. The security issue also requires the write system call to disallow batch descriptor submission. The sample program in

[1] Please see the Intel® DSA and Intel® IAA Advisory https://www.intel.com/content/www/us/en/security-center/advisory/intel-sa-01084.html.

Chapter 8 also shows how to check for the presence of WQ **mmap** support in the driver and how to use the write system call.

4.1.3 Descriptor Submission

Depending on the WQ type, the software may use the ENQCMD or MOVDIR64B instruction for descriptor submission. The Shared Work Queue section in the Intel® DSA Architecture Specification describes ENQCMD returning a non-zero value if the descriptor is not accepted into the device. **gcc10** supports the **enqcmd()** and **_movdir64b()** intrinsics for **ENQCMD** and **MOVDIR64B**, respectively, via the **-menqcmd** and **-mmovdir64b** switches; for older compiler versions, equivalent code is shown in Chapter 7.

Since MOVDIR64B and ENQCMD are not ordered relative to older stores in WB or WC memory, SW must ensure appropriate ordering (when required) by executing a fencing instruction such as SFENCE, preferably using a single fence for multiple updates to reduce the fencing instruction overhead.

Example 4-2 Descriptor Submission

```
#include <x86gprintrin.h>
_mm_sfence();
if (dedicated)
   _movdir64b(wq_portal, &desc);
else {
   retry = 0;
   while (_enqcmd(wq_portal, &desc) && retry++ < ENQ_RETRY_MAX);
}
```

4.1.4 Completion Polling

The Intel DSA hardware updates the status field of the completion record when it is done processing the descriptor. The completion check is shown in Example 4-3.

Example 4-3 Descriptor Completion Check

```
 retry = 0;
 while (comp.status == 0 && retry++ < COMP_RETRY_MAX);
 if (comp.status == DSA_COMP_SUCCESS) {
     /* Successful completion */
 } else {
    /* Descriptor failed or timed out
     * See the "Error Codes" section of the Intel® DSA Architecture Specification
     * for error code descriptions
     */
}
```

A pause instruction should be added to the spin loop to reduce the power a processor consumes.

Example 4-4 Descriptor Completion Check with Pause

```
#include <x86gprintrin.h>
retry = 0;
while (comp.status == 0 && retry++ < COMP_RETRY_MAX)
```

```
     __mm_pause();
```

Further power reduction can be achieved using the UMONITOR/UMWAIT instruction sequence.

UMONITOR provides an address, informing that the currently running application is interested in any writes to a range of memory (the range that the monitoring hardware checks for store operations can be determined using the CPUID monitor leaf function, EAX=05H).

UMWAIT instructs the processor to enter an implementation-dependent optimized state while monitoring a range of addresses. The optimized state may be either a lightweight power/performance optimized state or an improved power/performance optimized state. The selection between the two states is governed by the explicit input register bit[0] source operand.

Example 4-5 UMONITOR/UMWAIT Sequence to Reduce Power Consumption While Polling

```
#include <x86gprintrin.h>
/*
 * C0.2 Improves performance of the other SMT thread(s)
 * on the same core and has larger power savings
 * but has a longer wake-up time.
 */

#define UMWAIT_STATE_C0_2 0
#define UMWAIT_STATE_C0_1 1

retry = 0;
while (comp.status == 0 && retry++ < MAX_COMP_RETRY) {
    _umonitor(&comp);
    if (comp.status == 0) {
       uint64_t delay = __rdtsc() + UMWAIT_DELAY;
       _umwait(UMWAIT_STATE_C0_1, delay);
    }
}
```

4.1.5 Partial Completion Handling

Intel DSA supports the PCI Express Address Translation Service (ATS) and Page Request Service (PRS) capabilities and uses ATS requests to the IOMMU to translate virtual addresses in descriptors to host physical addresses. These translation requests can return faults due to not-present translations or a mismatch between access permissions and the access type.

The device may encounter a page fault on:

- A Completion Record address
- The Descriptor List address in a Batch descriptor
- Readback address in a Drain descriptor
- Source buffer or destination buffer address

For the first three cases, the device blocks until the page fault is resolved if PRS is enabled; otherwise, it is reported as an error. For the fourth case, the device can either block until the page fault is resolved or prematurely complete the descriptor and return a partial completion to the client, as specified by the Block On Fault flag in the descriptor.

The Block On Fault flag in the descriptor is set to zero in the descriptor preparation sample code in Example 4-1. Therefore, any page fault on the source or destination addresses would cause the operation to partially complete. The completion record reports the faulting address and the number of bytes processed completely. The application can choose between completing the operation in software and resubmitting the operation to Intel DSA after modifying the descriptor as necessary; for example, for a memmove descriptor, SW can touch the faulting address reported in the completion record and resubmit the operation after updating the source address, the destination address, and transfer size fields in the descriptor.

To maximize the utilization of the device, provide equitable BW allocation when configured as a shared device, and provide comparatively better execution predictability, it is recommended to configure the WQ with Block On Fault disabled.

Actions for Continuation After a Page Fault

Table 4-1 describes the changes software may make to a faulting descriptor to create a continuation descriptor after resolving the fault and clearing the Status field of the completion record. These steps apply to descriptors that complete with a page fault (status code 0x03, 0x04, 0x06, or 0x1f). When encountering "increase" or "decrease" in this table without a quantity, use Bytes Completed.

Table 4-1 Software Actions for Continuation After a Page Fault

Operation	Recommended adjustment after partial completion
Batch	• Increase Descriptor List Address by Descriptors Completed × 64 and decrease Descriptor Count by Descriptors Completed. • If Batch Continuation Support is 1, copy bit 0 of the Result field in the completion record into the Batch Error flag of the continuation descriptor. • If Batch Continuation Support is 0 and any operations before the fault are completed with Status ≠ Success and any descriptor after the fault has the Fence flag set, decrease the Descriptor Count not to execute the descriptor with the Fence. • If Batch1 Support is 0 and Descriptor Count is 1, submit the descriptor as a single descriptor rather than a batch.
Drain	No change. The descriptor may be resubmitted as-is.
Memory Move	If Result = 0, increase source and destination addresses and decrease transfer size. If Result = 1, decrease transfer size. (No change to source and destination addresses.)
Create Delta Record	• Increase source addresses and decrease transfer size by Bytes Completed. • Increase Delta Record Address and decrease Maximum Delta Record Size by Delta Record Size. • If the remaining Maximum Delta Record Size < 80, don't resubmit the descriptor; instead, treat it as completed with Result = 2. • The result must be saved and combined with the Result from the subsequent completion record. • Delta Record Size must be saved and combined with the Delta Record Size from the subsequent completion record. • Offsets in deltas created by the continuation descriptor will be incorrect.

Operation	Recommended adjustment after partial completion
Apply Delta Record	Increase Delta Record Address and decrease Delta Record Size. (No change to Destination Address or Transfer Size.)
Dualcast	Increase source and destination addresses and decrease transfer size.
CRC	Increase source address and decrease transfer size. Fill in CRC Seed with CRC Value.
CRC with Copy	• Increase source and destination addresses and decrease transfer size. • Fill in CRC Seed with CRC Value.
DIF Check	• Increase source address and decrease transfer size. • Fill in Reference and Application Tag Seeds with the values from the complete record.
DIF Insert	• Increase source address and decrease transfer size by Bytes Completed. • Increase destination address by Bytes Completed + (Bytes Completed / Block Size) × 8. Fill in Tag Seeds with the values from the completion record.
DIF Strip	• Increase source address and decrease transfer size by Bytes Completed. • Increase destination address by Bytes Completed - (Bytes Completed / Block Size) × 8. Fill in Tag Seeds with the values from the completion record.
DIF Update	Increase source and destination addresses and decrease transfer size. Fill in Tag Seeds with values from the completion record.
Cache Flush	Increase destination address and decrease transfer size.

4.2 Programming Considerations

4.2.1 Ordering/Fencing

Applications may need to guarantee ordering in descriptor execution. Please refer to the Ordering and Fencing section in the Intel® DSA Architecture Specification for details on conditions under which ordering is guaranteed and the utility of the fence flag in descriptors within a batch.

4.2.2 Caching and Persistent Memory Controls

DSA provides descriptor flags that indicate whether memory writes should be directed to cache, dynamic memory, persistent memory, or a peer device. There are three cache control flags, called Cache Control 1, Cache Control 2, and Cache Control 3. (Cache Control 3 was introduced in DSA Gen 3. Cache Control 2 was previously named Destination Readback.)

When Cache Control 1 is 0, DSA write operations are directed to memory. If the destination address is present in any cache, it may be evicted. When Cache Control 1 is 1, writes are directed to the cache. If the destination is present in a cache, it will be updated. If it is not present, a new cache entry may be allocated. Both of these values are hints.

When Cache Control 2 is 1, DSA performs a readback after performing the write operation. This is intended for use with memory writes that target a peer device, to ensure that the data has reached the peer device before the descriptor completes. (This is not necessary for writes that target memory addresses, because DSA always ensures that such writes are visible before completing the descriptor.)

When Cache Control 3 is 1, DSA treats the writes as writes to durable memory. This feature is available if the Durable Write Support capability in GENCAP is 1. For implementations where Durable Write Support is 0, software may indicate writes to durable memory by setting Cache Control 1 to 0 and Cache Control 2 to 1. The destination readback ensures that the data has reached persistent memory.

Some combinations of the cache control flags are reserved. Sections 3.10 Cache Control, 3.11 Persistent Memory Support, and 8.1.3.1 Cache Control Hints in the Intel® DSA Architecture Specification provide more information on cache controls.

4.3 Library Support for Intel® DSA

Table 4-2 Libraries with Support for Intel® DSA

Library	Description
Intel® DML	Intel has developed an open-source library, Intel Data Movement Library (DML), which provides a low-level C and high-level C++ API for data processing using Intel DSA and a software path in case Intel DSA is unavailable. The DML also includes sample applications that can help quickly enable support for Intel DSA in applications. https://intel.github.io/DML
Libfabric	Libfabric includes support for Intel DSA within its shared memory provider since libfabric version 1.17.0; this enables Intel DSA usage in HPC applications that use the Intel MPI, MPICH, OpenMPI, and MVAPICH libraries. The SHM provider for libfabric is described at https://ofiwg.github.io/libfabric/v1.19.0/man/fi_shm.7.html.
Intel® MPI	Intel MPI includes support for Intel DSA since version 2021.7. Instructions for enabling Intel DSA for the shm transport used for intranodal communication are available in the Intel MPI documentation.
SPDK	The Storage Performance Development Kit (SPDK) provides tools and libraries for writing high-performance, scalable, user-mode storage applications. SPDK support for Intel® DSA is described at https://spdk.io/doc/idxd.html.
DPDK	DPDK is the Data Plane Development Kit with libraries to accelerate packet processing workloads running on various CPU architectures. DPDK support for Intel® DSA is described at http://doc.dpdk.org/guides/dmadevs/idxd.html.
Intel® DTO	Intel DSA Transparent Offload (DTO) library is a Linux user-level library that allows applications to use Intel DSA transparently (i.e., without modification). DTO library links with the applications to intercept certain API calls (memcpy, memset, memcmp, and memmove) and uses Intel DSA to perform these operations. https://github.com/intel/DTO

CHAPTER 5
INTEL® DSA PERFORMANCE MICROS

The Intel® Data-Streaming Accelerator Performance Micros (Intel® DSA Performance Micros) utility allows software developers to characterize latency and bandwidth for Intel DSA operations and determine the device configuration and programming parameters that would work best for their application.

Table 5-1 Intel® DSA Performance Micro (dsa-perf-micros) Links

Description	**URL**
Download command ($ git clone)	https://github.com/intel/dsa-perf-micros
Build instructions	https://github.com/intel/dsa-perf-micros/blob/main/doc/build.rst
Sample command lines	https://github.com/intel/dsa-perf-micros/blob/main/doc/options.rst

CHAPTER 6
INTEL® DSA PERFORMANCE COUNTER REFERENCES

Intel helps collect information about key events occurring in different parts of the Intel® DSA hardware. These counters may be useful for debugging and performance tuning. The Performance Monitoring Events appendix in the Intel® DSA Architecture Specification describes events defined for different categories:

- WQs
- Engines
- Address Translation
- etc.

Intel DSA performance counters can be set up and read using the Linux **perf** command.[2] The **-e** option of the **perf stat** command can be used to program performance counters to count events.[3]

Parameters that can be specified for Intel DSA are listed in sysfs:

```
$ ls /sys/bus/event_source/devices/dsa0/format
event  event_category  filter_eng  filter_pgsz  filter_sz  filter_tc  filter_wq
```

A single event can be read every 1 s with the **-I** flag using the command syntax below.

```
$ perf stat -e dsa0/event_category=0x1,event=0x2/ -I 1000
```

Multiple events can be read using a comma-separated list.

```
$ perf stat -e dsa0/event_category=0x1,event=0x2/,dsa0/event_category=0x1,event=0x4/
-I 1000
```

Multiple events can be configured for a counter, and a set of filters can constrain each event. Examples of filters are WQ, Engine, Traffic Class, and Transfer Size. A command line with multiple events configured for a single counter and filtered by 4KB ≤ transfer size < 16KB follows.

```
$ perf stat -e dsa0/event_category=0x1,event=0x6,filter_sz=0x8/ -I 1000
```

Additional information on the usage of the perf command is available in the DSA Perfmon GitHub repository.

[2] See https://perf.wiki.kernel.org/index.php/Main_Page.
[3] See https://man7.org/linux/man-pages/man1/perf-stat.1.html.

CHAPTER 7
HELPER FUNCTIONS FOR LEGACY GCC VERSIONS

GCC supports the **ENQCMD** and **MOVDIR64B** instructions since the gcc10 release with the **-menqcmd** and **-movdir64b** switches, respectively. The **UMONITOR** and **UMWAIT** instructions have been supported since the gcc9 release with the **-mwaitpkg** switch.

For earlier versions of gcc, the following function definitions can be used.

Example 7-1 MOVDIR64B

```
static inline void
movdir64b(void *dst, const void *src)
{
    asm volatile(".byte 0x66, 0x0f, 0x38, 0xf8, 0x02\t\n"
                 : : "a" (dst), "d" (src));
}
```

Example 7-2 ENQCMD

```
static inline unsigned int
enqcmd(void *dst, const void *src)
{
     uint8_t retry;

     asm volatile(“.byte 0xf2, 0x0f, 0x38, 0xf8, 0x02\t\n”
                  “setz %0\t\n”
                  : “=r”(retry) : “a” (reg), “d” (desc));

     return (unsigned int)retry;
}
```

Example 7-3 UMWAIT

```
static inline unsigned char
umwait(unsigned int state, unsigned long long timeout)
{
     uint8_t r;
     uint32_t  timeout_low  =  (uint32_t)timeout;
     uint32_t timeout_high = (uint32_t)(timeout >> 32);

     asm volatile(".byte 0xf2, 0x48, 0x0f, 0xae, 0xf1\t\n"
                  "setc %0\t\n" :
                  "=r"(r) :
                  "c"(state), "a"(timeout_low), "d"(timeout_high));
     return r;
}
```

Example 7-4 UMONITOR

```
static inline void
umonitor(void *addr)
{
     asm volatile(".byte 0xf3, 0x48, 0x0f, 0xae, 0xf0" : : "a"(addr));
}
```

CHAPTER 8 SAMPLE C PROGRAM

8.1 Steps

1. Install the accel-config library from https://github.com/intel/idxd-config or your distribution's package manager.
2. Configure Shared WQ following Example 3-9 in Section 3.2.6. Assuming the source file is **intel_dsa_sample.c**. Use the following command to compile.

```
$ make intel_dsa_sample LDLIBS=-laccel-config
```

8.2 The Program

Example 8-1 Intel® DSA Shared WQ Sample Application

```
/* Intel DSA Shared WQ Sample Application */
/* Page 1 of 5 */

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <linux/idxd.h>
#include <xmmintrin.h>
#include <accel-config/libaccel_config.h>

#define NOP_RETRY   10000
#define BLEN        4096

struct wq_info {
        bool wq_mapped;
        void *wq_portal;
        int wq_fd;
};

static inline int enqcmd(volatile void *reg, struct dsa_hw_desc *desc)
{
        uint8_t retry;

        asm volatile (".byte 0xf2, 0x0f, 0x38, 0xf8, 0x02\t\n"
                      "setz %0\t\n":"=r" (retry):"a"(reg), "d"(desc));
        return (int)retry;
}

static inline void submit_desc(void *wq_portal, struct dsa_hw_desc *hw)
{
        while (enqcmd(wq_portal, hw)) _mm_pause();
}
```

```
/* Page 2 of 5 */

static uint8_t op_status(uint8_t status)
{
        return status & DSA_COMP_STATUS_MASK;
}

static bool is_write_syscall_success(int fd)
{
        struct dsa_hw_desc desc = {0};
        struct dsa_completion_record comp __attribute__((aligned(32)));
        int retry = 0;
        int rc;

        desc.opcode = DSA_OPCODE_NOOP;
        desc.flags = IDXD_OP_FLAG_CRAV | IDXD_OP_FLAG_RCR;
        comp.status = 0;

        desc.completion_addr = (unsigned long)&comp;

        rc = write(fd, &desc, sizeof(desc));

        if (rc == sizeof(desc)) {
                while (comp.status == 0 && retry++ < NOP_RETRY)
                        _mm_pause();

                if (comp.status == DSA_COMP_SUCCESS)
                        return true;
        }

        return false;
}

static int map_wq(struct wq_info *wq_info)
{
        void *wq_portal;
        struct accfg_ctx *ctx;
        struct accfg_wq *wq;
        struct accfg_device *device;
        char path[PATH_MAX];
        int fd;
        int wq_found;

        wq_portal = MAP_FAILED;

        accfg_new(&ctx);

        accfg_device_foreach(ctx, device) {

                /*
                 * Use accfg_device_(*) functions to select enabled device
                 * based on name, numa node
                 */
                accfg_wq_foreach(device, wq) {
                        if (accfg_wq_get_user_dev_path(wq, path, sizeof(path)))
                                continue;

/* Page 3 of 5 */
                        /*
                         * Use accfg_wq_(*) functions select WQ of type
                         * ACCFG_WQT_USER and desired mode
                         */
                        wq_found = accfg_wq_get_type(wq) == ACCFG_WQT_USER &&
                            accfg_wq_get_mode(wq) == ACCFG_WQ_SHARED;

                        if (wq_found)
                                break;
                }

                if (wq_found)
                        break;
        }

        accfg_unref(ctx);

        if (!wq_found)
                return -ENODEV;

        fd = open(path, O_RDWR);
        if (fd >= 0) {
                wq_portal =
                    mmap(NULL, 0x1000, PROT_WRITE, MAP_SHARED | MAP_POPULATE,
                         fd, 0);
        }

        if (wq_portal == MAP_FAILED) {
                /*
                 * EPERM means the driver doesn't support mmap but
                 * can support write syscall. So fallback to write syscall
                 */
                if (errno == EPERM && is_write_syscall_success(fd)) {
                        wq_info->wq_mapped = false;
                        wq_info->wq_fd = fd;
                        return 0;
                }

                        return -errno;
        }

        wq_info->wq_portal = wq_portal;
        wq_info->wq_mapped = true;
        wq_info->wq_fd = -1;

        return 0;
}

int main(int argc, char *argv[])
{
        int fd;
        struct dsa_hw_desc desc = { };
        char src[BLEN];
        char dst[BLEN];
        struct dsa_completion_record comp __attribute__((aligned(32)));
        uint32_t tlen;
        int rc;
/* Page 4 of 5 */
        struct wq_info wq_info;

        rc = map_wq(&wq_info);
        if (rc)
                return EXIT_FAILURE;

        memset(src, 0xaa, BLEN);

        desc.opcode = DSA_OPCODE_MEMMOVE;

        /*
         * Request a completion – since we poll on status, this flag
         * needs to be 1 for status to be updated on successful
         * completion
         */
        desc.flags |= IDXD_OP_FLAG_RCR;

        /* CRAV should be 1 since RCR = 1 */
        desc.flags |= IDXD_OP_FLAG_CRAV;

        /* Hint to direct data writes to CPU cache */
        desc.flags |= IDXD_OP_FLAG_CC;
        desc.xfer_size = BLEN;
        desc.src_addr = (uintptr_t) src;
        desc.dst_addr = (uintptr_t) dst;
        desc.completion_addr = (uintptr_t)&comp;

 retry:
        if (wq_info.wq_mapped) {
                submit_desc(wq_info.wq_portal, &desc);
        } else {
                int rc = write(wq_info.wq_fd, &desc, sizeof(desc));

                if (rc != sizeof(desc))
                        return EXIT_FAILURE;
        }

        while (comp.status == 0) _mm_pause();

        if (comp.status != DSA_COMP_SUCCESS) {
                if (op_status(comp.status) == DSA_COMP_PAGE_FAULT_NOBOF) {
                        int wr = comp.status & DSA_COMP_STATUS_WRITE;
                        volatile char *t;

                        t = (char *)comp.fault_addr;
                        wr ? *t = *t : *t;
                        desc.src_addr += comp.bytes_completed;
                        desc.dst_addr += comp.bytes_completed;
                        desc.xfer_size -= comp.bytes_completed;
                        goto retry;
                } else {
                        printf("desc failed status %u\n", comp.status);
                        rc = EXIT_FAILURE;
                }
        } else {
                printf("desc successful\n");

/* Page 5 of 5 */
                rc = memcmp(src, dst, BLEN);

                rc ? printf("memmove failed\n") :
                    printf("memmove successful\n");
                rc = rc ? EXIT_FAILURE : EXIT_SUCCESS;
        }

        return EXIT_SUCCESS;
}
```

CHAPTER 9 DEBUG AIDS FOR CONFIGURATION ERRORS

Suggestions for troubleshooting and debugging for commonly encountered error situations are provided below.

- Verify that VT-d is enabled in the BIOS.
- Verify that the Linux Kernel configuration options mentioned in Section 2.2 are enabled.
- Run **dmesg | grep ACPI | grep DMAR**. The output should be similar to the messages in Example 9-1.

Example 9-1 Linux Kernel ACPI Subsystem Messages When VT-d is Enabled

```
$ dmesg -t | grep ACPI | grep DMAR
ACPI: DMAR 0x000000007738F000 000550 (v01 INTEL INTEL ID 00000001 INTL 20091013)
ACPI: Reserving DMAR table memory at [mem 0x7738f000-0x7738f54f]
```

- Use lspci to ensure the expected Intel DSA devices exist, and the lspci output indicates that the “Kernel driver in use:” is set to idxd.
- Run **dmesg | grep -i dmar** to ensure the kernel enumerates DMAR (DMA remapping reporting) devices.
    - If VT-d is enabled in the BIOS and no DMAR devices are reported, then the IOMMU driver may not be enabled by default.
    - Reboot with “**intel_iommu=on,sm_on**” added to the kernel command line to enable VT-d scalable mode.
- Run **dmesg | grep -i idxd** if you see “Unable to turn on SVA feature,” VT-d scalable mode may not be enabled by default.
    - Reboot with “**intel_iommu=on,sm_on**” added to the kernel command line to enable VT-d scalable mode.
- On certain platforms, VT-d 5-level paging capability is disabled by the Basic Input/Output System (BIOS); you will see “SVM disabled, incompatible paging mode” in the **dmesg** output.
    - In this case, pass **no5lvl** on the kernel command line. This boot-time parameter disables the 5-level paging mode and forces the kernel to use the 4-level paging mode.