Intel® Data Streaming Accelerator User Guide

Ref#: 353216-002US
June 2024

Notices & Disclaimers

Intel technologies may require enabled hardware, software, or service activation. No product or component can be absolutely secure.

Your costs and results may vary.

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.

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

Intel uses code names to identify products, technologies, or services that are in development and not publicly available. These are not “commercial” names and are not intended to function as trademarks.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document, with the sole exception that a) you may publish an unmodified copy and b) code included in this document is licensed subject to the Zero-Clause BSD open source license (0BSD), https://opensource.org/licenses/0BSD. You may create software implementations based on this document and in compliance with the preceding that are intended to execute on the Intel product(s) referenced in this document. No rights are granted to create modifications or derivatives of this document.

© 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.

Revision History

Date	Revision	Description
November 2022	001	The initial release of the document.
June 2024	002	• Added Section 3.2.2 Intel® DSA Read Buffer Controls. • Updated Figure 3-2. • Updated Code in Chapter 8.

Table of Contents

TABLES

FIGURES

EXAMPLES

CHAPTER 1
INTRODUCTION

1.1 Audience

Intel® DSA is a high-performance data copy and transformation accelerator integrated into Intel® processors, starting with 4th Generation 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's intended audience includes 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 with the Intel® DSA Architecture Specification and documentation for SW utilities and libraries that support Intel DSA, such as accel- config/libaccel-config, Libfabric, and Intel® MPI.

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.
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 (e.g., processor cores, memory, devices) to ensure optimal usage by multiple concurrent applications.
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.

Acronym	Term	Definition
SKU	Stock-Keeping Unit	The part number or product number that identifies an item.
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://cdrdv2.intel.com/v1/dl/getContent/671116
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® Architecture Instruction Set Extensions Programming Reference	https://cdrdv2.intel.com/v1/dl/getContent/671368
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://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.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
accel-config (01.org)	https://01.org/blogs/2020/pedal-metal-accelerator-configuration- and-control-open-source.html
Libfabric	https://github.com/ofiwg/libfabric/blob/main/man/fi_shm.7.md

1.3 Document Organization

This document has two general sections. The first describes the Intel DSA setup/configuration, and the second describes the sample code for Intel DSA.

This document does not include configuration mechanisms and settings specific to domain-specific software stacks like DPDK, SPDK, etc. Please refer to the appropriate documentation for additional information.

CHAPTER 2
PLATFORM CONFIGURATION

2.1 BIOS Configuration

This document describes Intel® DSA usage with user space memory, requiring Intel® Virtualization Technology or Directed I/O (VT-d) to be enabled.

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.

```
CONFIG_INTEL_IOMMU=y CONFIG_INTEL_IOMMU_SVM=y
CONFIG_INTEL_IOMMU_DEFAULT_ON=y
CONFIG_INTEL_IOMMU_SCALABLE_MODE_DEFAULT_ON=y
```

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

2.2.2 Intel® DSA Driver

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

```
CONFIG_INTEL_IDXD=m
CONFIG_INTEL_IDXD_SVM=y
CONFIG_INTEL_IDXD_PERFMON=y
```

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

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 to print the kernel message buffer, as shown in Example 2-3.

```
$ dmesg | 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)
```

Example 2-3 IDXD Driver Initialization Messages

Distribution kernel versions with complete IDXD driver support are shown in Table 2-1. Please refer to vendor documentation for the latest information.

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

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

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 and cores on a fourth-generation Xeon® server processor. Depending on the processor SKU, one, two, or four Intel DSA devices exist per socket. A system with two sockets can have up to eight Intel DSA devices.

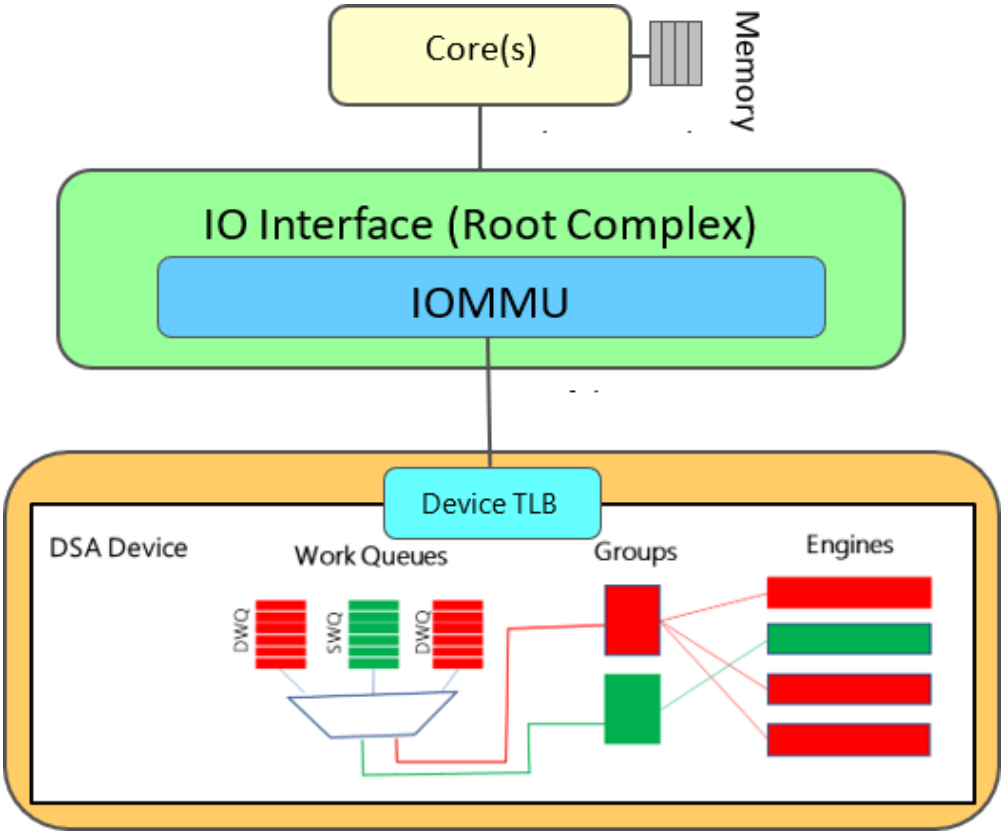

Figure 3-1 Logical Organization of an Intel® DSA and Cores on a 4th-Generation Xeon® Server Processor

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

Example 3-1 Listing All Intel® DSA Devices

```
$ 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
```

The complete lspci output for an Intel DSA device can be obtained, as shown in Figure 3-3. 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

```
/*1 of 2*/
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
```

```
                  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-
  /*2 of 2*/

                  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 within 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 a total of eight Intel DSA devices with 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 Intel® DSA WQs/Engines/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.

Dedicated WQs require SW to manage flow control by keeping track of descriptors submitted and completed to ensure the WQ capacity is not overrun. Hence, they are helpful in cases where a single OS-level process uses the WQ.

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 the Work Queues, Engines, and Groups sections in the *Intel® DSA Architecture Specification* for more details on Intel DSA WQ/Engine/Group capabilities and controls.

3.2.2 Intel® DSA Read Buffer Controls

DSA uses read buffers to hide memory read latency. Administrative software may utilize the read buffer controls to achieve an approximate range of read bandwidth numbers (min and max) for a group.

DSA per-group configuration provides two fields to control Read Buffers available to a group.

- Read Buffers Allowed: A non-zero value for the Read Buffers Allowed field indicates the maximum number of Read Buffers that may be in use at one time by all engines in the group. This field can limit the read bandwidth achievable on an idle system.
- 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 irrespective of system load.

The magnitude of memory read latency that a single read buffer can hide may vary depending on the system load. As a result, when Read Buffers Allowed equals Read Buffers Reserved for a given group, the achievable read bandwidth falls within a range of values, as shown in Figure 3-2. Under system idle conditions, a group can reach peak bandwidth with 48 or more read buffers available.

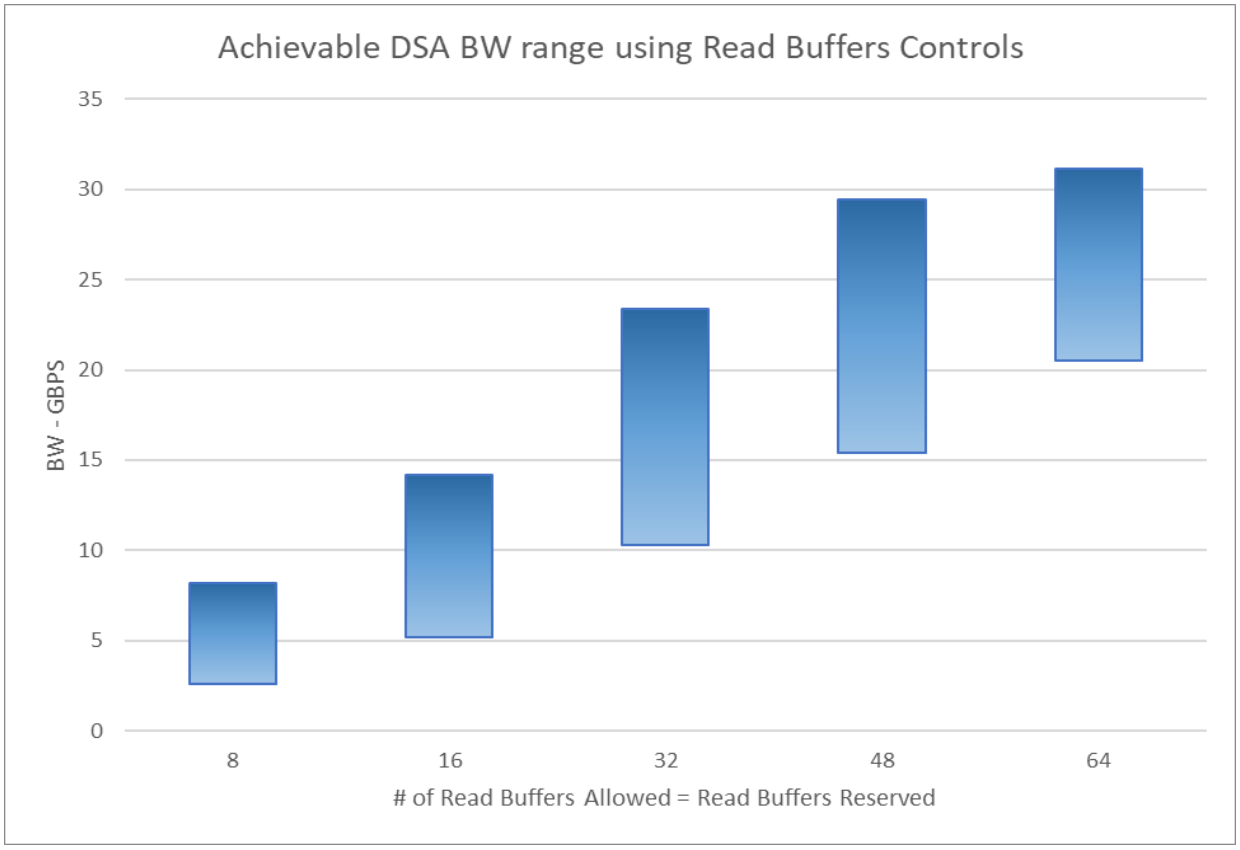

Figure 3-2 DSA BW Range & Read Buffer Controls

Please refer to section 4.3 of the DSA specification for restrictions on programming Read Buffers Allowed and Read Buffers Reserved controls.

3.2.3 Intel® DSA Traffic Class Configuration

First generation Intel® Data Streaming Accelerator (DSA) devices on 4th Gen and 5th Gen Intel® Xeon® Scalable Processors are limited to using a single virtual channel for DMA traffic that is different from the virtual channel used for descriptor submissions to the device (VC0). Software should avoid using Traffic Class 0 (TC0) for DMA traffic by programming TC-A (Bits 2:0) and TC-B (Bits 5:3) fields in the device Group Configuration Table (GRPFLAGS Register, BAR + 328h, 368h, 3A8h, or 3E8h) register to a non-zero TC value which is mapped to Virtual Channel 1 (VC1). Sharing a single virtual channel for DMA traffic to both high-bandwidth and low-bandwidth memory targets may impact the throughput of some operations.

The Linux driver default Traffic class configuration follows the above-mentioned guidelines for the respective device versions.

3.2.4 Linux Interfaces

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

Example 3-2 shows the configuration and control sysfs entries for the dsa0 device. WQs are configured by a superuser and must be configured with a size and type at a minimum. A group must have at least one constituent WQ and one Engine. The configuration using the accel-config utility is described in Section 3.2. An Intel DSA device and associated WQs may only be enabled after configuring the corresponding Group/WQ/Engine(s).

```
$ 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

3.2.5 accel-config

accel-config is a Linux application that provides a command line interface for Intel DSA configuration. It links to 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.

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
```

app_profile.conf is a configuration intended for user space applications and provides two groups with one SWQ and one engine each. 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_pro- file.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
```

Chapter 7 shows a command-line example for enabling an Intel DSA WQ with a custom configuration and saving the configuration to a file.

accel-config can be shown 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
```

3.2.6 WQ Device File Permissions

The **accel-config** command line creates WQ device files, as seen in Example 3-9.

Example 3-9 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 start an application that uses Intel® DSA once the superuser has configured an Intel DSA device and at least one associated WQ and enabled the user's access to the WQ character device file (see Section 3.2). The commands used to configure the device and a shared WQ are provided in Appendix A.

This section provides walk-through C program snippets to illustrate the steps needed to use Intel DSA. A complete source code listing for a C program that uses Intel DSA is provided in Appendix C.

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.

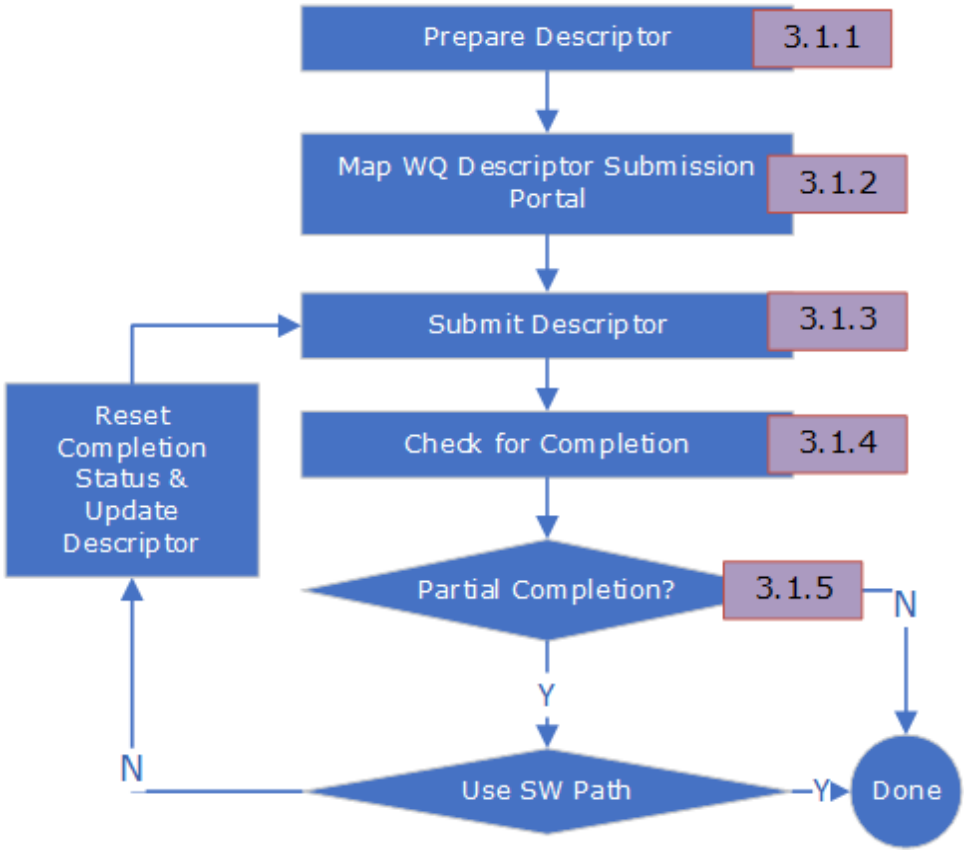

Figure 4-1 Descriptor Processing Sequence

4.1.1 Descriptor Preparation

Sample code to prepare a *Memory Move descriptor* (as described in the *Memory Move* section within the Intel® DSA Architecture Specification) 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. Please refer to Section 4.1.5 for details on the Block on Fault flag.

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;


/* 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; 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 (e.g., **/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 9 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 some Intel® Data Streaming Accelerator (Intel® DSA) and Intel® Analytics Accelerator (Intel® IAA) V1.0 for some Intel® 4th or 5th generation Xeon® processors may allow denial of service. Intel is releasing prescriptive guidance and software updates to mitigate this potential vulnerability; the OS may disallow an unprivileged process from mapping the work submission portal into its address space[1]. 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 Chapter 9 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 within 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 Appendix B.

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);
```

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
```

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

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 wakeup 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);
}
}
```

4.1.3 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 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 complete partially. 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. Please refer to Appendix D for further information on resubmitting descriptors for other operations.

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.

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 within 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 Destination Address in Persistent Memory

The Persistent Memory Support section of the Intel® DSA Architecture Specification describes how descriptor flags must be programmed to guarantee data persistence at descriptor completion.

4.2.3 Traffic Class Configuration

First generation Intel® Data Streaming Accelerator (DSA) devices on 4th Gen and 5th Gen Intel® Xeon® Scalable Processors are limited to using a single virtual channel for DMA traffic that is different from the virtual channel used for descriptor submissions to the device (VC0). Software should avoid using Traffic Class 0 (TC0) for DMA traffic by programming TC-A (Bits 2:0) and TC-B (Bits 5:3) fields in the device Group Configuration Table (GRPFLAGS Register, BAR + 328h, 368h, 3A8h, or 3E8h) register to a non-zero TC value which is mapped to Virtual Channel 1 (VC1). Sharing a single virtual channel for DMA traffic to both high-bandwidth and low-bandwidth memory targets may impact the throughput of some operations.

4.3 rary Support for Intel® DSA

Table 9-1 Libraries with Support for Intel® DSA

Library	Description
Intel® DML	Intel has also 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.[2]
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

[2] https://intel.github.io/DML/

Library	Description
	MVAPICH libraries. [3]
Intel® MPI	Intel MPI includes support for Intel DSA since version 2021.7; instructions for enabling Intel DSA for the shm transport used for intra-node 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.[4]
DPDK	DPDK is the Data Plane Development Kit with libraries to accelerate packet processing workloads running on various CPU architectures.[5]
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.[6]

[3] SHM provider for libfabric is described at: https://ofiwg.github.io/libfabric/v1.19.0/man/fi_shm.7.html.
[4] SPDK support for Intel® DSA is described at: https://spdk.io/doc/idxd.html.
[5] DPDK support for Intel® DSA is described at: http://doc.dpdk.org/guides/dmadevs/idxd.html.
[6] 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® Data-Streaming Accelerator 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-icros/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 of 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.[7] The **-e** option of the **perf stat** command can be used to program performance counters to count events.[8].

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 1s 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.[9]

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

[7] See https://perf.wiki.kernel.org/index.php/Main_Page.
[8] See https://man7.org/linux/man-pages/man1/perf-stat.1.html.
[9] Additional information on the usage of the perf command is available on the DSA Perf Micros GitHub repository.

CHAPTER 7
STEPS FOR USING ACCEL-CONFIG: EXAMPLES

1. Configure the device.

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

```
$ accel-config config-engine dsa0/engine0.2 --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
```

2. Configure the group by configuring the engine and WQ.

3. Enable the device and WQ.

```
$ accel-config enable-device dsa0

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

4. Save the configuration to the config file.

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

CHAPTER 7
C FUNCTIONS FOR GCC VERSIONS WITHOUT MOVDIRB64, ENQCMD, UMWAIT, OR UMONITOR SUPPORT

GCC supports the **ENQCMD** and **MOVDIR64B** 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.

Example 8-1 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 8-2 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 8-3 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);


le(".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 8-4 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

5. Install the accel-config library from https://github.com/intel/idxd-config or your distribution's package manager.
6. Configure Shared WQ using the example in Chapter 7. 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

```
/* 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();
}

static uint8_t op_status(uint8_t status)
{
```

```
/* 2 of 5 */

        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;

                        /* Use accfg_wq_(*) functions select WQ of type
                             * ACCFG_WQT_USER and desired mode
```

```
/* 3 of 5 */
                                */
                            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;
```

```
/* 4 of 5 */
    int rc;
          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 â€“ 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;
```

```
/* 5 of 5 */

                }
        } else {
                printf("desc successful\n");
                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
ACTIONS FOR CONTINUATION AFTER A PAGE FAULT

This table 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 status 0x03, 0x04, 0x06, 0x1a, or 0x1f. When encountering "increase" or "decrease" in this table without a quantity, use the Bytes Completed.

Table 10-1 SW Actions for Continuation After a Page Fault

Operation	Recommended adjustment after partial completion
Nop	No change. The descriptor may be resubmitted as-is.
Batch	Increase Descriptor List Address by Descriptors Completed × 64 and decrease Descriptor Count by Descriptors Completed. • If 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 the 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.
Copy	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.
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.

Operation	Recommended adjustment after partial completion
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.

CHAPTER 10
DEDICATED AND SHARED WQ COMPARISON DESCRIPTIONS

Table 11-1 Dedicated and Shared WQ Comparison

	Dedicated WQ	Shared WQ
Client Scaling	A limited number of DWQs intended only for specific vertical uses (e.g., networking and storage infrastructure).	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.
WQ Sharing	Single client per DWQ and SW tracks the number of outstanding submissions to ensure no queue overflow.	SW does not need to keep track of outstanding submissions and can use the ENQCMD ISA result to identify successful vs. unsuccessful submissions.
Submission Rate	The software can stream descriptors to the device at a very high rate using MOVDIR64B ISA with low latency.	Rate of descriptor submission limited by ENQCMD round-trip latency (approx. 200-250ns on 4th Generation Intel® Xeon®).

CHAPTER 11
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.
- Run **dmesg | grep -i ACPI | DMAR**. The output should be similar to the messages in
  - Verify that the Linux Kernel configuration options mentioned in Section 2.2 are enabled.

Table 12-1 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 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.