



# **NVIDIA DOCA DMA Copy Application Guide**

# Table of contents

1. Introduction

---

2. System Design

---

3. Application Architecture

---

4. DOCA Libraries

---

5. Compiling the Application

---

6. Running the Application

---

7. Application Code Flow

---

8. References

---

This guide provides an example of a DMA Copy implementation on top of NVIDIA® BlueField® DPU.

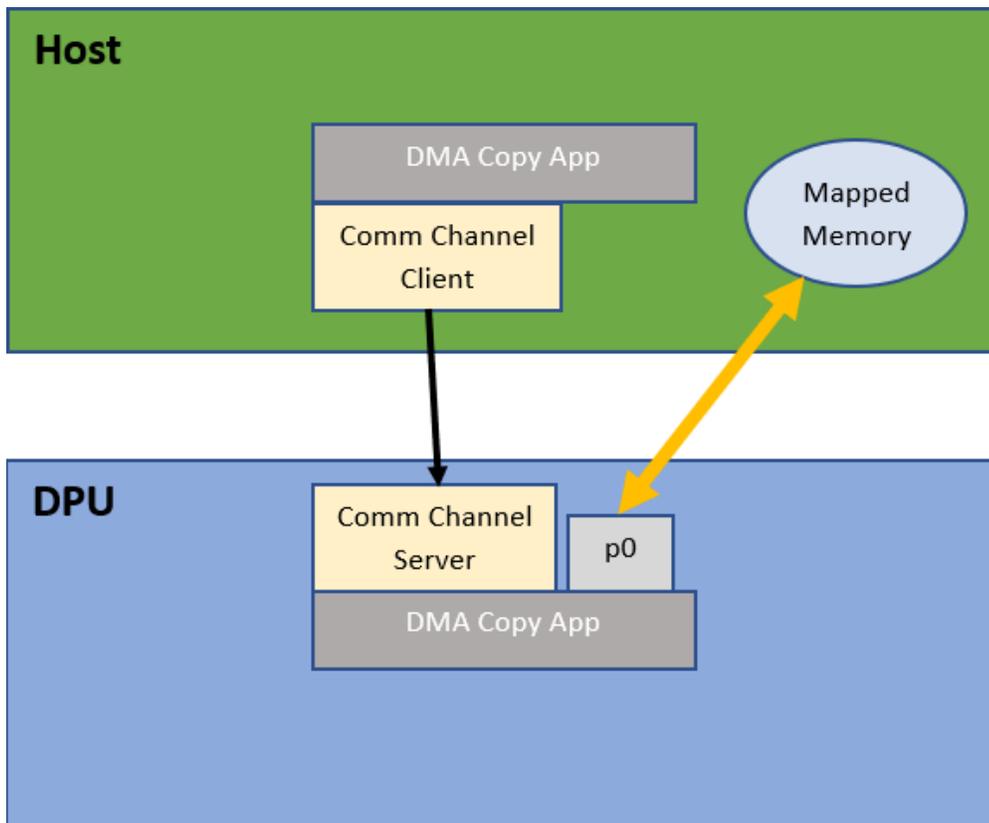
## 1. Introduction

DOCA DMA (direct memory access) Copy application transfers files (data path), up to the maximum supported size by the hardware, between the DPU and the x86 host using the [DOCA DMA Library](#) which provides an API to copy data between DOCA buffers using hardware acceleration, supporting both local and remote memory.

DOCA DMA allows complex memory copy operations to be easily executed in an optimized, hardware-accelerated manner.

## 2. System Design

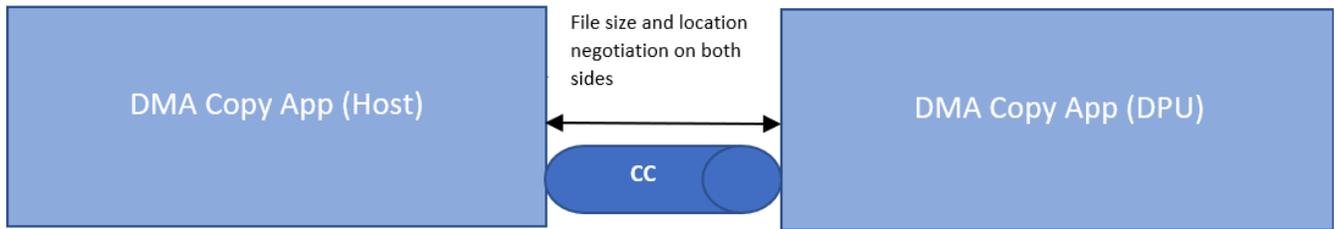
DOCA DMA Copy is designed to run on the instances of the BlueField DPU and x86 host. The DPU application must be the first to spawn as it opens the [DOCA Comch](#) server between the two sides on which all the necessary DOCA DMA library configuration files (control path) are transferred.



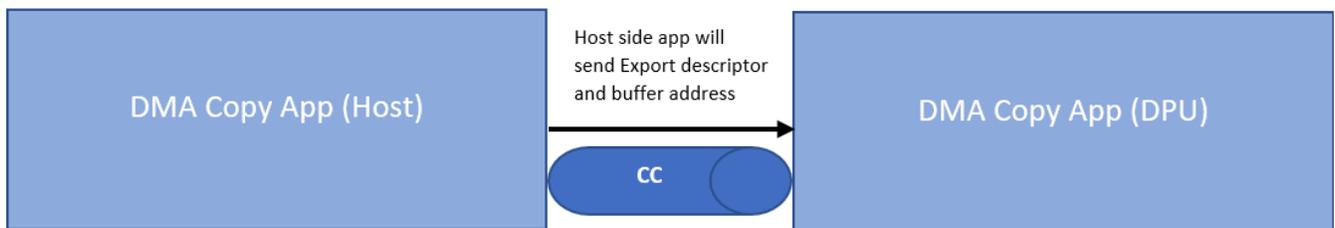
## 3. Application Architecture

DOCA DMA Copy runs on top of DOCA DMA to read/write directly from the host's memory without any user/kernel space context switches, allowing for a fast memory copy.

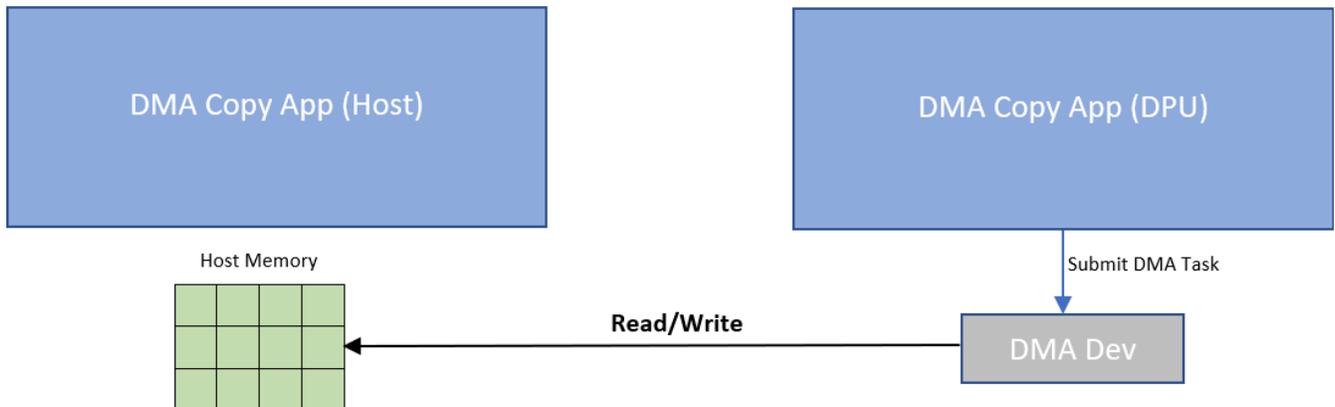
**First stage**



**Second stage**



**Third stage**



Flow:

1. The two sides initiate a short negotiation in which the file size and location are determined.
2. The host side creates the export descriptor with `doca_mmap_export_pci()` and sends it with the local buffer address and length on the Comch to the DPU side application.
3. The DPU side application uses the received export descriptor to create a remote memory map locally with `doca_mmap_create_from_export()` and the host

buffer information to create a remote DOCA buffer.

4. From this point on, the DPU side application has all the necessary memory information and the DMA copy can take place.

## 4. DOCA Libraries

This application leverages the following DOCA libraries:

- [DOCA DMA](#)
- [DOCA Comch](#)

Refer to their respective programming guide for more information.

## 5. Compiling the Application

### Info

Please refer to the [NVIDIA DOCA Installation Guide for Linux](#) for details on how to install BlueField-related software.

The installation of DOCA's reference applications contains the sources of the applications, alongside the matching compilation instructions. This allows for compiling the applications "as-is" and provides the ability to modify the sources, then compile a new version of the application.

### Tip

For more information about the applications as well as development and compilation tips, refer to the [DOCA Applications](#) page.

The sources of the application can be found under the application's directory:

```
/opt/mellanox/doca/applications/dma_copy/.
```

## 5.1 Compiling All Applications

All DOCA applications are defined under a single meson project. So, by default, the compilation includes all of them.

To build all the applications together, run:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

### Info

`doca_dma_copy` is created under `/tmp/build/dma_copy/`.

## 5.2 Compiling Only the Current Application

To directly build only the DMA Copy application:

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build -Denable_all_applications=false -  
Denable_dma_copy=true  
ninja -C /tmp/build
```

### Info

`doca_dma_copy` is created under `/tmp/build/dma_copy/`.

Alternatively, one can set the desired flags in the `meson_options.txt` file instead of providing them in the compilation command line:

1. Edit the following flags in

`/opt/mellanox/doca/applications/meson_options.txt`:

- Set `enable_all_applications` to `false`
- Set `enable_dma_copy` to `true`

2. Run the following compilation commands :

```
cd /opt/mellanox/doca/applications/  
meson /tmp/build  
ninja -C /tmp/build
```

### Info

`doca_dma_copy` is created under `/tmp/build/dma_copy/`.

## 5.3 Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the compilation of the application .

## 6. Running the Application

## 6.1 Application Execution

The DMA Copy application is provided in source form. Therefore, a compilation is required before the application can be executed.

### 1. Application usage instructions:

```
Usage: doca_dma_copy [DOCA Flags] [Program Flags]

DOCA Flags:
  -h, --help                Print a help synopsis
  -v, --version             Print program version
information
  -l, --log-level           Set the (numeric) log
level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR,
40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  --sdk-log-level           Set the SDK (numeric) log
level for the program <10=DISABLE, 20=CRITICAL, 30=ERROR,
40=WARNING, 50=INFO, 60=DEBUG, 70=TRACE>
  -j, --json <path>       Parse all command flags
from an input json file

Program Flags:
  -f, --file                Full path to file to be
copied/created after a successful DMA copy
  -p, --pci-addr           DOCA Comm Channel device
PCI address
  -r, --rep-pci            DOCA Comm Channel device
representor PCI address (needed only on DPU)
```

### Info

This usage printout can be printed to the command line using the `-h` (or `--help`) options:

```
./doca_dma_copy -h
```

### **(i) Info**

For additional information, refer to section "[Command Line Flags](#)".

2. CLI example for running the application on the BlueField:

```
./doca_dma_copy -p 03:00.0 -r 3b:00.0 -f received.txt
```

### **(i) Note**

Both the DOCA Comch device PCIe address (03:00.0) and the DOCA Comch device representor PCIe address (3b:00.0) should match the addresses of the desired PCIe devices.

3. CLI example for running the application on the host:

```
./doca_dma_copy -p 3b:00.0 -f send.txt
```

### **(i) Note**

The DOCA Comch device PCIe address, `3b:00.0`, should match the address of the desired PCIe device.

4. The application also supports a JSON-based deployment mode, in which all command-line arguments are provided through a JSON file:

```
./doca_dma_copy --json [json_file]
```

For example:

```
./doca_dma_copy --json ./dma_copy_params.json
```

### **Note**

Before execution, ensure that the used JSON file contains the correct configuration parameters, and especially the PCIe addresses necessary for the deployment.

## 6.2 Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
General flags	<code>h</code>	<code>help</code>	Print a help synopsis	N/A
	<code>v</code>	<code>version</code>	Print program version information	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	<code>l</code>	<code>log-level</code>	<p>Set the log level for the application:</p> <ul style="list-style-type: none"> <li>• DISABLE=10</li> <li>• CRITICAL=20</li> <li>• ERROR=30</li> <li>• WARNING=40</li> <li>• INFO=50</li> <li>• DEBUG=60</li> <li>• TRACE=70 (requires compilation with <code>TRACE</code> log level support)</li> </ul>	<pre>"log-level" : 60</pre>
	N/A	<code>sdk-log-level</code>	<p>Set the log level for the program:</p> <ul style="list-style-type: none"> <li>• DISABLE=10</li> <li>• CRITICAL=20</li> <li>• ERROR=30</li> <li>• WARNING=40</li> <li>• INFO=50</li> <li>• DEBUG=60</li> <li>• TRACE=70</li> </ul>	<pre>"sdk-log-level" : 40</pre>
	<code>j</code>	<code>json</code>	Parse all command flags from an input JSON file	N/A
Program flags	<code>f</code>	<code>file</code>	<p>Full path to file to be copied/created after a successful copy</p> <p><b>Note</b> This is a mandatory flag.</p>	<pre>"file" : "/tmp/sample.txt"</pre>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Content
	p	pci-addr	DOCA Comch device PCIe address.  <i>i</i> <b>Note</b> This is a mandatory flag.	<pre>"pci-addr" : "b1:00.0"</pre>
	r	rep-pci	DOCA Comch device representor PCIe address.  <i>i</i> <b>Note</b> This is a mandatory flag only on the DPU.	<pre>"rep-pci" : "b1:02.0"</pre>

*i* **Info**

Refer to [DOCA Arg Parser](#) for more information regarding the supported flags and execution modes.

### 6.3 Troubleshooting

Refer to the [NVIDIA DOCA Troubleshooting Guide](#) for any issue encountered with the installation or execution of the DOCA applications.

## 7. Application Code Flow

1. Parse application argument.

1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register DMA Copy application parameters.

```
register_dma_copy_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Initialize Comch endpoint.

```
init_cc();
```

1. Create Comch endpoint.

2. Parse user PCIe address for Comch device.

3. Open Comch DOCA device.

4. Parse user PCIe address for Comch device representor (on DPU side).

5. Open Comch DOCA device representor (on DPU side).

6. Set Comch endpoint properties.

2. Open the DOCA hardware device from which the copy would be made.

```
open_dma_device();
```

1. Parse the PCIe address provided by the user.
2. Create a list of all available DOCA devices.
3. Find the appropriate DOCA device according to specific properties.
4. Open the device.

3. Create all required DOCA core objects.

```
create_core_objects();
```

4. Initiate DOCA core objects.

```
init_core_objects();
```

5. Start host/DPU DMA Copy.

1. Host side application:

```
host_start_dma_copy();
```

1. Start negotiation with the DPU side application for the location and size of the file.
2. Allocate memory for the DMA buffer.

3. Export the memory map and send the output (export descriptor) to the DPU side application.
4. Send the host local buffer memory address and length on the Comch to the DPU side application.
5. Wait for the DPU to notify that DMA Copy ended.
6. Close all memory objects.
7. Clean resources.

## 2. DPU side application:

```
dpu_start_dma_copy();
```

1. Start negotiation with the host side application for file location and size.
2. Allocate memory for the DMA buffer.
3. Receive the export descriptor on the Comch.
4. Create the DOCA memory map for the remote buffer on the host.
5. Receive the host buffer information on the Comch.
6. Create two DOCA buffers, one for the remote (host) buffer and one for the local buffer.
7. Submit the DMA copy task.
8. Send a host message to notify that DMA copy ended.
9. Clean resources.

## 6. Destroy Comch.

```
destroy_cc();
```

## 7. Destroy DOCA core objects.

```
destroy_core_objects();
```

## 8. Arg parser destroy.

```
doca_argp_destroy();
```

# 8. References

- `/opt/mellanox/doca/applications/dma_copy/`
- `/opt/mellanox/doca/applications/dma_copy/dma_copy_params.json`

### Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer’s own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order

to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

## **Trademarks**

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

© Copyright 2025, NVIDIA. PDF Generated on 10/09/2025