NVIDIA DOCA Ethernet Programming Guide

Programming Guide

MLNX-15-060586 _v2.2.1 | October 2023

Table of Contents

Chapter 1. Introduction 1
Chapter 2. Prerequisites 2
Chapter 3. Architecture 3
Chapter 4. Dependencies 5
Chapter 5. Limitations 6
Chapter 6. DOCA Flow Interoperability 7
Chapter 7. API 8
7.1. DOCA ETH RXQ 8
7.1.1. Querying Device Capabilities 8
7.1.1.1. doca_eth_rxq_get_max_packet_size_supported() 8
7.1.1.2. doca_eth_rxq_get_type_supported() 8
7.1.2. Creating and Configuring DOCA_ETH_RXQ 9
7.1.2.1. doca_eth_rxq_create() 9
7.1.2.2. doca_eth_rxq_destroy() 9
7.1.2.3. doca_eth_rxq_set_num_packets() 9
7.1.2.4. doca_eth_rxq_set_max_packet_size() 10
7.1.2.5. doca_eth_rxq_set_type() 10
7.1.2.6. doca_eth_rxq_get_flow_queue_id() 10
7.1.2.7. doca_eth_rxq_get_gpu_handle() 11
7.1.2.8. doca_eth_rxq_get_pkt_buffer_size() 11
7.1.2.9. doca_eth_rxq_set_pkt_buffer() 12
7.2. DOCA ETH TXQ 12
7.2.1. Querying Device Capabilities 12
7.2.1.1. doca_eth_txq_get_max_queue_size_supported() 12
7.2.1.2. doca_eth_txq_get_type_supported() 13
7.2.1.3. doca_eth_txq_get_chksum_offload_supported() 13
7.2.1.4. doca_eth_txq_get_wait_on_time_offload_supported() 13
7.2.2. Creating and Configuring DOCA_ETH_TXQ 13
7.2.2.1. doca_eth_txq_create() 14
7.2.2.2. doca_eth_txq_destroy() 14
7.2.2.3. doca_eth_txq_set_queue_size() 14
7.2.2.4. doca_eth_txq_set_type() 14
7.2.2.5. doca_eth_txq_set_l4_chksum_offload() 15
7.2.2.6. doca_eth_txq_set_l3_chksum_offload() 15

7.2.2.7. doca_eth_txq_set_wait_on_time_offload() 15
7.2.2.8. doca_eth_txq_get_gpu_handle() 16

Chapter 1. Introduction

The DOCA ETH library comprises two APIs that represent two DOCA objects, namely `DOCA_ETH_RXQ` and `DOCA_ETH_TXQ`.

The control path API is handled on the host CPU side by the library. `DOCA_ETH_RXQ`/`DOCA_ETH_TXQ` are configured on the CPU and then exported to the GPU which performs its data path.

`DOCA_ETH_RXQ` offers an API to receive packets on an RX queue. The `doca_eth_rxq` object allows developers to receive Ethernet packets in CPU/GPU memory that they have allocated beforehand (Ethernet receive packets buffer). The library collects the user configuration, creates a receive queue object on the GPU memory (using the `DOCA_GPU` sub-device), and coordinates with the network card (NIC) to receive packets directly into GPU memory.

`DOCA_ETH_TXQ` provides an API to send packets on a TX queue. The library collects the user configuration and creates a send queue object on the GPU memory (using the `DOCA_GPU` sub-device) and exports its configuration to the GPU, enabling developers to send packets on this queue from the GPU side.

Chapter 2. Prerequisites

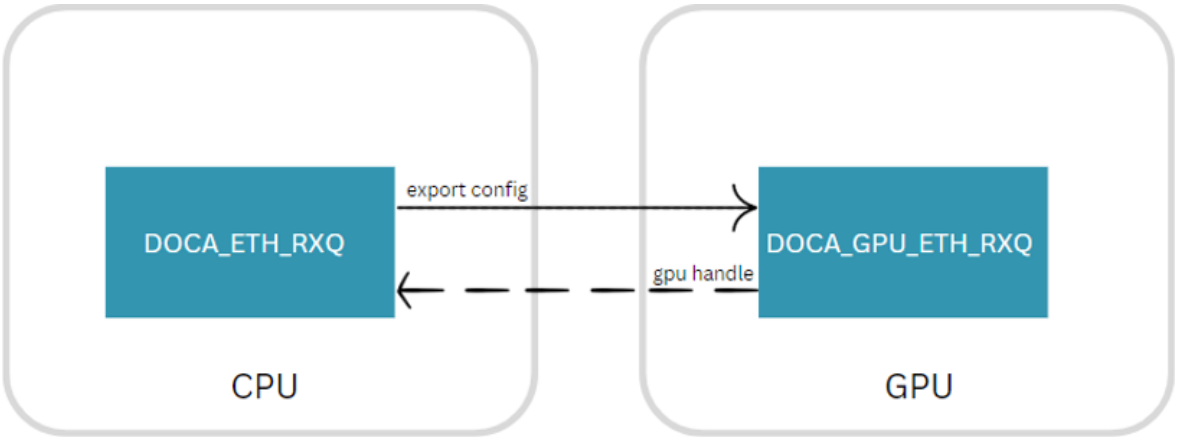
DOCA Ethernet library (Tx/Rx) is supported on the DPU control path and GPU as data path (see GPUNetIO library).

The machine must meet the following prerequisites:

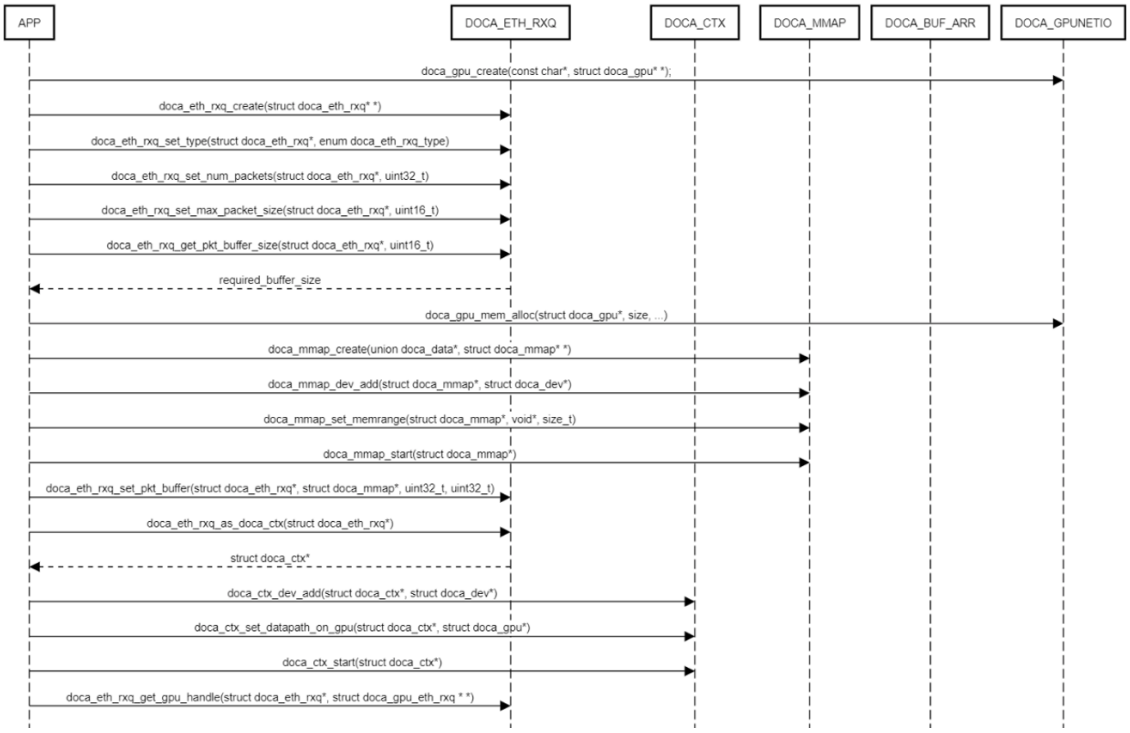
- DOCA version 2.0.2 or greater
- BlueField software 4.0.2 or greater
- BlueField-3 firmware 32.37.1000 or greater
- BlueField-2 firmware 24.37.1000 or greater

Chapter 3. Architecture

1. DOCA ETH provides a set of CPU functions to:
   a). Create a DOCA ETH context.
   b). Configure DOCA ETH context.
      i. Configure RX queue size.
      ii. Configure TX queue size.
      iii. Set TX checksum offloads.
   c). Get a GPU handle of the `DOCA_GPU_ETH` object.
   d). Destroy a DOCA ETH context.
2. The DOCA ETH context is configured on the CPU and then exported to the GPU:



3. Expected flow (`doca_eth_rxq`)
   a). Create a `DOCA_GPU` device handler.
   b). Create a `doca_eth_rxq` and configure its parameters (`type`, `num_packets`, `packet_size`, `packet_buffer`).
   c). Add a device to the context.
   d). Set the data path of the context to GPU.
   e). Start the context.


APP
DOCA_ETH_RXQ
DOCA_CTX
DOCA_MMAP
DOCA_BUF_ARR
DOCA_GPUNETIO
doca_gpu_create(const char*, struct doca_gpu**);
doca_eth_rxq_create(struct doca_eth_rxq**)
doca_eth_rxq_set_type(struct doca_eth_rxq*, enum doca_eth_rxq_type)
doca_eth_rxq_set_num_packets(struct doca_eth_rxq*, uint32_t)
doca_eth_rxq_set_max_packet_size(struct doca_eth_rxq*, uint16_t)
doca_eth_rxq_get_pkt_buffer_size(struct doca_eth_rxq*, uint16_t)
required_buffer_size
doca_gpu_mem_alloc(struct doca_gpu*, size, ...)
doca_mmap_create(union doca_data*, struct doca_mmap**)
doca_mmap_dev_add(struct doca_mmap*, struct doca_dev*)
doca_mmap_set_memrange(struct doca_mmap*, void*, size_t)
doca_mmap_start(struct doca_mmap*)
doca_eth_rxq_set_pkt_buffer(struct doca_eth_rxq*, struct doca_mmap*, uint32_t, uint32_t)
doca_eth_rxq_as_doca_ctx(struct doca_eth_rxq*)
struct doca_ctx*
doca_ctx_dev_add(struct doca_ctx*, struct doca_dev*)
doca_ctx_set_datapath_on_gpu(struct doca_ctx*, struct doca_gpu*)
doca_ctx_start(struct doca_ctx*)
doca_eth_rxq_get_gpu_handle(struct doca_eth_rxq*, struct doca_gpu_eth_rxq **)

Chapter 4. Dependencies

- DOCA ETH relies on the `doca_gpunetio` library to use GPU internal operations
- DOCA ETH relies on the `dpdk_bridge` library to allow interoperability between the library, `dpdk`, and `doca_flow`

Chapter 5. Limitations

- Data path on the CPU is not supported.
- RX/TX does not handle steering. Use DOCA Flow to do that.

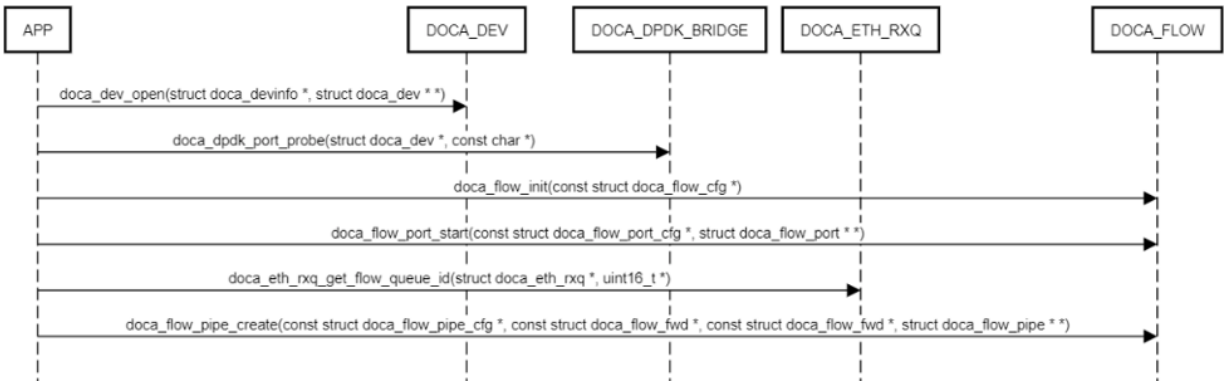
Chapter 6. DOCA Flow Interoperability

To use `DOCA_FLOW`:

1. Select a device.
2. Probe the device.

> Note: Only one port can be probed for the device.

3. Obtain the DOCA ETH queue ID.
4. Create a DOCA Flow pipe.



Chapter 7. API

This chapter provides a detailed description of the specific structures and operations related to the DOCA ETH control path API for general initialization, setup, and clean-up.

7.1. DOCA ETH RXQ

The `doca_eth_rxq` object allows querying device capabilities, setting object properties, creating and configuring an instance of `doca_eth_rxq`, and destroying it when no longer needed.

7.1.1. Querying Device Capabilities

Querying device capabilities provides information on the derived DOCA Ethernet limitations and enables users to set the properties of a `doca_eth_rxq` object accordingly.

The following subsections describe the functions which allow querying the device capabilities.

7.1.1.1. doca_eth_rxq_get_max_packet_size_supported()

This function queries the maximum packet size supported by the device. Any packet size bigger than this value is not accepted when trying to start the `doca_eth_rxq` object.

```
doca_error_t doca_eth_rxq_get_max_packet_size_supported(const struct doca_devinfo
 *devinfo, uint16_t *max_packet_size);
```

`devinfo [in]`
Pointer to a `doca_devinfo` instance.

`max_packet_size [out]`
Maximum packet size.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.1.2. doca_eth_rxq_get_type_supported()

This function checks if an RX queue type is supported by the device.

```
doca_error_t doca_eth_rxq_get_type_supported(const struct doca_devinfo
 *devinfo, enum doca_eth_rxq_type type, uint8_t *type_supported);
```

devinfo [in]
Pointer to a doca_devinfo instance.

type [in]
RX queue type (the values are documented in the header file).

type_supported [out]
Flag to indicate if the type is supported.

Returns
doca_error_t value. DOCA_SUCCESS if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2. Creating and Configuring DOCA_ETH_RXQ

7.1.2.1. doca_eth_rxq_create()

This function creates an instance of doca_eth_rxq.

```
doca_error_t doca_eth_rxq_create(struct doca_eth_rxq **eth_rxq);
```

eth_rxq [out]
Pointer to the newly created doca_eth_rxq instance.

Returns
doca_error_t value. DOCA_SUCCESS if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.2. doca_eth_rxq_destroy()

This function destroys a doca_eth_rxq instance.

```
doca_error_t doca_eth_rxq_destroy(struct doca_eth_rxq *eth_rxq);
```

eth_rxq [in]
Pointer to the doca_eth_rxq to be destroyed.

Returns
doca_error_t value. DOCA_SUCCESS if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.3. doca_eth_rxq_set_num_packets()

This function sets the number of packets the RX queue can hold.

Note: This function should only be called before invoking doca_ctx_start().

```
doca_error_t doca_eth_rxq_set_num_packets(struct doca_eth_rxq *eth_rxq, uint32_t
 num_packets);
```

eth_rxq [in]
Pointer to the doca_eth_rxq instance.

num_packets [in]
Maximum number of packets the queue can hold in the context.

Returns
doca_error_t value. DOCA_SUCCESS if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.4. doca_eth_rxq_set_max_packet_size()

This function sets the maximum Ethernet packet size the RX queue can accommodate.

Note: This function should only be called before invoking `doca_ctx_start()`.

```
doca_error_t doca_eth_rxq_set_max_packet_size(struct doca_eth_rxq *eth_rxq, uint16_t
 max_pkt_sz);
```

eth_rxq [in]
Pointer to the `doca_eth_rxq` instance.

max_pkt_sz [in]
Maximum packet size the RX queue can handle in the context.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.5. doca_eth_rxq_set_type()

This function sets the RX queue type.

Note: This function should only be called before invoking `doca_ctx_start()`.

```
doca_error_t doca_eth_rxq_set_type(struct doca_eth_rxq *eth_rxq, enum
 doca_eth_rxq_type type);
```

eth_rxq [in]
Pointer to the `doca_eth_rxq` instance.

type [in]
Type of the RX queue. Possible values are documented in the header file.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.6. doca_eth_rxq_get_flow_queue_id()

This function retrieves the DPDK queue ID of the receive queue, which can be used in `rte_flow` or `doca_flow`.

Note: This function should only be called after starting the context.

```
doca_error_t doca_eth_rxq_get_flow_queue_id(struct doca_eth_rxq *eth_rxq, uint16_t
 *flow_queue_id);
```

eth_rxq [in]
Pointer to the `doca_eth_rxq` instance.

flow_queue_id [out]
Queue ID, which can be used in `rte_flow` or `doca_flow`.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.7. doca_eth_rxq_get_gpu_handle()

This function retrieves the GPU handle of a `doca_eth_rxq` instance.

> Note: This function should only be called after starting the context.

The following is the expected flow:

1. Bind the context to a GPU device using `doca_ctx_set_data_path_on_gpu()`.
2. Start the context using `doca_ctx_start()`.
3. Call `doca_eth_rxq_get_gpu_handle()` to retrieve the `gpu_handle`.

```
doca_error_t doca_eth_rxq_get_gpu_handle(const struct doca_eth_rxq *eth_rxq, struct
 doca_gpu_eth_rxq **eth_rxq_ext);
```

`eth_rxq [in]`
Pointer to the `doca_eth_rxq` instance.

`eth_rxq_ext [out]`
Handle for the DOCA GPU `eth_rxq`.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.8. doca_eth_rxq_get_pkt_buffer_size()

This function retrieves the size for the Ethernet packet buffer of a `doca_eth_rxq`.

> Note: Use this value as the minimum size of the `doca_mmap` given to `doca_eth_rxq_set_pkt_buffer()`.

> Note: This function should only be called after `doca_eth_rxq_set_num_packets()` and `doca_eth_rxq_set_max_packet_size()`.

```
doca_error_t doca_eth_rxq_get_pkt_buffer_size(const struct doca_eth_rxq *eth_rxq,
 uint32_t *size);
```

`eth_rxq [in]`
Pointer to the `doca_eth_rxq` instance.

`size [out]`
Required size for the Ethernet packet buffer.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.1.2.9. doca_eth_rxq_set_pkt_buffer()

This function sets the Ethernet packet buffer of a `doca_eth_rxq`.

> Note: This function should only be called before invoking `doca_ctx_start()`.

```
doca_error_t doca_eth_rxq_set_pkt_buffer(struct doca_eth_rxq *eth_rxq, struct
 doca_mmap *mmap, uint32_t offset, uint32_t size);
```

`eth_rxq [in]`
Pointer to the `doca_eth_rxq` instance.

`mmap [in]`
Mmap with the memrange for the Ethernet packet buffer.

`size [in]`
Size of the Ethernet packet buffer.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2. DOCA ETH TXQ

The `doca_eth_txq` object allows querying device capabilities, setting object properties, creating and configuring an instance of `doca_eth_txq`, and destroying it when no longer needed.

7.2.1. Querying Device Capabilities

Querying device capabilities provides users with information regarding the derived DOCA Eth limitations and enables them to set the properties of a `doca_eth_txq` object accordingly.

The following subsections describe the functions which allow querying the device capabilities.

7.2.1.1. doca_eth_txq_get_max_queue_size_supported()

This function queries the maximum queue size supported by the device. Any queue size bigger than this value is not accepted when trying to start the `doca_eth_txq` object.

```
doca_error_t doca_eth_txq_get_max_queue_size_supported(const struct doca_devinfo
 *devinfo, uint32_t *max_queue_size);
```

`devinfo [in]`
Pointer to a `doca_devinfo` instance.

`max_queue_size [out]`
Maximum queue size supported by the device.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.1.2. doca_eth_txq_get_type_supported()

This function checks if a TX queue type is supported by the device.

```
doca_error_t doca_eth_txq_get_type_supported(const struct doca_devinfo
 *devinfo, enum doca_eth_txq_type type, uint8_t *type_supported);
```

devinfo [in]
Pointer to a `doca_devinfo` instance.

type [in]
TX queue type.

type_supported [out]
Flag to indicate if the type is supported.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.1.3. doca_eth_txq_get_chksum_offload_supported()

This function checks if checksum offload is supported by the device.

```
doca_error_t doca_eth_txq_get_chksum_offload_supported(const struct doca_devinfo
 *devinfo, uint8_t *offload_supported);
```

devinfo [in]
Pointer to a `doca_devinfo` instance.

offload_supported [out]
Flag to indicate if checksum offload is supported.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.1.4. doca_eth_txq_get_wait_on_time_offload_supported()

This function checks if wait-on-time offload is supported by the device.

```
doca_error_t doca_eth_txq_get_wait_on_time_offload_supported(const struct doca_dev
 *dev, enum doca_eth_wait_on_time *wait_on_time_mode);
```

dev [in]
Pointer to a `doca_dev` instance.

wait_on_time_mode [out]
Wait-on-time mode supported by the device. Valid mode values are documented in the header file.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2. Creating and Configuring DOCA_ETH_TXQ

7.2.2.1. doca_eth_txq_create()

This function creates an instance of `doca_eth_txq`.

```
doca_error_t doca_eth_txq_create(struct doca_eth_txq **eth_txq);
```

eth_txq [out]
Pointer to the newly created `doca_eth_txq` instance.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.2. doca_eth_txq_destroy()

This function destroys a `doca_eth_txq` instance.

```
doca_error_t doca_eth_txq_destroy(struct doca_eth_txq *eth_txq);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance to destroy.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.3. doca_eth_txq_set_queue_size()

This function sets the size of the TX queue.

Note: This function should only be called before starting the context (i.e., invoking `doca_ctx_start()`).

```
doca_error_t doca_eth_txq_set_queue_size(struct doca_eth_txq *eth_txq, uint32_t
 size);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance.

size [in]
TX queue size.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.4. doca_eth_txq_set_type()

This function sets the TX queue type.

Note: This function should only be called before starting the context (i.e., invoking `doca_ctx_start()`).

```
doca_error_t doca_eth_txq_set_type(struct doca_eth_txq *eth_txq, enum
 doca_eth_txq_type type);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance.

type [in]
Type of the TX queue. Possible values are documented in the header file.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.5. doca_eth_txq_set_l4_chksum_offload()

This function sets offload for the calculation of TCP/UDP checksum (L4) on transmitted packets.

Note: This function should only be called before starting the context (i.e., invoking `doca_ctx_start()`).

```
doca_error_t doca_eth_txq_set_l4_chksum_offload(struct doca_eth_txq *eth_txq);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.6. doca_eth_txq_set_l3_chksum_offload()

This function sets offload for the calculation of IPv4 checksum (L3) on transmitted packets.

Note: This function should only be called before starting the context (i.e., invoking `doca_ctx_start()`).

```
doca_error_t doca_eth_txq_set_l3_chksum_offload(struct doca_eth_txq *eth_txq);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.7. doca_eth_txq_set_wait_on_time_offload()

This function retrieves the GPU handle of a `doca_eth_txq` instance.

Note: This function should only be called before starting the context (i.e., invoking `doca_ctx_start()`).

```
doca_error_t doca_eth_txq_set_wait_on_time_offload(struct doca_eth_txq *eth_txq);
```

eth_txq [in]
Pointer to the `doca_eth_txq` instance.

Returns
`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

7.2.2.8. doca_eth_txq_get_gpu_handle()

This function retrieves the GPU handle of a `doca_eth_txq` instance.

> Note: This function should only be called after starting the context (i.e., invoking `doca_ctx_start()`).

The following is the expected flow:

1. Bind the context to a GPU device using `doca_ctx_set_data_path_on_gpu()`.
2. Start the context using `doca_ctx_start()`.
3. Call `doca_eth_txq_get_gpu_handle()` to retrieve the `gpu_handle`.

```
doca_error_t doca_eth_txq_get_gpu_handle(const struct doca_eth_txq *eth_txq, struct
 doca_gpu_eth_txq **eth_txq_ext);
```

`eth_txq [in]`
Pointer to the `doca_eth_txq` instance.

`eth_txq_ext [out]`
Handle for the DOCA GPU `eth_rxq`.

Returns

`doca_error_t` value. `DOCA_SUCCESS` if successful or an error value upon failure. Possible error values are documented in the header file.

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 nor any of its direct or indirect subsidiaries and affiliates (collectively: “NVIDIA”) make no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assume 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, the NVIDIA logo, and Mellanox are trademarks and/or registered trademarks of Mellanox Technologies Ltd. and/or NVIDIA Corporation in the U.S. and in other countries. The registered trademark Linux® is used pursuant to a sublicense from the Linux Foundation, the exclusive licensee of Linus Torvalds, owner of the mark on a world¬wide basis. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com