NVIDIA DOCA Rivermax

Programming Guide

MLNX-15-060569 _v2.2.0 | October 2023

Table of Contents

Chapter 1. Introduction

DOCA Rivermax (RMAX) is a DOCA API for NVIDIA Rivermax, an optimized networking SDK for media and data streaming applications. Rivermax leverages NVIDIA® BlueField® DPU hardware streaming acceleration technology which enables direct data transfers to and from the GPU, delivering best-in-class throughput and latency with minimal CPU utilization for streaming workloads.

This document is intended for software developers wishing to accelerate their networking operations.

Chapter 2. Prerequisites

DOCA Rivermax-based applications can run on the target DPU.

It is recommended to have at least 800 huge pages enabled to achieve maximum performance:

```
dpu> echo 1000000000 > /proc/sys/kernel/shmmax
dpu> echo 800 > /proc/sys/vm/nr_hugepages
```

Chapter 3. Architecture

DOCA Rivermax (RMAX) library contains two objects:

- `doca_rmax_flow` – is a flow object that represents an IP/port tuple
- `doca_rmax_in_stream` – represents the input stream and can be thought of as a receive queue which scatters the received data into memory. Each stream can receive one or more flows.

Chapter 4. Dependencies

The library requires Rivermax (RMAX) library to compile and run and Rivermax license to run applications. Refer to NVIDIA Rivermax SDK page to get the license.

Note: Currently, the DOCA Rivermax library is supported for BlueField-2 only.

Chapter 5. API

For the library API reference, refer to DOCA Rivermax (RMAX) API documentation in NVIDIA DOCA Libraries API Reference Manual.

Note: The pkg-config (`*.pc` file) for the DOCA Rivermax library is included in DOCA's regular definitions (i.e., `doca-rmax`).

The following sections provide additional details about the library API.

5.1. Objects

5.1.1. struct doca_rmax_in_stream

Represents Rivermax's input stream. This is a main object in the RX part.

5.1.2. struct doca_rmax_flow

Represents the steering flow for the input stream to filter incoming data flow by match criteria.

5.2. Library Initialization

Users must explicitly call library initialization and deinitialization functions. Global parameters such as internal thread affinity mask must be set before initialization.

5.2.1. doca_rmax_set_cpu_affinity_mask()

Set the CPU affinity mask for the Rivermax (RMAX) internal thread. This must be called before library initialization.

The function `doca_rmax_get_cpu_affinity_mask()` can be used to query the CPU affinity mask for the Rivermax internal thread at any time.

```
doca_error_t doca_rmax_set_cpu_affinity_mask(const struct
 doca_rmax_cpu_affinity_mask * mask);
```

mask [in]
Affinity mask. The CPU is included in the affinity mask if the corresponding bit is set. By default, affinity mask is not set so the internal thread can run on any CPU core.

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

5.2.2. doca_rmax_init()

This function initializes the DOCA Rivermax (RMAX) global resources. This function must be called after `doca_rmax_set_cpu_affinity_mask()` and before any other DOCA RMAX library call.

```
doca_error_t doca_rmax_init(void);
```

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

5.2.3. doca_rmax_release()

This function cleans up the DOCA Rivermax (RMAX) resources. No DOCA Rivermax function may be called after calling this function.

```
doca_error_t doca_rmax_release(void);
```

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

5.3. Checking Device Capability

DOCA Rivermax (RMAX) can query additional device capabilities related to the library.

5.3.1. doca_rmax_get_ptp_clock_supported()

Query PTP clock capability for device.

```
doca_error_t doca_rmax_get_ptp_clock_supported(const struct doca_devinfo * devinfo);
```

devinfo [in]
Device to query

Returns
`doca_error_t` value

- `DOCA_SUCCESS`– PTP clock is supported
- `DOCA_ERROR_NOT_SUPPORTED` – PTP clock is not supported
- Other possible error values are documented in the header file

5.4. Stream Life Cycle

The following sections detail the stages of the stream life cycle from creation to teardown.

5.4.1. Create and Configure Input Stream

5.4.1.1. doca_rmax_in_stream_create()

Create a DOCA Rivermax (RMAX) input stream context. The stream can be downcasted to the DOCA context using the function `doca_rmax_in_stream_as_ctx()`.

```
doca_error_t doca_rmax_in_stream_create(struct doca_rmax_in_stream ** stream);
```

stream [out]
The input stream context created for DOCA Rivermax. Non-NULL upon success, NULL otherwise.

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

5.4.1.2. doca_rmax_in_stream_set_*() and doca_rmax_in_stream_get_*()

Use `doca_rmax_in_stream_set_*()` functions to set the properties of the input stream and the corresponding `doca_rmax_in_stream_get_*()` functions to retrieve the current properties of the input stream.

5.4.1.2.1. Header-data Split Mode

By default, the DOCA Rivermax (RMAX) input stream places the whole packet data into a buffer (`memblk_count = 1`).

To enable header-data split mode when the headers and payload are placed to a different buffers, set the property `memblk_count` (`doca_rmax_in_stream_set_memblks_count`) to 2. In this case you must also set a non-zero value for the `min_size`/`max_size` property of memory descriptor (`doca_rmax_in_stream_memblk_desc_set_min_size`/`doca_rmax_in_stream_memblk_desc_set_max_siz`

5.4.1.2.1.1. doca_rmax_in_stream_set_memblks_count()

Set the number of configured memory blocks.

```
doca_error_t doca_rmax_in_stream_set_memblks_count(struct doca_rmax_in_stream *
 stream, uint32_t value);
```

stream [in]
The input stream to write the property.

value [in]
Property value.

Returns

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

5.4.1.2.2. Mandatory Properties

5.4.1.2.2.1. doca_rmax_in_stream_set_elements_count()

Set number of elements in the stream buffer. The actual number of elements in a buffer can be increased by the library, so the actual value can be queried using `doca_rmax_in_stream_get_elements_count()`.

```
doca_error_t doca_rmax_in_stream_set_elements_count(struct doca_rmax_in_stream *
 stream, uint32_t value);
```

stream [in]

The input stream to write the property.

value [in]

Property value.

Returns

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

5.4.1.2.2.2. doca_rmax_in_stream_memblk_desc_set_min_size()

Set minimal packet segment size(s). The value is an array of the minimal packet segment sizes received by the input stream.

```
doca_error_t doca_rmax_in_stream_memblk_desc_set_min_size(struct doca_rmax_in_stream
 * stream, uint16_t * value);
```

stream [in]

The input stream to write the property.

value [in]

- When `memblk_count`=1, pointer to a variable that contains the packet size
- When `memblk_count`>1, pointer to an array of packet sizes

Returns

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

5.4.1.2.2.3. doca_rmax_in_stream_memblk_desc_set_max_size()

Set maximal packet segment size(s). The value is an array of the maximal packet segment sizes received by the input stream.

```
doca_error_t doca_rmax_in_stream_memblk_desc_set_min_size(struct doca_rmax_in_stream
 * stream, uint16_t * value);
```

stream [in]

The input stream to write the property.

value [in]

- When `memblk_count`=1, pointer to a variable that contains the packet size
- When `memblk_count`>1, pointer to an array of packet sizes

Returns

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

5.4.1.2.3. Optional Properties

The following properties have a default value and may be set as long as the input stream is not yet active.

5.4.1.2.3.1. doca_rmax_in_stream_set_type()

Set the input stream type.

```
doca_error_t doca_rmax_in_stream_set_type(struct doca_rmax_in_stream * stream, enum
 doca_rmax_in_stream_type value);
```

stream [in]

The input stream to write the property.

value [in]

Property value.

Returns

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

5.4.1.2.3.2. doca_rmax_in_stream_set_scatter_type()

Set the type of packet's data scatter:

- All packet data including network headers
- Only User-Level Protocol data (discard network header up to L4)
- Payload data only (all headers will be discarded)

```
doca_error_t doca_rmax_in_stream_set_scatter_type(struct doca_rmax_in_stream *
 stream, enum doca_rmax_in_stream_scatter_type value);
```

stream [in]

The input stream to write the property.

value [in]

Property value.

Returns

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

5.4.1.2.3.3. doca_rmax_in_stream_set_timestamp_format()

Set stream timestamp format.

```
doca_error_t doca_rmax_in_stream_set_scatter_type(struct doca_rmax_in_stream *
 stream, enum doca_rmax_in_stream_scatter_type value);
```

stream [in]

The input stream to write the property.

value [in]

Property value.

Returns

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

5.4.1.2.4. Run-time Properties

The following properties have a default value and may be set at any time.

5.4.1.2.4.1. doca_rmax_in_stream_set_min_packets()

Set minimal number of packets that the input stream must return in a read event.

```
doca_error_t doca_rmax_in_stream_set_min_packets(struct doca_rmax_in_stream *
 stream, uint32_t value);
```

stream [in]
The input stream to write the property.

value [in]
Property value.

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

5.4.1.2.4.2. doca_rmax_in_stream_set_max_packets()

Set the maximal number of packets that the input stream must return in read event.

```
doca_error_t doca_rmax_in_stream_set_max_packets(struct doca_rmax_in_stream *
 stream, uint32_t value);
```

stream [in]
The input stream to write the property.

value [in]
Property value.

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

5.4.1.2.4.3. doca_rmax_in_stream_set_timeout_us()

Set receive timeout. The number of µsecs that library would do busy wait (polling) for reception of at least `min_packets` number of packets.

```
doca_error_t doca_rmax_in_stream_set_timeout_us(struct doca_rmax_in_stream * stream,
 int value);
```

stream [in]
The input stream to write the property.

value [in]
Property value.

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

5.4.2. Attach Network Device

Attach a network device to the stream context using the `doca_ctx_dev_add()` function. Only one device per stream is supported.

5.4.3. Query Buffer and Stride Size

Query the memory block size to determine the required size of the memory buffers.

5.4.3.1. doca_rmax_in_stream_get_memblk_size()

Get the size of the memory blocks.

```
doca_error_t doca_rmax_in_stream_get_memblk_size(struct doca_rmax_in_stream *
 stream, size_t * value);
```

stream [in]
The input stream to write the property.

value [out]
Size of the memory block (array of sizes for multiple memblks, the number of memory blocks in stream is more than one).

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

5.4.3.2. doca_rmax_in_stream_get_memblk_stride_size()

Get stride sizes.

```
doca_error_t doca_rmax_in_stream_get_memblk_stride_size(struct doca_rmax_in_stream *
 stream, uint16_t * value);
```

stream [in]
The input stream to write the property.

value [out]
Stride size of the memory block (array of stride sizes for multiple memory blocks)

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

5.4.4. Create and Set Memory Buffer (doca_buf)

Allocate the memory of at least one previously queried size and then configure the input stream to use this buffer.

For header-data split mode, users must use two buffers and the buffers for the header and the payload data must be chained. See `doca_buf_list_chain()` for more information.

5.4.4.1. doca_rmax_in_stream_set_memblk()

Set memory buffer(s) to be used as received data storage.


Note: Must be set before starting the stream context.

```
doca_error_t doca_rmax_in_stream_set_memblk(struct doca_rmax_in_stream * stream,
 struct doca_buf * buf);
```

stream [in]
The input stream to write the property.

buf [in]
Memory buffer (or head of linked list of memory buffers) to store received data. The length of the linked list must be the same as the number of memory blocks configured.

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

5.4.5. Start Stream Context

Cast a stream object to the DOCA context and start the context.

5.4.6. Attach to WorkQ

Use the `doca_ctx_workq_add()` function.

5.4.7. Attach Flows

You can attach one or more flows to a stream after start.

5.4.7.1. doca_rmax_flow_attach()

Attach a flow to a stream.

```
doca_error_t doca_rmax_flow_attach (const struct doca_rmax_flow * flow, const struct
 doca_rmax_in_stream * stream);
```

flow [in]
Flow to operate on.

buf [in]
The context for attaching a 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.

5.4.8. Receive Data

To initiate receiving of packets, submit job of type `DOCA_RMAX_ACTION_TYPE_RX_DATA` (structure `doca_rmax_job_rx_data`). The `user_data` field of the job structure is not used and should be set to 0.

You can retrieve available input packets by polling WorkQ progress. Use `doca_workq_progress_retrieve()` to query available data. Once this function returns with code `DOCA_SUCCESS` and action type `DOCA_RMAX_ACTION_TYPE_RX_DATA` you must resubmit the job to continue receiving data.


Note: Event-driven mode is not yet supported.

If progress status is `DOCA_SUCCESS`, then event of type `DOCA_RMAX_ACTION_TYPE_RX_DATA` contains a pointer to the `doca_rmax_in_stream_completion` structure in the field of the `result.ptr` event.

5.4.8.1. doca_rmax_in_stream_completion

Completion is returned by the input stream describing the incoming packets.

```
struct doca_rmax_in_stream_completion {
        uint32_t elements_count;
        uint64_t ts_first;
        uint64_t ts_last;
        uint32_t seqn_first;
        uint32_t memblk_ptr_arr_len;
        void **memblk_ptr_arr;
};
```

elements_count
Number of packets received.

ts_first
Time of arrival of the first packet.

ts_last
Time of arrival of the last packet.

seqn_first
Sequence number of the first packet.

memblk_ptr_arr_len
Number of memory blocks placed in `memblk_ptr_arr`.

memblk_ptr_arr
Array of pointers to the beginning of the memory block as configured by the input stream create step. The offset between packets inside the memory block is a stride size that can be queried using `doca_rmax_in_stream_get_memblk_stride_size`.

If the progress status is `DOCA_ERROR_IO_FAILED`, then the event of type `DOCA_RMAX_ACTION_TYPE_RX_DATA` would contain a pointer to `the doca_rmax_stream_error` structure in the event's `result.ptr` field.

5.4.8.2. doca_rmax_stream_error

Detailed completion error information.

```
struct doca_rmax_stream_error {
        int code;
        const char *message;
};
```

code
Raw Rivermax error code.

message
Human-readable error message.

Progress status `DOCA_ERROR_AGAIN` signifies that no data is available yet. All other codes should be handled as a DOCA error.

5.4.9. Teardown

5.4.9.1. Detach Flows

All flows must be detached from an input stream before stopping it.

5.4.9.1.1. doca_rmax_flow_detach()

Detach a flow from a stream.

```
doca_error_t doca_rmax_flow_detach (const struct doca_rmax_flow * flow, const struct
 doca_rmax_in_stream * stream);
```

flow [in]
Flow to operate on.

buf [in]
The context for detaching a 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.

5.4.9.2. Detach from WorkQ

Use the `doca_ctx_workq_rm()` function.

5.4.9.3. Stop Stream Context

Cast the stream object to the DOCA context and stop the context.

5.4.9.4. Detach Network Device

Use the `doca_ctx_dev_rm()` function.

5.4.9.5. Destroy Stream

5.4.9.5.1. doca_rmax_in_stream_destroy()

Destroy a DOCA input stream context. Free all allocated resources associated with a DOCA Rivermax (RMAX) input stream.

```
doca_error_t doca_rmax_in_stream_destroy(struct doca_rmax_in_stream * stream);
```

stream [in]
The context 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.

5.5. Flow Functions

5.5.1. Create Flow

5.5.1.1. doca_rmax_flow_create()

Create a steering flow for the input stream to filter the incoming data flow by match criteria.

```
doca_error_t doca_rmax_flow_create(struct doca_rmax_flow ** flow);
```

flow [out]
The context 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.

5.5.2. Configure Flow Properties

5.5.2.1. doca_rmax_flow_set_src_ip()

Set the source IP filter for the flow.

```
doca_error_t doca_rmax_flow_set_src_ip(struct doca_rmax_flow * flow, const struct
 in_addr * ip);
```

flow [in]
Flow to operate on.

ip [in]
Source IPv4 address.

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

5.5.2.2. doca_rmax_flow_set_dst_ip()

Set the destination IP filter for the flow.

```
doca_error_t doca_rmax_flow_set_dst_ip(struct doca_rmax_flow * flow, const struct
 in_addr * ip);
```

flow [in]
Flow to operate on.

ip [in]
Destination IPv4 address.

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

5.5.2.3. doca_rmax_flow_set_src_port()

Set the source port filter for the flow.

```
doca_error_t doca_rmax_flow_set_src_port(struct doca_rmax_flow * flow, uint16_t
 port);
```

flow [in]
Flow to operate on.

port [in]
Source port number. If zero, then any source port is accepted.

Returns

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

5.5.2.4. doca_rmax_flow_set_dst_port()

Set the destination port filter for the flow.

```
doca_error_t doca_rmax_flow_set_dst_port(struct doca_rmax_flow * flow, uint16_t
 port);
```

flow [in]

Flow to operate on.

port [in]

Destination port number (non-zero)

Returns

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

5.5.2.5. doca_rmax_flow_set_tag()

Set the tag for the flow.

```
doca_error_t doca_rmax_flow_set_tag(struct doca_rmax_flow * flow, uint32_t tag);
```

flow [in]

Flow to operate on.

tag [in]

Non-zero tag

Returns

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

5.5.3. Destroy Flow

5.5.3.1. doca_rmax_flow_destroy()

Set the tag for the flow.

```
doca_error_t doca_rmax_flow_destroy(struct doca_rmax_flow * flow);
```

flow [in]

Flow 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.

Chapter 6. Packaging

DOCA Rivermax (RMAX) is distributed separately from other DOCA libraries:

- `doca-rmax-libs` – library binary
- `libdoca-rmax-libs-dev` – header files and samples

DOCA Rivermax binaries depend on:

- `doca-runtime`
- `rivermax`

DOCA Rivermax headers and samples:

- `doca-sdk`
- `rivermax`
- `rivermax-ext`

Chapter 7. DOCA Rivermax Samples

Please refer to the NVIDIA DOCA Rivermax Sample Guide for more information about the API of this DOCA library.

7.1. Running the Sample

1. Refer to the following documents:
   - NVIDIA DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - NVIDIA DOCA Troubleshooting Guide for any issue you may encounter with the installation, compilation, or execution of DOCA samples.
2. To build a given sample:

```
cd /opt/mellanox/doca/samples/doca_rmax/<sample_name>
meson build
ninja -C build
```

> Note: The binary `doca_<sample_name>` will be created under `./build/`.

3. Sample (e.g., `doca_rivermax_create_stream`) usage:

```
Usage: doca_rivermax_create_stream [DOCA Flags] [Program Flags]
DOCA Flags:
  -h, --help                    Print a help synopsis
  -v, --version                 Print program version information
  -l, --log-level               Set the log level for the program <CRITICAL=20,
 ERROR=30, WARNING=40, INFO=50, DEBUG=60>

Program Flags:
  -p, --pci_addr <PCI-ADDRESS>  PCI device address
```

> Note: When running DOCA Rivermax (RMAX) samples, the IPv4 address 192.168.105.2 must be configured to an available uplink prior to running it for the samples to run as expected:
>
> ```
> $ifconfig p0 192.168.105.2
> ```

4. For additional information per sample, use the `-h` option:

```
./build/doca_<sample_name> -h
```

7.2. Samples

7.2.1. List Devices

This sample illustrates how to list all available devices, dump their IPv4 addresses, and tell whether or not the PTP clock is supported.

The sample logic includes:

1. Initializing DOCA Rivermax (RMAX) library.
2. Iterating over the available devices.
3. Dumping their IPv4 addresses
4. Dumping whether a PTP clock is supported for each device.
5. Releasing DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/rmax_list_devices_sample.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/rmax_list_devices_main.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_list_devices/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`; `/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

7.2.2. Set CPU Affinity

This sample illustrates how to set the CPU affinity mask for Rivermax internal thread to achieve better performance. This parameter must be set before library initialization; otherwise, it is not applied.

The sample logic includes:

1. Setting CPU affinity using the DOCA Rivermax (RMAX) API.
2. Initializing DOCA Rivermax library.
3. Releasing DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/rmax_set_affinity_sample.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/rmax_set_affinity_main.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_affinity/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`; `/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

7.2.3. Set Clock

This sample illustrates how to set the PTP clock device to be used internally in DOCA Rivermax (RMAX).

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initializing the DOCA Rivermax library.
3. Setting the device to use for obtaining PTP time.
4. Releasing the DOCA Rivermax library.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/rmax_set_clock_sample.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/rmax_set_clock_main.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_set_clock/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`; `/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

7.2.4. Create Stream

This sample illustrates how to create a stream, create a flow and attach it to the created stream, and finally to start receiving data buffers (based on the attached flow).

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initializing the DOCA Rivermax (RMAX) library.
3. Creating an input stream.
4. Creating the context from the created stream.
5. Initializing DOCA core related objects.
6. Setting the attributes of the created stream.
7. Creating a flow and attaching it to the created stream.
8. Starting to receive data buffers.
9. Clean up—detaches flow and destroys it, destroys created stream and DOCA core related objects.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/rmax_create_stream_sample.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/rmax_create_stream_main.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream/meson.build`

- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`; `/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

7.2.5. Create Stream – Header-data Split Mode

This sample illustrates how to create a stream in header-data split mode when packet headers and payload are split to different RX buffers.

The sample logic includes:

1. Opening a DOCA device with a given PCIe address.
2. Initialize the DOCA Rivermax (RMAX) library.
3. Creating an input stream.
4. Creating a context from the created stream.
5. Initializing DOCA core related objects.
6. Setting attributes of the created stream. Chaining buffers and setting header size to non-zero is essential to create a stream with header-data split mode.
7. Creating a flow and attaching it to the created stream.
8. Starting to receive data to split buffers.
9. Clean up—detaches flow and destroys it, destroys created stream and DOCA core related objects.

Note: When running "Create Stream" samples, with or without header-split mode, an IPv4 address of value 192.168.105.2 must be configured to the required PCIe device prior to the run so the samples could run as expected.

If a different IPv4 address is configured, the user can either change the IPv4 address to 192.168.105.2 or change the IPv4 addresses in `init_config()` function found in the samples source code to the configured one before building and compiling the samples.

References:

- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/rmax_create_stream_hds_sample.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/rmax_create_stream_hds_main.c`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_create_stream_hds/meson.build`
- `/opt/mellanox/doca/samples/doca_rmax/rmax_common.h`; `/opt/mellanox/doca/samples/doca_rmax/rmax_common.c`

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