



NVIDIA DOCA PCC Application Guide

Table of contents

1. Introduction

2. System Design

3. Application Architecture

4. DOCA Libraries

5. Dependencies

6. Compiling the Application

7. Running the Application

8. Application Code Flow

9. Port Programmable Congestion Control Register

10. References

This document provides a DOCA PCC implementation on top of NVIDIA® BlueField® networking platform .

1. Introduction


Programmable Congestion Control (PCC) allows users to design and implement their own congestion control (CC) algorithm, giving them the flexibility to work out an optimal solution to handle congestion in their clusters. On BlueField-3 networking platforms , PCC is provided as a component of DOCA.

The application leverages the [DOCA PCC API](#) to provide users the flexibility to manage allocation of DPA resources according to their requirements.

Typical DOCA application includes App running on host/Arm and App running on DPA. Developers are advised to use the host/Arm application with minimal changes and focus on developing their algorithm and integrating it into the DPA application.

2. System Design

DOCA PCC application consists of two parts:

- Host/Arm app is the control plane. It is responsible for allocating all resources and handover to the DPA app initially, then destroying everything when the DPA app finishes its operation. The host app must always be alive to stay in control while the device app is working.
- Device/DPA app is the data plane.
 - The default mode of the data plane is running as a reaction point (RP). When the first thread is activated, DPA App initialization is done in the DOCA PCC library by calling the algorithm initialization function implemented by the user in the app. Moreover, the user algorithm execution function is called when a CC event arrives. The user algorithm takes event data as input and performs a calculation, using per-flow context, and replies with the updated rate value and a flag to send an RTT request. The following is an illustration of the general RP application flow:

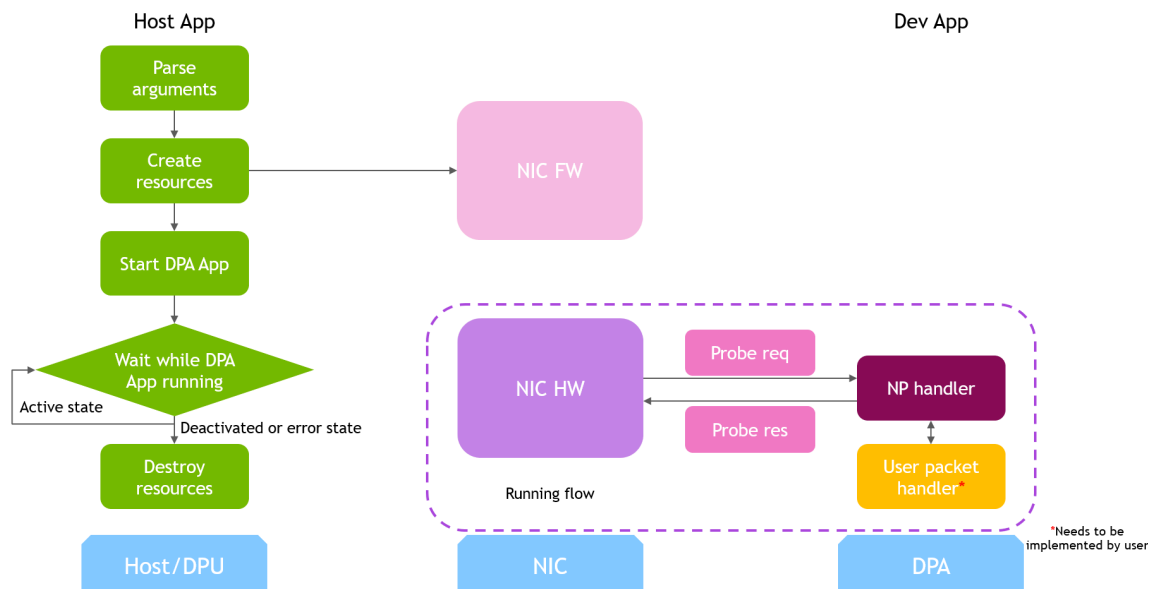
The host/Arm application sends a command to the BlueField platform firmware when allocating or destroying resources. CC events are generated by the BlueField platform hardware automatically when sending data or receiving ACK/NACK/CNP/RTT packets, then the device application handles these events

by calling the user algorithm. After the DPA application replies to hardware, handling of current event is done, and the next event can arrive.

i Info

The device/DPA app can also run different algorithms for the RP program, which users can configure as a runtime option.

- The device/DPA app can function as a notification point (NP). When a new probe request packet arrives, the user handler can read and analyze the data and send a probe response back. The following is an illustration of the general NP application flow:



i Info

The device/DPA app is as well capable of functioning as a telemetry program for a NP NIC or switch operations, which users can configure as a runtime option.

3. Application Architecture

```
/opt/mellanox/doca/applications/pcc/  
  host  
    pcc.c  
    pcc_core.c  
    pcc_core.h  
  device  
    pcc_common_dev.h  
    rp  
      rtt_template  
        algo  
          rtt_template.h  
            rtt_template_algo_params.h  
            rtt_template_ctxt.h  
            rtt_template.c  
          rp_rtt_template_dev_main.c  
        switch_telemetry  
          algo  
            telem_template.h  
              telem_template_algo_params.h  
              telem_template_ctxt.h  
              telem_template.c  
            rp_switch_telemetry_dev_main.c  
          np  
            np_nic_telemetry_dev_main.c  
            np_switch_telemetry_dev_main.c
```

The main content of the reference DOCA PCC application files are the following:

- `host/pcc.c` – entry point to entire application

- `host/pcc_core.c` – host functions to initialize and destroy the PCC application resources, parsers for PCC command line parameters
- `device/pcc_common_dev.h` – common util calls and definitions for device programs
- `device/rp/rtt_template/rp_rtt_template_dev_main.c` – callbacks for user CC algorithm initialization, user CC algorithm calculation and algorithm parameter change notification of the RTT template algorithm reference
- `device/rp/rtt_template/algo/*` – user CC algorithm reference for RTT template. Put user algorithm code here
- `device/rp/switch_telemetry/rp_switch_telemetry_dev_main.c` – callbacks for user CC algorithm initialization, user CC algorithm calculation, and algorithm parameter change notification of the switch telemetry algorithm reference
- `device/rp/switch_telemetry/algo/*` – user CC algorithm reference for switch telemetry. Put user algorithm code here.
- `device/np/np_nic_telemetry_dev_main.c` – callback for user NP handling, implemented as a NIC telemetry program to observe RX counters
- `device/np/np_switch_telemetry_dev_main.c` – callback for user NP handling, implemented as a switch telemetry program to observe last hop switch metadata

4. DOCA Libraries

This application leverages the following DOCA library:

- [DOCA PCC](#)

Refer to its respective programming guide for more information.

5. Dependencies

- NVIDIA BlueField-3 Platform is required
- Firmware 32.38.1000 and higher

- MFT 4.25 and higher

6. 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/pcc/.
```

6.1 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
```

i Info

`doca_pcc` is created under `/tmp/build/pcc/`.

6.2 Compiling Only the Current Application

To directly build only the PCC application:

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

i Info

`doca_pcc` is created under `/tmp/build/pcc/`.

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_pcc` to `true`

2. Run the following compilation commands :

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

Info

`doca_pcc` is created under `/tmp/build/pcc/`.

6.3 Compilation Options

The application offers specific compilation flags which one can set for a desired behavior in the device/DPA program.

In the `meson_options.txt` file, one can find the following options:

- `enable_pcc_application_tx_counter_sampling`: set to `true` to use TX counters sampled at runtime in the RP CC handling algorithm.
- `enable_pcc_application_np_rx_rate`: set to `true` to use RX counters received from NP in the RP CC handling algorithm.

6.4 Troubleshooting

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

7. Running the Application

7.1 Prerequisites

Enable `USER_PROGRAMMABLE_CC` in `mlxconfig`:

```
mlxconfig -y -d /dev/mst/mt41692_pciconf0 set
USER_PROGRAMMABLE_CC=1
```

Perform a [BlueField system reboot](#) for the `mlxconfig` settings to take effect.

7.2 Application Execution

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

1. Application usage instructions:

```
Usage: doca_pcc [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:
```

-d, --device <IB device names> IB device name that supports PCC (mandatory).

-np-nt, --np-nic-telemetry <PCC Notification Point NIC Telemetry> Flag to indicate running as a Notification Point NIC Telemetry (optional). The application will generate CCMAD probe packets. By default the flag is set to false.

-rp-st, --rp-switch-telemetry <PCC Reaction Point Switch Telemetry> Flag to indicate running as a Reaction Point Switch Telemetry (optional). The application will generate IFA2 probe packets. By default the flag is set to false.

-np-st, --np-switch-telemetry <PCC Notification Point Switch Telemetry> Flag to indicate running as a Notification Point Switch Telemetry (optional). The application will generate IFA2 probe packets. By default the flag is set to false.

-t, --threads <PCC threads list> A list of the PCC threads numbers to be chosen for the DOCA PCC context to run on (optional). Must be provided as a string, such that the number are separated by a space.

-w, --wait-time <PCC wait time> The duration of the DOCA PCC wait (optional), can provide negative values which means infinity. If not provided then -1 will be chosen.

-r-handler, --remote-sw-handler <CCMAD remote SW handler> CCMAD remote SW handler flag (optional). If not provided then false will be chosen.

-hl, --hop-limit <IFA2 hop limit> The IFA2 probe packet hop limit (optional). If not provided then 0xFE will be chosen.

-gns, --global-namespace <IFA2 global namespace> The IFA2 probe packet global namespace (optional). If not provided then 0xF will be chosen.

-gns-ignore_mask, --global-namespace-ignore-mask <IFA2 global namespace ignore mask> The IFA2 probe packet global namespace ignore mask (optional). If not provided then 0 will be chosen.

-gns-ignore_val, --global-namespace-ignore-value <IFA2 global namespace ignore value> The IFA2 probe packet global

namespace ignore value (optional). If not provided then 0 will be chosen.

-f, --coredump-file <PCC coredump file> A pathname to the file to write coredump data in case of unrecoverable error on the device (optional). Must be provided as a string.

-i, --port-id <Physical port ID> The physical port ID of the device running the application (optional). If not provided then ID 0 will be chosen.

Info

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

```
./doca_pcc -h
```

Info

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

2. CLI example for running the application on the BlueField Platform or the host:

```
./doca_pcc -d mlx5_0
```

Note

The IB device identifier (`m1x5_0`) should match the identifier of the desired IB device.

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

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

For example:

```
./doca_pcc --json ./pcc_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.

7.3 Command Line Flags

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
General flags	<code>h</code>	<code>help</code>	Prints a help synopsis	N/A
	<code>v</code>	<code>version</code>	Prints program version information	N/A
	<code>l</code>	<code>log-level</code>	Sets the log level for the program: <ul style="list-style-type: none">• DISABLE=10	N/A

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
			<ul style="list-style-type: none"> • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 <div style="background-color: #ffffcc; padding: 10px; margin-top: 10px;"> <p>i Info The application uses a unique logging implementation that makes use of DOCA's logging levels.</p> </div>	
	N/A	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">sdk-log-level</div>	Sets the log level for the program: <ul style="list-style-type: none"> • DISABLE=10 • CRITICAL=20 • ERROR=30 • WARNING=40 • INFO=50 • DEBUG=60 • TRACE=70 	N/A
	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">j</div>	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">json</div>	Parse all command flags from an input JSON file	N/A
Program flags	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">d</div>	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">device</div>	IB device name that supports PCC	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">"device</div>
	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">np-nt</div>	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">np-nic-telemetry</div>	(Optional) Flag to indicate running as a NP NIC telemetry. The DOCA PCC application can run as a NP NIC telemetry program. If this flag is used, the application loads a program to	<div style="border: 1px solid #ccc; padding: 2px; display: inline-block;">"np-ni</div>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
			run on the DPA to sample RX NIC counters and send them in a response packet.	
	rp-st	rp-switch-telemetry	(Optional) Flag to indicate running as a RP switch telemetry. The DOCA PCC application can run as a RP switch telemetry program. If this flag is used, the application loads a program to run on the DPA of a switch telemetry algorithm which receives metadata from the last hop switch congestion point from the NP node.	"rp-sw
	np-st	np-switch-telemetry	(Optional) Flag to indicate running as a NP switch telemetry. The DOCA PCC application can run as a NP switch telemetry program. If this flag is used, the application loads a program to run on the DPA to sample metadata from the last hop switch congestion point and send them in response packet.	"np-sw
	t	threads	(Optional) A list of the PCC EU indexes to be chosen for the DOCA PCC event handler threads to run on. Must be provided as a string, such that the numbers are separated by a space. The placement of the PCC threads per core can be controlled using the EU indexes. Utilizing a large number of EUs, while limiting the number of threads per core, gives the best event handling rate and lowest event latency. The last EU is used for communication with the BlueField Platform while all others are for data path CC event handling.	"pcc-threads" 184 1 200 2 216 2 232 2
 Note				

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
			<p>If <code>np-nic-telemetry</code> option is chosen by the user, a different set of threads will be chosen as default list.</p>	
	<code>w</code>	<code>wait-time</code>	(Optional) In seconds, the duration of the DOCA PCC wait. Negative values mean infinity.	<code>"wait-t</code>
	<code>r-handler</code>	<code>remote-sw-handler</code>	<p>(Optional) CCMAD remote SW handler flag. Relevant for RP contexts. This flag indicates whether the expected CCMAD probe packet responses are generated by a remote DOCA NP process or not.</p> <p>Note If using other probe types than CCMAD, probe packet responses are always expected to be generated from a remote DOCA NP process.</p>	<code>"remo</code>
	<code>hl</code>	<code>hop-limit</code>	(Optional) The IFA2 probe packet hop limit	<code>"hop-l</code>

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
			<p>i Info Relevant for RP contexts.</p>	
	gns	global-namespace	<p>(Optional) The IFA2 probe packet global namespace</p> <p>i Info Relevant for RP contexts.</p>	"globa
	gns-ignore-mask	global-namespace-ignore-mask	<p>(Optional) The IFA2 probe packet global namespace ignore mask</p> <p>i Info Relevant for NP contexts.</p>	"globa
	gns-ignore-val	global-namespace-ignore-value	<p>(Optional) The IFA2 probe packet global namespace ignore value</p> <p>i Info Relevant for NP contexts.</p>	"globa

Flag Type	Short Flag	Long Flag/JSON Key	Description	JSON Co
	f	coredump-file	(Optional) A pathname to the file to write core dump data if an unrecoverable error occurs on the device	"corec
	i	port-id	(Optional) The physical port ID of the device running the application	"port-i

Info

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

7.4 Troubleshooting

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

8. Application Code Flow

This section lists the application's configuration flow, explaining the different DOCA function calls and wrappers.

1. Parse application argument.
 1. Initialize arg parser resources and register DOCA general parameters.

```
doca_argp_init();
```

2. Register PCC application parameters.

```
register_pcc_params();
```

3. Parse the arguments.

```
doca_argp_start();
```

1. Parse DOCA flags.
2. Parse DOCA PCC parameters.

2. PCC initialization.

```
pcc_init();
```

1. Open DOCA device that supports PCC.
2. Create DOCA PCC context.
3. Configure affinity of threads handling CC events.

3. Start DOCA PCC.

```
doca_pcc_start();
```

1. Create PCC process and other resources.

2. Trigger initialization of PCC on device.
3. Register the PCC in the BlueField Platform hardware so CC events can be generated and an event handler can be triggered.
4. Process state monitor loop.

```
doca_pcc_get_process_state();
doca_pcc_wait();
```

1. Get the state of the process:

State	Description
DOCA_PCC_PS_ACTIVE = 0	The process handles CC events (only one process is active at a given time)
DOCA_PCC_PS_STANDBY = 1	The process is in standby mode (another process is already ACTIVE)
DOCA_PCC_PS_DEACTIVATED = 2	The process has been deactivated by the BlueField Platform firmware and should be destroyed
DOCA_PCC_PS_ERROR = 3	The process is in error state and should be destroyed

2. Wait on process events from the device.

5. PCC destroy.

```
doca_pcc_destroy();
```

1. Destroy PCC resources. The process stops handling PCC events.
2. Close DOCA device.
6. Arg parser destroy.

```
doca_argp_destroy()
```

9. Port Programmable Congestion Control Register

The Port Programmable Congestion Control (PPCC) register allows the user to configure and read PCC algorithms and their parameters/counters.

It supports the following functionalities:

- Enabling different algorithms on different ports
- Querying information of both algorithms and tunable parameters/counters
- Changing algorithm parameters without compiling and reburning user image
- Querying or clearing programmable counters

9.1 Usage

The PPCC register can be accessed using a string similar to the following:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op  
"cmd_type=0" --reg_name PPCC --indexes  
"local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"  
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set "cmd_type=1" --  
reg_name PPCC --indexes  
"local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

Where you must:

- Set the `cmd_type` and the indexes
- Give values for `algo_slot`, `algo_param_index`
- Keep `local_port=1`, `pnat=0`, `lp_msb=0`

- Keep `doca_pcc` application running

cmd_type	Description	Method	Index	Input (in --set)	Output
<code>0x0</code>	Get algorithm info	Get		N/A	<ul style="list-style-type: none"> • Value – 3 <code>algo_n</code> algo is an index • Text – algorithm description • <code>sl_bitr</code> – indicate device support <code>sl_bitr</code>
<code>0x1</code>	Enable algorithm	Set		<code>sl_bitmask</code> <code>trace_en</code> <code>counter_en</code>	N/A
<code>0x2</code>	Disable algorithm	Set	<code>algo_slot</code>	N/A	N/A
<code>0x3</code>	Get algorithm enabling status	Get		N/A	<ul style="list-style-type: none"> • Value: <ul style="list-style-type: none"> ◦ 0 – ◦ 1 – • <code>sl_bitr</code> field allow specific <code>sl_bitr</code> the bitma • <code>sl_bitr</code> – indicate device support <code>sl_bitr</code>
<code>0x4</code>	Get number of parameters	Get		N/A	<ul style="list-style-type: none"> • Value – number of algorithm
<code>0x5</code>	Get parameter information	Get	<code>algo_slot</code> <code>algo_param_index</code>	N/A	<ul style="list-style-type: none"> • <code>param_</code> default value • <code>param_</code> value of parameter

cmd_type	Description	Method	Index	Input (in --set)	Output
					<ul style="list-style-type: none"> param_ value of p prm - <ul style="list-style-type: none"> 0: r 1: r 2: r may usir clea
0x6	Get parameter value	Get		N/A	<ul style="list-style-type: none"> Value - p
0x7	Get and clear parameter	Get		N/A	<ul style="list-style-type: none"> Value - p
0x8	Set parameter value	Set		Parameter value	N/A
0xA	Bulk get parameters	Get	algo_slot	N/A	<ul style="list-style-type: none"> text_lo num x 4 text[0] param va
0xB	Bulk set parameters	Set		text_length - param num x 4 text[0]... text[n] - param values	N/A
0xC	Bulk get counters	Get		N/A	<ul style="list-style-type: none"> text_lo counter r text[0] counter v
0xD	Bulk get and clear counters	Get		N/A	<ul style="list-style-type: none"> text_lo counter r

cmd_type	Description	Method	Index	Input (in --set)	Output
					<ul style="list-style-type: none"> text[0] counter v
0xE	Get number of counters	Get		N/A	<ul style="list-style-type: none"> Value - n of algo
0xF	Get counter information	Get	algo_slot algo_param_index	N/A	<ul style="list-style-type: none"> param_ value of p prm - <ul style="list-style-type: none"> 0: re 1: re 2: re may "ge" con
0x10	Get algorithm info array	Get	N/A	N/A	<ul style="list-style-type: none"> text_l slot initia text[0] 32-bit a if no algo available index

9.2 Internal Default Algorithm

The internal default algorithm is used when enhanced connection establishment (ECE) negotiation fails. It is mainly used for backward compatibility and can be disabled using "force mode". Otherwise, users may change `doca_pcc_dev_user_algo()` in the device app to run a specific algorithm without considering the algorithm negotiation.

The force mode command is per port:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=2"
--reg_name PPCC --indexes
"local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
```

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0.1 -y --get --op "cmd_type=2"
--reg_name PPCC --indexes
"local_port=1,pnat=0,lp_msb=0,algo_slot=15,algo_param_index=0"
```

9.3 Counters

Counters are shared on the port and are only enabled on one `algo_slot` per port. The following command enables the counters while enabling the algorithm according to the `algo_slot`:

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --set
"cmd_type=1,counter_en=1" --reg_name PPCC --indexes
"local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

After counters are enabled on the `algo_slot`, they can be queried using `cmd_type` 0xC or 0xD.

```
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=12"
--reg_name PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
sudo mlxreg -d /dev/mst/mt41692_pciconf0 -y --get --op "cmd_type=13"
--reg_name PPCC --indexes "local_port=1,pnat=0,lp_msb=0,algo_slot=0,algo_param_index=0"
```

10. References

- `/opt/mellanox/doca/applications/pcc/`
- `/opt/mellanox/doca/applications/pcc/pcc_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 NON-INFRINGEMENT, 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