NVIDIA DOCA PCC Application Guide
# Table of contents

- **Introduction**
- **System Design**
- **Application Architecture**
- **DOCA Libraries**
- **Dependencies**
- **Compiling the Application**
  - Compiling All Applications
  - Compiling Only the Current Application
  - Compilation Options
  - Troubleshooting
- **Running the Application**
  - Prerequisites
  - Application Execution
  - Command Line Flags
  - Troubleshooting
- **Application Code Flow**
- **Port Programmable Congestion Control Register**
  - Usage
  - Internal Default Algorithm
  - Counters
- **References**
This document provides a DOCA PCC implementation on top of NVIDIA® BlueField® networking platform.

**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.

**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.

- 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:
**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.

**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
```
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/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/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

**DOCA Libraries**
This application leverages the following DOCA library:

- **DOCA PCC**

Refer to its respective programming guide for more information.

**Dependencies**

- NVIDIA BlueField-3 Platform is required
- Firmware 32.38.1000 and higher
- MFT 4.25 and higher

**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/
```
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
```

ℹ️ **Info**

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

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
```

ℹ️ **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:

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

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

### 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.
Troubleshooting

Refer to the NVIDIA DOCA Troubleshooting Guide for any issue encountered with the compilation of the application.

Running the Application

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.

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.
-tp-st, --tp-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:
2. CLI example for running the application on the BlueField Platform or the host:

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

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.

### Command Line Flags

<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>General flags</td>
<td>h</td>
<td>help</td>
<td>Prints a help synopsis</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>v</td>
<td>version</td>
<td>Prints program version information</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>l</td>
<td>log-level</td>
<td>Sets the log level for the program:</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DISABLE=10</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- CRITICAL=20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- ERROR=30</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- WARNING=40</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- INFO=50</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- DEBUG=60</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- TRACE=70</td>
<td></td>
</tr>
</tbody>
</table>

### Info

The application uses a
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>N/A</td>
<td>sdk-log-level</td>
<td></td>
<td>Sets the log level for the program:</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>json</td>
<td>Parse all command flags from an input JSON file</td>
<td>N/A</td>
</tr>
<tr>
<td>Program</td>
<td>d</td>
<td>device</td>
<td>IB device name that supports PCC</td>
<td>&quot;device&quot;: &quot;&quot;</td>
</tr>
<tr>
<td>Flag Type</td>
<td>Short Flag</td>
<td>Long Flag/JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>flags</td>
<td>np-nt</td>
<td>np-nic-telemetry</td>
<td>(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 run on the DPA to sample RX NIC counters and send them in a response packet.</td>
<td>&quot;np-nic-telemetry&quot;: false</td>
</tr>
<tr>
<td>flags</td>
<td>rp-st</td>
<td>rp-switch-telemetry</td>
<td>(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.</td>
<td>&quot;rp-switch-telemetry&quot;: false</td>
</tr>
<tr>
<td>flags</td>
<td>np-st</td>
<td>np-switch-telemetry</td>
<td>(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.</td>
<td>&quot;np-switch-telemetry&quot;: false</td>
</tr>
<tr>
<td>Flag Type</td>
<td>Short Flag</td>
<td>Long Flag/JSON Key</td>
<td>Description</td>
<td>JSON Content</td>
</tr>
<tr>
<td>-----------</td>
<td>------------</td>
<td>-------------------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>t</td>
<td>threads</td>
<td></td>
<td>(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.</td>
<td>&quot;pcc-threads&quot;:  &quot;176 177 178 179 180 181 182 183 184 185 186 187 192 193 194 195 196 197 198 199 200 201 202 203 208 209 210 211 212 213 214 215 216 217 218 219 224 225 226 227 228 229 230 231 232 233 234 235 240&quot;</td>
</tr>
</tbody>
</table>

**Note**

If np-nic-telemetry option is chosen by the user, a different set of threads will be chosen as...
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>default list.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>wait-time</td>
<td>(Optional) In seconds, the duration of the DOCA PCC wait. Negative values mean infinity.</td>
<td>&quot;wait-time&quot;: -1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>remote-sw-handler</td>
<td>(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.</td>
<td>&quot;remote-sw-handler&quot;: false</td>
</tr>
</tbody>
</table>

**Note**

If using other probe types than CCMAD, probe packet responses are always expected to be generated from a remote
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hop-limit</td>
<td>(Optional) The IFA2 probe packet hop limit</td>
<td>&quot;hop-limit&quot;: 0xFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>global-namespace</td>
<td>(Optional) The IFA2 probe packet global namespace</td>
<td>&quot;global-namespace&quot;: 0xF</td>
</tr>
<tr>
<td></td>
<td>gns-ignore-mask</td>
<td>global-namespace-ignore-mask</td>
<td>(Optional) The IFA2 probe packet global namespace ignore mask</td>
<td>&quot;global-namespace-ignore-mask&quot;: 0</td>
</tr>
</tbody>
</table>

**Info**

- Relevant for RP contexts.
<table>
<thead>
<tr>
<th>Flag Type</th>
<th>Short Flag</th>
<th>Long Flag/JSON Key</th>
<th>Description</th>
<th>JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>gns-ignore-val</td>
<td>(Optional) The IFA2 probe packet global namespace ignore value</td>
<td>&quot;global-nameSpace-ignore-value&quot;: 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>global-namespace-ignore-value</td>
<td>Info Relevant for NP contexts.</td>
<td>&quot;coredump-file&quot;: &quot;/tmp/doca_pcc_coredump.txt&quot;</td>
</tr>
<tr>
<td></td>
<td>f</td>
<td>coredump-file</td>
<td>(Optional) A pathname to the file to write core dump data if an unrecoverable error occurs on the device</td>
<td>&quot;port-id&quot;: 0</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>port-id</td>
<td>(Optional) The physical port ID of the device running the application</td>
<td>&quot;port-id&quot;: 0</td>
</tr>
</tbody>
</table>

ℹ️ **Info**

Refer to DOCA Arg Parser for more information regarding the supported flags and execution modes.
Troubleshooting

Refer to the NVIDIA DOCA Troubleshooting Guide for any issue encountered with the installation or execution of the DOCA applications.

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.


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

1. Get the state of the process:

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

2. Wait on process events from the device.
5. PCC destroy.

1. Destroy PCC resources. The process stops handling PCC events.

2. Close DOCA device.

6. Arg parser destroy.

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

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

<table>
<thead>
<tr>
<th>cmd_type</th>
<th>Description</th>
<th>Method</th>
<th>Index</th>
<th>Input (in --set)</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Get algorithm info</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>- Value – 32-bit algo_num or 0 if no algo is available at this index</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Text – algorithm description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- <code>sl_bitmask_support</code> – indicates whether the device supports sl_bitmask logic</td>
</tr>
<tr>
<td>0x1</td>
<td>Enable algorithm</td>
<td>Set</td>
<td><code>sl_bitmask</code> <code>trace_en counter_en</code></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>Disable algorithm</td>
<td>Set</td>
<td><code>algo_slot</code></td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Value:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 0 – disabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- 1 – enabled</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- <code>sl_bitmask</code> – this field allows to apply to specific SLs based on the bitmask</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- <code>sl_bitmask_support</code> – indicates whether the device supports sl_bitmask logic</td>
</tr>
<tr>
<td>0x3</td>
<td>Get algorithm enabling status</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>- Value – num of params of algo</td>
</tr>
<tr>
<td>0x4</td>
<td>Get number of parameters</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>- Value – num of params of algo</td>
</tr>
<tr>
<td>cmd_type</td>
<td>Description</td>
<td>Method</td>
<td>Index</td>
<td>Input (in -- set)</td>
<td>Output</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------</td>
<td>--------</td>
<td>-------</td>
<td>------------------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>0x5</td>
<td>Get parameter information</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• param_value1 – default value of param</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• param_value2 – min value of param</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• param_value3 – max value of param</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• prm –</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 0: read-only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 1: read-write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 2: read-only but may be cleared</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>using the &quot;get and clear&quot; command</td>
</tr>
<tr>
<td>0x6</td>
<td>Get parameter value</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• Value – param value</td>
</tr>
<tr>
<td>0x7</td>
<td>Get and clear parameter</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• Value – param value</td>
</tr>
<tr>
<td>0x8</td>
<td>Set parameter value</td>
<td>Set</td>
<td>Parameter value</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>0xA</td>
<td>Bulk get parameters</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• text_length – param num x 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• text[0]...text[n] – param values</td>
</tr>
<tr>
<td>0xB</td>
<td>Bulk set parameters</td>
<td>Set</td>
<td>text_length - param num x 4</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>text[0]...text[n] - param values</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0xC</td>
<td>Bulk get counters</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• text_length – counter num x 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• text[0]...text[n] – counter values</td>
</tr>
<tr>
<td>cmd_type</td>
<td>Description</td>
<td>Method</td>
<td>Index</td>
<td>Input (in --set)</td>
<td>Output</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------------------------</td>
<td>--------</td>
<td>-------</td>
<td>-----------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0xD</td>
<td>Bulk get and clear counters</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• text_length – counter num x 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• text[0]...text[n] – counter values</td>
</tr>
<tr>
<td>0xE</td>
<td>Get number of counters</td>
<td>Get</td>
<td>N/A</td>
<td></td>
<td>• Value – num of counters of algo</td>
</tr>
<tr>
<td>0xF</td>
<td>Get counter information</td>
<td>Get</td>
<td>algo_slot</td>
<td>algo_param_index</td>
<td>• param_value3 – max value of parameter</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
<td>• prm –</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 0: read-only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 1: read-write</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>○ 2: read-only but may be cleared via &quot;get &amp; clear&quot; command</td>
</tr>
<tr>
<td>0x10</td>
<td>Get algorithm info array</td>
<td>Get</td>
<td>N/A</td>
<td>N/A</td>
<td>• text_length – algo slot initialized x 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>• text[0]...text[n] – 32-bit algo_num or 0 if no algorithm is available at this slot index</td>
</tr>
</tbody>
</table>

**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"
```
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:

```bash
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"
```

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

```bash
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"
```

```bash
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"
```

```bash
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"
```

References

- /opt/mellanox/doca/applications/pcc/
- /opt/mellanox/doca/applications/pcc/pcc_params.json

© Copyright 2024, NVIDIA. PDF Generated on 08/15/2024