Sync Event
Table of contents

Introduction

Prerequisites

Environment

Architecture

DOCA Sync Event Objects

Configuration Phase

Operation Modes

Synchronous Mode

Asynchronous Mode

Configurations

Mandatory Configurations

Optional Configurations

Export DOCA Sync Event to Another Execution Unit

Device Support

Execution Phase

DOCA Sync Event Data Path Operations

Publishing on DOCA Sync Event

Subscribe on DOCA Sync Event

Tasks

Get Task

Notify Set Task

Sync Event
Notify Add Task
Wait Equal-to Task
Wait Not-equal-to Task

Events

State Machine

Idle
Starting
Running
Stopping

DOCA Sync Event Tear Down

Stopping DOCA Sync Event
Destroying DOCA Sync Event

Alternative Datapath Options

GPU Datapath
DPA Datapath

DOCA Sync Event Sample

Running DOCA Sync Event Sample

Samples
Sync Event Remote PCIe
Sync Event Local PCIe
Introduction

DOCA Sync Event (SE) is a software synchronization mechanism for parallel execution across the CPU, DPU, DPA and remote nodes. The SE holds a 64-bit counter which can be updated, read, and waited upon from any of these units to achieve synchronization between executions on them.

To achieve the best performance, DOCA SE defines a subscriber and publisher locality, where:

- Publisher – the entity which updates (sets or increments) the event value
- Subscriber – the entity which gets and waits upon the SE

Based on hints, DOCA selects memory locality of the SE counter, closer to the subscriber side. Each DOCA SE is configured with a single publisher location and a single subscriber location.
location which can be the CPU or DPU.

The SE control path happens on the CPU (either host CPU or DPU CPU) through the DOCA SE CPU handle. It is possible to retrieve different execution-unit-specific handles (DPU/DPA/GPU/remote handles) by exporting the SE instance through the CPU handle. Each SE handle refers to the DOCA SE instance from which it is retrieved. By using the execution-unit-specific handle, the associated SE instance can be operated from that execution unit.

In a basic scenario, synchronization is achieved by updating the SE from one execution and waiting upon the SE from another execution unit.

**Prerequisites**

DOCA SE can be used as a context which follows the architecture of a DOCA Core Context, it is recommended to read the following sections of the DOCA Core page before proceeding:

- [DOCA Execution Model](#)
- [DOCA Device](#)
- [DOCA Memory Subsystem](#)

**Environment**

DOCA SE based applications can run either on the host machine or on the NVIDIA® BlueField® DPU target and can involve DPA, GPU and other remote nodes.

Using DOCA SE with DPU requires BlueField to be configured to work in DPU mode as described in [NVIDIA BlueField Modes of Operation](#).

**Info**

Asynchronous wait on a DOCA SE requires NVIDIA® BlueField-3® or newer.
Architecture

DOCA SE can be converted to a DOCA Context as defined by DOCA Core. See DOCA Context for more information.

As a context, DOCA SE leverages DOCA Core architecture to expose asynchronous tasks/events offloaded to hardware.

The figure that follows demonstrates components used by DOCA SE. DOCA Device provides information on the capabilities of the configured HW used by SE to control system resources.

DOCA DPA, GPUNetIO, and RDMA modules are required for cross-device synchronization (could be DPA, GPU, or remote peer respectively).

DOCA SE allows flexible memory management by its ability to specify an external buffer, where a DOCA mmap module handles memory registration for advanced synchronization scenarios.

For asynchronous operation scheduling, SE uses the DOCA Progress Engine (PE) module.

**DOCA Sync Event Components Diagram**

The following diagram represents DOCA SE synchronization abilities on various devices.

**DOCA Sync Event Interaction Diagram**
DOCA Sync Event Objects

DOCA SE exposes different types of handles per execution unit as detailed in the following table.

<table>
<thead>
<tr>
<th>Execution Unit</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU (host/DPU)</td>
<td>struct doca_sync_event</td>
<td>Handle for interacting with the SE from the CPU</td>
</tr>
<tr>
<td>DPU</td>
<td>struct doca_sync_event</td>
<td>Handle for interacting with the SE from the DPU</td>
</tr>
<tr>
<td>DPA</td>
<td>doca_dpa_dev_sync_event_t</td>
<td>Handle for interacting with the SE from the DPA</td>
</tr>
<tr>
<td>GPU</td>
<td>doca_gpu_dev_sync_event_t</td>
<td>Handle for interacting with the SE from the GPU</td>
</tr>
<tr>
<td>Remote net CPU</td>
<td>doca_sync_event_remote_net</td>
<td>Handle for interacting with the SE from a remote CPU</td>
</tr>
<tr>
<td>Execution Unit</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>Remote net DPA</td>
<td>doca_dpa_dev_sync_event_remote_net_t</td>
<td>Handle for interacting with the SE from a remote DPA</td>
</tr>
<tr>
<td>Remote net GPU</td>
<td>doca_gpu_dev_sync_event_remote_net_t</td>
<td>Handle for interacting with the SE from a remote GPU</td>
</tr>
</tbody>
</table>

Each one of these handle types has its own dedicated API for creating the handle and interacting with it.

### Configuration Phase

Any DOCA SE creation starts with creating CPU handle by calling `doca_sync_event_create` API.

After creation, the SE entity could be shared with local PCIe, remote CPU, DPA, and GPU by a dedicated handle creation via the DOCA SE export flow, as illustrated in the following diagram:

![Diagram showing handle creation and sharing between CPU, DPA, and GPU](image)

### Operation Modes

DOCA SE exposes two different APIs for starting it depending on the desired operation mode, synchronous or asynchronous.

**Note**
Sync Event

Once started, SE operation mode cannot be changed.

**Synchronous Mode**

Start the SE to operate in synchronous mode by calling `doca_sync_event_start`.

In synchronous operation mode, each data path operation (get, update, wait) blocks the calling thread from continuing until the operation is done.

**Note**

An operation is considered done if the requested change fails and the exact error can be reported or if the requested change has taken effect.

**Asynchronous Mode**

To start the SE to operate in asynchronous mode, convert the SE instance to `doca_ctx` by calling `doca_sync_event_as_ctx`. Then use DOCA CTX API to start the SE and DOCA PE API to submit tasks on the SE (see section "DOCA Progress Engine" for more).

**Configurations**

**Mandatory Configurations**

These configurations must be set by the application before attempting to start the SE:

- DOCA SE CPU handle must be configured by providing the runtime hints on the publisher and subscriber locations. Both the subscriber and publisher locations
must be configured using the following APIs:

- `doca_sync_event_add_publisher_location_<cpu|dpa|gpu|remote_pci|remote_net>`
- `doca_sync_event_add_subscriber_location_<cpu|dpa|gpu|remote_pci>`

- For the asynchronous use case, at least one task/event type must be configured. See configuration of tasks.

### Optional Configurations

#### Info

If these configurations are not set, a default value is used.

- These configurations provide an 8-byte buffer to be used as the backing memory of the SE. If set, it is user responsibility to handle the memory (i.e., preserve the memory allocated during DOCA SE lifecycle and free it after DOCA SE destruction). If not provided, the SE backing memory is allocated by the SE.
  - `doca_sync_event_set_addr`
  - `doca_sync_event_set_doca_buf`

### Export DOCA Sync Event to Another Execution Unit

To use an SE from an execution unit other than the CPU, it must be exported to get a handle for the specific execution unit:

- DPA – `doca_sync_event_get_dpa_handle` returns a DOCA SE DPA handle `(doca_dpa_dev_sync_event_t)` which can be passed to the DPA SE data path APIs from the DPA kernel
- GPU – `doca_sync_event_get_gpu_handle` returns a DOCA SE GPU handle (`doca_gpu_dev_sync_event_t`) which can be passed to the GPU SE data path APIs for the CUDA kernel.

- DPU – `doca_sync_event_export_to_remote_pci` returns a blob which can be used from the DPU CPU to instantiate a DOCA SE DPU handle (`struct doca_sync_event`) using the `doca_sync_event_create_from_export` function.

DOCA SE allows notifications from remote peers (remote net) utilizing capabilities of the DOCA RDMA library. The following figure illustrates the remote net export flow:

- Remote net CPU – `doca_sync_event_export_to_remote_net` returns a blob which can be used from the remote net CPU to instantiate a DOCA SE remote net CPU handle (`struct doca_sync_event_remote_net`) using the `doca_sync_event_remote_net_create_from_export` function. The handle can be used directly for submitting asynchronous tasks through the `doca_rdma` library or exported to the remote DPA/GPU.

- Remote net DPA – `doca_sync_event_remote_net_get_dpa_handle` returns a DOCA SE remote net DPA handle (`doca_dpa_dev_sync_event_remote_net_t`) which can be passed to the DPA RDMA data path APIs from a DPA kernel.

- Remote net GPU – `doca_sync_event_remote_net_get_gpu_handle` returns a DOCA SE remote net GPU handle (`doca_gpu_dev_sync_event_remote_net_t`) which can be passed to the GPU RDMA data path APIs from a CUDA kernel.

---

**Note**
The CPU handle (struct `doca_sync_event`) can be exported only to the location where the SE is configured.

**Note**

Prior to calling any export function, users must first verify it is supported by calling the corresponding export capability getter:
- `doca_sync_event_cap_is_export_to_dpa_supported`,
- `doca_sync_event_cap_is_export_to_gpu_supported`,
- `doca_sync_event_cap_is_export_to_remote_pci_supported`, or
- `doca_sync_event_cap_is_export_to_remote_net_supported`.

**Note**

Prior to calling any `*_create_from_export` function, users must first verify it is supported by calling the corresponding create from the export capability getter: `doca_sync_event_cap_is_create_from_export_supported` or `doca_sync_event_cap_remote_net_is_create_from_export_supported`.

**Note**

Once created from an export, both the SE DPU handle `struct doca_sync_event` and the SE remote net CPU handle `struct doca_sync_event_remote_net` cannot be configured, but only the SE DPU handle must be started before it is used.

**Warning**
Device Support

DOCA SE needs a device to operate. For instructions on picking a device, see DOCA Core device discovery.

Info

Both NVIDIA® BlueField® -2 and BlueField® -3 devices are supported as well as any doca_dev is supported.

Info

Asynchronous wait (blocking/polling) is supported on NVIDIA® BlueField® -3 and NVIDIA® ConnectX®-7 and later.

As device capabilities may change in the future (see DOCA Capability Check), it is recommended to choose your device using any relevant capability method (starting with the prefix doca_sync_event_cap_*).

Capability APIs to query whether sync event can be constructed from export blob:

- doca_sync_event_cap_is_create_from_export_supported
- doca_sync_event_cap_remote_net_is_create_from_export_supported

Capability APIs to query whether sync event can be exported to other execution units:
- `doca_sync_event_cap_is_export_to_remote_pci_supported`
- `doca_sync_event_cap_is_export_to_dpa_supported`
- `doca_sync_event_cap_is_export_to_gpu_supported`
- `doca_sync_event_cap_is_export_to_remote_net_supported`
- `doca_sync_event_cap_remote_net_is_export_to_dpa_supported`
- `doca_sync_event_cap_remote_net_is_export_to_gpu_supported`

Capability APIs to query whether an asynchronous task is supported:

- `doca_sync_event_cap_task_get_is_supported`
- `doca_sync_event_cap_task_notify_set_is_supported`
- `doca_sync_event_cap_task_notify_add_is_supported`
- `doca_sync_event_cap_task_wait_eq_is_supported`
- `doca_sync_event_cap_task_wait_neq_is_supported`

**Execution Phase**

This section describes execution on CPU. For additional execution environments refer to section "Alternative Datapath Options".

**DOCA Sync Event Data Path Operations**

The DOCA SE synchronization mechanism is achieved using exposed datapath operations. The API exposes a function for "writing" to the SE and for "reading" the SE.

The **synchronous API** is a set of functions which can be called directly by the user, while the asynchronous API is exposed by defining a corresponding `doca_task` for each synchronous function to be submitted on a DOCA PE (see [DOCA Progress Engine](#) and [DOCA Context](#) for additional information).
The following subsections describe the DOCA SE datapath operation with respect to synchronous and asynchronous operation modes.

**Publishing on DOCA Sync Event**

**Setting DOCA Sync Event Value**

Users can set DOCA SE to a 64-bit value:

- Synchronously by calling `doca_sync_event_update_set`
- Asynchronously by submitting a `doca_sync_event_task_notify_set` task

**Adding to DOCA Sync Event Value**

Users can atomically increment the value of a DOCA SE:

- Synchronously by calling `doca_sync_event_update_add`
- Asynchronously by submitting a `doca_sync_event_task_notify_add` task
Subscribe on DOCA Sync Event

Getting DOCA Sync Event Value

Users can get the value of a DOCA SE:

- Synchronously by calling `doca_sync_event_get`
- Asynchronously by submitting a `doca_sync_event_task_get` task

Waiting on DOCA Sync Event

Waiting for an event is the main operation for achieving synchronization between different execution units.

Users can wait until an SE reaches a specific value in a variety of ways.

Synchronously

`doca_sync_event_wait_gt` waits for the value of a DOCA SE to be greater than a specified value in a "polling busy wait" manner (100% processor utilization). This API enables users to wait for an SE in real time.

`doca_sync_event_wait_gt_yield` waits for the value of a DOCA SE to be greater than a specified value in a "periodically busy wait" manner. After each polling iteration, the calling thread relinquishes the CPU, so a new thread gets to run. This API allows a tradeoff between real-time polling to CPU starvation.

`doca_sync_event_wait_eq` waits for the value of a DOCA SE to be equal to a specified value in a "polling busy wait" manner (100% processor utilization). This API enables users to wait for an SE in real time.

`doca_sync_event_wait_eq_yield` waits for the value of a DOCA SE to be equal to a specified value in a "periodically busy wait" manner. After each polling iteration, the calling thread relinquishes the CPU so a new thread gets to run. This API allows a tradeoff between real-time polling to CPU starvation.
**doca_sync_event_wait_neq** waits for the value of a DOCA SE to not be equal to a specified value in a "polling busy wait" manner (100% processor utilization). This API enables users to wait for an SE in real time.

**doca_sync_event_wait_neq_yield** waits for the value of a DOCA SE to not be equal to a specified value in a "periodically busy wait" manner. After each polling iteration, the calling thread relinquishes the CPU so a new thread gets to run. This API allows a tradeoff between real-time polling to CPU starvation.

---

**Note**

This wait method is supported only from the CPU.

---

**Asynchronously**

DOCA SE exposes an asynchronous wait method by defining a `doca_sync_event_task_wait_eq` and `doca_sync_event_task_wait_neq` tasks.

Users can wait for wait-job completion in the following methods:

- **Blocking** – get a `doca_event_handle_t` from the `doca_pe` to blocking-wait on
- **Polling** – poll the wait task by calling `doca_pe_progress`

---

**Info**

Asynchronous wait (blocking/polling) is supported on BlueField-3 and ConnectX-7 and later.

---

**Note**
Tasks

DOCA SE context exposes asynchronous tasks that leverage the DPU hardware according to the DOCA Core architecture. See DOCA Core Task.

Get Task

The get task retrieves the value of a DOCA SE.

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td>doca_sync_event_task_get_set_conf</td>
<td>doca_sync_event_cap_task_get_is_supported</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_sync_event_task_get_set_conf</td>
<td>-</td>
</tr>
</tbody>
</table>

Task Input

Common input described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return value</td>
<td>8-bytes memory pointer to hold the DOCA SE value</td>
</tr>
</tbody>
</table>

Task Output
Common output described in DOCA Core Task.

**Task Completion Success**

After the task is completed successfully, the return value memory holds the DOCA SE value.

**Task Completion Failure**

If the task fails midway:

- The context may enter a stopping state if a fatal error occurs
- The return value memory may be modified

**Task Limitations**

All limitations are described in DOCA Core Task.

**Notify Set Task**

The notify set task allows setting the value of a DOCA SE.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td>doca_sync_event_task_notify_set_set_conf</td>
<td>doca_sync_event_cap_task_notify_set_is_supported</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_sync_event_task_notify_set_set_conf</td>
<td>-</td>
</tr>
</tbody>
</table>
**Task Input**

Common input described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set value</td>
<td>64-bit value to set the DOCA SE value to</td>
</tr>
</tbody>
</table>

**Task Output**

Common output described in DOCA Core Task.

**Task Completion Success**

After the task is completed successfully, the DOCA SE value is set to the given set value.

**Task Completion Failure**

If the task fails midway, the context may enter a stopping state if a fatal error occurs.

**Task Limitations**

This operation is not atomic. Other limitations are described in DOCA Core Task.

**Notify Add Task**

The notify add task allows atomically setting the value of a DOCA SE.

**Task Configuration**
<table>
<thead>
<tr>
<th>Description</th>
<th>API to Set the Configuration</th>
<th>API to Query Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td>doca_sync_event_task_notify_add_set_conf</td>
<td>doca_sync_event_cap_task_notify_add_is_supported</td>
</tr>
<tr>
<td>Number of tasks</td>
<td>doca_sync_event_task_notify_add_set_conf</td>
<td>-</td>
</tr>
</tbody>
</table>

**Task Input**

Common input described in [DOCA Core Task](#).

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increment value</td>
<td>64-bit value to atomically increment the DOCA SE value by</td>
</tr>
<tr>
<td>Fetched value</td>
<td>8-bytes memory pointer to hold the DOCA SE value before the increment</td>
</tr>
</tbody>
</table>

**Task Output**

Common output described in [DOCA Core Task](#).

**Task Completion Success**

After the task is completed successfully, the following occurs:

- The DOCA SE value is incremented according to the given increment value
- The fetched value memory holds the DOCA SE value before the increment

**Task Completion Failure**

If the task fails midway:
- The context may enter a stopping state if a fatal error occurs
- The fetched value memory may be modified.

**Task Limitations**

All limitations are described in **DOCA Core Task**.

**Wait Equal-to Task**

The wait-equal task allows atomically waiting for a DOCA SE value to be equal to some threshold.

**Task Configuration**

<table>
<thead>
<tr>
<th>Description</th>
<th>API to set the configuration</th>
<th>API to query support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_sync_event_task_wait_eq_set_conf</code></td>
<td><code>doca_sync_event_cap_task_wait_eq_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_sync_event_task_wait_eq_set_conf</code></td>
<td>-</td>
</tr>
</tbody>
</table>

**Task Input**

Common input described in **DOCA Core Task**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait threshold</td>
<td>64-bit value to wait for the DOCA SE value to be equal to</td>
</tr>
<tr>
<td>Mask</td>
<td>64-bit mask to apply on the DOCA SE value before comparing with the wait threshold</td>
</tr>
</tbody>
</table>
Task Output

Common output described in DOCA Core Task.

Task Completion Success

After the task is completed successfully, the following occurs:

- The DOCA SE value is equal to the given wait threshold.

Task Completion Failure

If the task fails midway, the context may enter a stopping state if a fatal error occurs.

Task Limitations

Other limitations are described in DOCA Core Task.

Wait Not-equal-to Task

The wait-not-equal task allows atomically waiting for a DOCA SE value to not be equal to some threshold.

Task Configuration

<table>
<thead>
<tr>
<th>Description</th>
<th>API to set the configuration</th>
<th>API to query support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable the task</td>
<td><code>doca_sync_event_task_wait_neq_set_conf</code></td>
<td><code>doca_sync_event_cap_task_wait_neq_is_supported</code></td>
</tr>
<tr>
<td>Number of tasks</td>
<td><code>doca_sync_event_task_wait_neq_set_conf</code></td>
<td>-</td>
</tr>
</tbody>
</table>
Task Input

Common input described in DOCA Core Task.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wait threshold</td>
<td>64-bit value to wait for the DOCA SE value to be not equal to</td>
</tr>
<tr>
<td>Mask</td>
<td>64-bit mask to apply on the DOCA SE value before comparing with the wait threshold</td>
</tr>
</tbody>
</table>

Task Output

Common output described in DOCA Core Task.

Task Completion Success

After the task is completed successfully, the following occurs:

- The DOCA SE value is not equal to the given wait threshold.

Task Completion Failure

If the task fails midway, the context may enter a stopping state if a fatal error occurs.

Task Limitations

Limitations are described in DOCA Core Task.
Events

DOCA SE context exposes asynchronous events to notify about changes that happen unexpectedly, according to the DOCA Core architecture.

The only event DOCA SE context exposes is common events as described in DOCA Core Event.

State Machine

The DOCA SE context follows the Context state machine as described in DOCA Core Context State Machine.

The following subsection describe how to move to specific states and what is allowed in each state.

Idle

In this state, it is expected that the application will:

- Destroy the context; or
- Start the context

Allowed operations in this state:

- Configure the context according to section "Configurations"
- Start the context

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>Create the context</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop after making sure all tasks have been freed</td>
</tr>
<tr>
<td>Previous State</td>
<td>Transition Action</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Stopping</td>
<td>Call progress until all tasks are completed and then freed</td>
</tr>
</tbody>
</table>

**Starting**

This state cannot be reached.

**Running**

In this state, it is expected that the application will:

- Allocate and submit tasks
- Call progress to complete tasks and/or receive events

Allowed operations in this state:

- Allocate previously configured task
- Submit an allocated task
- Call stop

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idle</td>
<td>Call start after configuration</td>
</tr>
</tbody>
</table>

**Stopping**

In this state, it is expected that the application will:

- Call progress to complete all inflight tasks (tasks will complete with failure)
- Free any completed tasks

Allowed operations in this state:

- Call progress

It is possible to reach this state as follows:

<table>
<thead>
<tr>
<th>Previous State</th>
<th>Transition Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>Call progress and fatal error occurs</td>
</tr>
<tr>
<td>Running</td>
<td>Call stop without freeing all tasks</td>
</tr>
</tbody>
</table>

**DOCA Sync Event Tear Down**

Multiple SE handles (for different execution units) associated with the same DOCA SE instance can live simultaneously, though the teardown flow is performed only from the CPU on the CPU handle.

**Note**

Users must validate active handles associated with the CPU handle during the teardown flow because DOCA SE does not do that.

**Stopping DOCA Sync Event**

To stop a DOCA SE:

- Synchronous – call `doca_sync_event_stop` on the CPU handle
- Asynchronous – stop the DOCA context associated with the DOCA SE instance

**Note**
Stopping a DOCA SE must be followed by destruction. Refer to section "Destroying DOCA Sync Event" for details.

Destroying DOCA Sync Event

Once stopped, a DOCA SE instance can be destroyed by calling `doca_sync_event_destroy` on the CPU handle.

Remote net CPU handle instance terminates and frees by calling `doca_sync_event_remote_net_destroy` on the remote net CPU handle.

Upon destruction, all the internal resources are released, allocated memory is freed, associated `doca_ctx` (if it exists) is destroyed, and any associated exported handles (other than CPU handles) and their resources are destroyed.

Alternative Datapath Options

DOCA SE supports datapath on CPU (see section "Execution Phase") and also on DPA and GPU.

GPU Datapath

DOCA SE does not currently support GPU related features.

DPA Datapath
Once a DOCA SE DPA handle (doca_dpa_dev_sync_event_t) has been retrieved it can be used within a DOCA DPA kernel as described in DOCA DPA Sync Event.

**DOCA Sync Event Sample**

This section provides DOCA SE sample implementation on top of the BlueField DPU.

The sample demonstrates how to share an SE between the host and the DPU while simultaneously interacting with the event from both the host and DPU sides using different handles.

**Running DOCA Sync Event 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:

   ```bash
   cd /opt/mellanox/doca/samples/doca_common/sync_event_<local|remote>._pci
   meson /tmp/build
   ninja -C /tmp/build
   ```

**Note**
The binary `doca_sync_event_<local|remote>_pci` is created under
/ tmp/build/.

3. Sample usage:

Usage: `doca_sync_event_remote_pci [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, --pci-addr` Device PCI address
- `-r, --rep-pci-addr` DPU representor PCI address
- `--async` Start DOCA Sync Event in asynchronous mode (synchronous mode by default)
- `--async_num_tasks` Async num tasks for asynchronous mode
- `--atomic` Update DOCA Sync Event using Add operation (Set operation by default)

**Note**

The flag `--rep-pci-addr` is relevant only for the DPU.

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

```
/tmp/build/doca_sync_event_<local|remote>_pci -h
```
Samples

Sync Event Remote PCIe

Note

This sample should be run (on the DPU or on the host) before Sync Event Local PCIe.

This sample demonstrates creating an SE from an export which is associated with an SE on a local PCIe (host or the DPU) and interacting with the SE to achieve synchronization between the host and DPU.

The sample logic includes:

1. Reading configuration files and saving their content into local buffers.

2. Locating and opening DOCA devices and DOCA representors (if running on the DPU) matching the given PCIe addresses.

3. Initializing DOCA Comm Channel.

4. Receiving SE blob through Comm Channel.

5. Creating SE from export.

6. Starting the above SE in the requested operation mode (synchronous or asynchronous).

7. Interacting with the SE:

   1. Waiting for signal from the host – synchronously or asynchronously (with busy wait polling) according to user input.
2. Signaling the SE for the host – synchronously or asynchronously, using set or atomic add, according to user input.

8. Cleaning all resources.

Reference:

- /opt/mellanox/doca/samples/doca_common/sync_event_remote_pci/sync_event_remote_pci_sample.c
- /opt/mellanox/doca/samples/doca_common/sync_event_remote_pci/sync_event_remote_pci_main.c
- /opt/mellanox/doca/samples/doca_common/sync_event_remote_pci/meson.build

**Sync Event Local PCIe**

![Note]

This sample should run (on the DPU or on the Host) only after **Sync Event Remote PCIe** has been started.

This sample demonstrates how to initialize a SE to be shared with a remote PCIe (host or the DPU) how to export it to a remote PCIe, and how to interact with the SE to achieve synchronization between the host and DPU.

The sample logic includes:

1. Reading configuration files and saving their content into local buffers.

2. Locating and opening DOCA devices and DOCA representors (if running on the DPU) matching the given PCIe addresses.

3. Creating and configuring the SE to be shared with a remote PCIe.

4. Starting the above SE in the requested operation mode (synchronous or asynchronous).
5. Initializing DOCA Comm Channel.

6. Exporting the SE and sending it through the Comm Channel.

7. Interacting with the SE:
   1. Signaling the SE for the remote PCIe – synchronously or asynchronously, using set or atomic add, according to user input.
   2. Waiting for a signal – synchronously or asynchronously, with busy wait polling, according to user input.

8. Cleaning all resources.

Reference:

- /opt/mellanox/doca/samples/doca_common/sync_event_local_pci/sync_event_local_pci_sample.c
- /opt/mellanox/doca/samples/doca_common/sync_event_local_pci/sync_event_local_pci_main.c
- /opt/mellanox/doca/samples/doca_common/sync_event_local_pci/meson.build