Sync Event

Table of contents

1. Introduction
2. Prerequisites
3. Environment
4. Architecture
5. Configuration Phase
6. Execution Phase
7. State Machine
8. DOCA Sync Event Tear Down
9. Alternative Datapath Options
10. DOCA Sync Event Sample

DOCA Sync Event API is considered thread-unsafe

DOCA Sync Event does not currently support GPU related features.

1. 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

Both publisher and subscriber can read (get) the actual counter's value.

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 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.

2. 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

3. 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.

4. 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.

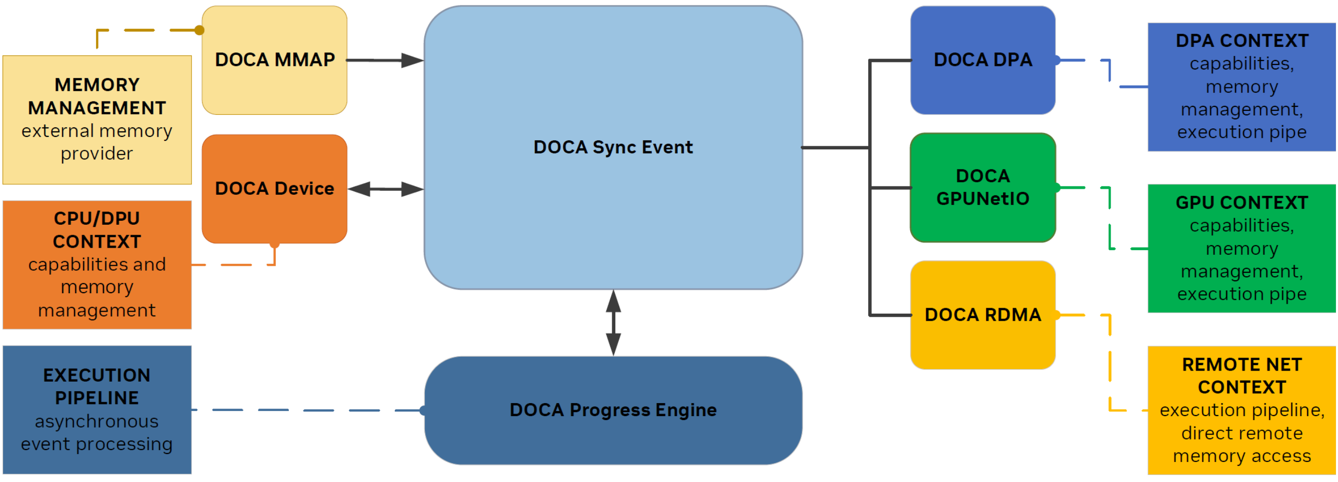
The figure that follows demonstrates components used by DOCA SE. DOCA Device provides information on the capabilities of the configured HW used by SE to control system resources.

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.

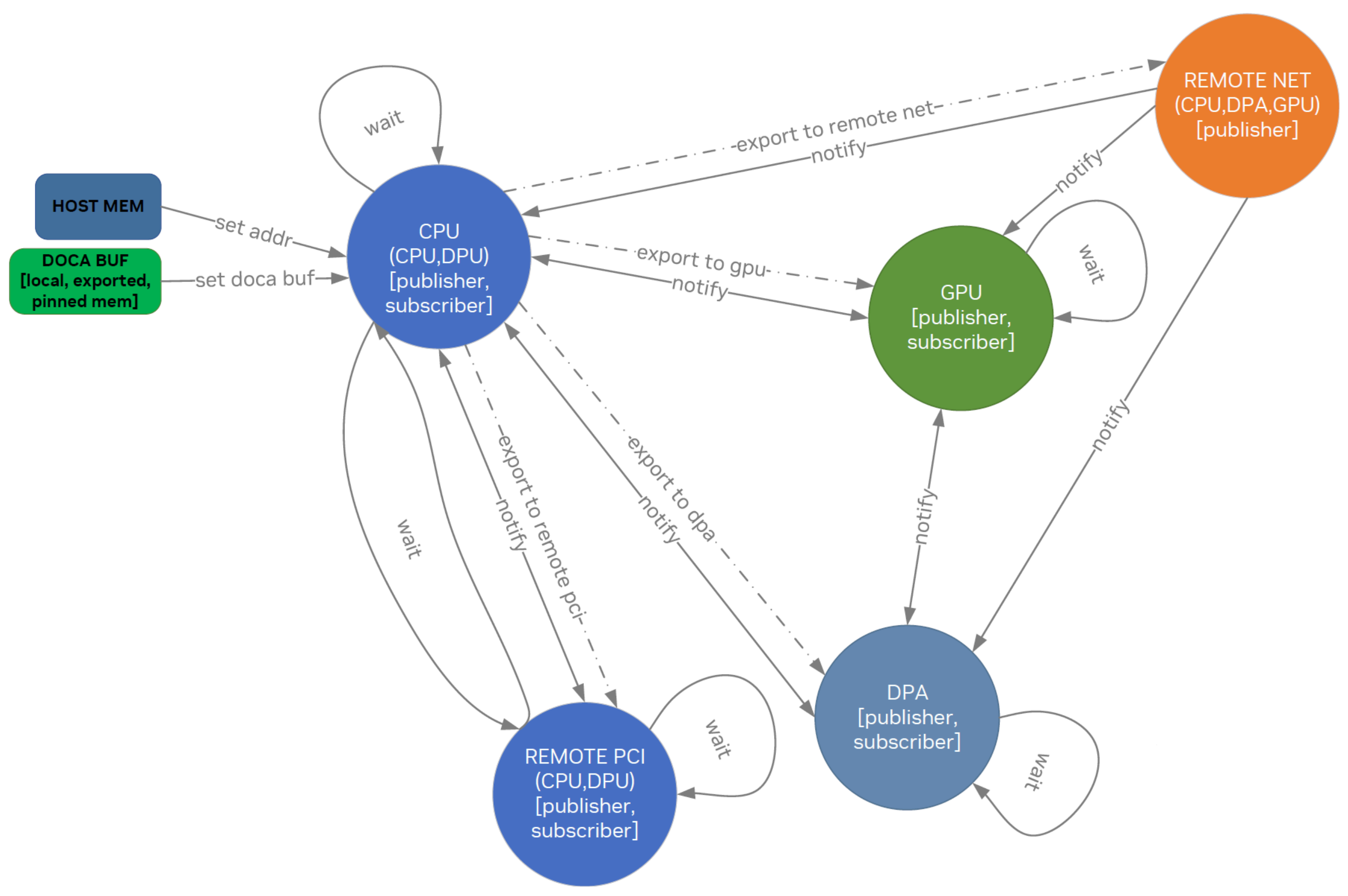
For asynchronous operation scheduling, SE uses the DOCA Progress Engine (PE) module.

DOCA Sync Event Components Diagram



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

DOCA Sync Event Interaction Diagram



4.1 DOCA Sync Event Objects

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

Execution Unit	Type	Description
CPU (host/DPU)	`struct doca_sync_event`	Handle for interacting with the SE from the CPU
DPU	`struct doca_sync_event`	Handle for interacting with the SE from the DPU
DPA	`doca_dpa_dev_sync_event_t`	Handle for interacting with the SE from the DPA
GPU	`doca_gpu_dev_sync_event_t`	Handle for interacting with the SE from the GPU

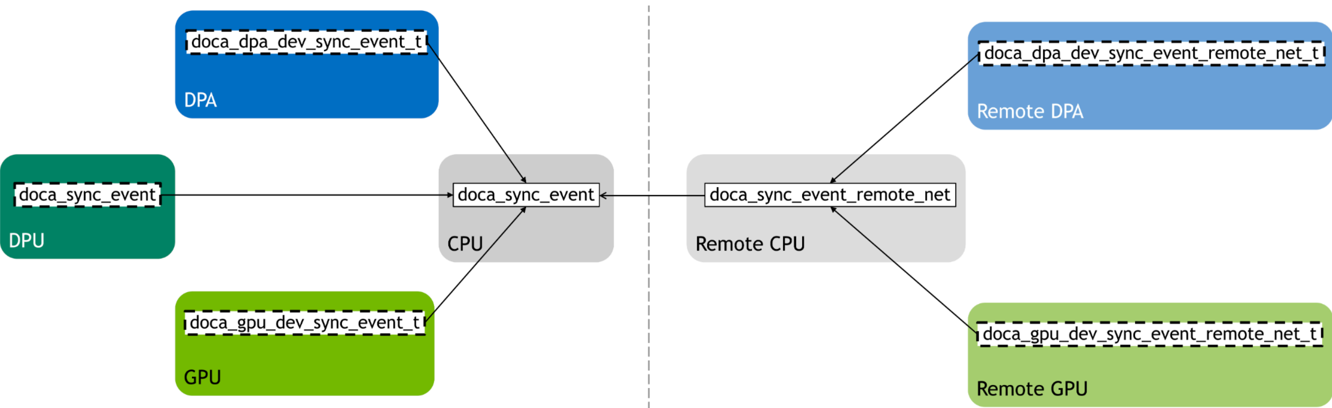
Execution Unit	Type	Description
Remote net CPU	`doca_sync_event_remote_net`	Handle for interacting with the SE from a remote CPU
Remote net DPA	`doca_dpa_dev_sync_event_remote_net_t`	Handle for interacting with the SE from a remote DPA
Remote net GPU	`doca_gpu_dev_sync_event_remote_net_t`	Handle for interacting with the SE from a remote GPU

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

5. 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:



5.1 Operation Modes

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

> **Note**
>
> Once started, SE operation mode cannot be changed.

5.1.1 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.

5.1.2 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).

5.2 Configurations

5.2.1 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_pc`
  - `doca_sync_event_add_subscriber_location_<cpu|dpa|gpu|remote_p`
- For the asynchronous use case, at least one task/event type must be configured. See configuration of tasks.

5.2.2 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`

5.3 Export DOCA Sync Event to Another Execution Unit

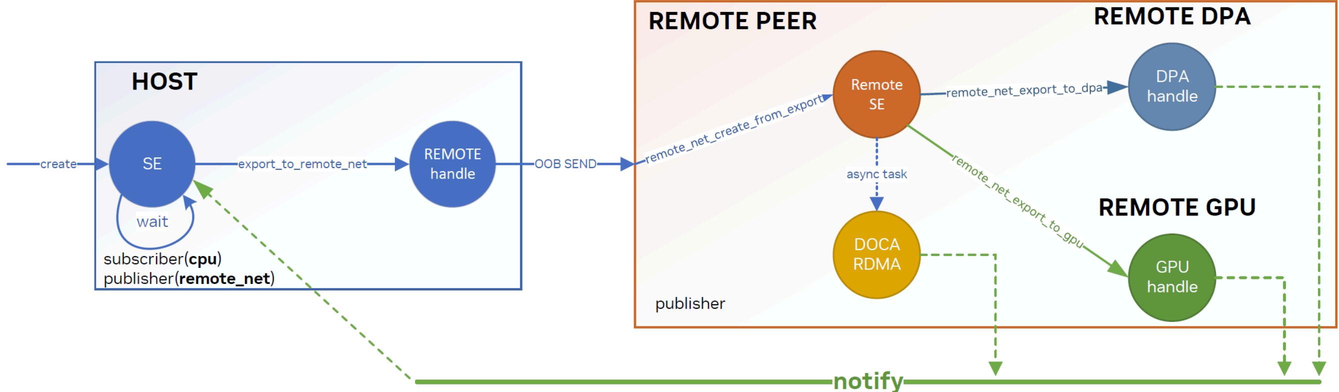
To use an SE from an execution unit other than the CPU, it must be exported to get a handle for the specific execution unit:

- 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

Data exported in `doca_sync_event_export_to_*` functions contains sensitive information. Make sure to pass this data through a secure channel!

5.4 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`

6. Execution Phase

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

6.1 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).

Info

Remote net CPU handle (`struct doca_sync_event_remote_net`) can be used for submitting asynchronous tasks using the DOCA RDMA library.

Note

Prior to asynchronous task submission, users must check if the job is supported using `doca_error_t doca_sync_event_cap_task_<task_type>_is_supported`.

The following subsections describe the DOCA SE datapath operation with respect to synchronous and asynchronous operation modes.

6.1.1 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

6.1.2 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

Users may leverage the `doca_sync_event_task_get` job to implement asynchronous wait by asynchronously submitting the task on a DOCA PE and comparing the result to some threshold.

6.2 Tasks

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

6.2.1 Get Task

The get task retrieves the value of a DOCA SE.

Task Configuration

Description	API to Set the Configuration	API to Query Support
Enable the task	`doca_sync_event_task_get_set_conf`	`doca_sync_event_cap_tas`
Number of tasks	`doca_sync_event_task_get_set_conf`	-

Task Input

Common input described in DOCA Core Task.

Name	Description
Return value	8-bytes memory pointer to hold the DOCA SE value

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.

6.2.2 Notify Set Task

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

Task Configuration

Description	API to Set the Configuration	API to Query Support
Enable the task	`doca_sync_event_task_notify_set_set_conf`	`doca_sync_event_`
Number of tasks	`doca_sync_event_task_notify_set_set_conf`	-

Task Input

Common input described in DOCA Core Task.

Name	Description
Set value	64-bit value to set the DOCA SE value to

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.

6.2.3 Notify Add Task

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

Task Configuration

Description	API to Set the Configuration	API to Query Support
Enable the task	`doca_sync_event_task_notify_add_set_conf`	`doca_sync_event_`
Number of tasks	`doca_sync_event_task_notify_add_set_conf`	-

Task Input

Common input described in DOCA Core Task.

Name	Description
Increment value	64-bit value to atomically increment the DOCA SE value by
Fetched value	8-bytes memory pointer to hold the DOCA SE value before the increment

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.

6.2.4 Wait Equal-to Task

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

Task Configuration

Description	API to set the configuration	API to query support
Enable the task	`doca_sync_event_task_wait_eq_set_conf`	`doca_sync_event_cap`
Number of tasks	`doca_sync_event_task_wait_eq_set_conf`	-

Task Input

Common input described in DOCA Core Task.

Name	Description
Wait threshold	64-bit value to wait for the DOCA SE value to be equal to
Mask	64-bit mask to apply on the DOCA SE value before comparing with the wait threshold

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.

6.2.5 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

Description	API to set the configuration	API to query support
Enable the task	`doca_sync_event_task_wait_neq_set_conf`	`doca_sync_event_ca`
Number of tasks	`doca_sync_event_task_wait_neq_set_conf`	-

Task Input

Common input described in DOCA Core Task.

Name	Description
Wait threshold	64-bit value to wait for the DOCA SE value to be not equal to
Mask	64-bit mask to apply on the DOCA SE value before comparing with the wait threshold

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.

6.3 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.

7. 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.

7.1 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:

Previous State	Transition Action
None	Create the context
Running	Call stop after making sure all tasks have been freed
Stopping	Call progress until all tasks are completed and then freed

7.2 Starting

This state cannot be reached.

7.3 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:

Previous State	Transition Action
Idle	Call start after configuration

7.4 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:

Previous State	Transition Action
Running	Call progress and fatal error occurs
Running	Call stop without freeing all tasks

8. 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.

8.1 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.

8.2 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.

9. Alternative Datapath Options

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

9.1 GPU Datapath

DOCA SE does not currently support GPU related features.

9.2 DPA Datapath

Info

An SE with DPA-subscriber configuration currently supports synchronous APIs only.

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.

10. 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.

10.1 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:

```
cd
/opt/mellanox/doca/samples/doca_common/sync_event_<local|remot
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
```

10.2 Samples

10.2.1 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`
- `/opt/mellanox/doca/samples/doca_common/sync_event_remote_pci/sync`
- `/opt/mellanox/doca/samples/doca_common/sync_event_remote_pci/mesc`

10.2.2 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_`
- `/opt/mellanox/doca/samples/doca_common/sync_event_local_pci/sync_`
- `/opt/mellanox/doca/samples/doca_common/sync_event_local_pci/meson`

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 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 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