DOCA SHA

Table of contents

This guide provides instructions on building and developing applications that calculate message digest using the SHA1, SHA2-256, or SHA2-512 algorithms.

Introduction

> **Note**
>
> The DOCA SHA library is currently supported at alpha level.

The library provides an API for executing SHA operations on DOCA buffers, where the buffers reside in either local memory (i.e., within the same host) or host memory accessible by the NVIDIA® BlueField®-2 device (remote memory). Using DOCA SHA, complex cryptographic hash operations can be easily executed in an optimized, hardware-accelerated manner.

> **Note**
>
> NVIDIA® BlueField®-3 does not support this library because it has no SHA acceleration engine.

This document is intended for software developers wishing to accelerate their applications' SHA calculations typically used in digital signature schemes or hash-based message authentication code calculations.

Prerequisites

This library follows the architecture of a DOCA Core context, it is recommended to read the following sections before:

- DOCA Core Execution Model
- DOCA Core Device
- DOCA Core Memory Subsystem

Environment

DOCA SHA-based applications can run either on the host machine or on the BlueField-2 DPU target.

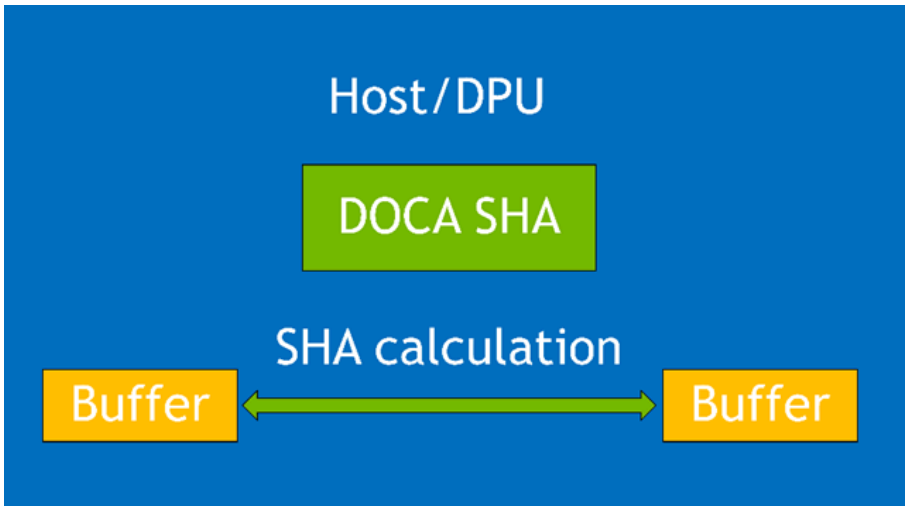
DOCA SHA calculations from the host to BlueField and vice versa can only be run when the DPU is configured in DPU mode.

Architecture

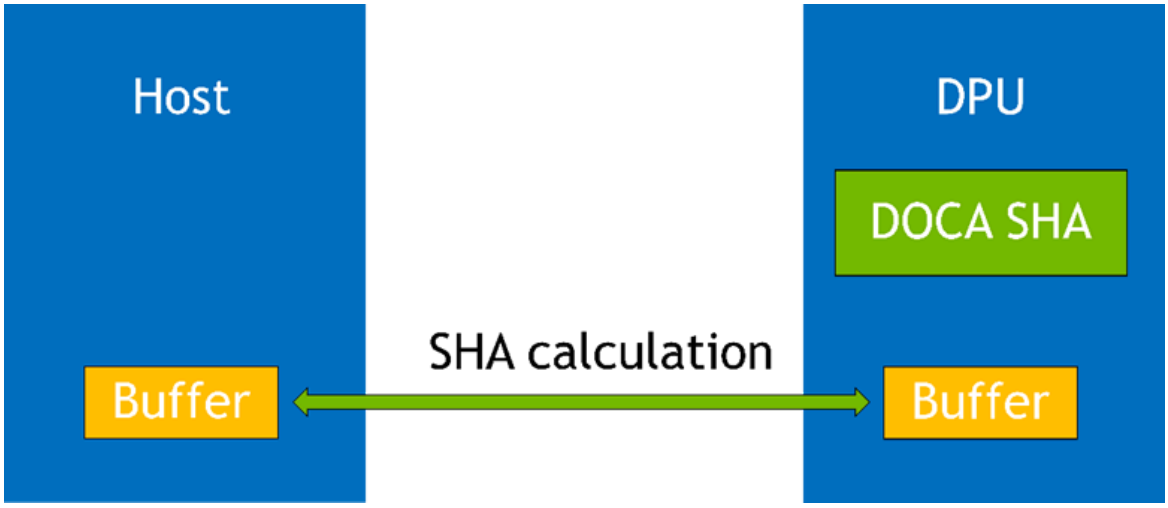
DOCA SHA is a DOCA Core Context. This library leverages the DOCA Core architecture to expose asynchronous tasks/events offloaded to hardware.

SHA can be used to calculate message digest as illustrated in the following diagrams:

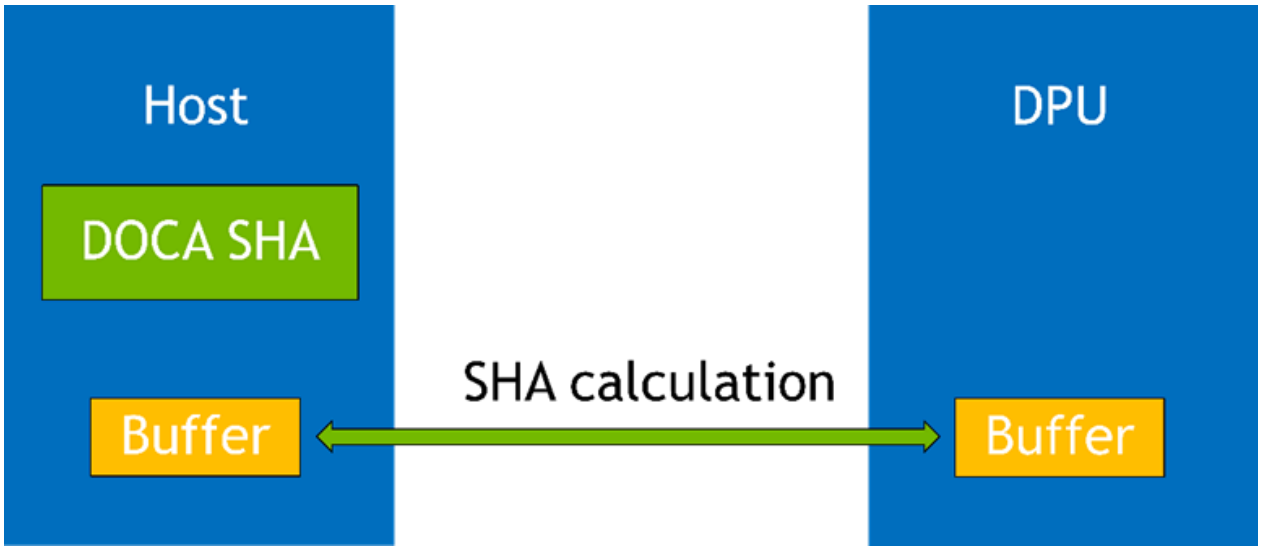
- SHA from local memory to local memory:



- Using the DPU to do SHA using the memory between the host and the DPU:



- Using the host to do SHA calculation using memory between the host and the DPU:



Objects

Device and Representor

The library requires a DOCA device to operate. The device is used to access memory and perform the actual SHA calculation. See DOCA Core Device Discovery.

For the same BlueField DPU, it does not matter which device is used (i.e., PF/VF/SF) as these devices utilize the same hardware component. If there are multiple DPUs, then it is possible to create a SHA instance per DPU, providing each instance with a device from a different DPU.

To access non-local memory (i.e., from the host to DPU or vice versa), the DPU side of the application must choose a device with an appropriate representor (see DOCA Core Device

Representor Discovery). The device must stay valid for as long as the SHA instance is not destroyed.

Memory Buffers

The SHA task requires at least two DOCA buffers containing the destination and the source.

The destination is always a single `doca_buf`. The source can be a single `doca_buf` or a `linked_list` of `doca_buf`. All destination and source `doca_buf`s can be allocated from `doca_buf_inventory`.

> **Info**
>
> For the usage of `doca_buf_inventory`, please refer to the DOCA Core Inventory Types table.

Buffers must not be modified or read during the SHA operation. For information on what kind of memory is supported, refer to the table in section "Buffer Support".

Configuration Phase

To start using the library, users must go through a configuration phase as described in DOCA Core Context Configuration Phase.

This section describes how to configure and start the context to allow the execution of tasks and retrieval of events.

Configurations

The context can be configured to match the application use case.

To find if a configuration is supported or its min/max value, refer to section "Device Support".

Mandatory Configurations

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

- At least one task/event type must be configured. See configuration of tasks and/or events in sections "Tasks" and "Events" respectively for information.
- A device with appropriate support must be provided upon creation

Device Support

DOCA SHA requires a device to operate. For information on choosing a device, see DOCA Core Device Discovery.

As device capabilities may change in the future (see DOCA Core Device Support) it is recommended to select your device using the following methods:

- `doca_sha_cap_task_hash_get_supported`
- `doca_sha_cap_task_partial_hash_get_supported`

Some devices can allow different capabilities such as:

- The maximum number of tasks
- The maximum source buffer size
- The minimum destination buffer size
- The maximum supported number of elements in DOCA linked-list buffer
- Check whether SHA1, SHA2-256 or SHA2-512 is supported

Buffer Support

Tasks support buffers with the following features:

Buffer Type	Source Buffer	Destination Buffer
Local mmap buffer	Yes	Yes
Mmap from PCIe export buffer	Yes	Yes
Mmap from RDMA export buffer	No	No
Linked list buffer	Yes	No

Execution Phase

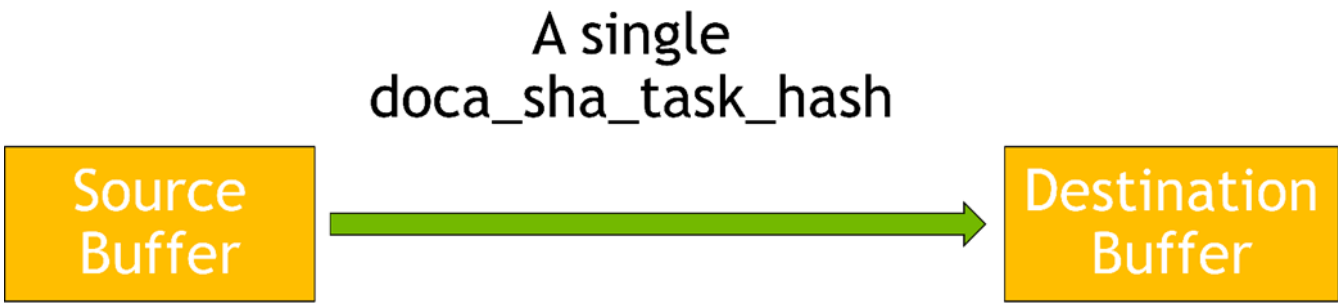
This section describes execution on the CPU using DOCA Core Progress Engine.

Tasks

DOCA SHA exposes asynchronous tasks that leverage DPU hardware according to DOCA Core architecture.

SHA Task

The SHA task `doca_sha_task_hash` allows one-shot SHA calculation using buffers as described in section "Buffer Support". One-shot means that the source buffer is used as a whole input, therefore, the SHA operation is completed after this task completion event arrives.



Task Configuration

Description	API to Set Configuration	API to Query Support
Enable the task	`doca_sha_task_hash_set_conf`	`doca_sha_cap_task_hash_get_supported`

Description	API to Set Configuration	API to Query Support
Number of tasks	`doca_sha_task_hash_set_conf`	`doca_sha_cap_get_max_num_tasks`
Maximal source buffer size	–	`doca_sha_cap_get_max_src_buf_size`
Maximum source buffer list size	–	`doca_sha_cap_get_max_list_buf_num_elem`
Minimum destination buffer size	–	`doca_sha_cap_get_min_dst_buf_size`

Task Input

Common input as described in DOCA Core Task.

Name	Description	Notes
Source buffer	Buffer pointing to the memory to be used for SHA calculation	Only the data residing in the data segment is to be used
Destination buffer	Buffer pointing to the memory used for writing the SHA calculation result	The SHA result is appended to the tail segment
SHA algorithm type	SHA algorithm to be used in SHA calculation	Must be one of `DOCA_SHA_ALGORITHM_SHA1`, `DOCA_SHA_ALGORITHM_SHA256`, `DOCA_SHA_ALGORITHM_SHA512`

Task Output

Common output as described in DOCA Core Task.

Task Completion Success

After the task completes successfully, the following happens:

- The SHA calculation of data from the source buffer is successfully completed and the result is written to the destination buffer
- The destination buffer data segment is extended to include the SHA result data

Task Completion Failure

If the task fails midway:

- The context may enter stopping state if a fatal error occurs
- The source and destination `doca_buf` objects are not modified
- The destination buffer contents may be modified

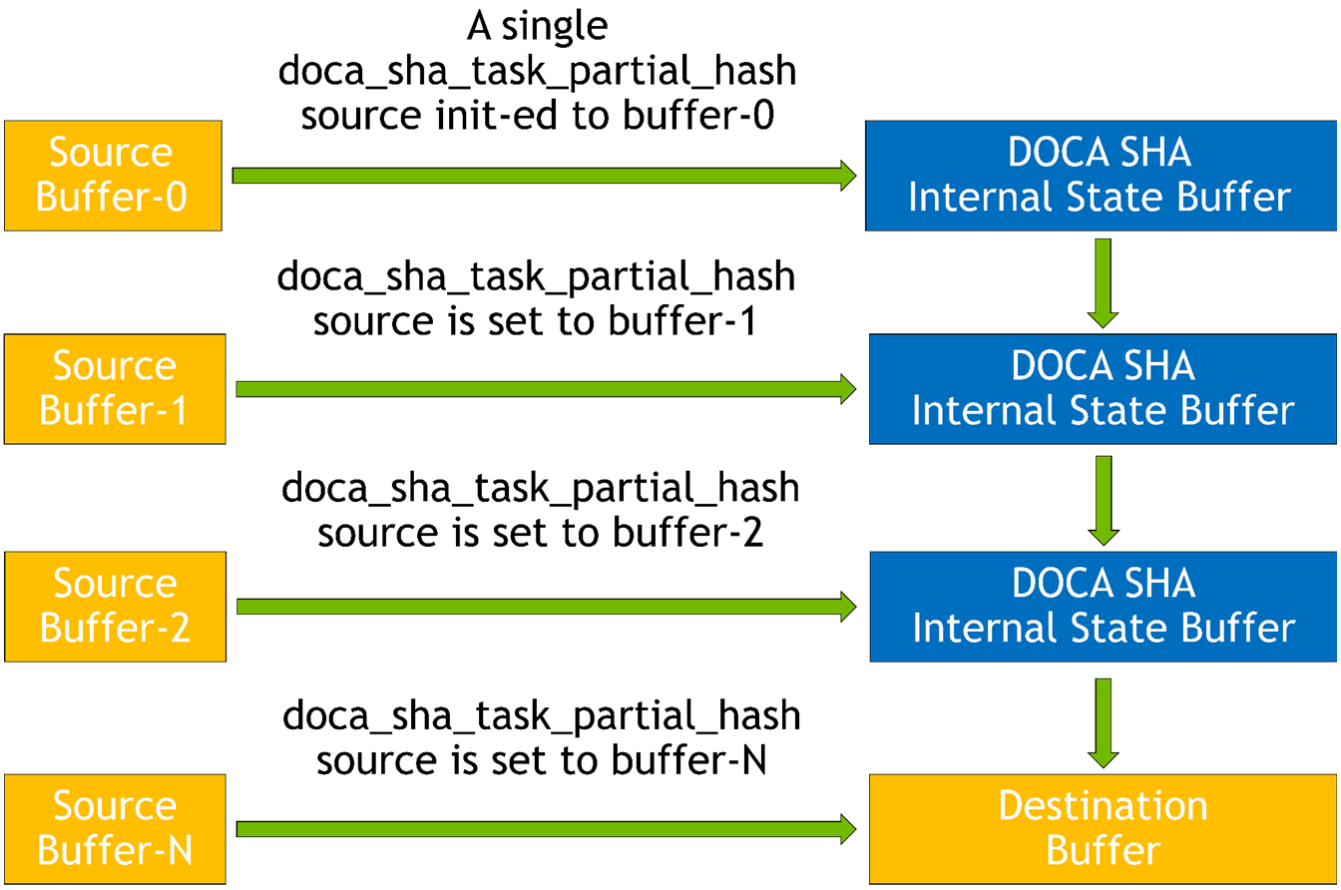
Task Limitations

- The operation is not atomic
- Once the task is submitted, the source and destination should not be read/written to
- Other limitations are described in DOCA Core Task

Partial-SHA Task

The partial-SHA task `doca_sha_task_partial_hash` allows stateful SHA calculation for a collection of messages. Using buffers as described in section "Buffer Support".

Stateful means that the input data is composed of many segments (may be spatial or timely non-consecutive), therefore, its SHA calculation requires more than one one-shot SHA operation to finish. During any stateful operation, other independent SHA tasks can also be executed.



Task Configuration

Description	API to Set Configuration	API to Query Support
Enable the task	`doca_sha_task_partial_hash_set_conf`	`doca_sha_cap_task_partial_hash_get_supported`
Number of tasks	`doca_sha_task_partial_hash_set_conf`	`doca_sha_cap_get_max_num_tasks`
Maximal source buffer size	–	`doca_sha_cap_get_max_src_buf_size`
Maximum source buffer list size	–	`doca_sha_cap_get_max_list_buf_num_elem`
Minimum destination buffer size	–	`doca_sha_cap_get_min_dst_buf_size`
SHA block size		`doca_sha_cap_get_partial_hash_block_size`

Task Input

Common input as described in DOCA Core Task.

Name	Description	Notes
Source buffer	Buffer pointing to the memory to be used for SHA calculation	Only the data residing in the data segment is to be used. And the data length for the non-last data segment must be multiple of the SHA block size queried by `doca_sha_cap_get_partial_hash_block_size`
Destination buffer	Buffer pointing to the memory is used for writing the SHA calculation result	The SHA result is appended to the tail segment. During the whole calculation process, this buffer cannot be modified.
SHA algorithm type	SHA algorithm to be used in SHA calculation	Must be one of `DOCA_SHA_ALGORITHM_SHA1`, `DOCA_SHA_ALGORITHM_SHA256`, `DOCA_SHA_ALGORITHM_SHA512`
Whether the current source buffer is the last segment	Indicate whether the current source Buffer is the last segment data to be used for partial-SHA calculation	Use `doca_sha_task_partial_hash_set_is_final_buf` to set this property
Set source buffer	Use to set the subsequent source segment buffer after the initial `doca_sha_task_partial_hash` task is allocated	`doca_sha_task_partial_hash_set_src`

Task Output

Common output as described in DOCA Core Task.

Task Completion Success

After the task completes successfully, the following happens:

- The SHA calculation of data from the source buffer is successfully completed and the result is written to the destination buffer
- The destination buffer data segment is extended to include the SHA result data

Task Completion Failure

If the task fails midway:

- The context may enter stopping state if a fatal error occurs
- The source and destination `doca_buf` objects is not modified
- The destination buffer contents may be modified

Task Limitations

- The operation is not atomic
- Once the task is submitted, the source and destination should not be read/written to
- Other limitations are described in DOCA Core Task

Events

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

The only events SHA exposes are common events as described in DOCA Core Event.

State Machine

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

The following section describes moving states and what is allowed in each state.

Idle

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

- Destroys the context
- Starts the context

Allowed operations:

- Configuring the context according to section "Configurations"
- Starting 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 freed

Starting

This state cannot be reached.

Running

In this state, it is expected that the application:

- Allocates and submits tasks
- Calls progress to complete tasks and/or receive events

Allowed operations:

- Allocating previously configured task
- Submitting a task
- Calling stop

It is possible to reach this state as follows:

Previous State	Transition Action
Idle	Call start after configuration

Stopping

In this state, it is expected that the application:

- Calls progress to complete all inflight tasks (tasks complete with failure)
- Frees any completed tasks

Allowed operations:

- Calling 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

Alternative Datapath Options

DOCA SHA only supports datapath on the CPU. See section "Execution Phase".

DOCA SHA Samples

This section describes DOCA SHA samples based on the DOCA SHA library.

The samples in this section illustrate how to use the DOCA SHA API to do the following:

- Do SHA calculation of contents of a buffer, and write result to another buffer
- Chop the contents of a buffer into a collection of segments, and do partial-SHA calculation of this collection of segments, and write result to another

> **Info**
>
> All the DOCA samples described in this section are governed under the BSD-3 software license agreement.

Running the Samples

1. Refer to the following documents:
   - DOCA Installation Guide for Linux for details on how to install BlueField-related software.
   - DOCA Troubleshooting 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_sha/<sample_name>
meson/tmp/build
ninja -C/tmp/build
```

Info

The binary `doca_<sample_name>` is created under `/tmp/build/`.

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

```
Usage: doca_sha_create [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, --data                        user data
```

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

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

Samples

SHA Create

This sample illustrates how to perform SHA calculation with DOCA SHA.

The sample logic includes:

1. Locating DOCA device.
2. Initializing required DOCA Core structures.
3. Setting the `task_pool` configuration for `doca_sha_task_hash`.
4. Populating DOCA memory map with two relevant buffers.
5. Allocating element in DOCA buffer inventory for each buffer.
6. Allocating and initializing a `doca_sha_task_hash`.
7. Submitting the task.
8. Retrieving task result once it is done.

Reference:

- `/opt/mellanox/doca/samples/doca_sha/sha_create/sha_create_sample.`
- `/opt/mellanox/doca/samples/doca_sha/sha_create/sha_create_main.c`
- `/opt/mellanox/doca/samples/doca_sha/sha_create/meson.build`

SHA-Partial Create

This sample illustrates how to perform partial-SHA calculation for a collection of data segments with DOCA SHA.

The sample logic includes:

1. Locating DOCA device.

2. Initializing the required DOCA Core structures.

3. Setting the `task_pool` configuration for `doca_sha_task_partial_hash`.

4. Chopping the source data into a collection of data segments according to the selected SHA algorithm's block size

5. Populating DOCA memory map with needed buffers for all source data segments and destination buffer.

6. Allocating element in DOCA buffer inventory for the first source buffer and destination buffer.

7. Allocating and initializing a `doca_sha_task_partial_hash` with the first source buffer and the destination buffer.

8. Iteratively repeating the following sub-steps until all data segments are consumed:

    1. Submitting the `doca_sha_task_partial_hash`.

    2. Waiting for the submitted task to finish.

    3. Allocating a `doca_buf` for the next source segment and use `doca_sha_task_partial_hash_set_src` to set it as source buffer of the above allocated task.

    4. If it is the final segment, use `doca_sha_task_partial_hash_set_is_final_buf` to mark it in the allocate task.

9. Retrieving the result of the final iteration in the destination buffer as the full partial-SHA calculation result.

10. Destroying all SHA and DOCA Core structures.

Reference:

- `/opt/mellanox/doca/samples/doca_sha/sha_partial_create/sha_partia`
- `/opt/mellanox/doca/samples/doca_sha/sha_partial_create/sha_partia`
- `/opt/mellanox/doca/samples/doca_sha/sha_partial_create/meson.buil`

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.
