HAILO

Empowering Intelligence

HailoRT User Guide

Release 4.14.0

28 June 2023

Table of Contents

Disclaimer and Proprietary Information Notice

Copyright

This version of the document supersedes all previous versions.

General Notice

Hailo, to the fullest extent permitted by law, provides this document "as-is" and disclaims all warranties, either express or implied, statutory or otherwise, including but not limited to the implied warranties of merchantability, non-infringement of third parties' rights, and fitness for particular purpose.

Although Hailo used reasonable efforts to ensure the accuracy of the content of this document, it is possible that this document may contain technical inaccuracies or other errors. Hailo assumes no liability for any error in this document, and for damages, whether direct, indirect, incidental, consequential or otherwise, that may result from such error, including, but not limited to loss of data or profits.

The content in this document is subject to change without prior notice and Hailo reserves the right to make changes to content of this document without providing a notification to its users.

Part I

User Guide

1. HailoRT Overview

Hailo SW products are set of frameworks and tools that enable you to compile, run and evaluate neural networks on Hailo devices. HailoRT is part of Hailo's **runtime environment**:

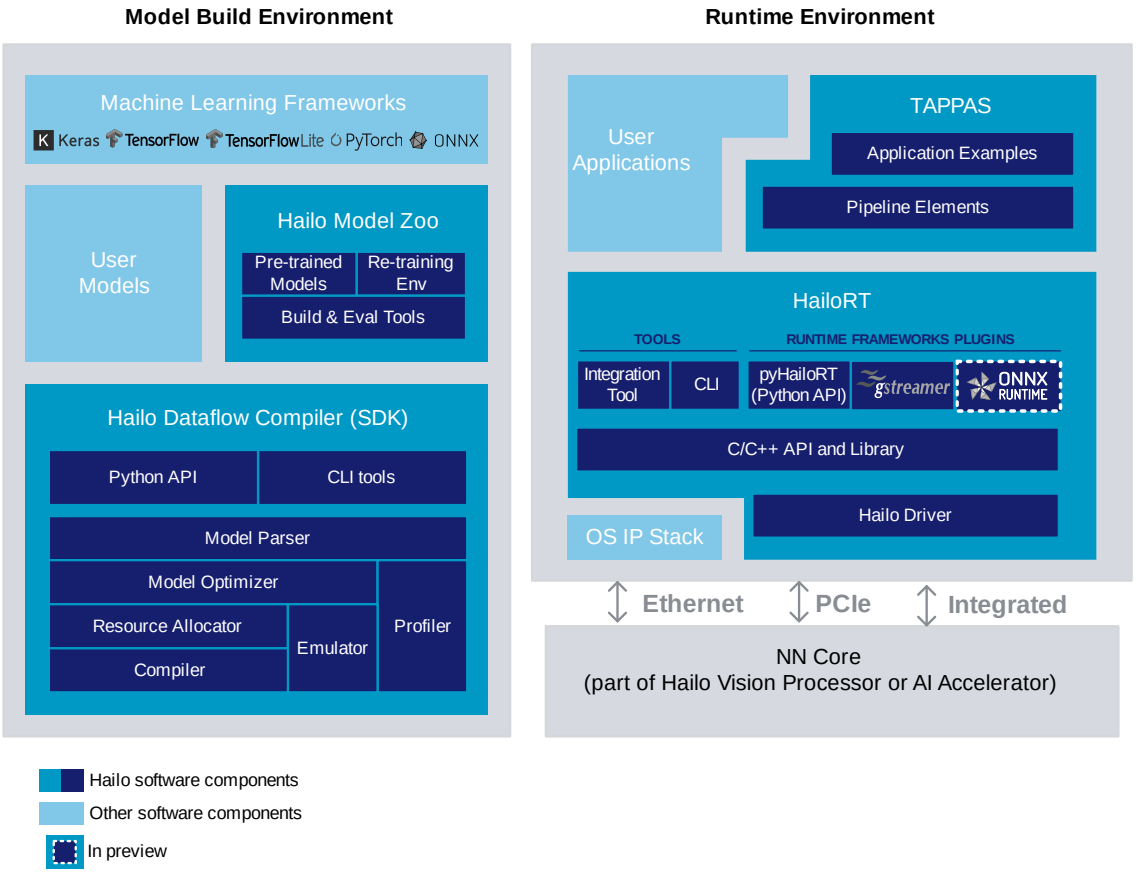

Figure 1. Detailed block diagram of Hailo software packages

Note: HailoRT supports all Hailo devices (including both Hailo-8 and Hailo-15).

1.1. Included in this Package

This software package includes the following parts:

- HailoRT library to run inference from C/C++ programs
- HailoRT Python package
- HailoRT CLI
- PCIe driver
- Yocto layer
- Hailo-8 Firmware
- Hailo GStreamer Plugin

- Additional files such as an installation script and a configuration file

1.2. HailoRT Library (C/C++ API)

The HailoRT library is a user-space run time library responsible for operating the Hailo device. It allows both to control the Hailo device and to transfer data to and from it. The HailoRT library implements robust and intuitive API's in C/C++ for optimized performance using the Hailo device.

1.3. HailoRT Python API

The HailoRT Python package wraps the library and exposes a Python interface that allows the loading of models to the device and to send and receive data from it.

1.4. HailoRT CLI

The command Line application used to control the Hailo device, run inference using the device, collect inference statistics and device events, etc.

1.5. PCIe Driver

The Hailo's PCIe driver is required when working via the PCIe interface, it links the HailoRT library and the device. It also loads the device's firmware when working through this interface. The device driver is used to manage the Hailo device, communicate with the device and transfer data to and from the device.

1.6. Yocto Layer

Hailo's Yocto layer allows the user to integrate Hailo's software into an existing Yocto environment. It includes recipes for the HailoRT library, Python package and the PCIe driver.

1.7. Hailo-8 Firmware

The firmware that runs on the Hailo device. It manages the boot and the control of the Hailo device.

1.8. Hailo GStreamer Plugin

The Hailo's GStreamer plugin provides the "hailonet" element which can infer GStreamer frames according to the configured network. This element can be used multiple times in a GStreamer pipeline to infer multiple networks in parallel.

2. Changelog

HailoRT v4.14.0 (July 2023)

Hailo-15

- This version is supported on Hailo-15H

HailoRT post-processing support (release)

- Added Yolox support
- Added Argmax support (preview), for single-output models

Model Scheduler

- Model `priority` is now supported (release), see *Model Scheduler*. For usage reference, see *C* and *C++*) examples
- `batch_size` parameter now reflects the number of frames (specifically maximum number of frames) sent to a network group when it is allocated by the scheduler

CLI

- `hailortcli run2` is now released
- New `hailortcli run2` functionality:
  - `--measure-latency` will measure NN core latency. Frames will be sent one at a time and FPS is not measured
  - `--measure-overall-latency` will measure the overall network latency, including pre/post inference, scheduler switches etc. Frames will be sent one at a time and FPS is not measured
  - On both latency measurement options – only 1 model is allowed

Examples

- Added support for Hailo-15H (relevant examples only, unsupported examples are mentioned explicitly)
- Changed the vStream examples (*C* and *C++*) to de-quantize the output data, and to use a non-default input format order (NCHW)
- Changed the multi-device examples (*C*, *C++*) and the switch-network-group examples (*C*, *C++*) to use non-default batch size and power_mode

Windows

- *Multi-Process service* can now work with PyHailoRT (preview), which enables using the Python streaming API (send/recv)

API

- **Async API** - Added the C/C++ Async API for raw streams (preview). See examples *C*, *C++ single thread*, *C++ multi-thread*
- Added the C-API function `hailo_demux_by_name_raw_frame_by_output_demuxer()` to improve usability of de-multiplexing streams over C API
- Added the C-API functions `hailo_init_configure_params_by_vdevice()` and `hailo_init_configure_params_by_device()` to improve usability. Changed relevant C examples to use these functions
- Added the C++ overload for `hailort::InputStream::write()` and `hailort::OutputStream::read()`, which work with raw pointers and sizes
- Python API (PyHailoRT) - `RateLimiterWrapper` now needs the target device ip (`remote_ip`) in its initialization

Note: PyHailoRT switched to import pybind11 using CMake's FetchContent. If encountering PyHailoRT compilation errors, it is recommended to delete the directory `hailort/pre_build/external/build/`.

HailoRT v4.13.0 (April 2023)

HailoRT post-processing support (preview)

- HailoRT now supports running SSD post-processing on the host CPU

Model Scheduler

- **Multi-Device** Model Scheduler is now enabled by default for C/C++ API
- Added support for model `priority` see `hailo_set_scheduler_priority()` (preview)

Note: Model Scheduler will be default for PyHailoRT in future versions, for Linux and Windows.

CLI

- Added various `hailortcli run2` functionality – including support for Ethernet inference, multiple-device inference, input data file and latency, power, current and temperature measurements. See `hailortcli run2 --help` for more information
- *Monitor* is now released:
  - Supports multiple-devices
  - Changed Active Time (%) to Device Utilization (%) metric

VDevice

- Added VDevice support for one Ethernet device (useful for a unified inference application using both PCIe and Ethernet devices)

Examples

- Added examples for using notification callbacks (*C*, *C++*)

Windows

- Windows 11 is now supported
- *Multi-Process service* is now supported for C/C++ API (preview)

API

- Python API
  - Model Scheduler and Multi-Process service are now supported (Linux only)

Note: Support for HEFs with multiple-network-groups will be deprecated from future versions.

HailoRT v4.12.1 (February 2023)

- Fixed a bug which might cause a race condition in multi context HEFs with NMS output
- Fixed a bug which sometimes caused a timeout when deactivating a network while running inference
- Fixed a bug in HailoNet which caused a memory leak
- Fixed a bug in `hailortcli` which caused some default flags to be ignored
- Fixed a bug which prevented VStream recovery-after-timeout, when NMS runs in the NN core

HailoRT v4.12.0 (January 2023)

Model Scheduler

- *Model Scheduler* is enabled by default for the C/C++ API, `hailortcli` and *HailoNet*. Working without the scheduler is still fully supported – to disable the Model Scheduler, See *user controlled switching example* (for C++) which demonstrates how to manually switch models

Note: To port applications that used previous versions of HailoRT to this version, either turn the scheduler off (see above), or change them to work with the scheduler by avoiding the calls to `hailo_activate_network_group()` and `hailo_deactivate_network_group()` (C) or `hailort::ConfiguredNetworkGroup::activate()` (C++) .

- Added support for **multi-device** Model Scheduler (preview) - VDevice with multiple physical devices and Model Scheduler enabled. See updated examples *C*, *C++*. To enable this feature - set `HAILO_ENABLE_MULTI_DEVICE_SCHEDULER` environment variable to `1`
- Updated switch_network_groups_examples (*C*, *C++*) to use Model Scheduler *parameters* `threshold` and `timeout`
- *Stream Multiplexer* is now released

Multi-Process service

- *Multi-Process service* is now released (C/C++ API), which enables multi-process inference:
  - Over a single device per `hailo_vdevice_params_t::group_id` – release grade
  - Via VDevice with more than 1 device – preview

HailoRT post-processing support (preview)

- HailoRT now supports running Yolov5 post-processing on the host CPU, under the HailoRT API

Note: Use the Dataflow Compiler to enable the host post-processing feature for a specific HEF.

Supported Environments

- Ubuntu 22.04 is now supported (in release grade)

Yocto

- Added support for *offline builds* (preview)

Firmware

- **Ethernet** and **PCIe** are now supported via two separate types of firmware
- The Hailo-8 PCIe firmware is an optimization feature that increased the number of supported contexts/Network Groups that can be configured

Logger

- A copy of the HailoRT log will be written to:

  - Linux - `/home/<user>/.hailo/hailort/`
  - Windows - `C:\Users\<user>\AppData\Local\HailoRT\`
- To disable both local log and its copy, set the `HAILORT_LOGGER_PATH` environment variable to 'NONE'

CLI

- Introducing a new `run2` command (preview) for running multiple HEFs simultaneously (using the Model Scheduler)

  See `hailortcli run2 --help` for more details
- Removed deprecated parameter `--device-type` for all commands that use Hailo devices

API

- Renamed `HAILO_STREAM_ABORTED` to `HAILO_STREAM_ABORTED_BY_HW`
- Renamed `HAILO_STREAM_INTERNAL_ABORT` to `HAILO_STREAM_ABORTED_BY_USER`
- Removed the option to create a `hailort::VDevice` from `hailo_pcie_device_info_t`, use `hailo_device_id_t` instead
- Removed the option to get physical_device_infos from `hailort::VDevice`

HailoRT v4.11.0 (November 2022)

Model Scheduler

- *Model Scheduler* is enabled by default (C/C++). See *user controlled switching example* which demonstrates how to manually switch models
- Default batch size is now 0 (was previously 1). When the batch size is 0, HailoRT will apply an automatic heuristic to choose the batch size
- Default threshold value is now 0 (was previously 1). When the threshold is 0, HailoRT will apply an automatic heuristic to choose the threshold

---

Note: When using Model Scheduler with batch size larger than 1, some recovery flows (from over-current or over-temperature events) may cause the process to get stuck. This issue will be solved in the next version.

Multi-Process service

- *Multi-Process service* is now supported in HailoNet GStreamer element (preview)

API

- Added the functions `hailo_init_configure_network_group_params()` and `hailo_init_configure_network_group_params_mipi_input()` to allow the creation of a single configured_network_group parameters in a multiple-network-groups HEF
- Changed functionality of the HEF configuration:
  - In case a HEF contains multiple network groups, only those that are present in the `configure_params` will be loaded to the device
  - In case no `configure_params` are passed, all network groups in the HEF will be loaded to the device

CLI

- Deprecated commands: `hailo udp`, `hailo udp-limiter`, use `hailo udp-rate-limiter` instead

Examples

- Changed the directory structure of HailoRT examples – now each example has its own CMakeList file, supporting copying a single example file and building it (standalone)
- Changed the C example name from "switch_single_io_network_groups_manually_example" to "switch_network_groups_manually_example"
- Removed hard-coded compiler flags (-O3 -DNDEBUG)

HailoRT v4.10.0 (October 2022)

Supported Environments

- Added support for *Yocto* Kirkstone
- Ubuntu 22.04 is now supported for C/C++ API (preview) via *source compilation*

Model Scheduler

- Model Scheduler is now released. To enable it, use `hailo_vdevice_params_t::scheduling_algorithm`
- Added *Stream Multiplexer* support (preview)
- Added *Monitor* support (preview)

Note: Model Scheduler will be default from version 4.12.0 (2023-01 suite).

Multi-Process support (preview)

- HailoRT now supports a *Multi-Process Service*, which enables multi-process inference (over a single device)
- HailoRT service is supported via API (C/C++) and `hailortcli`
- Added a Multi-Process Service *example*

Firmware

- Added over-current throttling mechanism
- Deprecated over-current monitoring thresholds in *user configuration*

API

- Added new functions to support new `hailo_device_id_t` API, such as create, scan, and get functions. Old API will be deprecated

CLI

- Operations will be executed on all connected devices (same as previously passing `-s *`) by default, unless a device ID is specified
- Added support for device IDs when using `-s`, to execute on the specified devices one-by-one
- In case of using `run` command with multiple device IDs, it is executed via a single VDevice which encapsulates the specified devices (and not one-by-one)
- Alignment of `hailo` command-line tool to `hailortcli`:
  - Deprecate parameter `--device-type` for all commands that uses Hailo devices.
  - Deprecation of the following commands: `hailo udp`, `hailo udp-limiter`, use `hailo udp-rate-limiter` instead
  - Added support for `hailo parse-hef`
- Updated CLI11 library to v2.2.0 in `hailortcli`

Windows

- Added Model Scheduler support (preview)

- Various bug fixes and stability improvements

HailoRT v4.9.0 (July 2022)

Supported Formats

- Added support for multi-network-group HEFs

Logger

- Added support for disabling or redirecting HailoRT log via `HAILORT_LOGGER_PATH` environment variable:
  - Not setting `HAILORT_LOGGER_PATH`, or setting it to an empty string, will set the log file path to the current directory
  - Setting `HAILORT_LOGGER_PATH` to 'NONE' will disable the log file

HailoRT v4.8.1 (July 2022)

Runtime Profiler

- Fixed a bug in the Runtime Profiler latency breakdown calculation, that caused all configuration to be twice as long from the actual latency

Model Scheduler

- Fixed a bug that caused random timeouts when switching between multiple network groups
- Fixed a bug that caused a timeout in network groups with defined threshold and timeout

HailoRT v4.8.0 (July 2022)

Model Scheduler (preview)

- Automatic switch between different network groups is now supported via *Model Scheduler* (only for C, C++ APIs)
- Scheduling algorithm is controlled via `hailo_vdevice_params_t::scheduling_algorithm`
- Added *switch_network_groups_example* which demonstrate automatic network group switch using model scheduler (C, C++)

Supported Frameworks

- Inference with ONNX Runtime is now supported (preview) - see *ONNX Runtime*

GStreamer

- Added HailoNet support for model scheduler

Windows

- HailoRT for Windows (driver, library and C/C++ API) are now released

Note: PyHailoRT for Windows is still in preview.

Note: Other components like *GStreamer Plugin* and TAPPAS are not yet supported on Windows.

Compilation

- Added support for CMake's `install`, which installs all HailoRT artifacts on the machine

- Added support for CMake's `find_package()`

Examples

- Removed `switch_hefs_example` (C, C++)
- Renamed `switch_hefs_example_threads_reuse` to `switch_network_groups_manually_examp` (C, C++)

CLI

- Removed support of `scan` as a sub command of `fw-control`. To scan all devices, see `scan` command
- Removed support of `udp` as device-type. Use `eth` instead
- Removed support of `--ip` in scan command. Use `--interface-ip` or `--interface` instead
- Removed support of `hef` as an optional argument (`run --hef <HEF_PATH>`). Passing HEF file/directory path is now positional (`run <HEF_PATH>`)
- hailo cli:
  - Removed support of `broadcast` command. To scan all devices, see `hailo scan` command
  - Removed support of `infer` command. To run inference on hailo device, see `hailo run` command

Note: Ubuntu 18.04 will be deprecated in HailoRT deb package future version.

Note: Python 3.6 will be deprecated in HailoRT future version.

HailoRT v4.7.0 (April 2022)

Supported Environments

- Python 3.9 is now supported
- Added support for Yocto Honister (preview)
- Yocto Hardknott is now fully supported

Power Measurements

- Added *power measurement example* that demonstrates how to perform a continuous power measurement
- Added measurement buffer index (replaced index), which represents the available buffers to store power measurements on the device for C, C++ (`hailo_measurement_buffer_index_t`) and Python API (`MeasurementBufferIndex`)
- The arguments delay_milliseconds (in C, C++ `hailort::Device::start_power_measurement()`) and delay (in Python `start_power_measurement()`) are deprecated. This value is now calculated in libhailort, and derived from other arguments passed to this function

CLI

- Added support for some `hailortcli` *operations* (like reset) which now can run on multiple PCIe devices one-by-one, by passing `-s *`. For more information, run `hailortcli --help`
- Added support for `--power-mode` in *hailortcli benchmarks*

API

- Python API

- Restructure of hailo_platform directory. Importing from *hailo_platform.drivers* is deprecated, please import from *hailo_platform* module directly
- Added support for clear function in InputVStreams (`clear()`) and OutputVStreams (`clear()`)

HailoRT v4.6.0 (March 2022)

- HailoRT is now open-source - see HailoRT GitHub
- All example files (C/C++/Python) have been moved to GitHub
- HailoRT can now be installed with new packages, see *installation*
- Added *multiple_device_example* that demonstrates how to work with multiple devices using virtual device (VDevice)
- Set all the examples except raw_streams_example to use virtual device (C and C++ examples)

API

- Replaced transformers with transform_context (C and C++ APIs):
  - hailo_input_transformer and its usages are deprecated, and replaced by `hailo_input_transform_context`
  - InputTransformer and its usages are deprecated, and replaced by `hailort::InputTransformContext`
  - hailo_output_transformer and its usages are deprecated, and replaced by `hailo_output_transform_context`
  - OutputTransformer and its usages are deprecated, and replaced by `hailort::OutputTransformContext`

Windows

- C++ API is now supported (Removed the usage of `/D_HAILO_EXPORTING`)
- Added Python support (preview)

HailoRT v4.5.0 (February 2022)

- Added On-chip NMS support for multi context networks
- Added support for Yocto Hardknott (preview)
- Unified the Linux and Windows user guides and added *Windows installation instructions*

Windows

- Added C examples to Windows package
- Fixed a bug that caused `hailortcli` to not run properly when the user does not have write privileges for the log file

GStreamer

- HailoNet Gstreamer plugin will not be active if there is only one HailoNet and `is-active` property is false
- Prevent multiple HailoNets with the same network

API

- get_device_id is now exported in Python API (via 'device_id')
- `is_multi_context` field is added to `hailo_network_group_info_t`
- Added a feature in C++ and Python APIs: Support Multiple network pipeline of only a subset of those networks and infer on a subset of the networks in the pipeline

CLI

- Added a progress bar per network to `hailortcli` infer
- `hailortcli` parse-hef now shows whether the HEF is single-context or multi-context

HailoRT v4.3.2 (February 2022)

- Fixed a bug in libhailort that caused a crash in rare cases where the batch size is larger than 1
- Fixed a bug in Yocto Gatesgarth recipes

HailoRT v4.3.1 (February 2022)

- This version was released for Windows only

HailoRT v4.4.0 (January 2022)

- Removed support for older Yocto versions, see *Yocto*)
- Added *user controlled switch* support and VDevice support (preview) in GStreamer plugin
- Added (`hailort::InputVStream::flush()`) function to InputVStream
- `hailortcli` now prints results per network, when multiple networks are used
- `hailortcli` parse-hef command now shows input/output format and type
- Data quantization functions (quantizaiton.hpp) are now exported for C++ (`hailort::Quantization`)
- Inference pipeline functions are now exported for C++ (`hailort::InferVStreams`)

3. Installation

This chapter presents the system requirements and installation instructions per operating system/distribution.

if *system requirements* are met, Hailo provides several possible ways to install HailoRT, depends on the operating system:

- Installing on *Ubuntu*
- Installing on a *Yocto-based distribution*
- Installing on *other Linux distributions* by compiling from source code
- Installing on *Windows*

3.1. System Requirements

For the installation of HailoRT, the following minimum system requirements are necessary:

- Linux or Windows
- 2+ GB RAM (4+ GB RAM recommended)
- Physical connection, depending on the board:
  - PCIe interface or Gigabit Ethernet (802.3) interface for the evaluation board, see *Hailo-8 Evaluation board*

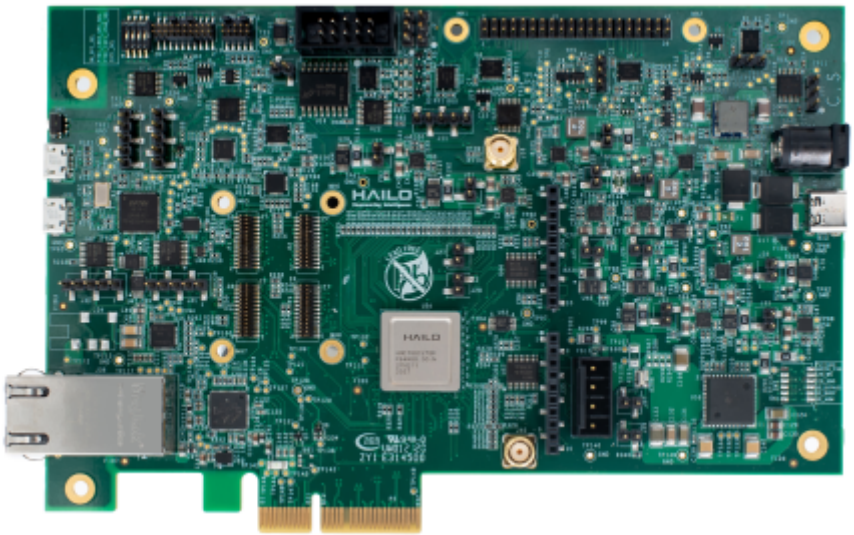

Figure 2. Hailo-8 Evaluation board

  - M.2 connector for the M.2 board, see *Hailo-8 M.2 board*

Figure 3. Hailo-8 M.2 board

- mPCIe connector for the mPCIe board, see *Hailo-8 mPCIe board*

Figure 4. Hailo-8 mPCIe board

- Python 3.8, 3.9 or 3.10 (optional)

Note: Additional requirements may be needed depending on the installation method, see below.

Note: The PCIe driver has been tested on several Linux kernel versions, including `4.15.0-39-generic`, `5.0.16-050016-generic`, `5.4.0-62-generic` and `5.11.0-040`.

Note: The RAM requirement depends on the number of the network's inputs and outputs, and on the data throughput.

3.2. Installation on Ubuntu

There are several ways to install HailoRT on Ubuntu:

- As part of the **Hailo Software Suite** – Please refer to the suite installation instructions inside the Hailo Dataflow Compiler User Guide.
- Using a **Docker container** (without the full suite) – Please refer to the *Docker instructions*.
- Using the **Ubuntu installer**.

Note: Supported versions - Ubuntu 20.04/22.04, x86_64/aarch64.

The rest of this section focuses on the Ubuntu installer.

3.2.1. Ubuntu Installer Requirements

Note: These software requirements are additionally required *system requirements*.

- build-essential package (needed to compile the PCIe driver)
- (Optional) bison, flex, libelf-dev and dkms packages (needed to register the PCIe driver using DKMS)
- (Optional) curl (needed to download the firmware when installing without the PCIe driver)
- (Optional) CMake (needed to compile the HailoRT examples)

- (Optional) `pip` and `virtualenv` (needed for pyhailort)
- (Optional) `systemd` (needed for *Multi-Process service*)

3.2.2. Installing HailoRT on Ubuntu

The HailoRT Ubuntu offers several files, select the files according to your requirements:

1. **HailoRT PCIe driver** and FW - `hailort-pcie-driver_<version>_all.deb`
2. **HailoRT** for the platform architecture - one of the following:
   - `hailort_<version>_amd64.deb` – HailoRT for amd64
   - `hailort_<version>_arm64.deb` – HailoRT for arm64
   - `hailort_<version>_armel.deb` – HailoRT for armel
   - `hailort_<version>_armv4.deb` – HailoRT for armv4
3. **PyHailoRT** for the the platform architecture and installed Python version - one of the following:
   - `hailort-<version>-cp38-cp38-linux_x86_64.whl` – PyHailoRT for python3.8, x86_64
   - `hailort-<version>-cp39-cp39-linux_x86_64.whl` – PyHailoRT for python3.9, x86_64
   - `hailort-<version>-cp310-cp310-linux_x86_64.whl` – PyHailoRT for python3.10, x86_64
   - `hailort-<version>-cp38-cp38-linux_aarch64.whl` – PyHailoRT for python3.8, aarch64
   - `hailort-<version>-cp39-cp39-linux_aarch64.whl` – PyHailoRT for python3.9, aarch64
   - `hailort-<version>-cp310-cp310-linux_aarch64.whl` – PyHailoRT for python3.10, aarch64

Note: PyHailoRT is optional and is needed for using the Python API

Download

Download the relevant files for your environment from our developer zone: https://hailo.ai/developer-zone/sw-downloads/

To install HailoRT `hailort_<version>_<arch>.deb` is required. Information about machine's architecture can be found using:

```
# .deb architecture
dpkg --print-architecture
```

To install the PCIe driver `hailort-pcie-driver_<version>_all.deb` is required.

Optional - To install PyHailoRT, you'll also need: `hailort-<version>-<python-tag>-<abi-tag>-<platform-tag>.whl`.

You can find your python version and architecture using:

```
# Python versions' major and minor digits
python -V | cut -d. -f1,2 | tr -dc '0-9' |
# .whl architecture
uname -m
```


Installation of the PCIe Driver Only

Run the following command to install only the PCIe driver:

```
sudo dpkg --install hailort-pcie-driver_<version>_all.deb
```

Note: A prompt regarding using DKMS (Dynamic Kernel Module Support) is expected and it is recommended to approve it.

Note: PC restart is required after driver installation.

Installation with the PCIe Driver

Run the following command to install HailoRT including the PCIe driver:

```
sudo dpkg --install hailort_<version>_$(dpkg --print-architecture).deb hailort-
↪pcie-driver_<version>_all.deb
```

This command will install HailoRT. It will also compile the PCIe driver for the machine's kernel version and install the driver. The command needs root permissions (sudo).

Note: PC restart is required after driver installation.

After boot, you can use the *hailortcli tool* and run `hailortcli scan` to validate that the device is identified:

```
hailortcli scan
```

The scan command should find the device.

Note: The PCIe driver is not signed. In some systems it means that secure boot has to be disabled to load the driver. Users who wish to use secure boot while HailoRT PCIe driver is not signed, can use MOK (Machine-Owner Key), which can be used for (locally) signing third-party drivers. Please notice this is an advanced feature and is recommended for users who are familiar with the process of locally-signing drivers.

Identifying Device's Serial Number

Run the following command to obtain the serial number from the device:


```
hailortcli fw-control identify
```

Installation Without the PCIe Driver (for Ethernet based platforms)

Run the following command to install HailoRT only (without the PCIe driver):

```
sudo dpkg --install hailort_<version>_$(dpkg --print-architecture).deb
```

Run the following command to download the Ethernet firmware (only needed when loading the firmware is not via the driver):

```
sudo rm /lib/firmware/hailo/hailo8_fw.bin
sudo curl https://hailo-hailort.s3.eu-west-2.amazonaws.com/Hailo8/<VERSION>/FW/
↪hailo8_fw.<VERSION>_eth.bin --create-dirs -o /lib/firmware/hailo/hailo8_fw.bin
```

Note: If the Hailo device is connected via Ethernet, use the *Ethernet command line tools*.

To load the firmware via Ethernet, see *Firmware Update Tool*.

Also, the *Ethernet related shell scripts* are useful to configure the Ethernet connection in the host's side.

Installation of pyHailoRT into a New Environment

Run the following command to install PyHailoRT into a new virtual environment:

```
virtualenv -p python<python_version> hailo_platform_venv && . hailo_platform_venv/
↪bin/activate && pip install ./hailort-<version>-<python_tag>-<abi_tag>-<platform_
↪tag>.whl
```

For example, for installing PyHailoRT **v4.12.0** to a **Python-3.8** environment on a **x86_64 platform**, the command would be:

```
virtualenv -p python3.8 hailo_platform_venv && . hailo_platform_venv/bin/activate &&
↪ pip install ./hailort-4.12.0-cp38-cp38-linux_x86_64.whl
```

Installation of pyHailoRT into an Existing Environment

Run the following command to install PyHailoRT into existing virtual environment:

```
source <virtualenv>/bin/activate && pip install ./hailort-<version>-<python_tag>-
↪<abi_tag>-<platform_tag>.whl
```

3.2.3. Uninstalling HailoRT

For installed versions **older than 4.6** - assuming HailoRT was installed using the Ubuntu installer, run the following commands to uninstall HailoRT:

```
source hailo_platform_venv/bin/activate
uninstall-hailo-platform
```

For versions **newer or equal to 4.6**, run the following commands:

- Remove the PCIe driver:


```
sudo dpkg --purge hailort-pcie-driver
```

- Remove HailoRT (excluding pyhailort):

```
sudo dpkg --purge hailort
```

- Remove pyhailort (if installed):

```
source hailo_platform_venv/bin/activate
pip uninstall hailort
```

3.3. Installation on Yocto-based Linux Distribution

See the *Yocto* page for details about Yocto integration.

3.4. Compiling from Sources

3.4.1. Compiling Hailo PCIe Driver from Sources

See the *PCIe driver* page for details about compiling the PCIe driver from sources.

3.4.2. Compiling HailoRT from Sources

Using HailoRT with other Linux distributions is possible via source compilation. On Ubuntu, it is even sometimes useful to compile from sources, for example in order to keep ABI integrity. HailoRT sources can be cloned from GitHub using:

```
git clone https://github.com/hailo-ai/hailort.git
```

Compiling the sources is done with the following command:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release && cmake --build build --config release
```

The compilation will create two artifacts:

- Binary called `hailortcli` located in `build/hailort/hailortcli/`
- Library called `libhailort.so.<version>` located in `build/hailort/libhailort/src/`

Note: By adding `--target install` to the CMake command, HailoRT artifacts will be installed on the machine.

Linux:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release && sudo cmake --build build --config
↪release --target install
```

after installation (either by installer or CMake install) one can link with libhailort by using CMake's `find_package()`. See 'hailort/libhailort/examples/CMakeLists.txt' for reference.

Note: By adding `-DHAILO_BUILD_EXAMPLES=1` to the CMake command, examples targets will be added to the project (useful for debug build of C++ examples, instead of linking them with pre-installed HailoRT).

You can now run it with:


```
build/hailort/hailortcli/hailortcli
```

Note: To compile sources on Windows please see *Windows compile from sources*

3.4.3. Compiling specific HailoRT targets

Compiling a specific target is supported using CMake's API `--target`

Supported targets are: `libhailort`, `hailortcli`

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release && cmake --build build --config release -
↪-target libhailort
```

Some HailoRT targets require additional cmake flags to be built.

Compilation of hailort-python-binding:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_PYBIND=1 -DPYBIND11_
↪PYTHON_VERSION=<py_version> && cmake --build build --config release --target _
↪pyhailort
```

Compilation of hailort-gstreamer-binding:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_GSTREAMER=1 && cmake --
↪build build --config release --target gsthailo
```

Compilation of hailort-examples:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release -HAILO_BUILD_EXAMPLES=1 && cmake --
↪build build --config release --target hailort_examples
```

3.5. Validating that the PCIe Driver was Successfully Installed on Linux

Make sure your device is connected to a PCIe/M.2/mPCIe slot. Run the following command:

```
lspci | grep Co-processor
```

The expected output is one of the following:

```
65:00.0 Co-processor: Hailo Technologies Ltd. Hailo-8 AI Processor (rev 01)
```

Figure 5. Results if PCIe IDs are up to date

```
04:00.0 Co-processor: Device 1e60:2864 (rev 01)
```

Figure 6. Results if PCIe IDs are not up to date

If the second output is received, update the PCIe IDs file on your device by running:

```
sudo update-pciids
```

If the Hailo device does not appear in the result, please ensure the following:

- The device is connected properly to the slot.
- The PCIe driver installation did not output any error messages.

- The machine was rebooted after installing the PCIe driver.
- The PCIe slot is working properly.

3.6. Linux Installation Troubleshooting

This section contains common possible issues that you may encounter after connecting Hailo-8 and installing the driver and library.

3.6.1. Common Errors

Improper PCIe device enumeration

How to verify? From the terminal, run

```
lspci | grep hailo
```

The device should be listed in the terminal output, see *driver validation*

Possible root cause Improper mechanical installation

Possible solution Verify the module is properly attached and secured into the M.2 slot

Possible root cause Slot is not functional

Possible solution Verify the slot in use is a valid M.2 slot. Check to see if the slot is disabled in the platform BIOS

Device driver is not properly installed

How to verify? From the terminal, run

```
lsmod | grep hailo_pci
```

The device should be listed in the terminal output

Possible root cause Driver not installed

Possible solution Re-install the driver, see *driver installation only*

Device firmware not loaded

How to verify? From the terminal, run

```
dmesg | grep hailo
```

The firmware load process and events appear there - and if either the load process has ended with success or failure

Possible root cause Firmware not loaded

Possible solution If the firmware load has failed (during boot) - the reason may be specified in the log. Re-install the driver, see *driver installation only*

Module not identified by HailoRT

How to verify? From the terminal, run

```
hailortcli scan
```

The module should be listed in the terminal output

Possible root cause HailoRT library not installed correctly

Possible solution Re-install the library, see *HailoRT installation*

3.7. Installation on Windows

3.7.1. Windows Requirements

Note: These software installations are additionally required: *system requirements*.

- Windows 10/11 64-bit
- (Optional) CMake and Visual Studio Build Tools – in order to compile applications that use HailoRT

Note: Validated on Visual Studio Build Tools 2019.

3.7.2. Windows Installation Instructions

1. Connect the Hailo device to the host.
2. Download the installer file `hailort_<version>_windows_installer.msi` from Hailo's website.
3. Run the installer and follow the instructions. Check the `pyHailoRT` or `multi-process service` boxes if you wish to install these features.
4. Reboot the host.
5. After boot, you can use the *hailortcli tool* and run `hailortcli scan` to validate that the device is identified.
6. If `pyHailoRT - preview` was checked, the Python wheel will be found in the directory `C:\Program Files\HailoRT\python`. Refer to section *Installation of pyHailoRT into a new environment* to finish the installation of pyHailoRT.

Note: The .whl cannot be installed (via pip) from the Program Files folder.

3.7.3. Compiling HailoRT from Sources on Windows

Note: Compilation of HailoRT from sources is recommended, for example, in order to keep ABI integrity. See *clone HailoRT* and *compile HailoRT*. Notice that when compiling the HailoRT library from sources, the library will not be signed.

To compile HailoRT on Windows, run:

```
cmake -H. -Bbuild -A=x64 -DCMAKE_BUILD_TYPE=Release && cmake --build build --config
↪release --target install
```

(continues on next page)

(continued from previous page)

Compilation Notes:

- This section refers only to HailoRT library itself and not the PCIe driver, it is recommended to install to PCIe driver only via the .msi installation file.
- When compiling for Windows, add the define `NOMINMAX` to prevent collisions.

The compilation will create three artifacts:

- Binary called `hailortcli.exe` located in `build\hailort\hailortcli\Release\`
- Library called `libhailort.lib` located in `build\hailort\libhailort\src\Release\`
- Library(DLL) called `libhailort.dll` located in `build\hailort\libhailort\src\Release\`

You can now run it with:

```
hailortcli.exe
```

Note: Windows Python API support is still in *preview* stage. Python applications are supported only through the `infer()` API. See the *Python API* section and the `InferVStreams` API reference for more information.

4. Installation of HailoRT Inside a Docker Container

For easier installation HailoRT can be installed via a Docker container. HailoRT Docker contains all of the HailoRT components, including libhailort, `hailortcli`, pyhailort (Python 3.6) and HailoRT GStreamer plugin. HailoRT PCIe driver must be installed separately on the system, prior to using HailoRT inside a Docker.

4.1. Docker Installation

To install Docker:

- Install curl

```
sudo apt-get install -y curl
```

- Install Docker

```
curl -fsSL https://get.docker.com -o get-docker.sh
sh get-docker.sh
```

- Add your user (which has root privileges) to the Docker group

```
sudo usermod -aG docker $USER
```

- Reboot/log out in order to apply the changes to the group

4.2. Running HailoRT Container from Pre-Built Docker Image

Note: Requirements - Docker package, either docker.io 20.10.07 (from Ubuntu repo), or docker-ce 20.10.6 (from Docker website).

Using this method, the following intermediate steps are handled by the script:

- Installing relevant Linux kernel headers inside the container.
- Running the container with required arguments.

Note: The directory from which you use the Docker should contain following Hailo files:

- `hailo_docker_hailort_VERSION.tar`
- `run_hailort_docker.sh`

4.2.1. Installing PCIe Driver on the Host

The following step is required if the PC does not have a working `hailo_pci` driver installed, or its version is different from the current one. Install the PCIe driver on the host. See *how to download the pcie driver* and *how to install PCIe driver*

Note: PC restart is required after driver installation. After PC restart, one should be able to resume to existing container or to create a new one and start working with Hailo device.

4.2.2. Running the Container

In order to use HailoRT release Docker image, one should run the following script:

```
./run_hailort_docker.sh <hailort_image_name> <hailort_container_name>
```

For example:

```
./run_hailort_docker.sh hailo_docker_hailort_4.0.tar hailort_01
```


5. Tutorials

The tutorials below go through the inference steps in C, C++ and Python.

Note: These tutorials are part of the Hailo Software Suite.

To download the tutorials, clone the tutorials from GitHub:

```
git clone https://github.com/hailo-ai/hailort.git
```

Run the download script to download the HEFs used by the tutorials:

- Ubuntu

  ```
  cd hailort/hailort/scripts/ && ./download_hefs.sh && cd -
  ```

- Windows:

  Double-click `hailort\hailort\scripts\download_hefs.cmd`

Note: The C and C++ tutorials require HailoRT to be installed.

Note: If running only C and C++ tutorials, PyHailoRT installation can be skipped.

Note: Python tutorials require HailoRT and PyHailoRT to be installed, and the additional following packages:

1. `jupyter`
2. `matplotlib`

To install these packages, from the virtual environment PyHailoRT is installed into run:

```
pip install jupyter matplotlib
```

5.1. C inference tutorial

In this section we will provide various examples of the HailoRT API. The code mentioned below is included in the release under `hailort/libhailort/examples/c` in the github repository. In order to compile and run the examples, one should copy the examples into a machine with installed hailort, or install hailort from the github repository. See *documentation*.

All functions calls are based on the header provided in `hailort/include/hailo/hailort.h`. See the *API documentation* for more details.

Note: Write permissions are required to compile the examples from their current directory. If this is not the case, copy the examples directory to another location with the required permissions.

5.1.1. Compilation and Execution

Compiling examples is done using the following commands from examples directory:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release
```

In order to compile a specific example, either add the example name as target with a `c` prefix, or run the cmake command from the target example dir:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release --target c_vstreams_example
```

```
cd c/vstreams_example
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release
```

Note: When copying an example to a different directory, make sure to copy the `common` directory as well.

Running examples:

Run the examples using the following command, from the examples directory:

```
build/c/<example_name>/c_<example_name> [params...]
```

Cross-Compile Tutorials

Add the following options while configuring the examples using CMake:

- DCMAKE_TOOLCHAIN_FILE="<toolchain file>"
- DCMAKE_FIND_ROOT_PATH="<target dir>"
- DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ONLY

Note: For more details, see:

- Cross Compiling With CMake
- find_package — CMake 3.26.1 Documentation

5.1.2. Inference using virtual streams – *vstreams_example*

See: `hailort/libhailort/examples/c/vstreams_example.c`

Summary

- Demonstrates basic inference of a shortcut network (i.e inputs are sent through the vdevice and right back out, without any changes made to the data).
- The program uses the Hailo virtual device with default values for initialization.
- The data is sent to the vdevice via input vstreams and received via output vstreams.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Code structure

Device initialization The first step is to initialize the vdevice, we initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*. By creating the *VDevice* with a `hailo_vdevice_params_t` with default values, the model scheduler is enabled (with its default scheduling scheme - Round Robin). For that reason, we don't need to activate/deactivate the network groups in the user application, since when the scheduler is enabled, the activation and deactivation is done automatically.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()`

Configuring the vdevice from HEF The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. We init a `hailo_configure_params_t` with default values, configure the vdevice and gets a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_vdevice()`.

Creating vstreams First we need to initialize vstream params (both input and output), then we are ready to create the vstreams.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()` and `hailo_create_output_vstreams()`

Inference In this example, we use `p_thread` for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize a buffer used for send. Then the thread will send all frames in a loop.

Used APIs: `hailo_get_input_vstream_frame_size()` and `hailo_vstream_write_raw_buffer()`.

Read from device The read function is analogical to the write. Firstly we initialize a recv buffer. Then we will recv all frames in a loop.

Used APIs: `hailo_get_output_vstream_frame_size()` and `hailo_vstream_read_raw_buffer()`.

5.1.3. Multiple Device inference, using vstreams – *multi_device_example*

See: `hailort/libhailort/examples/c/multi_device_example.c`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates inference of a single network group over multiple devices.
- The program scans for Hailo devices.
- Creates a virtual device from all Hailo devices that are connected to the computer.
- The program infers HEFs with multiple virtual input streams and multiple virtual output streams.
- The program creates a thread for each virtual output stream and a thread for each virtual input stream.

Code structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the *vstreams_example* first.

Device initialization First we scan for Hailo devices, then we initialize the `hailo_vdevice_params_t` with the device count we scanned before. The next step is to create the *VDevice*.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()` `hailo_scan_devices()`.

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. We init a `hailo_configure_params_t` with default values. Then we modify batch-size to 1 for each network group. This is the default batch-size value and will not have any effect (just to demonstrate API usage). We configure the vdevice and gets a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_vdevice()`.

Build vstreams The next step is creating the input and output virtual streams for the configured network. First, we are making default virtual stream params for each virtual stream. Second, we create the virtual streams (inputs and outputs). Finally, we extract the vstream's frame size and allocating the host input and output buffers and initialize the source with random data to send to inference.

Used APIs: `hailo_make_input_vstream_params()`,`hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()`, `hailo_create_output_vstreams()`. `hailo_get_input_vstream_frame_size()` and `hailo_get_output_vstream_frame_size()`.

Infer stage

Inference In this example, we create a new thread for each input vstream using the function create_input_vstream_thread. Each created thread sends data to the vdevice using the function write_to_vdevice.

Write to device The write thread gets as input the following:

- *input_vstream* – The virtual input stream.
- *src_data* – The actual data sent to the HW.
- *src_frame_size* – Size of the data to be sent to the HW.

The write thread will send all frames by writing the HW buffer to the vdevice.

Used APIs: `hailo_vstream_write_raw_buffer()`.

We also create a new thread for each output vstream using the function create_output_vstream_thread. Each created thread receives data from the vdevice using the function read_from_vdevice.

Read from device The read function gets as input the following:

- *output_vstream* – The virtual output stream.
- *dst_data* – The actual data sent by the HW back to the host.
- *dst_frame_size* – Size of the data expected to be received by the host.

The read function will receive all frames in the following flow:

- Reading data from the device.
- No post processing is made on the received data.

Used APIs: `hailo_vstream_read_raw_buffer()`.

In order to complete the inference on each network group, the main thread then waits for all the input vstream threads to finish via hailo_join_thread, and afterwards waits for all the output vstreams threads to finish before deactivating the current activated network group and continuing onto the next.

5.1.4. Multi network inference, using virtual streams – *multi_network_vstream_example*

See: `hailort/libhailort/examples/c/multi_network_vstream_example.c`

Summary

- Demonstrates how to work with multiple networks compiled together into the same network group (co-compilation), using virtual streams.
- Working with multiple networks allows to configure network-based parameters (such as different batch per network) and gives a more native approach to work with send/receive threads per network.
- The example works with an HEF containing one network group, and two networks in the network group.
- The example uses the first Hailo device found.
- Configure the network group and get the networks information to create the vstreams for each network.
- The data is sent to the device via input vstreams and received via output vstreams.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Code structure

Device initialization The first step is to initialize the vdevice, we initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()`

Configuring the vdevice from HEF The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. We init a `hailo_configure_params_t` with default values, and change the batch size for each network in the network_group. Then, configure the vdevice and get a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_vdevice()`.

Get the networks information We use the `hailo_configured_network_group` object we got in the previous step, to get the networks information, `hailo_network_info_t` objects.

Used APIs: *hailo_get_network_infos()*.

Creating vstreams For each network in the network group, we first need to initialize vstream params (both input and output). Then, we are ready to create the vstreams for each network. The vstreams will be ready for use only after activating the network group.

Note: To create the vstream for a specific network, one should pass the specific network's name when making the vstream's params. It is also possible to use the network group's name to create all vstreams for all networks together. For more information see the documentation for `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()` and `hailo_create_output_vstreams()`

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailo_activate_network_group()`.

Inference In this example, we use `p_thread` for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize a buffer used for send. Then the thread will send all frames in a loop.

Used APIs: `hailo_get_input_vstream_frame_size()` and `hailo_vstream_write_raw_buffer()`.

Read from device

The read function is analogical to the write. Firstly we initialize a recv buffer. Then we will recv all frames in a loop.

Used APIs: `hailo_get_output_vstream_frame_size()` and `hailo_vstream_read_raw_buffer()`.

Deactivating a network group After inference is done, we need to deactivate the network group.

Used APIs: `hailo_deactivate_network_group()`.

5.1.5. Automatic switch between multiple network groups using virtual streams – *switch_network_groups_example*

See: `hailort/libhailort/examples/c/switch_network_groups_example.c`

Summary

- Demonstrates inference of multiple network groups over a single device, using VDevice and Round-Robin scheduling algorithm.
- Multiple HEFs are configured into the VDevice, creating multiple network groups.
- For each network group, create virtual streams, send threads and recv threads.
- Wait for all threads to finish, and validate their results.

See also:

See the *comparison table* between context switching and network groups switching for more information about running multiple models.

Code structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the *vstreams_example* first.

Device initialization We initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*. The `hailo_vdevice_params_t::scheduling_algorithm` is set by default to `HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN`.

Used APIs: `hailo_create_vdevice()`.

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailo_hef`, and to use it to configure the vdevice for inference. We init a `hailo_configure_params_t` with default values, configure the vdevice and gets a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_vdevice()`.

Set scheduler's timeout threshold, and priority We set higher threshold and timeout for the first model scheduling. It will give priority to the scheduling of the second model. We also set `HAILO_SCHEDULER_PRIORITY_NORMAL+1` priority for the first model scheduling. The practical meaning is that the first network will be ready to run only if `SCHEDULER_THRESHOLD` send requests have been accumulated, or more than `SCHEDULER_TIMEOUT_MS` time has passed and at least one send request has been accumulated. However when both the first and the second networks are ready to run, the first network will be preferred over the second network.

Used APIs: `hailo_set_scheduler_timeout()`, `hailo_set_scheduler_threshold()` and `hailo_set_scheduler_priority()`.

Build vstreams (for each network group) The next step is creating the input and output virtual streams for each configured network. First, we are making default virtual stream params for each virtual stream. Second, we create the virtual streams (inputs and outputs). Finally, we extract the vstream's frame size and allocating the host input and output buffers and initialize the source with random data to send to inference.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()`, `hailo_create_input_vstreams()`, `hailo_create_output_vstreams()`. `hailo_get_input_vstream_frame_size()` and `hailo_get_output_vstream_frame_size()`.

Inference In this example, we create a new thread for each input vstream using the function create_input_vstream_thread. Each created thread sends data to the device using the function write_to_device.

Write to device The write thread gets as input the following:

- *input_vstream* – The virtual input stream.
- *src_data* – The actual data sent to the HW.
- *src_frame_size* – Size of the data to be sent to the HW.

The write thread will send all frames by writing the HW buffer to the device.

Used APIs: `hailo_vstream_write_raw_buffer()`.

We also create a new thread for each output vstream using the function create_output_vstream_thread. Each created thread receives data from the device using the function read_from_device.

Read from device The read function gets as input the following:

- *output_vstream* – The virtual output stream.
- *dst_data* – The actual data sent by the HW back to the host.
- *dst_frame_size* – Size of the data expected to be received by the host.

The read function will receive all frames in the following flow:

- Reading data from the device.
- No post processing is made on the received data.

Used APIs: `hailo_vstream_read_raw_buffer()`.

In order to complete the inference on each network group, the main thread then waits for all the input vstream threads to finish via hailo_join_thread, and afterwards waits for all the output vstreams threads to finish.

5.1.6. User controlled switch between multiple HEFs using virtual streams – *switch_network_groups_manually_example*

See: `hailort/examples/c/switch_network_groups_manually_example.c`

Summary

- Demonstrates inference of multiple network groups over a single device, using VDevice.
- For simplicity, we use single input / single output network groups.
- Multiple HEFs are configured into the VDevice, creating multiple network groups.
- Create send thread and recv thread. the vstreams for each network group will be created inside these threads.
- Wait for all threads to finish, and validate their results.

See also:

See the *comparison table* between context switching and network groups switching for more information about running multiple models.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, then we set the `hailo_vdevice_params_t::scheduling_algorithm` to be `HAILO_SCHEDULING_ALGORITHM_NONE` as we want to manually switch the network groups. We create the *VDevice*.

Used APIs: `hailo_create_vdevice()`.

Pre infer stage The next step is to create a `hailo_hef`, and to use it to configure the device for inference. We init a `hailo_configure_params_t` with default values, configure the device and get a `hailo_configured_network_group` object. Afterwards, we create `hailo_input_vstream_params_by_name_t` and `hailo_output_vstream_params_by_name_t` for the generated configured network group. We do this step `HEF_COUNT` times, and store all the configured network groups and vstream parameters in an array.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()`, `hailo_configure_vdevice()`, `hailo_make_input_vstream_params()`, and `hailo_make_output_vstream_params()`.

Creating VStream params and Vstream infos

The next step is to create the virtual stream params. We create the virtual stream params from the network group using the functions `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`. Then, we get all vstream infos from the HEF.

Used APIs: `hailo_make_input_vstream_params()`, `hailo_make_output_vstream_params()` and `hailo_hef_get_all_vstream_infos()`

Inference In this example, we create one thread for inputs, and one thread for outputs, assuming we have only one input and one output per network group. The threads are responsible for creating the VStreams, allocating buffers for send / recv, and synchronizing the network group switches using `hailo_activate_network_group()` and `hailo_deactivate_network_group()` APIs. the first activation of the first network group is done in the main thread.

Used APIs: `hailo_activate_network_group()`.

Input vstream thread function The write thread gets as input the following:

- *configured_networks* – The `hailo_configured_network_group` array.
- *input_vstream_params* – The `hailo_input_vstream_params_by_name_t` array.

Both arrays are in the length of HEF_COUNT.

The write thread will first create hailo_input_vstream, and allocate a buffer (src_data) representing the input buffers.

Used APIs: hailo_create_input_vstreams() and hailo_get_input_vstream_frame_size().

Afterwards the write thread will loop over the configured network group and wait for activation (which will take place in the read thread). Once activated, we loop for INFER_FRAME_COUNT, generating random data over src_data, and write it to the device.

Used APIs: hailo_wait_for_network_group_activation() and hailo_vstream_write_raw_buffer().

Once finished, we release the created hailo_input_vstream using hailo_release_input_vstreams().

Output vstream thread function The read thread gets as input the following:

- *configured_networks* – The hailo_configured_network_group array.
- *activated_network_group* – a hailo_activated_network_group, holding the first activated network group.
- *output_vstream_params* – The hailo_output_vstream_params_by_name_t array.

Both arrays are in the length of HEF_COUNT.

The read thread will first create hailo_output_vstream, and allocate a buffer (dst_data) representing the output buffers.

Used APIs: hailo_create_output_vstreams() and hailo_get_output_vstream_frame_size().

Afterwards the read thread will loop over the configured network group and wait for activation (which should return immediately, as the network group is already activated). Once activated, we loop for INFER_FRAME_COUNT, reading from the device.

Used APIs: hailo_wait_for_network_group_activation() and hailo_vstream_read_raw_buffer().

When the reading is completed, we deactivate the current activated network group, and activate the next network group so that the input thread will be able to start sending again.

Used APIs: hailo_deactivate_network_group() and hailo_activate_network_group().

Once finished, we release the created hailo_output_vstream using hailo_release_output_vstreams().

In order to complete the inference on each network group, the main thread waits for all the threads to finish via hailo_join_thread, and release the resources.

Used APIs: hailo_release_hef() and hailo_release_vdevice().

5.1.7. Quantization of inputs/outputs – *data_quantization_example*

See: hailort/libhailort/examples/c/data_quantization_example.c

Summary

Hailo devices require input data to be quantized/scaled before it is sent to the device via a `hailo_input_stream`. Similarly, data outputted from the device via a `hailo_output_stream` needs to be 'de-quantized'/rescaled as well.

When a neural network (NN from now on) is compiled by the Dataflow Compiler, each input/output layer in the NN is assigned two floating point values that are parameters to an input/output transformation: `qp_zp` (zero_point) and `qp_scale` (fields of the struct `hailo_quant_info_t`). These values are stored in the HEF.

- Input transformation: input data is divided by `qp_scale` and then `qp_zp` is added to the result.
- Output transformation: `qp_zp` is subtracted from output data and then the result is multiplied by `qp_scale`.

In the context of the HailoRT library, there are two options to quantize the data:

- Use a virtual stream, and mark it as non-quantized in the vstream params creation. The data sent to the virtual stream will be quantized as part as the transformation process. See the *vstreams example* for more info about virtual streams.
- Use a raw stream. In this case the data needs to be quantized using `hailo_input_transform_context` or `hailo_output_transform_context`. See the *raw streams example* for more info about streams and transformations.

In this tutorial we use virtual streams for the quantization process.

In this tutorial we use the following arguments:

- *in* – The input is quantized, hence HailoRT won't quantize the input.
- *out* – The output is to be left quantized, hence HailoRT won't de-quantize the output.
- *both* – The input is quantized and the output is to be left quantized, hence HailoRT won't do either.
- *none* – The input isn't quantized and the output is to be de-quantized, hence HailoRT will do both.

Code structure

The code is similar to `hailort/libhailort/examples/c/vstreams_example.c`, so be sure to read the *vstreams example* first.

Device initialization and configuration We create a vdevice and configure it using HEF. The code is similar to the one in the *vstreams example*

Creating virtual streams In the function *create_vstreams* we create both input and output virtual stream. At this point we can set the desired quantization behavior for input streams and de-quantization behavior for output streams.

- Passing *true* under the argument *quantized* in the function `hailo_make_input_vstream_params()`, means that the input data sent to the stream (via `hailo_vstream_write_raw_buffer()`) is quantized to begin with. This will result in an input stream that doesn't quantize the input data. Passing *false* under the argument *quantized*, will lead to input data being quantized.
- Passing *true* under the argument *quantized* in the function `hailo_make_output_vstream_params()`, means that the output data received from the stream (via `hailo_vstream_read_raw_buffer()`) is to remain quantized (such as it is upon exiting the device). This will result in an output stream that doesn't de-quantize the output data. Passing *false* under the argument *quantized*, will lead to output data being de-quantized.

Note: In the call to the function `hailo_make_input_vstream_params()`, if the input data isn't quantized (i.e. *quantized_in* == *false*), we pass `HAILO_FORMAT_TYPE_FLOAT32` under the *format_type* argument. This means that the input buffer contains *float32_t*.

In addition, quantization of *uint8_t* (i.e. `HAILO_FORMAT_TYPE_UINT8`) is possible.

However, the device doesn't support sending `HAILO_FORMAT_TYPE_FLOAT32` data if *true* is passed under the argument *quantized* in the function `hailo_make_input_stream_params()`.

The rest of the code After creating the virtual streams and activating the network group, we can get the stream handles and commence with the inference. This code is almost identical to the code used in `hailort/libhailort/examples/c/vstreams_example.c`. For an explanation, see the *vstreams example*.

5.1.8. Inference using raw streams – *raw_streams_example*

See: `hailort/libhailort/examples/c/raw_streams_example.c`

Summary

- Demonstrates basic inference of a shortcut network using raw stream API.
- The program uses the first identified Hailo device.
- The data sent/received does not undergo the transformation process as required on raw streams. For a complete raw stream example with transformation, see *raw_streams_example*.

Code structure

Device initialization We open the first enumerated Hailo device (by passing NULL as value for parameter *device_id* to `hailo_create_device_by_id()`)

Used APIs: `hailo_create_device_by_id()`.

Configuring the device from HEF The next step is to create a `hailo_hef`, and to use it to configure the device for inference. We init a `hailo_configure_params_t` with default values, configure the device and gets a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_device()`.

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailo_activate_network_group()`.

Inference In this example, we use `p_thread` for inference, with a sub-thread for each output vstream receiving data and a sub-thread for each input vstream sending data.

Write to device The write thread, will firstly initialize any necessarily resources, including:

- *host_data* – Will contain the actual user data (for example - frame).
- *hw_data* – The actual data sent to the hw.
- *transform_context* – a `hailo_input_transform_context` - used to transform the data from host format to HW.

Used APIs: `hailo_network_group_get_input_stream_infos()`, `hailo_get_input_stream()`, `hailo_create_input_transform_context()` and `hailo_get_host_frame_size()`.

After initialization, the write thread will send all frames:

- First, transform the buffer from host format to HW format
- Then writing the HW buffer to the device.

Used APIs: `hailo_transform_frame_by_input_transform_context()` and `hailo_stream_write_raw_buffer()`.

Read from device The read thread, will firstly initialize any necessarily resources, including:

- *host_data* - Will contain the actual user data (The network result).
- *hw_data* - The actual data received from the hw.
- *transform_context* a `hailo_output_transform_contexts` - used to transform the data from HW format to host.

Used APIs: `hailo_network_group_get_output_stream_infos()`, `hailo_get_output_stream()`, `hailo_create_output_transform_context()` and `hailo_get_host_frame_size()`.

After initialization, the write thread will send all frames:

- First, reading data from the device.
- Then transform is from HW format to host format.

Used APIs: `hailo_stream_read_raw_buffer()` and `hailo_transform_frame_by_output_transform`

5.1.9. Async Inference Using Raw Streams – *raw_async_streams_single_thread_example*

See: `hailort/libhailort/examples/c/raw_async_streams_single_thread_example.c`

Summary

- Demonstrates basic async inference of a shortcut network using raw async stream API.
- Using single thread for the inference.
- The program uses the first Hailo device found.
- The data sent/received does not undergo the transformation process as required on raw streams. For a complete raw stream example with transformation, see *raw_streams_example*.

Code Structure

Device Initialization We open the first enumerated Hailo device (by passing NULL as value for parameter *device_id* to `hailo_create_device_by_id()`)

Used APIs: `hailo_create_device_by_id()`.

Configuring The Device From HEF The next step is to create a `hailo_hef`, and use it to configure the device for inference. We initialize a `hailo_configure_params_t` with default values and to support async streams API we set the `HAILO_STREAM_FLAGS_ASYNC` flag inside `hailo_stream_parameters_t`. We configure the device using the `hailo_configure_device()` function which returns a `hailo_configured_network_group` object.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_device()`.

Prepare Async Inference Before launching async operations, some preparation is required.

Async streams are backed by limited-sized queue for pending async transfers. The max queue sizes are provided by `hailo_input_stream_get_async_max_queue_size()` and `hailo_output_stream_get_async_max_queue_size()`. In this example, the maximum number of ongoing transfers across all streams, denoted as *ongoing_transfers*, is calculated by taking the minimum value among the queues of all streams.

If a user attempts to perform more concurrent async operations than the value of *ongoing_transfers*, they may encounter an error with the code `HAILO_QUEUE_IS_FULL`.

Used APIs: `hailo_input_stream_get_async_max_queue_size()` and `hailo_output_stream_get_async_max_queue_size()`.

Async Inference

- The network group can now be activated.
- For each output stream, *ongoing_transfers* async read requests are launched.
  - First we allocate memory for each transfer, ensuring it is aligned to the system page size, as required by `hailo_stream_read_raw_buffer_async()`.
  - The async read operation is initiated by calling `hailo_stream_read_raw_buffer_async()`, passing the allocated buffer.
  - Once a read operation is finished, the associated async callback function is triggered. In this example, the callback function examines the `hailo_stream_read_async_completion_info_t::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async read request is initiated using the same buffer and callback.

    In a real application the results obtained by the read operations can be passed for post-processing or display.
- For each input stream, *ongoing_transfers* async writes requests are launched.
  - First we allocate memory for each transfer, ensuring it is aligned to the system page size, as required by `hailo_stream_write_raw_buffer_async()`. In this example, the buffer initially contains garbage data.
  - The async write operation is initiated by calling `hailo_stream_write_raw_buffer_async()`, passing the allocated buffer.
  - Once a write operation is finished, the associated async callback function is triggered. In this example, the callback function examines the `hailo_stream_write_async_completion_info_t::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async write request is initiated using the same buffer and callback.

    Real applications may choose to fill the buffer with a new frame, free it, or return it to some buffer pool for later use.
- Since further async operations are launched within each callback, the inference is executed in parallel to the main thread. In this example, the main thread remains idle and sleeps for 5 seconds, allowing the inference to run in the background.
- To stop the inference process, the network group must be deactivated by calling the `hailo_deactivate_network_group()` function. After this function is called, it is guaranteed that all pending async callbacks will be invoked. Any transfers that have not been completed will be called with the status `HAILO_STREAM_ABORTED_BY_USER`

Used APIs: `hailo_stream_read_raw_buffer_async()` `hailo_stream_write_raw_buffer_async()` `hailo_activate_network_group()` and `hailo_deactivate_network_group()`.

5.1.10. Inference using inference pipeline – *infer_pipeline_example*

See: `hailort/examples/c/infer_pipeline_example.c`

Summary

- Demonstrates basic inference of a shortcut network (i.e inputs are sent through the device and right back out, without any changes made to the data).
- The program uses the first Hailo Ethernet device found connected to a specified interface.
- The data is both sent to the device and received using the inference pipeline.

Code structure

Device initialization We open the first enumerated Hailo Ethernet device found on the specified interface.

Used APIs: `hailo_scan_ethernet_devices()`, `hailo_create_ethernet_device()`

Configuring the device from HEF The next step is to create a `hailo_hef`, and to use it to configure the device for inference. We init a `hailo_configure_params_t` with default values, configure the device and get a `hailo_configured_network_group` object.

Note - It is possible to limit the rate in which data is sent using a rate limiter: One way of doing that is by using the command line tool *hailortcli udp-rate-limiter*. Another way is calling `hailo_calculate_eth_input_rate_limits()`, which returns the input bandwidth that each stream should be limited to. These rates should then be assigned to their respective fields in `hailo_eth_input_stream_params_t::rate_limit_bytes_per_sec` under `hailo_configure_params_t`. See also `hailo_eth_input_stream_params_t`.

Used APIs: `hailo_create_hef_file()`, `hailo_init_configure_params()` and `hailo_configure_device()`.

Creating VStream params and Vstream infos The next step is creating the virtual stream params. We create the virtual stream params from the network group using the functions `hailo_make_input_vstream_params()` and `hailo_make_output_vstream_params()`. Then, we get all vstream infos from the HEF.

Used APIs: `hailo_make_input_vstream_params()`,`hailo_make_output_vstream_params()` and `hailo_hef_get_all_vstream_infos()`

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailo_activate_network_group()`.

Prepareing buffers before inference We need to prepare the input data buffers, and also the buffers for the inference output.

Used APIs: `hailo_get_vstream_frame_size()`.

Inference Finally, we run inference on the input data using the inference pipeline.

Used APIs: `hailo_infer()`.

5.1.11. Power measurement – *power_measurement_example*

See: `hailort/libhailort/examples/c/power_measurement_example.c`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates how to perform a continuous power measurement on the chip.
- The program uses a VDevice created from all the Hailo devices that are connected to the computer.
- In this tutorial we use the following arguments to control the measurement type:
  - *power* – measure power consumption in W.
  - *current* – measure current in mA.

Note: For instantaneous power measurement, see `hailo_power_measurement()`.

For instantaneous temperature measurement, see `hailo_get_chip_temperature()`.

Code structure

VDevice initialization First we scan for devices, then we initialize the `hailo_vdevice_params_t` with the device count we scanned before. The next step is to create the *VDevice*.

Used APIs: `hailo_create_vdevice()`, `hailo_init_vdevice_params()` `hailo_scan_devices()`.

Power measurement

Initialization For sending controls to a `hailo_device`, we need to get the underlying physical devices from the `hailo_vdevice`, using the function `hailo_get_physical_devices()`. For each physical device, we first call the function `hailo_stop_power_measurement()` to make sure there aren't any former measurements currently running. The next step is to set power measurement arguments using the function `hailo_set_power_measurement()`, and to start the measurement using the function `hailo_start_power_measurement()`

Used APIs: `hailo_get_physical_devices()`, `hailo_stop_power_measurement()`, `hailo_set_power_measurement()` and `hailo_start_power_measurement()`.

Stopping measurement and fetching the results After starting the measurement, we sleep for a constant period of time (5 seconds). Once we finish sleeping, we go over all physical devices and do the following:

- Stopping the current measurement using the function `hailo_stop_power_measurement()`.
- Getting the measurement results using the function `hailo_get_power_measurement()`.
- Printing measurement results.

Used APIs: `hailo_stop_power_measurement()`, `hailo_get_power_measurement()`.

5.1.12. Notification Callback – *notification_callback_example*

See: `hailort/libhailort/examples/c/notification_callback_example.c`

Summary

- Demonstrates the basic usage of notification callbacks.
- The program creates a device and then sets and removes a notification callback on it.

Code structure

Device initialization

First we scan for a Hailo device using `hailo_scan_devices()`. Then, we use the `hailo_device_id_t` we got from the scan to create the device by calling `hailo_create_device_by_id()` with it.

Used APIs: `hailo_scan_devices()`, `hailo_create_device_by_id()`.

Creating the callback We create a `hailo_notification_callback` and assign it to the address of the callback function void callback(hailo_device device, const hailo_notification_t *notification, void *opaque), that prints an overcurrent alarm message.

Set the callback notification

By calling `hailo_set_notification_callback()` with the following notification id: `hailo_notification_id_t::HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM`, the callback function will be called when we will get this notification.

Used APIs: `hailo_set_notification_callback()`.

Remove the callback notification

By calling `hailo_remove_notification_callback()` with the same `hailo_notification_id_t` as in the `hailo_set_notification_callback()`, we remove this notification callback.

Used APIs: `hailo_remove_notification_callback()`.

5.2. C++ inference tutorial

In this section we will provide various examples of the HailoRT API. The code mentioned below is included in the release under `hailort/libhailort/examples/cpp` in the github repository. In order to compile and run the examples, one should clone the repository, and install hailort from the github repository. See *documentation*.

Note: *(for C++ API users)* In order to maintain ABI Integrity between libhailort and the examples code that uses the C++ API, you should *compile libhailort* instead of using the pre-compiled binaries.

All functions calls are based on the headers provided in the directory `hailort/include/hailo/`. One can include `hailort/include/hailo/hailort.hpp` to access all functions. See the *API documentation* for more details.

5.2.1. Compilation and Execution

Compiling examples is done using the following commands from examples directory:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release
```

In order to compile a specific example, either add the example name as target with a `cpp` prefix, or run the cmake command from the target example dir:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release --target cpp_vstreams_example
```

```
cd cpp/vstreams_example
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release
cmake --build build --config release
```

Note: For debug builds of C++ examples, it is recommended to build the examples as part of *source compilation of HailoRT*.

Running examples:

Run the examples using the following command, from the examples directory:

```
build/cpp/<example_name>/cpp_<example_name> [params...]
```

Cross-Compile Tutorials

Add the following options while configuring the examples using CMake:

- DCMAKE_TOOLCHAIN_FILE="<toolchain file>"
- DCMAKE_FIND_ROOT_PATH="<target dir>"
- DCMAKE_FIND_ROOT_PATH_MODE_PACKAGE=ONLY

Note: For more details, see:

- Cross Compiling With CMake
- find_package — CMake 3.26.1 Documentation

5.2.2. Inference using virtual streams – *vstreams_example*

See: `hailort/libhailort/examples/cpp/vstreams_example.cpp`

Summary

- Demonstrates basic inference of a shortcut network using the virtual stream C++ API.
- The program uses the Hailo vdevice.
- The program infers hefs with multiple virtual input streams and multiple virtual output streams.
- The program creates a thread for each virtual output stream and a thread for each virtual input stream.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*. By creating the *VDevice* with a `hailo_vdevice_params_t` with default values, the model scheduler is enabled (with its default scheduling scheme - Round Robin). For that reason, we don't need to activate/deactivate the network groups in the user application, since when the scheduler is enabled, the activation and deactivation is done automatically.

Used APIs: `hailort::VDevice::create()`

Configuring the vdevice from HEF The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We init a `hailort::NetworkGroupsParamsMap` with default values, configure the vdevice using the VDevice class function `hailort::VDevice::configure()` and gets a `hailort::ConfiguredNetworkGroupVector` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::VDevice::configure()`.

Build VStreams The next step is creating the virtual streams. First we initialize vstream params (both input and output), then we create the virtual streams from the network group.

Used APIs: `hailort::NetworkGroup::make_input_vstream_params()`, `hailort::VStreamsBuilde` `hailort::NetworkGroup::make_output_vstream_params()` and `hailort::VStreamsBuilder::create_output_vstreams()`,

Inference In this example, we use *std::thread* for inference, with a sub-thread for each output thread receiving data from the device calling the read_all() function, and another sub-thread for each input thread calling the write_all() function for sending data to the device.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the device. Then the read_all thread will receive all of the frames from the device.

Used APIs: `hailort::OutputVStream::read()`.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device. Then the write_all thread will send all of the frames to the device.

Used APIs: `hailort::InputVStream::write()`.

5.2.3. Multiple Device inference using vstreams – *multi_device_example*

See: `hailort/libhailort/examples/cpp/multi_device_example.cpp`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates inference of a single network group over multiple devices.
- The program scans for Hailo devices.
- Creates a virtual device from all Hailo devices that are connected to the computer.
- The program infers HEFs with multiple virtual input streams and multiple virtual output streams.
- The program creates a thread for each virtual output stream and a thread for each virtual input stream.

Code structure

The code is similar to `hailort/libhailort/examples/cpp/vstream_example.cpp`, so be sure to read the *vstreams_example* first.

VDevice initialization First we scan for Hailo devices, then we initialize the `hailo_vdevice_params_t` with the device count we scanned before. The next step is to create the vdevice.

Used APIs: `hailort::Device::scan()`, `hailort::VDevice::create()`.

Configuring the vdevice from HEF The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We init a `hailort::NetworkGroupsParamsMap` with default values. Then we modify batch-size to 1 for each network group. This is the default batch-size value and will not have any effect (just to demonstrate API usage). We configure the vdevice using the VDevice class function `hailort::VDevice::configure()` and gets a `hailort::ConfiguredNetworkGroupVector` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::VDevice::configure()`.

Build VStreams The next step is creating the virtual streams. We create the virtual streams from the network group using the function `hailort::VStreamsBuilder::create_vstreams()`.

Used APIs: `hailort::VStreamsBuilder::create_vstreams()`

Inference In this example, we use *std::thread* for inference, with a sub-thread for each output thread receiving data from the vdevice calling the read_all() function, and another sub-thread for each input thread calling the write_all() function for sending data to the device.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the vdevice. Then the read_all thread will receive all of the frames from the device.

Used APIs: `hailort::OutputVStream::read()`.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device. Then the write_all thread will send all of the frames to the device.

Used APIs: `hailort::InputVStream::write()`.

5.2.4. Multi network inference, using virtual streams – *multi_network_vstream_example*

See: `hailort/libhailort/examples/cpp/multi_network_vstream_example.cpp`

Summary

- Demonstrates how to work with multiple networks compiled together into the same network group (co-compilation), using virtual streams.
- Working with multiple networks allows to configure network-based parameters (such as different batch per network) and gives a more native approach to work with send/receive threads per network.
- The example works with an HEF containing one network group, and two networks in the network group.
- The example uses the first Hailo device found.
- Configure the network group and get the networks information to create the vstreams for each network.
- The data is sent to the device via input vstreams and received via output vstreams.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*.

Used APIs: `hailort::VDevice::create()`

Configuring the vdevice from HEF The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We init a `hailort::NetworkGroupsParamsMap` with default values, and change the batch size for each network in the network_group. Then, we configure the vdevice using the VDevice class function `hailort::VDevice::configure()` and get a `hailort::ConfiguredNetworkGroupVector` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::VDevice::configure()`.

Get the networks information The next step is to get the networks information. The function get_network_infos in the example uses the (shared ptr to a) `hailort::ConfiguredNetworkGroup` object to get the networks information, a vector of `hailo_network_info_t` objects is returned.

Used APIs: `hailort::ConfiguredNetworkGroup::get_network_infos()`

Build VStreams The next step is creating the virtual streams. The function create_vstreams_per_network is creating the vstreams for each network using the network group and the networks information.

Note: To create the vstream for a specific network, one should pass the specific network's name. It is also possible to use the network group's name to create all vstreams for all networks together. For more information see the documentation for `hailort::VStreamsBuilder::create_vstreams()`.

Used APIs: `hailort::VStreamsBuilder::create_vstreams()`

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailort::ConfiguredNetworkGroup::activate()`.

Inference In this example, we use *std::thread* for inference, with a sub-thread for each output thread receiving data from the device calling the read_all() function, and another sub-thread for each input thread calling the write_all() function for sending data to the device.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the device. Then the read_all thread will receive all of the frames from the device.

Used APIs: `hailort::OutputVStream::read()`.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device. Then the write_all thread will send all of the frames to the device.

Used APIs: `hailort::InputVStream::write()`.

5.2.5. Automatic switch between multiple network groups using virtual streams – *switch_network_groups_example*

See: `hailort/libhailort/examples/cpp/switch_network_groups_example.cpp`

Summary

- Demonstrates inference of multiple network groups over a single device, using VDevice and Round-Robin scheduling algorithm.
- Multiple HEFs are configured into the VDevice, creating multiple network groups.
- For each network group, create virtual streams, send threads and recv threads.
- Wait for all threads to finish, and validate their results.

See also:

See the *comparison table* between context switching and network groups switching for more information about running multiple models.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*. The `hailo_vdevice_params_t::scheduling_algorithm` is set by default to `HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN`.

Used APIs: `hailort::VDevice::create()`

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We configure the vdevice using the VDevice class function `hailort::VDevice::configure()`, and getting a `hailort::ConfiguredNetworkGroupVector` object. We store all the configured network groups in a vector.

Used APIs: `hailort::Hef::create()` and `hailort::VDevice::configure()`.

Set scheduler's timeout, threshold and priority We set higher threshold `SCHEDULER_THRESHOLD` and higher timeout `SCHEDULER_TIMEOUT_MS` for the first model scheduling. It will give priority to the scheduling of the second model. We also set `HAILO_SCHEDULER_PRIORITY_NORMAL+1` priority for the first model scheduling. The practical meaning is that the first network will be ready to run only if `SCHEDULER_THRESHOLD` send requests have been accumulated, or more than `SCHEDULER_TIMEOUT_MS` time has passed and at least one send request has been accumulated. However when both the first and the second networks are ready to run, the first network will be preferred over the second network.

Used APIs: `hailort::ConfiguredNetworkGroup::set_scheduler_timeout()`, `hailort::ConfiguredNetworkGroup::set_scheduler_threshold()` and `hailort::ConfiguredNetworkGroup::set_scheduler_priority()`.

Build vstreams (for each network group) The next step is creating the input and output virtual streams for each configured network group. We create the vstreams using the function `hailort::VStreamsBuilder::create_vstreams()`.

Used APIs: `hailort::VStreamsBuilder::create_vstreams()`

Inference In this example, we create a new thread for each input vstream using the function create_write_threads. Each created thread sends data to the device using the function write_all.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device, using the function `hailort::InputVStream::get_frame_size()`. The main loop of this function sends the allocated frame to the device.

Used APIs: `hailort::InputVStream::get_frame_size()` and `hailort::InputVStream::write()`.

We also create a new thread for each output vstream using the function create_read_threads. Each created thread receives data from the device using the function read_all.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the device, using the function `hailort::OutputVStream::get_frame_size()`. The main loop of this function read frames from the device.

Used APIs: `hailort::OutputVStream::get_frame_size()` and `hailort::OutputVStream::read()`.

After joining the read and write threads, we iterate over all returned results to validate success.

5.2.6. User controlled switch between multiple HEFs using virtual streams – *switch_network_groups_manually_example*

See: `hailort/libhailort/examples/cpp/switch_network_groups_manually_example.cpp`

Summary

- Demonstrates inference of multiple network groups over a single device.
- The program uses the first Hailo device found.
- Random input data is generated.
- Multiple HEFs are configured into the device.
- For each network group, create virtual streams, send threads and recv threads. those threads will be re-used.
- For each network group, virtual streams are created.
- The main loop runs over several network groups. It activates a single network group each time.
- For each activated network group, The data is sent to the device via input vstreams and received via output vstreams.

See also:

See the *comparison table* between context switching and network groups switching for more information about running multiple models.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, then we set the `hailo_vdevice_params_t::scheduling_algorithm` to be `HAILO_SCHEDULING_ALGORITHM_NONE` as we want to manually switch the network groups. We create the *VDevice*.

Used APIs: `hailort::VDevice::create()`

Pre infer stage

Configuring the vdevice from HEF (for each HEF) The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We configure the vdevice using the VDevice class function `hailort::VDevice::configure()`, and getting a `hailort::ConfiguredNetworkGroupVector` object. We store all the configured network groups in a vector.

Used APIs: `hailort::Hef::create()` and `hailort::VDevice::configure()`.

SyncObject SyncObject is a simple barrier-like synchronization mechanism, which allows re-usage of the same threads over several iterations. We create SyncObject per network group, initializing it with the number of vstreams (threads) in that network group. After activating a network group from main thread, we wait for its I/O threads to finish the iteration. when I/O threads finishes the iteration, they signal the main thread to 'switch' network group.

Build vstreams - main thread per network group The next step is creating the virtual streams and I/O threads. We create the virtual using the function `hailort::VStreamsBuilder::create_vstreams()`. In this example, we use *std::thread* per network group. in this thread we create the virtual streams and I/O threads. We create the virtual streams using the function `hailort::VStreamsBuilder::create_vstreams()`. We also create sub-thread for each output vstream responsible for receiving data from the device calling the read_all() function, and another sub-thread for each input vstream calling the write_all() function for sending data to the device.

Used APIs: `hailort::VStreamsBuilder::create_vstreams()`

Infer stage This stage is done multiple times for each HEF loaded into the device, to simulate multiple network group changes.

Activating a network group Before starting inference, we need to activate the network group. We iterate over all configured network groups, activating one at a time. After the activation, we wait on its matching synchronization object until I/O threads will finish the iteration.

Used APIs: `hailort::ConfiguredNetworkGroup::activate()`.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device. The main loop of this function check first that its network group is activated, using the function `hailort::ConfiguredNetworkGroup::wait_for_activation()`. After making sure the network group is activated, write frames to the device. When finishes, signal the synchronization object and wait for next iteration.

Used APIs: `hailort::InputVStream::get_frame_size()`, `hailort::ConfiguredNetworkGroup::wait_for_activation()` and `hailort::InputVStream::write()`.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the device. The main loop of this function check first that its network group is activated, using the function `hailort::ConfiguredNetworkGroup::wait_for_activation()`. After making sure the network group is activated, read frames from the device. When finishes, signal the synchronization object and wait for next iteration.

Used APIs: `hailort::OutputVStream::get_frame_size()`, `hailort::ConfiguredNetworkGroup::wait_for_activation()` and `hailort::OutputVStream::read()`.

5.2.7. Inference using raw streams – *raw_streams_example*

See: `hailort/libhailort/examples/cpp/raw_streams_example.cpp`

Summary

- Demonstrates basic inference of a shortcut network using the raw stream C++ API.
- The program uses the Hailo device.
- The program infers hefs with multiple input streams and multiple output streams.
- The program creates a thread for each input stream and a thread for each output stream.
- The data is transformed using the transformation API when sent / received to / from the device in the same thread that sends / receives the data to/from the device.

Code structure

Device initialization

We use the first device found on the system using the function `hailort::Device::create()`.

Used APIs: `hailort::Device::create()`.

Configuring the device from HEF The next step is to create a `hailort::Hef`, and to use it to configure the device for inference. We init a `hailort::NetworkGroupsParamsMap` with default values, configure the device using the Device class function `hailort::Device::configure()` and gets a `hailort::ConfiguredNetworkGroupVector` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::Device::configure()`.

Get Input/Output streams

The next step we get the input streams and output streams from the network group.

Used APIs: `hailort::ConfiguredNetworkGroup::get_input_streams()` and `hailort::ConfiguredNetworkGroup::get_output_streams()`.

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailort::ConfiguredNetworkGroup::activate()`.

Inference In this example, we use *std::thread* for inference, with a sub-thread for each output thread receiving data from the device calling the read_all() function, and another sub-thread for each input thread calling the write_all() function for sending data to the device.

Read all The read_all() thread, will firstly initialize any necessary resources, including:

- *transform_context* a `hailort::OutputTransformContext` - used to transform the data from HW format to host.
- *hw_data* - The actual data received from the hw.
- *host_data* - Will contain the actual user data (The network result).

Used APIs: `hailort::OutputTransformContext::create()`,

After initialization, the read thread will recv all frames:

- First, reading data from the device.
- Then transform it from HW format to host format.

Used APIs: `hailort::OutputStream::read()` and `hailort::OutputTransformContext::transform()`.

Write all The write_all() function, will firstly create and initialize any necessary resources, including:

- *transform_context* – a `hailort::InputTransformContext` - used to transform the data from host format to HW.
- *host_data* – Will contain the actual user data (for example - frame).
- *hw_data* – The actual data sent to the hw.

Used APIs: `hailort::InputTransformContext::create()`,

After initialization, the write_all thread will send all frames:

- First, transform the buffer from host format to HW format
- Then writing the HW buffer to the device.

Used APIs: `hailort::InputStream::write()` and `hailort::InputTransformContext::transform()`.

5.2.8. Async Inference Using Raw Streams With Single Thread – *raw_async_streams_single_thread_example*

See: `hailort/libhailort/examples/cpp/raw_async_streams_single_thread_example.cpp`

Summary

- Demonstrates basic async inference of a shortcut network using raw async stream API.
- Using a single thread for the inference.
- The program uses the first Hailo device found.
- The data sent/received does not undergo preprocessing/postprocessing. For a complete raw stream example with preprocessing/postprocessing, see *raw_streams_example*

Code Structure

Device Initialization

We use the first device found on the system using the function `hailort::Device::create()`.

Used APIs: `hailort::Device::create()`.

Configuring The Device From HEF The next step is to create a `hailort::Hef` and to use it to configure the device for inference. We initialize a `hailort::NetworkGroupsParamsMap` with default values. To support async streams API the `HAILO_STREAM_FLAGS_ASYNC` flag is set inside `hailo_stream_parameters_t`. We configure the device using the `hailort::Device::configure()` function which returns a `hailort::ConfiguredNetworkGroup` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::Device::configure()`.

Async Inference

- The network group can now be activated.
- For the output stream, `hailort::OutputStream::get_async_max_queue_size()` async read requests are launched. Launching more async read requests may fill the stream's queue, resulting in subsequent async read operations failing with `HAILO_QUEUE_IS_FULL`.
  - First we allocate memory for each transfer, ensuring it is aligned to the system page size, as required by `hailort::OutputStream::read_async()`.
  - The async read operation is initiated by calling `hailort::OutputStream::read_async()`, passing the allocated buffer.
  - Once a read operation has finished, the associated async callback function is triggered. In this example, the callback function examines the `hailort::OutputStream::CompletionInfo::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async read request is initiated using the same buffer and callback.

    Real applications may pass the results obtained by the read operations for post-processing or display.
- For the input stream, `hailort::InputStream::get_async_max_queue_size()` async write requests are launched. Launching more async write requests may fill the stream's queue, resulting in subsequent async write operations failing with `HAILO_QUEUE_IS_FULL`.
  - First we allocate memory for each transfer, ensuring it is aligned to the system page size, as required by `hailort::InputStream::write_async()`.

  - The async write operation is initiated by calling `hailort::InputStream::write_async()`, passing the allocated buffer.
  - Once a write operation has finished, the associated async callback function is triggered. In this example, the callback function examines the `hailort::InputStream::CompletionInfo::status` that is passed to it. If the status is `HAILO_SUCCESS`, indicating a successful transfer, a new async write request is initiated using the same buffer and callback.

    Real applications may choose to fill the buffer with a new frame, free it, or return it to some buffer pool for later use.

- Since further async operations are launched within each callback, the inference is executed in parallel to the main thread. In this example, the main thread remains idle and sleeps for 5 seconds, allowing the inference to run in the background.
- To stop the inference process, the network group must be deactivated. It is done automatically by the destructor of cpp:type:*hailort::ActivatedNetworkGroup*. Any transfers that have not been completed will be called with the status `HAILO_STREAM_ABORTED_BY_USER`.

Used APIs: `hailort::OutputStream::read_async()`, `hailort::InputStream::write_async()`, and `hailort::ConfiguredNetworkGroup::activate()`.

Note: It is important to ensure that the buffers passed to the inference and any variables captured in the async operations callback remain valid until the network group becomes inactive. Failure to do so may result in the callback being called with invalid variables, leading to undefined behavior.

In the provided example, although no variables are captured, the buffers are reused. To guarantee their validity, it is crucial to follow the C++ destruction order. We accomplish this by defining the *buffer_guards* variable **before** creating the *activated_network_group*. By doing so, the buffers will remain valid throughout the execution of the network group, ensuring correct and predictable behavior.

5.2.9. Async Inference Using Raw Streams With Multiple Threads – *raw_async_streams_multi_thread_example*

See: `hailort/libhailort/examples/cpp/raw_async_streams_multi_thread_example.cpp`

Summary

- Demonstrates basic async inference of a shortcut network using raw async stream API.
- Using thread for each stream.
- The program uses the first Hailo device found.
- The data sent/received does not undergo preprocessing/postprocessing. For a complete raw stream example with preprocessing/postprocessing, see *raw_streams_example*

Code Structure

Device Initialization

We use the first device found on the system using the function `hailort::Device::create()`.

Used APIs: `hailort::Device::create()`.

Configuring The Device From HEF The next step is to create a `hailort::Hef` and to use it to configure the device for inference. We initialize a `hailort::NetworkGroupsParamsMap` with default values. To support async streams API the `HAILO_STREAM_FLAGS_ASYNC` flag is set inside `hailo_stream_parameters_t`. We configure the device using the `hailort::Device::configure()` function which returns a `hailort::ConfiguredNetworkGroup` object.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::Device::configure()`.

Async Inference

- First, we allocate memory for each stream, ensuring it is aligned to the system page size, as required by the `hailort::OutputStream::read_async()` and `hailort::InputStream::write_async()`. For simplicity, in this example, for each stream we maintain a single buffer that it is used for all transfers. In real applications, it may be problematic in output since the buffer will be overridden on each read.
- We activate the network group. It is important to activate the network group **after** allocating the buffers to ensure that the buffers remain valid until the network group becomes deactivated.
- We create a thread for the output stream that continuously initiate async read operations. The thread will:
  - Wait until the `hailort::OutputStream` is ready to initiate a new async read operation by calling `hailort::OutputStream::wait_for_async_ready()`.
  - Then, it will initiate the async operation by calling `hailort::OutputStream::read_async()`. The callback pass to the async operation will only examine the `hailort::OutputStream::CompletionInfo::status` and print on failure.

    Real applications may pass the results obtained by the read operations for post-processing or display.
  - The loop ends when the network becomes deactivated. All pending read will be completed with the status `HAILO_STREAM_ABORTED_BY_USER`. Either `hailort::OutputStream::wait_for_async_ready()` or `hailort::OutputStream::read_async()` will return `HAILO_STREAM_NOT_ACTIVATED`.
- We create a thread for the input stream that continuously initiate async write operations. The thread will:
  - Wait until the :cpp:type::*hailort::InputSteam* is ready to initiate a new async write operation by calling `hailort::InputSteam::wait_for_async_ready()`.
  - Then, it will initiate the async operation by calling `hailort::InputSteam::write_async()`. The callback passed to the async operation will only examine the `hailort::InputSteam::CompletionInfo::status` and print error message on failure.

    Real applications may choose to fill the buffer with a new frame, free it, or return it to some buffer pool for later use.
  - The loop ends when the network becomes deactivated. All pending read will be completed with the status `HAILO_STREAM_ABORTED_BY_USER`. Either `hailort::InputSteam::wait_for_async_ready()` or `hailort::InputSteam::write_async()` will return `HAILO_STREAM_NOT_ACTIVATED`.
- The inference is executed in parallel to the main thread. In this example, the main thread remains idle and sleeps for 5 seconds, allowing the inference to run in the background.

- To stop the inference process, the network group must be deactivated. It is done explicitly by calling the destructor of `hailort::ActivatedNetworkGroup`. Any transfers that have not been completed will be called with the status `HAILO_STREAM_ABORTED_BY_USER`.

Used APIs: `hailort::ConfiguredNetworkGroup::activate()`, `hailort::OutputStream::wait_for_async_ready()`, `hailort::OutputStream::read_async()`, `hailort::InputStream::wait_for_async_ready()` and `hailort::InputStream::write_async()`.

Note: It is important to ensure that the buffers passed to the inference and any variables captured in the async operations callback remain valid until the network group becomes inactive. Failure to do so may result in the callback being called with invalid variables, leading to undefined behavior.

In the provided example, although no variables are captured, the buffers must stay alive until the inference ends. To guarantee their validity, we deactivate the network group before releasing the buffers - either explicitly by calling *activated_network_group->reset()* or implicitly on the destructor (can happen on an unexpected exception or function return).

5.2.10. Multi-process inference using HailoRT multi-process service – *multi_process_example*

See: `hailort/libhailort/examples/cpp/multi_process_example.cpp` and `hailort/libhailort/examples/cpp/multi_process_example.sh -h`

Note: To use *Multi-Process service* with libhailort when compiling from sources, use the compilation flag `HAILO_BUILD_SERVICE`.

Summary

- Demonstrates how to work with HailoRT multi-process service over a single device, using VDevice and Round-Robin scheduling algorithm for network group switching.
- The program uses the Hailo vdevice with *multi_process_service* flag.
- For each network group, the program creates a thread for each input vstream and a thread for each output vstream.
- The program waits for all threads to finish, and validate their results.

Prerequisites to run the example

In order to run the example, one should first enable and start the HailoRT service as follow:

- To enable and start the service, run: `sudo systemctl enable --now hailort.service`
- If service is enabled but not active, run: `sudo systemctl start hailort.service`
- To see the service status and make sure it is active, run: `sudo systemctl status hailort.service`

Running the multi-process example

Options for running with multiple processes:

1) **From the examples directory, in each terminal run the following command:** `` `./build/cpp/cpp_multi_process_example <hef_path>` ``
2) **Using the script** `hailort/libhailort/examples/cpp/multi_process_example.sh`**:** The script is running the example in multiple processes, its default is to run two processes:
   1) Inference on shortcut network hef
   2) Inference on multi shortcut network hef

   In the script options, one can specify how many processes to run inference on each hef. See `hailort/libhailort/examples/cpp/multi_process_example.sh -h` for more information.

Code structure

Device initialization We initialize the `hailo_vdevice_params_t` with default values, and then we create the *VDevice*. We set the `hailo_vdevice_params_t::scheduling_algorithm` to be `HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN` and `hailo_vdevice_params_t::multi_process_service` to be `true`.

Used APIs: `hailort::VDevice::create()`

Pre infer stage

Configuring the vdevice from HEF The next step is to create a `hailort::Hef`, and to use it to configure the vdevice for inference. We configure the vdevice using the pcie Device class function `hailort::VDevice::configure()`, and getting a `hailort::ConfiguredNetworkGroupVector` object. We store all the configured network groups in a vector.

Build VStreams The next step is creating the virtual streams. We create the virtual streams from the network group using the function `hailort::VStreamsBuilder::create_vstreams()`.

Used APIs: `hailort::VStreamsBuilder::create_vstreams()`

Inference In this example, we use *std::thread* for inference, with a sub-thread for each output thread receiving data from the device calling the read_all() function, and another sub-thread for each input thread calling the write_all() function for sending data to the device.

Read all The read_all() function, will firstly create and initialize a vector of data to recv data from the device. Then the read_all thread will receive all of the frames from the device.

Used APIs: `hailort::OutputVStream::read()`.

Write all The write_all() function, will firstly create and initialize a vector of data to send to the device. Then the write_all thread will send all of the frames to the device.

Used APIs: `hailort::InputVStream::write()`.

After joining the read and write threads, we iterate over all returned results to validate success.

5.2.11. Inference using virtual streams – *infer_pipeline_example*

See: `hailort/libhailort/examples/cpp/infer_pipeline_example.cpp`

Summary

- Demonstrates basic inference of a shortcut network (i.e inputs are sent through the device and right back out, without any changes made to the data).
- The program uses the Hailo Ethernet device.
- The data is both sent to the device and received using the inference pipeline.

Code structure

Device initialization We open the Hailo Ethernet device.

Used APIs: `hailort::Device::create_eth()`

Configuring the device from HEF The next step is to create a `hailort::Hef`, and to use it to configure the device for inference. We init a `hailort::NetworkGroupsParamsMap` with default values, configure the device using the Ethernet Device class function `hailort::Device::configure()` and get a `hailort::ConfiguredNetworkGroupVector` object.

Note - It is possible to limit the rate in which data is sent using a rate limiter: One way of doing that is by using the command line tool *hailortcli udp-rate-limiter*. Another way is creating a `NetworkUdpRateCalculator` object with `NetworkUdpRateCalculator::create()`, then calling `NetworkUdpRateCalculator::calculate_inputs_bandwith()`, which returns the input bandwidth that each stream should be limited to. These rates should then be assigned to their respective fields in `hailo_eth_input_stream_params_t::rate_limit_bytes_per_sec` under `hailo_configure_params_t`. See also `hailo_eth_input_stream_params_t`.

Used APIs: `hailort::Hef::create()`, `hailort::Hef::create_configure_params()` **and** `hailort::Device::configure()`.

Creating VStream params and Vstream infos The next step is creating the virtual stream params. We create the virtual stream params from the network group using the functions `hailort::ConfiguredNetworkGroup::make_input_vstream_params()` and `hailort::ConfiguredNetworkGroup::make_output_vstream_params()`.

Used APIs: `hailort::ConfiguredNetworkGroup::make_input_vstream_params()` **and** `hailort::ConfiguredNetworkGroup::make_output_vstream_params()`

Activating a network group Before starting inference, we need to activate the network group.

Used APIs: `hailort::ConfiguredNetworkGroup::activate()`.

Creating a pipeline

We can now create the inference pipeline object using `hailort::InferVStreams::create()`.

Used APIs: `hailort::InferVStreams::create()`.

Prepareing buffers before inference We need to prepare the input data buffers, and also the buffers for the inference output.

Used APIs: `hailort::InferVStreams::get_input_vstreams()`, :`hailort::InputVStream::get_f`
`hailort::InferVStreams::get_output_vstreams()`, :`hailort::OutputVStream::get_frame_size()`,

Inference Finally, we run inference on the input data using the inference pipeline.

Used APIs: and `hailort::InferVStreams::infer()`.

5.2.12. Power measurement – *power_measurement_example*

See: `hailort/libhailort/examples/cpp/power_measurement_example.cpp`

Note: This example is not supported on Hailo-15.

Summary

- Demonstrates how to perform a continuous power measurement on the chip.
- The program uses a VDevice created from all the Hailo devices that are connected to the computer.
- In this tutorial we use the following arguments to control the measurement type:
  - *power* – measure power consumption in W.
  - *current* – measure current in mA.

Note: For instantaneous power measurement, see `hailort::Device::power_measurement()`.

For instantaneous temperature measurement, see `hailort::Device::get_chip_temperature()`.

Code structure

VDevice initialization First we scan for PCIe devices, then we initialize the `hailo_vdevice_params_t` with the device count we scanned before. The next step is to create the *VDevice*.

Used APIs: `hailort::VDevice::create()`, `hailo_init_vdevice_params()` `hailort::Device::scan_pcie()`.

Power measurement

Initialization For sending controls to a `hailort::Device`, we need to get the underlying physical devices from the `hailort::VDevice`, using the function `hailort::VDevice::get_physical_devices()`. For each physical device, we first call the function `hailort::Device::stop_power_measurement()` to make sure there aren't any former measurements currently running. The next step is to set power measurement arguments using the function `hailort::Device::set_power_measurement()`, and to start the measurement using the function `hailort::Device::start_power_measurement()`

Used APIs: `hailort::VDevice::get_physical_devices()`, `hailort::Device::stop_power_measurement()`, `hailort::Device::set_power_measurement()` and `hailort::Device::start_power_measurement()`.

Stopping measurement and fetching the results After starting the measurement, we sleep for a constant period of time (5 seconds). Once we finish sleeping, we go over all physical devices and do the following:

- Stopping the current measurement using the function `hailort::Device::stop_power_measurement()`.
- Getting the measurement results using the function `hailort::Device::get_power_measurement()`.
- Printing measurement results.

Used APIs: `hailort::Device::stop_power_measurement()`, `hailort::Device::get_power_measurement()`.

5.2.13. Notification Callback – *notification_callback_example*

See: `hailort/libhailort/examples/cpp/notification_callback_example.cpp`

Summary

- Demonstrates the basic usage of notification callbacks.
- The program creates a device and then sets and removes a notification callback on it.

Code structure

Device initialization

First we scan for Hailo devices using `hailort::Device::scan()`, then we use the first device_id we found for creating the device, using the function `hailort::Device::create()`.

Used APIs: `hailort::Device::create()`, `hailort::Device::scan()`.

Set the callback notification

By calling `Device::set_notification_callback()` with a lambda function that prints an overcurrent alaram message, and the following notification id :`hailo_notification_id_t::HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM`, the callback function will be called when we will get this notification.

Used APIs: `Device::set_notification_callback()`.

Remove the callback notification

By calling `Device::remove_notification_callback()` with the same `hailo_notification_id_t` as in the `Device::set_notification_callback()`, we remove this notification callback.

Used APIs: `Device::remove_notification_callback()`.

5.3. Python inference tutorial

This tutorial will walk you through the inference process.

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

It is recommended to use the command `hailo tutorial` (when inside the virtualenv) to open a Jupyter server that contains the tutorials.

5.3.1. Standalone hardware deployment

The standalone flow allows direct access to the HW, developing applications directly on top of Hailo core HW, using HailoRT. This way we can use the Hailo hardware without Tensorflow, and even without the Hailo SDK (after the HEF is built).

An HEF is Hailo's binary format for neural networks. The HEF files contain:

- Target HW configuration
- Weights
- Metadata for HailoRT (e.g. input/output scaling)

First create the desired target object. In our example we use the Hailo-8 PCIe interface:

```
[ ]: import numpy as np
from multiprocessing import Process
from hailo_platform import (HEF, VDevice, HailoStreamInterface, InferVStreams,
↪ConfigureParams,
  InputVStreamParams, OutputVStreamParams, InputVStreams, OutputVStreams,
↪FormatType)

# The target can be used as a context manager ("with" statement) to ensure it's released
↪on time.
# Here it's avoided for the sake of simplicity
target = VDevice()

# Loading compiled HEFs to device:
model_name = 'resnet_v1_18'
hef_path = '../hefs/{}.hef'.format(model_name)
hef = HEF(hef_path)

# Configure network groups
configure_params = ConfigureParams.create_from_hef(hef=hef,
↪interface=HailoStreamInterface.PCIe)
network_groups = target.configure(hef, configure_params)
network_group = network_groups[0]
network_group_params = network_group.create_params()

# Create input and output virtual streams params
# Quantized argument signifies whether or not the incoming data is already quantized.
# Data is quantized by HailoRT if and only if quantized == False .
input_vstreams_params = InputVStreamParams.make(network_group, quantized=False,
↪format_type=FormatType.FLOAT32)
output_vstreams_params = OutputVStreamParams.make(network_group, quantized=True,
↪format_type=FormatType.UINT8)

# Define dataset params
input_vstream_info = hef.get_input_vstream_infos()[0]
output_vstream_info = hef.get_output_vstream_infos()[0]
image_height, image_width, channels = input_vstream_info.shape
num_of_images = 10
low, high = 2, 20

# Generate random dataset
dataset = np.random.randint(low, high, (num_of_images, image_height, image_width,
↪channels)).astype(np.float32)
```

Running hardware inference

Infer the model and then display the output shape:

```
[ ]: # Infer
with InferVStreams(network_group, input_vstreams_params, output_vstreams_params)
↪as infer_pipeline:
  input_data = {input_vstream_info.name: dataset}
  with network_group.activate(network_group_params):
    infer_results = infer_pipeline.infer(input_data)
    print('Stream output shape is {}'.format(infer_results[output_vstream_info.
↪name].shape))
```


5.3.2. Streaming inference

This section shows how to run streaming inference using multiple processes in Python.

We will not use infer. Instead we will use a send and receive model. The send function and the receive function will run in different processes.

Define the send and receive functions:

```
[ ]: def send(configured_network, num_frames):
       configured_network.wait_for_activation(1000)
       vstreams_params = InputVStreamParams.make(configured_network)
       with InputVStreams(configured_network, vstreams_params) as vstreams:
         vstream_to_buffer = {vstream: np.ndarray([1] + list(vstream.shape),
     ↪dtype=vstream.dtype) for vstream in vstreams}
         for _ in range(num_frames):
           for vstream, buff in vstream_to_buffer.items():
             vstream.send(buff)

     def recv(configured_network, vstreams_params, num_frames):
       configured_network.wait_for_activation(1000)
       with OutputVStreams(configured_network, vstreams_params) as vstreams:
         for _ in range(num_frames):
           for vstream in vstreams:
             data = vstream.recv()

     def recv_all(configured_network, num_frames):
       vstreams_params_groups = OutputVStreamParams.make_groups(configured_network)
       recv_procs = []
       for vstreams_params in vstreams_params_groups:
         proc = Process(target=recv, args=(configured_network, vstreams_params, num_
     ↪frames))
         proc.start()
         recv_procs.append(proc)
       for proc in recv_procs:
         proc.join()
```

Define the amount of frames to stream, define the processes, create the target and run processes:

```
[ ]: # Define the amount of frames to stream
     num_of_frames = 1000

     send_process = Process(target=send, args=(network_group, num_of_frames))
     recv_process = Process(target=recv_all, args=(network_group, num_of_frames))
     recv_process.start()
     send_process.start()
     print('Starting streaming (hef=\'{}\', num_of_frames={})'.format(model_name, num_
     ↪of_frames))
     with network_group.activate(network_group_params):
       send_process.join()
       recv_process.join()
     print('Done')
```


5.4. Python power measurement tutorial

This tutorial will show how to perform a power measurement on the chip.

The Hailo chip supports power measurement which is done via the control protocol.

Requirements:

- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

Attention:

- These examples should run in a different process than the one that performs the actual inference.

It is recommended to use the command `hailo tutorial` (when inside the virtualenv) to open a Jupyter server that contains the tutorials.

5.4.1. Single power measurement

```
[ ]: %matplotlib inline
import time

from hailo_platform import Device, DvmTypes, PowerMeasurementTypes, SamplingPeriod,
↪AveragingFactor, MeasurementBufferIndex # noqa F401
```

Initialize the hardware object:

```
[ ]: target = Device()
```

When using the `power_measurement()` function with no parameters, the function tries to detect which board is connected (evaluation board, M.2 or mPCIe) and determine the DVM accordingly (at the moment only the mentioned boards are supported).

The parameter `dvm` (of type `DvmTypes`) defines which DVM will be measured. The user can choose a specific DVM or choose the default DVM. The meaning of the default DVM changes according to the board or module in use.

The default for the evaluation board is the sum of three DVMs: `DvmTypes.VDD_CORE`, `DvmTypes.MIPI_AVDD` and `DvmTypes.AVDD_H`. The sum of these three DVMs approximates of the total power consumption of the chip in PCIe setups. Only power can be measured using this default option, as voltage and current can't be summed up this way.

The default for platforms supporting current monitoring, such as M.2 and mPCIe modules, is `DvmTypes.OVERCURRENT_PROTECTION`, which measures the power consumption of the whole module.

See the API documentation for further details about the supported DVMs and measurement types.

```
[ ]: single_power_measurement = target.control.power_measurement()
print('Power from single measurement: {} W'.format(single_power_measurement))
```


5.4.2. Periodic power measurement

In the following example, a periodic power measurement is taken.

A measurement index is selected. It is a member of the enum class MeasurementBufferIndex (between 0 and 3). The DVM is not given so the default DVM will be used, as explained above.

```
[ ]: buffer_index = MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0
target.control.set_power_measurement(buffer_index=buffer_index)
```

The following call to `start_power_measurement()` allows to configure the power sampling frequency. In this case we keep the default configuration. The API documentation provides more details.

```
[ ]: target.control.start_power_measurement()
```

Clear old samples and statistics (min, max, average) each time the measurement is taken from the chip.

```
[ ]: should_clear = True
```

The power measurement is read every second, for 10 seconds.

The firmware calculates the min, max, and average of all the values from the sensor. Note that the average calculated by the firmware is the "average of averages". This is the average of all values that have already been averaged by the sensor. The host then requests these values from the firmware every second by calling the `get_power_measurement()` function. The host can also read the average time interval between new sensor values.

```
[ ]: for _ in range(10):
    time.sleep(1)
    # Get saved power measurement values from the firmware.
    measurements = target.control.get_power_measurement(buffer_index=buffer_index,
↪should_clear=should_clear)
    print('Average power is {} W. Min power is {} W. Max power is {} W.\nAverage time
↪between power samples is {} mS\n'.format(measurements.average_value, measurements.
↪min_value, measurements.max_value, measurements.average_time_value_
↪milliseconds))

# Stop performing periodic power measurement
target.control.stop_power_measurement()
```

5.5. Python inference tutorial - Multi Process Service and Model Scheduler

This tutorial will walk you through the inference process using The Model Scheduler.

Requirements:

- Run HailoRT Multi-Process Service before running inference. See installation steps in *Multi-Process Service*
- Run the notebook inside the Python virtual environment: `source hailo_virtualenv/bin/activate`

It is recommended to use the command `hailo tutorial` (when inside the virtualenv) to open a Jupyter server that contains the tutorials.

5.5.1. Running Inference using HailoRT

In this example we will use the Model Scheduler to run inference on multiple models. Each model is represented by an HEF which is built using the Hailo Dataflow Compiler. An HEF is Hailo's binary format for neural networks. The HEF files contain:

- Target HW configuration
- Weights
- Metadata for HailoRT (e.g. input/output scaling)

The Model Scheduler is an HailoRT component that comes to enhance and simplify the usage of the same Hailo device by multiple networks. The responsibility for activating/deactivating the network groups is now under HailoRT, and done **automatically** without user application intervention. In order to use the Model Scheduler, create the VDevice with scheduler enabled, configure all models to the device, and start inference on all models:

[ ]:
```
import numpy as np
from multiprocessing import Process
from hailo_platform import (HEF, VDevice, HailoStreamInterface, InferVStreams,
↪ConfigureParams,
  InputVStreamParams, OutputVStreamParams, InputVStreams, OutputVStreams,
↪FormatType, HailoSchedulingAlgorithm)


# Define the function to run inference on the model
def infer(network_group, input_vstreams_params, output_vstreams_params, input_
↪data):
  rep_count = 100
  with InferVStreams(network_group, input_vstreams_params, output_vstreams_
↪params) as infer_pipeline:
    for i in range(rep_count):
      infer_results = infer_pipeline.infer(input_data)


# Loading compiled HEFs:
first_hef_path = '../hefs/resnet_v1_18.hef'
second_hef_path = '../hefs/shortcut_net.hef'
first_hef = HEF(first_hef_path)
second_hef = HEF(second_hef_path)
hefs = [first_hef, second_hef]

# Creating the VDevice target with scheduler enabled
params = VDevice.create_params()
params.scheduling_algorithm = HailoSchedulingAlgorithm.ROUND_ROBIN
with VDevice(params) as target:
  infer_processes = []

  # Configure network groups
  for hef in hefs:
    configure_params = ConfigureParams.create_from_hef(hef=hef,
↪interface=HailoStreamInterface.PCIe)
    network_groups = target.configure(hef, configure_params)
    network_group = network_groups[0]

    # Create input and output virtual streams params
    # Quantized argument signifies whether or not the incoming data is already
↪quantized.
    # Data is quantized by HailoRT if and only if quantized == False.
    input_vstreams_params = InputVStreamParams.make(network_group,
↪quantized=False, format_type=FormatType.FLOAT32)
    output_vstreams_params = OutputVStreamParams.make(network_group,
↪quantized=True, format_type=FormatType.UINT8)
```

(continues on next page)

(continued from previous page)

```
    # Define dataset params
    input_vstream_info = hef.get_input_vstream_infos()[0]
    image_height, image_width, channels = input_vstream_info.shape
    num_of_frames = 10
    low, high = 2, 20

    # Generate random dataset
    dataset = np.random.randint(low, high, (num_of_frames, image_height, image_
↪width, channels)).astype(np.float32)
    input_data = {input_vstream_info.name: dataset}

    # Create infer process
    infer_process = Process(target=infer, args=(network_group, input_vstreams_
↪params, output_vstreams_params, input_data))
    infer_processes.append(infer_process)

  print(f'Starting streaming on multiple models using scheduler')
  for infer_process in infer_processes:
    infer_process.start()
  for infer_process in infer_processes:
    infer_process.join()

  print('Done inference')
```


6. Running Inference

The following section covers several use cases of the inference process. Inference is the process in which the host sends the data to the Hailo device (and its NN Core) and receives the output from it.

6.1. Inference Stages

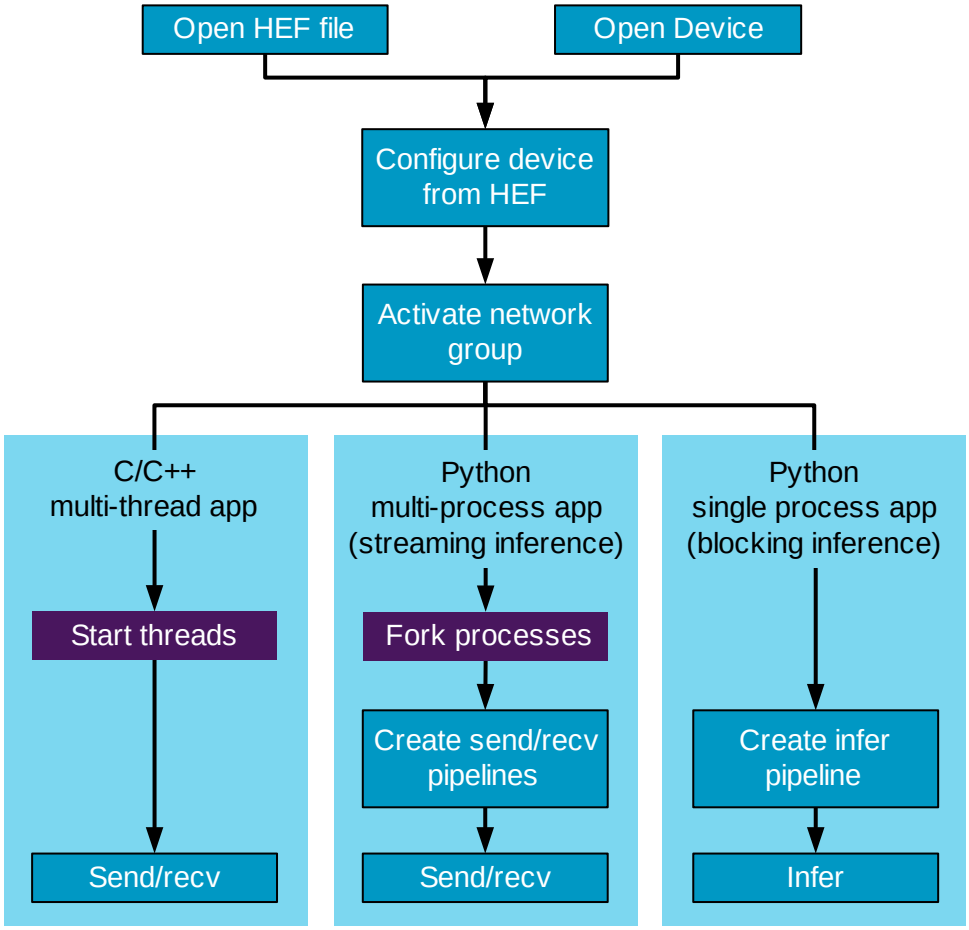

Figure 7. Illustration of an inference flow with the Hailo-8 device. The first stages are common to all application architectures. The last stages, with the light blue background, are architecture dependent. Three architectures are shown: C/C++ app, multi-process Python app, and single process Python app.

6.1.1. Preparation

In order to start working with the device, the user opens the device and the HEF file (or files). The user then configures the device from the HEF and activates one of the network groups.

Note: In most simple use cases, there is a single network group. When there are multiple network groups, for example when multiple HEFs are loaded, the user can choose which one to load.

6.1.2. Sending and Receiving Data

Once the HEF files are downloaded to the Hailo-8 chip and a network group is activated, the device is ready to start running data. To start running data the following steps are required:

Pre-Inference Prepare the data to be sent to the chip (For example quantizing the images).

Send Send the input data to the Hailo-8 chip.

Receive Receive the output data from the Hailo-8 chip.

Post-Inference Ensure the data is returned in the format expected by the host.

6.2. Python Inference API

There are two different methods for working with the inference in Python:

Infer (Blocking) A blocking function that wraps all the four steps mentioned.

Send/recv (Streaming) In this case the user has two processes, each one in charge of the two steps mentioned above:

- **Send** - in charge of pre-inference and sending the data from the host to the Hailo-8 chip.
- **Recv** - in charge of receiving the data from the Hailo-8 chip and post-inference.

A single process should be used to receive the outputs from the device, even if there are multiple output tensors.

See also:

The *Python inference tutorial* demonstrates how to run blocking and streaming inference in Python.

6.3. C/C++ Inference API

Note: In this section we explain the common use case which is multi-thread. For multi-process please refer to *Multi-Process Service*.

In C/C++ there is usually no need to open multiple processes. Multiple threads are used instead. We use a thread per input to commence pre-inference and sending the data. We use a thread per output to receive the data from the Hailo-8 chip and post-inference.

Two levels of API are provided in C/C++:

- **Raw streams** – Allow low level interactions with the device. They are used to send and receive data in the same format expected by the hardware. Pre- and post-inference stages are done separately by calling dedicated transformation functions.
- **Virtual streams** (vstreams) – Provide high level interface that is used to interact with the device. The virtual streams manage the data transformation (pre- and post-inference).

See also:

The *C inference tutorial* shows how to run inference in C. It demonstrates how to use both raw streams and virtual streams.

See also:

The *C++ inference tutorial* shows how to run inference in C++. It demonstrates how to use both raw streams and virtual streams.

6.4. Device virtualization

HailoRT supports a Virtual Device - `hailort::VDevice` interface which encapsulates multiple devices into one instance. Controlling the underlying physical-devices of a VDevice is done using `hailo_vdevice_params_t::device_count` and `hailo_vdevice_params_t::device_ids`.

Using the VDevice interface enables other HailoRT features such as *Model Scheduler*, *Multi-Process Service* and *Stream Multiplexer*, and therefore it is recommended to use this interface also with just one physical device.

Note: VDevice is currently supported for PCIe devices only.

6.5. Environment Variables

The HailoRT environment variables are used to enable/disable different features.

Environment Variable name	Default Value	Description
HAILORT_LOGGER_PATH	""	• Not setting this variable, or setting it to an empty string, will set the log file path to the current directory • Setting the value to NONE will disable the log file
HAILO_MONITOR	0	see *Monitor*

6.6. Model Scheduler and Compiling Models Together

HailoRT provides several mechanisms that allow to run multiple models:

- The **Network groups switching using HailoRT Model Scheduler** – The Model scheduler is a HailoRT component that comes to enhance and simplify the usage of the same Hailo device by multiple networks. The responsibility for activating/deactivating the network groups is now under HailoRT, and done **automatically** without user application intervention. In order to use the Model Scheduler, create the VDevice with scheduler enabled, configure all models to the device, and let HailoRT do the rest.

  Example usage - *C example* and *C++ example*.

- The manual **user controlled switching** mechanism which allows the user to switch between several models manually. In order to do it, first both models are configured and the first one is activated. When the switch should take place, the first model is de-activated and the second model is activated.

  Example usage - *C++ example*.

  The following table compares between these two methods:

	Automatic Model Scheduler	Manual Switching Between Models
Purpose	For most multiple models scenarios.	When the app needs full (low-level) control on when to run each model.
How to compile the model?	No special compilation flow. Any HEF can be used.	No special compilation flow. Any HEF can be used.
How to deploy the model?	Create VDevice with the scheduler enabled. Then the switching will be done automatically behind the scenes.	Use the activate() and deactivate() APIs to switch models.

- The **compiling models together** mechanism allows to run multiple models by **automatically** switching between models that were compiled together using the DFC join() API, which constitute the full model.

6.7. Model Scheduler

The **Model Scheduler**'s behavior and performance is controlled using the following parameters:

- Configured network group **batch-size** affects the size of the networks group's input and output queues. The larger the batch-size parameter, the larger the queues. For multi context models the the actual infer batch will be up to the batch-size.

  To modify this parameter, change the `batch_size` memeber of `hailo_configure_network_group_params_t`.

- Model scheduler **Scheduling Scheme** determines how the next network to be activated is selected. Currently, the only scheme is Round-Robin, which maintains fairness across different networks so each network in turn will be checked to be identified as ready for activation

- Model Scheduler **threshold** is the minimum required number of inference requests, before the network group is eligible to be scheduled (unless the timeout has passed). Default is 0. Once the scheduler switches to the new model, it will infer all available frames in its queues, even if they are more than the threshold.

  To modify this parameter, use `hailo_set_scheduler_threshold()` (C) or `hailort::ConfiguredNetworkGroup::set_scheduler_threshold()` (C++).

- Model Scheduler **timeout** is the maximum time period [ms] that may pass before a configured network group is scheduled. This counter is reset when the network group is activated, and at least one frame is sent to it. Default is 0ms.

  To modify this parameter, use `hailo_set_scheduler_timeout()` (C) or `hailort::ConfiguredNetworkGroup::set_scheduler_timeout()` (C++).

- Model Scheduler **priority** determines the relative importance of the network. When the scheduler comes to select the next network to run it first looks at the the priority of the network. That means that, the scheduler will first evaluate all the networks that have the same highest priority, and if non can be selected then it will move to consider lower priority networks. By default, the priority level is set to HAILO_SCHEDULER_PRIORITY_NORMAL.

  To modify this parameter, use `hailo_set_scheduler_priority()` (C) or `hailort::ConfiguredNetworkGroup::set_scheduler_priority()` (C++).

Note: The Model Scheduler default behavior is using round-robin scheduling scheme with default values of threshold (0) and timeout (0ms). These defaults will schedule each network as soon as it receives a single frame.

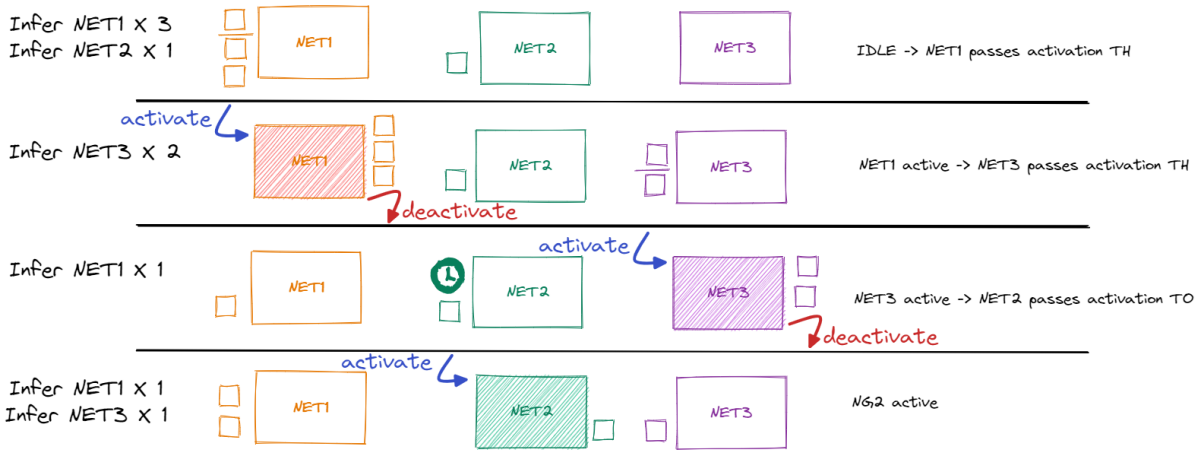

Figure 8. Model Scheduler Example

6.8. Model Scheduler Optimizations

When examining the scheduler behavior and performance we always need to keep in mind that it is all about tradeoffs: **Memory vs Performance** and **FPS vs Latency**.

The scheduler monitors the fullness of each network's low-level input and output queues (these queues are the buffer queues that reside just before and after the HW) and determine if the associated network needs and can be activated. The scheduler also monitors how much time has passed from the last time it was activated. The network is considered ready for activation if one of its activation criteria is reached. The network is considered able for activation if it has enough free space in its output queues.

Batch-size: The first step that will improve performance and make the other configurations more effective is to make the input and output queues larger by increasing the batch-size. Large queues improve performance by performing any of the following steps:

1. Pre and post processing can run in parallel to the actual HW inference.
2. Pre and post processing can run while the network is deactivated.
3. Once the network is activated more frames are ready to be inferred immediately, so the inference is more effective.
4. Large queues can flatten jitters in FPS rate.

The tradeoff of having larger queues is:

1. HailoRT consumes more host memory (in particular DMA-able memory which may be limited in some platforms).
2. Increase the overall latency.

The following parameters are activation parameters that take effect when the network needs to be activated:

Threshold (TH): The TH controls the number of frames that will be accumulated inside the input queue before the network should be activated. Since the TH is measured against the queue's fullness then the larger the input queue the more configuration options we have.

The TH is affecting the performance by:

1. A network with a lower TH will be activated more often.
2. The higher the TH the more frames will be accumulated so each time the network is activated the inference will be more effective.
3. The higher the TH the more frames will be accumulated so the latency will be higher.

Timeout (TO): The TO controls the maximum time from the last time the network was activated. Even if there are not enough frames in the input queue (a minimum of 1 frame is required) but enough time has passed the network will be activated in order to limit the latency, which may increase as a result of high TH.

Priority: The priority reflects the relative importance of each network. Networks with a higher priority will be given priority during selection, resulting in better performance and longer running time for these networks.

The priority is affecting the performance by:

1. Network with higher priority will get more running time.
2. Giving higher priority will enhance both the latency and the FPS rates.
3. Increasing the priority of a network will impact the performance of other networks.
4. When the priority of a network is increased, it may cause starvation to other networks if the higher-priority networks are always ready to run (reach their TH or TO).

The general rule of thumb should be:

1. For multi model scheduling we can set the TH to aggregate more frames before the network is scheduled, doing this will make the inference more effective but will raise the overall latency.
2. Using the TH we can perform “soft” prioritization, a smaller TH value will cause the network to be scheduled more often.
3. To limit the latency of the network inference the TO parameter can be set to ensure that a network will be scheduled for running if it has at least a single frame in its input queue and enough time has passed from its previous activation.
4. To enable the schedule to be more effective, we need to make the input and output queues larger so more frames can be accumulated and thus we can parallel the host’s pre and post processing to the actual inference. The tradeoff of this is consuming more memory.

6.9. Stream Multiplexer

The Model Scheduler enables **Stream Multiplexer** which automatically aggregates different streams of the same model, and saves the model’s activation/deactivation times.

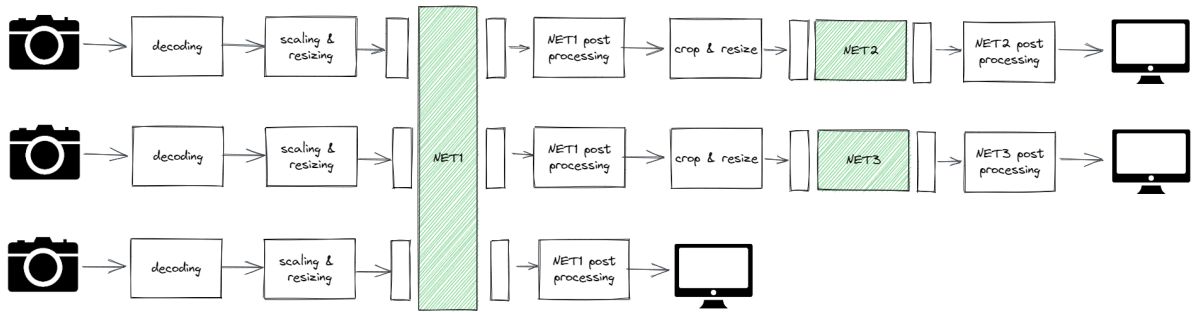

Figure 9. Stream Multiplexer

6.10. Multi-Process Service

The **Multi-Process Service** enables the ability to manage and share a Hailo device between multiple processes, thus providing the ability to use multi-process inference.

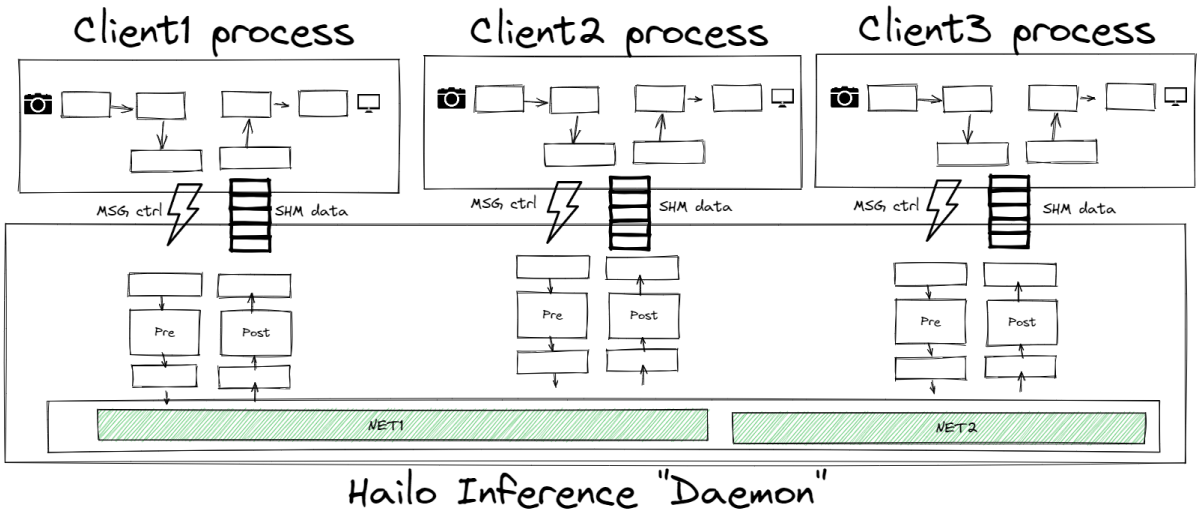

Figure 10. Multi-Process Service

6.10.1. Multi-Process Service on Linux

Installation

When using the Ubuntu installer or a Docker Container, the multi-process service is installed by default with libhailort.

Source Compilation

To compile the multi-process service with libhailort, use the compilation flag `HAILO_BUILD_SERVICE`:

```
cmake -H. -Bbuild -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_SERVICE=1 && sudo cmake --
↪build build --config release --target install
```

Note: Some Linux distributions are delivered without the init system `systemd`, which is required for the multi-process service. In such a case, it is possible to run the service using the compiled executable `hailort_service`.

Enable Service

Once the multi-process service is installed, to enable it, run (`--now` starts the service)

```
sudo systemctl enable --now hailort.service
```

The service can be disabled accordingly.

Setting environment variables when working with HailoRT service

The environment variables for HailoRT service are defined in the file `/etc/default/hailort_service`:

To change an environment variable value, do the following:

1) Change the desired environment variable in `/etc/default/hailort_service`
2) Reload systemd unit files by running: `sudo systemctl daemon-reload`
3) Enable and start service: `sudo systemctl enable --now hailort.service`

Setting environment variables for HailoRT service when working with a Docker Container

To see the options for enabling environment variables in HailoRT service, run:

```
./run_hailort_docker.sh --help
```

Or when using the Hailo SW Suite:

```
./hailo_sw_suite_docker_run.sh --help
```

Example for enabling environment variables inside the HailoRT Docker Container:

```
./run_hailort_docker.sh --hailort-enable-service --service-enable-multi-device-
↪scheduler --service-enable-monitor
```

Usage

- To create a shared VDevice with the service, use the same `hailo_vdevice_params_t::group_id` and enable `hailo_vdevice_params_t::multi_process_service`
- When using `hailortcli run` for inference with multi-process service, use the `--multi-process-service` flag
- When using HailoNet GStreamer element, use the `multi-process-service` property

6.10.2. Multi-Process Service on Windows

Service Installation with Windows Installer

When running the installer, check the `multi-process service` box.

Service Installation with Source Compilation

To compile the multi-process service with libhailort, use the compilation flag `HAILO_BUILD_SERVICE`:

```
cmake -H. -Bbuild -A=x64 -DCMAKE_BUILD_TYPE=Release -DHAILO_BUILD_SERVICE=1 &&
↪cmake --build build --config release --target install
```

Note: For more information about how to compile sources on Windows please see *Windows compile from sources*

To install the multi-process service run the `hailort_service` executable with the flag `install`.

```
bin\windows.AMD64.Release\Release\hailort_service.exe install
```


Start Service

To start the service run:

```
sc start hailort_service
```

The service can be stopped accordingly.

Setting environment variables when working with HailoRT service

The HailoRT environment variables are used to enable/disable different features. The environment variables for HailoRT Windows service are defined in the script `hailort\hailort_service\windows\hailort_service_env_vars.bat`.

To change an environment variable value, do the following:

1) Change the desired environment variable in `hailort\hailort_service\windows\hailort_service_env_vars.bat`
2) Run the script `hailort\hailort_service\windows\hailort_service_env_vars.bat`
3) Restart the service

7. Command Line Tools

The HailoRT package offers several command line tools related to the device and its firmware, providing measurement, configuration and update capabilities. They can be executed from the Linux shell.

Two families of tools are provided:

1. Under the `hailo` command – Python based tools that require a Python environment. When using a Python virtual environment, it should be activated before running any of the `hailo` tools.
2. Under the `hailortcli` command – Native (C++ based) tools that don't depend on Python.

To list the available `hailortcli` tools, run:

```
hailortcli --help
```

To list the available `hailo` tools, run:

```
hailo --help
```

The examples below are given for both families.

Note: For additional information and parameters of the sub-commands, use `--help`.

7.1. Scan Tool

The `scan` tool is used to scan all the devices that are connected to the host. It scans the PCIe hierarchy (or Ethernet port) for connected Hailo devices. If found, it will retrieve the bus/device/ function (or IP address) of the device. This tool supports both PCIe and Ethernet interfaces.

```
hailortcli scan
```

```
hailo scan
```

Note: The `--interface-ip` flag of this command refers to the IP address of the *host's* interface that is connected to the Hailo device. It does not refer to the address of the Hailo device itself.

7.2. Parse-HEF Tool

The `parse-hef` tool is used for parsing an existing HEF file and getting information about its content, such as input/output formats and type, singe/multi context, and so on.


```
hailortcli parse-hef example.hef
```

7.3. Inference

The `run` tool is used for inference. It loads a given HEF to the device and then sends and receives data. While running, it performs several measurements such as FPS and power. It can also control the running mode:

- `streaming` – Streaming inference with random data.
- `hw-only` – Similar to the full streaming mode, but skips pre-infer and post-infer steps on host.

This tool supports both PCIe and Ethernet interfaces.

```
hailortcli run example.hef
```

```
hailo run example.hef
```

The `--batch-size` option of this tool sets the batch size in case of context switch, which means after how many frames a context switch will take place. It should not be confused with the `--frames-count` option that sets the total frame count to be sent to the device.

Note: `--frames-count` sets the total frame count, and must be dividable by `--batch-size`.

7.3.1. Multiple HEF Inference

`run2` is a new subcommand used to run inference over multiple networks simultaneously using the model scheduler.

The advantage of `run2` over `run` is in its parameters - `run2` allows to individually control the parameters per network and per vStream within the network (see `hailortcli run2 --help`), for example:

```
hailortcli run2 set-net hefs/ssd_mobilenet_v1.hef --batch-size 60 set-net hefs/
↪yolov5l.hef --batch-size 10
```

7.3.2. Collect runtime data for the Runtime Profiler

The `collect-runtime-data` option of the run tool creates a `runtime_data_<hef_name>.json` file which can be used for the NN Runtime Profiler (see Dataflow Compiler documentation):

```
hailortcli run example.hef collect-runtime-data
```

```
hailo run example.hef collect-runtime-data
```

Note: `collect-runtime-data` option is currently not supported on Windows.

7.3.3. Pipeline Graph Visualization

The `dot` option (DOT - graph description language), if set, prints the pipeline graph as a .dot file at the specified path, which can be used to analyze the current pipeline. This format is textual, and can be converted to an image (e.g. .png/.svg) using DOT, an executable provided by Graphviz, which is available on Ubuntu.

```
hailortcli run example.hef --dot output_path.dot
```

To get the full-detailed pipeline graph, run an example HEF with the measurements flag:

```
hailortcli run example.hef --dot output_path.dot measure-stats --elem-fps
```

This runs inference on example.hef using VStreams (by default). After the inference is complete, a .csv file containing the gathered statistics collected from the VStream pipeline elements is created, and a .dot file containing the pipeline graph is written to output_path.dot.

An example output for a selected HEF:

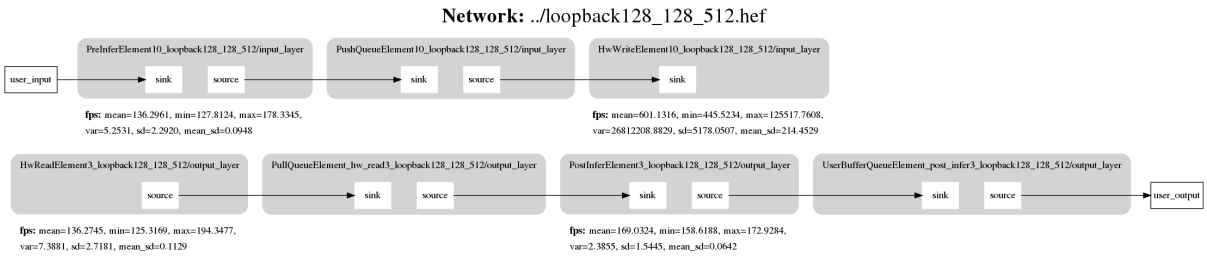

Figure 11. Pipeline Graph Visualization example

7.4. Benchmarking Tool

The `hailortcli benchmark` tool measures performance on a compiled network. This tool wraps the `hailortcli run` tool and makes it easier to perform the full set of measurements for a given HEF. While running, it performs several measurements such as FPS, latency and power. The FPS measurement is performed twice: first, when taking only the Hailo device into account ("hardware only") and second, when taking the full system into account - including the host.

```
hailortcli benchmark example.hef
```

Note: Benchmark HEFs can be compiled using Model Zoo.

- The `--batch-size` option of this tool sets the batch size in case of context switch, which means after how many frames a context switch will take place.
- In some cases, the benchmarking results might slightly vary depending on input data. To generate your own data, see Using Datasets from the Hailo Model Zoo. The `--input-files` flag is used to provide the data-file path.

7.5. Monitor

The `monitor` command prints to the screen information about the models and devices which are currently running. The information is updated every second and is presented in three tables:

1. The first table shows information about the running devices:
    - Device Id – Identification string of the device, BDF for PCIe devices
    - Utilization (%) – Device utilization of the scheduler's runtime
    - Architecture – Device Architecture
2. The second table shows general information about the running models:
    - FPS – Frames per second
    - Utilization (%) – Model utilization of the scheduler's runtime
    - PID – Process id
3. The third table shows the frames state for each stream of each model:
    - For host-to-device stream it will show the number of frames that can be queued, and the number of frames pending to be written to the device.
    - For device-to-host stream it will show the number of frames that can be queued, and the number of frames that were read from the device and are pending to be read by the user.

Usage steps:

1) In the app's process, set environment variable: `HAILO_MONITOR=1`
2) Run the inference application
3) In a different process run:

```
hailortcli monitor
```

Note: Monitor is not currently supported on Windows.

7.6. Firmware Tools

7.6.1. Firmware Configuration Tool

The `fw-config` tool allows reading and writing the firmware configuration. This operation supports both PCIe and Ethernet interfaces. For more information please see *User configuration documentation*.

Run one of the following commands in the Linux terminal to display the tool's help message:

```
hailortcli fw-config --help
```

```
hailo fw-config --help
```


Reading the Firmware Configuration

Reading the firmware configuration can be done by executing one of the following:

```
hailortcli fw-config read
```

```
hailo fw-config read
```

This will return a JSON string that contains the current configuration. The output can be written to a file, for example:

```
hailortcli fw-config read --output-file config.json
```

```
hailo fw-config read --output-file config.json
```

Modifying the Firmware Configuration

In the case where a user config is already loaded, the easiest method is to change the configuration, that is to to read it, modify it, and then write the updated JSON file. Use the following steps:

1. Read the configuration:

   ```
   hailortcli fw-config read --output-file config.json
   ```

   ```
   hailo fw-config read --output-file config.json
   ```

   If the configs are not already loaded, the reading will fail. In that case, a sample user config at `hailort/hailortcli/example_config.json`, or it's symbolic link `platform/hailo_platform/tools/hailocli/example_config.json`, can be used as a reference. Further instructions will refer to the given json as `config.json` but are relevant for the `example_config.json` as well. For further information about the user config optional fields, please refer to the *User configuration documentation*.

2. Modify `config.json` as needed.

3. Write the updated configuration:

   ```
   hailortcli fw-config write config.json
   ```

   ```
   hailo fw-config write config.json
   ```

The changes are only written to the Flash or EEPROM so a restart is required for the changes to take effect.

7.6.2. Firmware Logger Tool

The `fw-logger` tool allows to write the fw-logs to a binary file , which can be later parsed with Hailo costumer support. This operation is supported only via PCIe interface.

Writing the firmware logs can be done by executing the following:

```
hailortcli fw-logger fw_logs.txt
```

```
hailortcli fw-logger fw_logs.txt --overwrite
```

By default, the command appends the log entries to the end of the file. For overwriting the file, use the `--overwrite` flag.

7.6.3. Firmware Control Tool

The `fw-control` tool provides useful firmware control operations.

Note: For running control operations on all existing devices over PCIe interface, use `-s *`.

Identify

The `identify` operation is used to present details about a device. This operation supports both PCIe and Ethernet interfaces.

```
hailortcli fw-control identify
```

```
hailo fw-control identify
```

Reset

The `reset` operation is used to reset a device. There are four different reset types:

- **hard reset** – a full reset of the Hailo-8 chip. This is the default type.
- **nn_core reset** – resets only the neural network core of the Hailo-8 chip.
- **soft reset** – resets the firmware of the chip. A reset of the NN core is done as a part of this process.
- **forced_soft reset** – same as soft reset, but will be performed even if some failure happened in the closing flow that is made as part of the reset.

```
hailortcli fw-control reset --reset-type nn_core
```

```
hailo fw-control reset --reset-type nn_core
```

Hard reset is supported only via the Ethernet interface. NN core reset is supported in both PCIe and Ethernet interfaces.

7.7. Ethernet Related Command Line Tools

The following tools are only applicable when using the Ethernet interface of the device.

7.7.1. Firmware Update Tool

When using the Ethernet interface and a board with flash storage, the firmware update is performed over Ethernet. The update process is verified before finishing, so in case of failure it is safe to rerun the update.

Basic firmware update operation is done by one of the following commands:

```
hailortcli fw-update --ip 1.2.3.4 ./hailo_firmware.bin
```

```
hailo fw-update --ip 1.2.3.4 ./hailo_firmware.bin
```

By default, the board restarts after updating. This behavior can be canceled using the `--skip-reset` flag.

Note: This tool is applicable only when using the Ethernet interface. When using the PCIe interface, the PCIe driver is responsible to load the correct firmware version.

Note: In all of the examples, `1.2.3.4` represents the IP address of the target device.

7.7.2. UDP Rate Limiter

The `udp-limiter` tool allows to limit the UDP communication rate with a given board. When using UDP dataflow, the user may need to limit the bandwidth in order to keep in pace with the neural network core. This command uses the linux `tc` command on the host to configure Linux's `qdisc` mechanism, and therefore, requires sudo.

Set

The set operation will limit to UDP rate to the number of kilo-bits specified in the `--kbit-rate` argument.

```
hailortcli udp-rate-limiter set --kbit-rate=200000 --board-ip 1.2.3.4
```

```
hailo udp-limiter set --kbit-rate=200000 --board-ip 1.2.3.4
```

Reset

The reset operation will cancel the board's UDP rate limitation.

```
hailortcli udp-rate-limiter reset --board-ip 1.2.3.4
```

```
hailo udp-limiter reset --board-ip 1.2.3.4
```

Autoset

The autoset operation will determine the necessary UDP rate limitation based on an HEF and a desired FPS rate specified in `--fps`. The limitation takes into account the input/output throughput of the HEF.

```
hailortcli udp-rate-limiter autoset --fps 200 --hef example.hef --board-ip 1.2.3.4
```

```
hailo udp-limiter autoset --fps 200 --hef example.hef --board-ip 1.2.3.4
```

7.7.3. Ethernet Related Shell Scripts

The Ethernet interface related shell scripts are provided in GitHub. The scripts are located at `hailort/scripts/` .

- `configure_interface.sh` - This script assigns a manual IP address to the host's Ethernet interface. For example:

  ```
  ./configure_interface.sh eth0 -i 10.0.0.50
  ```

- `configure_ethernet_buffers.sh` - This script configures several buffer sizes in the Linux kernel and the driver to ensure the stability of the UDP data communication with the device. For example:


```
./configure_ethernet_buffers.sh eth0
```

- `ubuntu_ethernet_statistics.sh` – This script prints statistics regarding Ethernet communication errors, to assist with dataflow debugging:

```
./ubuntu_ethernet_statistics.sh eth0
```


8. PCIe Driver

In order to work with Hailo device over PCIe, a kernel driver must be installed. The PCIe driver for Linux is an external kernel module (also known as out-of-tree kernel module).

Normally, the driver is installed from the *hailort-pcie-driver.deb* or the *Yocto recipes* (for embedded platforms).

Note: By default, the PCIe driver is installed using the DKMS framework (if installed). This framework automatically rebuilds the driver when a new kernel is installed.

8.1. Manual Setup

Manual setup may be required on other platforms (For example - embedded environment without Hailo Yocto layer).

8.1.1. Software Requirements

- wget (needed to download the firmware)
- CMake (needed to compile the driver)

8.1.2. Download

HailoRT PCIe driver sources can be cloned from GitHub using:

```
git clone https://github.com/hailo-ai/hailort-drivers.git
```

8.1.3. Compilation

Note: In order to compile the driver, a pre-built Linux kernel, including all configuration and headers is required.

To compile the driver locally, run the following commands from the driver source path:

```
cd linux/pcie
make all
```

The compiled driver binary will be created in the same directory as `hailo_pci.ko`.

8.1.4. Installation

In order to install the driver, run the following:

```
# Install the driver in /lib/modules
sudo make install

# Load the driver (needs to be done once, after installation the driver will be loaded
↪on boot)
sudo modprobe hailo_pci
```

After installation, run the following:


```
./download_firmware.sh
sudo mv hailo8_fw.<VERSION>.bin /lib/firmware/hailo/hailo8_fw.bin
```

Then update the device manager:

- Copy the udev rules

```
sudo cp ./linux/pcie/51-hailo-udev.rules /etc/udev/rules.d/
```

- To apply the changes, either reboot or run:

```
sudo udevadm control --reload-rules && sudo udevadm trigger
```

8.2. Parameters

The PCIe driver supports parameters that can change the driver's behavior:

- **no_power_mode**
  - Disables automatic PCIe D0->D3 transition.
- **force_desc_page_size**
  - Determines the maximum DMA descriptor page size.
  - Must be a power of 2.
  - Needs to be set on platforms that do not support large PCIe transactions.

There are two options for changing the parameters:

1. Edit/create the file `/etc/modprboe.d/hailo_pci.conf`
   - A template file (with the same name) is provided in the driver's source directory.
   - Reloading the driver is required.
2. Define the desired parameters upon loading the driver via modprobe. For example:

```
sudo modprobe hailo_pci force_desc_page_size=128
```


9. SoC Features

9.1. Temperature Monitoring

During operation, the system continuously monitors the chip's internal temperature and verifies that the device is operating within the safe temperature range by executing the following mechanisms:

- Temperature throttling (may be disabled)
- Over-temperature protection (always on, kicks in when reaching the temperature AMR)

When the temperature throttling feature is enabled, the FW is monitoring the chip's junction temperature (Tjunction), performing frequency throttling (as required) to avoid exceeding the temperature AMR.

To that end the chip's junction temperature range has been divided to three zones:

- Green Zone - Normal operation (no throttling). The Green Zone's temperature range is determined by `temperature_orange_threshold` and `temperature_orange_hysteresis_threshold`.
- Orange Zone - Throttling is active (if enabled) to supervise the chip's temperature. The Orange Zone's temperature range is determined by `temperature_orange_threshold`, `temperature_orange_hysteresis_threshold`, `temperature_red_threshold` and `temperature_red_hysteresis_threshold`.
- Red Zone - Not safe for operation. The Red Zone's temperature range is determined by `temperature_red_threshold` and `temperature_red_hysteresis_threshold`.

As long as Tjunction remains below the `temperature_orange_threshold`, the FW remains in the Green Zone and continues normal operation. After exceeding the `temperature_orange_threshold`, the FW enters the Orange Zone and decreases the internal clock frequency gradually, from a performance of 100% down to a performance of 50% and recovers as explained below. In case the `temperature_red_threshold` is exceeded, the chip enters the Red Zone, in which the over-temperature protection kicks in, and the input and output streams will be shut down. Any attempt to open new streams when the device is still on the Red Zone will fail.

In case the chip is cooling down, the FW increases the internal clock frequency gradually, recovering the performance degradation caused by the thermal throttling. Performance recovery is based on a hysteresis threshold for each of the corresponding thresholds mentioned above, this is intended to stabilize the throttling process and prevent frequent clock and zones changes.

The chip changes from the Orange Zone to the Red Zone when Tjunction exceeds `temperature_red_threshold` and returns to the Orange Zone when Tjunction is lower than `temperature_red_hysteresis_threshold`. The same logic applies when returning from the Orange Zone to the Green Zone, with `temperature_orange_threshold` and `temperature_orange_hysteresis_threshold` respectively.

The parameters `temperature_orange_threshold`, `temperature_orange_hysteresis_threshold`, `temperature_red_threshold` and `temperature_red_hysteresis_threshold` can be overwritten by the *User configuration*, although adjusting them is not recommended.

Table 1. Temperature Thresholds

Threshold Type	Description	Default value (in degree Celsius)
temperature_orange_threshold	Exceeding this threshold triggers entrance to the Orange Zone, which activates the throttling mechanism	104
temperature_orange_hysteresis_threshold	After being in the Orange Zone, moving back below this threshold returns the chip back to the Green Zone, where it is functioning normally	99

Continued on next page

Table 1 – continued from previous page

Threshold Type	Description	Default value (in degree Celsius)
temperature_red_threshold	Exceeding this threshold triggers entrance to the Red Zone, which is considered not safe for operation	120
temperature_red_hysteresis_threshold	After being in the Red Zone, moving back below this threshold returns the chip back to the Orange Zone	116

Note: Current is also monitored continuously, and overcurrent throttling is enabled by default and should not be disabled.

9.2. User Configuration

The firmware has various settings that can be managed by the user. This configuration is stored on the board's flash or EEPROM memory. It is read at boot time and contains several entries such as IP address, MAC address, and board name. The configuration consists of several sections, for example:

- **Network** – controls network parameters such as DHCP usage (boolean), MAC address, static IP address, static gateway address, and static netmask.
- **System** – includes fields related to power, frequency, and maintenance:
  - `name` – contains the board name. This name is used for identification.
  - `max_neural_network_core_clock_rate` – sets the clock rate of the neural network core. The supported values for this field are 100, 200 and 400 MHz, but it cannot raise the clock rate above the maximum supported value for each board or module.
  - `overcurrent_parameters_source` – can be set to `USER_CONFIG_VALUES` to let the values in the other over current configuration fields to take effect. Note that over current protection is only available on the mPCIe and M.2 modules.
  - `overcurrent_monitoring_red_threshold` – sets the maximum current threshold. This field supports any integer. It is given in mA units.
  - `overcurrent_conversion_time_microseconds` – sets the current sensor averaging period. It supports the following values: 140, 204, 332, 588, 1100, 2116, 4156, 8244. All values are given in microseconds.
  - `overcurrent_throttling_enable` – enables the overcurrent throttling option of the overcurrent monitoring component. Note that overcurrent throttling is not available on the mPCIe module.
  - `temperature_parameters_source` – can be set to `USER_CONFIG_VALUES` to let the values in the other temperature configuration fields to take effect.
  - `temperature_red_threshold` – sets the red threshold of the temperature monitoring component. When Tjunction reaches above this threshold, all PCIe streams are forcefully closed. All values are given in degree Celsius. Adjusting this parameter is not recommended.
  - `temperature_red_hysteresis_threshold` – sets the red hysteresis threshold of the temperature monitoring component. Tjunction has to decrease below this threshold (while being in the Red Zone) in order to return for the device to return back to the Orange Zone. All values are given in degree Celsius. Adjusting this parameter is not recommended.
  - `temperature_orange_threshold` – sets the orange threshold of the temperature monitoring component. When Tjunction reaches above this threshold, the temperature throttling begins (in case

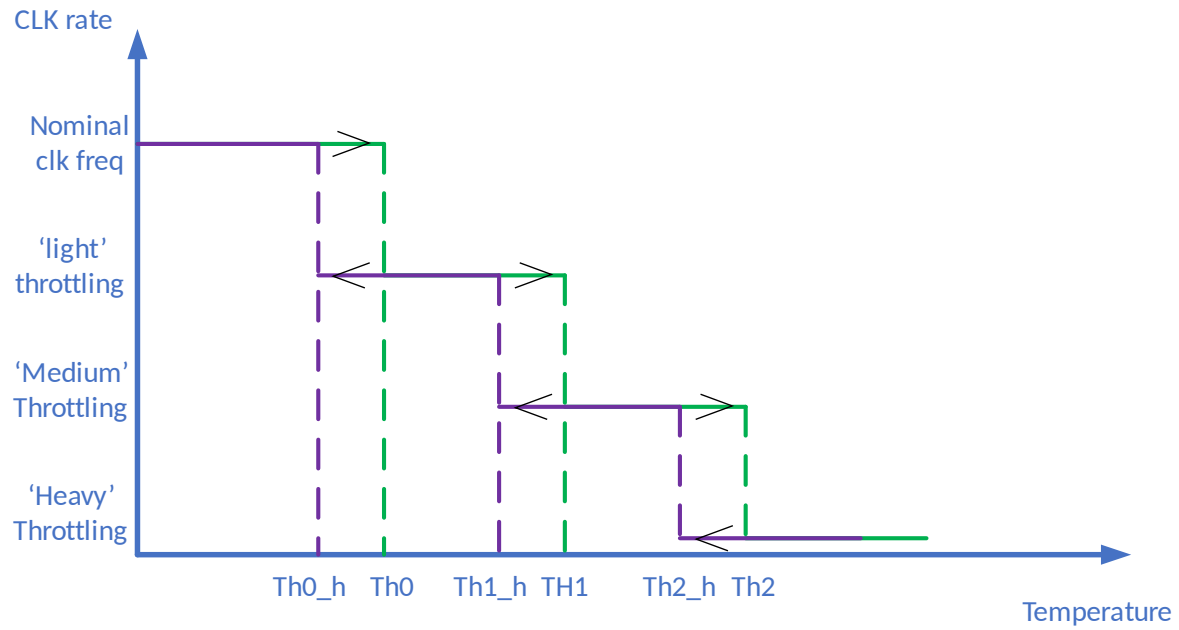

Figure 12. A qualitative frequency/temperature graph

all other conditions are met). All values are given in degree Celsius. Adjusting this parameter is not recommended.

- `temperature_orange_hysteresis_threshold` – sets the orange hysteresis threshold of the temperature monitoring component. Tjunction has to decrease below this threshold (while being in the Orange Zone) in order for the device to return back to the Green Zone. All values are given in degree Celsius. Adjusting this parameter is not recommended.
- `temperature_throttling_enable` – enables the temperature throttling option of the temperature monitoring component. Note that temperature throttling is not available on the mPCIe module.

Warning: Temperature and overcurrent throttling should remain enabled, disabling the features should occur only in special cases.

- **Control** – includes the `udp_port` field, which is used for changing the control port. Typically this is left at the default value.

9.2.1. Example Configuration

The configuration uses the JSON format. For example:

```
{
  "network": {
    "static_ip_address": "10.0.0.1",
    "static_netmask": "255.255.255.0",
    "should_use_dhcp": false,
    "mac_address": "80:00:DE:AD:BE:EF"
  },
  "system": {
    "name": "Hailo-8 Test",
    "max_neural_network_core_clock_rate": "100MHZ",
    "overcurrent_parameters_source": "USER_CONFIG_VALUES",
    "overcurrent_monitoring_red_threshold": 1100,
```

(continues on next page)

(continued from previous page)

```
    "overcurrent_conversion_time_microseconds": 140
  }
}
```

This example configuration sets several fields related to networking such as IP and MAC addresses. These are applicable when the device is connected through the ethernet interface. They are not applicable for the mPCIe and M.2 modules.

In addition, the clock rate of the neural network core is reduced to 100 MHz in order to consume less power. This example also changes the parameters of the mPCIe and M.2 modules over current protection.

For additional information regarding reading or editing the user config please refer to the *Firmware configuration tool doc*.

9.3. Power Modes

Hailo-8 supports two power modes (via `hailo_power_mode_t`):

- Performance power mode (default)
- Ultra-performance power mode

The difference between these two modes is the PCIe data-transfer size. In performance power mode - each PCIe data transfer is larger, allowing the PCIe data lanes to remain quiet for a longer duration and as a result switch to deeper PCIe power states in case the PCIe ASPM (Active-State Power Management) mode is enabled (leads to lower power consumption). This is the default working mode. In contrast to that, in ultra-performance power mode, each PCIe data transfer is smaller, and therefore might not be long enough to allow switching to a lower power state.

Note that in some cases, working in the ultra-performance power mode might lead to increased performance (at the cost of power) when compared to the default performance power mode. This might be the result of other system flows, such as the CPU entering a C-state low power mode due to the transfer size changes. To maximize performance it is possible to enable ultra-performance power mode.

10. Yocto

10.1. The Meta-Hailo Layers

This section will guide through the integration of Hailo's Yocto layers into your own Yocto environment.

The layers are stored in Meta-Hailo GitHub, with a branch for each supported yocto release:

- Zeus (kernel 5.4.24)
- Dunfell (kernel 5.4.85)
- Honister (kernel 5.14)
- Kirkstone (kernel 5.15)

Warning: Kirkstone branch does not support i.MX6 devices.

Note: Zeus will not be supported by HailoRT future version.

The instructions were written after being tested on the DART-MX8M embedded platform.

There are 3 exported layers:

10.1.1. HailoRT

- `meta-hailo-libhailort` - contains HailoRT usermode library, binaries, python, etc.
- `meta-hailo-accelerator` - contains HailoRT PCIe driver, and Hailo-8 firmware.

10.1.2. TAPPAS

- `meta-hailo-tappas` - contains *TAPPAS* recipes including libgsthailotools and tappas-apps. More details about these recipes, how to setup and integrate them, can be found on the *TAPPAS* documentation.

10.2. Recipes in Meta-Hailo-Libhailort

10.2.1. Libhailort

Hailo's API for running inference on the Hailo-8 chip. The recipe compiles libhailort shared-object into the target device's root file system (`/usr/bin`)

10.2.2. Hailortcli

Hailortcli is a command line utility wrapper for libhailort operations, including inference fw controls, measurements and more.

10.2.3. Pyhailort

Hailo's python API. The recipe compiles pyhailort shared object, unpacks the python directory, and uses `setup-tools` to install into `python/site-packages` on the target device's root file system.

The recipe depends on all the python recipes under `meta-hailo/recipes-python/` and it's currently supported by python 3.6, 3.7, 3.8 and 3.9.

Note: pyhailort is not supported on Hardknott and later versions.

Python dependencies:

```
python3-argcomplete, python3-cppheaderparser, python3-netifaces, python3-tqdm,
↪python3-aspy.yaml,
python3-distlib, python3-pyzmq, python3-verboselogs,python3-cfgv, python3-future,
python3-zipp.python3-contextlib2, python3-identify, python3-textutils, python3-zmq
```

10.2.4. Libgsthailo

Hailo's GStreamer plugin for running inference on the Hailo-8 chip. Depends on `libhailort` and GStreamer. The recipe compiles and copies the `libgsthailo.so` file to `/usr/lib/gstreamer-1.0` on the target device's root file system, enabling it to be loaded by GStreamer as a plugin.

10.3. Recipes in Meta-Hailo-Accelerator

10.3.1. Hailo-Firmware

Hailo-8 firmware (`hailo8_fw.bin`). The recipe copies the file to the `/lib/firmware/hailo` directory on the target device's root file system.

Note: For development with the Ethernet interface , it is possible to use the Ethernet-firmware recipe. This recipe is not mandatory for deployment, since in Ethernet cases – the firmware is loaded from flash.

10.3.2. Hailo-Pci

Compiles Hailo-8's PCIe driver and installs it. The source files can be found at GitHub

10.4. Integrating with an Existing Yocto Environment

1. Meta-Hailo sources can be cloned from GitHub using:

   ```
   git clone https://github.com/hailo-ai/meta-hailo.git
   ```

   Note: It is recommended to clone meta-hailo into the sources directory.

2. After cloning meta_hailo, checkout a specific branch, for example (for Dunfell)


```
cd meta-hailo
git checkout dunfell
```

3. The Yocto directory of all supported Yocto releases includes the meta-hailo layers. Open the directory of your required Yocto release and copy the meta-hailo layer directory to your sources directory (same directory as poky, meta-openembedded etc.). Then add it to your `conf/bblayers.conf` like so:

```
BBLAYERS += " ${BSPDIR}/sources/meta-hailo/meta-hailo-accelerator \
       ${BSPDIR}/sources/meta-hailo/meta-hailo-libhailort"
```

4. Add the recipes to your image in `conf/local.conf`:

```
IMAGE_INSTALL_append = "libhailort hailortcli pyhailort libgsthailo hailo-
↪pci hailo-firmware"
```

An example of the expected directory structure:

```
/local
  yocto-dunfell/
    build_xwayland/
      conf/
        local.conf
        bblayers.conf
      cache/
      tmp/
      workspace/
    setup-environment
    README
    sources/
      poky/
      meta-hailo/
        meta-hailo-libhailort/
        meta-hailo-accelerator/
      meta-openembedded/
```

10.5. Validating the Integration's Success

Make sure that the following conditions have been met on the target device:

Recipe	Details
libhailort	Libhailort exists at: `/usr/lib/libhailort.so`
hailortcli	`hailortcli` command works from the global environment. After connecting the hailo8 chip – `scan` and `run` commands should communicate with the board.
pyhailort	Python3 includes the `hailo_platform` module and all dependent modules. `hailo` command works from the global environment. After connecting the hailo8 chip - `fw-control identify` recognizes it and prints information.

libgsthailo	Running `gst-inspect-1.0 | grep hailo` returns hailo elements:
	`hailo: hailonet: hailonet element`
	`hailodevicestats: hailodevicestats element`

hailo-firmware	Hailo's firmware exists at: `/lib/firmware/hailo/hailo_fw.bin`
hailo-pci	Running `modinfo hailo-pci` should return a successful result and info about the driver.

10.6. Offline Builds

Recipes in Yocto may be built offline, if the source code was previously collected on a machine that was connected to a network. The sources collected for the desired recipes will be found at `$DL_DIR` and they are suitable for mirroring (see DL_DIR and source mirrors for more information).

It is assumed that a Yocto environment exists on a machine with network access, which will be used to collect sources for the desired recipes. These sources will be copied to an offline machine with a Yocto environment, which will be used to build the aforementioned recipes. See *Usage Example* for detailed instructions.

10.6.1. Prepare HailoRT External Dependencies

The `prepare_hailort_external_dependencies` task will collect the recipe's sources and external dependencies, packaging them in a *.tar* file in `$DL_DIR`. The following recipes implement this task:

- libhailort
- libgsthailo
- hailortcli
- pyhailort-shared

10.6.2. Environment Variables

Two environment variables control the offline build flow of HailoRT recipes:

- `HAILORT_OFFLINE_BUILD_ENABLE` - Set to "1" to run the offline build flow, both on the machine with network access (that is used to collect sources) and the offline machine (that is used for building).

> **Note:** If this variable is set to "1", the `$DL_DIR` isn't suitable for sharing between parallel builds of HailoRT recipes.

- `HAILORT_OFFLINE_BUILD_USE_EXISTING_TAR` - Set to "1" on the offline machine. The recipe will search for the *.tar* file previously created by `prepare_hailort_external_dependencies`.

These variables are set in `conf/local.conf`.

10.6.3. Usage Example

The steps outlined in *Integrating with an Existing Yocto Environment* are to be run prior to this example. It is assumed that we are operating with the *directory structure* described above.

- On a machine with network connection:
  1. Add the following lines to `conf/local.conf`:
     - `HAILORT_OFFLINE_BUILD_ENABLE="1"` - Required for collecting and packing sources and external dependencies for HailoRT recipes.
     - `BB_GENERATE_MIRROR_TARBALLS = "1"` - This is required to make the `$DL_DIR` safe for mirroring or copying (see BB_GENERATE_MIRROR_TARBALLS for more information).
  2. From the `sources` directory run:

     ```
     source poky/oe-init-build-env
     ```

  3. Collect sources for the the desired hailort recipes. In this example, sources for the `libhailort` and `hailortcli` recipes are collected.
     1. Collect sources for the desired hailort recipes using `--runall=fetch`:

        ```
        bitbake libhailort --runall=fetch
        bitbake hailortcli --runall=fetch
        ```

     2. Run the `prepare_hailort_external_dependencies` task on the desired hailort recipes:

        ```
        bitbake -c prepare_hailort_external_dependencies libhailort hailortcli
        ```

     3. Collect any other sources needed for the image. Sources for `core-image-minimal` are collected in this example:

        ```
        bitbake core-image-minimal --runall=fetch
        ```

- The contents of `$DL_DIR` can now be copied to a machine without network access, where they can be built. For example, building the `libhailort` and `hailortcli` recipes is done as follows on a machine without network access:
  1. From the `sources` directory run:

     ```
     source poky/oe-init-build-env
     ```

  2. Clean artifacts from previous runs:

     ```
     bitbake libhailort --runall=clean
     bitbake hailortcli --runall=clean
     ```

  3. Copy the `$DL_DIR` created on the machine with network connection to the `$DL_DIR` used on the offline machine.
  4. Add the following lines to `conf/local.conf`:
     - `HAILORT_OFFLINE_BUILD_ENABLE = "1"` - Required for unpacking the HailoRT recipe's sources and external dependencies.

- `HAILORT_OFFLINE_BUILD_USE_EXISTING_TAR = "1"` - The recipe will search for the *.tar* file previously created by `prepare_hailort_external_dependencies`, that was copied to the offline machine's `$DL_DIR`.

Note: The following may also be added to `conf/local.conf`:

- `BB_NO_NETWORK = "1"` - In order to validate that no network access is made by bitbake (see BB_NO_NETWORK).
- `BB_SRCREV_POLICY = "cache"` - Might be required in order to stop `git update` from being called during some tasks (see BB_SRCREV_POLICY).

5. Build the desired hailort recipes:

```
bitbake -c build libhailort hailortcli
```


11. Integration With Frameworks

11.1. GStreamer

GStreamer multimedia framework is a framework for creating streaming media applications. GStreamer's development framework makes it possible to write any type of streaming multimedia application. The GStreamer framework is designed to make it easy to write applications that handle audio or video or both. It isn't restricted to audio and video and can process any kind of data flow. The framework is based on plugins that will provide various codecs and other functionality. The plugins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. The GStreamer core function is to provide a framework for plugins, data flow, and media type handling/negotiation. It also provides an API to write applications using the various plugins.

Hailo's application framework for optimized execution of video-processing pipelines (TAPPAS) is GStreamer based.

11.1.1. HailoNet

`HailoNet` GStreamer element Infers data using the Hailo8 chip.

HailoNet is a bin element which contains a `hailosend` element, a `hailorecv` element and a queue between them. The `hailosend` element is responsible for sending the data received from the HailoNet's sink to the Hailo-8 device for inference. Inference is done via the `infer()` API. The `hailorecv` element will read the output buffers from the device and attach them as metadata to the source frame that inferred them. That is why the `HailoNet` has only one source, even in cases where the HEF has more than one output layer.

Parameters

- The data is inferred according to the selected HEF (`hef-path`).
- Selecting a specific PCIe device can be done with the `device-id` property.
- If the pipeline contains a single HailoNet then the single HailoNet will be activated by default, otherwise each HailoNet must explicitly specify if it is active or not by setting the `is_active` property. This property can also be used for switching the activated networks on a specific device during runtime.
- For multi-context networks the `batch-size` property can be used to specify the batch size.
- In case the HEF contains multiple networks, a specific network must be specified for inference using the `net-name` property.

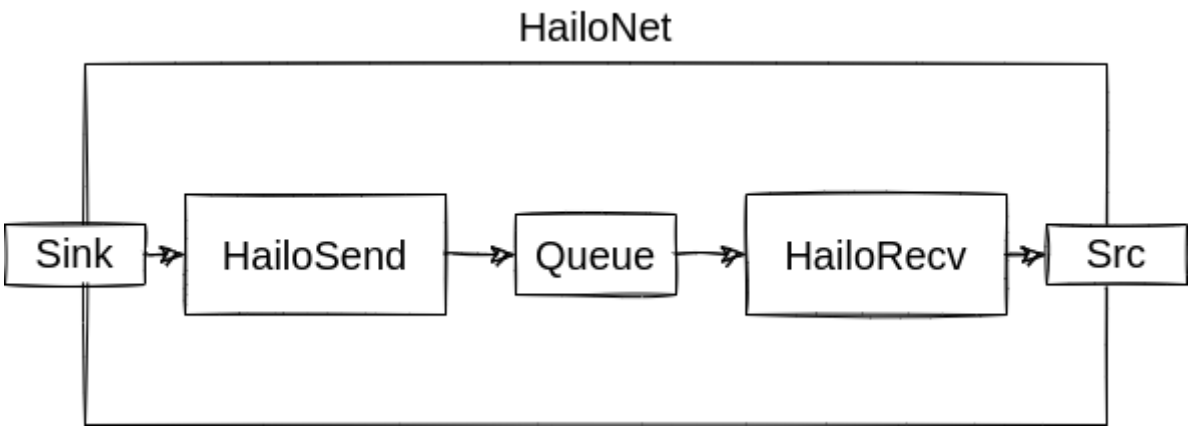

Figure 13. HailoNet Element

11.2. ONNX Runtime (Preview)

ONNX Runtime is a cross-platform inference and training machine-learning accelerator, which allows to inference ONNX models on various supported platforms.

Hailo ONNX Runtime is a fork of ONNX Runtime modified to work on Hailo-8 devices.

Part II

API Reference

12. HailoRT C API Reference

HailoRT is Hailo's runtime library. The C API is used for running inference over compiled models (HEFs) in a C/C++ program.

12.1. Device and control API functions

hailo_status `hailo_scan_devices` (*hailo_scan_devices_params_t* *params, *hailo_device_id_t* *device_ids, ...)

Returns information on all available devices in the system.

Note: ethernet devices are not considered "devices in the system", so they are not scanned in this function. use :hailo_scan_ethernet_devices for ethernet devices.

Parameters

- `params` – **[in]** Scan params, used for future compatibility, only NULL is allowed.
- `device_ids` – **[out]** Array of *hailo_device_id_t* to be fetched from vdevice.
- `device_ids_length` – **[inout]** As input - the size of *device_ids* array. As output - the number of device scanned.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_device_by_id` (const *hailo_device_id_t* *device_id, *hailo_device* *device)

Creates a device by the given device id.

Note: To release a device, call the *hailo_release_device* function with the returned *hailo_device*.

Parameters

- `device_id` – **[in]** Device id, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X or XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx) If NULL is given, uses an arbitrary device found on the system.
- `device` – **[out]** A pointer to a *hailo_device* that receives the allocated PCIe device.

Returns Upon success, returns Expected of a unique_ptr to Device object. Otherwise, returns Unexpected of *hailo_status* error.

hailo_status `hailo_scan_pcie_devices` (*hailo_pcie_device_info_t* *pcie_device_infos, size_t ...)

Returns information on all available pcie devices in the system.

Parameters

- `pcie_device_infos` – **[out]** A pointer to a buffer of *hailo_pcie_device_info_t* that receives the information.
- `pcie_device_infos_length` – **[in]** The number of *hailo_pcie_device_info_t* elements in the buffer pointed to by *pcie_device_infos*.
- `number_of_devices` – **[out]** This variable will be filled with the number of devices. If the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* error is returned.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_parse_pcie_device_info` (const char *device_info_str, *hailo_pcie_device_info_t* ...)

Parse PCIe device BDF string into hailo device info structure.

Note: Call *hailo_scan_pcie_devices* to get all available hailo pcie devices.

Parameters

- `device_info_str` – **[in]** BDF device info, format [<domain>].<bus>.<device>.<func>, same format as in lspci.
- `device_info` – **[out]** A pointer to a *hailo_pcie_device_info_t* that receives the parsed device info.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_create_pcie_device` (*hailo_pcie_device_info_t* *device_info, *hailo_device* *device)

Creates a PCIe device.

Note: To release a device, call the *hailo_release_device* function with the returned *hailo_device*.

Parameters

- `device_info` – **[in]** Information about the device to open. If NULL is given, uses an arbitrary device found on the system.
- `device` – **[out]** A pointer to a *hailo_device* that receives the allocated PCIe device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_scan_ethernet_devices` (const char *interface_name, *hailo_eth_device_info_t* ...)

Returns information on all available ethernet devices in the system.

Parameters

- `interface_name` – **[in]** The name of the network interface to scan.
- `eth_device_infos` – **[out]** A pointer to a buffer of *hailo_eth_device_info_t* that receives the information.
- `eth_device_infos_length` – **[in]** The number of *hailo_eth_device_info_t* elements in the buffer pointed to by *eth_device_infos*.
- `number_of_devices` – **[out]** This variable will be filled with the number of devices. If the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* error is returned.
- `timeout_ms` – **[in]** The time in milliseconds to scan devices.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_ethernet_device` (*hailo_eth_device_info_t* *device_info, *hailo_device* *device)

Creates an ethernet device.

Note: To release a device, call the *hailo_release_device* function with the returned *hailo_device*.

Parameters

- `device_info` – **[in]** Information about the device to open.
- `device` – **[out]** A pointer to a *hailo_device* that receives the allocated ethernet device corresponding to the given information.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_release_device`(*hailo_device* device)

Release an open device.

Parameters `device` – **[in]** A *hailo_device* object to be released.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_device_get_type_by_device_id`(const *hailo_device_id_t* *device_id, ...)

Returns the device type of the given device id string.

Parameters

- `device_id` – **[in]** A :*hailo_device_id_t* device id to check.
- `device_type` – **[out]** A :hailo_device_type_t returned device type.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_identify`(*hailo_device* device, *hailo_device_identity_t* *device_identity)

Sends identify control to a Hailo device.

Parameters

- `device` – **[in]** A *hailo_device* to be identified.
- `device_identity` – **[out]** Information about the device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_core_identify`(*hailo_device* device, *hailo_core_information_t* *core_information)

Receive information about the core cpu.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `core_information` – **[out]** Information about the device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_get_extended_device_information`(*hailo_device* device, ...)

Get extended device information from a Hailo device.

Parameters

- `device` – **[in]** A *hailo_device* to get extended device info from.
- `extended_device_information` – **[out]** Extended information about the device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_set_fw_logger`(*hailo_device* device, *hailo_fw_logger_level_t* level, uint32_t interface_mask)

Configure fw logger level and interface of sending.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `level` – **[in]** The minimum logger level.
- `interface_mask` – **[in]** Output interfaces (mix of *hailo_fw_logger_interface_t*).

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_set_throttling_state`(*hailo_device* device, bool should_activate)

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `should_activate` – **[in]** Should be true to enable or false to disable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_get_throttling_state`(*hailo_device* device, bool *is_active)

Get current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `is_active` – **[out]** A pointer to the temperature protection or overcurrent protection components throttling state: true if active, false otherwise.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_wd_enable`(*hailo_device* device, *hailo_cpu_id_t* cpu_id)

Enable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to enable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_wd_disable`(*hailo_device* device, *hailo_cpu_id_t* cpu_id)

Disable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to disable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_wd_config`(*hailo_device* device, *hailo_cpu_id_t* cpu_id, uint32_t wd_cycles, ...)

Configure firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to configure.
- `wd_cycles` – **[in]** Number of cycles until watchdog is triggered.
- `wd_mode` – **[in]** A *hailo_watchdog_mode_t* indicating which WD mode to configure.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_get_previous_system_state` (*hailo_device* device, *hailo_cpu_id_t* cpu_id, uint32_t ...)

Read the FW previous system state.

Note: Advanced API. Please use with care.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU to state to read.
- `previous_system_state` – **[out]** A *uint32_t* to be filled with the previous system state. 0 indicating external reset, 1 indicating WD HW reset, 2 indicating WD SW reset, 3 indicating SW control reset.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_set_pause_frames` (*hailo_device* device, bool rx_pause_frames_enable)

Enable/Disable Pause frames.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `rx_pause_frames_enable` – **[in]** Indicating weather to enable or disable pause frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_get_device_id` (*hailo_device* device, *hailo_device_id_t* *id)

Get device id which is the identification string of the device. BDF for PCIe devices, IP address for Ethernet devices, “Core” for core devices.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `id` – **[out]** The returned device id.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_chip_temperature` (*hailo_device* device, *hailo_chip_temperature_info_t* *temp_info)

Get temperature information on the device

Note: Temperature in Celsius of the 2 internal temperature sensors (TS).

Parameters

- `device` – **[in]** A *hailo_device* object.
- `temp_info` – **[out]** A hailo_chip_temperature_info_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_reset_device` (*hailo_device* device, *hailo_reset_device_mode_t* mode)

Reset device

Parameters

- `device` – **[in]** A *hailo_device* object.
- `mode` – **[in]** The mode of the reset.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_update_firmware` (*hailo_device* device, void *firmware_buffer, uint32_t ...)

Updates firmware to device flash.

Note: Check *hailo_extended_device_information_t.boot_source* returned from *hailo_get_extended_device_information* to verify if the fw is booted from flash.

Parameters

- `device` – **[in]** A *hailo_output_stream* object.
- `firmware_buffer` – **[in]** A pointer to a buffer that contains the firmware to be updated on the *device*.
- `firmware_buffer_size` – **[in]** The size in bytes of the buffer pointed by *firmware_buffer*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_update_second_stage` (*hailo_device* device, void *second_stage_buffer, uint32_t ...)

Updates second stage to device flash.

Note: Check *hailo_extended_device_information_t.boot_source* returned from *hailo_get_extended_device_information* to verify if the fw is booted from flash.

Parameters

- `device` – **[in]** A *hailo_output_stream* object.
- `second_stage_buffer` – **[in]** A pointer to a buffer that contains the second_stage to be updated on the *device*.
- `second_stage_buffer_size` – **[in]** The size in bytes of the buffer pointed by *second_stage_buffer*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_notification_callback` (*hailo_device* device, *hailo_notification_callback* ...)

Sets a callback to be called when a notification with ID *notification_id* will be received

Parameters

- `device` – **[in]** A *hailo_device* to register the callback to.
- `callback` – **[in]** The callback function to be called.
- `notification_id` – **[in]** The ID of the notification.
- `opaque` – **[in]** User specific data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_remove_notification_callback` (*hailo_device* device, *hailo_notification_id_t* ...)

Removes a previously set callback with ID *notification_id*

Parameters

- `device` – **[in]** A *hailo_device* to register the callback to.
- `notification_id` – **[in]** The ID of the notification to remove.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_reset_sensor` (*hailo_device* device, uint8_t section_index)

Reset the sensor that is related to the section index config.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `section_index` – **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_sensor_i2c_bus_index` (*hailo_device* device, *hailo_sensor_types_t* ...)

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `sensor_type` – **[in]** The sensor type.
- `bus_index` – **[in]** The I2C bus index of the sensor.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_load_and_start_sensor` (*hailo_device* device, uint8_t section_index)

Load the configuration with I2C in the section index.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `section_index` – **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_i2c_read` (*hailo_device* device, const *hailo_i2c_slave_config_t* *slave_config, uint32_t ...)

Read data from an I2C slave over a hailo device.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `slave_config` – **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- `register_address` – **[in]** The address of the register from which the data will be read.
- `data` – **[in]** Pointer to a buffer that would store the read data.
- `length` – **[in]** The number of bytes to read into the buffer pointed by *data*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_i2c_write` (*hailo_device* device, const *hailo_i2c_slave_config_t* *slave_config, uint32_t ...)

Write data to an I2C slave over a hailo device.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `slave_config` – **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- `register_address` – **[in]** The address of the register to which the data will be written.
- `data` – **[in]** A pointer to a buffer that contains the data to be written to the slave.
- `length` – **[in]** The size of *data* in bytes.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_dump_sensor_config` (*hailo_device* device, uint8_t section_index, const char ...)

Dump config of given section index into a csv file.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `section_index` – **[in]** Flash section index to load config from. [0-7]
- `config_file_path` – **[in]** File path to dump section configuration into.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_store_sensor_config` (*hailo_device* device, uint32_t section_index, ...)

Store sensor configuration to Hailo chip flash memory.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `section_index` – **[in]** Flash section index to write to. [0-6]
- `sensor_type` – **[in]** Sensor type.
- `reset_config_size` – **[in]** Size of reset configuration.
- `config_height` – **[in]** Configuration resolution height.
- `config_width` – **[in]** Configuration resolution width.
- `config_fps` – **[in]** Configuration FPS.
- `config_file_path` – **[in]** Sensor configuration file path.
- `config_name` – **[in]** Sensor configuration name.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_store_isp_config` (*hailo_device* device, uint32_t reset_config_size, uint16_t ...)

Store sensor ISP configuration to Hailo chip flash memory.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `reset_config_size` – **[in]** Size of reset configuration.
- `config_height` – **[in]** Configuration resolution height.
- `config_width` – **[in]** Configuration resolution width.
- `config_fps` – **[in]** Configuration FPS.
- `isp_static_config_file_path` – **[in]** ISP static configuration file path.
- `isp_runtime_config_file_path` – **[in]** ISP runtime configuration file path.
- `config_name` – **[in]** Sensor configuration name.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_test_chip_memories` (*hailo_device* device)

Test chip memories using smart BIST mechanism.

Parameters `device` – **[in]** - A *hailo_device* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

12.2. VDevice API functions

hailo_status `hailo_init_vdevice_params` (*hailo_vdevice_params_t* *params)

Init vdevice params with default values.

Parameters `params` – **[out]** A hailo_vdevice_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_vdevice` (*hailo_vdevice_params_t* *params, *hailo_vdevice* *vdevice)

Creates a vdevice.

Note: To release a vdevice, call the *hailo_release_vdevice* function with the returned *hailo_vdevice*.

Parameters

- `params` – **[in]** A hailo_vdevice_params_t (may be NULL). Can be initialzed to default values using *hailo_init_vdevice_params*.
- `vdevice` – **[out]** A pointer to a *hailo_vdevice* that receives the allocated vdevice.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_configure_vdevice` (*hailo_vdevice* vdevice, *hailo_hef* hef, *hailo_configure_params_t* ...)

Configure the vdevice from an hef.

Parameters

- `vdevice` – **[in]** A *hailo_vdevice* object to be configured.
- `hef` – **[in]** A *hailo_hef* object to configure the *vdevice* by.
- `params` – **[in]** A hailo_configure_params_t (may be NULL). Can be initialzed to default values using *hailo_init_configure_params*.
- `network_groups` – **[out]** Array of network_groups that were loaded from the HEF file.
- `number_of_network_groups` – **[inout]** As input - the size of network_groups array. As output - the number of network_groups loaded.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_physical_devices` (*hailo_vdevice* vdevice, *hailo_device* *devices, size_t ...)

Gets the underlying physical devices from a vdevice.

Note: The returned physical devices are held in the scope of *vdevice*.

Parameters

- `vdevice` – **[in]** A *hailo_vdevice* object to fetch physical devices from.
- `devices` – **[out]** Array of *hailo_device* to be fetched from vdevice.
- `number_of_devices` – **[inout]** As input - the size of *devices* array. As output - the number of physical devices under vdevice.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_vdevice_get_physical_devices_ids` (*hailo_vdevice* vdevice, *hailo_device_id_t* ...)

Gets the physical devices' ids from a vdevice.

Note: The returned physical devices are held in the scope of *vdevice*.

Parameters

- `vdevice` – **[in]** A *hailo_vdevice* object to fetch physical devices from.
- `devices_ids` – **[out]** Array of *hailo_device_id_t* to be fetched from vdevice.
- `number_of_devices` – **[inout]** As input - the size of *devices_ids* array. As output - the number of physical devices under vdevice.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_release_vdevice` (*hailo_vdevice* vdevice)

Release an open vdevice.

Parameters `vdevice` – **[in]** A :: hailo_vdevice object to be released.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.3. HEF API functions

hailo_status `hailo_create_hef_file` (*hailo_hef* *hef, const char *file_name)

Creates an HEF from file.

Note: To release an HEF, call the *hailo_release_hef* function with the returned *hef*.

Parameters

- `hef` – **[out]** A pointer to a *hailo_hef* that receives the allocated HEF.
- `file_name` – **[in]** The name of the hef file.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_hef_buffer` (*hailo_hef* *hef, const void *buffer, size_t size)

Creates an HEF from buffer.

Note: To release an HEF, call the *hailo_release_hef* function with the returned *hef*.

Parameters

- `hef` – **[out]** A pointer to a *hailo_hef* that receives the allocated HEF.
- `buffer` – **[in]** A pointer to a buffer that contains the HEF content.
- `size` – **[in]** The size in bytes of the HEF content pointed by *buffer*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_release_hef` (*hailo_hef* hef)

Release an open HEF.

Parameters `hef` – **[in]** A *hailo_hef* object to be released.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_all_stream_infos` (*hailo_hef* hef, const char *name, *hailo_stream_info_t* ...)

Gets all stream infos.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `name` – **[in]** The name of the network or network_group which contains the stream_infos. In case network group name is given, the function returns all stream infos of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.
- `stream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- `stream_infos_length` – **[in]** The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- `number_of_streams` – **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status `hailo_hef_get_stream_info_by_name` (*hailo_hef* hef, const char ...)

Gets stream info by name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `stream_name` – **[in]** The name of the stream as presented in the hef.
- `stream_direction` – **[in]** Indicates the stream direction.
- `stream_info` – **[out]** A pointer to a *hailo_stream_info_t* that receives the stream information.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_all_vstream_infos` (*hailo_hef* hef, const char *name, ...)

Gets all virtual stream infos.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `name` – **[in]** The name of the network or network_group which contains the virtual stream infos. In case network group name is given, the function returns all virtual stream infos of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.

- `vstream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- `vstream_infos_count` – **[inout]** As input - the maximum amount of entries in *vstream_infos* array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* and the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_vstream_name_from_original_name` (*hailo_hef* hef, const char ...)

Gets vstream name from original layer name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `original_name` – **[in]** The original layer name as presented in the hef.
- `vstream_name` – **[out]** The name of the vstream for the provided original name, ends with NULL terminator.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_original_names_from_vstream_name` (*hailo_hef* hef, const char ...)

Gets all original layer names from vstream name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `vstream_name` – **[in]** The name of the stream as presented in the hef.
- `original_names` – **[out]** Array of *hailo_layer_name_t*, all original names linked to the provided stream (each name ends with NULL terminator).
- `original_names_length` – **[inout]** As input - the size of original_names array. As output - the number of original_layers names.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_vstream_names_from_stream_name` (*hailo_hef* hef, const char ...)

Gets all vstream names from stream name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `stream_name` – **[in]** The name of the stream as presented in the hef.
- `vstream_names` – **[out]** Array of *hailo_layer_name_t*, all vstream names linked to the provided stream (each name ends with NULL terminator).
- `vstream_names_length` – **[inout]** As input - the size of vstream_names array. As output - the number of vstream names.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_stream_names_from_vstream_name` (*hailo_hef* hef, const char ...)

Gets all stream names from vstream name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `vstream_name` – **[in]** The name of the vstream as presented in the hef.
- `stream_names` – **[out]** Array of *hailo_layer_name_t*, all stream names linked to the provided vstream (each name ends with NULL terminator).
- `stream_names_length` – **[inout]** As input - the size of stream_names array. As output - the number of stream names.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_sorted_output_names` (*hailo_hef* hef, const char ...)

Gets sorted output names from network group name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group. If NULL is passed, the first network_group in the HEF will be addressed.
- `sorted_output_names` – **[out]** List of sorted outputs names.
- `sorted_output_names_count` – **[inout]** As input - the expected size of sorted_output_names list. As output - the number of sorted_output_names.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_get_bottleneck_fps` (*hailo_hef* hef, const char *network_group_name, ...)

Gets bottleneck fps from network group name.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** The name of the network_group. If NULL is passed, the first network_group in the HEF will be addressed.
- `bottleneck_fps` – **[out]** Bottleneck FPS. Note: This is not relevant in the case of multi context.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_calculate_eth_input_rate_limits` (*hailo_hef* hef, const char ...)

Calculate the inputs bandwidths supported by the network described by *hef*. Rate limiting of this manner is to be used for ethernet *hailo_input_stream*.

Note: There are two options to limit the rate of an ethernet input stream to the desired bandwidth:

- Set *hailo_eth_input_stream_params_t.rate_limit_bytes_per_sec* inside *hailo_configure_params_t* before passing it to *hailo_configure_device*.
- On Unix platforms:
  - You may use the command line tool `hailortcli udp-rate-limiter` instead of using this API

Note: The resulting rates calculated assures that *HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC* will not be exceeded. The actual fps must be lower than given, and appropriate log will be printed.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the stream's information.
- `network_group_name` – **[in]** The name of the network_group which contains the stream_infos. If NULL is passed, the first network_group in the HEF will be addressed.
- `fps` – **[in]** The desired fps.
- `rates` – **[out]** A pointer to an array of *hailo_rate_limit_t* that receives the rates.
- `rates_length` – **[inout]** As input - the length of *rates* array. As output - the number of H2D streams.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.4. Network group API functions

hailo_status `hailo_init_configure_params` (*hailo_hef* hef, *hailo_stream_interface_t* stream_interface, ...)

Init configure params with default values for a given hef.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `stream_interface` – **[in]** A *hailo_stream_interface_t* indicating which hailo_stream_parameters_t to create.
- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_init_configure_params_by_vdevice` (*hailo_hef* hef, *hailo_vdevice* vdevice, ...)

Init configure params with default values for a given hef by virtual device.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `vdevice` – **[in]** A *hailo_vdevice* for which we init the params for.
- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_init_configure_params_by_device` (*hailo_hef* hef, *hailo_device* device, ...)

Init configure params with default values for a given hef by device.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `device` – **[in]** A *hailo_device* for which we init the params for.
- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_init_configure_params_mipi_input` (*hailo_hef* hef, *hailo_stream_interface_t* ...)

Init configure params with default values for a given hef, where all input_streams_params are init to be MIPI type.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `output_interface` – **[in]** A *hailo_stream_interface_t* indicating which hailo_stream_parameters_t to create for the output streams.
- `mipi_params` – **[in]** A *hailo_mipi_input_stream_params_t* object which contains the MIPI params for the input streams.

- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_init_configure_network_group_params` (*hailo_hef* hef, ...)

Init configure params with default values for a given hef.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `stream_interface` – **[in]** A *hailo_stream_interface_t* indicating which hailo_stream_parameters_t to create.
- `network_group_name` – **[in]** The name of the network_group to make configure params for. If NULL is passed, the first network_group in the HEF will be addressed.
- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_init_configure_network_group_params_mipi_input` (*hailo_hef* hef, ...)

Init configure params with default values for a given hef, where all input_streams_params are init to be MIPI type.

Parameters

- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `output_interface` – **[in]** A *hailo_stream_interface_t* indicating which hailo_stream_parameters_t to create for the output streams.
- `mipi_params` – **[in]** A *hailo_mipi_input_stream_params_t* object which contains the MIPI params for the input streams.
- `network_group_name` – **[in]** The name of the network_group to make configure params for. If NULL is passed, the first network_group in the HEF will be addressed.
- `params` – **[out]** A hailo_configure_params_t to be filled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_configure_device` (*hailo_device* device, *hailo_hef* hef, *hailo_configure_params_t* ...)

Configure the device from an hef.

Parameters

- `device` – **[in]** A *hailo_device* object to be configured.
- `hef` – **[in]** A *hailo_hef* object to configure the *device* by.
- `params` – **[in]** A hailo_configure_params_t (may be NULL). Can be initialzed to default values using *hailo_init_configure_params*.
- `network_groups` – **[out]** Array of network_groups that were loaded from the HEF file.
- `number_of_network_groups` – **[inout]** As input - the size of network_groups array. As output - the number of network_groups loaded.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_wait_for_network_group_activation` (*hailo_configured_network_group* ...)

Block until network_group is activated, or until timeout_ms is passed.

Parameters

- `network_group` – **[in]** A *hailo_configured_network_group* to wait for activation.
- `timeout_ms` – **[in]** The timeout in milliseconds. If *timeout_ms* is zero, the function returns immediately. If *timeout_ms* is *HAILO_INFINITE*, the function returns only when the event is signaled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_network_groups_infos` (*hailo_hef* hef, *hailo_network_group_info_t* *infos, ...)

Get network group infos from an hef.

Parameters

- `hef` – **[in]** A *hailo_hef* object.
- `infos` – **[out]** Array of hailo_network_group_info_t to be filled.
- `number_of_infos` – **[inout]** As input - the size of infos array. As output - the number of infos loaded.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_network_group_get_all_stream_infos` (*hailo_configured_network_group* ...)

Gets all stream infos from a configured network group

Parameters

- `network_group` – **[in]** A *hailo_configured_network_group* object.
- `stream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- `stream_infos_length` – **[in]** The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- `number_of_streams` – **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status `hailo_network_group_get_input_stream_infos` (*hailo_configured_network_group* ...)

Gets all input stream infos from a configured network group

Parameters

- `network_group` – **[in]** A *hailo_configured_network_group* object.
- `stream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- `stream_infos_length` – **[in]** The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- `number_of_streams` – **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status `hailo_network_group_get_output_stream_infos` (*hailo_configured_network_group* ...)

Gets all output stream infos from a configured network group

Parameters

- `network_group` – **[in]** A *hailo_configured_network_group* object.
- `stream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- `stream_infos_length` – **[in]** The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*

- `number_of_streams` – **[out]** This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status `hailo_activate_network_group`(*hailo_configured_network_group* network_group, ...)

Activates hailo_device inner-resources for context_switch inference.

Parameters

- `network_group` – **[in]** NetworkGroup to be activated.
- `activation_params` – **[in]** Optional parameters for the activation (may be NULL).
- `activated_network_group_out` – **[out]** Handle for the activated network_group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_deactivate_network_group`(*hailo_activated_network_group* ...)

De-activates hailo_device inner-resources for context_switch inference.

Parameters `activated_network_group` – **[in]** NetworkGroup to deactivate.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_input_stream`(*hailo_configured_network_group* configured_network_group, ...)

Return input stream from configured network_group by stream name.

Parameters

- `configured_network_group` – **[in]** NetworkGroup to get stream from.
- `stream_name` – **[in]** The name of the input stream to retrieve.
- `stream` – **[out]** The returned input stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_stream`(*hailo_configured_network_group* configured_network_group, ...)

Return output stream from configured network_group by stream name.

Parameters

- `configured_network_group` – **[in]** NetworkGroup to get stream from.
- `stream_name` – **[in]** The name of the output stream to retrieve.
- `stream` – **[out]** The returned output stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_latency_measurement`(*hailo_configured_network_group* ...)

Returns the network latency (only available if latency measurement was enabled).

Parameters

- `configured_network_group` – **[in]** NetworkGroup to get the latency measurement from.
- `network_name` – **[in]** Network name of the requested latency measurement. If NULL is passed, all the networks in the network group will be addressed, and the resulted measurement is avarage latency of all networks.
- `result` – **[out]** Output latency result.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_scheduler_timeout` (*hailo_configured_network_group* ...)

Sets the maximum time period that may pass before getting run time from the scheduler, even without reaching the minimum required send requests (e.g. threshold - see *hailo_set_scheduler_threshold()*), as long as at least one send request has been sent. This time period is measured since the last time the scheduler gave this network group run time.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*, and before the creation of any vstreams.

Note: The default timeout is 0ms.

Note: Currently, setting the timeout for a specific network is not supported.

Note: The timeout may be ignored to prevent idle time from the device.

Parameters

- `configured_network_group` – **[in]** NetworkGroup for which to set the scheduler timeout.
- `timeout_ms` – **[in]** Timeout in milliseconds.
- `network_name` – **[in]** Network name for which to set the timeout. If NULL is passed, the timeout will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_scheduler_threshold` (*hailo_configured_network_group* ...)

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler. If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see *hailo_set_scheduler_timeout()*), the scheduler will consider the network ready regardless.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*, and before the creation of any vstreams.

Note: The default threshold is 0, which means HailoRT will apply an automatic heuristic to choose the threshold.

Note: Currently, setting the threshold for a specific network is not supported.

Note: The threshold may be ignored to prevent idle time from the device.

Parameters

- `configured_network_group` – **[in]** NetworkGroup for which to set the scheduler threshold.
- `threshold` – **[in]** Threshold in number of frames.

- `network_name` – **[in]** Network name for which to set the threshold. If NULL is passed, the threshold will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_scheduler_priority`(*hailo_configured_network_group* ...)

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. bigger number represent higher priority.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*.

Note: The default priority is HAILO_SCHEDULER_PRIORITY_NORMAL.

Note: Currently, setting the priority for a specific network is not supported.

Parameters

- `configured_network_group` – **[in]** NetworkGroup for which to set the scheduler priority.
- `priority` – **[in]** Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.
- `network_name` – **[in]** Network name for which to set the priority. If NULL is passed, the priority will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.5. Virtual stream API functions

hailo_status `hailo_hef_make_input_vstream_params`(*hailo_hef* hef, const char *name, bool ...)

Creates input virtual stream params linked to a network or a network group.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `name` – **[in]** The name of the network group or network which contains the input virtual streams. In case network group name is given, the function returns input virtual stream params of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns input virtual stream params of the given network. If NULL is passed, the function returns the input virtual stream params of all the networks of the first network group.
- `quantized` – **[in]** Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all input virtual streams.
- `input_params` – **[out]** List of params for input virtual streams.
- `input_params_count` – **[inout]** On input: Amount of *input_params* array. On output: Will be filled with the detected amount of input vstreams on the *network* or *network_group*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_hef_make_output_vstream_params` (*hailo_hef* hef, const char *name, bool ...)

Creates output virtual stream params linked to a network or a network group.

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `name` – **[in]** The name of the network group or network which contains the output virtual streams. In case network group name is given, the function returns output virtual stream params of all the networks of the given network group. In case network name is given (provided by *hailo_hef_get_network_infos*), the function returns output virtual stream params of the given network. If NULL is passed, the function returns the output virtual stream params of all the networks of the first network group.
- `quantized` – **[in]** Whether the data returned from the device should be quantized. True means that the data returned to the user is still quantized. False means it's HailoRT's responsibility to de-quantize (rescale) the data.
- `format_type` – **[in]** The default format type for all output virtual streams.
- `output_params` – **[out]** List of params for output virtual streams.
- `output_params_count` – **[inout]** On input: Amount of *output_params* array. On output: Will be filled with the detected amount of output vstreams on the *network* or *network_group*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_make_input_vstream_params` (*hailo_configured_network_group* network_group, ...)

Creates input virtual stream params for a given network_group.

Parameters

- `network_group` – **[in]** Network group that owns the streams.
- `quantized` – **[in]** Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all input virtual streams.
- `input_params` – **[out]** List of params for input virtual streams.
- `input_params_count` – **[inout]** On input: Amount of *input_params* array. On output: Will be filled with the detected amount of input vstreams on the *network_group*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_make_output_vstream_params` (*hailo_configured_network_group* network_group, ...)

Creates output virtual stream params for given network_group.

Parameters

- `network_group` – **[in]** Network group that owns the streams.
- `quantized` – **[in]** Whether the data returned from the device should be quantized. True means that the data returned to the user is still quantized. False means it's HailoRT's responsibility to de-quantize (rescale) the data.
- `format_type` – **[in]** The default format type for all output virtual streams.
- `output_params` – **[out]** List of params for output virtual streams.
- `output_params_count` – **[inout]** On input: Amount of *output_params* array. On output: Will be filled with the detected amount of output vstreams on the *network_group*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_vstream_groups` (*hailo_configured_network_group* network_group, ...)

Gets output virtual stream groups for given network_group. The groups are splitted with respect to their low-level streams.

Parameters

- `network_group` – **[in]** Network group that owns the streams.
- `output_name_by_group` – **[out]** List of params for output virtual streams.
- `output_name_by_group_count` – **[inout]** On input: Amount of *output_name_by_group* array. On output: Will be filled with the detected amount of output vstreams on the *network_group*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_input_vstreams` (*hailo_configured_network_group* ...)

Creates input virtual streams.

Note: To release input virtual streams, call the *hailo_release_input_vstreams* function with the returned *input_vstreams* and *inputs_count*.

Parameters

- `configured_network_group` – **[in]** Network group that owns the streams.
- `inputs_params` – **[in]** List of input virtual stream params to create input virtual streams from.
- `inputs_count` – **[in]** How many members in *input_params*.
- `input_vstreams` – **[out]** List of input virtual streams.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_output_vstreams` (*hailo_configured_network_group* ...)

Creates output virtual streams.

Note: If not creating all output vstreams together, one should make sure all vstreams from the same group are created together. See *hailo_get_output_vstream_groups*

Note: To release output virtual streams, call the *hailo_release_output_vstreams* function with the returned *output_vstreams* and *outputs_count*.

Parameters

- `configured_network_group` – **[in]** Network group that owns the streams.
- `outputs_params` – **[in]** List of output virtual stream params to create output virtual streams from.
- `outputs_count` – **[in]** How many members in *outputs_params*.
- `output_vstreams` – **[out]** List of output virtual streams.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_input_vstream_frame_size` (*hailo_input_vstream* input_vstream, size_t ...)

Gets the size of a virtual stream's frame on the host side in bytes (the size could be affected by the format type - for example using UINT16, or by the data not being quantized yet)

Parameters

- `input_vstream` – **[in]** A *hailo_input_vstream* object.
- `frame_size` – **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_input_vstream_info` (*hailo_input_vstream* input_vstream, *hailo_vstream_info_t* ...)

Gets the *hailo_vstream_info_t* struct for the given vstream.

Parameters

- `input_vstream` – **[in]** A *hailo_input_vstream* object.
- `vstream_info` – **[out]** Will be filled with *hailo_vstream_info_t*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_input_vstream_user_format` (*hailo_input_vstream* input_vstream, ...)

Gets the user buffer format struct for the given vstream.

Parameters

- `input_vstream` – **[in]** A *hailo_input_vstream* object.
- `user_buffer_format` – **[out]** Will be filled with *hailo_format_t*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_vstream_frame_size` (*hailo_output_vstream* output_vstream, ...)

Gets the size of a virtual stream's frame on the host side in bytes (the size could be affected by the format type - for example using UINT16, or by the data not being quantized yet)

Parameters

- `output_vstream` – **[in]** A *hailo_output_vstream* object.
- `frame_size` – **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_vstream_info` (*hailo_output_vstream* output_vstream, ...)

Gets the *hailo_vstream_info_t* struct for the given vstream.

Parameters

- `output_vstream` – **[in]** A *hailo_output_vstream* object.
- `vstream_info` – **[out]** Will be filled with *hailo_vstream_info_t*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_output_vstream_user_format` (*hailo_output_vstream* output_vstream, ...)

Gets the user buffer format struct for the given vstream.

Parameters

- `output_vstream` – **[in]** A *hailo_output_vstream* object.
- `user_buffer_format` – **[out]** Will be filled with *hailo_format_t*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_vstream_frame_size` (*hailo_vstream_info_t* *vstream_info, *hailo_format_t* ...)

Gets the size of a virtual stream's frame in bytes (the size could be affected by the format type - for example using UINT16, or by the data not being quantized yet)

Parameters

- `vstream_info` – **[in]** A *hailo_vstream_info_t* object.
- `user_buffer_format` – **[in]** A *hailo_format_t* object.

- `frame_size` – **[out]** The size of the frame on the host side in bytes.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_vstream_write_raw_buffer` (*hailo_input_vstream* input_vstream, const void ...)

Writes buffer to hailo device via input virtual stream *input_vstream*.

Parameters

- `input_vstream` – **[in]** A *hailo_input_vstream* object.
- `buffer` – **[in]** A pointer to a buffer to be sent. The buffer format comes from the vstream's *format* (Can be obtained using *hailo_get_input_vstream_user_format*) and the shape comes from *shape* inside *hailo_vstream_info_t* (Can be obtained using *hailo_get_input_vstream_info*).
- `buffer_size` – **[in]** *buffer* buffer size in bytes. The size is expected to be the size returned from *hailo_get_input_vstream_frame_size*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_flush_input_vstream` (*hailo_input_vstream* input_vstream)

Blocks until the pipeline buffers of *input_vstream* are flushed.

Parameters `input_vstream` – **[in]** The input vstream to flush.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_vstream_read_raw_buffer` (*hailo_output_vstream* output_vstream, void *buffer, ...)

Reads data from hailo device via *output_vstream* into *dst*.

Parameters

- `output_vstream` – **[in]** A *hailo_output_vstream* object.
- `buffer` – **[in]** A pointer to the received buffer. The buffer format comes from the vstream's *format* (Can be obtained using *hailo_get_output_vstream_user_format*) and the shape comes from *shape* or *nms_shape* inside *hailo_vstream_info_t* (Can be obtained using *hailo_get_output_vstream_info*).
- `buffer_size` – **[in]** *dst* buffer size in bytes.The size is expected to be the size returned from *hailo_get_output_vstream_frame_size*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_release_input_vstreams` (const *hailo_input_vstream* *input_vstreams, size_t ...)

Release input virtual streams.

Parameters

- `input_vstreams` – **[in]** List of input virtual streams to be released.
- `inputs_count` – **[in]** The amount of elements in *input_params*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_release_output_vstreams` (const *hailo_output_vstream* *output_vstreams, size_t ...)

Release output virtual streams.

Parameters

- `output_vstreams` – **[in]** List of output virtual streams to be released.
- `outputs_count` – **[in]** The amount of elements in *output_vstreams*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_clear_input_vstreams` (const *hailo_input_vstream* *input_vstreams, size_t ...)

Clears the pipeline buffers of each vstream in *input_vstreams*.

Parameters

- `input_vstreams` – **[in]** List of input virtual streams to be cleared.
- `inputs_count` – **[in]** The amount of elements in *input_params*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_clear_output_vstreams` (const *hailo_output_vstream* *output_vstreams, size_t ...)

Clears the pipeline buffers of each vstream in *output_vstreams*.

Note: If not all output vstreams from the same group are passed together, it will cause an **undefined behavior** . See *hailo_get_output_vstream_groups*, to get the output vstreams' groups.

Parameters

- `output_vstreams` – **[in]** List of output virtual streams to be cleared.
- `outputs_count` – **[in]** The amount of elements in *output_vstreams*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_infer` (*hailo_configured_network_group* configured_network_group, ...)

Run simple inference using vstreams pipelines.

Note: *configured_network_group* should be activated before calling this function.

Note: the size of each element in *input_buffers* and *output_buffers* should match the product of *frames_count* and the frame size of the matching *hailo_input_vstream* / *hailo_output_vstream*.

Parameters

- `configured_network_group` – **[in]** A *hailo_configured_network_group* to run the inference on.
- `inputs_params` – **[in]** Array of input virtual stream params, indicates *input_buffers* format.
- `input_buffers` – **[in]** Array of *hailo_stream_raw_buffer_by_name_t*. Ths input dataset of the inference.
- `inputs_count` – **[in]** The amount of elements in *inputs_params* and *input_buffers*.
- `outputs_params` – **[in]** Array of output virtual stream params, indicates *output_buffers* format.
- `output_buffers` – **[out]** Array of *hailo_stream_raw_buffer_by_name_t*. Ths results of the inference.
- `outputs_count` – **[in]** The amount of elements in *outputs_params* and *output_buffers*.
- `frames_count` – **[in]** The amount of inferred frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.6. Stream API functions

hailo_status `hailo_set_input_stream_timeout` (*hailo_input_stream* stream, uint32_t timeout_ms)

Set new timeout value to an input stream

Parameters

- `stream` – **[in]** A *hailo_input_stream* object to get the new timeout value.
- `timeout_ms` – **[in]** the new timeout value to be set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_output_stream_timeout` (*hailo_output_stream* stream, uint32_t timeout_ms)

Set new timeout value to an output stream

Parameters

- `stream` – **[in]** A *hailo_output_stream* object to get the new timeout value.
- `timeout_ms` – **[in]** the new timeout value to be set.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

size_t `hailo_get_input_stream_frame_size` (*hailo_input_stream* stream)

Gets the size of a stream's frame on the host side in bytes

Parameters `stream` – **[in]** A *hailo_input_stream* object.

Returns The size of the frame on the host side in bytes

size_t `hailo_get_output_stream_frame_size` (*hailo_output_stream* stream)

Gets the size of a stream's frame on the host side in bytes

Parameters `stream` – **[in]** A *hailo_output_stream* object.

Returns The size of the frame on the host side in bytes

hailo_status `hailo_get_input_stream_info` (*hailo_input_stream* stream, *hailo_stream_info_t* ...)

Gets stream info from the given input stream

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.
- `stream_info` – **[out]** An output *hailo_stream_info_t*.

Returns The size of the frame on the host side in bytes

hailo_status `hailo_get_output_stream_info` (*hailo_output_stream* stream, *hailo_stream_info_t* ...)

Gets stream info from the given output stream

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.
- `stream_info` – **[out]** An output *hailo_stream_info_t*.

Returns The size of the frame on the host side in bytes

hailo_status `hailo_stream_read_raw_buffer` (*hailo_output_stream* stream, void *buffer, size_t size)

Synchronously reads data from a stream.

Note: The output buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: *size* is expected to be stream_info.hw_frame_size.

Parameters

- `stream` – **[in]** A *hailo_output_stream* object.
- `buffer` – **[in]** A pointer to a buffer that receives the data read from *stream*.
- `size` – **[in]** The amount of bytes to read, should be the frame size.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_stream_write_raw_buffer` (*hailo_input_stream* stream, const void *buffer, size_t ...)

Synchronously writes all data to a stream.

Note: The input buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: *size* is expected to be stream_info.hw_frame_size.

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.
- `buffer` – **[in]** A pointer to a buffer that contains the data to be written to *stream*.
- `size` – **[in]** The amount of bytes to write.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_stream_wait_for_async_output_ready` (*hailo_output_stream* stream, size_t ...)

Waits until the stream is ready to launch a new *hailo_stream_read_raw_buffer_async* operation. Each stream has a limited-size queue for ongoing transfers. You can retrieve the queue size for the given stream by calling *hailo_output_stream_get_async_max_queue_size*.

Parameters

- `stream` – **[in]** A *hailo_output_stream* object.
- `transfer_size` – **[in]** Must be the result of *hailo_get_output_stream_frame_size* for the given stream.
- `timeout_ms` – **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If *timeout_ms* has passed and the stream is not ready, returns *HAILO_TIMEOUT*.
- In any other error case, returns *hailo_status* error.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_stream_wait_for_async_input_ready` (*hailo_input_stream* stream, size_t ...)

Waits until the stream is ready to launch a new *hailo_stream_write_raw_buffer_async* operation. Each stream has a limited-size queue for ongoing transfers. You can retrieve the queue size for the given stream by calling *hailo_input_stream_get_async_max_queue_size*.

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.

- `transfer_size` – **[in]** Must be the result of *hailo_get_input_stream_frame_size* for the given stream.
- `timeout_ms` – **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If *timeout_ms* has passed and the stream is not ready, returns *HAILO_TIMEOUT*.
- In any other error case, returns *hailo_status* error.

hailo_status `hailo_output_stream_get_async_max_queue_size` (*hailo_output_stream* stream, ...)

Returns the maximum amount of frames that can be simultaneously read from the stream (by *hailo_stream_read_raw_buffer_async* calls) before any one of the read operations is complete, as signified by *user_callback* being called.

Parameters

- `stream` – **[in]** A *hailo_output_stream* object.
- `queue_size` – **[out]** Returns value of the queue

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_input_stream_get_async_max_queue_size` (*hailo_input_stream* stream, ...)

Returns the maximum amount of frames that can be simultaneously written to the stream (by *hailo_stream_write_raw_buffer_async* calls) before any one of the write operations is complete, as signified by *user_callback* being called.

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.
- `queue_size` – **[out]** Returns value of the queue

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_stream_read_raw_buffer_async` (*hailo_output_stream* stream, void *buffer, ...)

Reads into *buffer* from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *hailo_stream_read_raw_buffer_async* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *hailo_stream_read_raw_buffer_async* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *hailo_stream_read_async_completion_info_t* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*). If the operation has completed successfully, the contents of *buffer* will have been updated by the read operation.

Note: *user_callback* should execute as quickly as possible.

Note: The output buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `stream` – **[in]** A *hailo_output_stream* object.
- `buffer` – **[in]** The buffer to be read into. The buffer must be aligned to the system page size.
- `size` – **[in]** The size of the given buffer, expected to be the result of *hailo_get_output_stream_frame_size*.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.
- `opaque` – **[in]** Optional pointer to user-defined context (may be NULL if not desired).

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case, please wait until *user_callback* is called on previous reads, or call *hailo_stream_wait_for_async_output_ready*. The size of the queue can be determined by calling *hailo_output_stream_get_async_max_queue_size*.
- In any other error case, returns a *hailo_status* error.

hailo_status `hailo_stream_write_raw_buffer_async` (*hailo_input_stream* stream, const void ...)

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *hailo_stream_write_raw_buffer_async* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *hailo_stream_write_raw_buffer_async* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *hailo_stream_write_async_completion_info_t* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The input buffer format comes from the *format* field inside *hailo_stream_info_t* and the shape comes from the *hw_shape* field inside *hailo_stream_info_t*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `stream` – **[in]** A *hailo_input_stream* object.
- `buffer` – **[in]** The buffer to be written. The buffer must be aligned to the system page size.
- `size` – **[in]** The size of the given buffer, expected to be the result of *hailo_get_input_stream_frame_size*.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.
- `opaque` – **[in]** Optional pointer to user-defined context (may be NULL if not desired).

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case please wait until *user_callback* is called on previous writes, or call *hailo_stream_wait_for_async_input_ready*. The size of the queue can be determined by calling *hailo_input_stream_get_async_max_queue_size*.
- In any other error case, returns a *hailo_status* error.

size_t `hailo_get_host_frame_size`(const *hailo_stream_info_t* *stream_info, const ...)

Gets the size of a stream's frame on the host side in bytes (the size could be affected by the format type - for example using UINT16, or by the data not being quantized yet)

Parameters

- `stream_info` – **[in]** The stream's info represented by *hailo_stream_info_t*
- `transform_params` – **[in]** Host side transform parameters

Returns The size of the frame on the host side in bytes

12.7. Transformation API functions

hailo_status `hailo_create_input_transform_context`(const *hailo_stream_info_t* *stream_info, ...)

Creates an input transform_context object. Allocates all necessary buffers used for the transformation (pre-process).

Note: To release the transform_context, call the *hailo_release_input_transform_context* function with the returned *hailo_input_transform_context*.

Parameters

- `stream_info` – **[in]** - A *hailo_stream_info_t* object
- `transform_params` – **[in]** - A *hailo_transform_params_t* user transformation parameters.
- `transform_context` – **[out]** - A *hailo_input_transform_context*

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_release_input_transform_context`(*hailo_input_transform_context* ...)

Releases a transform_context object including all allocated buffers.

Parameters `transform_context` – **[in]** - A *hailo_input_transform_context* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_is_input_transformation_required`(const *hailo_3d_image_shape_t* ...)

Check whether or not a transformation is needed.

Note: In case *transformation_required* is false, the src frame is ready to be sent to HW without any transformation.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer (host shape).
- `src_format` – **[in]** The format of the src buffer (host format).

- `dst_image_shape` – **[in]** The shape of the dst buffer (hw shape).
- `dst_format` – **[in]** The format of the dst buffer (hw format).
- `quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.
- `transformation_required` – **[out]** Indicates whether or not a transformation is needed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_transform_frame_by_input_transform_context` (*hailo_input_transform_context* ...)

Transforms an input frame pointed to by *src* directly to the buffer pointed to by *dst*.

Warning: The buffers must not overlap.

Parameters

- `transform_context` – **[in]** A *hailo_input_transform_context*.
- `src` – **[in]** A pointer to a buffer to be transformed.
- `src_size` – **[in]** The number of bytes to transform. This number must be equal to the input host_frame_size, and less than or equal to the size of *src* buffer.
- `dst` – **[out]** A pointer to a buffer that receives the transformed data.
- `dst_size` – **[in]** The number of bytes in *dst* buffer. This number must be equal to the input hw_frame_size, and less than or equal to the size of *dst* buffer.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_output_transform_context` (const *hailo_stream_info_t* *stream_info, ...)

Creates an output transform_context object. Allocates all necessary buffers used for the transformation (post-process).

Note: To release the transform_context, call the *hailo_release_output_transform_context* function with the returned *hailo_output_transform_context*.

Parameters

- `stream_info` – **[in]** - A *hailo_stream_info_t* object
- `transform_params` – **[in]** - A *hailo_transform_params_t* user transformation parameters.
- `transform_context` – **[out]** - A *hailo_output_transform_context*

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_release_output_transform_context` (*hailo_output_transform_context* ...)

Releases a transform_context object including all allocated buffers.

Parameters `transform_context` – **[in]** - A *hailo_output_transform_context* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_is_output_transformation_required` (const *hailo_3d_image_shape_t* ...)

Check whether or not a transformation is needed.

Note: In case *transformation_required* is false, the src frame is already in the required format without any transformation.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer (hw shape).
- `src_format` – **[in]** The format of the src buffer (hw format).
- `dst_image_shape` – **[in]** The shape of the dst buffer (host shape).
- `dst_format` – **[in]** The format of the dst buffer (host format).
- `quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.
- `transformation_required` – **[out]** Indicates whether or not a transformation is needed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_transform_frame_by_output_transform_context`(*hailo_output_transform_context* ...)

Transforms an output frame pointed to by *src* directly to the buffer pointed to by *dst*.

Warning: The buffers must not overlap.

Parameters

- `transform_context` – **[in]** A *hailo_output_transform_context*.
- `src` – **[in]** A pointer to a buffer to be transformed.
- `src_size` – **[in]** The number of bytes to transform. This number must be equal to the output hw_frame_size, and less than or equal to the size of *src* buffer.
- `dst` – **[out]** A pointer to a buffer that receives the transformed data.
- `dst_size` – **[in]** The number of bytes in *dst* buffer. This number must be equal to the output host_frame_size, and less than or equal to the size of *dst* buffer.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_create_demuxer_by_stream`(*hailo_output_stream* stream, const ...)

Creates an demuxer for the given mux stream. Allocates all necessary buffers used for the demux process.

Note: To release the demuxer, call the *hailo_release_output_demuxer* function with the returned *hailo_output_demuxer*.

Parameters

- `stream` – **[in]** - A *hailo_output_stream* object
- `demux_params` – **[in]** - A *hailo_demux_params_t* user demux parameters.
- `demuxer` – **[out]** - A *hailo_output_demuxer*, used to transform output frames

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_release_output_demuxer`(*hailo_output_demuxer* demuxer)

Releases a demuxer object.

Parameters `demuxer` – **[in]** - A *hailo_output_demuxer* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `hailo_demux_raw_frame_by_output_demuxer` (*hailo_output_demuxer* demuxer, ...)

Demultiplexing an output frame pointed to by *src* directly to the buffers pointed to by *raw_buffers*.

Note: The order of *raw_buffers* should be the same as returned from the function '*hailo_get_mux_infos_by_output_demuxer()*'.

Parameters

- `demuxer` – **[in]** A *hailo_output_demuxer* object used for the demuxing.
- `src` – **[in]** A pointer to a buffer to be demultiplexed.
- `src_size` – **[in]** The number of bytes to demultiplexed. This number must be equal to the hw_frame_size, and less than or equal to the size of *src* buffer.
- `raw_buffers` – **[inout]** A pointer to an array of *hailo_stream_raw_buffer_t* that receives the demultiplexed data read from the *stream*.
- `raw_buffers_count` – **[in]** The number of *hailo_stream_raw_buffer_t* elements in the array pointed to by *raw_buffers*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_demux_by_name_raw_frame_by_output_demuxer` (*hailo_output_demuxer* ...)

Demultiplexing an output frame pointed to by *src* directly to the buffers pointed to by *raw_buffers_by_name*.

Parameters

- `demuxer` – **[in]** A *hailo_output_demuxer* object used for the demuxing.
- `src` – **[in]** A pointer to a buffer to be demultiplexed.
- `src_size` – **[in]** The number of bytes to demultiplexed. This number must be equal to the hw_frame_size, and less than or equal to the size of *src* buffer.
- `raw_buffers_by_name` – **[inout]** A pointer to an array of *hailo_stream_raw_buffer_by_name_t* that receives the demultiplexed data read from the *stream*. hailo_stream_raw_buffer_by_name_t::name should be filled with the demuxes names.
- `raw_buffers_count` – **[in]** The number of *hailo_stream_raw_buffer_by_name_t* elements in the array pointed to by *raw_buffers_by_name*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_mux_infos_by_output_demuxer` (*hailo_output_demuxer* demuxer, ...)

Gets all multiplexed stream infos.

Parameters

- `demuxer` – **[in]** A *hailo_output_demuxer* object.
- `stream_infos` – **[out]** A pointer to a buffer of *hailo_stream_info_t* that receives the information.
- `number_of_streams` – **[inout]** The maximum amount of streams_info to fill. This variable will be filled with the actual number of multiplexed stream_infos. If the buffer is insufficient to hold the information this variable will be set to the requested value, *HAILO_INSUFFICIENT_BUFFER* would be returned.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_fuse_nms_frames` (const *hailo_nms_fuse_input_t* *nms_fuse_inputs, uint32_t ...)

Fuse multiple defused NMS buffers pointed by *nms_fuse_inputs* to the buffer pointed to by *fused_buffer*. This function should be called on *nms_fuse_inputs* after receiving them from HW, and before transformation. This function expects *nms_fuse_inputs* to be ordered by their *class_group_index* (lowest to highest).

Parameters

- `nms_fuse_inputs` – **[in]** Array of hailo_nms_fuse_input_t structs which contain the buffers to be fused.
- `inputs_count` – **[in]** How many members in *nms_fuse_inputs*.
- `fused_buffer` – **[out]** A pointer to a buffer which will contain the fused buffer.
- `fused_buffer_size` – **[in]** The fused buffer size.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.8. Power measurement API functions

hailo_status `hailo_power_measurement` (*hailo_device* device, *hailo_dvm_options_t* dvm, ...)

Perform a single power measurement.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `dvm` – **[in]** Which DVM will be measured. Default (*HAILO_DVM_OPTIONS_AUTO*) will be different according to the board:
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums *HAILO_DVM_OPTIONS_VDD_CORE*, *HAILO_DVM_OPTIONS_MIPI_AVDD* and *HAILO_DVM_OPTIONS_AVDD_H*. Only *HAILO_POWER_MEASUREMENT_TYPES__POWER* can measured with this option.
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- `measurement_type` – **[in]** The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES__AUTO* will select the default value according to the supported features.
- `measurement` – **[out]** The measured value. Measured units are determined due to *hailo_power_measurement_types_t*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_start_power_measurement` (*hailo_device* device, *hailo_averaging_factor_t* ...)

Start performing a long power measurement.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `averaging_factor` – **[in]** Number of samples per time period, sensor configuration value.
- `sampling_period` – **[in]** Related conversion time, sensor configuration value. The sensor samples the power every sampling_period {ms} and averages every averaging_factor samples. The sensor provides a new value every: 2 * sampling_period * averaging_factor {ms}. The firmware wakes up every interval_milliseconds {ms} and checks the sensor. If there is a new value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is 'average of averages', because it averages values that have already been averaged by the sensor.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_set_power_measurement` (*hailo_device* device, *hailo_measurement_buffer_index_t* ...)

Set parameters for long power measurement.

Parameters

- `device` – **[in]** A *hailo_device* object.
- `buffer_index` – **[in]** A *hailo_measurement_buffer_index_t* represents the buffer on the firmware the data would be saved at. Should match the one passed to *hailo_get_power_measurement*.
- `dvm` – **[in]** Which DVM will be measured. Default (*HAILO_DVM_OPTIONS_AUTO*) will be different according to the board:
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums *HAILO_DVM_OPTIONS_VDD_CORE*, *HAILO_DVM_OPTIONS_MIPI_AVDD* and *HAILO_DVM_OPTIONS_AVDD_H*. Only *HAILO_POWER_MEASUREMENT_TYPES__POWER* can measured with this option.
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- `measurement_type` – **[in]** The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES__AUTO* will select the default value according to the supported features.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_get_power_measurement` (*hailo_device* device, *hailo_measurement_buffer_index_t* ...)

Read measured power from a long power measurement

Parameters

- `device` – **[in]** A *hailo_device* object.
- `buffer_index` – **[in]** A *hailo_measurement_buffer_index_t* represents the buffer on the firmware the data would be saved at. Should match the one passed to *hailo_set_power_measurement*.
- `should_clear` – **[in]** Flag indicating if the results saved at the firmware will be deleted after reading.
- `measurement_data` – **[out]** The measurement data, *hailo_power_measurement_data_t*. Measured units are determined due to *hailo_power_measurement_types_t* passed to *hailo_set_power_measurement*

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `hailo_stop_power_measurement` (*hailo_device* device)

Stop performing a long power measurement.

Parameters `device` – **[in]** A *hailo_device* object.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

12.9. Multiple networks API functions

hailo_status `hailo_hef_get_network_infos` (*hailo_hef* hef, const char *network_group_name, ...)

Gets all network infos under a given network group

Parameters

- `hef` – **[in]** A *hailo_hef* object that contains the information.
- `network_group_name` – **[in]** Name of the network_group to get the network infos by. If NULL is passed, the first network_group in the HEF will be addressed.
- `networks_infos` – **[out]** A pointer to a buffer of *hailo_network_info_t* that receives the informations.
- `number_of_networks` – **[inout]** As input - the maximum amount of entries in hailo_network_info_t array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* and the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

hailo_status `hailo_get_network_infos` (*hailo_configured_network_group* network_group, ...)

Gets all network infos under a given network group

Parameters

- `network_group` – **[in]** A *hailo_configured_network_group* object to get the network infos from.
- `networks_infos` – **[out]** A pointer to a buffer of *hailo_network_info_t* that receives the informations.
- `number_of_networks` – **[inout]** As input - the maximum amount of entries in hailo_network_info_t array. As output - the actual amount of entries written if the function returns with *HAILO_SUCCESS* and the amount of entries needed if the function returns *HAILO_INSUFFICIENT_BUFFER*.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, if the buffer is insufficient to hold the information a *HAILO_INSUFFICIENT_BUFFER* would be returned. In any other case, returns a *hailo_status* error.

12.10. Other API definitions

enum `hailo_status`

Values:

enumerator `HAILO_SUCCESS`

Success - No error

enumerator `HAILO_UNINITIALIZED`

No error code was initialized

enumerator `HAILO_INVALID_ARGUMENT`

Invalid argument passed to function

enumerator HAILO_OUT_OF_HOST_MEMORY

Cannot allocate more memory at host

enumerator HAILO_TIMEOUT

Received a timeout

enumerator HAILO_INSUFFICIENT_BUFFER

Buffer is insufficient

enumerator HAILO_INVALID_OPERATION

Invalid operation

enumerator HAILO_NOT_IMPLEMENTED

Code has not been implemented

enumerator HAILO_INTERNAL_FAILURE

Unexpected internal failure

enumerator HAILO_DATA_ALIGNMENT_FAILURE

Data is not aligned

enumerator HAILO_CHUNK_TOO_LARGE

Chunk too large

enumerator HAILO_INVALID_LOGGER_LEVEL

Used non-compiled level

enumerator HAILO_CLOSE_FAILURE

Failed to close fd

enumerator HAILO_OPEN_FILE_FAILURE

Failed to open file

enumerator HAILO_FILE_OPERATION_FAILURE

File operation failure

enumerator HAILO_UNSUPPORTED_CONTROL_PROTOCOL_VERSION

Unsupported control protocol version

enumerator HAILO_UNSUPPORTED_FW_VERSION

Unsupported firmware version

enumerator HAILO_INVALID_CONTROL_RESPONSE

Invalid control response

enumerator HAILO_FW_CONTROL_FAILURE

Control failed in firmware

enumerator HAILO_ETH_FAILURE

Ethernet operation has failed

enumerator HAILO_ETH_INTERFACE_NOT_FOUND

Ethernet interface not found

enumerator HAILO_ETH_RECV_FAILURE

Ethernet failed at recv operation

enumerator HAILO_ETH_SEND_FAILURE

Ethernet failed at send operation

enumerator HAILO_INVALID_FIRMWARE

Firmware bin is invalid

enumerator HAILO_INVALID_CONTEXT_COUNT

Host build too many contexts

enumerator HAILO_INVALID_FRAME

Part or all of the result data is invalid

enumerator HAILO_INVALID_HEF

Invalid HEF

enumerator HAILO_PCIE_NOT_SUPPORTED_ON_PLATFORM

PCIe not supported on platform

enumerator HAILO_INTERRUPTED_BY_SIGNAL

Blocking syscall was interrupted by a signal

enumerator HAILO_START_VDMA_CHANNEL_FAIL

Starting VDMA channel failure

enumerator HAILO_SYNC_VDMA_BUFFER_FAIL

Synchronizing VDMA buffer failure

enumerator HAILO_STOP_VDMA_CHANNEL_FAIL

Stopping VDMA channel failure

enumerator HAILO_CLOSE_VDMA_CHANNEL_FAIL

Closing VDMA channel failure

enumerator HAILO_ATR_TABLES_CONF_VALIDATION_FAIL

Validating address translation tables failure, for FW control use

enumerator HAILO_CONTROL_EVENT_CREATE_FAIL

Creating control event failure

enumerator HAILO_READ_EVENT_FAIL

Reading event failure

enumerator HAILO_DRIVER_FAIL

Driver failure

enumerator HAILO_INVALID_FIRMWARE_MAGIC

Invalid FW magic

enumerator HAILO_INVALID_FIRMWARE_CODE_SIZE

Invalid FW code size

enumerator HAILO_INVALID_KEY_CERTIFICATE_SIZE

Invalid key certificate size

enumerator HAILO_INVALID_CONTENT_CERTIFICATE_SIZE

Invalid content certificate size

enumerator HAILO_MISMATCHING_FIRMWARE_BUFFER_SIZES

FW buffer sizes mismatch

enumerator HAILO_INVALID_FIRMWARE_CPU_ID

Invalid CPU ID in FW

enumerator HAILO_CONTROL_RESPONSE_MD5_MISMATCH

MD5 of control response does not match expected MD5

enumerator HAILO_GET_CONTROL_RESPONSE_FAIL

Get control response failed

enumerator HAILO_GET_D2H_EVENT_MESSAGE_FAIL

Reading device-to-host message failure

enumerator HAILO_MUTEX_INIT_FAIL

Mutex initialization failure

enumerator HAILO_OUT_OF_DESCRIPTORS

Cannot allocate more descriptors

enumerator HAILO_UNSUPPORTED_OPCODE

Unsupported opcode was sent to device

enumerator HAILO_USER_MODE_RATE_LIMITER_NOT_SUPPORTED

User mode rate limiter not supported on platform

enumerator HAILO_RATE_LIMIT_MAXIMUM_BANDWIDTH_EXCEEDED

Rate limit exceeded HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC

enumerator HAILO_ANSI_TO_UTF16_CONVERSION_FAILED
Failed converting ANSI string to UNICODE

enumerator HAILO_UTF16_TO_ANSI_CONVERSION_FAILED
Failed converting UNICODE string to ANSI

enumerator HAILO_UNEXPECTED_INTERFACE_INFO_FAILURE
Failed retrieving interface info

enumerator HAILO_UNEXPECTED_ARP_TABLE_FAILURE
Failed retrieving arp table

enumerator HAILO_MAC_ADDRESS_NOT_FOUND
MAC address not found in the arp table

enumerator HAILO_NO_IPV4_INTERFACES_FOUND
No interfaces found with an IPv4 address

enumerator HAILO_SHUTDOWN_EVENT_SIGNALED
A shutdown event has been signaled

enumerator HAILO_THREAD_ALREADY_ACTIVATED
The given thread has already been activated

enumerator HAILO_THREAD_NOT_ACTIVATED
The given thread has not been activated

enumerator HAILO_THREAD_NOT_JOINABLE
The given thread is not joinable

enumerator HAILO_NOT_FOUND
Could not find element

enumerator HAILO_STREAM_ABORTED_BY_HW
Stream aborted due to an external event

enumerator HAILO_STREAM_ABORTED_BY_USER
Stream recv/send was aborted

enumerator HAILO_PCIE_DRIVER_NOT_INSTALLED
Pcie driver is not installed

enumerator HAILO_NOT_AVAILABLE
Component is not available

enumerator HAILO_TRAFFIC_CONTROL_FAILURE
Traffic control failure

enumerator HAILO_INVALID_SECOND_STAGE

Second stage bin is invalid

enumerator HAILO_INVALID_PIPELINE

Pipeline is invalid

enumerator HAILO_NETWORK_GROUP_NOT_ACTIVATED

Network group is not activated

enumerator HAILO_VSTREAM_PIPELINE_NOT_ACTIVATED

VStream pipeline is not activated

enumerator HAILO_OUT_OF_FW_MEMORY

Cannot allocate more memory at fw

enumerator HAILO_STREAM_NOT_ACTIVATED

Stream is not activated

enumerator HAILO_DEVICE_IN_USE

The device is already in use

enumerator HAILO_OUT_OF_PHYSICAL_DEVICES

There are not enough physical devices

enumerator HAILO_INVALID_DEVICE_ARCHITECTURE

Invalid device architecture

enumerator HAILO_INVALID_DRIVER_VERSION

Invalid driver version

enumerator HAILO_RPC_FAILED

RPC failed

enumerator HAILO_INVALID_SERVICE_VERSION

Invalid service version

enumerator HAILO_NOT_SUPPORTED

Not supported operation

enumerator HAILO_NMS_BURST_INVALID_DATA

Invalid data in NMS burst

enumerator HAILO_OUT_OF_HOST_CMA_MEMORY

Cannot allocate more CMA memory at host

enumerator HAILO_QUEUE_IS_FULL

Cannot push more items into the queue

enumerator HAILO_DMA_MAPPING_ALREADY_EXISTS

DMA mapping already exists

enumerator HAILO_STATUS_COUNT

Must be last!

enumerator HAILO_STATUS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_dvm_options_e

Enum that represents the type of devices that would be measured

Values:

enumerator HAILO_DVM_OPTIONS_VDD_CORE

VDD_CORE DVM

enumerator HAILO_DVM_OPTIONS_VDD_IO

VDD_IO DVM

enumerator HAILO_DVM_OPTIONS_MIPI_AVDD

MIPI_AVDD DVM

enumerator HAILO_DVM_OPTIONS_MIPI_AVDD_H

MIPI_AVDD_H DVM

enumerator HAILO_DVM_OPTIONS_USB_AVDD_IO

USB_AVDD_IO DVM

enumerator HAILO_DVM_OPTIONS_VDD_TOP

VDD_TOP DVM

enumerator HAILO_DVM_OPTIONS_USB_AVDD_IO_HV

USB_AVDD_IO_HV DVM

enumerator HAILO_DVM_OPTIONS_AVDD_H

AVDD_H DVM

enumerator HAILO_DVM_OPTIONS_SDIO_VDD_IO

SDIO_VDD_IO DVM

enumerator HAILO_DVM_OPTIONS_OVERCURRENT_PROTECTION

OVERCURRENT_PROTECTION DVM

enumerator HAILO_DVM_OPTIONS_COUNT

Must be last!

enumerator HAILO_DVM_OPTIONS_AUTO

Select the default DVM option according to the supported features

enumerator HAILO_DVM_OPTIONS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_power_measurement_types_e

Enum that represents what would be measured on the selected device

Values:

enumerator HAILO_POWER_MEASUREMENT_TYPES__SHUNT_VOLTAGE

SHUNT_VOLTAGE measurement type, measured in mV

enumerator HAILO_POWER_MEASUREMENT_TYPES__BUS_VOLTAGE

BUS_VOLTAGE measurement type, measured in mV

enumerator HAILO_POWER_MEASUREMENT_TYPES__POWER

POWER measurement type, measured in W

enumerator HAILO_POWER_MEASUREMENT_TYPES__CURRENT

CURRENT measurement type, measured in mA

enumerator HAILO_POWER_MEASUREMENT_TYPES__COUNT

Must be last!

enumerator HAILO_POWER_MEASUREMENT_TYPES__AUTO

Select the default measurement type according to the supported features

enumerator HAILO_POWER_MEASUREMENT_TYPES__MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_sampling_period_e

Enum that represents all the bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage

Values:

enumerator HAILO_SAMPLING_PERIOD_140US

enumerator HAILO_SAMPLING_PERIOD_204US

enumerator HAILO_SAMPLING_PERIOD_332US

enumerator HAILO_SAMPLING_PERIOD_588US

enumerator HAILO_SAMPLING_PERIOD_1100US

enumerator HAILO_SAMPLING_PERIOD_2116US

enumerator HAILO_SAMPLING_PERIOD_4156US

enumerator HAILO_SAMPLING_PERIOD_8244US

enumerator HAILO_SAMPLING_PERIOD_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_averaging_factor_e

Enum that represents all the AVG bit settings and related number of averages for each bit setting

Values:

enumerator HAILO_AVERAGE_FACTOR_1

enumerator HAILO_AVERAGE_FACTOR_4

enumerator HAILO_AVERAGE_FACTOR_16

enumerator HAILO_AVERAGE_FACTOR_64

enumerator HAILO_AVERAGE_FACTOR_128

enumerator HAILO_AVERAGE_FACTOR_256

enumerator HAILO_AVERAGE_FACTOR_512

enumerator HAILO_AVERAGE_FACTOR_1024

enumerator HAILO_AVERAGE_FACTOR_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_measurement_buffer_index_e

Enum that represents buffers on the device for power measurements storing

Values:

enumerator HAILO_MEASUREMENT_BUFFER_INDEX_0

enumerator HAILO_MEASUREMENT_BUFFER_INDEX_1

enumerator HAILO_MEASUREMENT_BUFFER_INDEX_2

enumerator HAILO_MEASUREMENT_BUFFER_INDEX_3

enumerator HAILO_MEASUREMENT_BUFFER_INDEX_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_device_type_t

Hailo device type

Values:

enumerator HAILO_DEVICE_TYPE_PCIE

enumerator HAILO_DEVICE_TYPE_ETH

enumerator HAILO_DEVICE_TYPE_INTEGRATED

enumerator HAILO_DEVICE_TYPE_MAX_ENUM
Max enum value to maintain ABI Integrity

enum hailo_scheduling_algorithm_e
Scheduler algorithm

Values:

enumerator HAILO_SCHEDULING_ALGORITHM_NONE
Scheduling disabled

enumerator HAILO_SCHEDULING_ALGORITHM_ROUND_ROBIN
Round Robin

enumerator HAILO_SCHEDULING_ALGORITHM_MAX_ENUM
Max enum value to maintain ABI Integrity

enum hailo_device_architecture_e
Device architecture

Values:

enumerator HAILO_ARCH_HAILO8_A0

enumerator HAILO_ARCH_HAILO8

enumerator HAILO_ARCH_HAILO8L

enumerator HAILO_ARCH_HAILO15

enumerator HAILO_ARCH_MAX_ENUM
Max enum value to maintain ABI Integrity

enum hailo_cpu_id_t

Values:

enumerator HAILO_CPU_ID_0

enumerator HAILO_CPU_ID_1

enumerator HAILO_CPU_ID_MAX_ENUM
Max enum value to maintain ABI Integrity

enum `hailo_device_boot_source_t`

Values:

enumerator `HAILO_DEVICE_BOOT_SOURCE_INVALID`

enumerator `HAILO_DEVICE_BOOT_SOURCE_PCIE`

enumerator `HAILO_DEVICE_BOOT_SOURCE_FLASH`

enumerator `HAILO_DEVICE_BOOT_SOURCE_MAX`

Max enum value to maintain ABI Integrity

enum `hailo_endianness_t`

Endianness (byte order)

Values:

enumerator `HAILO_BIG_ENDIAN`

enumerator `HAILO_LITTLE_ENDIAN`

enumerator `HAILO_ENDIANNESS_MAX_ENUM`

Max enum value to maintain ABI Integrity

enum `hailo_format_type_t`

Data format types

Values:

enumerator `HAILO_FORMAT_TYPE_AUTO`

Chosen automatically to match the format expected by the device, usually UINT8. Can be checked using *hailo_stream_info_t* format.type.

enumerator `HAILO_FORMAT_TYPE_UINT8`

Data format type uint8_t - 1 byte per item, host/device side

enumerator `HAILO_FORMAT_TYPE_UINT16`

Data format type uint16_t - 2 bytes per item, host/device side

enumerator `HAILO_FORMAT_TYPE_FLOAT32`

Data format type float32_t - used only on host side (Translated in the quantization process)

enumerator `HAILO_FORMAT_TYPE_MAX_ENUM`

Max enum value to maintain ABI Integrity

enum `hailo_format_order_t`

Data format orders, i.e. how the rows, columns and features are ordered:

- N: Number of images in the batch
- H: Height of the image

- W: Width of the image
- C: Number of channels of the image (e.g. 3 for RGB, 1 for grayscale...)

Values:

enumerator `HAILO_FORMAT_ORDER_AUTO`

Chosen automatically to match the format expected by the device.

enumerator `HAILO_FORMAT_ORDER_NHWC`

- Host side: [N, H, W, C]
- Device side: [N, H, W, C], where width is padded to 8 elements

enumerator `HAILO_FORMAT_ORDER_NHCW`

- Not used for host side
- Device side: [N, H, C, W], where width is padded to 8 elements

enumerator `HAILO_FORMAT_ORDER_FCR`

FCR means first channels (features) are sent to HW:

- Host side: [N, H, W, C]
- Device side: [N, H, W, C]:
  - Input - channels are expected to be aligned to 8 elements
  - Output - width is padded to 8 elements

enumerator `HAILO_FORMAT_ORDER_F8CR`

F8CR means first 8-channels X width are sent to HW:

- Host side: [N, H, W, C]
- Device side: [N, H, W, 8C], where channels are padded to 8 elements:
- ROW1:
  - W X 8C_1, W X 8C_2, ... , W X 8C_n
- ROW2:
  - W X 8C_1, W X 8C_2, ... , W X 8C_n ...

enumerator `HAILO_FORMAT_ORDER_NHW`

Output format of argmax layer:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 elements

enumerator `HAILO_FORMAT_ORDER_NC`

Channels only:

- Host side: [N,C]
- Device side: [N, C], where channels are padded to 8 elements

enumerator HAILO_FORMAT_ORDER_BAYER_RGB

Bayer format:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 elements

enumerator HAILO_FORMAT_ORDER_12_BIT_BAYER_RGB

Bayer format, same as *HAILO_FORMAT_ORDER_BAYER_RGB* where Channel is 12 bit

enumerator HAILO_FORMAT_ORDER_HAILO_NMS

NMS bbox

- Host side

  For each class (*hailo_nms_shape_t.number_of_classes*), the layout is

  ```
  struct (packed) {
    uint16_t/float32_t bbox_count;
    hailo_bbox_t/hailo_bbox_float32_t bbox[bbox_count];
  };
  ```

  The host format type can be either *HAILO_FORMAT_TYPE_FLOAT32* or *HAILO_FORMAT_TYPE_UINT16*.

  Maximum amount of bboxes per class is *hailo_nms_shape_t.max_bboxes_per_class*.

- Device side output (result of NMS layer): Internal implementation

enumerator HAILO_FORMAT_ORDER_RGB888

- Not used for host side
- Device side: [N, H, W, C], where channels are 4 (RGB + 1 padded zero byte) and width is padded to 8 elements

enumerator HAILO_FORMAT_ORDER_NCHW

- Host side: [N, C, H, W]
- Not used for device side

enumerator HAILO_FORMAT_ORDER_YUY2

YUV format, encoding 2 pixels in 32 bits [Y0, U0, Y1, V0] represents [Y0, U0, V0], [Y1, U0, V0]

- Host side: [Y0, U0, Y1, V0]
- Device side: [Y0, U0, Y1, V0]

enumerator HAILO_FORMAT_ORDER_NV12

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, V0, U1, V1] represents [Y0, U0, V0], [Y1, U0, V0], [Y2, U0, V0], [Y3, U0, V0], [Y4, U1, V1], [Y5, U1, V1], [Y6, U1, V1], [Y7, U1, V1]

enumerator HAILO_FORMAT_ORDER_NV21

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, V0, U0, V1, U1] represents [Y0, V0, U0], [Y1, V0, U0], [Y2, V0, U0], [Y3, V0, U0], [Y4, V1, U1], [Y5, V1, U1], [Y6, V1, U1], [Y7, V1, U1]

enumerator HAILO_FORMAT_ORDER_HAILO_YYUV

Internal implementation for HAILO_FORMAT_ORDER_NV12 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, V0, U1, V1] is represented by [Y0, Y1, Y2, Y3, U0, V0, Y4, Y5, Y6, Y7, U1, V1]

enumerator HAILO_FORMAT_ORDER_HAILO_YYVU

Internal implementation for HAILO_FORMAT_ORDER_NV21 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, V0, U0, V1, U1] is represented by [Y0, Y1, Y2, Y3, V0, U0, Y4, Y5, Y6, Y7, V1, U1]

enumerator HAILO_FORMAT_ORDER_RGB4

RGB, where every row is padded to 4.

- Host side: [N, H, W, C], where width*channels are padded to 4.
- Not used for device side

enumerator HAILO_FORMAT_ORDER_I420

YUV format, encoding 8 pixels in 96 bits [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, U1, V0, V1] represents [Y0, U0, V0,], [Y1, U0, V0], [Y2, U0, V0], [Y3, U0, V0], [Y4, U1, V1], [Y5, U1, V1], [Y6, U1, V1], [Y7, U1, V1]

enumerator HAILO_FORMAT_ORDER_HAILO_YYYYUV

Internal implementation for HAILO_FORMAT_ORDER_I420 format [Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7, U0, U1, V0, V1] is represented by [Y0, Y1, Y2, Y3, U0, V0, Y4, Y5, Y6, Y7, U1, V1]

enumerator HAILO_FORMAT_ORDER_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_format_flags_t

Data format flags

Values:

enumerator HAILO_FORMAT_FLAGS_NONE

enumerator HAILO_FORMAT_FLAGS_QUANTIZED

If not set, HailoRT performs the quantization (scaling) step. If set:

- Input data: HailoRT assumes that the data is already quantized (scaled) by the user, so it does not perform the quantization (scaling) step.
- Output data: The data will be returned to the user without rescaling (i.e., the data won't be rescaled by HailoRT).

enumerator HAILO_FORMAT_FLAGS_TRANSPOSED

If set, the frame height/width are transposed. Supported orders:

- *HAILO_FORMAT_ORDER_NHWC*
- *HAILO_FORMAT_ORDER_NHW*
- *HAILO_FORMAT_ORDER_BAYER_RGB*
- *HAILO_FORMAT_ORDER_12_BIT_BAYER_RGB*
- *HAILO_FORMAT_ORDER_FCR*
- *HAILO_FORMAT_ORDER_F8CR*

When set on host side, *hailo_stream_info_t* shape of the stream will be a transposed version of the host buffer (The height and width will be swapped)

enumerator HAILO_FORMAT_FLAGS_HOST_ARGMAX

If set, argmax will be called on the feature dimension. Only set on device side.

enumerator HAILO_FORMAT_FLAGS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_transform_mode_t

Indicates how transformations on the data should be done

Values:

enumerator HAILO_STREAM_NO_TRANSFORM

The vstream will not run the transformation (The data will be in hw format)

enumerator HAILO_STREAM_TRANSFORM_COPY

The transformation process will be part of the vstream send/recv (The data will be in host format).

enumerator HAILO_STREAM_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_direction_t

Stream direction - host to device or device to host

Values:

enumerator HAILO_H2D_STREAM

enumerator HAILO_D2H_STREAM

enumerator HAILO_STREAM_DIRECTION_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_flags_t

Stream flags

Values:

enumerator HAILO_STREAM_FLAGS_NONE

No flags

enumerator HAILO_STREAM_FLAGS_ASYNC

Async stream

enumerator HAILO_STREAM_FLAGS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_dma_buffer_direction_t

Hailo dma buffer direction

Values:

enumerator HAILO_DMA_BUFFER_DIRECTION_H2D

enumerator HAILO_DMA_BUFFER_DIRECTION_D2H

enumerator HAILO_DMA_BUFFER_DIRECTION_BOTH

enumerator HAILO_DMA_BUFFER_DIRECTION_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_buffer_flags_t

Hailo buffer flags

Values:

enumerator HAILO_BUFFER_FLAGS_NONE

No flags - heap allocated buffer

enumerator HAILO_BUFFER_FLAGS_DMA

Buffer is mapped to DMA (will be page aligned implicitly)

enumerator HAILO_BUFFER_FLAGS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_pixels_per_clock_t

Indicates amount of pixels per clock on a MIPI stream

Values:

enumerator HAILO_MIPI_PIXELS_PER_CLOCK_1

enumerator HAILO_MIPI_PIXELS_PER_CLOCK_2

enumerator HAILO_MIPI_PIXELS_PER_CLOCK_4

enumerator HAILO_MIPI_PIXELS_PER_CLOCK_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_clock_selection_t

Indicates Range of MIPI clock selection for a MIPI stream

Values:

enumerator HAILO_MIPI_CLOCK_SELECTION_80_TO_100_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_100_TO_120_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_120_TO_160_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_160_TO_200_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_200_TO_240_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_240_TO_280_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_280_TO_320_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_320_TO_360_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_360_TO_400_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_400_TO_480_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_480_TO_560_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_560_TO_640_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_640_TO_720_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_720_TO_800_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_800_TO_880_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_880_TO_1040_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_1040_TO_1200_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_1200_TO_1350_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_1350_TO_1500_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_1500_TO_1750_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_1750_TO_2000_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_2000_TO_2250_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_2250_TO_2500_MBPS

enumerator HAILO_MIPI_CLOCK_SELECTION_AUTOMATIC

The clock selection is calculated from the data rate.

enumerator HAILO_MIPI_CLOCK_SELECTION_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_data_type_rx_t

Indicates MIPI Rx data type

Values:

enumerator HAILO_MIPI_RX_TYPE_RGB_444

enumerator HAILO_MIPI_RX_TYPE_RGB_555

enumerator HAILO_MIPI_RX_TYPE_RGB_565

enumerator HAILO_MIPI_RX_TYPE_RGB_666

enumerator HAILO_MIPI_RX_TYPE_RGB_888

enumerator HAILO_MIPI_RX_TYPE_RAW_6

enumerator HAILO_MIPI_RX_TYPE_RAW_7

enumerator HAILO_MIPI_RX_TYPE_RAW_8

enumerator HAILO_MIPI_RX_TYPE_RAW_10

enumerator HAILO_MIPI_RX_TYPE_RAW_12

enumerator HAILO_MIPI_RX_TYPE_RAW_14

enumerator HAILO_MIPI_RX_TYPE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_image_in_order_t

Indicates ISP input bayer pixel order

Values:

enumerator HAILO_MIPI_ISP_IMG_IN_ORDER_B_FIRST

enumerator HAILO_MIPI_ISP_IMG_IN_ORDER_GB_FIRST

enumerator HAILO_MIPI_ISP_IMG_IN_ORDER_GR_FIRST

enumerator HAILO_MIPI_ISP_IMG_IN_ORDER_R_FIRST

enumerator HAILO_MIPI_ISP_IMG_IN_ORDER_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_image_out_data_type_t

Indicates ISP output data type

Values:

enumerator HAILO_MIPI_IMG_OUT_DATA_TYPE_RGB_888

enumerator HAILO_MIPI_IMG_OUT_DATA_TYPE_YUV_422

enumerator HAILO_MIPI_IMG_OUT_DATA_TYPE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_light_frequency_t

Values:

enumerator HAILO_MIPI_ISP_LIGHT_FREQUENCY_60HZ

enumerator HAILO_MIPI_ISP_LIGHT_FREQUENCY_50HZ

enumerator ISP_LIGHT_FREQUENCY_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_interface_t

Values:

enumerator HAILO_STREAM_INTERFACE_PCIE

enumerator HAILO_STREAM_INTERFACE_ETH

enumerator HAILO_STREAM_INTERFACE_MIPI

enumerator HAILO_STREAM_INTERFACE_INTEGRATED

enumerator HAILO_STREAM_INTERFACE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_vstream_stats_flags_t

Virtual stream statistics flags

Values:

enumerator HAILO_VSTREAM_STATS_NONE

No stats

enumerator HAILO_VSTREAM_STATS_MEASURE_FPS

Measure vstream FPS

enumerator HAILO_VSTREAM_STATS_MEASURE_LATENCY

Measure vstream latency

enumerator HAILO_VSTREAM_STATS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_pipeline_elem_stats_flags_t

Pipeline element statistics flags

Values:

enumerator HAILO_PIPELINE_ELEM_STATS_NONE

No stats

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS

Measure element FPS

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY

Measure element latency

enumerator HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE

Measure element queue size

enumerator HAILO_PIPELINE_ELEM_STATS_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_nms_burst_type_t

Values:

enumerator HAILO_BURST_TYPE_NO_BURST

enumerator HAILO_BURST_TYPE_H8_PER_CLASS

enumerator HAILO_BURST_TYPE_H15_PER_CLASS

enumerator HAILO_BURST_TYPE_H15_PER_FRAME

enum hailo_power_mode_t

Power modes

Values:

enumerator HAILO_POWER_MODE_PERFORMANCE

enumerator HAILO_POWER_MODE_ULTRA_PERFORMANCE

enumerator HAILO_POWER_MODE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_latency_measurement_flags_t

Latency measurement flags

Values:

enumerator HAILO_LATENCY_NONE

enumerator HAILO_LATENCY_MEASURE

enumerator HAILO_LATENCY_CLEAR_AFTER_GET

enumerator HAILO_LATENCY_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_notification_id_t

Notification IDs, for each notification, one of the *hailo_notification_message_parameters_t* union will be set.

Values:

enumerator HAILO_NOTIFICATION_ID_ETHERNET_RX_ERROR

Matches *hailo_notification_message_parameters_t::rx_error_notification*.

enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_TEMPERATURE_ALARM

Matches *hailo_notification_message_parameters_t::health_monitor_temperature_alarm_notification*

enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_DATAFLOW_SHUTDOWN

Matches *hailo_notification_message_parameters_t::health_monitor_dataflow_shutdown_notification*

enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM

Matches *hailo_notification_message_parameters_t::health_monitor_overcurrent_alert_notification*

enumerator HAILO_NOTIFICATION_ID_LCU_ECC_CORRECTABLE_ERROR

Matches *hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification*

enumerator HAILO_NOTIFICATION_ID_LCU_ECC_UNCORRECTABLE_ERROR

Matches *hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification*

enumerator HAILO_NOTIFICATION_ID_CPU_ECC_ERROR

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

enumerator HAILO_NOTIFICATION_ID_CPU_ECC_FATAL

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

enumerator HAILO_NOTIFICATION_ID_DEBUG

Matches *hailo_notification_message_parameters_t::debug_notification*

enumerator HAILO_NOTIFICATION_ID_CONTEXT_SWITCH_BREAKPOINT_REACHED

Matches *hailo_notification_message_parameters_t::context_switch_breakpoint_reached_notification*

enumerator HAILO_NOTIFICATION_ID_HEALTH_MONITOR_CLOCK_CHANGED_EVENT

Matches *hailo_notification_message_parameters_t::health_monitor_clock_changed_notification*

enumerator HAILO_NOTIFICATION_ID_HW_INFER_MANAGER_INFER_DONE

Matches hailo_notification_message_parameters_t::hailo_hw_infer_manager_infer_done_notification

enumerator HAILO_NOTIFICATION_ID_COUNT

Must be last!

enumerator HAILO_NOTIFICATION_ID_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_temperature_protection_temperature_zone_t

Values:

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__GREEN

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__ORANGE

enumerator HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__RED

enum hailo_overcurrent_protection_overcurrent_zone_t

Values:

enumerator HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__GREEN

enumerator HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__RED

enum hailo_reset_device_mode_t

Hailo device reset modes

Values:

enumerator HAILO_RESET_DEVICE_MODE_CHIP

enumerator HAILO_RESET_DEVICE_MODE_NN_CORE

enumerator HAILO_RESET_DEVICE_MODE_SOFT

enumerator HAILO_RESET_DEVICE_MODE_FORCED_SOFT

enumerator HAILO_RESET_DEVICE_MODE_MAX_ENUM

enum hailo_watchdog_mode_t

Values:

enumerator HAILO_WATCHDOG_MODE_HW_SW

enumerator HAILO_WATCHDOG_MODE_HW_ONLY

enumerator HAILO_WATCHDOG_MODE_MAX_ENUM

enum hailo_sensor_types_t

Values:

enumerator HAILO_SENSOR_TYPES_GENERIC

enumerator HAILO_SENSOR_TYPES_ONSEMI_AR0220AT

enumerator HAILO_SENSOR_TYPES_RASPICAM

enumerator HAILO_SENSOR_TYPES_ONSEMI_AS0149AT

enumerator HAILO_SENSOR_TYPES_HAILO8_ISP

enumerator HAILO_SENSOR_TYPES_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_fw_logger_interface_t

Values:

enumerator HAILO_FW_LOGGER_INTERFACE_PCIE

enumerator HAILO_FW_LOGGER_INTERFACE_UART

enumerator HAILO_FW_LOGGER_INTERFACE_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_fw_logger_level_t

Values:

enumerator HAILO_FW_LOGGER_LEVEL_TRACE

enumerator HAILO_FW_LOGGER_LEVEL_DEBUG

enumerator HAILO_FW_LOGGER_LEVEL_INFO

enumerator HAILO_FW_LOGGER_LEVEL_WARN

enumerator HAILO_FW_LOGGER_LEVEL_ERROR

enumerator HAILO_FW_LOGGER_LEVEL_FATAL

enumerator HAILO_FW_LOGGER_LEVEL_MAX_ENUM

Max enum value to maintain ABI Integrity

typedef float float32_t

typedef double float64_t

typedef uint16_t nms_bbox_counter_t

typedef struct _hailo_device *hailo_device

Represents the device (chip)

typedef struct _hailo_vdevice *hailo_vdevice

Represents a virtual device which manages several physical devices

typedef struct _hailo_hef *hailo_hef

Compiled HEF model that can be loaded to Hailo devices

typedef struct _hailo_input_stream *hailo_input_stream

Input (host to device) stream representation

typedef struct _hailo_output_stream *hailo_output_stream

Output (device to host) stream representation

typedef struct _hailo_configured_network_group *hailo_configured_network_group

Loaded network_group that can be activated

typedef struct _hailo_activated_network_group *hailo_activated_network_group

Activated network_group that can be used to send/receive data

typedef struct _hailo_input_transform_context *hailo_input_transform_context

Object used for input stream transformation, store all necessary allocated buffers

typedef struct _hailo_output_transform_context *hailo_output_transform_context

Object used for output stream transformation, store all necessary allocated buffers

typedef struct _hailo_output_demuxer *hailo_output_demuxer

Object used to demux muxed stream

typedef struct _hailo_input_vstream *hailo_input_vstream

Input virtual stream

typedef struct _hailo_output_vstream *hailo_output_vstream

Output virtual stream

typedef enum *hailo_dvm_options_e* hailo_dvm_options_t

Enum that represents the type of devices that would be measured

typedef enum *hailo_power_measurement_types_e* hailo_power_measurement_types_t

Enum that represents what would be measured on the selected device

typedef enum *hailo_sampling_period_e* hailo_sampling_period_t

Enum that represents all the bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage

typedef enum *hailo_averaging_factor_e* hailo_averaging_factor_t

Enum that represents all the AVG bit settings and related number of averages for each bit setting

typedef enum *hailo_measurement_buffer_index_e* hailo_measurement_buffer_index_t

Enum that represents buffers on the device for power measurements storing

typedef struct _hailo_scan_devices_params_t `hailo_scan_devices_params_t`

Additional scan params, for future compatibility

typedef enum *hailo_scheduling_algorithm_e* `hailo_scheduling_algorithm_t`

Scheduler algorithm

typedef enum *hailo_device_architecture_e* `hailo_device_architecture_t`

Device architecture

typedef void (*`hailo_stream_write_async_callback_t`)(const *hailo_stream_write_async_completion_info_t* *info)

Async stream write complete callback prototype.

typedef void (*`hailo_stream_read_async_callback_t`)(const *hailo_stream_read_async_completion_info_t* *info)

Async stream read complete callback prototype.

typedef void (*`hailo_notification_callback`)(*hailo_device* device, const *hailo_notification_t* *notification, void *opaque)

A notification callback. See *hailo_set_notification_callback*

Warning: Throwing exceptions in the callback is not supported!

Param device **[in]** The *hailo_device* that got the notification.

Param notification **[in]** The notification data.

Param opaque **[in]** User specific data.

hailo_status `hailo_get_library_version`(*hailo_version_t* *version)

Retrieves hailort library version.

Parameters `version` – **[out]** Will be filled with hailort library version.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

const char *`hailo_get_status_message`(*hailo_status* status)

Returns a string format of @ status.

Parameters `status` – **[in]** A *hailo_status* to be converted to string format.

Returns Upon success, returns @ status as a string format. Otherwise, returns *nullptr*.

`HAILO_MAX_ENUM`

`HAILO_INFINITE`

`HAILO_DEFAULT_ETH_SCAN_TIMEOUT_MS`

`HAILO_DEFAULT_ETH_CONTROL_PORT`

`HAILO_DEFAULT_ETH_DEVICE_PORT`

HAILO_DEFAULT_ETH_MAX_PAYLOAD_SIZE

HAILO_DEFAULT_ETH_MAX_NUMBER_OF_RETRIES

HAILO_ETH_ADDRESS_ANY

HAILO_ETH_PORT_ANY

HAILO_MAX_NAME_SIZE

HAILO_MAX_STREAM_NAME_SIZE

HAILO_MAX_BOARD_NAME_LENGTH

HAILO_MAX_DEVICE_ID_LENGTH

HAILO_MAX_SERIAL_NUMBER_LENGTH

HAILO_MAX_PART_NUMBER_LENGTH

HAILO_MAX_PRODUCT_NAME_LENGTH

HAILO_DEFAULT_INIT_SAMPLING_PERIOD_US

HAILO_DEFAULT_INIT_AVERAGING_FACTOR

HAILO_DEFAULT_BUFFERS_THRESHOLD

HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC

HAILO_MAX_STREAMS_COUNT

HAILO_DEFAULT_BATCH_SIZE

HAILO_MAX_NETWORK_GROUPS

HAILO_MAX_NETWORK_GROUP_NAME_SIZE

HAILO_MAX_NETWORK_NAME_SIZE

HAILO_MAX_NETWORKS_IN_NETWORK_GROUP

HAILO_PCIE_ANY_DOMAIN

HAILO_DEFAULT_VSTREAM_QUEUE_SIZE

HAILO_DEFAULT_VSTREAM_TIMEOUT_MS

HAILO_DEFAULT_DEVICE_COUNT

HAILO_SOC_ID_LENGTH

HAILO_ETH_MAC_LENGTH

HAILO_UNIT_LEVEL_TRACKING_BYTES_LENGTH

HAILO_SOC_PM_VALUES_BYTES_LENGTH

HAILO_MAX_TEMPERATURE_THROTTLING_LEVELS_NUMBER

HAILO_UNIQUE_VDEVICE_GROUP_ID

HAILO_DEFAULT_VDEVICE_GROUP_ID

HAILO_SCHEDULER_PRIORITY_NORMAL

HAILO_SCHEDULER_PRIORITY_MAX

HAILO_SCHEDULER_PRIORITY_MIN

HAILO_STATUS_VARIABLES
HailoRT return codes

HAILO_STREAM_ABORTED

HAILO_STREAM_INTERNAL_ABORT

HAILO_DEFAULT_TRANSFORM_PARAMS

HAILO_DEFAULT_SOCKADDR

HAILO_ETH_INPUT_STREAM_PARAMS_DEFAULT

HAILO_ETH_OUTPUT_STREAM_PARAMS_DEFAULT

HAILO_PCIE_STREAM_PARAMS_DEFAULT

HAILO_MIPI_INPUT_STREAM_PARAMS_DEFAULT

HAILO_ACTIVATE_NETWORK_GROUP_PARAMS_DEFAULT

struct hailo_version_t
#include <hailort.h> HailoRT library version

Public Members

uint32_t `major`

uint32_t `minor`

uint32_t `revision`

struct `hailo_power_measurement_data_t`

#include <hailort.h> Data of the power measurement samples

Public Members

float32_t `average_value`

float32_t `average_time_value_milliseconds`

float32_t `min_value`

float32_t `max_value`

uint32_t `total_number_of_samples`

struct `hailo_eth_device_info_t`

#include <hailort.h> Ethernet device information

Public Members

struct sockaddr_in `host_address`

struct sockaddr_in `device_address`

uint32_t `timeout_millis`

uint8_t `max_number_of_attempts`

uint16_t `max_payload_size`

struct `hailo_pcie_device_info_t`

#include <hailort.h> PCIe device information

Public Members

uint32_t `domain`

uint32_t `bus`

uint32_t `device`

uint32_t `func`

struct `hailo_device_id_t`

#include <hailort.h> Hailo device ID string - BDF for PCIe devices, IP address for Ethernet devices.

Public Members

char `id`[(32)]

struct `hailo_vdevice_params_t`

#include <hailort.h> Virtual device parameters

Public Members

uint32_t `device_count`

Requested number of physical devices. if *device_ids* is not NULL represents the number of *hailo_device_id_t* in *device_ids*.

hailo_device_id_t *`device_ids`

Specific device ids to create the vdevice from. If NULL, the vdevice will try to occupy devices from the available pool.

hailo_scheduling_algorithm_t `scheduling_algorithm`

The scheduling algorithm to use for network group scheduling

const char *`group_id`

Key for using a shared VDevice. To create a unique VDevice, use HAILO_UNIQUE_VDEVICE_GROUP_ID

bool `multi_process_service`

Flag specifies whether to create the VDevice in HailoRT service or not. Defaults to false

struct `hailo_firmware_version_t`

#include <hailort.h> Hailo firmware version

Public Members

uint32_t `major`

uint32_t `minor`

uint32_t `revision`

struct `hailo_device_identity_t`

#include <hailort.h> Hailo device identity

Public Members

uint32_t `protocol_version`

hailo_firmware_version_t `fw_version`

uint32_t `logger_version`

uint8_t `board_name_length`

char `board_name`[(32)]

bool `is_release`

bool `extended_context_switch_buffer`

hailo_device_architecture_t `device_architecture`

uint8_t `serial_number_length`

char `serial_number`[(16)]

uint8_t `part_number_length`

char `part_number`[(16)]

uint8_t `product_name_length`

char `product_name`[(42)]

struct `hailo_core_information_t`

Public Members

bool `is_release`

bool `extended_context_switch_buffer`

hailo_firmware_version_t `fw_version`

struct `hailo_device_supported_features_t`

#include <hailort.h> Hailo device supported features

Public Members

bool `ethernet`

Is ethernet supported

bool `mipi`

Is mipi supported

bool `pcie`

Is pcie supported

bool `current_monitoring`

Is current monitoring supported

bool `mdio`

Is current mdio supported

struct `hailo_extended_device_information_t`

#include <hailort.h> Hailo extended device information

Public Members

uint32_t `neural_network_core_clock_rate`

The core clock rate

hailo_device_supported_features_t `supported_features`

Hailo device supported features

hailo_device_boot_source_t `boot_source`

Device boot source

uint8_t `soc_id`[(32)]

SOC id

uint8_t `lcs`

Device lcs

uint8_t `eth_mac_address`[(6)]

Device Ethernet Mac address

uint8_t `unit_level_tracking_id`[(12)]

Hailo device unit level tracking id

uint8_t `soc_pm_values`[(24)]

Hailo device pm values

struct `hailo_i2c_slave_config_t`

#include <hailort.h> I2C slave configuration

Public Members

hailo_endianness_t endianness

uint16_t slave_address

uint8_t register_address_size

uint8_t bus_index

bool should_hold_bus

struct hailo_fw_user_config_information_t

#include <hailort.h> Firmware user config information

Public Members

uint32_t version

uint32_t entry_count

uint32_t total_size

struct hailo_format_t

#include <hailort.h> Hailo data format

Public Members

hailo_format_type_t type

hailo_format_order_t order

hailo_format_flags_t flags

struct hailo_buffer_heap_params_t

#include <hailort.h> Hailo buffer heap parameters

Public Members

uint8_t reserved

struct hailo_buffer_dma_mapping_params_t

Public Members

hailo_device `device`

hailo_vdevice `vdevice`

hailo_dma_buffer_direction_t `direction`

struct `hailo_buffer_parameters_t`

#include <hailort.h> Hailo buffer parameters

Public Members

hailo_buffer_flags_t `flags`

hailo_buffer_heap_params_t `heap_params`

hailo_buffer_dma_mapping_params_t `dma_mapping_params`

union *hailo_buffer_parameters_t*::[anonymous] `[anonymous]`

struct `hailo_transform_params_t`

#include <hailort.h> Input or output data transform parameters

Public Members

hailo_stream_transform_mode_t `transform_mode`

hailo_format_t `user_buffer_format`

struct `hailo_demux_params_t`

#include <hailort.h> Demuxer params

Public Members

uint8_t `reserved`

struct `hailo_quant_info_t`

#include <hailort.h> Quantization information. Property of *hailo_stream_info_t*, *hailo_vstream_info_t* .

Hailo devices require input data to be quantized/scaled before it is sent. Similarly, data outputted from the device needs to be 'de-quantized'/rescaled as well. Each input/output layer is assigned two floating point values that are parameters to an input/output transformation: qp_zp (zero_point) and qp_scale. These values are stored in the HEF.

- Input transformation: Input data is divided by qp_scale and then qp_zp is added to the result.
- Output transformation: qp_zp is subtracted from output data and then the result is multiplied by qp_scale.

Public Members

float32_t `qp_zp`

zero_point

float32_t `qp_scale`

scale

float32_t `limvals_min`

min limit value

float32_t `limvals_max`

max limit value

struct `hailo_eth_input_stream_params_t`

#include <hailort.h> Ethernet input stream (host to device) parameters

Public Members

struct sockaddr_in `host_address`

port_t `device_port`

bool `is_sync_enabled`

uint32_t `frames_per_sync`

uint16_t `max_payload_size`

uint32_t `rate_limit_bytes_per_sec`

Stream may be rate limited by setting this member to te desired rate other than zero. The limition will only effect the corresponding stream (other network traffic won't be effected).

- On linux the "Traffic Control" tool will be used to limit the network rate (see `man tc`):
  - This will result in external processes being created at the creation and destruction of the stream
  - `sudo` privileges are required.
  - Alternatively, use the command line tool `hailortcli udp-rate-limiter`, which is also implemented using "Traffic Control". In this case this member must be set to zero.
- On Windows user-mode rate limiting (via a token-bucket) is used:
  - User-mode rate limiting is designed to consistently keep the stream at the desired rate, however fluctuations will occur. This member parameter provides an upper bound on the bandwidth at which the stream will operate.

uint32_t `buffers_threshold`

struct `hailo_eth_output_stream_params_t`

#include <hailort.h> Ethernet output stream (device to host) parameters

Public Members

struct sockaddr_in `host_address`

port_t `device_port`

bool `is_sync_enabled`

uint16_t `max_payload_size`

uint32_t `buffers_threshold`

struct `hailo_pcie_input_stream_params_t`

#include <hailort.h> PCIe input stream (host to device) parameters

Public Members

uint8_t `reserved`

struct `hailo_pcie_output_stream_params_t`

#include <hailort.h> PCIe output stream (device to host) parameters

Public Members

uint8_t `reserved`

struct `hailo_mipi_common_params_t`

#include <hailort.h> MIPI params

Public Members

uint16_t `img_width_pixels`

The width in pixels of the image that enter to the mipi CSI. The sensor output. When isp_enable and isp_crop_enable is false, is also the stream input.

uint16_t `img_height_pixels`

The height in pixels of the image that enter to the mipi CSI. The sensor output. When isp_enable and isp_crop_enable is false, is also the stream input.

hailo_mipi_pixels_per_clock_t `pixels_per_clock`

Number of pixels transmitted at each clock.

uint8_t `number_of_lanes`

Number of lanes to use.

hailo_mipi_clock_selection_t `clock_selection`

Selection of clock range that would be used. Setting *HAILO_MIPI_CLOCK_SELECTION_AUTOMATIC* means that the clock selection is calculated from the data rate.

uint8_t `virtual_channel_index`

The virtual channel index of the MIPI dphy.

uint32_t `data_rate`

Rate of the passed data (MHz).

struct `hailo_isp_params_t`

#include <hailort.h> ISP params

Public Members

hailo_mipi_isp_image_in_order_t `isp_img_in_order`

The ISP Rx bayer pixel order. Only relevant when the ISP is enabled.

hailo_mipi_isp_image_out_data_type_t `isp_img_out_data_type`

The data type that the mipi will take out. Only relevant when the ISP is enabled.

bool `isp_crop_enable`

Enable the crop feature in the ISP. Only relevant when the ISP is enabled.

uint16_t `isp_crop_output_width_pixels`

The width in pixels of the output window that the ISP take out. The stream input. Useful when isp_crop_enable is True. Only relevant when the ISP is enabled.

uint16_t `isp_crop_output_height_pixels`

The height in pixels of the output window that the ISP take out. The stream input. Useful when isp_crop_enable is True. Only relevant when the ISP is enabled.

uint16_t `isp_crop_output_width_start_offset_pixels`

The width start point of the output window that the ISP take out. Useful when isp_crop_enable is True. Only relevant when the ISP is enabled.

uint16_t `isp_crop_output_height_start_offset_pixels`

The height start point of the output window that the ISP take out. Useful when isp_crop_enable is True. Only relevant when the ISP is enabled.

bool `isp_test_pattern_enable`

Enable Test pattern from the ISP. Only relevant when the ISP is enabled.

bool `isp_configuration_bypass`

Don't load the ISP configuration file from the FLASH. Only relevant when the ISP is enabled.

bool `isp_run_time_ae_enable`

Enable the run-time Auto Exposure in the ISP. Only relevant when the ISP is enabled.

bool `isp_run_time_awb_enable`

Enable the run-time Auto White Balance in the ISP. Only relevant when the ISP is enabled.

bool `isp_run_time_adt_enable`

Enable the run-time Adaptive Function in the ISP. Only relevant when the ISP is enabled.

bool `isp_run_time_af_enable`

Enable the run-time Auto Focus in the ISP. Only relevant when the ISP is enabled.

uint16_t `isp_run_time_calculations_interval_ms`

Interval in milliseconds between ISP run time calculations. Only relevant when the ISP is enabled.

hailo_mipi_isp_light_frequency_t `isp_light_frequency`

Selection of the light frequency. This parameter varies depending on the power grid of the country where the product is running. Only relevant when the ISP is enabled.

struct `hailo_mipi_input_stream_params_t`

#include <hailort.h> MIPI input stream (host to device) parameters

Public Members

hailo_mipi_common_params_t `mipi_common_params`

uint8_t `mipi_rx_id`

Selection of which MIPI Rx device to use.

hailo_mipi_data_type_rx_t `data_type`

The data type which will be passed over the MIPI.

bool `isp_enable`

Enable the ISP block in the MIPI dataflow. The ISP is not supported yet.

hailo_isp_params_t `isp_params`

struct `hailo_integrated_input_stream_params_t`

#include <hailort.h> Core input stream (host to device) parameters

Public Members

uint8_t `reserved`

struct `hailo_integrated_output_stream_params_t`

#include <hailort.h> Core output stream (device to host) parameters

Public Members

uint8_t `reserved`

struct `hailo_stream_parameters_t`

#include <hailort.h> Hailo stream parameters

Public Members

hailo_stream_interface_t `stream_interface`

hailo_stream_direction_t `direction`

hailo_stream_flags_t `flags`

hailo_pcie_input_stream_params_t `pcie_input_params`

hailo_integrated_input_stream_params_t `integrated_input_params`

hailo_eth_input_stream_params_t `eth_input_params`

hailo_mipi_input_stream_params_t `mipi_input_params`

hailo_pcie_output_stream_params_t `pcie_output_params`

hailo_integrated_output_stream_params_t `integrated_output_params`

hailo_eth_output_stream_params_t `eth_output_params`

union *hailo_stream_parameters_t*::[anonymous] `[anonymous]`

struct `hailo_stream_parameters_by_name_t`

#include <hailort.h> Hailo stream parameters per stream_name

Public Members

char `name[(128)]`

hailo_stream_parameters_t `stream_params`

struct `hailo_vstream_params_t`

#include <hailort.h> Virtual stream params

Public Members

hailo_format_t `user_buffer_format`

uint32_t `timeout_ms`

uint32_t `queue_size`

hailo_vstream_stats_flags_t `vstream_stats_flags`

hailo_pipeline_elem_stats_flags_t `pipeline_elements_stats_flags`

struct `hailo_input_vstream_params_by_name_t`

#include <hailort.h> Input virtual stream parameters

Public Members

char name[((128))]

hailo_vstream_params_t params

struct hailo_output_vstream_params_by_name_t

#include <hailort.h> Output virtual stream parameters

Public Members

char name[((128))]

hailo_vstream_params_t params

struct hailo_output_vstream_name_by_group_t

#include <hailort.h> Output virtual stream name by group

Public Members

char name[((128))]

uint8_t pipeline_group_index

struct hailo_3d_image_shape_t

#include <hailort.h> Image shape

Public Members

uint32_t height

uint32_t width

uint32_t features

struct hailo_nms_defuse_info_t

Public Members

uint32_t class_group_index

char original_name[((128))]

struct hailo_nms_info_t

#include <hailort.h> NMS Internal HW Info

Public Members

uint32_t `number_of_classes`

Amount of NMS classes

uint32_t `max_bboxes_per_class`

Maximum amount of bboxes per nms class

uint32_t `bbox_size`

Internal usage

uint32_t `chunks_per_frame`

Internal usage

bool `is_defused`

hailo_nms_defuse_info_t `defuse_info`

uint32_t `burst_size`

Size of NMS burst in bytes

hailo_nms_burst_type_t `burst_type`

NMS burst type

struct `hailo_nms_fuse_input_t`

#include <hailort.h> NMS Fuse Input

Public Members

void *`buffer`

size_t `size`

hailo_nms_info_t `nms_info`

struct `hailo_nms_shape_t`

#include <hailort.h> Shape of nms result

Public Members

uint32_t `number_of_classes`

Amount of NMS classes

uint32_t `max_bboxes_per_class`

Maximum amount of bboxes per nms class

struct `hailo_bbox_t`

Public Members

uint16_t `y_min`

uint16_t `x_min`

uint16_t `y_max`

uint16_t `x_max`

uint16_t `score`

struct `hailo_bbox_float32_t`

Public Members

float32_t `y_min`

float32_t `x_min`

float32_t `y_max`

float32_t `x_max`

float32_t `score`

struct `hailo_stream_write_async_completion_info_t`

#include <*hailort.h*> Completion info struct passed to the *hailo_stream_write_async_callback_t* after the async operation is done or has failed.

Public Members

hailo_status `status`

Status of the async transfer:

- *HAILO_SUCCESS* - The transfer is complete.
- *HAILO_STREAM_ABORTED_BY_USER* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

const void *`buffer_addr`

Address of the buffer passed to the async operation

size_t `buffer_size`

Size of the buffer passed to the async operation.

void *`opaque`

User specific data. Can be used as a context for the callback.

struct `hailo_stream_read_async_completion_info_t`

#include <hailort.h> Completion info struct passed to the *hailo_stream_read_async_callback_t* after the async operation is done or has failed.

Public Members

hailo_status `status`

Status of the async transfer:

- *HAILO_SUCCESS* - The transfer is complete.
- *HAILO_STREAM_ABORTED_BY_USER* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

void *`buffer_addr`

Address of the buffer passed to the async operation

size_t `buffer_size`

Size of the buffer passed to the async operation.

void *`opaque`

User specific data. Can be used as a context for the callback.

struct `hailo_stream_info_t`

#include <hailort.h> Input or output stream information. In case of multiple inputs or outputs, each one has its own stream.

Public Members

hailo_3d_image_shape_t `shape`

hailo_3d_image_shape_t `hw_shape`

hailo_nms_info_t `nms_info`

union *hailo_stream_info_t*::[anonymous] `[anonymous]`

uint32_t `hw_data_bytes`

uint32_t `hw_frame_size`

hailo_format_t `format`

hailo_stream_direction_t `direction`

uint8_t `index`

char `name`[((128))]

hailo_quant_info_t `quant_info`

bool `is_mux`

struct `hailo_vstream_info_t`

#include <hailort.h> Input or output vstream information.

Public Members

char `name`[((128))]

char `network_name`[((((128)) + 1 + HAILO_MAX_NAME_SIZE)]

hailo_stream_direction_t `direction`

hailo_format_t `format`

hailo_3d_image_shape_t `shape`

hailo_nms_shape_t `nms_shape`

union *hailo_vstream_info_t*::[anonymous] `[anonymous]`

hailo_quant_info_t `quant_info`

struct `hailo_network_parameters_t`

Public Members

uint16_t `batch_size`

This parameter is only used in multi-context network_groups. User is advised to modify this (single network parameter) or hailo_configure_network_group_params_t batch size parameter. Not both. In case user wishes to work with the same batch size for all networks inside a network group, user is advised to set batch_size in hailo_configure_network_group_params_t. In case user wished to work with batch size per network, user is advised to use this parameter. Default network batch size is *HAILO_DEFAULT_BATCH_SIZE*

struct `hailo_network_parameters_by_name_t`

Public Members

char `name`[(((128)) + 1 + HAILO_MAX_NAME_SIZE)]

hailo_network_parameters_t `network_params`

struct `hailo_configure_network_group_params_t`

#include <hailort.h> Hailo configure parameters per network_group

Public Members

char name[((128))]

uint16_t batch_size

This parameter is only used in multi-context network_groups. In case of name missmatch, default value *HAILO_DEFAULT_BATCH_SIZE* is used

hailo_power_mode_t power_mode

hailo_latency_measurement_flags_t latency

size_t stream_params_by_name_count

hailo_stream_parameters_by_name_t stream_params_by_name[(40)]

size_t network_params_by_name_count

hailo_network_parameters_by_name_t network_params_by_name[(8)]

struct hailo_configure_params_t

#include <hailort.h> Hailo configure parameters

Public Members

size_t network_group_params_count

hailo_configure_network_group_params_t network_group_params[(8)]

struct hailo_activate_network_group_params_t

#include <hailort.h> Hailo network_group parameters

Public Members

uint8_t reserved

struct hailo_network_group_info_t

#include <hailort.h> Hailo network group info

Public Members

char name[((128))]

bool is_multi_context

struct hailo_layer_name_t

#include <hailort.h> Hailo layer name

Public Members

char name[((128))]

struct hailo_network_info_t

Public Members

char name[(((128)) + 1 + HAILO_MAX_NAME_SIZE)]

struct hailo_rx_error_notification_message_t

#include <hailort.h> Rx error notification message

Public Members

uint32_t error

uint32_t queue_number

uint32_t rx_errors_count

struct hailo_debug_notification_message_t

#include <hailort.h> Debug notification message

Public Members

uint32_t connection_status

uint32_t connection_type

uint32_t vdma_is_active

uint32_t host_port

uint32_t host_ip_addr

struct hailo_health_monitor_dataflow_shutdown_notification_message_t

#include <hailort.h> Health monitor - Dataflow shutdown notification message

Public Members

uint32_t closed_input_streams

Bit mask of closed input streams indices

uint32_t closed_output_streams

Bit mask of closed output streams indices

float32_t ts0_temperature

float32_t ts1_temperature

struct hailo_health_monitor_temperature_alarm_notification_message_t

#include <hailort.h> Health monitor - Temperature alarm notification message

Public Members

hailo_temperature_protection_temperature_zone_t temperature_zone

uint32_t alarm_ts_id

float32_t ts0_temperature

float32_t ts1_temperature

struct hailo_health_monitor_overcurrent_alert_notification_message_t

#include <hailort.h> Health monitor - Overcurrent alert notification message

Public Members

hailo_overcurrent_protection_overcurrent_zone_t overcurrent_zone

float32_t exceeded_alert_threshold

bool is_last_overcurrent_violation_reached

struct hailo_health_monitor_lcu_ecc_error_notification_message_t

#include <hailort.h> Health monitor - LCU ECC error notification message

Public Members

uint16_t cluster_error

struct hailo_health_monitor_cpu_ecc_notification_message_t

#include <hailort.h> Health monitor - CPU ECC error notification message

Public Members

uint32_t memory_bitmap

struct hailo_context_switch_breakpoint_reached_message_t

#include <hailort.h> Context switch - breakpoint reached notification message

Public Members

uint8_t network_group_index

uint16_t batch_index

uint8_t context_index

uint16_t action_index

struct `hailo_health_monitor_clock_changed_notification_message_t`

#include <hailort.h> Health monitor - System's clock has been changed notification message

Public Members

uint32_t `previous_clock`

uint32_t `current_clock`

struct `hailo_hw_infer_manager_infer_done_notification_message_t`

Public Members

uint32_t `infer_cycles`

union `hailo_notification_message_parameters_t`

#include <hailort.h> Union of all notification messages parameters. See *hailo_notification_t*

Public Members

hailo_rx_error_notification_message_t `rx_error_notification`

Ethernet rx error

hailo_debug_notification_message_t `debug_notification`

Internal usage

hailo_health_monitor_dataflow_shutdown_notification_message_t
`health_monitor_dataflow_shutdown_notification`

Dataflow shutdown due to health monitor event

hailo_health_monitor_temperature_alarm_notification_message_t
`health_monitor_temperature_alarm_notification`

Chip temperature alarm

hailo_health_monitor_overcurrent_alert_notification_message_t
`health_monitor_overcurrent_alert_notification`

Chip overcurrent alert

hailo_health_monitor_lcu_ecc_error_notification_message_t
`health_monitor_lcu_ecc_error_notification`

Core ecc error notification

hailo_health_monitor_cpu_ecc_notification_message_t `health_monitor_cpu_ecc_notification`

Chip ecc error notification

hailo_context_switch_breakpoint_reached_message_t
`context_switch_breakpoint_reached_notification`

Internal usage

hailo_health_monitor_clock_changed_notification_message_t
`health_monitor_clock_changed_notification`

Neural network core clock changed due to health monitor event

hailo_hw_infer_manager_infer_done_notification_message_t hw_infer_manager_infer_done_notification

struct hailo_notification_t

#include <hailort.h> Notification data that will be passed to the callback passed in *hailo_notification_callback*

Public Members

hailo_notification_id_t id

uint32_t sequence

hailo_notification_message_parameters_t body

struct hailo_chip_temperature_info_t

Public Members

float32_t ts0_temperature

float32_t ts1_temperature

uint16_t sample_count

struct hailo_throttling_level_t

Public Members

float32_t temperature_threshold

float32_t hysteresis_temperature_threshold

uint32_t throttling_nn_clock_freq

struct hailo_health_info_t

Public Members

bool overcurrent_protection_active

uint8_t current_overcurrent_zone

float32_t red_overcurrent_threshold

bool overcurrent_throttling_active

bool temperature_throttling_active

uint8_t current_temperature_zone

int8_t current_temperature_throttling_level

hailo_throttling_level_t temperature_throttling_levels[(4)]

int32_t orange_temperature_threshold

int32_t orange_hysteresis_temperature_threshold

int32_t red_temperature_threshold

int32_t red_hysteresis_temperature_threshold

uint32_t requested_overcurrent_clock_freq

uint32_t requested_temperature_clock_freq

struct hailo_stream_raw_buffer_t

Public Members

void *buffer

size_t size

struct hailo_stream_raw_buffer_by_name_t

Public Members

char name[((128))]

hailo_stream_raw_buffer_t raw_buffer

struct hailo_latency_measurement_result_t

Public Members

float64_t avg_hw_latency_ms

struct hailo_rate_limit_t

Public Members

char stream_name[((128))]

uint32_t rate

13. HailoRT C++ API Reference

HailoRT is Hailo's runtime library. The C++ API is used for running inference over compiled models (HEFs) in a C++ program.

Note: *(for C++ API users)* In order to maintain ABI Integrity between libhailort and your application that uses the C++ API, you should *compile libhailort* instead of using the pre-compiled binaries.

13.1. Device and control API functions

class `hailort::Device`

Represents the Hailo device (chip).

Public Types

enum `Type`

The device type

Values:

enumerator `PCIE`

enumerator `ETH`

enumerator `INTEGRATED`

Public Functions

Expected<NetworkGroupsParamsMap> `create_configure_params`(*Hef* &hef) const

Create the default configure params from an hef.

Parameters `hef` – **[in]** A reference to an *Hef* object to create configure params by

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*ConfigureNetworkParams*> `create_configure_params`(*Hef* &hef, const std::string ...)

Create the default configure params from an hef.

Parameters

- `hef` – **[in]** A reference to an *Hef* object to create configure params by
- `network_group_name` – **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<ConfiguredNetworkGroupVector> `configure`(*Hef* &hef, const ...)

Configure the device from an hef.

Parameters

- `hef` – **[in]** A reference to an *Hef* object to configure the device by.
- `configure_params` – **[in]** A map of configured network group name and parameters.

Returns Upon success, returns *Expected* of a vector of configured network groups. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<size_t> `read_log` (MemoryView &buffer, *hailo_cpu_id_t* cpu_id) = 0

Read data from the debug log buffer.

Parameters

- `buffer` – **[in]** A buffer that would receive the debug log data.
- `cpu_id` – **[in]** The cpu source of the debug log.

Returns Upon success, returns *Expected* of the number of bytes that were read. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_device_identity_t*> `identify()`

Sends identify control to a Hailo device.

Returns Upon success, returns *Expected* of *hailo_device_identity_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_core_information_t*> `core_identify()`

Receive information about the core cpu.

Returns Upon success, returns *Expected* of *hailo_core_information_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_extended_device_information_t*> `get_extended_device_information()`

Get extended device information about the Hailo device.

Returns Upon success, returns *Expected* of *hailo_extended_device_information_t* containing the extended information about the device. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `set_fw_logger` (*hailo_fw_logger_level_t* level, uint32_t interface_mask)

Configure fw logger level and interface of sending.

Parameters

- `level` – **[in]** The minimum logger level.
- `interface_mask` – **[in]** Output interfaces (mix of *hailo_fw_logger_interface_t*).

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `set_throttling_state` (bool should_activate)

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters `should_activate` – **[in]** Should be true to enable or false to disable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `write_memory` (uint32_t address, const MemoryView &data)

Writes data to device memory.

Parameters

- `address` – **[in]** The address data would be written to.
- `data` – **[in]** A buffer that contains the data to be written to the memory.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `read_memory` (uint32_t address, MemoryView &data)

Reads data from device memory.

Parameters

- `address` – **[in]** The address data would be read from.

- `data` – **[in]** A buffer that receives the data read from memory.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<bool> `get_throttling_state()`

Get current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Returns Upon success, returns *Expected* of *bool*, indicates weather the throttling state is active or not. Otherwise, returns *Unexpected* of *hailo_status* error.

hailo_status `wd_enable`(*hailo_cpu_id_t* cpu_id)

Enable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to enable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `wd_disable`(*hailo_cpu_id_t* cpu_id)

Disable firmware watchdog.

Note: Advanced API. Please use with care.

Parameters `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to disable.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `wd_config`(*hailo_cpu_id_t* cpu_id, uint32_t wd_cycles, *hailo_watchdog_mode_t* wd_mode)

Configure firmware watchdog.

Note: Advanced API. Please use with care.

Parameters

- `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU WD to configure.
- `wd_cycles` – **[in]** Number of cycles until watchdog is triggered.
- `wd_mode` – **[in]** A *hailo_watchdog_mode_t* indicating which WD mode to configure.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Expected<uint32_t> `previous_system_state`(*hailo_cpu_id_t* cpu_id)

Read the FW previous system state.

Note: Advanced API. Please use with care.

Parameters `cpu_id` – **[in]** A *hailo_cpu_id_t* indicating which CPU to state to read.

Returns Upon success, returns *Expected* of *uint32_t* indicating the previous system state. 0 indicating external reset, 1 indicating WD HW reset, 2 indicating WD SW reset, 3 indicating SW control reset. Otherwise, returns an *hailo_status* error.

hailo_status `set_pause_frames` (bool rx_pause_frames_enable)

Enable/Disable Pause frames.

Parameters `rx_pause_frames_enable` – **[in]** Indicating weather to enable or disable pause frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `i2c_read` (const *hailo_i2c_slave_config_t* &slave_config, uint32_t register_address, ...)

Read data from an I2C slave.

Parameters

- `slave_config` – **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- `register_address` – **[in]** The address of the register from which the data will be read.
- `data` – **[in]** A buffer that would store the read data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

hailo_status `i2c_write` (const *hailo_i2c_slave_config_t* &slave_config, uint32_t register_address, const ...)

Write data to an I2C slave.

Parameters

- `slave_config` – **[in]** The *hailo_i2c_slave_config_t* configuration of the slave.
- `register_address` – **[in]** The address of the register to which the data will be written.
- `data` – **[in]** A buffer that contains the data to be written to the slave.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Expected<*float32_t*> `power_measurement` (*hailo_dvm_options_t* dvm, ...)

Perform a single power measurement.

Parameters

- `dvm` – **[in]** Which DVM will be measured. Default (*HAILO_DVM_OPTIONS_AUTO*) will be different according to the board:
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums *HAILO_DVM_OPTIONS_VDD_CORE*, *HAILO_DVM_OPTIONS_MIPI_AVDD* and *HAILO_DVM_OPTIONS_AVDD_H*. Only *HAILO_POWER_MEASUREMENT_TYPES__POWER* can measured with this option.
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- `measurement_type` – **[in]** The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES__AUTO* will select the default value according to the supported features.

Returns Upon success, returns *uint32_t* mesuremenet. Measured units are determined due to *hailo_power_measurement_types_t*. Otherwise, returns a *hailo_status* error.

hailo_status `start_power_measurement` (*hailo_averaging_factor_t* averaging_factor, ...)

Start performing a long power measurement.

Parameters

- `averaging_factor` – **[in]** Number of samples per time period, sensor configuration value.

- `sampling_period` – **[in]** Related conversion time, sensor configuration value. The sensor samples the power every sampling_period {ms} and averages every averaging_factor samples. The sensor provides a new value every: 2 * sampling_period * averaging_factor {ms}. The firmware wakes up every interval_milliseconds {ms} and checks the sensor. If there is a new value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is 'average of averages', because it averages values that have already been averaged by the sensor.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `set_power_measurement` (*hailo_measurement_buffer_index_t* buffer_index, ...)

Set parameters for long power measurement.

Parameters

- `buffer_index` – **[in]** A *hailo_measurement_buffer_index_t* represents the buffer on the firmware the data would be saved at. Should match the one passed to '*Device::get_power_measurement*'.
- `dvm` – **[in]** Which DVM will be measured. Default (*HAILO_DVM_OPTIONS_AUTO*) will be different according to the board:
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums *HAILO_DVM_OPTIONS_VDD_CORE*, *HAILO_DVM_OPTIONS_MIPI_AVDD* and *HAILO_DVM_OPTIONS_AVDD_H*. Only *HAILO_POWER_MEASUREMENT_TYPES__POWER* can measured with this option.
  - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- `measurement_type` – **[in]** The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES__AUTO* will select the default value according to the supported features.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<*hailo_power_measurement_data_t*> `get_power_measurement` (*hailo_measurement_buffer_index_t* ...)

Read measured power from a long power measurement

Parameters

- `buffer_index` – **[in]** A *hailo_measurement_buffer_index_t* represents the buffer on the firmware the data would be saved at. Should match the one passed to '*Device::set_power_measurement*'.
- `should_clear` – **[in]** Flag indicating if the results saved at the firmware will be deleted after reading.

Returns Upon success, returns hailo_power_measurement_data_t. Measured units are determined due to *hailo_power_measurement_types_t* passed to '*Device::set_power_measurement*'. Otherwise, returns a *hailo_status* error.

hailo_status `stop_power_measurement` ( )

Stop performing a long power measurement.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<*hailo_chip_temperature_info_t*> `get_chip_temperature` ( )

Get temperature information on the device

Note: Temperature in Celsius of the 2 internal temperature sensors (TS).

Returns Upon success, returns hailo_chip_temperature_info_t, containing temperature information on the device. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `reset`(*hailo_reset_device_mode_t* mode) = 0

Reset device.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters `mode` – **[in]** The mode of the reset.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `set_notification_callback`(const *NotificationCallback* &func, ...)

Sets a callback to be called when a notification with ID *notification_id* will be received.

Parameters

- `func` – **[in]** The callback function to be called.
- `notification_id` – **[in]** The ID of the notification.
- `opaque` – **[in]** User specific data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `remove_notification_callback`(*hailo_notification_id_t* notification_id) = 0

Removes a previously set callback with ID *notification_id*.

Parameters `notification_id` – **[in]** The ID of the notification to remove.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `test_chip_memories`( )

Test chip memories using BIST.

Note: Cannot be called during inference!

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `set_sleep_state`(hailo_sleep_state_t sleep_state)

Set chip sleep state..

Note: This is an advanced API. Please be advised not to use this API, unless supported by Hailo.

Parameters `sleep_state` – **[in]** The requested sleep state of the chip

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `firmware_update`(const MemoryView &firmware_binary, bool should_reset) = 0

Update the firmware of a Hailo device.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters

- `firmware_binary` – **[in]** The firmware code to be updated to the device.
- `should_reset` – **[in]** Bool indicating weather to reset the device after updating.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `second_stage_update` (uint8_t *second_stage_binary, uint32_t ...)

Update the second stage binary.

Note: Calling this function while running other operations on the device (including inference) will lead to unexpected results!

Note: The object used to call this function is not to be used after calling this function! A new instance should be created.

Parameters

- `second_stage_binary` – **[in]** The SSB code to be updated to the device.
- `second_stage_binary_length` – **[in]** The length of the SSB to be updated.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `store_sensor_config` (uint32_t section_index, *hailo_sensor_types_t* ...)

Store sensor configuration to Hailo chip flash memory.

Parameters

- `section_index` – **[in]** Flash section index to write to. [0-6]
- `sensor_type` – **[in]** Sensor type.
- `reset_config_size` – **[in]** Size of reset configuration.
- `config_height` – **[in]** Configuration resolution height.
- `config_width` – **[in]** Configuration resolution width.
- `config_fps` – **[in]** Configuration FPS.
- `config_file_path` – **[in]** Sensor configuration file path.
- `config_name` – **[in]** Sensor configuration name.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `store_isp_config` (uint32_t reset_config_size, uint16_t config_height, uint16_t ...)

Store sensor ISP configuration to Hailo chip flash memory.

Parameters

- `reset_config_size` – **[in]** Size of reset configuration.
- `config_height` – **[in]** Configuration resolution height.
- `config_width` – **[in]** Configuration resolution width.
- `config_fps` – **[in]** Configuration FPS.

- `isp_static_config_file_path` – **[in]** ISP static configuration file path.
- `isp_runtime_config_file_path` – **[in]** ISP runtime configuration file path.
- `config_name` – **[in]** Sensor configuration name.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<Buffer> `sensor_get_sections_info()` = 0

Gets the sections information of the sensor.

Returns Upon success, returns *Expected* of a buffer containing the sensor's sections information. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `sensor_dump_config`(uint32_t section_index, const std::string ...)

Dump config of given section index into a csv file.

Parameters

- `section_index` – **[in]** Flash section index to load config from. [0-7]
- `config_file_path` – **[in]** File path to dump section configuration into.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `sensor_set_i2c_bus_index`(*hailo_sensor_types_t* sensor_type, uint32_t ...)

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- `sensor_type` – **[in]** The sensor type.
- `bus_index` – **[in]** The I2C bus index of the sensor.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `sensor_load_and_start_config`(uint32_t section_index) = 0

Load the configuration with I2C in the section index.

Parameters `section_index` – **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `sensor_reset`(uint32_t section_index) = 0

Reset the sensor that is related to the section index config.

Parameters `section_index` – **[in]** Flash section index to load config from. [0-6]

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `sensor_set_generic_i2c_slave`(uint16_t slave_address, uint8_t ...)

Set a generic I2C slave for sensor usage.

Parameters

- `slave_address` – **[in]** The address of the i2c slave.
- `offset_size` – **[in]** Slave offset size (in bytes).
- `bus_index` – **[in]** The bus number the i2c slave is connected to.
- `should_hold_bus` – **[in]** Should hold the bus during the read.
- `slave_endianness` – **[in]** BIG_ENDIAN or LITTEL_ENDIAN.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<Buffer> `read_board_config()` = 0

Reads board configuration from device.

Returns Upon success, returns *Expected* of a buffer containing the data read. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `write_board_config`(const MemoryView &buffer) = 0

Write board configuration to device

Parameters `buffer` – **[in]** A buffer that contains the data to be written .

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<*hailo_fw_user_config_information_t*> `examine_user_config`() = 0

Gets firmware user configuration information from device.

Returns Upon success, returns *Expected* of *hailo_fw_user_config_information_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<Buffer> `read_user_config`() = 0

Reads firmware user configuration from device.

Returns Upon success, returns *Expected* of a buffer containing the data read. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `write_user_config`(const MemoryView &buffer) = 0

Write firmware user configuration to device.

Parameters `buffer` – **[in]** A buffer that contains the data to be written .

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `erase_user_config`() = 0

Erase firmware user configuration from the device.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<*hailo_device_architecture_t*> `get_architecture`() const = 0

Gets the device architecture.

Returns Upon success, returns *Expected* of *hailo_device_architecture_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Type `get_type`() const

Gets the device type.

Returns Upon success, returns Type. Otherwise, returns a *hailo_status* error.

virtual const char *`get_dev_id`() const = 0

Gets the device id.

Returns An identification string of the device. For Pcie device, returns the BDF. For Ethernet device, returns the IP address. For Core device, returns “Core”.

Expected<*hailo_stream_interface_t*> `get_default_streams_interface`() const

Gets the stream’s default interface.

Returns Upon success, returns *Expected* of *hailo_stream_interface_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual bool `is_stream_interface_supported`(const *hailo_stream_interface_t* ...)

Returns true if the stream’s interface is supported, false otherwise.

Public Static Functions

static *Expected*<std::vector<std::string» `scan()`

Returns the device_id string on all available devices in the system. The device id is a unique identitier for the device on the system.

Note: ethernet devices are not considered “devices in the system”, so they are not scanned in this function. use :scan_eth for ethernet devices.

Returns Upon success, returns *Expected* of a vector of std::string containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::vector<*hailo_pcie_device_info_t*» `scan_pcie()`

Returns information on all available pcie devices in the system.

Returns Upon success, returns *Expected* of a vector of *hailo_pcie_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::vector<*hailo_eth_device_info_t*» `scan_eth(`const std::string &interface_name, ... `)`

Returns information on all available ethernet devices in the system.

Parameters

- `interface_name` – **[in]** The name of the network interface to scan.
- `timeout` – **[in]** The time in milliseconds to scan devices.

Returns Upon success, returns *Expected* of a vector of *hailo_eth_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::vector<*hailo_eth_device_info_t*» `scan_eth_by_host_address(`const ... `)`

Scans ethernet device by host address.

Parameters

- `host_address` – **[in]** The IP address of the network interface to scan.
- `timeout` – **[in]** The time in milliseconds to scan devices.

Returns Upon success, returns *Expected* of a vector of *hailo_eth_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*» `create()`

Creates a device. If there are more than one device detected in the system, an arbitrary device is returned.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*» `create(`const std::string &device_id `)`

Creates a device by the given device id.

Parameters `device_id` – **[in]** *Device* id string, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx)

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*» `create_pcie()`

Creates pcie device. If there are more than one device detected in the system, an arbitrary pcie device is returned.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*>> `create_pcie`(const *hailo_pcie_device_info_t* &device_info)

Creates a PCIe device by the given info.

Parameters `device_info` – **[in]** Information about the device to open.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*>> `create_eth`(const *hailo_eth_device_info_t* &device_info)

Creates an ethernet device by the given info.

Parameters `device_info` – **[in]** Information about the device to open.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*>> `create_eth`(const std::string &ip_addr)

Creates an ethernet device by IP address.

Parameters `ip_addr` – **[in]** The device IP address.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*Device*>> `create_eth`(const std::string &device_address, uint16_t port, ...)

Creates an ethernet device by IP address, port number, timeout duration and max number of attempts

Parameters

- `device_address` – **[in]** The device IP address.
- `port` – **[in]** The port number that the device will use for the Ethernet communication.
- `timeout_milliseconds` – **[in]** The time in milliseconds to scan devices.
- `max_number_of_attempts` – **[in]** The number of attempts to find a device.

Returns Upon success, returns *Expected* of a unique_ptr to *Device* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<*hailo_pcie_device_info_t*> `parse_pcie_device_info`(const std::string ...)

Parse PCIe device BDF string into hailo device info structure.

Parameters `device_info_str` – **[in]** BDF device info, format [<domain>].<bus>.<device>.<func>, same format as in lspci.

Returns Upon success, returns *Expected* of *hailo_pcie_device_info_t* containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::string> `pcie_device_info_to_string`(const *hailo_pcie_device_info_t* ...)

Returns a string of pcie device info.

Parameters `device_info` – **[in]** A *hailo_pcie_device_info_t* containing the pcie device information.

Returns Upon success, returns *Expected* of a string containing the information. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<*Type*> `get_device_type`(const std::string &device_id)

Returns the device type of the given device id string.

Parameters `device_id` – **[in]** A std::string device id to check.

Returns Upon success, returns *Expected* of the device type. Otherwise, returns *Unexpected* of *hailo_status* error.

static bool `device_ids_equal` (const std::string &first, const std::string &second)

Checks if 2 device ids represents the same device.

Parameters

- `first` – **[in]** A std::string first device id to check.
- `second` – **[in]** A std::string second device id to check.

Returns true if the device ids represents the same device.

13.2. VDevice API functions

class `hailort::VDevice`

Represents a bundle of physical devices.

Public Functions

virtual *Expected*<ConfiguredNetworkGroupVector> `configure` (*Hef* &hef, const ...)

Configure the vdevice from an hef.

Parameters

- `hef` – **[in]** A reference to an *Hef* object to configure the vdevice by.
- `configure_params` – **[in]** A map of configured network group name and parameters.

Returns Upon success, returns *Expected* of a vector of configured network groups. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::reference_wrapper<*Device*>>> `get_physical_devices` ( ) const = 0

Gets the underlying physical devices.

Note: The returned physical devices are held in the scope of *vdevice*.

Returns Upon success, returns *Expected* of a vector of device objects. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::string» `get_physical_devices_ids` ( ) const = 0

Gets the physical device IDs.

Returns Upon success, returns *Expected* of a vector of std::string device ids objects. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<*hailo_stream_interface_t*> `get_default_streams_interface` ( ) const = 0

Gets the stream's default interface.

Returns Upon success, returns *Expected* of *hailo_stream_interface_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<NetworkGroupsParamsMap> `create_configure_params` (*Hef* &hef) const

Create the default configure params from an hef.

Parameters `hef` – **[in]** A reference to an *Hef* object to create configure params by

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*ConfigureNetworkParams*> `create_configure_params` (*Hef* &hef, const std::string ...)

Create the default configure params from an hef.

Parameters

- `hef` – **[in]** A reference to an *Hef* object to create configure params by
- `network_group_name` – **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of a NetworkGroupsParamsMap (map of string and ConfiguredNetworkParams). Otherwise, returns *Unexpected* of *hailo_status* error.

Public Static Functions

static *Expected*<std::unique_ptr<*VDevice*>» `create` (const *hailo_vdevice_params_t* &params)

Creates a vdevice.

Parameters `params` – **[in]** A hailo_vdevice_params_t.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*VDevice*>» `create` ( )

Creates a vdevice.

Note: calling this create method will apply default vdevice params.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*VDevice*>» `create` (const std::vector<std::string> &device_ids)

Creates a vdevice from the given phyiscal device ids.

Note: calling this create method will apply default vdevice params.

Returns Upon success, returns *Expected* of a unique_ptr to *VDevice* object. Otherwise, returns *Unexpected* of *hailo_status* error.

13.3. HEF API defines and functions

struct `hailort::ConfigureNetworkParams`

Hailo configure parameters per network_group. Analogical to *hailo_configure_network_group_params_t*

Public Functions

`ConfigureNetworkParams` ( ) = default

inline `ConfigureNetworkParams` (const *hailo_configure_network_group_params_t* &params)

bool `operator==` (const *ConfigureNetworkParams* &other) const

bool `operator!=` (const *ConfigureNetworkParams* &other) const

Public Members

uint16_t `batch_size`

hailo_power_mode_t `power_mode`

hailo_latency_measurement_flags_t `latency`

std::map<std::string, *hailo_stream_parameters_t*> `stream_params_by_name`

std::map<std::string, *hailo_network_parameters_t*> `network_params_by_name`

class `hailort::Hef`

HEF model that can be loaded to Hailo devices

Public Functions

Expected<std::vector<*hailo_stream_info_t*>> `get_input_stream_infos`(const std::string &name = "")

Gets input streams informations.

Parameters `name` – **[in]** The name of the network or network_group which contains the input stream_infos. In case network group name is given, the function returns the input stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the input stream infos of the given network. If NULL is passed, the function returns the input stream infos of all the networks of the first network group.

Returns Upon success, returns a vector of *hailo_stream_info_t*, containing each stream's information. Otherwise, returns a *hailo_status* error.

Expected<std::vector<*hailo_stream_info_t*>> `get_output_stream_infos`(const std::string &name = "")

Gets output streams informations.

Parameters `name` – **[in]** The name of the network or network_group which contains the output stream_infos. In case network group name is given, the function returns the output stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the output stream infos of the given network. If NULL is passed, the function returns the output stream infos of all the networks of the first network group.

Returns Upon success, returns a vector of *hailo_stream_info_t*, containing each stream's information. Otherwise, returns a *hailo_status* error.

Expected<std::vector<*hailo_stream_info_t*>> `get_all_stream_infos`(const std::string &name = "")

Gets all streams informations.

Parameters `name` – **[in]** The name of the network or network_group which contains the stream_infos. In case network group name is given, the function returns all stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns all stream infos of the given network. If NULL is passed, the function returns all the stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_stream_info_t*, containing each stream's informmation. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_stream_info_t*> `get_stream_info_by_name`(const std::string &stream_name, ...)

Gets stream's information from it's name.

Parameters

- `stream_name` – **[in]** The name of the stream as presented in the *Hef*.
- `stream_direction` – **[in]** Indicates the stream direction.
- `net_group_name` – **[in]** The name of the network_group which contains the stream's information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of *hailo_stream_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<*hailo_vstream_info_t*» `get_input_vstream_infos` (const std::string &name = "")

Gets input virtual streams infos.

Parameters `name` – **[in]** The name of the network or network_group which contains the input virtaul stream_infos. In case network group name is given, the function returns the input virtual stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the input virtual stream infos of the given network. If NULL is passed, the function returns the input virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_vstream_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<*hailo_vstream_info_t*» `get_output_vstream_infos` (const std::string &name ...)

Gets output virtual streams infos.

Parameters `name` – **[in]** The name of the network or network_group which contains the output virtual stream_infos. In case network group name is given, the function returns the output virtual stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the output virtual stream infos of the given network. If NULL is passed, the function returns the output virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_vstream_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<*hailo_vstream_info_t*» `get_all_vstream_infos` (const std::string &name = "")

Gets all virtual streams infos.

Parameters `name` – **[in]** The name of the network or network_group which contains the virtual stream_infos. In case network group name is given, the function returns all virtual stream infos of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns all virtual stream infos of the given network. If NULL is passed, the function returns all the virtual stream infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_vstream_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<std::string» `get_sorted_output_names` (const std::string ...)

Gets sorted output vstreams names.

Parameters `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a sorted vector of output vstreams names. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<size_t> `get_number_of_input_streams` (const std::string &net_group_name = "")

Gets the number of low-level input streams.

Parameters `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* containing the number of low-level input streams. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<size_t> `get_number_of_output_streams` (const std::string &net_group_name = "")

Gets the number of low-level output streams.

Parameters `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* containing the number of low-level output streams. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*float64_t*> `get_bottleneck_fps` (const std::string &net_group_name = "")

Gets bottleneck FPS.

Parameters `net_group_name` – **[in]** The name of the network_group which contains the information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* containing the bottleneck FPS number. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*hailo_device_architecture_t*> `get_hef_device_arch` ( )

Get device Architecture HEF was compiled for.

Returns Upon success, returns *Expected* containing the device architecture the HEF was compiled for. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<std::string» `get_stream_names_from_vstream_name` (const std::string ...)

Gets all stream names under the given vstream name

Parameters

- `vstream_name` – **[in]** The name of the vstream.
- `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a vector of all stream names linked to the provided vstream. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<std::string» `get_vstream_names_from_stream_name` (const std::string ...)

Get all vstream names under the given stream name

Parameters

- `stream_name` – **[in]** The name of the low-level stream.
- `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a vector of all stream names linked to the provided vstream.

Returns Upon success, returns of a vector of all vstream names linked to the provided stream. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::string> `get_vstream_name_from_original_name` (const std::string ...)

Gets vstream name from original layer name.

Parameters

- `original_name` – **[in]** The original layer name as presented in the *Hef*.
- `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* containing the vstream's name. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<std::string>> `get_original_names_from_vstream_name`(const ...)

Gets original names from vstream name.

Parameters

- `vstream_name` – **[in]** The name of the vstream as presented in the *Hef*.
- `net_group_name` – **[in]** The name of the network_group which contains the streams information. If not passed, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a vector of all original names linked to the provided vstream. Otherwise, returns *Unexpected* of *hailo_status* error.

std::vector<std::string> `get_network_groups_names`()

Gets all network groups names in the *Hef*.

Returns Returns a vector of all network groups names.

Expected<std::vector<*hailo_network_group_info_t*>> `get_network_groups_infos`()

Gets all network groups infos in the *Hef*.

Returns Upon success, returns *Expected* of a vector of *hailo_network_group_info_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<NetworkGroupsParamsMap> `create_configure_params`(*hailo_stream_interface_t* ...)

Creates the default configure params for the *Hef*. The user can modify the given params before configuring the network.

Parameters `stream_interface` – **[in]** Stream interface to use on the network.

Returns Upon success, returns *Expected* of NetworkGroupsParamsMap. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*ConfigureNetworkParams*> `create_configure_params`(*hailo_stream_interface_t* ...)

Creates the default configure params for the *Hef*. The user can modify the given params before configuring the network.

Parameters

- `stream_interface` – **[in]** Stream interface to use on the network.
- `network_group_name` – **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of *ConfigureNetworkParams*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<NetworkGroupsParamsMap> `create_configure_params_mipi_input`(*hailo_stream_interface_t* ...)

Creates the default configure params for the *Hef*, where all inputs stream params are init to be MIPI type. The user can modify the given params before configuring the network.

Parameters

- `output_interface` – **[in]** Stream interface to use on the output streams.
- `mipi_params` – **[in]** Specific mipi params.

Returns Upon success, returns *Expected* of NetworkGroupsParamsMap, which maps network group name to configured network group params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<*ConfigureNetworkParams*> `create_configure_params_mipi_input`(*hailo_stream_interface_t* ...)

Creates the default configure params for the *Hef*, where all inputs stream params are init to be MIPI type. The user can modify the given params before configuring the network.

Parameters

- `output_interface` – **[in]** Stream interface to use on the output streams.

- `mipi_params` – **[in]** Specific mipi params
- `network_group_name` – **[in]** Name of network_group to make configure params for.

Returns Upon success, returns *Expected* of *ConfigureNetworkParams*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_stream_parameters_t*» `create_stream_parameters_by_name(const ...)`

Creates streams params with default values.

Parameters

- `net_group_name` – **[in]** The name of the network_group for which to create the stream parameters for. If an empty string is given, the first network_group in the *Hef* will be addressed.
- `stream_interface` – **[in]** A *hailo_stream_interface_t* indicating which *hailo_stream_parameters_t* to create for the output streams.

Returns Upon success, returns *Expected* of a map of stream name to stream params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_network_parameters_t*» `create_network_parameters_by_name(const ...)`

Creates networks params with default values.

Parameters `net_group_name` – **[in]** The name of the network_group for which to create the network parameters for. If an empty string is given, the first network_group in the *Hef* will be addressed.

Returns Upon success, returns *Expected* of a map of network name to network params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_stream_parameters_t*» `create_stream_parameters_by_name_mipi_input(`

Creates MIPI input stream params.

Parameters

- `net_group_name` – **[in]** The name of the network_group for which to create the stream parameters for. If an empty string is given, the first network_group in the *Hef* will be addressed.
- `output_interface` – **[in]** A hailo_stream_interface_t indicating which *hailo_stream_parameters_t* to create for the input streams params.
- `mipi_params` – **[in]** A *hailo_mipi_input_stream_params_t* object which contains the MIPI params.

Returns Upon success, returns *Expected* of a map of input stream name to stream params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_vstream_params_t*» `make_input_vstream_params(const ...)`

Creates input virtual stream params for a given network_group.

Parameters

- `name` – **[in]** The name of the network or network_group which user wishes to create input virtual stream params for. In case network group name is given, the function returns the input virtual stream params of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the input virtual stream params of the given network. If NULL is passed, the function returns the input virtual stream params of all the networks of the first network group.
- `quantized` – **[in]** Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all input virtual streams.

- `timeout_ms` – **[in]** The default timeout in milliseconds for all input virtual streams.
- `queue_size` – **[in]** The default queue size for all input virtual streams.

Returns Upon success, returns *Expected* of a map of input virtual stream name to params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<std::string, *hailo_vstream_params_t*» `make_output_vstream_params`(const ...)

Creates output virtual stream params for a given network_group.

Parameters

- `name` – **[in]** The name of the network or network_group which user wishes to create output virtual stream params for. In case network group name is given, the function returns the output virtual stream params of all the networks of the given network group. In case network name is given (provided by *get_network_infos*), the function returns the output virtual stream params of the given network. If NULL is passed, the function returns the output virtual stream params of all the networks of the first network group.
- `quantized` – **[in]** Whether the data returned from the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all output virtual streams.
- `timeout_ms` – **[in]** The default timeout in milliseconds for all output virtual streams.
- `queue_size` – **[in]** The default queue size for all output virtual streams.

Returns Upon success, returns *Expected* of a map of output virtual stream name to params. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::vector<*hailo_network_info_t*» `get_network_infos`(const std::string &net_group_name ...)

Gets all networks informations.

Parameters `net_group_name` – **[in]** The name of the network_group which contains the network information. If NULL is passed, the function returns the network infos of all the networks of the first network group.

Returns Upon success, returns *Expected* of a vector of *hailo_network_info_t*, containing each networks's informmation. Otherwise, returns *Unexpected* of *hailo_status* error.

std::string `hash`( ) const

Returns a unique hash for the specific *Hef*.

Returns A unique string hash for the *Hef*. Hefs created based on identical files will return identical hashes.

Public Static Functions

static *Expected*<*Hef*> `create`(const std::string &hef_path)

Creates an *Hef* from a file.

Parameters `hef_path` – **[in]** The path of the *Hef* file.

Returns Upon success, returns *Expected* of *Hef*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<*Hef*> `create`(const MemoryView &hef_buffer)

Creates an *Hef* from a buffer.

Parameters `hef_buffer` – **[in]** A buffer that contains the *Hef* content.

Returns Upon success, returns *Expected* of *Hef*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::string> `device_arch_to_string`(const *hailo_device_architecture_t* arch)

Get string of device architecture HEF was compiled for.

Parameters `arch` – **[in]** hailo_device_architecture_t representing the device architecture of the HEF

Returns Upon success, returns string representing the device architecture the HEF was compiled for. Otherwise, returns *Unexpected* of *hailo_status* error.

13.4. Network group API defines and functions

class `hailort::ActivatedNetworkGroup`

Activated network_group that can be used to send/receive data

Public Functions

virtual const std::string &`get_network_group_name`() const = 0

Returns The network group name.

virtual uint32_t `get_invalid_frames_count`() = 0

Returns The number of invalid frames.

class `hailort::ConfiguredNetworkGroup`

Loaded network_group that can be activated

Public Functions

virtual const std::string &`get_network_group_name`() const = 0

Returns The network group name.

virtual const std::string &`name`() const = 0

Returns The network group name.

virtual *Expected*<*hailo_stream_interface_t*> `get_default_streams_interface`() = 0

Gets the stream's default interface.

Returns Upon success, returns *Expected* of *hailo_stream_interface_t*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual std::vector<std::reference_wrapper<*InputStream*>> `get_input_streams_by_interface`(*hailo_stream_interface_*

Gets all input streams with the given *interface*.

Parameters `stream_interface` – **[in]** A *hailo_stream_interface_t* indicating which *InputStream* to return.

Returns Upon success, returns a vector of *InputStream*.

virtual std::vector<std::reference_wrapper<*OutputStream*>> `get_output_streams_by_interface`(*hailo_stream_interfa*

Gets all output streams with the given *interface*.

Parameters `stream_interface` – **[in]** A *hailo_stream_interface_t* indicating which *OutputStream* to return.

Returns Upon success, returns a vector of *OutputStream*.

virtual ExpectedRef<*InputStream*> `get_input_stream_by_name`(const std::string &name) = 0

Gets input stream by stream name.

Parameters `name` – **[in]** The name of the input stream to retrieve.

Returns Upon success, returns ExpectedRef of *InputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual ExpectedRef<*OutputStream*> `get_output_stream_by_name`(const std::string &name) = 0

Gets output stream by stream name.

Parameters `name` – **[in]** The name of the input stream to retrieve.

Returns Upon success, returns ExpectedRef of *OutputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<InputStreamRefVector> `get_input_streams_by_network`(const std::string ...)

Parameters `network_name` – **[in]** Network name of the requested input streams. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of vector *InputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<OutputStreamRefVector> `get_output_streams_by_network`(const ...)

Parameters `network_name` – **[in]** Network name of the requested output streams. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of vector *OutputStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual InputStreamRefVector `get_input_streams`( ) = 0

Returns All input streams.

virtual OutputStreamRefVector `get_output_streams`( ) = 0

Returns All output streams.

virtual *Expected*<LatencyMeasurementResult> `get_latency_measurement`(const std::string ...)

Parameters `network_name` – **[in]** Network name of the requested latency measurement. If not passed, all the networks in the network group will be addressed, and the resulted measurement is avarage latency of all networks.

Returns Upon success, returns *Expected* of LatencyMeasurementResult object containing the output latency result. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<OutputStreamWithParamsVector> `get_output_streams_from_vstream_names`(const ...)

Gets output streams and their vstream params from vstreams names.

Note: The output streams returned here are streams that corresponds to the names in outputs_params. If outputs_params contains a demux edge name, then all its predecessors names must be in outputs_params as well and the return value will contain the stream that corresponds to those edges.

Parameters `outputs_params` – **[in]** Map of output vstream name and params.

Returns Upon success, returns *Expected* of OutputStreamWithParamsVector. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::unique_ptr<*ActivatedNetworkGroup*>» `activate`( )

Activates hailo device inner-resources for context_switch inference.

Returns Upon success, returns *Expected* of a pointer to *ActivatedNetworkGroup* object. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::unique_ptr<*ActivatedNetworkGroup*>> `activate`(const ...)

Activates hailo device inner-resources for context_switch inference.

Parameters `network_group_params` – **[in]** Parameters for the activation.

Returns Upon success, returns *Expected* of a pointer to *ActivatedNetworkGroup* object. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `wait_for_activation`(const std::chrono::milliseconds &timeout) = 0

Block until network group is activated, or until timeout is passed.

Parameters `timeout` – **[in]** The timeout in milliseconds. If *timeout* is zero, the function returns immediately. If *timeout* is HAILO_INFINITE, the function returns only when the event is signaled.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *Expected*<std::map<std::string, *hailo_vstream_params_t*>> `make_input_vstream_params`(bool ...)

Creates input virtual stream params.

Parameters

- `quantized` – **[in]** Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all input virtual streams.
- `timeout_ms` – **[in]** The default timeout in milliseconds for all input virtual streams.
- `queue_size` – **[in]** The default queue size for all input virtual streams.
- `network_name` – **[in]** Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a map of name to vstream params. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::map<std::string, *hailo_vstream_params_t*>> `make_output_vstream_params`(bool ...)

Creates output virtual stream params.

Parameters

- `quantized` – **[in]** Whether the data returned from the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all output virtual streams.
- `timeout_ms` – **[in]** The default timeout in milliseconds for all output virtual streams.
- `queue_size` – **[in]** The default queue size for all output virtual streams.
- `network_name` – **[in]** Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a map of name to vstream params. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::map<std::string, *hailo_vstream_params_t*>>> `make_output_vstream_params_groups`

Creates output virtual stream params. The groups are splitted with respect to their low-level streams.

Parameters

- `quantized` – **[in]** Whether the data returned from the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data.
- `format_type` – **[in]** The default format type for all output virtual streams.

- `timeout_ms` – **[in]** The default timeout in milliseconds for all output virtual streams.
- `queue_size` – **[in]** The default queue size for all output virtual streams.

Returns Upon success, returns *Expected* of a vector of maps, mapping name to vstream params, where each map represents a params group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<std::vector<std::string>> `get_output_vstream_groups()` = 0

Gets output virtual stream groups for given network_group. The groups are splitted with respect to their low-level streams.

Returns Upon success, returns *Expected* of a map of vstream name to group index. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<*hailo_network_info_t*> `get_network_infos()` const = 0

Gets all network infos of the configured network group.

Returns Upon success, returns *Expected* of a vector of all network infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<*hailo_stream_info_t*> `get_all_stream_infos(`const std::string ... `)`

Parameters `network_name` – **[in]** Network name of the requested stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all stream infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<*hailo_vstream_info_t*> `get_input_vstream_infos(`const std::string ... `)`

Parameters `network_name` – **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all input vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<*hailo_vstream_info_t*> `get_output_vstream_infos(`const std::string ... `)`

Parameters `network_name` – **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all output vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *Expected*<std::vector<*hailo_vstream_info_t*> `get_all_vstream_infos(`const std::string ... `)`

Parameters `network_name` – **[in]** Network name of the requested virtual stream infos. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of all vstreams infos of the configured network group. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual bool `is_scheduled()` const = 0

Returns whether the network group is managed by the model scheduler.

virtual *hailo_status* `set_scheduler_timeout(`const std::chrono::milliseconds &timeout, const ... `)`

Sets the maximum time period that may pass before getting run time from the scheduler, even without reaching the minimum required send requests (e.g. threshold - see *set_scheduler_threshold()*), as long as at least one send request has been sent. This time period is measured since the last time the scheduler gave this network group run time.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*, and before the creation of any vstreams.

Note: The default timeout is 0ms.

Note: Currently, setting the timeout for a specific network is not supported.

Parameters

- `timeout` – **[in]** Timeout in milliseconds.
- `network_name` – **[in]** Network name for which to set the timeout. If not passed, the timeout will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `set_scheduler_threshold` (uint32_t threshold, const std::string ...)

Sets the minimum number of send requests required before the network is considered ready to get run time from the scheduler. If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see *hailo_set_scheduler_timeout()*), the scheduler will consider the network ready regardless.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*, and before the creation of any vstreams.

Note: The default threshold is 1.

Note: Currently, setting the threshold for a specific network is not supported.

Parameters

- `threshold` – **[in]** Threshold in number of frames.
- `network_name` – **[in]** Network name for which to set the threshold. If not passed, the threshold will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `set_scheduler_priority` (uint8_t priority, const std::string &network_name = ...)

Sets the priority of the network. When the network group scheduler will choose the next network, networks with higher priority will be prioritized in the selection. bigger number represent higher priority.

Note: Using this function is only allowed when scheduling_algorithm is not *HAILO_SCHEDULING_ALGORITHM_NONE*.

Note: The default priority is HAILO_SCHEDULER_PRIORITY_NORMAL.

Note: Currently, setting the priority for a specific network is not supported.

Parameters

- `priority` – **[in]** Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.
- `network_name` – **[in]** Network name for which to set the Priority. If not passed, the priority will be set for all the networks in the network group.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual bool `is_multi_context()` const = 0

Returns Is the network group multi-context or not.

virtual const *ConfigureNetworkParams* `get_config_params()` const = 0

Returns The configuration parameters this network group was initialized with.

13.5. Stream API functions

class `hailort::InputStream`

Input (host to device) stream representation

Public Types

using `TransferDoneCallback` = std::function<void(const *CompletionInfo* &completion_info)>

Async transfer complete callback prototype.

Public Functions

virtual *hailo_status* `set_timeout(`std::chrono::milliseconds timeout`)` = 0

Set new timeout value to the input stream

Parameters `timeout` – **[in]** The new timeout value to be set in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual std::chrono::milliseconds `get_timeout()` const = 0

Returns The input stream's timeout in milliseconds.

virtual *hailo_stream_interface_t* `get_interface()` const = 0

Returns The input stream's interface.

virtual *hailo_status* `abort()` = 0

Aborting the stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `clear_abort()` = 0

Clearing the aborted state of the stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `flush()`

Writes all pending data to the underlying stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

EventPtr &`get_network_group_activated_event()`

Returns a pointer for network group activated event.

virtual bool `is_scheduled()` = 0

Returns whether the stream is managed by the model scheduler.

virtual *hailo_status* `write(` const MemoryView &buffer `)`

Writes the entire buffer to the stream without transformations.

Note: *buffer* is expected to be in the format dictated by this.stream_info.format

Note: *buffer.size()* is expected to be *get_frame_size()*.

Parameters `buffer` – **[in]** The buffer to be written.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

virtual *hailo_status* `write(` const void *buffer, size_t size `)`

Writes the entire buffer to the stream without transformations.

Note: *buffer* is expected to be in the format dictated by this.stream_info.format

Note: *size* is expected to be *get_frame_size()*.

Parameters

- `buffer` – **[in]** The buffer to be written.
- `size` – **[in]** The size of the buffer given.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

virtual *hailo_status* `wait_for_async_ready(` size_t transfer_size, std::chrono::milliseconds timeout `)`

Waits until the stream is ready to launch a new *write_async()* operation. Each stream contains some limited sized queue for ongoing transfers. Calling *get_async_max_queue_size()* will return the queue size for current stream.

Parameters

- `transfer_size` – **[in]** Must be *get_frame_size()*.
- `timeout` – **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If *timeout* has passed and the stream is not ready, returns *HAILO_TIMEOUT*.
- In any other error case, returns *hailo_status* error.

virtual *Expected*<size_t> `get_async_max_queue_size()` const

Returns the maximum amount of frames that can be simultaneously written to the stream (by *write_async()* calls) before any one of the write operations is complete, as signified by *user_callback* being called.

Returns Upon success, returns *Expected* of a the queue size. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `write_async` (const MemoryView &buffer, const *TransferDoneCallback* ...)

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *write_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *write_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The buffer's format comes from the *format* field inside *get_info()* and the shape comes from the *hw_shape* field inside *get_info()*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `buffer` – **[in]** The buffer to be written. The buffer must be aligned to the system page size.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case please wait until *user_callback* is called on previous writes, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

virtual *hailo_status* `write_async` (const void *buffer, size_t size, const *TransferDoneCallback* ...)

Writes the contents of *buffer* to the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *write_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *write_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*).

Note: *user_callback* should run as quickly as possible.

Note: The buffer's format comes from the *format* field inside *get_info()* and the shape comes from the *hw_shape* field inside *get_info()*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `buffer` – **[in]** The buffer to be written. The buffer must be aligned to the system page size.
- `size` – **[in]** The size of the given buffer, expected to be *get_frame_size()*.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case please wait until *user_callback* is called on previous writes, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

inline virtual const *hailo_stream_info_t* `&get_info()` const

Returns A *hailo_stream_info_t* object containing the stream's info.

inline virtual size_t `get_frame_size()` const

Returns the stream's hw frame size in bytes.

inline std::string `name()` const

Returns the stream's name.

virtual std::string `to_string()` const

Returns the stream's description containing it's name and index.

struct `CompletionInfo`

Context passed to the *TransferDoneCallback* after the async operation is done or has failed.

Public Members

hailo_status `status`

Status of the async transfer.
- *HAILO_SUCCESS* - When transfer is complete successfully.
- *HAILO_STREAM_ABORTED_BY_USER* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

class `hailort::OutputStream`

Output (device to host) stream representation

Public Types

using `TransferDoneCallback` = std::function<void(const *CompletionInfo* &completion_info)>

Async transfer complete callback prototype.

Public Functions

virtual *hailo_status* `set_timeout`(std::chrono::milliseconds timeout) = 0

Set new timeout value to the output stream

Parameters `timeout` – **[in]** The new timeout value to be set in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual std::chrono::milliseconds `get_timeout`() const = 0

Returns returns the output stream's timeout in milliseconds.

virtual *hailo_stream_interface_t* `get_interface`() const = 0

Returns returns the output stream's interface.

virtual *hailo_status* `abort`() = 0

Aborting the stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `clear_abort`() = 0

Clearing the abort flag of the stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

EventPtr &`get_network_group_activated_event`()

Returns a pointer for network group activated event.

virtual bool `is_scheduled`() = 0

Returns whether the stream is managed by the model scheduler.

inline virtual const *hailo_stream_info_t* &`get_info`() const

Returns the stream's info.

inline virtual size_t `get_frame_size`() const

Returns the stream's hw frame size.

inline std::string `name`() const

Returns the stream's name.

virtual std::string `to_string`() const

Returns the stream's description containing it's name and index.

uint32_t `get_invalid_frames_count`() const

Returns the number of invalid frames received by the stream.

virtual *hailo_status* `read`(MemoryView buffer)

Reads the entire buffer from the stream without transformations

Note: Upon return, *buffer* is expected to be in the format dictated by this.get_info().format

Note: *size* is expected to be *get_frame_size()*.

Parameters `buffer` – **[in]** A buffer that receives the data read from the stream.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

virtual *hailo_status* `read` (void *buffer, size_t size)

Reads the entire buffer from the stream without transformations

Note: Upon return, *buffer* is expected to be in the format dictated by this.get_info().format

Note: *size* is expected to be *get_frame_size()*.

Parameters

- `buffer` – **[in]** A pointer to a buffer that receives the data read from the stream.
- `size` – **[in]** The size of the given buffer.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

virtual *hailo_status* `wait_for_async_ready` (size_t transfer_size, std::chrono::milliseconds timeout)

Waits until the stream is ready to launch a new *read_async()* operation. Each stream contains some limited sized queue for ongoing transfers. Calling *get_async_max_queue_size()* will return the queue size for current stream.

Parameters

- `transfer_size` – **[in]** Must be *get_frame_size()*.
- `timeout` – **[in]** Amount of time to wait until the stream is ready in milliseconds.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If *timeout* has passed and the stream is not ready, returns *HAILO_TIMEOUT*.
- In any other error case, returns *hailo_status* error.

virtual *Expected*<size_t> `get_async_max_queue_size` ( ) const

Returns the maximum amount of frames that can be simultaneously read from the stream (by *read_async()* calls) before any one of the read operations is complete, as signified by *user_callback* being called.

Returns Upon success, returns *Expected* of a the queue size. Otherwise, returns *Unexpected* of *hailo_status* error.

virtual *hailo_status* `read_async` (MemoryView buffer, const *TransferDoneCallback* &user_callback) = 0

Reads into *buffer* from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *read_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *read_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*). If the operation has completed successfully, the contents of *buffer* will have been updated by the read operation.

Note: *user_callback* should execute as quickly as possible.

Note: The buffer's format is determined by the *format* field inside *get_info()*, and the shape is determined by the *hw_shape* field inside *get_info()*.

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `buffer` – **[in]** The buffer to be read into. The buffer must be aligned to the system page size.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case, please wait until *user_callback* is called on previous reads, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

virtual *hailo_status* `read_async` (void *buffer, size_t size, const *TransferDoneCallback* &user_callback) = 0

Reads into *buffer* from the stream asynchronously, initiating a deferred operation that will be completed later.

- If the function call succeeds (i.e., *read_async()* returns *HAILO_SUCCESS*), the deferred operation has been initiated. Until *user_callback* is called, the user cannot change or delete *buffer*.
- If the function call fails (i.e., *read_async()* returns a status other than *HAILO_SUCCESS*), the deferred operation will not be initiated and *user_callback* will not be invoked. The user is free to change or delete *buffer*.
- *user_callback* is triggered upon successful completion or failure of the deferred operation. The callback receives a *CompletionInfo* object containing a pointer to the transferred buffer (*buffer_addr*) and the transfer status (*status*). If the operation has completed successfully, the contents of *buffer* will have been updated by the read operation.

Note: *user_callback* should execute as quickly as possible.

Note: The buffer's format is determined by the *format* field inside *get_info()*, and the shape is determined by the *hw_shape* field inside *get_info()*

Note: The address provided must be aligned to the system's page size, and the rest of the page should not be in use by any other part of the program to ensure proper functioning of the DMA operation. Memory for the provided address can be allocated using `mmap` on Unix-like systems or `VirtualAlloc` on Windows.

Parameters

- `buffer` – **[in]** The buffer to be read into. The buffer must be aligned to the system page size.

- `size` – **[in]** The size of the given buffer, expected to be *get_frame_size()*.
- `user_callback` – **[in]** The callback that will be called when the transfer is complete or has failed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise:

- If the stream queue is full, returns *HAILO_QUEUE_IS_FULL*. In this case, please wait until *user_callback* is called on previous reads, or call *wait_for_async_ready()*. The size of the queue can be determined by calling *get_async_max_queue_size()*.
- In any other error case, returns a *hailo_status* error.

struct `CompletionInfo`

Context passed to the *TransferDoneCallback* after the async operation is done or has failed.

Public Members

hailo_status `status`

Status of the async transfer.
- *HAILO_SUCCESS* - When transfer is complete successfully.
- *HAILO_STREAM_ABORTED_BY_USER* - The transfer was canceled (can happen after network deactivation).
- Any other *hailo_status* on unexpected errors.

13.6. Transformation API functions

HAILORTAPI *hailo_status* `transpose_buffer`(const MemoryView src, const *hailo_3d_image_shape_t* &shape, ...)

Transposed *src* buffer (whose shape and format are given) to *dst* buffer (whose shape will be the same as `shape` but the height and width are replaced)

Note: assumes src and dst not overlas

Parameters

- `src` – **[in]** A buffer containing the data to be transposed.
- `shape` – **[in]** The shape of *src*.
- `format` – **[in]** The format of *src*.
- `dst` – **[out]** The dst buffer to contain the transposed data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

HAILORTAPI inline size_t `get_transpose_buffer_size`(const *hailo_3d_image_shape_t* &src_shape, ...)

Returns The size of transpose buffer by its src shape and dst format type.

HAILORTAPI *hailo_status* `fuse_buffers`(const std::vector<MemoryView> &buffers, const ...)

Fuse multiple defused NMS buffers referred by *buffers* to the buffer referred by *dst*. This function expects *buffers* to be ordered by their *class_group_index* (lowest to highest).

Parameters

- `buffers` – **[in]** A vector of buffers to be fused.
- `infos_of_buffers` – **[in]** A vector of *hailo_nms_info_t* containing the *buffers* information.

- `dst` – **[out]** A pointer to a buffer which will contain the fused buffer.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

class `hailort::InputTransformContext`

Object used for input stream transformation

Public Functions

hailo_status `transform`(const MemoryView src, MemoryView dst)

Transforms an input frame referred by *src* directly to the buffer referred by *dst*.

Parameters

- `src` – **[in]** A src buffer to be transformed.
- `dst` – **[out]** A dst buffer that receives the transformed data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

size_t `get_src_frame_size`() const

Returns The size of the src frame on the host side in bytes.

size_t `get_dst_frame_size`() const

Returns The size of the dst frame on the hw side in bytes.

virtual std::string `description`() const

Returns A human-readable description of the transformation parameters.

Public Static Functions

static *Expected*<std::unique_ptr<*InputTransformContext*>> `create`(const *hailo_3d_image_shape_t* ...)

Creates input transform_context.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer to be transformed.
- `src_format` – **[in]** The format of the src buffer to be transformed.
- `dst_image_shape` – **[in]** The shape of the dst buffer that receives the transformed data.
- `dst_format` – **[in]** The format of the dst buffer that receives the transformed data.
- `dst_quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*InputTransformContext*>> `create`(const *hailo_stream_info_t* &stream_info, ...)

Creates input transform_context.

Parameters

- `stream_info` – **[in]** Creates transform_context that fits this stream info.
- `transform_params` – **[in]** A *hailo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*InputTransformContext*» `create`(const *hailo_stream_info_t* &stream_info, ...)

Creates input transform_context.

Parameters

- `stream_info` – **[in]** Creates transform_context that fits this stream info.
- `quantized` – **[in]** Whether the data fed into the transform_context is already quantized. True means the data is already quantized. False means it's transform_context responsibility to quantize (scale) the data.
- `format_type` – **[in]** The type of the buffer sent to the transform_context.

Returns Upon success, returns *Expected* of a pointer to *InputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static bool `is_transformation_required`(const *hailo_3d_image_shape_t* &src_image_shape, ...)

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is ready to be sent to HW without any transformation.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer (host shape).
- `src_format` – **[in]** The format of the src buffer (host format).
- `dst_image_shape` – **[in]** The shape of the dst buffer (hw shape).
- `dst_format` – **[in]** The format of the dst buffer (hw format).
- `quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.

Returns Returns whether or not a transformation is needed.

class `hailort::OutputTransformContext`

Object used for output stream transformation

Public Functions

virtual *hailo_status* `transform`(const MemoryView src, MemoryView dst) = 0

Transforms an output frame referred by *src* directly to the buffer referred by *dst*.

Parameters

- `src` – **[in]** A src buffer to be transformed.
- `dst` – **[out]** A dst buffer that receives the transformed data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

size_t `get_src_frame_size`() const

Returns The size of the src frame on the hw side in bytes.

size_t `get_dst_frame_size`() const

Returns The size of the dst frame on the host side in bytes.

virtual std::string `description`() const = 0

Returns A human-readable description of the transformation parameters.

Public Static Functions

static *Expected*<std::unique_ptr<*OutputTransformContext*>» `create` (const *hailo_3d_image_shape_t* ...)

Creates output transform_context.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer to be transformed.
- `src_format` – **[in]** The format of the src buffer to be transformed.
- `dst_image_shape` – **[in]** The shape of the dst buffer that receives the transformed data.
- `dst_format` – **[in]** The format of the dst buffer that receives the transformed data.
- `dst_quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.
- `nms_info` – **[in]** A *hailo_nms_info_t* object containing nms information.

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*OutputTransformContext*>» `create` (const *hailo_stream_info_t* ...)

Creates output transform_context.

Parameters

- `stream_info` – **[in]** Creates transform_context that fits this stream info.
- `transform_params` – **[in]** A *hailo_transform_params_t* object containing user transformation parameters.

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::unique_ptr<*OutputTransformContext*>» `create` (const *hailo_stream_info_t* ...)

Creates output transform_context with default transform parameters

Parameters

- `stream_info` – **[in]** Creates transform_context that fits this stream info.
- `quantized` – **[in]** Whether the data returned from the transform_context should be quantized. True means that the data returned to the user is still quantized. False means it's the transform_context responsibility to de-quantize (rescale) the data.
- `format_type` – **[in]** The type of the buffer returned from the transform_context

Returns Upon success, returns *Expected* of a pointer to *OutputTransformContext*. Otherwise, returns *Unexpected* of *hailo_status* error.

static bool `is_transformation_required` (const *hailo_3d_image_shape_t* &src_image_shape, ...)

Check whether or not a transformation is needed.

Note: In case the function returns false, the src frame is already in the required format without any transformation.

Parameters

- `src_image_shape` – **[in]** The shape of the src buffer (hw shape).
- `src_format` – **[in]** The format of the src buffer (hw format).
- `dst_image_shape` – **[in]** The shape of the dst buffer (host shape).
- `dst_format` – **[in]** The format of the dst buffer (host format).

- `quant_info` – **[in]** A *hailo_quant_info_t* object containing quantization information.

Returns Returns whether or not a transformation is needed.

class `hailort::OutputDemuxer`

Object used to demux muxed stream

Public Functions

virtual std::vector<*hailo_stream_info_t*> `get_edges_stream_info()` = 0

Returns The demuxer's edges information.

virtual *hailo_status* `transform_demux(` const MemoryView src, const std::map<std::string, ...)

Demultiplexing an output frame referred by *src* directly into the buffers referred by *dst_ptrs*.

Parameters

- `src` – **[in]** A buffer to be demultiplexed.
- `dst_ptrs` – **[out]** A mapping of output_name to its resulted demuxed data.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

virtual *hailo_status* `transform_demux(` const MemoryView src, std::vector<MemoryView> ...)

Demultiplexing an output frame referred by *src* directly into the buffers referred by *raw_buffers*.

Note: The order of *raw_buffers* should be the same as returned from the function '*get_edges_stream_info()*'.

Parameters

- `src` – **[in]** A buffer to be demultiplexed.
- `raw_buffers` – **[out]** A vector of buffers that receives the demultiplexed data read from the stream. The order of *raw_buffers* vector will remain as is.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Public Static Functions

static *Expected*<std::unique_ptr<*OutputDemuxer*>» `create(` *OutputStream* &output_stream)

Creates an output demuxer for the given *output_stream*.

Parameters `output_stream` – **[in]** An *OutputStream* object to create demuxer from.

Returns Upon success, returns *Expected* of a pointer to *OutputDemuxer*. Otherwise, returns *Unexpected* of *hailo_status* error.

class `hailort::Quantization`

Hailo device requires input data to be quantized/scaled before it is sent. Similarly, data outputted from the device needs to be 'de-quantized'/rescaled as well. When a neural network is compiled, each input/output layer in the neural network is assigned two floating point values that are parameters to an input/output transformation: *hailo_quant_info_t::qp_zp* (zero_point) and *hailo_quant_info_t::qp_scale* (fields of the struct *hailo_quant_info_t*). These values are stored in the *Hef*.

- Input transformation: input data is divided by *hailo_quant_info_t::qp_scale* and then *hailo_quant_info_t::qp_zp* is added to the result.
- Output transformation: *hailo_quant_info_t::qp_zp* is subtracted from output data and then the result is multiplied by *hailo_quant_info_t::qp_scale*.

Public Static Functions

template<typename T, typename Q>
static inline void `dequantize_output_buffer`(*Q* *src_ptr, *T* *dst_ptr, uint32_t ...)

De-quantize output buffer pointed by *src_ptr* from data type *Q* into the buffer pointed by *dst_ptr* of data type *T*.

Parameters

- `src_ptr` – **[in]** A pointer to the buffer containing the data that will be de-quantized.
- `dst_ptr` – **[out]** A pointer to the buffer that will contain the output de-quantized data.
- `buffer_elements_count` – **[in]** The number of elements in *src_ptr* and *dst_ptr* arrays.
- `quant_info` – **[in]** *Quantization* info.

template<typename T, typename Q>
static inline void `dequantize_output_buffer_in_place`(*T* *dst_ptr, uint32_t ...)

De-quantize in place the output buffer pointed by *dst_ptr* from data type *Q* to data type *T*.

Parameters

- `dst_ptr` – **[inout]** A pointer to the buffer to be de-quantized.
- `buffer_elements_count` – **[in]** The number of elements in *dst_ptr* array.
- `quant_info` – **[in]** *Quantization* info.

template<typename T, typename Q>
static inline void `dequantize_output_buffer_in_place`(*T* *dst_ptr, uint32_t offset, uint32_t ...)

De-quantize in place the output buffer pointed by *dst_ptr* starting from *offset* from data type *Q* to data type *T*.

Parameters

- `dst_ptr` – **[inout]** A pointer to the buffer to be de-quantized.
- `offset` – **[in]** The offset in *dst_ptr* array to start from.
- `buffer_elements_count` – **[in]** The number of elements in *dst_ptr* array.
- `qp_zp` – **[in]** *Quantization* zero point.
- `qp_scale` – **[in]** *Quantization* scale.

template<typename T, typename Q>
static inline void `quantize_input_buffer`(*T* *src_ptr, *Q* *dst_ptr, uint32_t buffer_elements_count, ...)

Quantize input buffer pointed by *src_ptr* of data type *T*, into the buffer pointed by *dst_ptr* of data type *Q*.

Parameters

- `src_ptr` – **[in]** A pointer to the buffer containing the data that will be quantized.
- `dst_ptr` – **[out]** A pointer to the buffer that will contain the output quantized data.
- `buffer_elements_count` – **[in]** The number of elements in *src_ptr* and *dst_ptr* arrays.
- `quant_info` – **[in]** *Quantization* info.

template<typename T, typename Q>
static inline void `dequantize_output_buffer_nms`(*Q* *src_ptr, *T* *dst_ptr, uint32_t ...)

De-quantize output NMS buffer pointed by *src_ptr* of data type *Q*, into the buffer pointed by *dst_ptr* of data type *T*.

Parameters

- `src_ptr` – **[in]** A pointer to the buffer containing the data that will be de-quantized.

- `dst_ptr` – **[out]** A pointer to the buffer that will contain the output de-quantized data.
- `buffer_elements_count` – **[in]** The number of elements in *src_ptr* and *dst_ptr* arrays.
- `quant_info` – **[in]** *Quantization* info.
- `number_of_classes` – **[in]** Amount of NMS classes.

static inline bool `is_identity_qp`(const *hailo_quant_info_t* &quant_info)

Indicates whether the *quant_info* contains the identity scale. If true there is no need to fix the data's scale.

static inline bool `is_identity_qp`(*float32_t* qp_zp, *float32_t* qp_scale)

Indicates whether the *qp_zp* and *qp_scale* is the identity scale. If true there is no need to fix the data's scale.

template<typename `T`, typename `Q`>
static inline *T* `dequantize_output`(*Q* number, *hailo_quant_info_t* quant_info)

De-quantize *number* from data type *Q* to data type *T* and fix it's scale according to *quant_info*.

Parameters

- `number` – **[in]** The value to be de-quantized.
- `quant_info` – **[in]** *Quantization* info.

Returns Returns the dequantized value of *number*.

template<typename `T`, typename `Q`>
static inline *T* `dequantize_output`(*Q* number, *float32_t* qp_zp, *float32_t* qp_scale)

De-quantize *number* from data type *Q* to data type *T* and fix it's scale according to *qp_zp* and *qp_scale*.

Parameters

- `number` – **[in]** The value to be de-quantized.
- `qp_zp` – **[in]** *Quantization* zero point.
- `qp_scale` – **[in]** *Quantization* scale.

Returns Returns the dequantized value of *number*.

13.7. Virtual Stream API functions

class `hailort::InputVStream`

Public Functions

hailo_status `write`(const MemoryView &buffer)

Writes *buffer* to hailo device.

Parameters `buffer` – **[in]** The buffer containing the data to be sent to device. The buffer's format can be obtained by *get_user_buffer_format()*, and the buffer's shape can be obtained by calling *get_info()*.shape.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `flush()`

Flushes the vstream pipeline buffers. This will block until the vstream pipeline is clear.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `abort()`

Aborts vstream until its resumed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `resume()`

Resumes vstream after it was aborted.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

size_t `get_frame_size()` const

Note: The size could be affected by the format type - using UINT16, or by the data not being quantized yet.

Returns the size of a virtual stream's frame on the host side in bytes.

const *hailo_vstream_info_t* `&get_info()` const

Returns *hailo_vstream_info_t* object containing the vstream info.

const *hailo_format_t* `&get_user_buffer_format()` const

Returns *hailo_format_t* object containing the user buffer format.

std::string `name()` const

Returns the virtual stream's name.

std::string `network_name()` const

Returns the virtual stream's network name.

const std::map<std::string, AccumulatorPtr> `&get_fps_accumulators()` const

Gets a reference to a map between pipeline element names to their respective fps accumulators. These accumulators measure the net throughput of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: FPS accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective fps accumulators.

const std::map<std::string, AccumulatorPtr> `&get_latency_accumulators()` const

Gets a reference to a map between pipeline element names to their respective latency accumulators. These accumulators measure the net latency of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: Latency accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective latency accumulators.

const std::map<std::string, std::vector<AccumulatorPtr» &`get_queue_size_accumulators()` const

Gets a reference to a map between pipeline element names to their respective queue size accumulators. These accumulators measure the number of free buffers in the queue, right before a buffer is removed from the queue to be used.

Note: Queue size accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective queue size accumulators.

AccumulatorPtr `get_pipeline_latency_accumulator()` const

Gets a shared_ptr to the vstream's latency accumulator. This accumulator measures the time it takes for a frame to pass through an entire vstream pipeline. Specifically:

- For *InputVStream*s: The time it takes a frame from the call to *InputVStream::write*, until the frame is written to the HW.
- For *OutputVStream*s: The time it takes a frame from being read from the HW, until it's returned to the user via *OutputVStream::read*.

Note: A pipeline-wide latency accumulator is created for the vstream, if the vstream is created with the flag *HAILO_VSTREAM_STATS_MEASURE_LATENCY* set under the *vstream_stats_flags* field of *hailo_vstream_params_t*.

Returns A shared pointer to the vstream's latency accumulator.

const std::vector<std::shared_ptr<PipelineElement» &`get_pipeline()` const

Returns A const reference to the *PipelineElements* that this vstream is comprised of.

Public Static Functions

static *hailo_status* `clear` (std::vector<*InputVStream*> &vstreams)

Clears the vstreams' pipeline buffers.

Parameters `vstreams` – **[in]** The vstreams to be cleared.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

static *hailo_status* `clear` (std::vector<std::reference_wrapper<*InputVStream*» &vstreams)

Clears the vstreams' pipeline buffers.

Parameters `vstreams` – **[in]** The vstreams to be cleared.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

class `hailort::OutputVStream`

Public Functions

hailo_status `read` (MemoryView buffer)

Reads data from hailo device into *buffer*.

Parameters `buffer` – **[in]** The buffer to read data into. The buffer's format can be obtained by *get_user_buffer_format()*, and the buffer's shape can be obtained by calling *get_info()*.shape.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `abort()`

Aborts vstream until its resumed.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

hailo_status `resume()`

Resumes vstream after it was aborted.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

size_t `get_frame_size()` const

Note: The size could be affected by the format type - using UINT16, or by the data not being quantized yet.

Returns the size of a virtual stream's frame on the host side in bytes.

const *hailo_vstream_info_t* `&get_info()` const

Returns *hailo_vstream_info_t* object containing the vstream info.

const *hailo_format_t* `&get_user_buffer_format()` const

Returns *hailo_format_t* object containing the user buffer format.

std::string `name()` const

Returns the virtual stream's name.

std::string `network_name()` const

Returns the virtual stream's network name.

const std::map<std::string, AccumulatorPtr> `&get_fps_accumulators()` const

Gets a reference to a map between pipeline element names to their respective fps accumulators. These accumulators measure the net throughput of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: FPS accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_FPS* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective fps accumulators.

const std::map<std::string, AccumulatorPtr> `&get_latency_accumulators()` const

Gets a reference to a map between pipeline element names to their respective latency accumulators. These accumulators measure the net latency of each pipeline element. This means that the effects of queuing in the vstream pipeline (between elements) are not accounted for by these accumulators.

Note: Latency accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_LATENCY* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective latency accumulators.

const std::map<std::string, std::vector<AccumulatorPtr» &`get_queue_size_accumulators()` const

Gets a reference to a map between pipeline element names to their respective queue size accumulators. These accumulators measure the number of free buffers in the queue, right before a buffer is removed from the queue to be used.

Note: Queue size accumulators are created for pipeline elements, if the vstream is created with the flag *HAILO_PIPELINE_ELEM_STATS_MEASURE_QUEUE_SIZE* set under the *pipeline_elements_stats_flags* field of *hailo_vstream_params_t*.

Returns A const reference to a map between pipeline element names to their respective queue size accumulators.

AccumulatorPtr `get_pipeline_latency_accumulator()` const

Gets a shared_ptr to the vstream's latency accumulator. This accumulator measures the time it takes for a frame to pass through an entire vstream pipeline. Specifically:

- For *InputVStream*s: The time it takes a frame from the call to *InputVStream::write*, until the frame is written to the HW.
- For *OutputVStream*s: The time it takes a frame from being read from the HW, until it's returned to the user via *OutputVStream::read*.

Note: A pipeline-wide latency accumulator is created for the vstream, if the vstream is created with the flag *HAILO_VSTREAM_STATS_MEASURE_LATENCY* set under the *vstream_stats_flags* field of *hailo_vstream_params_t*.

Returns A shared pointer to the vstream's latency accumulator.

const std::vector<std::shared_ptr<PipelineElement» &`get_pipeline()` const

Returns A const reference to the *PipelineElements* that this vstream is comprised of.

Public Static Functions

static *hailo_status* `clear` (std::vector<*OutputVStream*> &vstreams)

Clears the vstreams' pipeline buffers.

Parameters `vstreams` – **[in]** The vstreams to be cleared.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

static *hailo_status* `clear` (std::vector<std::reference_wrapper<*OutputVStream*» &vstreams)

Clears the vstreams' pipeline buffers.

Parameters `vstreams` – **[in]** The vstreams to be cleared.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

class `hailort::VStreamsBuilder`

Contains the virtual streams creation functions

Public Static Functions

static *Expected*<std::pair<std::vector<*InputVStream*>, std::vector<*OutputVStream*>> `create_vstreams` (*ConfiguredNetworkGro*

Creates input virtual streams and output virtual streams.

Parameters

- `net_group` – **[in]** Configured network group that owns the streams.
- `quantized` – **[in]** Default quantized parameter for all virtual streams. For input vstreams indicates whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data. For output vstreams indicates whether the data returned from the device should be quantized. True means that the data returned to the user is still quantized. False means it's HailoRT's responsibility to de-quantize (rescale) the data.
- `format_type` – **[in]** The default format type for all virtual streams.
- `network_name` – **[in]** Request to create vstreams of specific network inside the configured network group. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a pair of input vstreams and output vstreams. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::pair<std::vector<*InputVStream*>, std::vector<*OutputVStream*>> `create_vstreams` (*ConfiguredNetworkGro*

Creates input virtual streams and output virtual streams.

Parameters

- `net_group` – **[in]** Configured network group that owns the streams.
- `vstreams_params` – **[in]** A *hailo_vstream_params_t* containing default params for all virtual streams.
- `network_name` – **[in]** Request to create vstreams of specific network inside the configured network group. If not passed, all the networks in the network group will be addressed.

Returns Upon success, returns *Expected* of a vector of input virtual streams. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::vector<*InputVStream*» `create_input_vstreams` (*ConfiguredNetworkGroup* ...)

Creates input virtual streams.

Parameters

- `net_group` – **[in]** Configured network group that owns the streams.
- `inputs_params` – **[in]** Map of input vstreams <name, params> to create input vstreams from.

Returns Upon success, returns *Expected* of a vector of input virtual streams. Otherwise, returns *Unexpected* of *hailo_status* error.

static *Expected*<std::vector<*OutputVStream*» `create_output_vstreams` (*ConfiguredNetworkGroup* ...)

Creates output virtual streams.

Note: If not creating all output vstreams together, one should make sure all vstreams from the same group are created together. See *ConfiguredNetworkGroup::make_output_vstream_params_groups*

Parameters

- `net_group` – **[in]** Configured network group that owns the streams.

- `outputs_params` – **[in]** Map of output vstreams <name, params> to create output vstreams from.

Returns Upon success, returns *Expected* of a vector of output virtual streams. Otherwise, returns *Unexpected* of *hailo_status* error.

13.8. InferVStreams

class `hailort::InferVStreams`

Pipeline used to run inference

Public Functions

hailo_status `infer`(const std::map<std::string, MemoryView> &input_data, std::map<std::string, ...)

Run inference on dataset *input_data*.

Note: *ConfiguredNetworkGroup* must be activated before calling this function.

Note: The size of each element in *input_data* and *output_data* must match the frame size of the matching vstream name multiplied by *frames_count*.

Note: If at least one input/output of some network is present, all inputs and outputs of that network must also be present.

Parameters

- `input_data` – **[in]** A mapping of vstream name to MemoryView containing input dataset for inference.
- `output_data` – **[out]** A mapping of vstream name to MemoryView containing the inference output data.
- `frames_count` – **[in]** The amount of inferred frames.

Returns Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Expected<std::reference_wrapper<*InputVStream*>> `get_input_by_name`(const std::string &name)

Get *InputVStream* by name.

Parameters `name` – **[in]** The vstream's name.

Returns Upon success, returns *Expected* of *InputVStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::reference_wrapper<*OutputVStream*>> `get_output_by_name`(const std::string &name)

Get *OutputVStream* by name.

Parameters `name` – **[in]** The vstream's name.

Returns Upon success, returns *Expected* of *OutputVStream*. Otherwise, returns *Unexpected* of *hailo_status* error.

std::vector<std::reference_wrapper<*InputVStream*>> `get_input_vstreams`( )

Returns Returns a vector of all *InputVStream*s.

std::vector<std::reference_wrapper<*OutputVStream*>> `get_output_vstreams()`

Returns Returns a vector of all *OutputVStream*s.

Public Static Functions

static *Expected*<*InferVStreams*> `create`(*ConfiguredNetworkGroup* &net_group, const std::map<std::string, ...)

Creates vstreams pipelines to be used later for inference by calling the *InferVStreams::infer()* function.

Note: If at least one input/output of some network is present, all inputs and outputs of that network must also be present.

Parameters

- `net_group` – **[in]** A *ConfiguredNetworkGroup* to run the inference on.
- `input_params` – **[in]** A mapping of input vstream name to its' params. Can be achieved by calling *ConfiguredNetworkGroup::make_input_vstream_params()* or *Hef::make_input_vstream_params* functions.
- `output_params` – **[in]** A mapping of output vstream name to its' params. Can be achieved by calling *ConfiguredNetworkGroup::make_output_vstream_params()* or *Hef::make_output_vstream_params()* functions.

Returns Upon success, returns *Expected* of *InferVStreams*. Otherwise, returns *Unexpected* of *hailo_status* error.

13.9. Common HailoRT utility functions

namespace `hailort`

class `HailoRTCommon`

#include <hailort_common.hpp>

Public Static Functions

static inline constexpr uint32_t `get_nms_host_shape_size`(const *hailo_nms_info_t* &nms_info)

Gets the NMS host shape size (number of elements) from NMS info.

Note: The size in bytes can be calculated using *get_nms_host_frame_size(const hailo_nms_info_t &nms_info, const hailo_format_t &format)*.

Parameters `nms_info` – **[in]** The NMS info to get shape size from.
Returns The host shape size (number of elements).

static inline constexpr uint32_t `get_nms_host_shape_size`(const *hailo_nms_shape_t* ...)

Gets the NMS host shape size (number of elements) from NMS shape.

Note: The size in bytes can be calculated using *get_nms_host_frame_size(const hailo_nms_shape_t &nms_shape, const hailo_format_t &format)*.

Parameters `nms_shape` – **[in]** The NMS shape to get size from.
Returns The host shape size (number of elements).

static inline constexpr uint32_t `align_to`(uint32_t num, uint32_t alignment)

Rounds an integer value up to the next multiple of a specified size.

Parameters

- `num` – **[in]** Original number.
- `alignment` – **[in]** Returned number should be aligned to this parameter.

Returns aligned number

static inline constexpr uint32_t `get_shape_size`(const *hailo_3d_image_shape_t* &shape, ...)

Gets the shape size.

Parameters

- `shape` – **[in]** The shape to get size from.
- `row_alignment` – **[in]** The size the shape row is aligned to.

Returns The shape size.

static inline constexpr uint8_t `get_data_bytes`(*hailo_format_type_t* type)

Gets the size of each element in bytes from buffer's format type.

Parameters `type` – **[in]** A *hailo_format_type_t* object.

Returns The data bytes.

static inline *Expected*<*hailo_format_type_t*> `get_format_type`(uint32_t hw_data_bytes)

Gets the format type of a stream by the hw data bytes parameter.

Parameters `hw_data_bytes` – **[in]** The stream's info's hw_data_bytes parameter.

Returns Upon success, returns *Expected* of *hailo_format_type_t*, The format type that the hw_data_type correlates to. Otherwise, returns *Unexpected* of *hailo_status* error.

static inline std::string `get_format_type_str`(const *hailo_format_type_t* &type)

Gets a string reprenestation of the given format type.

Parameters `type` – **[in]** A *hailo_format_type_t* object.

Returns The string representation of the format type.

static inline std::string `get_device_arch_str`(const *hailo_device_architecture_t* &arch)

Gets a string reprenestation of the given device architecture.

Parameters `arch` – **[in]** A *hailo_device_architecture_t* object.

Returns The string representation of the device architecture.

static inline std::string `get_format_order_str`(const *hailo_format_order_t* &order)

Gets a string reprenestation of the given format order.

Parameters `order` – **[in]** A *hailo_format_order_t* object.

Returns The string representation of the format order.

static inline constexpr uint8_t `get_format_data_bytes`(const *hailo_format_t* &format)

Gets the size of each element in bytes from buffer's format.

Parameters `format` – **[in]** A *hailo_format_t* object.

Returns The format's data bytes.

static inline constexpr uint32_t `get_nms_host_frame_size`(const *hailo_nms_info_t* ...)

Gets NMS host frame size in bytes by nms info and buffer format.

Parameters

- `nms_info` – **[in]** A *hailo_nms_info_t* object.
- `format` – **[in]** A *hailo_format_t* object.

Returns The NMS host frame size in bytes.

static inline constexpr uint32_t `get_nms_host_frame_size`(const *hailo_nms_shape_t* ...)

Gets NMS host frame size in bytes by nms shape and buffer format.

Parameters

- `nms_shape` – **[in]** A *hailo_nms_shape_t* object.
- `format` – **[in]** A *hailo_format_t* object.

Returns The NMS host frame size in bytes.

static inline constexpr uint32_t `get_nms_hw_frame_size`(const *hailo_nms_info_t* &nms_info)

Gets NMS hw frame size in bytes by nms info.

Parameters `nms_info` – **[in]** A *hailo_nms_info_t* object.

Returns The NMS hw frame size in bytes.

static inline constexpr uint32_t `get_frame_size`(const *hailo_3d_image_shape_t* &shape, const ...)

Gets frame size in bytes by image shape and format.

Parameters

- `shape` – **[in]** A *hailo_3d_image_shape_t* object.
- `format` – **[in]** A *hailo_format_t* object.

Returns The frame's size in bytes.

static inline constexpr uint32_t `get_frame_size`(const *hailo_stream_info_t* &stream_info, ...)

Gets frame size in bytes by stream info and transformation params.

Parameters

- `stream_info` – **[in]** A *hailo_stream_info_t* object.
- `trans_params` – **[in]** A *hailo_transform_params_t* object.

Returns The frame's size in bytes.

static inline constexpr uint32_t `get_frame_size`(const *hailo_vstream_info_t* &vstream_info, ...)

Gets frame size in bytes by stream info and transformation params.

Parameters

- `vstream_info` – **[in]** A *hailo_vstream_info_t* object.
- `format` – **[in]** A *hailo_format_t* object.

Returns The frame's size in bytes.

static inline constexpr bool `is_vdma_stream_interface`(*hailo_stream_interface_t* ...)

static *Expected*<*hailo_device_id_t*> `to_device_id`(const std::string &device_id)

static *Expected*<std::vector<*hailo_device_id_t*» `to_device_ids_vector`(const ...)

Public Static Attributes

static const uint32_t `BBOX_PARAMS` = sizeof(*hailo_bbox_t*) / sizeof(uint16_t)

static const uint32_t `MAX_DEFUSED_LAYER_COUNT` = 9

static const size_t `HW_DATA_ALIGNMENT` = 8

static const uint32_t `MUX_INFO_COUNT` = 32

static const uint32_t `MAX_MUX_PREDECESSORS` = 4

static const uint16_t `ETH_INPUT_BASE_PORT` = 32401

static const uint16_t `ETH_OUTPUT_BASE_PORT` = 32501

13.10. Runtime statistics

class `hailort::AccumulatorResults`

Results obtained by an *Accumulator* at a given point in time via *Accumulator::get_and_clear* or *Accumulator::get*

Public Functions

inline *Expected*<size_t> `count()` const

Returns Returns *Expected* of the number of datapoints added to the *Accumulator*.

inline *Expected*<double> `min()` const

Returns Returns *Expected* of the minimal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

inline *Expected*<double> `max()` const

Returns Returns *Expected* of the maximal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

inline *Expected*<double> `mean()` const

Returns Returns *Expected* of the mean of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

inline *Expected*<double> `var()` const

Returns Returns *Expected* of the sample variance of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

inline *Expected*<double> `sd()` const

Returns Returns *Expected* of the sample standard deviation of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

inline *Expected*<double> `mean_sd()` const

Returns Returns *Expected* of the sample standard deviation of the mean of values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

template<typename T, std::enable_if_t<std::is_arithmetic<*T*>::value, int> = 0>

class `hailort::Accumulator`

The *Accumulator* interface supports the measurement of various statistics incrementally. I.e. upon each addition of a measurement to the *Accumulator*, via *Accumulator::add_data_point*, all the statistics are updated. Implementations of this interface are to be thread-safe, meaning that adding measurements in one thread, while another thread reads the current statistics (via the various getters provided by the interface) will produce correct values.

Public Functions

inline `Accumulator(`const std::string &data_type`)`

Constructs a new *Accumulator* with a given *data_type*

Parameters `data_type` – The type of data that will be measured by the *Accumulator*. Used to differentiate between different types of measurements (e.g. fps, latency).

inline std::string `get_data_type()` const

Returns The data_type of the *Accumulator*.

virtual void `add_data_point(`*T* data, uint32_t samples_count = 1`)` = 0

Add a new measurement to the *Accumulator*, updating the statistics measured.

Note: Implementations of this interface are to update the statistics in constant time

Parameters

- `data` – The measurement to be added.
- `samples_count` – The weight of the measurement to be considered in average calculations.

virtual *AccumulatorResults* `get_and_clear()` = 0

Gets the current statistics of the data added to the *Accumulator*, clearing the statistics afterwards.

Returns The current statistics of the data added to the *Accumulator*

virtual *AccumulatorResults* `get()` const = 0

Returns The current statistics of the data added to the *Accumulator*

virtual *Expected*<size_t> `count()` const = 0

Returns The number of datapoints added to the *Accumulator*.

virtual *Expected*<double> `min()` const = 0

Returns Returns *Expected* of the minimal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

virtual *Expected*<double> `max()` const = 0

Returns Returns *Expected* of the maximal value added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

virtual *Expected*<double> `mean()` const = 0

Returns Returns *Expected* of the mean of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

virtual *Expected*<double> `var()` const = 0

Returns Returns *Expected* of the sample variance of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

virtual *Expected*<double> `sd()` const = 0

Returns Returns *Expected* of the sample standard deviation of the values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

virtual *Expected*<double> `mean_sd()` const = 0

Returns Returns *Expected* of the sample standard deviation of the mean of values added to the *Accumulator*, or *Unexpected* of *HAILO_UNINITIALIZED* if no data has been added.

13.11. Error Handling

class `Unexpected`

Unexpected is an object containing *hailo_status* error, used when an unexpected outcome occurred.

template<typename `T`>

class `hailort::Expected`

Expected<T> is either a T or the *hailo_status* preventing T to be created.

Public Functions

inline bool `has_value()` const

Checks whether the object contains a value.

inline *T* `&value()&`

Returns the contained value.

> **Note:** You must call this method with a valid value inside! otherwise it can lead to undefined behavior.

inline const *T* `&value()` const `&`

Returns the contained value.

> **Note:** You must call this method with a valid value inside! otherwise it can lead to undefined behavior.

inline *hailo_status* `status()` const

Returns the status.

inline *T* `release()`

Releases ownership of its stored value, by returning its value and making this object *Unexpected*.

> **Note:** You must call this method with a valid value inside! otherwise it can lead to undefined behavior.

13.12. UDP Rate Limiter

class `hailort::NetworkUdpRateCalculator`

Public Functions

Expected<std::map<std::string, uint32_t» `calculate_inputs_bandwith(` uint32_t fps, uint32_t ...)

Calculate the inputs bandwidths supported by the configured network. Rate limiting of this manner is to be used for ethernet input streams.

> **Note:** There are two options to limit the rate of an ethernet input stream to the desired bandwidth:
>
> - Set *hailo_eth_input_stream_params_t.rate_limit_bytes_per_sec* inside *hailo_stream_parameters_t*, under NetworkGroupsParamsMap before passing it to *Device::configure*.
> - On Unix platforms:
>   - You may use the command line tool `hailortcli udp-rate-limiter` instead of using this API

Parameters

- `fps` – **[in]** The desired fps.
- `max_supported_bandwidth` – **[in]** The maximun supported bandwidth. Neither the calculated input rate, nor the corresponding output rate will exceed this value. If no value is given, those rates will not exceed *HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC*.

Returns Upon success, returns *Expected* of a map of stream names and their corresponding rates. Otherwise, returns *Unexpected* of *hailo_status* error.

Expected<std::map<uint16_t, uint32_t» `get_udp_ports_rates_dict` (std::vector<std::reference_wrapper<*InputStream*»

Calculate the inputs bandwidths supported by the configured network, and returns a map of stream ports and their corresponding rates.

Parameters

- `udp_input_streams` – **[in]** UDP input streams.
- `fps` – **[in]** The desired fps.
- `max_supported_bandwidth` – **[in]** The maximun supported bandwidth. Neither the calculated input rate, nor the corresponding output rate will exceed this value. If no value is given, those rates will not exceed *HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC*.

Returns Upon success, returns *Expected* of a map of stream ports and their corresponding rates. Otherwise, returns *Unexpected* of *hailo_status* error.

13.13. Type Definitions

using `NotificationCallback` = std::function<void(Device &device, const *hailo_notification_t* &notification, void *opaque)>

14. HailoRT Python API Reference

14.1. hailo_platform.pyhailort.hw_object

Hailo hardware API

class hailo_platform.pyhailort.hw_object.InferenceTargets

Bases: object

Enum-like class with all inference targets supported by the HailoRT.

UNINITIALIZED = 'uninitialized'

UDP_CONTROLLER = 'udp'

PCIE_CONTROLLER = 'pcie'

exception hailo_platform.pyhailort.hw_object.HailoHWObjectException

Bases: Exception

Raised in any error related to Hailo hardware.

class hailo_platform.pyhailort.hw_object.HailoHWObject

Bases: object

Abstract Hailo hardware device representation (deprecated)

NAME = 'uninitialized'

IS_HARDWARE = True

__init__()

Create the Hailo hardware object.

property name

The name of this target. Valid values are defined by InferenceTargets (deprecated)

Type str

property is_hardware

Indicates this target runs on a physical hardware device. (deprecated)

Type bool

property device_id

Getter for the device_id. :returns: A string ID of the device. BDF for PCIe devices, IP address for Ethernet devices, "Core" for core devices. :rtype: str

property sorted_output_layer_names

Getter for the property sorted_output_names (deprecated). :returns: Sorted list of the output layer names. :rtype: list of str

use_device(**args*, ***kwargs*)

A context manager that wraps the usage of the device. (deprecated)

get_output_device_layer_to_original_layer_map()

Get a mapping between the device outputs to the layers' names they represent (deprecated).

Returns Keys are device output names and values are lists of layers' names.

Return type dict

`get_original_layer_to_device_layer_map()`

Get a mapping between the layer names and the device outputs that contain them (deprecated).

Returns Keys are the names of the layers and values are device outputs names.

Return type dict

`property device_input_layers`

Get a list of the names of the device's inputs. (deprecated)

`property device_output_layers`

Get a list of the names of the device's outputs. (deprecated)

`hef_loaded()`

Return True if this object has loaded the model HEF to the hardware device. (deprecated)

`outputs_count()`

Return the amount of output tensors that are returned from the hardware device for every input image (deprecated).

`property model_name`

Get the name of the current model (deprecated).

Returns Model name.

Return type str

`get_output_shapes()`

Get the model output shapes, as returned to the user (without any hardware padding) (deprecated).

Returns Tuple of output shapes, sorted by the output names.

`class hailo_platform.pyhailort.hw_object.HailoChipObject`

Bases: `hailo_platform.pyhailort.hw_object.HailoHWObject`

Hailo hardware device representation (deprecated)

`__init__()`

Create the Hailo Chip hardware object.

`property control`

Returns the control object of this device, which implements the control API of the Hailo device. .. attention:: Use the low level control API with care.

Type `HailoControl`

`get_all_input_layers_dtype()`

Get the model inputs dtype (deprecated).

Returns obj:'numpy.dtype': where the key is model input_layer name, and the value is dtype as the device expect to get for this input.

Return type dict of

`get_input_vstream_infos(`*network_name=None*`)`

Get input vstreams information of a specific network group (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`: with all the information objects of all input vstreams

`get_output_vstream_infos`(*network_name=None*)

Get output vstreams information of a specific network group (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`: with all the information objects of all output vstreams

`get_all_vstream_infos`(*network_name=None*)

Get input and output vstreams information (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`: with all the information objects of all input and output vstreams

`get_input_stream_infos`(*network_name=None*)

Get the input low-level streams information of a specific network group (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`: with information objects of all input low-level streams.

`get_output_stream_infos`(*network_name=None*)

Get the output low-level streams information of a specific network group (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`: with information objects of all output low-level streams.

`get_all_stream_infos`(*network_name=None*)

Get input and output streams information of a specific network group (deprecated).

Parameters `network_name` (`str, optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.pyhailort._pyhailort.StreamInfo`: with all the information objects of all input and output streams

`property loaded_network_groups`

Getter for the property _loaded_network_groups. :returns: List of the the configured network groups loaded on the device. :rtype: list of `ConfiguredNetwork`

`configure`(*hef*, *configure_params_by_name={}*)

Configures target device from HEF object. :param hef: HEF to configure the device from :type hef: `HEF` :param configure_params_by_name: Maps between each net_group_name to configure_params. If not provided, default params will be applied :type configure_params_by_name: dict, optional

`get_input_shape`(*name=None*)

Get the input shape (not padded) of a network (deprecated).

Parameters `name` (`str, optional`) – The name of the desired input. If a name is not provided, return the first input_dataflow shape.

Returns Tuple of integers representing the input_shape.

get_index_from_name(*name*)

Get the index in the output list from the name (deprecated).

Parameters name (str) – The name of the output.

Returns The index of the layer name in the output list.

Return type int

release()

Release the allocated resources of the device. This function should be called when working with the device not as context-manager. Note: After calling this function, the device will not be usable.

class hailo_platform.pyhailort.hw_object.EthernetDevice(*remote_ip*, ...)

Bases: hailo_platform.pyhailort.hw_object.HailoChipObject

Represents any Hailo hardware device that supports UDP control and dataflow (deprecated)

NAME = 'udp'

__init__(*remote_ip*, *remote_control_port=22401*)

Create the Hailo UDP hardware object.

Parameters

- remote_ip (str) – Device IP address.
- remote_control_port (int, optional) – UDP port to which the device listens for control. Defaults to 22401.

static scan_devices(*interface_name*, *timeout_seconds=3*)

Scans for all eth devices on a specific network interface.

Parameters

- interface_name (str) – Interface to scan.
- timeout_seconds (int, optional) – timeout for scan operation. Defaults to 3.

Returns IPs of scanned devices.

Return type list of str

property remote_ip

Return the IP of the remote device (deprecated).

class hailo_platform.pyhailort.hw_object.PcieDevice(*device_info=None*)

Bases: hailo_platform.pyhailort.hw_object.HailoChipObject

Hailo PCIe production device representation (deprecated)

NAME = 'pcie'

__init__(*device_info=None*)

Create the Hailo PCIe hardware object.

Parameters device_info (hailo_platform.pyhailort.pyhailort.PcieDeviceInfo, optional) – Device info to create, call PcieDevice.scan_devices() to get list of all available devices.

static scan_devices()

Scans for all pcie devices on the system (deprecated).

Returns list of hailo_platform.pyhailort.pyhailort.PcieDeviceInfo

14.2. hailo_platform.pyhailort.pyhailort

class hailo_platform.pyhailort.pyhailort.HailoRTException

Bases: Exception

class hailo_platform.pyhailort.pyhailort.UdpRecvError

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.InvalidProtocolVersionException

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTFirmwareControlFailedException

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTInvalidFrameException

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTUnsupportedOpcodeException

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTTimeout

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTStreamAborted

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HailoRTStreamAbortedByUser

Bases: hailo_platform.pyhailort.pyhailort.HailoRTException

class hailo_platform.pyhailort.pyhailort.HEF(*hef_source*)

Bases: object

Python representation of the Hailo Executable Format, which contains one or more compiled models.

__init__(*hef_source*)

Constructor for the HEF class.

Parameters hef_source (str or bytes) – The source from which the HEF object will be created. If the source type is *str*, it is treated as a path to an hef file. If the source type is *bytes*, it is treated as a buffer. Any other type will raise a ValueError.

get_networks_names(*network_group_name=None*)

Gets the names of all networks in a specific network group.

Parameters network_group_name (str, optional) – The name of the network group to access. If not given, first network_group is addressed.

Returns The names of the networks.

Return type list of str

property path

HEF file path.

get_network_group_names()

Get the names of the network groups in this HEF.

get_network_groups_infos()

Get information about the network groups in this HEF.

`get_input_vstream_infos(`*name=None*`)`

Get input vstreams information.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input vstreams.

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_output_vstream_infos(`*name=None*`)`

Get output vstreams information.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_all_vstream_infos(`*name=None*`)`

Get input and output vstreams information.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input and output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_input_stream_infos(`*name=None*`)`

Get the input low-level streams information.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with information objects of all input low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_output_stream_infos(`*name=None*`)`

Get the output low-level streams information of a specific network group.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with information objects of all output low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_all_stream_infos(`*name=None*`)`

Get input and output streams information of a specific network group.

Parameters `name (str, optional)` – The name of the network or network_group to access. In case network_group name is given, Address all networks of the given network_group. In case not given, first network_group is addressed.

Returns with all the information objects of all input and output streams

Return type list of `hailo_platform.pyhailort._pyhailort.StreamInfo`

get_sorted_output_names(*network_group_name=None*)

Get the names of the outputs in a network group. The order of names is determined by the SDK. If the network group is not given, the first one is used.

get_vstream_name_from_original_name(*original_name*, *network_group_name=None*)

Get vstream name from original layer name for a specific network group.

Parameters

- original_name (str) – The original layer name.
- network_group_name (str, optional) – The name of the network group to access. If not given, first network_group is addressed.

Returns the matching vstream name for the provided original name.

Return type str

get_original_names_from_vstream_name(*vstream_name*, *network_group_name=None*)

Get original names list from vstream name for a specific network group.

Parameters

- vstream_name (str) – The stream name.
- network_group_name (str, optional) – The name of the network group to access. If not given, first network_group is addressed.

Returns all the matching original layers names for the provided vstream name.

Return type list of str

get_vstream_names_from_stream_name(*stream_name*, *network_group_name=None*)

Get vstream names list from their underlying stream name for a specific network group.

Parameters

- stream_name (str) – The underlying stream name.
- network_group_name (str, optional) – The name of the network group to access. If not given, first network_group is addressed.

Returns All the matching vstream names for the provided stream name.

Return type list of str

get_stream_names_from_vstream_name(*vstream_name*, *network_group_name=None*)

Get stream name from vstream name for a specific network group.

Parameters

- vstream_name (str) – The name of the vstreams.
- network_group_name (str, optional) – The name of the network group to access. If not given, first network_group is addressed.

Returns All the underlying streams names for the provided vstream name.

Return type list of str

class hailo_platform.pyhailort.pyhailort.PcieDeviceInfo(*bus*, *device*, *func*, *...*)

Bases: hailo_platform.pyhailort._pyhailort.PcieDeviceInfo

Represents pcie device info, includeing domain, bus, device and function.

BOARD_LOCATION_HELP_STRING = 'Board location in the format of the command: "lspci -d 1e60: | cut -d\' \' -f1" ([<domain>]:<bus>:<device>.<func>). If not specified the first board is taken.'

__init__(*self: hailo_platform.pyhailort._pyhailort.PcieDeviceInfo*) → None

classmethod from_string(*board_location_str*)

Parse pcie device info BDF from string. The format is [<domain>]:<bus>:<device>.<func>

classmethod argument_type(*board_location_str*)

PcieDeviceInfo Argument type for argparse parsers

class hailo_platform.pyhailort.pyhailort.ConfiguredNetwork(*configured_network*)

Bases: object

Represents a network group loaded to the device.

__init__(*configured_network*)

get_networks_names()

activate(*network_group_params=None*)

Activate this network group in order to infer data through it.

Parameters network_group_params (hailo_platform.pyhailort._pyhailort.ActivateNetworkGroupParams, optional) – Network group activation params. If not given, default params will be applied,

Returns Context manager that returns the activated network group.

Return type ActivatedNetworkContextManager

Note: Usage of *activate* when scheduler enabled is deprecated. On this case, this function will return None and print deprecation warning.

wait_for_activation(*timeout_ms=None*)

Block until activated, or until timeout_ms is passed.

Parameters timeout_ms (int, optional) – Timeout value in milliseconds to wait for activation. Defaults to HAILO_INFINITE.

Raises HailoRTTimeout – In case of timeout.

static create_params()

Create activation params for network_group.

Returns hailo_platform.pyhailort._pyhailort.ActivateNetworkGroupParams.

property name

get_output_shapes()

get_sorted_output_names()

get_input_vstream_infos(*network_name=None*)

Get input vstreams information.

Parameters network_name (str, optional) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all input vstreams

Return type list of hailo_platform.pyhailort._pyhailort.VStreamInfo

`get_output_vstream_infos`(*network_name=None*)

Get output vstreams information.

Parameters `network_name` (`str`, `optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_all_vstream_infos`(*network_name=None*)

Get input and output vstreams information.

Parameters `network_name` (`str`, `optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all input and output vstreams

Return type list of `hailo_platform.pyhailort._pyhailort.VStreamInfo`

`get_input_stream_infos`(*network_name=None*)

Get the input low-level streams information of a specific network group.

Parameters `network_name` (`str`, `optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with information objects of all input low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_output_stream_infos`(*network_name=None*)

Get the output low-level streams information of a specific network group.

Parameters `network_name` (`str`, `optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with information objects of all output low-level streams.

Return type List of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_all_stream_infos`(*network_name=None*)

Get input and output streams information of a specific network group.

Parameters `network_name` (`str`, `optional`) – The name of the network to access. In case not given, all the networks in the network group will be addressed.

Returns with all the information objects of all input and output streams

Return type list of `hailo_platform.pyhailort._pyhailort.StreamInfo`

`get_udp_rates_dict`(*fps*, *max_supported_rate_bytes*)

`get_stream_names_from_vstream_name`(*vstream_name*)

Get stream name from vstream name for a specific network group.

Parameters `vstream_name` (`str`) – The name of the vstreams.

Returns All the underlying streams names for the provided vstream name.

Return type list of str

`get_vstream_names_from_stream_name`(*stream_name*)

Get vstream names list from their underlying stream name for a specific network group.

Parameters `stream_name` (`str`) – The underlying stream name.

Returns All the matching vstream names for the provided stream name.

Return type list of str

`set_scheduler_timeout(`*timeout_ms*, *network_name=None*`)`

Sets the maximum time period that may pass before getting run time from the scheduler, even without reaching the minimum required send requests (e.g. threshold - see set_scheduler_threshold()), as long as at least one send request has been sent. This time period is measured since the last time the scheduler gave this network group run time.

Parameters `timeout_ms` (`int`) – Timeout in milliseconds.

`set_scheduler_threshold(`*threshold*`)`

Sets the minimum number of send requests required before the network is considered ready to get run time from th If at least one send request has been sent, but the threshold is not reached within a set time period (e.g. timeout - see hailo_set_scheduler_timeout()), the scheduler will consider the network ready regardless.

Parameters `threshold` (`int`) – Threshold in number of frames.

`set_scheduler_priority(`*priority*`)`

Sets the priority of the network. When the model scheduler will choose the next network, networks with higher priority will be prioritized in the selection. bigger number represent higher priority.

Parameters `priority` (`int`) – Priority as a number between HAILO_SCHEDULER_PRIORITY_MIN - HAILO_SCHEDULER_PRIORITY_MAX.

`class hailo_platform.pyhailort.pyhailort.ActivatedNetworkContextManager(`*configured_network*, ...

Bases: `object`

A context manager that returns the activated network group upon enter.

`__init__(`*configured_network*, *activated_network*`)`

`class hailo_platform.pyhailort.pyhailort.ActivatedNetwork(`*configured_network*, ...`)`

Bases: `object`

The network group that is currently activated for inference.

`__init__(`*configured_network*, *activated_network*`)`

`get_number_of_invalid_frames(`*clear=True*`)`

Returns number of invalid frames.

Parameters `clear` (`bool`) – If set, the returned value will be the number of invalid frames read since the last call to this function.

Returns Number of invalid frames.

Return type int

`validate_all_frames_are_valid()`

Validates that all of the frames so far are valid (no invalid frames).

`class hailo_platform.pyhailort.pyhailort.FormatType`

Bases: `pybind11_builtins.pybind11_object`

Data formats accepted by HailoRT.

Members:

AUTO : Chosen automatically to match the format expected by the device, usually UINT8.

UINT8

UINT16

FLOAT32

AUTO = <FormatType.AUTO: 0>

FLOAT32 = <FormatType.FLOAT32: 3>

UINT16 = <FormatType.UINT16: 2>

UINT8 = <FormatType.UINT8: 1>

__init__(*self:* hailo_platform.pyhailort._pyhailort.FormatType, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.PowerMeasurementData

Bases: pybind11_builtins.pybind11_object

__init__(**args*, ***kwargs*)

property average_time_value_milliseconds

float, Average time in milliseconds between sampels

property average_value

float, The average value of the samples that were sampled

equals(*self:* hailo_platform.pyhailort._pyhailort.PowerMeasurementData, *arg0:* ...)

property max_value

float, The maximun value of the samples that were sampled

property min_value

float, The minimum value of the samples that were sampled

property total_number_of_samples

uint, The number of samples that were sampled

class hailo_platform.pyhailort.pyhailort.MeasurementBufferIndex

Bases: pybind11_builtins.pybind11_object

Enum-like class representing all FW buffers for power measurements storing.

Members:

MEASUREMENT_BUFFER_INDEX_0

MEASUREMENT_BUFFER_INDEX_1

MEASUREMENT_BUFFER_INDEX_2

MEASUREMENT_BUFFER_INDEX_3

MEASUREMENT_BUFFER_INDEX_0 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0: 0>

MEASUREMENT_BUFFER_INDEX_1 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_1: 1>

MEASUREMENT_BUFFER_INDEX_2 =
<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_2: 2>

MEASUREMENT_BUFFER_INDEX_3 = <MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_3: 3>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MeasurementBufferIndex, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.PowerMeasurementTypes

Bases: pybind11_builtins.pybind11_object

Enum-like class representing the different power measurement types. This determines what would be measured on the device.

Members:

AUTO : Choose the default value according to the supported features.

SHUNT_VOLTAGE : Measure the shunt voltage. Unit is mV

BUS_VOLTAGE : Measure the bus voltage. Unit is mV

POWER : Measure the power. Unit is W

CURRENT : Measure the current. Unit is mA

AUTO = <PowerMeasurementTypes.AUTO: 2147483647>

BUS_VOLTAGE = <PowerMeasurementTypes.BUS_VOLTAGE: 1>

CURRENT = <PowerMeasurementTypes.CURRENT: 3>

POWER = <PowerMeasurementTypes.POWER: 2>

SHUNT_VOLTAGE = <PowerMeasurementTypes.SHUNT_VOLTAGE: 0>

__init__(*self:* hailo_platform.pyhailort._pyhailort.PowerMeasurementTypes, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.DvmTypes

Bases: pybind11_builtins.pybind11_object

Enum-like class representing the different DVMs that can be measured. This determines the device that would be measured.

Members:

AUTO : Choose the default value according to the supported features.

VDD_CORE : Perform measurements over the core. Exists only in Hailo-8 EVB.

VDD_IO : Perform measurements over the IO. Exists only in Hailo-8 EVB.

MIPI_AVDD : Perform measurements over the MIPI avdd. Exists only in Hailo-8 EVB.

MIPI_AVDD_H : Perform measurements over the MIPI avdd_h. Exists only in Hailo-8 EVB.

USB_AVDD_IO : Perform measurements over the IO. Exists only in Hailo-8 EVB.

VDD_TOP : Perform measurements over the top. Exists only in Hailo-8 EVB.

USB_AVDD_IO_HV : Perform measurements over the USB_AVDD_IO_HV. Exists only in Hailo-8 EVB.

AVDD_H : Perform measurements over the AVDD_H. Exists only in Hailo-8 EVB.

SDIO_VDD_IO : Perform measurements over the SDIO_VDDIO. Exists only in Hailo-8 EVB.

OVERCURRENT_PROTECTION : Perform measurements over the OVERCURRENT_PROTECTION dvm. Exists only for Hailo-8 platforms supporting current monitoring (such as M.2 and mPCIe).

AUTO = <DvmTypes.AUTO: 2147483647>

AVDD_H = <DvmTypes.AVDD_H: 7>

MIPI_AVDD = <DvmTypes.MIPI_AVDD: 2>

MIPI_AVDD_H = <DvmTypes.MIPI_AVDD_H: 3>

OVERCURRENT_PROTECTION = <DvmTypes.OVERCURRENT_PROTECTION: 9>

SDIO_VDD_IO = <DvmTypes.SDIO_VDD_IO: 8>

USB_AVDD_IO = <DvmTypes.USB_AVDD_IO: 4>

USB_AVDD_IO_HV = <DvmTypes.USB_AVDD_IO_HV: 6>

VDD_CORE = <DvmTypes.VDD_CORE: 0>

VDD_IO = <DvmTypes.VDD_IO: 1>

VDD_TOP = <DvmTypes.VDD_TOP: 5>

__init__(*self:* hailo_platform.pyhailort._pyhailort.DvmTypes, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.SamplingPeriod

Bases: pybind11_builtins.pybind11_object

Enum-like class representing all bit options and related conversion times for each bit setting for Bus Voltage and Shunt Voltage.

Members:

PERIOD_140us : The sensor provides a new sampling every 140us.

PERIOD_204us : The sensor provides a new sampling every 204us.

PERIOD_332us : The sensor provides a new sampling every 332us.

PERIOD_588us : The sensor provides a new sampling every 588us.

PERIOD_1100us : The sensor provides a new sampling every 1100us.

PERIOD_2116us : The sensor provides a new sampling every 2116us.

PERIOD_4156us : The sensor provides a new sampling every 4156us.

PERIOD_8244us : The sensor provides a new sampling every 8244us.

PERIOD_1100us = <SamplingPeriod.PERIOD_1100us: 4>

PERIOD_140us = <SamplingPeriod.PERIOD_140us: 0>

PERIOD_204us = <SamplingPeriod.PERIOD_204us: 1>

PERIOD_2116us = <SamplingPeriod.PERIOD_2116us: 5>

PERIOD_332us = <SamplingPeriod.PERIOD_332us: 2>

PERIOD_4156us = <SamplingPeriod.PERIOD_4156us: 6>

PERIOD_588us = <SamplingPeriod.PERIOD_588us: 3>

PERIOD_8244us = <SamplingPeriod.PERIOD_8244us: 7>

__init__(*self:* hailo_platform.pyhailort._pyhailort.SamplingPeriod, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.AveragingFactor

Bases: pybind11_builtins.pybind11_object

Enum-like class representing all the AVG bit settings and related number of averages for each bit setting.

Members:

AVERAGE_1 : Each sample reflects a value of 1 sub-samples.

AVERAGE_4 : Each sample reflects a value of 4 sub-samples.

AVERAGE_16 : Each sample reflects a value of 16 sub-samples.

AVERAGE_64 : Each sample reflects a value of 64 sub-samples.

AVERAGE_128 : Each sample reflects a value of 128 sub-samples.

AVERAGE_256 : Each sample reflects a value of 256 sub-samples.

AVERAGE_512 : Each sample reflects a value of 512 sub-samples.

AVERAGE_1024 : Each sample reflects a value of 1024 sub-samples.

AVERAGE_1 = <AveragingFactor.AVERAGE_1: 0>

AVERAGE_1024 = <AveragingFactor.AVERAGE_1024: 7>

AVERAGE_128 = <AveragingFactor.AVERAGE_128: 4>

AVERAGE_16 = <AveragingFactor.AVERAGE_16: 2>

AVERAGE_256 = <AveragingFactor.AVERAGE_256: 5>

AVERAGE_4 = <AveragingFactor.AVERAGE_4: 1>

AVERAGE_512 = <AveragingFactor.AVERAGE_512: 6>

AVERAGE_64 = <AveragingFactor.AVERAGE_64: 3>

__init__(*self:* hailo_platform.pyhailort._pyhailort.AveragingFactor, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.MipiDataTypeRx

Bases: pybind11_builtins.pybind11_object

Members:

RGB_444

RGB_555

RGB_565

RGB_666

RGB_888

RAW_6

RAW_7

RAW_8

RAW_10

RAW_12

RAW_14

RAW_10 = <MipiDataTypeRx.RAW_10: 43>

RAW_12 = <MipiDataTypeRx.RAW_12: 44>

RAW_14 = <MipiDataTypeRx.RAW_14: 45>

RAW_6 = <MipiDataTypeRx.RAW_6: 40>

RAW_7 = <MipiDataTypeRx.RAW_7: 41>

RAW_8 = <MipiDataTypeRx.RAW_8: 42>

RGB_444 = <MipiDataTypeRx.RGB_444: 32>

RGB_555 = <MipiDataTypeRx.RGB_555: 33>

RGB_565 = <MipiDataTypeRx.RGB_565: 34>

RGB_666 = <MipiDataTypeRx.RGB_666: 35>

RGB_888 = <MipiDataTypeRx.RGB_888: 36>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MipiDataTypeRx, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.MipiPixelsPerClock

Bases: pybind11_builtins.pybind11_object

Members:

PIXELS_PER_CLOCK_1

PIXELS_PER_CLOCK_2

PIXELS_PER_CLOCK_4

PIXELS_PER_CLOCK_1 = <MipiPixelsPerClock.PIXELS_PER_CLOCK_1: 0>

PIXELS_PER_CLOCK_2 = <MipiPixelsPerClock.PIXELS_PER_CLOCK_2: 1>

PIXELS_PER_CLOCK_4 = <MipiPixelsPerClock.PIXELS_PER_CLOCK_4: 2>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MipiPixelsPerClock, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.MipiClockSelection

Bases: pybind11_builtins.pybind11_object

Members:

SELECTION_80_TO_100_MBPS

SELECTION_100_TO_120_MBPS

SELECTION_120_TO_160_MBPS

SELECTION_160_TO_200_MBPS

SELECTION_200_TO_240_MBPS

SELECTION_240_TO_280_MBPS

SELECTION_280_TO_320_MBPS

SELECTION_320_TO_360_MBPS

SELECTION_360_TO_400_MBPS

SELECTION_400_TO_480_MBPS

SELECTION_480_TO_560_MBPS

SELECTION_560_TO_640_MBPS

SELECTION_640_TO_720_MBPS

SELECTION_720_TO_800_MBPS

SELECTION_800_TO_880_MBPS

SELECTION_880_TO_1040_MBPS

SELECTION_1040_TO_1200_MBPS

SELECTION_1200_TO_1350_MBPS

SELECTION_1350_TO_1500_MBPS

SELECTION_1500_TO_1750_MBPS

SELECTION_1750_TO_2000_MBPS

SELECTION_2000_TO_2250_MBPS

SELECTION_2250_TO_2500_MBPS

SELECTION_AUTOMATIC

```
SELECTION_100_TO_120_MBPS =
<MipiClockSelection.SELECTION_100_TO_120_MBPS: 1>

SELECTION_1040_TO_1200_MBPS =
<MipiClockSelection.SELECTION_1040_TO_1200_MBPS: 16>

SELECTION_1200_TO_1350_MBPS =
<MipiClockSelection.SELECTION_1200_TO_1350_MBPS: 17>

SELECTION_120_TO_160_MBPS =
<MipiClockSelection.SELECTION_120_TO_160_MBPS: 2>

SELECTION_1350_TO_1500_MBPS =
<MipiClockSelection.SELECTION_1350_TO_1500_MBPS: 18>

SELECTION_1500_TO_1750_MBPS =
<MipiClockSelection.SELECTION_1500_TO_1750_MBPS: 19>

SELECTION_160_TO_200_MBPS =
<MipiClockSelection.SELECTION_160_TO_200_MBPS: 3>

SELECTION_1750_TO_2000_MBPS =
<MipiClockSelection.SELECTION_1750_TO_2000_MBPS: 20>

SELECTION_2000_TO_2250_MBPS =
<MipiClockSelection.SELECTION_2000_TO_2250_MBPS: 21>
```


SELECTION_200_TO_240_MBPS =
<MipiClockSelection.SELECTION_200_TO_240_MBPS: 4>

SELECTION_2250_TO_2500_MBPS =
<MipiClockSelection.SELECTION_2250_TO_2500_MBPS: 22>

SELECTION_240_TO_280_MBPS =
<MipiClockSelection.SELECTION_240_TO_280_MBPS: 5>

SELECTION_280_TO_320_MBPS =
<MipiClockSelection.SELECTION_280_TO_320_MBPS: 6>

SELECTION_320_TO_360_MBPS =
<MipiClockSelection.SELECTION_320_TO_360_MBPS: 7>

SELECTION_360_TO_400_MBPS =
<MipiClockSelection.SELECTION_360_TO_400_MBPS: 8>

SELECTION_400_TO_480_MBPS =
<MipiClockSelection.SELECTION_400_TO_480_MBPS: 9>

SELECTION_480_TO_560_MBPS =
<MipiClockSelection.SELECTION_480_TO_560_MBPS: 10>

SELECTION_560_TO_640_MBPS =
<MipiClockSelection.SELECTION_560_TO_640_MBPS: 11>

SELECTION_640_TO_720_MBPS =
<MipiClockSelection.SELECTION_640_TO_720_MBPS: 12>

SELECTION_720_TO_800_MBPS =
<MipiClockSelection.SELECTION_720_TO_800_MBPS: 13>

SELECTION_800_TO_880_MBPS =
<MipiClockSelection.SELECTION_800_TO_880_MBPS: 14>

SELECTION_80_TO_100_MBPS =
<MipiClockSelection.SELECTION_80_TO_100_MBPS: 0>

SELECTION_880_TO_1040_MBPS =
<MipiClockSelection.SELECTION_880_TO_1040_MBPS: 15>

SELECTION_AUTOMATIC = <MipiClockSelection.SELECTION_AUTOMATIC: 63>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MipiClockSelection, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.MipiIspImageInOrder

Bases: pybind11_builtins.pybind11_object

Members:

B_FIRST

GB_FIRST

GR_FIRST

R_FIRST

B_FIRST = <MipiIspImageInOrder.B_FIRST: 0>

GB_FIRST = <MipiIspImageInOrder.GB_FIRST: 1>

GR_FIRST = <MipiIspImageInOrder.GR_FIRST: 2>

R_FIRST = <MipiIspImageInOrder.R_FIRST: 3>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MipiIspImageInOrder, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.MipiIspImageOutDataType

Bases: pybind11_builtins.pybind11_object

Members:

RGB_888

YUV_422

RGB_888 = <MipiIspImageOutDataType.RGB_888: 36>

YUV_422 = <MipiIspImageOutDataType.YUV_422: 30>

__init__(*self:* hailo_platform.pyhailort._pyhailort.MipiIspImageOutDataType, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.IspLightFrequency

Bases: pybind11_builtins.pybind11_object

Members:

LIGHT_FREQ_60_HZ

LIGHT_FREQ_50_HZ

LIGHT_FREQ_50_HZ = <IspLightFrequency.LIGHT_FREQ_50_HZ: 1>

LIGHT_FREQ_60_HZ = <IspLightFrequency.LIGHT_FREQ_60_HZ: 0>

__init__(*self:* hailo_platform.pyhailort._pyhailort.IspLightFrequency, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.Endianness

Bases: pybind11_builtins.pybind11_object

Members:

BIG_ENDIAN

LITTLE_ENDIAN

BIG_ENDIAN = <Endianness.BIG_ENDIAN: 0>

LITTLE_ENDIAN = <Endianness.LITTLE_ENDIAN: 1>

__init__(*self:* hailo_platform.pyhailort._pyhailort.Endianness, *value: int*) → None

property name

property value

class hailo_platform.pyhailort.pyhailort.InputVStreamParams

Bases: object

Parameters of an input virtual stream (host to device).

static make(*configured_network*, *quantized=True*, *format_type=None*, *timeout_ms=None*, ...)

Create input virtual stream params from a configured network group. These params determine the format of the data that will be fed into the network group.

Parameters

- configured_network (ConfiguredNetwork) – The configured network group for which the params are created.
- quantized (bool) – Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data. Defaults to True.
- format_type (FormatType) – The default format type of the data for all input virtual streams. If quantized is False, the default is FLOAT32. Otherwise, the default is AUTO, which means the data is fed in the same format expected by the device (usually uint8).
- timeout_ms (int) – The default timeout in milliseconds for all input virtual streams. Defaults to DEFAULT_VSTREAM_TIMEOUT_MS. In case of timeout, HailoRTTimeout will be raised.
- queue_size (int) – The pipeline queue size. Defaults to DEFAULT_VSTREAM_QUEUE_SIZE.
- network_name (str) – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

static make_from_network_group(*configured_network*, *quantized=True*, *format_type=None*, ...)

Create input virtual stream params from a configured network group. These params determine the format of the data that will be fed into the network group.

Parameters

- configured_network (ConfiguredNetwork) – The configured network group for which the params are created.
- quantized (bool) – Whether the data fed into the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data. Defaults to True.
- format_type (FormatType) – The default format type of the data for all input virtual streams. If quantized is False, the default is FLOAT32. Otherwise, the default is AUTO, which means the data is fed in the same format expected by the device (usually uint8).
- timeout_ms (int) – The default timeout in milliseconds for all input virtual streams. Defaults to DEFAULT_VSTREAM_TIMEOUT_MS. In case of timeout, HailoRTTimeout will be raised.
- queue_size (int) – The pipeline queue size. Defaults to DEFAULT_VSTREAM_QUEUE_SIZE.
- network_name (str) – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

class hailo_platform.pyhailort.pyhailort.OutputVStreamParams

Bases: object

Parameters of an output virtual stream (device to host).

static make(*configured_network*, *quantized=True*, *format_type=None*, *timeout_ms=None*, ...)

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group.

Parameters

- configured_network (ConfiguredNetwork) – The configured network group for which the params are created.
- quantized (bool) – Whether the data returned from the chip should be quantized. True means the data is still quantized. False means it's HailoRT's responsibility to de-quantize (rescale) the data. Defaults to True.
- format_type (FormatType) – The default format type of the data for all output virtual streams. If quantized is False, the default is FLOAT32. Otherwise, the default is AUTO, which means the returned data is in the same format returned from the device (usually uint8).
- timeout_ms (int) – The default timeout in milliseconds for all output virtual streams. Defaults to DEFAULT_VSTREAM_TIMEOUT_MS. In case of timeout, HailoRTTimeout will be raised.
- queue_size (int) – The pipeline queue size. Defaults to DEFAULT_VSTREAM_QUEUE_SIZE.
- network_name (str) – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

static make_from_network_group(*configured_network*, *quantized=True*, *format_type=None*, ...)

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group.

Parameters

- configured_network (ConfiguredNetwork) – The configured network group for which the params are created.
- quantized (bool) – Whether the data returned from the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data. Defaults to True.
- format_type (FormatType) – The default format type of the data for all output virtual streams. If quantized is False, the default is FLOAT32. Otherwise, the default is AUTO, which means the data is fed in the same format expected by the device (usually uint8).
- timeout_ms (int) – The default timeout in milliseconds for all output virtual streams. Defaults to DEFAULT_VSTREAM_TIMEOUT_MS. In case of timeout, HailoRTTimeout will be raised.
- queue_size (int) – The pipeline queue size. Defaults to DEFAULT_VSTREAM_QUEUE_SIZE.

- `network_name` (`str`) – Network name of the requested virtual stream params. If not passed, all the networks in the network group will be addressed.

Returns The created virtual streams params. The keys are the vstreams names. The values are the params.

Return type dict

`static make_groups(`*configured_network*, *quantized=True*, *format_type=None*, *timeout_ms=None*, ...`)`

Create output virtual stream params from a configured network group. These params determine the format of the data that will be returned from the network group. The params groups are splitted with respect to their underlying streams for multi process usges.

Parameters

- `configured_network` (`ConfiguredNetwork`) – The configured network group for which the params are created.
- `quantized` (`bool`) – Whether the data returned from the chip is already quantized. True means the data is already quantized. False means it's HailoRT's responsibility to quantize (scale) the data. Defaults to True.
- `format_type` (`FormatType`) – The default format type of the data for all output virtual streams. If quantized is False, the default is `FLOAT32`. Otherwise, the default is `AUTO`, which means the data is fed in the same format expected by the device (usually uint8).
- `timeout_ms` (`int`) – The default timeout in milliseconds for all output virtual streams. Defaults to DEFAULT_VSTREAM_TIMEOUT_MS. In case of timeout, `HailoRTTimeout` will be raised.
- `queue_size` (`int`) – The pipeline queue size. Defaults to DEFAULT_VSTREAM_QUEUE_SIZE.

Returns Each element in the list represent a group of params, where the keys are the vstreams names, and the values are the params. The params groups are splitted with respect to their underlying streams for multi process usges.

Return type list of dicts

`class hailo_platform.pyhailort.pyhailort.InputVStreams(`*configured_network*, ...`)`

Bases: `object`

Input vstreams pipelines that allows to send data, to be used as a context manager.

`__init__(`*configured_network*, *input_vstreams_params*`)`

Constructor for the InputVStreams class.

Parameters

- `configured_network` (`ConfiguredNetwork`) – The configured network group for which the pipeline is created.
- `input_vstreams_params` (dict from str to `InputVStreamParams`) – Params for the input vstreams in the pipeline.

`get(`*name=None*`)`

Return a single input vstream by its name.

Parameters `name` (`str`) – The vstream name. If name=None and there is a single input vstream, that single (`InputVStream`) will be returned. Otherwise, if name=None and there are multiple input vstreams, an exception will be thrown.

Returns The (`InputVStream`) that corresponds to the given name.

Return type `InputVStream`

clear()

Clears the vstreams' pipeline buffers.

class hailo_platform.pyhailort.pyhailort.OutputVStreams(*configured_network*, ...)

Bases: object

Output virtual streams pipelines that allows to receive data, to be used as a context manager.

__init__(*configured_network*, *output_vstreams_params*, *tf_nms_format=False*)

Constructor for the OutputVStreams class.

Parameters

- configured_network (ConfiguredNetwork) - The configured network group for which the pipeline is created.
- output_vstreams_params (dict from str to OutputVStreamParams) - Params for the output vstreams in the pipeline.
- tf_nms_format (bool, optional) - indicates whether the returned nms outputs should be in Hailo format or TensorFlow format. Default is False (using Hailo format).
  - Hailo format - list of numpy.ndarray. Each element represents th detections (bboxes) for the class, and its shape is [number_of_detections, BBOX_PARAMS]
  - TensorFlow format - numpy.ndarray of shape [class_count, BBOX_PARAMS, detections_count] padded with empty bboxes.

get(*name=None*)

Return a single output vstream by its name.

Parameters name (str) - The vstream name. If name=None and there is a single output vstream, that single (OutputVStream) will be returned. Otherwise, if name=None and there are multiple output vstreams, an exception will be thrown.

Returns The (OutputVStream) that corresponds to the given name.

Return type OutputVStream

clear()

Clears the vstreams' pipeline buffers.

class hailo_platform.pyhailort.pyhailort.InputVStream(*send_object*)

Bases: object

Represents a single virtual stream in the host to device direction.

__init__(*send_object*)

property shape

property dtype

property name

property network_name

send(*input_data*)

Send frames to inference.

Parameters input_data (numpy.ndarray) - Data to run inference on.

flush()

Blocks until there are no buffers in the input VStream pipeline.

property info

class hailo_platform.pyhailort.pyhailort.OutputVStream(*configured_network*, ...)

Bases: object

Represents a single output virtual stream in the device to host direction.

__init__(*configured_network*, *recv_object*, *name*, *tf_nms_format=False*, *net_group_name=''*)

property shape

property dtype

property name

property network_name

recv()

Receive frames after inference.

Returns The output of the inference for a single frame. The returned tensor does not include the batch dimension. In case of nms output and tf_nms_format=False, returns list of numpy.ndarray.

Return type numpy.ndarray

property info

class hailo_platform.pyhailort.pyhailort.InferVStreams(*configured_net_group*, ...)

Bases: object

Pipeline that allows to call blocking inference, to be used as a context manager.

__init__(*configured_net_group*, *input_vstreams_params*, *output_vstreams_params*, *tf_nms_format=False*)

Constructor for the InferVStreams class.

Parameters

- configured_net_group (ConfiguredNetwork) – The configured network group for which the pipeline is created.
- input_vstreams_params (dict from str to InputVStreamParams) – Params for the input vstreams in the pipeline. Only members of this dict will take part in the inference.
- output_vstreams_params (dict from str to OutputVStreamParams) – Params for the output vstreams in the pipeline. Only members of this dict will take part in the inference.
- tf_nms_format (bool, optional) – indicates whether the returned nms outputs should be in Hailo format or TensorFlow format. Default is False (using Hailo format).
  - Hailo format – list of numpy.ndarray. Each element represents the detections (bboxes) for the class, and its shape is [number_of_detections, BBOX_PARAMS]
  - TensorFlow format – numpy.ndarray of shape [class_count, BBOX_PARAMS, detections_count] padded with empty bboxes.

infer(*input_data*)

Run inference on the hardware device.

Parameters input_data (dict of numpy.ndarray) – Where the key is the name of the input_layer, and the value is the data to run inference on.

Returns Output tensors of all output layers. The keys are outputs names and the values are output data tensors as `numpy.ndarray` (or list of `numpy.ndarray` in case of nms output and tf_nms_format=False).

Return type dict

`get_hw_time()`

Get the hardware device operation time it took to run inference over the last batch.

Returns Time in seconds.

Return type float

`get_total_time()`

Get the total time it took to run inference over the last batch.

Returns Time in seconds.

Return type float

`class hailo_platform.pyhailort.pyhailort.BoardInformation(`*protocol_version*, ...`)`

Bases: `object`

`__init__(`*protocol_version*, *fw_version_major*, *fw_version_minor*, *fw_version_revision*, *logger_version*, ...`)`

`static get_hw_arch_str(`*device_arch*`)`

`class hailo_platform.pyhailort.pyhailort.CoreInformation(`*fw_version_major*, ...`)`

Bases: `object`

`__init__(`*fw_version_major*, *fw_version_minor*, *fw_version_revision*, *is_release*, *extended_context_switch_buffer*`)`

`class hailo_platform.pyhailort.pyhailort.ExtendedDeviceInformation(`*neural_network_core_clock_rat*

Bases: `object`

`__init__(`*neural_network_core_clock_rate*, *supported_features*, *boot_source*, *lcs*, *soc_id*, *eth_mac_address*, ...`)`

`class hailo_platform.pyhailort.pyhailort.TemperatureInfo`

Bases: `pybind11_builtins.pybind11_object`

`__init__(`**args*, ***kwargs*`)`

`equals(`*self:* hailo_platform.pyhailort._pyhailort.TemperatureInfo, *arg0:* ...`)`

`property sample_count`

`property ts0_temperature`

`property ts1_temperature`

`class hailo_platform.pyhailort.pyhailort.Control(`*device:* ...`)`

Bases: `object`

The control object of this device, which implements the control API of the Hailo device. Should be used only from Device.control

`WORD_SIZE = 4`

`__init__(`*device: hailo_platform.pyhailort._pyhailort.Device*`)`

`property device_id`

Getter for the device_id.

Returns A string ID of the device. BDF for PCIe devices, IP address for Ethernet devices, "Integrated" for integrated nnc devices.

Return type str

`open()`

Initializes the resources needed for using a control device.

`close()`

Releases the resources that were allocated for the control device.

`chip_reset()`

Resets the device (chip reset).

`nn_core_reset()`

Resets the nn_core.

`soft_reset()`

reloads the device firmware (soft reset)

`forced_soft_reset()`

reloads the device firmware (forced soft reset)

`read_memory`(*address*, *data_length*)

Reads memory from the Hailo chip. Byte order isn't changed. The core uses little-endian byte order.

Parameters

- `address` (`int`) – Physical address to read from.
- `data_length` (`int`) – Size to read in bytes.

Returns Memory read from the chip, each index in the list is a byte

Return type list of str

`write_memory`(*address*, *data_buffer*)

Write memory to Hailo chip. Byte order isn't changed. The core uses little-endian byte order.

Parameters

- `address` (`int`) – Physical address to write to.
- `data_buffer` (`list of str`) – Data to write.

`configure`(*hef*, *configure_params_by_name={}*)

Configures device from HEF object.

Parameters

- `hef` (`HEF`) – HEF to configure the device from.
- `configure_params_by_name` (`dict, optional`) - Maps between each net_group_name to configure_params. In case of a mismatch with net_groups_names, default params will be used.

`power_measurement`(*dvm=<DvmTypes.AUTO: 2147483647>*, ...)

Perform a single power measurement on an Hailo chip. It works with the default settings where the sensor returns a new value every 2.2 ms without averaging the values.

Parameters

- `dvm` (`DvmTypes`) – Which DVM will be measured. Default (`AUTO`) will be different according to the board:

  Default (`AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `VDD_CORE`, `MIPI_AVDD` and `AVDD_H`. Only `POWER` can measured with this option.

  Default (`AUTO`) for platforms supporting current monitoring (such as M.2 and mP-CIe): `OVERCURRENT_PROTECTION`

- `measurement_type` – (`PowerMeasurementTypes`): The type of the measurement.

Returns

The measured power.

For `PowerMeasurementTypes`:

- `SHUNT_VOLTAGE`: Unit is mV.
- `BUS_VOLTAGE`: Unit is mV.
- `POWER`: Unit is W.
- `CURRENT`: Unit is mA.

Return type float

Note: This function can perform measurements for more than just power. For all supported measurement types, please look at `PowerMeasurementTypes`.

`start_power_measurement`(*averaging_factor=<AveragingFactor.AVERAGE_256: 5>*, ...)

Start performing a long power measurement.

Parameters

- `averaging_factor` (`AveragingFactor`) – Number of samples per time period, sensor configuration value.
- `sampling_period` (`SamplingPeriod`) – Related conversion time, sensor configuration value. The sensor samples the power every `sampling_period` [ms] and averages every `averaging_factor` samples. The sensor provides a new value every: 2 * sampling_period * averaging_factor [ms]. The firmware wakes up every `delay` [ms] and checks the sensor. If there is a new' value to read from the sensor, the firmware reads it. Note that the average calculated by the firmware is "average of averages", because it averages values that have already been averaged by the sensor.

`stop_power_measurement`( )

Stop performing a long power measurement. Deletes all saved results from the firmware. Calling the function eliminates the start function settings for the averaging the samples, and returns to the default values, so the sensor will return a new value every 2.2 ms without averaging values.

`set_power_measurement`(*buffer_index=<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0:* ...)

Set parameters for long power measurement on an Hailo chip.

Parameters

- `buffer_index` (`MeasurementBufferIndex`) – Index of the buffer on the firmware the data would be saved at. Default is `MEASUREMENT_BUFFER_INDEX_0`
- `dvm` (`DvmTypes`) – Which DVM will be measured. Default (`AUTO`) will be different according to the board:

  Default (`AUTO`) for EVB is an approximation to the total power consumption of the chip in PCIe setups. It sums `VDD_CORE`, `MIPI_AVDD` and `AVDD_H`. Only `POWER` can measured with this option.

  Default (`AUTO`) for platforms supporting current monitoring (such as M.2 and mPCIe): `OVERCURRENT_PROTECTION`

- `measurement_type` – (`PowerMeasurementTypes`): The type of the measurement.

Note: This function can perform measurements for more than just power. For all supported measurement types view PowerMeasurementTypes

get_power_measurement(*buffer_index=<MeasurementBufferIndex.MEASUREMENT_BUFFER_INDEX_0: ...*)

Read measured power from a long power measurement

Parameters

- buffer_index (MeasurementBufferIndex) – Index of the buffer on the firmware the data would be saved at. Default is MEASUREMENT_BUFFER_INDEX_0
- should_clear (bool) – Flag indicating if the results saved at the firmware will be deleted after reading.

Returns

Object containing measurement data

For PowerMeasurementTypes:

- SHUNT_VOLTAGE: Unit is mV.
- BUS_VOLTAGE: Unit is mV.
- POWER: Unit is W.
- CURRENT: Unit is mA.

Return type PowerMeasurementData

Note: This function can perform measurements for more than just power. For all supported measurement types view PowerMeasurementTypes.

read_user_config()

Read the user configuration section as binary data.

Returns User config as a binary buffer.

Return type str

write_user_config(*configuration*)

Write the user configuration.

Parameters configuration (str) – A binary representation of a Hailo device configuration.

read_board_config()

Read the board configuration section as binary data.

Returns Board config as a binary buffer.

Return type str

write_board_config(*configuration*)

Write the static configuration.

Parameters configuration (str) – A binary representation of a Hailo device configuration.

identify()

Gets the Hailo chip identification.

Returns BoardInformation

`core_identify()`

Gets the Core Hailo chip identification.

Returns `CoreInformation`

`set_fw_logger(`*level*, *interface_mask*`)`

Configure logger level and interface of sending.

Parameters

- `level` (`FwLoggerLevel`) – The minimum logger level.
- `interface_mask` (`int`) – Output interfaces (mix of FwLoggerInterface).

`set_throttling_state(`*should_activate*`)`

Change throttling state of temperature protection and overcurrent protection components. In case that change throttling state of temperature protection didn't succeed, the change throttling state of overcurrent protection is executed.

Parameters `should_activate` (`bool`) – Should be true to enable or false to disable.

`get_throttling_state()`

Get the current throttling state of temperature protection and overcurrent protection components. If any throttling is enabled, the function return true.

Returns true if temperature or overcurrent protection throttling is enabled, false otherwise.

Return type bool

`i2c_write(`*slave*, *register_address*, *data*`)`

Write data to an I2C slave.

Parameters

- `slave` (`hailo_platform.pyhailort.i2c_slaves.I2CSlave`) – I2C slave configuration.
- `register_address` (`int`) – The address of the register to which the data will be written.
- `data` (`str`) – The data that will be written.

`i2c_read(`*slave*, *register_address*, *data_length*`)`

Read data from an I2C slave.

Parameters

- `slave` (`hailo_platform.pyhailort.i2c_slaves.I2CSlave`) – I2C slave configuration.
- `register_address` (`int`) – The address of the register from which the data will be read.
- `data_length` (`int`) – The number of bytes to read.

Returns Data read from the I2C slave.

Return type str

`read_register(`*address*`)`

Read the value of a register from a given address.

Parameters `address` (`int`) – Address to read register from.

Returns Value of the register

Return type int

`set_bit`(*address*, *bit_index*)

Set (turn on) a specific bit at a register from a given address.

Parameters

- `address` (`int`) – Address of the register to modify.
- `bit_index` (`int`) – Index of the bit that would be set.

`reset_bit`(*address*, *bit_index*)

Reset (turn off) a specific bit at a register from a given address.

Parameters

- `address` (`int`) – Address of the register to modify.
- `bit_index` (`int`) – Index of the bit that would be reset.

`firmware_update`(*firmware_binary*, *should_reset=True*)

Update firmware binary on the flash.

Parameters

- `firmware_binary` (`bytes`) – firmware binary stream.
- `should_reset` (`bool`) – Should a reset be performed after the update (to load the new firmware)

`second_stage_update`(*second_stage_binary*)

Update second stage binary on the flash

Parameters `second_stage_binary` (`bytes`) – second stage binary stream.

`store_sensor_config`(*section_index*, *reset_data_size*, *sensor_type*, *config_file_path*, *config_height=0*, ...)

Store sensor configuration to Hailo chip flash memory.

Parameters

- `section_index` (`int`) – Flash section index to write to. [0-6]
- `reset_data_size` (`int`) – Size of reset configuration.
- `sensor_type` (`SensorConfigTypes`) – Sensor type.
- `config_file_path` (`str`) – Sensor configuration file path.
- `config_height` (`int`) – Configuration resolution height.
- `config_width` (`int`) – Configuration resolution width.
- `config_fps` (`int`) – Configuration FPS.
- `config_name` (`str`) – Sensor configuration name.

`store_isp_config`(*reset_config_size*, *isp_static_config_file_path*, *isp_runtime_config_file_path*, ...)

Store sensor isp configuration to Hailo chip flash memory.

Parameters

- `reset_config_size` (`int`) – Size of reset configuration.
- `isp_static_config_file_path` (`str`) – Sensor isp static configuration file path.
- `isp_runtime_config_file_path` (`str`) – Sensor isp runtime configuration file path.
- `config_height` (`int`) – Configuration resolution height.
- `config_width` (`int`) – Configuration resolution width.

- `config_fps` (`int`) – Configuration FPS.
- `config_name` (`str`) – Sensor configuration name.

`get_sensor_sections_info()`

Get sensor sections info from Hailo chip flash memory.

Returns Sensor sections info read from the chip flash memory.

`sensor_set_generic_i2c_slave(`*slave_address*, *register_address_size*, *bus_index*, ... `)`

Set a generic I2C slave for sensor usage.

Parameters

- `sequence` (`int`) – Request/response sequence.
- `slave_address` (`int`) – The address of the I2C slave.
- `register_address_size` (`int`) – The size of the offset (in bytes).
- `bus_index` (`int`) – The number of the bus the I2C slave is behind.
- `should_hold_bus` (`bool`) – Hold the bus during the read.
- `endianness` (`Endianness`) – Big or little endian.

`set_sensor_i2c_bus_index(`*sensor_type*, *i2c_bus_index* `)`

Set the I2C bus to which the sensor of the specified type is connected.

Parameters

- `sensor_type` (`SensorConfigTypes`) – The sensor type.
- `i2c_bus_index` (`int`) – The I2C bus index of the sensor.

`load_and_start_sensor(`*section_index* `)`

Load the configuration with I2C in the section index.

Parameters `section_index` (`int`) – Flash section index to load config from. [0-6]

`reset_sensor(`*section_index* `)`

Reset the sensor that is related to the section index config.

Parameters `section_index` (`int`) – Flash section index to reset. [0-6]

`wd_enable(`*cpu_id* `)`

Enable firmware watchdog.

Parameters `cpu_id` (`HailoCpuId`) – 0 for App CPU, 1 for Core CPU.

`wd_disable(`*cpu_id* `)`

Disable firmware watchdog.

Parameters `cpu_id` (`HailoCpuId`) – 0 for App CPU, 1 for Core CPU.

`wd_config(`*cpu_id*, *wd_cycles*, *wd_mode* `)`

Configure a firmware watchdog.

Parameters

- `cpu_id` (`HailoCpuId`) – 0 for App CPU, 1 for Core CPU.
- `wd_cycles` (`int`) – number of cycles until watchdog is triggered.
- `wd_mode` (`int`) – 0 - HW/SW mode, 1 - HW only mode

`previous_system_state(`*cpu_id* `)`

Read the FW previous system state.

Parameters `cpu_id` (`HailoCpuId`) – 0 for App CPU, 1 for Core CPU.

`get_chip_temperature()`

Returns the latest temperature measurements from the 2 internal temperature sensors of the Hailo chip.

Returns Temperature in celsius of the 2 internal temperature sensors (TS), and a sample count (a running 16-bit counter)

Return type `TemperatureInfo`

`get_extended_device_information()`

Returns extended information about the device

Return type `ExtendedDeviceInformation`

`set_pause_frames(`*rx_pause_frames_enable*`)`

Enable/Disable Pause frames.

Parameters `rx_pause_frames_enable` (`bool`) – False for disable, True for enable.

`test_chip_memories()`

test all chip memories using smart BIST

`set_notification_callback(`*callback_func*, *notification_id*, *opaque*`)`

Set a callback function to be called when a notification is received.

Parameters

- `callback_func` (`function`) – Callback function with the parameters (device, notification, opaque). Note that throwing exceptions is not supported and will cause the program to terminate with an error!
- `notification_id` (`NotificationId`) – Notification ID to register the callback to.
- `opauqe` (`object`) – User defined data.

Note: The notifications thread is started and closed in the use_device() context, so notifications can only be received there.

`remove_notification_callback(`*notification_id*`)`

Remove a notification callback which was already set.

Parameters `notification_id` (`NotificationId`) – Notification ID to remove the callback from.

`class hailo_platform.pyhailort.pyhailort.Device(`*device_id=None*`)`

Bases: `object`

Hailo device object representation (for inference use VDevice)

`classmethod scan()`

Scans for all devices on the system.

Returns list of str, device ids.

`__init__(`*device_id=None*`)`

Create the Hailo device object.

Parameters `device_id` (`str`) – Device id string, can represent several device types: [-] for pcie devices - pcie bdf (XXXX:XX:XX.X) [-] for ethernet devices - ip address (xxx.xxx.xxx.xxx)

`release()`

Release the allocated resources of the device. This function should be called when working with the device not as context-manager.

property device_id

Getter for the device_id.

Returns A string ID of the device. BDF for PCIe devices, IP address for Ethernet devices, "Integrated" for integrated nnc devices.

Return type str

configure(*hef*, *configure_params_by_name={}*)

Configures target device from HEF object.

Parameters

- hef (HEF) – HEF to configure the vdevice from
- configure_params_by_name (dict, optional) - Maps between each net_group_name to configure_params. If not provided, default params will be applied

Note: This function is deprecated. Support will be removed in future versions.

property control

Returns the control object of this device, which implements the control API of the Hailo device.

Return type Control

Attention: Use the low level control API with care.

read_log(*count*, *cpu_id*)

Returns Returns buffer with debug log data.

Parameters

- count (int) – bytes count to read
- cpu_id (HailoCpuId) – cpu id

property loaded_network_groups

Getter for the property _loaded_network_groups.

Returns List of the the configured network groups loaded on the device.

Return type list of ConfiguredNetwork

class hailo_platform.pyhailort.pyhailort.VDevice(*params=None*, ***, *device_ids=None*)

Bases: object

Hailo virtual device representation.

__init__(*params=None*, ***, *device_ids=None*)

Create the Hailo virtual device object.

Parameters

- params (hailo_platform.pyhailort.pyhailort.VDeviceParams, optional) – VDevice params, call VDevice.create_params() to get default params. Excludes 'device_ids'.
- device_ids (list of str, optional) – devices ids to create VDevice from, call Device.scan() to get list of all available devices. Excludes 'params'. Cannot be used together with device_id.

release()

Release the allocated resources of the device. This function should be called when working with the device not as context-manager.

static create_params()

configure(*hef*, *configure_params_by_name={}*)

Configures target vdevice from HEF object.

Parameters

- hef (HEF) – HEF to configure the vdevice from
- configure_params_by_name (dict, optional) – Maps between each net_group_name to configure_params. If not provided, default params will be applied

get_physical_devices()

Gets the underlying physical devices.

Returns The underlying physical devices.

Return type list of Device

get_physical_devices_ids()

Gets the physical devices ids.

Returns The underlying physical devices infos.

Return type list of str

property loaded_network_groups

Getter for the property _loaded_network_groups.

Returns List of the the configured network groups loaded on the device.

Return type list of ConfiguredNetwork

class hailo_platform.pyhailort.pyhailort.HailoRTTransformUtils

Bases: object

static get_dtype(*data_bytes*)

Get data type from the number of bytes.

static dequantize_output_buffer(*src_buffer*, *dst_buffer*, *elements_count*, *quant_info*)

De-quantize the data in input buffer *src_buffer* and output it to the buffer *dst_buffer*

Parameters

- src_buffer (numpy.ndarray) – The input buffer containing the data to be de-quantized. The buffer's data type is the source data type.
- dst_buffer (numpy.ndarray) – The buffer that will contain the de-quantized data. The buffer's data type is the destination data type.
- elements_count (int) – The number of elements to de-quantize. This number must not exceed 'src_buffer' or 'dst_buffer' sizes.
- quant_info (QuantInfo) – The quantization info.

static dequantize_output_buffer_in_place(*raw_buffer*, *dst_dtype*, *elements_count*, ...)

De-quantize the output buffer *raw_buffer* to data type *dst_dtype*.

Parameters

- raw_buffer (numpy.ndarray) – The output buffer to be de-quantized. The buffer's data type is the source data type.
- dst_dtype (numpy.dtype) – The data type to de-quantize *raw_buffer* to.

- `elements_count` (`int`) – The number of elements to de-quantize. This number must not exceed 'raw_buffer' size.
- `quant_info` (`QuantInfo`) – The quantization info.

`static quantize_input_buffer(`*src_buffer*, *dst_buffer*, *elements_count*, *quant_info*`)`

Quantize the data in input buffer *src_buffer* and output it to the buffer *dst_buffer*

Parameters

- `src_buffer` (`numpy.ndarray`) – The input buffer containing the data to be quantized. The buffer's data type is the source data type.
- `dst_buffer` (`numpy.ndarray`) – The buffer that will contain the quantized data. The buffer's data type is the destination data type.
- `elements_count` (`int`) – The number of elements to quantize. This number must not exceed 'src_buffer' or 'dst_buffer' sizes.
- `quant_info` (`QuantInfo`) – The quantization info.

14.3. hailo_platform.pyhailort.control_object

Control operations for the Hailo hardware device.

`exception hailo_platform.pyhailort.control_object.ControlObjectException`

Bases: `Exception`

Raised on illegal ContolObject operation.

`exception hailo_platform.pyhailort.control_object.FirmwareUpdateException`

Bases: `Exception`

`class hailo_platform.pyhailort.control_object.HailoControl(`*device: ...*`)`

Bases: `hailo_platform.pyhailort.pyhailort.Control`

Control object that sends control operations to a Hailo hardware device.

`class hailo_platform.pyhailort.control_object.HcpControl(`*device: ...*`)`

Bases: `hailo_platform.pyhailort.control_object.HailoControl`

Control object that uses the HCP protocol for controlling the device.

`class hailo_platform.pyhailort.control_object.UdpHcpControl(`*remote_ip*, *...*`)`

Bases: `hailo_platform.pyhailort.control_object.HcpControl`

Control object that uses a HCP over UDP controller interface.

`__init__(`*remote_ip*, *device=None*, *remote_control_port=22401*, *retries=2*, *response_timeout_seconds=10.0*, *...*`)`

Initializes a new UdpControllerControl object.

Parameters

- `remote_ip` (`str`) – The IPv4 address of the remote Hailo device (X.X.X.X).
- `remote_control_port` (`int, optional`) – The port that the remote Hailo device listens on.
- `response_timeout_seconds` (`float, optional`) – Number of seconds to wait until a response is received.
- `ignore_socket_errors` (`bool, optional`) – Ignore socket error (might be usefull for debugging).

class hailo_platform.pyhailort.control_object.PcieHcpControl(*device=None*, ...)

Bases: hailo_platform.pyhailort.control_object.HcpControl

Control object that uses a HCP over PCIe controller interface.

__init__(*device=None*, *device_info=None*)

Initializes a new HailoPcieController object.

14.4. hailo_platform.pyhailort.hailo_controller.i2c_slaves

hailo_platform.pyhailort.i2c_slaves.NO_I2C_SWITCH = 5

Variable which defines that the I2C slave is not behind a switch.

exception hailo_platform.pyhailort.i2c_slaves.I2CSlavesException

Bases: Exception

class hailo_platform.pyhailort.i2c_slaves.I2CSlave(*name*, *bus_index*, *slave_address*, ...)

Bases: object

__init__(*name*, *bus_index*, *slave_address*, *switch_number=5*, *register_address_size=1*, ...)

Initialize a class which describes an I2C slave.

Parameters

- name (str) – The name of the I2C slave.
- bus_index (int) – The bus number the I2C slave is connected to.
- slave_address (int) – The address of the I2C slave.
- switch_number (int) – The number of the switch the i2c salve is connected to.
- register_address_size (int) – Slave register address length (in bytes).
- endianness (Endianness) – The endianness of the slave.
- should_hold_bus (bool) – Should hold the bus during the read.

property name

Get the name of the I2C slave.

Returns Name of the I2C slave.

Return type str

property bus_index

Get bus index the I2C slave is connected to.

Returns Index of the bus the I2C slave is connected to.

Return type int

property slave_address

Get the address of the salve.

Returns The address of the I2C slave.

Return type int

property register_address_size

Get the slave register address length (in bytes). This number represents how many bytes are in the register address the slave can access.

Returns Slave register address length.

Return type int

Note: Pay attention to the slave endianness (`Endianness`).

property switch_number

Get the switch number the slave is connected to.

Returns The number of the switch the I2C is behind.

Return type int

Note: If `NO_I2C_SWITCH` is returned, it means the slave is not behind a switch.

property endianness

Get the slave endianness.

Returns The slave endianness.

Return type `Endianness`

property should_hold_bus

Returns a Boolean indicating if the bus will be held while reading from the slave.

Returns True if the bus would be held, otherwise False.

Return type bool

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_MIPI_AVDD

Class which represents the MIPI AVDD I2C slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_USB_AVDD_IO

Class which represents the USB AVDD IO slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_VDD_CORE

Class which represents the V_CORE slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_VDD_TOP

Class which represents the VDD TOP slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_MIPI_AVDD_H

Class which represents the MIPI AVDD_H I2C slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_USB_AVDD_IO_HV

Class which represents the DVM USB AVDD IO HV slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_VDD_IO

Class which represents the DVM_VDDIO slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_AVDD_H

Class which represents the DVM_AVDD_H slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_SDIO_VDD_IO

Class which represents the DVM_SDIO_VDDIO slave.

hailo_platform.pyhailort.i2c_slaves.
I2C_SLAVE_M_DOT_2_OVERCURREN_PROTECTION

Class which represents the DVM_SDIO_VDDIO slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_I2S_CODEC

Class which represents the I2S codec I2C slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_I2C_TO_GPIO

Class which represents the I2C to gpio I2C slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_SWITCH

Class which represents the I2C switch slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_TEMP_SENSOR_0

Class which represents the I2C TEMP_sensor_0 slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_TEMP_SENSOR_1

Class which represents the I2S TEMP_sensor_1 slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_EEPROM

Class which represents the EEPROM I2C slave.

hailo_platform.pyhailort.i2c_slaves.I2C_SLAVE_RASPICAM

Class which represents the raspicam I2C slave.

hailo_platform.pyhailort.i2c_slaves.set_i2c_switch(*control_object*, *slave*, ...)

Set the I2C switch in order to perform actions from the I2C slave.

Parameters

- control_object (HcpControl) – Control object which communicates with the Hailo chip.
- slave (I2CSlave) – Slave which the switch is set for.
- slave_switch (I2CSlave) – The I2C slave for the switch it self. Defaults to I2C_SLAVE_SWITCH.

14.5. hailo_platform.tools.udp_rate_limiter

Tool for limiting the packet sending rate via UDP. Needed to ensure the board will not get more traffic than it can handle, which would cause packet loss.

exception hailo_platform.tools.udp_rate_limiter.RateLimiterException

Bases: Exception

A problem has occurred during the rate setting.

exception hailo_platform.tools.udp_rate_limiter.BadTCParamError

Bases: Exception

One of shell's tc command params is wrong.

exception hailo_platform.tools.udp_rate_limiter.BadTCCallError

Bases: Exception

Shell's tc command has failed.

class hailo_platform.tools.udp_rate_limiter.RateLimiterWrapper(*configured_network_group*, ...)

Bases: object

UDPRateLimiter wrapper enabling with statements.

__init__(*configured_network_group*, *fps=1*, *fps_factor=1.0*, *remote_ip=None*)

RateLimiterWrapper constructor.

Parameters

- configured_network_group (ConfiguredNetwork) – The target network_group.
- fps (int) – Frame rate.
- fps_factor (float) – Safety factor by which to multiply the calculated UDP rate.

- `remote_ip` (`str`) – Device IP address.

class `hailo_platform.tools.udp_rate_limiter.UDPRateLimiter`(*remote_ip*, *port*, ...)

Bases: `object`

Enables limiting or removing limits on UDP communication rate to a board.

`__init__`(*remote_ip*, *port*, *rate_kbits_per_sec=0*)

`set_rate_limit()`

`reset_rate_limit()`

static `calc_udp_rate`(*hef*, *network_group_name*, *fps*, *fps_factor=1*, ...)

Calculates the proper UDP rate according to an HEF.

Parameters

- `hef` (`str`) – Path to an HEF file containing the network_group.
- `network_group_name` (`str`) – Name of the network_group to configure rates for.
- `fps` (`int`) – Frame rate.
- `fps_factor` (`float, optional`) – Safety factor by which to multiply the calculated UDP rate.
- `max_supported_kbps_rate` (`int, optional`) – Max supported Kbits per second. Defaults to 850 Mbit/s (850,000 Kbit/s).

Returns Maps between each input default dport to its calculated Rate in Kbits/sec.

Return type dict

Python Module Index

h

