



HailoRT User Guide

Release 4.0.1

9 September 2021

Table of Contents

I	User Guide	2
1	HailoRT Overview	3
1.1	Included in this package	3
1.2	HailoRT library (C/C++ API)	3
1.3	HailoRT Python API	3
1.4	PCIe driver	3
1.5	Yocto layer	3
2	Changelog	4
3	Installation	7
3.1	System requirements	7
3.2	Running HailoRT installation	8
3.3	Uninstalling HailoRT	9
3.4	HailoRT Python API configuration file	9
4	Installation of HailoRT inside Docker container	10
4.1	Running HailoRT container from pre-built Docker image	10
4.2	Building HailoRT image	10
5	Tutorials	13
5.1	C inference tutorial	13
5.2	Python inference tutorial	20
5.3	Python power measurement tutorial	23
6	Running Inference	25
6.1	Inference stages	25
6.2	Python inference API	26
6.3	C inference API	26
6.4	Context switching and user controlled switching	27
7	Command Line Tools	28
7.1	Firmware configuration tool	28
7.2	Firmware update tool	30
7.3	Firmware control tool	30
7.4	Scan tool	31
7.5	Inference	31
7.6	Benchmarking tool	32
7.7	Ethernet related command line tools	32
8	Yocto	34
8.1	The meta-hailo layer	34
8.2	Recipes	34
8.3	Integrating to an existing Yocto environment	35
8.4	Build your image	37
8.5	Validating the integration's success	38
II	API Reference	39
9	HailoRT C/C++ API Reference	40
9.1	Device and control API functions	40
9.2	HEF API functions	45

9.3	Network group API functions	48
9.4	Virtual stream API functions	51
9.5	Stream API functions	54
9.6	Transformation API functions	55
9.7	Power measurement API functions	58
9.8	Other API definitions	60
10	HailoRT Python API Reference	87
10.1	hailo_platform.drivers.hw_object	87
10.2	hailo_platform.drivers.hailort.pyhailort	90
10.3	hailo_platform.drivers.control_object	101
10.4	hailo_platform.drivers.hailo_controller.i2c_slaves	110
10.5	hailo_platform.tools.udp_rate_limiter	112
10.6	hailo_platform.common.targets.inference_targets	113
10.7	hailo_platform.tools.firmware.sensor_config	114
	Python Module Index	115

Disclaimer and Proprietary Information Notice

Copyright

© 2021 Hailo Technologies Ltd ("Hailo"). All Rights Reserved.

No part of this document may be reproduced or transmitted in any form without the expressed, written permission of Hailo. Nothing contained in this document should be construed as granting any license or right to use proprietary information for that matter, without the written permission of Hailo.

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

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
- PCIe driver
- Yocto layer
- Additional files such as an installation script and a configuration file

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

The HailoRT library implements a userspace C/C++ API that is called from the user's applications. It allows both to control the Hailo device and to send and receive data from it. It supports both the PCIe interface and the Ethernet interface.

1.3. HailoRT Python API

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

1.4. PCIe driver

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.

1.5. 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.

2. Changelog

HailoRT v4.0.1 (September 2021)

- Fixed a bug in the `hailortcli` benchmark tool when the HEF includes NMS in chip

HailoRT v4.0.0 LTS (September 2021)

- HailoRT plugin for the GStreamer framework is now included in the release (preview)
- C and Python API enhancements and fixes
- Temperature throttling is supported and enabled by default

Note: Version numbers 3.x have been skipped to avoid confusion with the Dataflow Compiler (SDK) version numbers.

HailoRT v2.10.0 (July 2021)

- *Virtual streams API*
- *User controlled switch* optimization that allows shorter switch times
- HailoRT *Docker* deployment support (preview)
- The Gatesgarth *Yocto* version is supported

Note: This version includes several new C and Python APIs, that allow easier application development using virtual streams and better HEF switch performance. All old APIs are still supported in this version, but some of them are expected to become deprecated in future versions. Deprecation warnings are printed when using these APIs.

HailoRT v2.9.0 (June 2021)

- *Context switch support* (preview). This feature allows to run big models that utilize more than a single device's resources, by splitting them into several contexts and switching between them over PCIe
- *User controlled switch* between different models (preview)
- Buffering long skip connections in the host's RAM over PCIe
- PCIe link power optimization. A new power mode parameter controls the power-performance trade-off in PCIe systems
- PCIe D3 power state support

Note: With some hosts, keeping the new optimized power mode `hailo_power_mode_t` default value (`HAILO_POWER_MODE_PERFORMANCE`) will not result in the highest possible performance. In such cases, the highest performance mode `HAILO_POWER_MODE_ULTRA_PERFORMANCE` can be chosen instead in order to reach it.

HailoRT v2.8.0 (May 2021)

- PCIe link's L1 ASPM low power mode is supported
- Additional [Yocto](#) versions are now supported: Zeus and Dunfell
- Enabled the pyhailort Yocto recipe (that was temporarily disabled in a previous version)
- Changed the evaluation board's default [power measurement](#) behavior to measure the full chip (not only the NN core)
- Python package code cleanup
- Enabled DKMS by default in the PCIe driver installation
- Reduced chip clock frequency configuration support (preview)
- Python 3.8 support

Note: With some devices, the host should be configured to enable the ASPM L1 link state. The [E21001 errata](#), which is available from Hailo, elaborates how to configure the host.

HailoRT v2.7.0 (April 2021)

- Improved the hailortcli command line tool
- Refactored the hailo and hailortcli command line tools to share most of the implementation
- Enabled DKMS support (that was temporarily disabled in the previous version)
- Added new C examples
- Old API cleanup
- Ethernet packet sizes are now configurable
- Ethernet pause frames support for the device's outputs
- Python 3.7 support

Note: The version numbering of the HailoRT Python package has been changed to 2.x.x (2.7.0 in this version) in order to align it with the version of the other HailoRT components such as libhailort.

HailoRT v2.6.0 (March 2021)

- Improved the hailortcli command line tool
- Performance improvements
- Fixed the PCIe driver compilation bug in certain Linux kernel versions (such as 5.3.0-62-generic)
- Added an uninstall script

Note: This version only supports the HEF format for compiled Hailo models. The HEF format is supported through both the PCIe and Ethernet interfaces. The older JLF format is deprecated.

Note: This version only supports Python 3.6. Python 3.5 is deprecated.

Note: The pyhailort Yocto recipe is temporarily unsupported in this version. It will be re-enabled in future releases.

Note: The PCIe driver's DKMS support is temporarily unsupported in this version. It will be re-enabled in future releases.

HailoRT v2.5.1 (February 2021)

- Buffering long skip connections in the host's RAM over PCIe (preview)
- Enabled the hardware watchdog that resets the device if it malfunctions

HailoRT v2.5.0 (February 2021)

- Bug fixes and stability improvements
- Error handling improvements and clearer error messages

HailoRT v2.4.0 (January 2021)

- HEF format support through the Ethernet interface
- hailortcli improvements
- NCHW <-> NHCW transformation support
- Logging improvements
- In-chip NMS runtime API improvements
- PCIe driver compilation now supports a flag to set the descriptor size

HailoRT v2.3.0 (December 2020)

- New HEF format support and new inference API that works with HEF binaries
- Power measurement support
- mPCIe board support
- Temperature and current monitoring support
- PCIe driver DKMS registration support
- New firmware version v2.3.0
- New CLI tool (hailortcli) that does not depend on Python
- Added a new user guide for HailoRT

Note: The HEF format is currently supported only via the PCIe interface. It is expected to replace the JLF format in the following versions. The JLF format is still supported in this version, through both PCIe and Ethernet, but it is expected to be deprecated.

3. Installation

3.1. System requirements

To install the HailoRT, the following minimum system is required:

- Linux
- x86_64 or aarch64 architecture
- 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
 - M.2 connector for the M.2 board
 - mPCIe connector for the mPCIe board
- Python 3.6, 3.7 or 3.8 (optional)

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

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

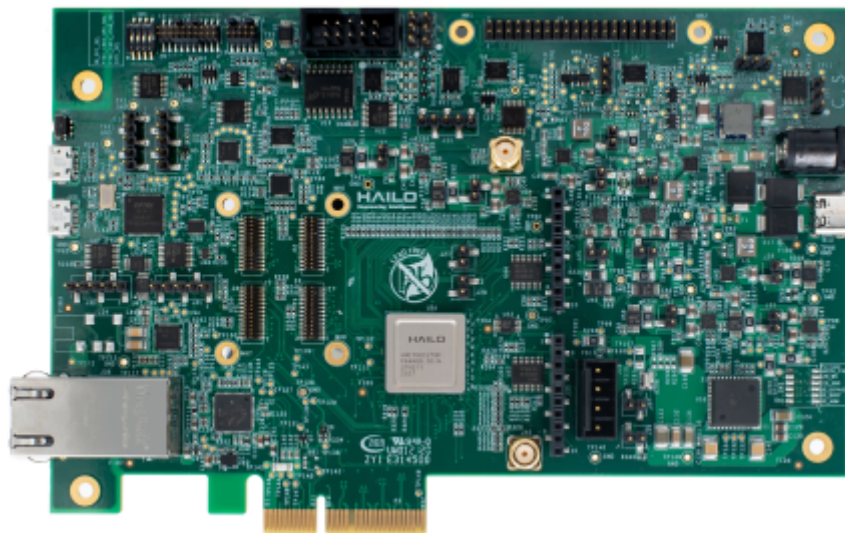


Figure 1: Hailo-8 Evaluation board



Figure 2: Hailo-8 M.2 board



Figure 3: Hailo-8 mPCIe board

3.2. Running HailoRT installation

The HailoRT package includes 3 files:

- `platform.tar.gz` – Archive of the HailoRT binaries and Python modules.
- `install.sh` – Installation script.
- `md5s.txt` – MD5 hashes of the HailoRT files.

3.2.1. Installation with the PCIe driver

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

```
chmod u+x install.sh
./install.sh
```

This script extracts the tar.gz archive and installs HailoRT. It will also compile the PCIe driver for the machine's kernel version and install the driver. The command will ask for root permissions (sudo) during the installation in order to install the driver.

Reboot the machine after the installation is done. The driver will be loaded automatically after reboot. Run the following commands in order to verify it:

```
source hailo_platform_venv/bin/activate
hailo 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.

Note: By default, the installation creates a new Python virtual environment, which requires an Internet connection (or a local pip server) in order to download Python packages.

3.2.2. Installation without the PCIe driver

Alternatively, run the following command to install HailoRT without the PCIe driver:

```
chmod u+x install.sh
./install.sh --skip-pcie-driver
```

Note: If the Hailo device is connected via Ethernet, the *Ethernet related shell scripts* are useful to configure the Ethernet connection in the host's side.

3.2.3. Additional installation flags

- Running `install.sh` with a `--justextract` flag will extract the archive without installing.
- The `--help` flag is used to display the all of the usage options of the installation script.
- Use the `--tutorials` option to install the Python dependencies needed to run the tutorials' Jupyter notebooks.
- Use the `--skip-dkms` flag to skip the DKMS installation.

3.3. Uninstalling HailoRT

Run the following commands to uninstall HailoRT:

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

3.4. HailoRT Python API configuration file

HailoRT's Python API configuration is defined in a file named `config`.

A default configuration file is included in the HailoRT package. It is copied to `~/.hailo` during installation. If the config file already exists at `~/.hailo`, the user will be prompted. HailoRT searches for the configuration using the following conditions: If a file named `config` exists in the current working directory, it is used. Otherwise, it looks at `~/.hailo/config`.

4. Installation of HailoRT inside Docker container

4.1. Running HailoRT container from pre-built Docker image

Using this method, the following intermediate steps are handled by 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
- dockerfile.hailort_run

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
```

The following step is required if the PC does not have working hailo_pci driver installed, or its version is different from the current one. Run the following command inside HailoRT container:

```
./install.sh --pcie-driver-only
```

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. Building HailoRT image

4.2.1. Creating the Dockerfile

Note: The following procedure has been tested with docker-ce (version 20.10.6) and docker.io (version 20.10.7)

Create a Dockerfile with the following content:

Note: TZ can be set to any existing Timezone.

```
FROM ubuntu:18.04

ENV WORKSPACE=/local/workspace
ENV PLATFORM_VENV=hailo_platform_venv
ENV TZ=Asia/Jerusalem
ENV PATH="${WORKSPACE}/${PLATFORM_VENV}/bin:$PATH"
ENV LD_LIBRARY_PATH="${WORKSPACE}/${PLATFORM_VENV}/lib"

RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime && echo $TZ > /etc/timezone
RUN mkdir -p /lib/firmware /lib/udev/rules.d /local/workspace
```

(continues on next page)

(continued from previous page)

```
RUN apt-get update && apt-get install -y linux-headers-$(uname -r)-generic \
    bsdmainutils \
    build-essential \
    cmake \
    dkms \
    graphviz \
    kmod \
    libgraphviz-dev \
    mokutil \
    python3.6-dev \
    python3-numpy \
    python3-tk \
    sudo \
    virtualenv
```

```
ADD . ${WORKSPACE}
WORKDIR ${WORKSPACE}
```

4.2.2. Building the Docker image

Note: The directory from which you build the image (your current working directory) should also contain HailoRT files:

- platform.tar.gz – Archive of the HailoRT binaries and Python modules.
- install.sh – Installation script.
- check_system_requirements.sh – System requirements check script.
- md5s.txt – MD5 hashes of the HailoRT files.

To build HailoRT Docker image run the following command:

```
docker build . -f Dockerfile -t hailo_docker_hailort
```

4.2.3. Creating the Docker container

To create the HailoRT Docker container from previously built image, run the following command:

```
docker run \
    --privileged \
    -v /dev:/dev \
    -v /lib:/lib \
    --net host \
    --name hailort_container \
    -ti hailo_docker_hailort
```

4.2.4. Running HailoRT installation

To install HailoRT, run the following command inside the container:

```
./install.sh
```

After installation is done, the PC should be restarted.

4.2.5. Confirming the installation

To confirm the installation run the following commands inside HailoRT docker container:

```
hailo fw-control identify  
hailo infer performance -t 10 \  
--streaming-mode hw-only \  
--config-path platform/tutorials/hefs/resnet_v1_18.hef
```

5. Tutorials

The tutorials below go through the inference steps in C and Python. The Python tutorials are also available as Jupyter notebook files in the directory *tutorials/notebooks/*.

Note: When installing only HailoRT (without the SDK), pass the `--tutorials` option to the `install.sh` installation script to install the necessary dependencies for Jupyter in the Python virtual environment.

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/examples/`.

All functions calls are based on the header provided in `hailort/include/hailo/hailort.h`. See the [API documentation](#) for more details.

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

```
cmake -H. -Bbuild [-DHAILO_TOOLCHAIN=<toolchain>]
cmake --build build
```

The optional argument `HAILO_TOOLCHAIN` specifies the toolchain file that will be used:

- Toolchain files include the host processor architecture and the tools that will be used to build the example (i.e. compiler and linker).
 - The variable should be of the form `<OS>.<ARCH>`:
 - All the toolchains currently supported are supplied under the directory `hailort/examples/cmake/toolchains/`.
 - E.g. `linux.x86_64` will use the `hailort/examples/cmake/toolchains/linux.x86_64.cmake` toolchain file.
 - If not provided, the toolchain will be set according to the host's OS and processor architecture (`uname -m`).
-

Running examples:

```
LD_LIBRARY_PATH=../lib/ build/<example_name> [params...]
```

Note: The environment variable `LD_LIBRARY_PATH` is set to `../lib/` to tell the dynamic link loader where to search for dynamic shared libraries.

5.1.1. Inference using virtual streams – *vstream_example*

See: `hailort/examples/vstream_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 PCIe device found.
- The data is sent to the device via an input vstream and received via an output vstream.
- The data is transformed before sent and after receiving in a different thread using the virtual stream pipeline.

Running the example

```
LD_LIBRARY_PATH=../lib/ build/vstream_example
```

Code structure

Device initialization We open the first enumerated Hailo PCIe device.

Used APIs: `hailo_create_pcie_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 gets a `hailo_configured_network_group` object.

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

Creating vstreams First we need to initialize vstream params (both input and output), then we are ready to create the vstreams. The vstreams will be ready only after activating them.

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 data sending, and the main thread for data receiving.

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

Used APIs: `hailo_activate_input_vstream()`, `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 and activate the output vstream. Then we will recv all frames in a loop.

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

5.1.2. Inference using raw streams – *stream_example*

See: `hailort/examples/stream_example.c`

Summary

- Demonstrates basic inference of a shortcut network using raw stream API.
- The program uses the first Hailo PCIe device found.
- The data is transformed before sent and after received in the same thread sending/receiving using the transformation api.

Running the example

```
LD_LIBRARY_PATH=../lib/ build/stream_example
```

Code structure

Device initialization We open the first enumerated Hailo PCIe device.

Used APIs: `hailo_create_pcie_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 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 data sending, and the main thread for data receiving.

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.
- `transformer` – a `hailo_input_transformer` - 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_transformer()` 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_transformer()` 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.
- `transformer` a `hailo_output_transformer` - 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_transformer()` 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_transformer()`.

5.1.3. Quantization of inputs/outputs – *data_quantization_example*

See: `hailort/examples/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_transformer` or `hailo_output_transformer`. See the *raw streams example* for more info about streams and transformations.

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

Running the example

```
LD_LIBRARY_PATH=../lib/ build/data_quantization_example [in/out/both/none]
```

- *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/examples/vstream_example.c`, so be sure to read the *vstreams example* first.

Device initialization and configuration We create a device 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/examples/vstream_example.c`. For an explanation, see the [vstreams example](#).

5.1.4. User controlled switch between multiple HEFs – *multiple_hefs_example*

See: `hailort/examples/multiple_hefs_example.c`

Summary

- Demonstrates basic inference of a multiple networks over single device.
- The program uses the first Hailo PCIe device found.
- Random input data is generated.
- Multiple HEFs are configured into the device.
- main loop run over several HEFs. Activate single HEFs each time, running infer on this HEFs, and deactivating it after.
- For each activated HEF, The data is sent to the device via an input stream and received via an output stream.

See also:

See the [comparison table](#) between context switching and user controlled switching for more information about running multiple models.

Running the example

```
LD_LIBRARY_PATH=../lib/ build/multiple_hefs_example
```

Code structure

The code is similar to `hailort/examples/stream_example.c`, so be sure to read the [stream_example](#) first.

Device initialization We open the first enumerated Hailo PCIe device.

Used APIs: `hailo_create_pcie_device()`

Pre infer stage

Configuring the device from HEF (for each 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()`.

Build streams (for each HEF) The next step is building the input and output streams for each configured network. for the selected network group, we get the input and output stream info. Using this info, we extract the stream, and allocating the host input and output buffers.

Used APIs: `hailo_network_group_get_input_stream_infos()`, `hailo_get_input_stream()`, `hailo_network_group_get_output_stream_infos()` and `hailo_get_output_stream()`.

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

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 sending data, and the main thread for receiving data.

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

- `input_stream` – the raw 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:

- First, prepare the data. In this example we randomize values in the [0-255] range.
- Then writing the HW buffer to the device.

Used APIs: `hailo_stream_write_raw_buffer()`.

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

- `output_stream` – the raw 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:

- First, reading data from the device.
- No post processing is made on the received data

Used APIs: `hailo_stream_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. User controlled switch between multiple HEFs using vstreams – *multiple_hefs_vstreams_example*

See: `hailort/examples/multiple_hefs_vstream_example.c`

Summary

- Demonstrates basic inference of a multiple networks over single device.
- The program uses the first Hailo PCIe device found.
- Random input data is generated.
- Multiple HEFs are configured into the device.
- For each network group, virtual streams are created.
- Main loop run over several HEFs. Activate single HEFs each time, running infer on this HEFs, and deactivating it after.
- For each activated HEF, The data is sent to the device via an input virtual stream and received via an output virtual stream.

See also:

See the [comparison table](#) between context switching and user controlled switching for more information about running multiple models.

Running the example

```
LD_LIBRARY_PATH=../lib/ build/multiple_hefs_vstream_example
```

Code structure

The code is similar to `hailort/examples/vstream_example.c` and `multiple_hefs_example.c`, so be sure to read the [stream_example](#) and [multiple_hefs_example](#) first.

Device initialization We open the first enumerated Hailo PCIe device.

Used APIs: `hailo_create_pcie_device()`.

Pre infer stage

Configuring the device from HEF (for each 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()`.

Build streams (for each HEF) 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 stream's frame size and allocating the host input and output buffers.

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

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 pthread for inference, with a sub-thread for sending data, and the main thread for receiving data.

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 in the following flow:

- Activate the virtual stream.
- Prepare the data. In this example we randomize values in the [0-255] range.
- Writing the HW buffer to the device.
- After all processing is done - the virtual stream is deactivated.

Used APIs: `hailo_activate_input_vstream()`, `hailo_stream_write_raw_buffer()` and `hailo_deactivate_input_vstream()`.

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:

- Activating the output virtual stream.
- Reading data from the device.
- No post processing is made on the received data.
- After all processing is done - the virtual stream is deactivated.

Used APIs: `hailo_activate_output_vstream()`, `hailo_stream_read_raw_buffer()` and `hailo_deactivate_output_vstream()`.

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

Used APIs: `hailo_deactivate_network_group()`.

5.2. 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`

5.2.1. Standalone hardware deployment

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

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

- Target HW configuration
- Weights
- Metadata for the platform SW (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, PcieDevice, 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 = PcieDevice()

# 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_from_network_group(network_group, quantized=False,
    ↪format_type=FormatType.FLOAT32)
output_vstreams_params = OutputVStreamParams.make_from_network_group(network_group, quantized=True,
    ↪format_type=FormatType.UINT8)

# Define dataset params
input_layer = hef.get_input_layers_info()[0]
output_layer = hef.get_output_layers_info()[0]
image_height, image_width, channels = input_layer.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_layer.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_layer.name].shape))
```

5.2.2. 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):
    vstreams_params = InputVStreamParams.make_from_network_group(configured_network)
    configured_network.wait_for_activation(1000)
    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, num_frames):
    vstreams_params = OutputVStreamParams.make_from_network_group(configured_network)
    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()
```

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

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

# The target used as a context manager
with PcieDevice() as target:
    configure_params = ConfigureParams.create_from_hef(hef, interface=HailoStreamInterface.PCIe)
    network_group = target.configure(hef, configure_params)[0]
    network_group_params = network_group.create_params()
    send_process = Process(target=send, args=(network_group, num_of_frames))
    recv_process = Process(target=recv, 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')
```

5.3. 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.

5.3.1. Single power measurement

```
[ ]: %matplotlib inline
import time

from hailo_platform.drivers.hailo_controller.power_measurement import DvmTypes, PowerMeasurementTypes, \
    SamplingPeriod, AveragingFactor # noqa F401
from hailo_platform import PcieDevice
```

Initialize the hardware object:

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

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))
```

5.3.2. Periodic power measurement

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

A measurement index is selected. It is a number between 0 and 3. The DVM is not given so the default DVM will be used, as explained above.

```
[ ]: index = 0
target.control.set_power_measurement(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(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()
```

6. Running Inference

The following section covers the several use cases of the inference process. Inference is the process in which the host sends the data to the Hailo-8 chip and receives the output from it.

6.1. Inference stages

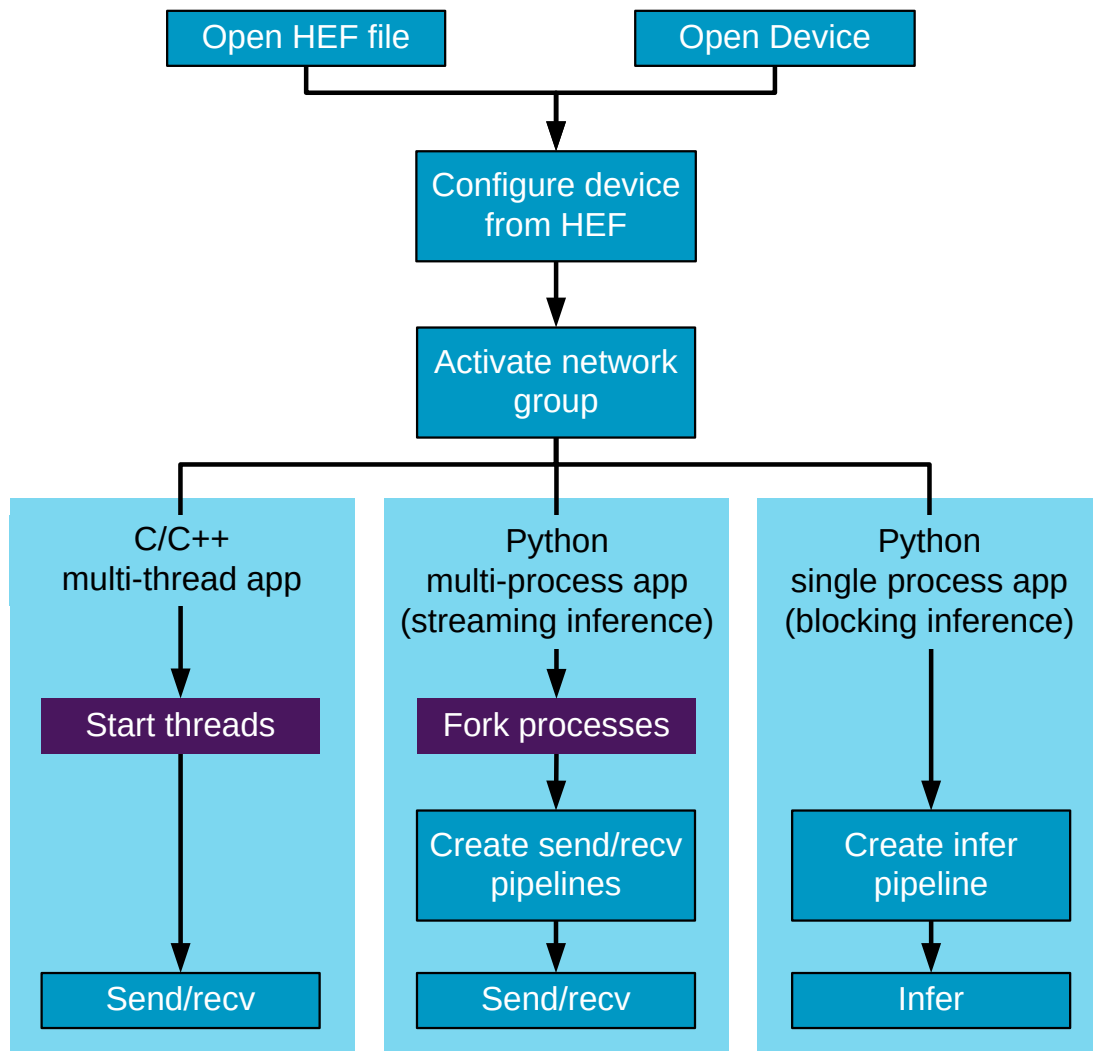


Figure 4: Illustration of an inference flow with the Hailo-8 device. The first stages are common to all application architectures. The last stages, with the light blue background, are architecture dependent. Three architectures are shown: C/C++ app, multi-process Python app, and single process Python app.

6.1.1. Preparation

In order to start working with the device, the user opens the device and the HEF file (or files). Then, the user 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. The following steps are taken in order to do it:

Pre-inference Goes over the data and prepares it before it is sent to the chip (For example quantizing the images).

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

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

Post-inference Makes sure the data is returned in the format expected by the host.

6.2. Python inference API

There are two different ways to work with the inference in Python:

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

Send/recv (streaming) In this case the user has two processes, each one in charge of 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](#) shows how to run blocking and streaming inference in Python.

6.3. C inference API

In C/C++ there is usually no need to open multiple processes. Multiple threads are used instead. One thread is in charge of pre-inference and sending the data. The other thread is in charge of receiving the data from the Hailo-8 chip and post-inference.

Two levels of API are provided in C:

- **Raw streams** – Allow low level interaction with the device. They 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.

6.4. Context switching and user controlled switching

HailoRT provides two mechanisms that allow to run big models and multiple models that don't fit together in a single Hailo-8 device.

The **context switch** mechanism allows to run a big model by automatically switching between several contexts that together constitute the full model.

The **user controlled switching** mechanism allows the user to switch between several models. 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.

The following table compares between these two methods:

	Context switch	User controlled switch
Purpose	<ul style="list-style-type: none"> Running a single big model. Running multiple models (<i>future support</i>). 	<ul style="list-style-type: none"> Running multiple models.
Platform requirements	<ul style="list-style-type: none"> Hailo-8 should be connected to the host via PCIe. 	<ul style="list-style-type: none"> Both PCIe and Ethernet are supported, but PCIe is recommended for faster switch.
How to compile the model?	<ul style="list-style-type: none"> Context switch should be enabled in the Dataflow Compiler. Enabled automatically for big models (<i>future support</i>). 	<ul style="list-style-type: none"> Any HEF can be used.
How to deploy the model?	<ul style="list-style-type: none"> No special HailoRT API. Switching is done automatically behind the scenes. 	<ul style="list-style-type: none"> Switching is done using the HailoRT API, by deactivating the first model and activating the second model.

See also:

The [C inference tutorial](#) shows how to use the user controlled switching in C.

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 `hailo` tools, run:

```
hailo --help
```

To list the available `hailortcli` tools, run:

```
hailortcli --help
```

The examples below are given for both families.

7.1. Firmware configuration tool

The `fw-config` tool allows reading and writing the firmware configuration. 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. This operation supports both PCIe and Ethernet interfaces.

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

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

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

The configuration consists of the 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_monitoring_orange_threshold_enable` – enables the orange alert of the over-current monitoring. If enabled, the current monitoring mechanism emits an alert whenever the orange threshold is reached. Defaults to true.
 - `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 protection component. When reaching above this threshold, all PCIe streams are closed. All values are given in degree Celsius.
 - `temperature_red_hysteresis_threshold` – sets the red hysteresis threshold of the temperature protection component. When reaching below this threshold, the device returns to orange zone. All values are given in degree Celsius.
 - `temperature_orange_threshold` – sets the orange threshold of the temperature protection component. When reaching above this threshold, frequency throttling begins, in case all other conditions are met. All values are given in degree Celsius.
 - `temperature_orange_hysteresis_threshold` – sets the orange hysteresis threshold of the temperature protection component. When reaching below this threshold, the device returns to green zone. All values are given in degree Celsius.
 - `temperature_throttling_enable` – enables the frequency throttling option. Note that temperature throttling is not available on the mPCIe module.
- **Control** – includes the `udp_port` field, which is used for changing the control port. Typically this is left at the default value.

7.1.1. 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,
    "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.

7.1.2. Reading the firmware configuration

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

```
hailo fw-config --target pcie read
```

```
hailortcli fw-config read --target pcie
```

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

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



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

7.1.3. Modifying the firmware configuration

The easiest method to change the configuration is to read it, modify it, and then write the updated JSON file. Use the following steps:

1. Read the configuration:

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

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

2. Modify config.json as needed.

3. Write the updated configuration:

```
hailo fw-config --target pcie write --input-file config.json
```

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

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

7.2. 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:

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

```
hailortcli fw-update --target udp --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.

7.3. Firmware control tool

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

7.3.1. Identify

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

```
hailo fw-control identify
```

```
hailortcli fw-control identify
```

7.3.2. 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.

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

```
hailortcli 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.

7.4. 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.

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.

```
hailo scan --target udp --interface-ip 10.0.0.100
```

```
hailortcli scan --target udp --interface-ip 10.0.0.100
```

7.5. Inference

Two tools are provided to run inference on the hardware: `hailo infer` and `hailortcli run`. They load a given HEF to the device and then send and receive data.

7.5.1. hailo infer

The `hailo infer` tool supports three modes:

- `simple` – Blocking inference that sends data to the device and waits for the results.
- `performance` (with full streaming mode) – Streaming inference with random data.
- `performance` (with hw-only streaming mode) – Similar to the full streaming mode, but skips pre-infer and post-infer steps on host.

This tool supports both PCIe and Ethernet interfaces.

```
hailo infer simple --config-path example.hef
```

7.5.2. hailortcli run

This tool also runs inference. While running, it performs several measurements such as FPS and power. It can also control the running mode:

- `streaming` – Similar to the full performance mode of `hailo infer`.
- `hw-only` – Similar to the hw-only performance mode of `hailo infer`.

This tool supports both PCIe and Ethernet interfaces.

```
hailortcli 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.

7.6. 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 done twice: firstly when taking only the Hailo device into account ("hardware only") and secondly when taking the full system into account including the host.

```
hailortcli benchmark 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.

7.7. Ethernet related command line tools

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

7.7.1. 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.

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

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

Reset

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

```
hailo udp-limiter reset --board-ip 1.2.3.4
```

```
hailortcli udp-rate-limiter reset --board-ip 1.2.3.4
```

Autoset

The autoset operation will determine the necessary UDP rate limitation based on a HEF and a desired FPS rate specified in `--fps`. The limitation takes into account the input/output throughput of the HEF.

```
hailo udp-limiter autoset --fps 200 --hef example.hef --board-ip 1.2.3.4
```

```
hailortcli udp-rate-limiter autoset --fps 200 --hef example.hef --board-ip 1.2.3.4
```

7.7.2. Ethernet related shell scripts

Three Ethernet interface related shell scripts are provided in the release. The scripts are located at `platform/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. Yocto

8.1. The meta-hailo layer

This section will guide through the integration of Hailo's Yocto layer into your own Yocto environment.

The instructions were written after being tested on the [DART-MX8M](#) embedded platform. The layer was built and validated with Yocto releases:

- Sumo (kernel 4.14.98)
- Warrior (kernel 4.19.35)
- Zeus (kernel 5.4.24)
- Dunfell (kernel 5.4.85)
- Gatesgarth (kernel 5.10.9)

These instructions assume that you have the following:

- An Embedded platform
- An Hailo8 chip
- A Yocto Environment to integrate to
- Hailo's platform release

8.2. Recipes

8.2.1. libhailort

Hailo's API for running inference on the hailo8 chip. The `libhailort.so` file is located under `platform/hailort`. The recipe copies the file to `/usr/lib` on the target device's root file system.

The recipe installs the `hailortcli` module on the `/usr/bin/` on the target's root file system. Make it usable from the global environment with `hailortcli` command

8.2.2. hailo-firmware

Hailo8's chip firmware (`hailo_fw.bin`). The firmware bin file is located in the release under `platform/firmware`. The recipe copies the file to the `/lib/firmware/hailo` directory on the target device's root file system.

8.2.3. hailo-pci

Compiles Hailo8's PCIe communications driver and installs it. The source files are located under `platform/drivers/pcie`. The output of the compilation is `hailo_pci.ko` under `platform/drivers/pcie/build/release/${TARGET_GOARCH}`. This kernel object file will be copied to `lib/modules/${KERNEL_VERSION}/usr/lib` on the target device's root file system.

8.2.4. pyhailort

Hailo's python API. The recipe copies hailo configuration file, unpacks the `/platform/python/hailo_platform.whl` into `python/site-packages` on the target device's root file system. The recipe creates a link to hailo cli under `/usr/bin` make it usable from the global environment with `hailo` command.

The recipe depends on all the python recipes under `meta-hailo/recipes-python/` and it's currently supported by python 3.6, 3.7 and 3.8.

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
```

8.2.5. hailo-config

Create the Hailo configuration directory under the user's home directory, and copy hailo's configuration to it

8.2.6. hailo-board-config

This recipe is not needed for Hailo's M.2 and mPCIe modules. In cases where the board used doesn't include either EEPROM or Flash, Hailo's board configuration tool should be used (refer to the board-configuration user guide for further details). This recipe copies Hailo's board-configuration tool binary output `platform/firmware/hailo8_board_cfg.bin` (board-configuration tool default path) to the target device root file system.

8.3. Integrating to an existing Yocto environment

8.3.1. Extraction

If you install HailoRT, either directly or via the SDK, the installation will extract the HailoRT with the Yocto files to `platform/` directory. Another option is to manually untar `platform.tar.gz` without installing HailoRT locally. In this case the Yocto files are located under `platform/` too. The following tree is the expected directory structure:

```
firmware/
  hailo8_fw.X.X.X.bin
hailort/
  doc/
  examples/
  include/
drivers/
  pcie/
    51-hailo-udev.rules
    include/
    internal_inc/
    Makefile
    src/
  common/
python/
  hailo_platform-X.X.X-cp36-cp36m-linux_aarch64.whl
  hailo_platform-X.X.X-cp36-cp36m-linux_x86_64.whl
  hailo_platform-X.X.X-cp37-cp37m-linux_aarch64.whl
  hailo_platform-X.X.X-cp37-cp37m-linux_x86_64.whl
  hailo_platform-X.X.X-cp38-cp38m-linux_aarch64.whl
  hailo_platform-X.X.X-cp38-cp38m-linux_x86_64.whl
hailortcli/
```

(continues on next page)

(continued from previous page)

```

aarch64/
    hailortcli
armv7lhf/
    hailortcli
x86_64/
    hailortcli
doc/
tutorials/
scripts/
    configure_ethernet_buffers.sh
    configure_interface.sh
    ubuntu_ethernet_statistics.sh
yocto/
    sumo/
        (similar structure as warrior)
    warrior/
        meta-hailo/
            conf/
                layer.conf
            licenses/
                LICENSE/
                LICENSE
            recipes-hailo/
                hailo-firmware/
                    hailo-firmware_X.X.X.bb
                libhailort/
                    libhailort_X.X.X.bb
                hailo-config/
                    hailo-config_X.X.X.bb
                hailo-board-config/ (Only for custom boards without flash/EEPROM)
                    hailo-board-config_X.X.X.bb
            recipes-kernel/
                hailo-pci/
                    hailo-pci_X.X.X.bb
            recipes-python/
                python3-argcomplete/
                python3-aspy.yaml/
                python3-cfgv/
                python3-contextlib2/
                python3-cppheaderparser/
                python3-distlib/
                python3-future/
                python3-identify/
                python3-netifaces/
                python3-pyzmq/
                python3-textutils/
                python3-tqdm/
                python3-verboselogs/
                python3-zipp/
                python3-zmq/
        zeus/
            (similar structure as warrior)
        dunfell/
            (similar structure as warrior)
        gatesgarth/
            (similar structure as warrior)

```

8.3.2. Setup

1. The Yocto directory of all supported Yocto releases include the meta-hailo layer. Get into the directory of your desired 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 "
```

2. The layer uses the unpacked release directory as an external source. In order for the build to work you'll have to set a HAILO_EXTERNALSRC variable in your conf/local.conf file to point to the root directory of the Hailo's platform release you have extracted:

```
HAILO_EXTERNALSRC = "<insert full path here>"
```

3. Add the recipes to your image in your conf/local.conf:

```
IMAGE_INSTALL_append = " hailo-pci libhailort hailo-firmware pyhailort"
```

An example to how your files should look like:

```
/local
  yocto-dunfell/
    build_xwayland/
      conf/
        local.conf
        bblayers.conf
      cache/
      tmp/
      workspace/
    setup-environment
    README
    sources/
      poky/
      meta-hailo/
      meta-openembedded/
    platform_release/
```

In this example local.conf will contain - HAILO_EXTERNALSRC = "/local/platform_release"

8.4. Build your image

Run bitbake and build your image. If your HAILO_EXTERNALSRC variable points to a wrong path you may receive a similar error:

```
ERROR: EXTERNALSRC must be an absolute path
```

or

```
ERROR: hailo-pci-X.X.0-r0 do_configure: [Errno 2] No such file or directory: '<build directory of the_
recipe>' -> '<given invalid path>'
ERROR: hailo-pci-X.X.0-r0 do_configure: Function failed: externalsrc_configure_prefunc
```

After the build successfully finished, burn the Image to your embedded device.

8.5. Validating the integration's success

Make sure that the following conditions have been met on the target device:

Recipe	Details
hailo-firmware	Hailo's firmware exists at: <code>/lib/firmware/hailo/hailo_fw.bin</code>
libhailort	Libhailort exists at: <code>/usr/lib/libhailort.so</code> hailortcli command works from the global environment. After connecting the hailo8 chip – scan and infer commands should communicate with the board.
hailo-pci	Running <code>modinfo hailo-pci</code> should return a successful result and info about the driver.
pyhailort	Python3 includes the <code>hailo_platform</code> module and all dependent modules. hailo command works from the global environment. After connecting the hailo8 chip – <code>fw-control identify</code> recognizes it and prints information.
hailo-board-configuration	Only for custom boards without flash/EEPROM. Hailo's board configurations exists at <code>/lib/firmware/hailo/hailo8_board_cfg.bin</code> After connecting the Hailo-8 chip, <code>hailortcli fw-control identify</code> prints the right configurations.

Part II

API Reference

9. HailoRT C/C++ API Reference

HailoRT is Hailo's C runtime library. It is used to run inference over compiled models (HEFs) in a C/C++ program.

9.1. Device and control API functions

HAILORTAPI *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.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [out] pcie_device_infos: A pointer to a buffer of *hailo_pcie_device_info_t* that receives the information.
- [in] pcie_device_infos_length: The number of *hailo_pcie_device_info_t* elements in the buffer pointed to by *pcie_device_infos*.
- [out] number_of_devices: 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.

HAILORTAPI *hailo_status* **hailo_parse_pcie_device_info**(const char *device_info_str, ...)

Parse PCIe device BDF string into hailo device info structure.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Note Call *hailo_scan_pcie_devices* to get all available hailo pcie devices.

Parameters

- [in] device_info_str: BDF device info, format [<domain>].<bus>.<device>.<func>, same format as in lspci.
- [out] device_info: A pointer to a *hailo_pcie_device_info_t* that receives the parsed device info.

HAILORTAPI *hailo_status* **hailo_create_pcie_device**(*hailo_pcie_device_info_t* *device_info, *hailo_device* ...)

Creates a PCIe device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Note To release a device, call the *hailo_release_device* function with the returned *hailo_device*.

Parameters

- [in] device_info: Information about the device to open. If NULL is given and there is only one available PCIe device, use this device.
- [out] device: A pointer to a *hailo_device* that receives the allocated PCIe device.

HAILORTAPI *hailo_status* **hailo_scan_ethernet_devices**(const char *interface_name, ...)

Returns information on all available ethernet devices in the system.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] interface_name: The name of the network interface to scan.
- [out] eth_device_infos: A pointer to a buffer of *hailo_eth_device_info_t* that receives the information.
- [in] eth_device_infos_length: The number of *hailo_eth_device_info_t* elements in the buffer pointed to by *eth_device_infos*.

- [out] number_of_devices: 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.
- [in] timeout_ms: The time in milliseconds to scan devices.

HAILORTAPI *hailo_status* **hailo_create_ethernet_device**(*hailo_eth_device_info_t* *device_info, ...)
Creates an ethernet device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Note To release a device, call the *hailo_release_device* function with the returned *hailo_device*.

Parameters

- [in] device_info: Information about the device to open.
- [out] device: A pointer to a *hailo_device* that receives the allocated ethernet device corresponding to the given information.

HAILORTAPI *hailo_status* **hailo_release_device**(*hailo_device* device)
Release an open device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object to be released.

HAILORTAPI *hailo_status* **hailo_identify**(*hailo_device* device, *hailo_device_identity_t* *device_identity)
Sends identify control to a Hailo device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* to be identified.
- [out] device_identity: Information about the device.

HAILORTAPI *hailo_status* **hailo_get_extended_device_information**(*hailo_device* device, ...)
Get extended device information from a Hailo device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* to get extended device info from.
- [out] extended_device_information: Extended information about the device.

HAILORTAPI *hailo_status* **hailo_set_fw_logger**(*hailo_device* device, *hailo_fw_logger_level_t* level, uint32_t ...)
Configure fw logger level and interface of sending.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] level: The minimum logger level.
- [in] interface_mask: Output interfaces (mix of *hailo_fw_logger_interface_t*).

HAILORTAPI *hailo_status* **hailo_set_throttling_state**(*hailo_device* device, bool should_activate)
Change throttling state of temperature protection component.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] should_activate: Should be true to enable or false to disable.

HAILORTAPI *hailo_status* **hailo_get_throttling_state**(*hailo_device* device, bool *is_active)

Get current throttling state of temperature protection component.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [out] is_active: A pointer to the temperature protection throttling state: true if active, false otherwise.

HAILORTAPI *hailo_status* **hailo_reset_device**(*hailo_device* device, *hailo_reset_device_mode_t* mode)

Reset device

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] mode: The mode of the reset.

HAILORTAPI *hailo_status* **hailo_update_firmware**(*hailo_device* device, void *firmware_buffer, uint32_t ...)

Updates firmware to device flash.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

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

- [in] device: A *hailo_output_stream* object.
- [in] firmware_buffer: A pointer to a buffer that contains the firmware to be updated on the device.
- [in] firmware_buffer_size: The size in bytes of the buffer pointed by *firmware_buffer*.

HAILORTAPI *hailo_status* **hailo_update_second_stage**(*hailo_device* device, void *second_stage_buffer, ...)

Updates second stage to device flash.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

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

- [in] device: A *hailo_output_stream* object.
- [in] second_stage_buffer: A pointer to a buffer that contains the second_stage to be updated on the device.
- [in] second_stage_buffer_size: The size in bytes of the buffer pointed by *second_stage_buffer*.

HAILORTAPI *hailo_status* **hailo_set_notification_callback**(*hailo_device* device, ...)

Sets a callback to be called when a notification with ID *notification_id* will be received

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A [hailo_device](#) to register the callback to.
- [in] callback: The callback function to be called.
- [in] notification_id: The ID of the notification.
- [in] opaque: User specific data.

HAILORTAPI [hailo_status](#) **hailo_remove_notification_callback**([hailo_device](#) device, ...)
Removes a previously set callback with ID *notification_id*

Return Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Parameters

- [in] device: A [hailo_device](#) to register the callback to.
- [in] notification_id: The ID of the notification to remove.

HAILORTAPI [hailo_status](#) **hailo_reset_sensor**([hailo_device](#) device, uint8_t section_index)
Reset the sensor that is related to the section index config.

Return Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Parameters

- [in] device: A [hailo_device](#) object.
- [in] section_index: Flash section index to load config from. [0-6]

HAILORTAPI [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.

Return Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Parameters

- [in] device: A [hailo_device](#) object.
- [in] sensor_type: The sensor type.
- [in] bus_index: The I2C bus index of the sensor.

HAILORTAPI [hailo_status](#) **hailo_load_and_start_sensor**([hailo_device](#) device, uint8_t section_index)
Load the configuration with I2C in the section index.

Return Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns a [hailo_status](#) error.

Parameters

- [in] device: A [hailo_device](#) object.
- [in] section_index: Flash section index to load config from. [0-6]

HAILORTAPI [hailo_status](#) **hailo_i2c_read**([hailo_device](#) device, const [hailo_i2c_slave_config_t](#) *slave_config, ...)
Read data from an I2C slave over a hailo device.

Return Upon success, returns [HAILO_SUCCESS](#). Otherwise, returns an [hailo_status](#) error.

Parameters

- [in] device: A [hailo_device](#) object.
- [in] slave_config: The [hailo_i2c_slave_config_t](#) configuration of the slave.
- [in] register_address: The address of the register from which the data will be read.

- [in] data: Pointer to a buffer that would store the read data.
- [in] length: The number of bytes to read into the buffer pointed by *data*.

HAILORTAPI *hailo_status* **hailo_i2c_write**(*hailo_device* device, **const** *hailo_i2c_slave_config_t* *slave_config, ...)
Write data to an I2C slave over a hailo device.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] slave_config: The *hailo_i2c_slave_config_t* configuration of the slave.
- [in] register_address: The address of the register to which the data will be written.
- [in] data: A pointer to a buffer that contains the data to be written to the slave.
- [in] length: The size of *data* in bytes.

HAILORTAPI *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.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] section_index: Flash section index to load config from. [0-7]
- [in] config_file_path: File path to dump section configuration into.

HAILORTAPI *hailo_status* **hailo_store_sensor_config**(*hailo_device* device, uint32_t section_index, ...)
Store sensor configuration to Hailo chip flash memory.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] section_index: Flash section index to write to.
- [in] sensor_type: Sensor type.
- [in] reset_config_size: Size of reset configuration.
- [in] config_height: Configuration resolution height.
- [in] config_width: Configuration resolution width.
- [in] config_fps: Configuration FPS.
- [in] config_file_path: Sensor configuration file path.
- [in] config_name: Sensor configuration name.

HAILORTAPI *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.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] reset_config_size: Size of reset configuration.
- [in] config_height: Configuration resolution height.

- [in] config_width: Configuration resolution width.
- [in] config_fps: Configuration FPS.
- [in] isp_static_config_file_path: ISP static configuration file path.
- [in] isp_runtime_config_file_path: ISP runtime configuration file path.
- [in] config_name: Sensor configuration name.

HAILORTAPI *hailo_status* **hailo_test_chip_memories**(*hailo_device* device)

Test chip memories using smart BIST mechanism.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] device: - A *hailo_device* object.

9.2. HEF API functions

HAILORTAPI *hailo_status* **hailo_create_hef_file**(*hailo_hef* *hef, const char *file_name)

Creates a HEF from file.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Note To release a HEF, call the *hailo_release_hef* function with the returned *hef*.

Parameters

- [out] hef: A pointer to a *hailo_hef* that receives the allocated HEF.
- [in] file_name: The name of the hef file.

HAILORTAPI *hailo_status* **hailo_create_hef_buffer**(*hailo_hef* *hef, const void *buffer, size_t size)

Creates a HEF from buffer.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Note To release a HEF, call the *hailo_release_hef* function with the returned *hef*.

Parameters

- [out] hef: A pointer to a *hailo_hef* that receives the allocated HEF.
- [in] buffer: A pointer to a buffer that contains the HEF content.
- [in] size: The size in bytes of the HEF content pointed by *buffer*.

HAILORTAPI *hailo_status* **hailo_release_hef**(*hailo_hef* hef)

Release an open HEF.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] hef: A *hailo_hef* object to be released.

HAILORTAPI *hailo_status* **hailo_hef_get_all_stream_infos**(*hailo_hef* hef, const char ...)

Gets all stream infos.

Return 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.

Parameters

- [in] hef: A *hailo_hef* object that contains the information.
- [in] network_group_name: 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.
- [out] stream_infos: A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- [in] stream_infos_length: The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- [out] number_of_streams: This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

HAILORTAPI *hailo_status* **hailo_hef_get_stream_info_by_name**(*hailo_hef* hef, const char ...)
Gets stream info by name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] hef: A *hailo_hef* object that contains the information.
- [in] network_group_name: 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.
- [in] stream_name: The name of the stream as presented in the hef.
- [in] stream_direction: Indicates the stream direction.
- [out] stream_info: A pointer to a *hailo_stream_info_t* that receives the stream information.

HAILORTAPI *hailo_status* **hailo_hef_get_all_vstream_infos**(*hailo_hef* hef, const char ...)
Gets all virtual stream infos.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] hef: A *hailo_hef* object that contains the information.
- [in] network_group_name: The name of the network_group which contains the vstream infos. If NULL is passed, the first network_group in the HEF will be addressed.
- [out] vstream_infos: A pointer to a buffer of *hailo_stream_info_t* that receives the informations.
- [inout] vstream_infos_count: 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*.

HAILORTAPI *hailo_status* **hailo_hef_get_stream_name_from_original_name**(*hailo_hef* hef, const char ...)
Gets stream name from original layer name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] hef: A *hailo_hef* object that contains the information.
- [in] network_group_name: 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.
- [in] original_name: The name of the original layer as presented in the hef.
- [out] stream_name: The name of the stream for the provided original name, ends with NULL terminator.

HAILORTAPI *hailo_status* **hailo_hef_get_original_names_from_stream_name**(*hailo_hef* hef, const char ...)
Gets all original layer names from stream name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *hef*: A *hailo_hef* object that contains the information.
- [in] *network_group_name*: 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.
- [in] *stream_name*: The name of the stream as presented in the *hef*.
- [out] *original_names*: Array of *hailo_layer_name_t*, all original names linked to the provided stream (each name ends with NULL terminator).
- [inout] *original_names_length*: As input - the size of *original_names* array. As output - the number of *original_layers* names.

HAILORTAPI *hailo_status* **hailo_hef_get_sorted_output_names**(*hailo_hef* *hef*, const char ...)

Gets sorted output names from *network_group* name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *hef*: A *hailo_hef* object that contains the information.
- [in] *network_group_name*: The name of the *network_group*. If NULL is passed, the first *network_group* in the HEF will be addressed.
- [out] *sorted_output_names*: List of sorted outputs names.
- [inout] *sorted_output_names_count*: As input - the expected size of *sorted_output_names* list. As output - the number of *sorted_output_names*.

HAILORTAPI *hailo_status* **hailo_hef_get_bottleneck_fps**(*hailo_hef* *hef*, const char ...)

Gets bottleneck fps from *network_group* name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *hef*: A *hailo_hef* object that contains the information.
- [in] *network_group_name*: The name of the *network_group*. If NULL is passed, the first *network_group* in the HEF will be addressed.
- [out] *bottleneck_fps*: Bottleneck FPS. Note: This is not relevant in the case of multi context.

HAILORTAPI *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*.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

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

- [in] *hef*: A *hailo_hef* object that contains the stream's information.

- [in] `network_group_name`: 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.
- [in] `fps`: The desired fps.
- [out] `rates`: A pointer to an array of `hailo_rate_limit_t` that receives the rates.
- [inout] `rates_length`: As input - the length of `rates` array. As output - the number of H2D streams.

9.3. Network group API functions

HAILORTAPI `hailo_status hailo_init_configure_params(hailo_hef hef, hailo_stream_interface_t interface, ...)`

Init configure params with default values for a given hef.

Return Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Parameters

- [in] `hef`: A `hailo_hef` object to configure the *device* by.
- [in] `interface`: A `hailo_stream_interface_t` indicating which `hailo_stream_parameters_t` to create.
- [out] `params`: A `hailo_configure_params_t` to be filled.

HAILORTAPI `hailo_status hailo_init_configure_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.

Return Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Parameters

- [in] `hef`: A `hailo_hef` object to configure the *device* by.
- [in] `output_interface`: A `hailo_stream_interface_t` indicating which `hailo_stream_parameters_t` to create for the output streams.
- [in] `mipi_params`: A `hailo_mipi_input_stream_params_t` object which contains the MIPI params for the input streams.
- [out] `params`: A `hailo_configure_params_t` to be filled.

HAILORTAPI `hailo_status hailo_configure_device(hailo_device device, hailo_hef hef, ...)`

Configure the device from a hef.

Return Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Parameters

- [in] `device`: A `hailo_device` object to be configured.
- [in] `hef`: A `hailo_hef` object to configure the *device* by.
- [in] `params`: A `hailo_configure_params_t` (may be NULL). Can be initialized to default values using `hailo_init_configure_params`.
- [out] `network_groups`: Array of `network_groups` that were loaded from the HEF file.
- [inout] `number_of_network_groups`: As input - the size of `network_groups` array. As output - the number of `network_groups` loaded.

HAILORTAPI `hailo_status hailo_wait_for_network_group_activation(hailo_configured_network_group ...)`

Block until `network_group` is activated, or until `timeout_ms` is passed.

Return Upon success, returns `HAILO_SUCCESS`. Otherwise, returns a `hailo_status` error.

Parameters

- [in] network_group: A [hailo_configured_network_group](#) to wait for activation.
- [in] timeout_ms: The timeout in milliseconds. If *timeout_ms* is zero, the function will return immediately. If *timeout_ms* is *HAILO_INFINITE*, the function will return only when the event is signaled.

HAILORTAPI [hailo_status](#) **hailo_get_network_groups_infos**([hailo_hef](#) hef, [hailo_network_group_info_t](#) ...)
Get network group infos from a hef.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a [hailo_status](#) error.

Parameters

- [in] hef: A [hailo_hef](#) object.
- [out] infos: Array of [hailo_network_group_info_t](#) to be filled.
- [inout] number_of_infos: As input - the size of infos array. As output - the number of infos loaded.

HAILORTAPI [hailo_status](#) **hailo_network_group_get_all_stream_infos**([hailo_configured_network_group](#) ...)
Gets all stream infos from a configured network group

Return 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.

Parameters

- [in] network_group: A [hailo_configured_network_group](#) object.
- [out] stream_infos: A pointer to a buffer of [hailo_stream_info_t](#) that receives the informations.
- [in] stream_infos_length: The number of [hailo_stream_info_t](#) elements in the buffer pointed by *stream_infos*
- [out] number_of_streams: This variable will be filled with the number of stream_infos if the function returns with *HAILO_SUCCESS* or *HAILO_INSUFFICIENT_BUFFER*.

HAILORTAPI [hailo_status](#) **hailo_network_group_get_input_stream_infos**([hailo_configured_network_group](#) ...)
Gets all input stream infos from a configured network group

Return 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.

Parameters

- [in] network_group: A [hailo_configured_network_group](#) object.
- [out] stream_infos: A pointer to a buffer of [hailo_stream_info_t](#) that receives the informations.
- [in] stream_infos_length: The number of [hailo_stream_info_t](#) elements in the buffer pointed by *stream_infos*
- [out] number_of_streams: This variable will be filled with the number of stream_infos if the function returns with *HAILO_SUCCESS* or *HAILO_INSUFFICIENT_BUFFER*.

HAILORTAPI [hailo_status](#) **hailo_network_group_get_output_stream_infos**([hailo_configured_network_group](#) ...)
Gets all output stream infos from a configured network group

Return 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.

Parameters

- [in] network_group: A [hailo_configured_network_group](#) object.
- [out] stream_infos: A pointer to a buffer of [hailo_stream_info_t](#) that receives the informations.

- [in] stream_infos_length: The number of *hailo_stream_info_t* elements in the buffer pointed by *stream_infos*
- [out] number_of_streams: This variable will be filled with the number of stream_infos if the function returns with HAILO_SUCCESS or HAILO_INSUFFICIENT_BUFFER.

HAILORTAPI *hailo_status* **hailo_activate_network_group**(*hailo_configured_network_group* ...)

Activates hailo_device inner-resources for context_switch inference.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] network_group: NetworkGroup to be activated.
- [in] activation_params: Optional parameters for the activation (may be NULL).
- [out] activated_network_group_out: Handle for the activated network_group.

HAILORTAPI *hailo_status* **hailo_deactivate_network_group**(*hailo_activated_network_group* ...)

De-activates hailo_device inner-resources for context_switch inference.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] activated_network_group: NetworkGroup to deactivate.

HAILORTAPI *hailo_status* **hailo_get_input_stream**(*hailo_configured_network_group* ...)

Return input stream from configured network_group by stream name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] configured_network_group: NetworkGroup to get stream from.
- [in] stream_name: The name of the input stream to retrieve.
- [out] stream: The returned input stream.

HAILORTAPI *hailo_status* **hailo_get_output_stream**(*hailo_configured_network_group* ...)

Return output stream from configured network_group by stream name.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] configured_network_group: NetworkGroup to get stream from.
- [in] stream_name: The name of the output stream to retrieve.
- [out] stream: The returned output stream.

HAILORTAPI *hailo_status* **hailo_get_latency_measurement_from_network_group**(*hailo_configured_network_group* ...)

Returns the network latency (only available if latency measurement was enabled).

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] configured_network_group: NetworkGroup to get the latency measurement from.
- [out] result: Output latency result.

9.4. Virtual stream API functions

HAILORTAPI *hailo_status* **hailo_make_input_vstream_params**(*hailo_configured_network_group* ...)

Creates input virtual stream params for a given network_group.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] network_group: Network group that owns the streams.
- [in] quantized: 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.
- [in] format_type: The default format type for all input virtual streams.
- [out] input_params: List of params for input virtual streams.
- [inout] input_params_count: On input: Amount of *input_params* array. On output: Will be filled with the detected amount of input vstreams on the *network_group*.

HAILORTAPI *hailo_status* **hailo_make_output_vstream_params**(*hailo_configured_network_group* ...)

Creates output virtual stream params for given network_group.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] network_group: Network group that owns the streams.
- [in] quantized: 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.
- [in] format_type: The default format type for all output virtual streams.
- [out] output_params: List of params for output virtual streams.
- [inout] output_params_count: On input: Amount of *output_params* array. On output: Will be filled with the detected amount of output vstreams on the *network_group*.

HAILORTAPI *hailo_status* **hailo_create_input_vstreams**(*hailo_configured_network_group* ...)

Creates input virtual streams.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Note To release input virtual streams, call the *hailo_release_input_vstreams* function with the returned *input_vstreams* and *inputs_count*.

Parameters

- [in] configured_network_group: Network group that owns the streams.
- [in] inputs_params: List of input virtual stream params to create input virtual streams from.
- [in] inputs_count: How many members in *input_params*.
- [out] input_vstreams: List of input virtual streams.

HAILORTAPI *hailo_status* **hailo_create_output_vstreams**(*hailo_configured_network_group* ...)

Creates output virtual streams.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Note To release output virtual streams, call the *hailo_release_output_vstreams* function with the returned *output_vstreams* and *outputs_count*.

Parameters

- [in] `configured_network_group`: Network group that owns the streams.
- [in] `outputs_params`: List of output virtual stream params to create output virtual streams from.
- [in] `outputs_count`: How many members in `outputs_params`.
- [out] `output_vstreams`: List of output virtual streams.

HAILORTAPI *hailo_status* **hailo_get_input_vstream_frame_size**(*hailo_input_vstream* input_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)

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `input_vstream`: A *hailo_input_vstream* object.
- [out] `frame_size`: The size of the frame on the host side in bytes.

HAILORTAPI *hailo_status* **hailo_get_input_vstream_info**(*hailo_input_vstream* input_vstream, ...)

Gets the *hailo_vstream_info_t* struct for the given vstream.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `input_vstream`: A *hailo_input_vstream* object.
- [out] `vstream_info`: Will be filled with *hailo_vstream_info_t*.

HAILORTAPI *hailo_status* **hailo_get_input_vstream_user_format**(*hailo_input_vstream* input_vstream, ...)

Gets the user buffer format struct for the given vstream.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `input_vstream`: A *hailo_input_vstream* object.
- [out] `user_buffer_format`: Will be filled with *hailo_format_t*.

HAILORTAPI *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)

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `output_vstream`: A *hailo_output_vstream* object.
- [out] `frame_size`: The size of the frame on the host side in bytes.

HAILORTAPI *hailo_status* **hailo_get_output_vstream_info**(*hailo_output_vstream* output_vstream, ...)

Gets the *hailo_vstream_info_t* struct for the given vstream.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `output_vstream`: A *hailo_output_vstream* object.
- [out] `vstream_info`: Will be filled with *hailo_vstream_info_t*.

HAILORTAPI *hailo_status* **hailo_get_output_vstream_user_format**(*hailo_output_vstream* ...)
Gets the user buffer format struct for the given vstream.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] output_vstream: A *hailo_output_vstream* object.
- [out] user_buffer_format: Will be filled with *hailo_format_t*.

HAILORTAPI *hailo_status* **hailo_vstream_write_raw_buffer**(*hailo_input_vstream* input_vstream, const ...)
Writes buffer to hailo device via input virtual stream *input_vstream*.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] input_vstream: A *hailo_input_vstream* object.
- [in] buffer: 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*).
- [in] buffer_size: *buffer* buffer size in bytes. The size is expected to be the size returned from *hailo_get_input_vstream_frame_size*.

HAILORTAPI *hailo_status* **hailo_vstream_read_raw_buffer**(*hailo_output_vstream* output_vstream, void ...)
Reads data from hailo device via *output_vstream* into *dst*.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] output_vstream: A *hailo_output_vstream* object.
- [in] buffer: 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*).
- [in] buffer_size: *dst* buffer size in bytes. The size is expected to be the size returned from *hailo_get_output_vstream_frame_size*.

HAILORTAPI *hailo_status* **hailo_release_input_vstreams**(const *hailo_input_vstream* *input_vstreams, ...)
Release input virtual streams.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] input_vstreams: List of input virtual streams to be released.
- [in] inputs_count: How many members in *input_params*.

HAILORTAPI *hailo_status* **hailo_release_output_vstreams**(const *hailo_output_vstream* ...)
Release output virtual streams.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] output_vstreams: List of output virtual streams to be released.
- [in] outputs_count: How many members in *output_vstreams*.

9.5. Stream API functions

HAILORTAPI *hailo_status* **hailo_set_input_stream_timeout**(*hailo_input_stream* stream, uint32_t timeout_ms)
Set new timeout value to an input stream

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] stream: A *hailo_input_stream* object to get the new timeout value.
- [in] timeout_ms: the new timeout value to be set.

HAILORTAPI *hailo_status* **hailo_set_output_stream_timeout**(*hailo_output_stream* stream, uint32_t ...)
Set new timeout value to an output stream

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] stream: A *hailo_output_stream* object to get the new timeout value.
- [in] timeout_ms: the new timeout value to be set.

HAILORTAPI 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

Return The size of the frame on the host side in bytes

Parameters

- [in] stream: A *hailo_input_stream* object.

HAILORTAPI 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

Return The size of the frame on the host side in bytes

Parameters

- [in] stream: A *hailo_output_stream* object.

HAILORTAPI *hailo_status* **hailo_get_input_stream_info**(*hailo_input_stream* stream, *hailo_stream_info_t* ...)
Gets stream info from the given input stream

Return The size of the frame on the host side in bytes

Parameters

- [in] stream: A *hailo_input_stream* object.
- [out] stream_info: An output *hailo_stream_info_t*.

HAILORTAPI *hailo_status* **hailo_get_output_stream_info**(*hailo_output_stream* stream, ...)
Gets stream info from the given output stream

Return The size of the frame on the host side in bytes

Parameters

- [in] stream: A *hailo_input_stream* object.
- [out] stream_info: An output *hailo_stream_info_t*.

HAILORTAPI *hailo_status* **hailo_stream_read_raw_buffer**(*hailo_output_stream* stream, void *buffer, size_t ...)
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 a product of stream_info.hw_frame_size (i.e. more than one frame may be read)

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] stream: A *hailo_output_stream* object.
- [in] buffer: A pointer to a buffer that receives the data read from *stream*.
- [in] size: The amount of bytes to read, should be the frame size.

HAILORTAPI *hailo_status* **hailo_stream_write_raw_buffer**(*hailo_input_stream* stream, const void ...)
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 a product of stream_info.hw_frame_size (i.e. more than one frame may be read)

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] stream: A *hailo_input_stream* object.
- [in] buffer: A pointer to a buffer that contains the data to be written to *stream*.
- [in] size: The amount of bytes to write.

HAILORTAPI 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)

Return The size of the frame on the host side in bytes

Parameters

- [in] stream_info: The stream's info represented by *hailo_stream_info_t*
- [in] transform_params: Host side transform parameters

9.6. Transformation API functions

HAILORTAPI *hailo_status* **hailo_create_input_transformer**(const *hailo_stream_info_t* *stream_info, ...)
Creates an input transformer object. Allocates all necessary buffers used for the transformation (pre-process).

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Note To release the transformer, call the *hailo_release_input_transformer* function with the returned *hailo_input_transformer*.

Parameters

- [in] stream_info: - A *hailo_stream_info_t* object
- [in] transform_params: - A *hailo_transform_params_t* user transformation parameters.
- [out] transformer: - A *hailo_input_transformer*

HAILORTAPI *hailo_status* **hailo_release_input_transformer**(*hailo_input_transformer* transformer)
Releases a transformer object including all allocated buffers.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] transformer: - A *hailo_input_transformer* object.

HAILORTAPI *hailo_status* **hailo_transform_frame_by_input_transformer**(*hailo_input_transformer* ...) *hailo_status*
Transforms an input frame pointed to by *src* directly to the buffer pointed to by *dst*.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Warning The buffers must not overlap.

Parameters

- [in] transformer: A *hailo_input_transformer*.
- [in] src: A pointer to a buffer to be transformed.
- [in] src_size: 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.
- [out] dst: A pointer to a buffer that receives the transformed data.
- [in] dst_size: 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.

HAILORTAPI *hailo_status* **hailo_create_output_transformer**(const *hailo_stream_info_t* *stream_info, ...) *hailo_status*
Creates an output transformer object. Allocates all necessary buffers used for the transformation (post-process).

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Note To release the transformer, call the *hailo_release_output_transformer* function with the returned *hailo_output_transformer*.

Parameters

- [in] stream_info: - A *hailo_stream_info_t* object
- [in] transform_params: - A *hailo_transform_params_t* user transformation parameters.
- [out] transformer: - A *hailo_output_transformer*

HAILORTAPI *hailo_status* **hailo_release_output_transformer**(*hailo_output_transformer* transformer)
Releases a transformer object including all allocated buffers.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] transformer: - A *hailo_output_transformer* object.

HAILORTAPI *hailo_status* **hailo_transform_frame_by_output_transformer**(*hailo_output_transformer* ...) *hailo_status*
Transforms an output frame pointed to by *src* directly to the buffer pointed to by *dst*.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Warning The buffers must not overlap.

Parameters

- [in] transformer: A *hailo_output_transformer*.
- [in] src: A pointer to a buffer to be transformed.

- [in] `src_size`: 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.
- [out] `dst`: A pointer to a buffer that receives the transformed data.
- [in] `dst_size`: 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.

HAILORTAPI *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.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Note To release the demuxer, call the *hailo_release_output_demuxer* function with the returned *hailo_output_demuxer*.

Parameters

- [in] `stream`: - A *hailo_output_stream* object
- [in] `demux_params`: - A *hailo_demux_params_t* user demux parameters.
- [out] `demuxer`: - A *hailo_output_demuxer*, used to transform output frames

HAILORTAPI *hailo_status* **hailo_release_output_demuxer**(*hailo_output_demuxer* demuxer)

Releases a demuxer object.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns an *hailo_status* error.

Parameters

- [in] `demuxer`: - A *hailo_output_demuxer* object.

HAILORTAPI *hailo_status* **hailo_demux_raw_frame_by_output_demuxer**(*hailo_output_demuxer* demuxer, ...)

Demultiplexing an output frame pointed to by `src` directly to the buffer pointed to by `dst`.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `demuxer`: A *hailo_output_demuxer* object used for the demuxing.
- [in] `src`: A pointer to a buffer to be demultiplexed.
- [in] `src_size`: 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.
- [inout] `raw_buffers`: A pointer to an array of *hailo_stream_raw_buffer_t* that receives the demultiplexed data read from the `stream`.
- [in] `raw_buffers_count`: The number of *hailo_stream_raw_buffer_t* elements in the array pointed to by `raw_buffers`.

HAILORTAPI *hailo_status* **hailo_get_mux_infos_by_output_demuxer**(*hailo_output_demuxer* demuxer, ...)

Gets all multiplexed stream infos.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] `demuxer`: A *hailo_output_demuxer* object.
- [out] `stream_infos`: A pointer to a buffer of *hailo_stream_info_t* that receives the information.
- [inout] `number_of_streams`: 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.

HAILORTAPI *hailo_status* **hailo_fuse_nms_frames**(const *hailo_nms_fuse_input_t* *nms_fuse_inputs, ...)

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).

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *nms_fuse_inputs*: Array of *hailo_nms_fuse_input_t* structs which contain the buffers to be fused.
- [in] *inputs_count*: How many members in *nms_fuse_inputs*.
- [out] *fused_buffer*: A pointer to a buffer which will contain the fused buffer.
- [in] *fused_buffer_size*: The fused buffer size.

9.7. Power measurement API functions

HAILORTAPI *hailo_status* **hailo_power_measurement**(*hailo_device* device, *hailo_dvm_options_t* dvm, ...)

Perform a single power measurement.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *device*: A *hailo_device* object.
- [in] *dvm*: 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 be measured with this option.
 - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): *OVERCURRENT_PROTECTION*.
- [in] *measurement_type*: The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES_AUTO* will select the default value according to the supported features.
- [out] *measurement*: The measured value. Measured units are determined due to *hailo_power_measurement_types_t*.

HAILORTAPI *hailo_status* **hailo_start_power_measurement**(*hailo_device* device, uint32_t ...)

Start performing a long power measurement.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] *device*: A *hailo_device* object.
- [in] *delay_milliseconds*: Amount of time between each measurement interval. This time period is sleep time of the core.
- [in] *averaging_factor*: Number of samples per time period, sensor configuration value.
- [in] *sampling_period*: 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: $2 * \text{sampling_period} * \text{averaging_factor}$ {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.

HAILORTAPI *hailo_status* **hailo_set_power_measurement**(*hailo_device* device, uint32_t index, ...)
Set parameters for long power measurement.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] index: Index of the buffer on the firmware the data would be saved at.
- [in] dvm: 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 be measured with this option.
 - Default (*HAILO_DVM_OPTIONS_AUTO*) for platforms supporting current monitoring (such as M.2 and mPCIe): OVERCURRENT_PROTECTION.
- [in] measurement_type: The type of the measurement. Choosing *HAILO_POWER_MEASUREMENT_TYPES_AUTO* will select the default value according to the supported features.

HAILORTAPI *hailo_status* **hailo_get_power_measurement**(*hailo_device* device, uint32_t index, bool ...)
Read measured power from a long power measurement

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.
- [in] index: Index of the buffer on the firmware the data would be saved at.
- [in] should_clear: Flag indicating if the results saved at the firmware will be deleted after reading.
- [out] measurement_data: 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*

HAILORTAPI *hailo_status* **hailo_stop_power_measurement**(*hailo_device* device)
Stop performing a long power measurement.

Return Upon success, returns *HAILO_SUCCESS*. Otherwise, returns a *hailo_status* error.

Parameters

- [in] device: A *hailo_device* object.

9.8. Other API definitions

enum hailo_status

HailoRT return codes

Values:

HAILO_SUCCESS = 0

Success - No error

HAILO_UNINITIALIZED = 1

No error code was initialized

HAILO_INVALID_ARGUMENT = 2

Invalid argument passed to function

HAILO_OUT_OF_HOST_MEMORY = 3

Cannot allocate more memory at host

HAILO_TIMEOUT = 4

Received a timeout

HAILO_INSUFFICIENT_BUFFER = 5

Buffer is insufficient

HAILO_INVALID_OPERATION = 6

Invalid operation

HAILO_NOT_IMPLEMENTED = 7

Code has not been implemented

HAILO_INTERNAL_FAILURE = 8

Unexpected internal failure

HAILO_DATA_ALIGNMENT_FAILURE = 9

Data is not aligned

HAILO_CHUNK_TOO_LARGE = 10

Chunk too large

HAILO_INVALID_LOGGER_LEVEL = 11

Used non-compiled level

HAILO_CLOSE_FAILURE = 12

Failed to close fd

HAILO_OPEN_FILE_FAILURE = 13

Failed to open file

HAILO_FILE_OPERATION_FAILURE = 14

File operation failure

HAILO_UNSUPPORTED_CONTROL_PROTOCOL_VERSION = 15

Unsupported control protocol version

HAILO_UNSUPPORTED_FW_VERSION = 16

Unsupported firmware version

HAILO_INVALID_CONTROL_RESPONSE = 17

Invalid control response

HAILO_FW_CONTROL_FAILURE = 18

Control failed in firmware

HAILO_ETH_FAILURE = 19

Ethernet operation has failed

HAILO_ETH_INTERFACE_NOT_FOUND = 20	Ethernet interface not found
HAILO_ETH_RECV_FAILURE = 21	Ethernet failed at recv operation
HAILO_ETH_SEND_FAILURE = 22	Ethernet failed at send operation
HAILO_INVALID_FIRMWARE = 23	Firmware bin is invalid
HAILO_INVALID_CONTEXT_COUNT = 24	Host build too many contexts
HAILO_INVALID_FRAME = 25	Part or all of the result data is invalid
HAILO_INVALID_HEF = 26	Invalid HEF
HAILO_PCIE_NOT_SUPPORTED_ON_PLATFORM = 27	PCIe not supported on platform
HAILO_INTERRUPTED_BY_SIGNAL = 28	Blocking syscall was interrupted by a signal
HAILO_START_VDMA_CHANNEL_FAIL = 29	Starting VDMA channel failure
HAILO_SYNC_VDMA_BUFFER_FAIL = 30	Synchronizing VDMA buffer failure
HAILO_STOP_VDMA_CHANNEL_FAIL = 31	Stopping VDMA channel failure
HAILO_CLOSE_VDMA_CHANNEL_FAIL = 32	Closing VDMA channel failure
HAILO_ATR_TABLES_CONF_VALIDATION_FAIL = 33	Validating address translation tables failure, for FW control use
HAILO_CONTROL_EVENT_CREATE_FAIL = 34	Creating control event failure
HAILO_READ_EVENT_FAIL = 35	Reading event failure
HAILO_PCIE_DRIVER_FAIL = 36	PCIe driver failure
HAILO_INVALID_FIRMWARE_MAGIC = 37	Invalid FW magic
HAILO_INVALID_FIRMWARE_CODE_SIZE = 38	Invalid FW code size
HAILO_INVALID_KEY_CERTIFICATE_SIZE = 39	Invalid key certificate size
HAILO_INVALID_CONTENT_CERTIFICATE_SIZE = 40	Invalid content certificate size
HAILO_MISMATCHING_FIRMWARE_BUFFER_SIZES = 41	FW buffer sizes mismatch
HAILO_INVALID_FIRMWARE_CPU_ID = 42	Invalid CPU ID in FW

HAILO_CONTROL_RESPONSE_MD5_MISMATCH = 43	MD5 of control response does not match expected MD5
HAILO_GET_CONTROL_RESPONSE_FAIL = 44	Get control response failed
HAILO_GET_D2H_EVENT_MESSAGE_FAIL = 45	Reading device-to-host message failure
HAILO_MUTEX_INIT_FAIL = 46	Mutex initialization failure
HAILO_OUT_OF_DESCRIPTOR = 47	Cannot allocate more descriptors
HAILO_UNSUPPORTED_OPCODE = 48	Unsupported opcode was sent to device
HAILO_USER_MODE_RATE_LIMITER_NOT_SUPPORTED = 49	User mode rate limiter not supported on platform
HAILO_RATE_LIMIT_MAXIMUM_BANDWIDTH_EXCEEDED = 50	Rate limit exceeded HAILO_DEFAULT_MAX_ETHERNET_BANDWIDTH_BYTES_PER_SEC
HAILO_ANSI_TO_UTF16_CONVERSION_FAILED = 51	Failed converting ANSI string to UNICODE
HAILO_UTF16_TO_ANSI_CONVERSION_FAILED = 52	Failed converting UNICODE string to ANSI
HAILO_UNEXPECTED_INTERFACE_INFO_FAILURE = 53	Failed retrieving interface info
HAILO_UNEXPECTED_ARP_TABLE_FAILURE = 54	Failed retrieving arp table
HAILO_MAC_ADDRESS_NOT_FOUND = 55	MAC address not found in the arp table
HAILO_NO_IPV4_INTERFACES_FOUND = 56	No interfaces found with an IPv4 address
HAILO_SHUTDOWN_EVENT_SIGNED = 57	A shutdown event has been signaled
HAILO_THREAD_ALREADY_ACTIVATED = 58	The given thread has already been activated
HAILO_THREAD_NOT_ACTIVATED = 59	The given thread has not been activated
HAILO_THREAD_NOT_JOINABLE = 60	The given thread is not joinable
HAILO_NOT_FOUND = 61	Could not find element
HAILO_STREAM_ABORTED = 62	Stream aborted due to an external event
HAILO_STREAM_INTERNAL_ABORT = 63	Stream recv/send was aborted
HAILO_PCIE_DRIVER_NOT_INSTALLED = 64	Pcie driver is not installed
HAILO_NOT_AVAILABLE = 65	Component is not available

HAILO_TRAFFIC_CONTROL_FAILURE = 66

Traffic control failure

HAILO_INVALID_SECOND_STAGE = 67

Second stage bin is invalid

HAILO_INVALID_PIPELINE = 68

Pipeline is invalid

HAILO_NETWORK_GROUP_NOT_ACTIVATED = 69

Network group is not activated

HAILO_VSTREAM_PIPELINE_NOT_ACTIVATED = 70

Vstream pipeline is not activated

HAILO_OUT_OF_FW_MEMORY = 71

Cannot allocate more memory at fw

HAILO_STREAM_NOT_ACTIVATED = 72

Stream is not activated

HAILO_STATUS_MAX_ENUM = HAILO_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:

HAILO_DVM_OPTIONS_VDD_CORE = 0

VDD_CORE DVM

HAILO_DVM_OPTIONS_VDD_IO

VDD_IO DVM

HAILO_DVM_OPTIONS_MIPi_AVDD

MIPi_AVDD DVM

HAILO_DVM_OPTIONS_MIPi_AVDD_H

MIPi_AVDD_H DVM

HAILO_DVM_OPTIONS_USB_AVDD_IO

USB_AVDD_IO DVM

HAILO_DVM_OPTIONS_VDD_TOP

VDD_TOP DVM

HAILO_DVM_OPTIONS_USB_AVDD_IO_HV

USB_AVDD_IO_HV DVM

HAILO_DVM_OPTIONS_AVDD_H

AVDD_H DVM

HAILO_DVM_OPTIONS_SDIO_VDD_IO

SDIO_VDD_IO DVM

HAILO_DVM_OPTIONS_OVERCURRENT_PROTECTION

OVERCURRENT_PROTECTION DVM

HAILO_DVM_OPTIONS_COUNT

Must be last!

HAILO_DVM_OPTIONS_AUTO = INT_MAX

Select the default DVM option according to the supported features

HAILO_DVM_OPTIONS_MAX_ENUM = HAILO_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:

HAILO_POWER_MEASUREMENT_TYPES__SHUNT_VOLTAGE = 0

SHUNT_VOLTAGE measurement type, measured in mV

HAILO_POWER_MEASUREMENT_TYPES__BUS_VOLTAGE

BUS_VOLTAGE measurement type, measured in mV

HAILO_POWER_MEASUREMENT_TYPES__POWER

POWER measurement type, measured in W

HAILO_POWER_MEASUREMENT_TYPES__CURRENT

CURRENT measurement type, measured in mA

HAILO_POWER_MEASUREMENT_TYPES__COUNT

Must be last!

HAILO_POWER_MEASUREMENT_TYPES__AUTO = INT_MAX

Select the default measurement type according to the supported features

HAILO_POWER_MEASUREMENT_TYPES__MAX_ENUM = HAILO_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:

HAILO_SAMPLING_PERIOD_140US = 0

HAILO_SAMPLING_PERIOD_204US

HAILO_SAMPLING_PERIOD_332US

HAILO_SAMPLING_PERIOD_588US

HAILO_SAMPLING_PERIOD_1100US

HAILO_SAMPLING_PERIOD_2116US

HAILO_SAMPLING_PERIOD_4156US

HAILO_SAMPLING_PERIOD_8244US

HAILO_SAMPLING_PERIOD_MAX_ENUM = HAILO_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:

HAILO_AVERAGE_FACTOR_1 = 0

HAILO_AVERAGE_FACTOR_4

HAILO_AVERAGE_FACTOR_16

HAILO_AVERAGE_FACTOR_64

HAILO_AVERAGE_FACTOR_128

HAILO_AVERAGE_FACTOR_256

HAILO_AVERAGE_FACTOR_512

HAILO_AVERAGE_FACTOR_1024

HAILO_AVERAGE_FACTOR_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_device_architecture_e

Values:

HAILO_ARCH_HAILO8_A0 = 0

HAILO_ARCH_HAILO8_B0

HAILO_ARCH_MERCURY_CA

HAILO_ARCH_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_device_boot_source_t

Values:

HAILO_DEVICE_BOOT_SOURCE_INVALID = 0

HAILO_DEVICE_BOOT_SOURCE_PCIE

HAILO_DEVICE_BOOT_SOURCE_FLASH

HAILO_DEVICE_BOOT_SOURCE_MAX = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_endianness_t

Endianness (byte order)

Values:

HAILO_BIG_ENDIAN = 0

HAILO_LITTLE_ENDIAN = 1

HAILO_ENDIANNES_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_format_type_t

Data format types

Values:

HAILO_FORMAT_TYPE_AUTO = 0

Chosen automatically to match the format expected by the device, usually UINT8. Can be checked using [hailo_stream_info_t](#) format.type.

HAILO_FORMAT_TYPE_UINT8 = 1

Data format type uint8_t - 1 byte per item, host/device side

HAILO_FORMAT_TYPE_UINT16 = 2

Data format type uint16_t - 2 bytes per item, host/device side

HAILO_FORMAT_TYPE_FLOAT32 = 3

Data format type float32_t - used only on host side (Translated in the quantization process)

HAILO_FORMAT_TYPE_MAX_ENUM = HAILO_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:

HAILO_FORMAT_ORDER_AUTO = 0

Chosen automatically to match the format expected by the device.

HAILO_FORMAT_ORDER_NHWC = 1

- Host side: [N, H, W, C]
- Device side: [N, H, W, C], where width is padded to 8 elements

HAILO_FORMAT_ORDER_NHCW = 2

- Not used for host side
- Device side: [N, H, C, W], where width is padded to 8 elements

HAILO_FORMAT_ORDER_FCR = 3

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

HAILO_FORMAT_ORDER_F8CR = 4

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 8C₁, W X 8C₂, ... , W X 8C_n
- ROW2:
 - W X 8C₁, W X 8C₂, ... , W X 8C_n ...

HAILO_FORMAT_ORDER_NHW = 5

Output format of argmax layer:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 elements

HAILO_FORMAT_ORDER_NC = 6

Channels only:

- Host side: [N,C]
- Device side: [N, C], where channels are padded to 8 elements

HAILO_FORMAT_ORDER_BAYER_RGB = 7

Bayer format:

- Host side: [N, H, W, 1]
- Device side: [N, H, W, 1], where width is padded to 8 elements

HAILO_FORMAT_ORDER_12_BIT_BAYER_RGB = 8

Bayer format, same as [HAILO_FORMAT_ORDER_BAYER_RGB](#) where Channel is 12 bit

HAILO_FORMAT_ORDER_HAILO_NMS = 9

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

HAILO_FORMAT_ORDER_RGB888 = 10

- 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

HAILO_FORMAT_ORDER_NCHW = 11

- Host side: [N, C, H, W]
- Not used for device side

HAILO_FORMAT_ORDER_YUY2 = 12

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]

HAILO_FORMAT_ORDER_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_format_flags_t

Data format flags

Values:

HAILO_FORMAT_FLAGS_NONE = 0

HAILO_FORMAT_FLAGS_QUANTIZED = 1 « 0

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).

HAILO_FORMAT_FLAGS_TRANSPOSED = 1 « 1

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)

HAILO_FORMAT_FLAGS_MAX_ENUM = HAILO_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:

HAILO_STREAM_NO_TRANSFORM = 0

The vstream will not run the transformation (The data will be in hw format)

HAILO_STREAM_TRANSFORM_COPY = 1

The transformation process will be part of the vstream send/recv (The data will be in host format).

HAILO_STREAM_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_direction_t

Stream direction - host to device or device to host

Values:

HAILO_H2D_STREAM = 0

HAILO_D2H_STREAM = 1

HAILO_STREAM_DIRECTION_MAX_ENUM = HAILO_MAX_ENUM

enum hailo_mipi_pixels_per_clock_t

Indicates amount of pixels per clock on a MIPI stream

Values:

HAILO_MIPI_PIXELS_PER_CLOCK_1 = 0b0

HAILO_MIPI_PIXELS_PER_CLOCK_2 = 0b01

HAILO_MIPI_PIXELS_PER_CLOCK_4 = 0b10

HAILO_MIPI_PIXELS_PER_CLOCK_MAX_ENUM = HAILO_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:

HAILO_MIPI_CLOCK_SELECTION_80_TO_100_MBPS = 0b00000

HAILO_MIPI_CLOCK_SELECTION_100_TO_120_MBPS = 0b00001

HAILO_MIPI_CLOCK_SELECTION_120_TO_160_MBPS = 0b00010

HAILO_MIPI_CLOCK_SELECTION_160_TO_200_MBPS = 0b00011

HAILO_MIPI_CLOCK_SELECTION_200_TO_240_MBPS = 0b00100

HAILO_MIPI_CLOCK_SELECTION_240_TO_280_MBPS = 0b00101

HAILO_MIPI_CLOCK_SELECTION_280_TO_320_MBPS = 0b00110

HAILO_MIPI_CLOCK_SELECTION_320_TO_360_MBPS = 0b00111

HAILO_MIPI_CLOCK_SELECTION_360_TO_400_MBPS = 0b01000

HAILO_MIPI_CLOCK_SELECTION_400_TO_480_MBPS = 0b01001

HAILO_MIPI_CLOCK_SELECTION_480_TO_560_MBPS = 0b01010

HAILO_MIPI_CLOCK_SELECTION_560_TO_640_MBPS = 0b01011

HAILO_MIPI_CLOCK_SELECTION_640_TO_720_MBPS = 0b01100

HAILO_MIPI_CLOCK_SELECTION_720_TO_800_MBPS = 0b01101

HAILO_MIPI_CLOCK_SELECTION_800_TO_880_MBPS = 0b01110

HAILO_MIPI_CLOCK_SELECTION_880_TO_1040_MBPS = 0b01111

HAILO_MIPI_CLOCK_SELECTION_1040_TO_1200_MBPS = 0b10000

HAILO_MIPI_CLOCK_SELECTION_1200_TO_1350_MBPS = 0b10001

HAILO_MIPI_CLOCK_SELECTION_1350_TO_1500_MBPS = 0b10010

HAILO_MIPI_CLOCK_SELECTION_1500_TO_1750_MBPS = 0b10011

HAILO_MIPI_CLOCK_SELECTION_1750_TO_2000_MBPS = 0b10100

HAILO_MIPI_CLOCK_SELECTION_2000_TO_2250_MBPS = 0b10101

HAILO_MIPI_CLOCK_SELECTION_2250_TO_2500_MBPS = 0b10110

HAILO_MIPI_CLOCK_SELECTION_AUTOMATIC = 0b111111

The clock selection is calculated from the data rate.

HAILO_MIPI_CLOCK_SELECTION_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_data_type_rx_t

Indicates MIPI Rx data type

Values:

HAILO_MIPI_RX_TYPE_RGB_444 = 0x20

HAILO_MIPI_RX_TYPE_RGB_555 = 0x21

HAILO_MIPI_RX_TYPE_RGB_565 = 0x22

HAILO_MIPI_RX_TYPE_RGB_666 = 0x23

HAILO_MIPI_RX_TYPE_RGB_888 = 0x24

HAILO_MIPI_RX_TYPE_RAW_6 = 0x28

HAILO_MIPI_RX_TYPE_RAW_7 = 0x29

HAILO_MIPI_RX_TYPE_RAW_8 = 0x2a

HAILO_MIPI_RX_TYPE_RAW_10 = 0x2b

HAILO_MIPI_RX_TYPE_RAW_12 = 0x2c

HAILO_MIPI_RX_TYPE_RAW_14 = 0x2d

HAILO_MIPI_RX_TYPE_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_image_in_order_t

Indicates ISP input bayer pixel order

Values:

HAILO_MIPI_ISP_IMG_IN_ORDER_B_FIRST = 0

HAILO_MIPI_ISP_IMG_IN_ORDER_GB_FIRST = 1

HAILO_MIPI_ISP_IMG_IN_ORDER_GR_FIRST = 2

HAILO_MIPI_ISP_IMG_IN_ORDER_R_FIRST = 3

HAILO_MIPI_ISP_IMG_IN_ORDER_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_image_out_data_type_t

Indicates ISP output data type

Values:

HAILO_MIPI_IMG_OUT_DATA_TYPE_RGB_888 = 0x24

HAILO_MIPI_IMG_OUT_DATA_TYPE_YUV_422 = 0x1E

HAILO_MIPI_IMG_OUT_DATA_TYPE_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_mipi_isp_light_frequency_t

Values:

HAILO_MIPI_ISP_LIGHT_FREQUENCY_60HZ = 0

HAILO_MIPI_ISP_LIGHT_FREQUENCY_50HZ = 1

ISP_LIGHT_FREQUENCY_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_stream_interface_t

Values:

HAILO_STREAM_INTERFACE_PCIE = 0

HAILO_STREAM_INTERFACE_ETH

HAILO_STREAM_INTERFACE_MIPI

HAILO_STREAM_INTERFACE_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_power_mode_t

Power modes

Values:

HAILO_POWER_MODE_PERFORMANCE = 0

HAILO_POWER_MODE_ULTRA_PERFORMANCE = 1

HAILO_POWER_MODE_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_latency_measurement_flags_t

Latency measurement flags

Values:

HAILO_LATENCY_NONE = 0

HAILO_LATENCY_MEASURE = 1 « 0

HAILO_LATENCY_CLEAR_AFTER_GET = 1 « 1

HAILO_LATENCY_MAX_ENUM = HAILO_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:

HAILO_NOTIFICATION_ID_ETHERNET_RX_ERROR = 0

Matches [hailo_notification_message_parameters_t::rx_error_notification](#).

HAILO_NOTIFICATION_ID_HEALTH_MONITOR_TEMPERATURE_ALARM

Matches [hailo_notification_message_parameters_t::health_monitor_temperature_alarm_notification](#)

HAILO_NOTIFICATION_ID_HEALTH_MONITOR_DATAFLOW_SHUTDOWN

Matches [hailo_notification_message_parameters_t::health_monitor_dataflow_shutdown_notification](#)

HAILO_NOTIFICATION_ID_HEALTH_MONITOR_OVERCURRENT_ALARM

Matches [hailo_notification_message_parameters_t::health_monitor_overcurrent_alert_notification](#)

HAILO_NOTIFICATION_ID_LCU_ECC_CORRECTABLE_ERROR

Matches [hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification](#)

HAILO_NOTIFICATION_ID_LCU_ECC_UNCORRECTABLE_ERROR

Matches *hailo_notification_message_parameters_t::health_monitor_lcu_ecc_error_notification*

HAILO_NOTIFICATION_ID_CPU_ECC_ERROR

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

HAILO_NOTIFICATION_ID_CPU_ECC_FATAL

Matches *hailo_notification_message_parameters_t::health_monitor_cpu_ecc_notification*

HAILO_NOTIFICATION_ID_DEBUG

Matches *hailo_notification_message_parameters_t::debug_notification*

HAILO_NOTIFICATION_ID_CONTEXT_SWITCH_BREAKPOINT_REACHED

Matches *hailo_notification_message_parameters_t::context_switch_breakpoint_reached_notification*

HAILO_NOTIFICATION_ID_HEALTH_MONITOR_CLOCK_CHANGED_EVENT

Matches *hailo_notification_message_parameters_t::health_monitor_clock_changed_notification*

HAILO_NOTIFICATION_ID_COUNT

Must be last!

HAILO_NOTIFICATION_ID_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

enum hailo_temperature_protection_temperature_zone_t

Values:

HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__GREEN = 0

HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__ORANGE = 1

HAILO_TEMPERATURE_PROTECTION_TEMPERATURE_ZONE__RED = 2

enum hailo_overcurrent_protection_overcurrent_zone_t

Values:

HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__NONE = 0

HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__ORANGE = 1

HAILO_OVERCURRENT_PROTECTION_OVERCURRENT_ZONE__RED = 2

enum hailo_reset_device_mode_t

Hailo device reset modes

Values:

HAILO_RESET_DEVICE_MODE_CHIP = 0

HAILO_RESET_DEVICE_MODE_NN_CORE = 1

HAILO_RESET_DEVICE_MODE_SOFT = 2

HAILO_RESET_DEVICE_MODE_FORCED_SOFT = 3

HAILO_RESET_DEVICE_MODE_MAX_ENUM = HAILO_MAX_ENUM

enum hailo_sensor_types_t

Values:

HAILO_SENSOR_TYPES_GENERIC = 0

HAILO_SENSOR_TYPES_ONSEMI_AR0220AT

HAILO_SENSOR_TYPES_RASPICAM

HAILO_SENSOR_TYPES_ONSEMI_AS0149AT

HAILO_SENSOR_TYPES_HAILO8_ISP = 0x80000000

HAILO_SENSOR_TYPES_MAX_ENUM = HAILO_MAX_ENUM

Max enum value to maintain ABI Integrity

```
enum hailo_fw_logger_interface_t
```

Values:

```
HAILO_FW_LOGGER_INTERFACE_PCIE = 1 « 0
```

```
HAILO_FW_LOGGER_INTERFACE_UART = 1 « 1
```

```
HAILO_FW_LOGGER_INTERFACE_MAX_ENUM = HAILO_MAX_ENUM
```

Max enum value to maintain ABI Integrity

```
enum hailo_fw_logger_level_t
```

Values:

```
HAILO_FW_LOGGER_LEVEL_TRACE = 0
```

```
HAILO_FW_LOGGER_LEVEL_DEBUG = 1
```

```
HAILO_FW_LOGGER_LEVEL_INFO = 2
```

```
HAILO_FW_LOGGER_LEVEL_WARN = 3
```

```
HAILO_FW_LOGGER_LEVEL_ERROR = 4
```

```
HAILO_FW_LOGGER_LEVEL_FATAL = 5
```

```
HAILO_FW_LOGGER_LEVEL_MAX_ENUM = HAILO_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 void *hailo_device
```

Represents the device (chip)

```
typedef void *hailo_hef
```

Compiled HEF model that can be loaded to Hailo devices

```
typedef void *hailo_input_stream
```

Input (host to device) stream representation

```
typedef void *hailo_output_stream
```

Output (device to host) stream representation

```
typedef void *hailo_configured_network_group
```

Loaded network_group that can be activated

```
typedef void *hailo_activated_network_group
```

Activated network_group that can be used to send/receive data

```
typedef void *hailo_input_transformer
```

Object used for input stream transformation, store all necessary allocated buffers

```
typedef void *hailo_output_transformer
```

Object used for output stream transformation, store all necessary allocated buffers

```
typedef void *hailo_output_demuxer
```

Object used to demux muxed stream

```
typedef void *hailo_input_vstream
```

Input virtual stream

```
typedef void *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_device_architecture_e hailo_device_architecture_t
```

```
typedef void (*hailo_notification_callback)(hailo_device, const hailo_notification_t*, void*)
```

A notification callback. See [hailo_set_notification_callback](#)

Warning Throwing exceptions in the callback is not supported!

Parameters

- [in] device: The [hailo_device](#) that got the notification.
- [in] notification: The notification data.
- [in] opaque: User specific data.

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_MAX_STREAM_NAME_SIZE

HAILO_MAX_BOARD_NAME_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_PCIE_ANY_DOMAIN

HAILO_DEFAULT_VSTREAM_QUEUE_SIZE

HAILO_DEFAULT_VSTREAM_TIMEOUT_MS

HAILO_SOC_ID_LENGTH

HAILO_ETH_MAC_LENGTH

HAILO_UNIT_LEVEL_TRACKING_BYTES_LENGTH

HAILO_SOC_PM_VALUES_BYTES_LENGTH

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_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**[HAILO_MAX_BOARD_NAME_LENGTH]

bool **is_release**

hailo_device_architecture_t **device_architecture**

uint8_t **serial_number_length**

char **serial_number**[HAILO_MAX_SERIAL_NUMBER_LENGTH]

uint8_t **part_number_length**

char **part_number**[HAILO_MAX_PART_NUMBER_LENGTH]

uint8_t **product_name_length**

char **product_name**[HAILO_MAX_PRODUCT_NAME_LENGTH]

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**[HAILO_SOC_ID_LENGTH]
SOC id

uint8_t **lcs**
Device lcs

uint8_t **eth_mac_address**[HAILO_ETH_MAC_LENGTH]
Device Ethernet Mac address

uint8_t **unit_level_tracking_id**[HAILO_UNIT_LEVEL_TRACKING_BYTES_LENGTH]
Hailo device unit level tracking id

uint8_t **soc_pm_values**[HAILO_SOC_PM_VALUES_BYTES_LENGTH]
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_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

hailo_demux_params_t::EMPTY_STRUCT_PLACEHOLDER

struct hailo_quant_info_t

#include <hailort.h> Quantization information

Public Members

float32_t qp_zp

float32_t qp_scale

float32_t limvals_min

float32_t limvals_max

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 the desired rate other than zero. The limitation 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

hailo_pcie_input_stream_params_t::EMPTY_STRUCT_PLACEHOLDER

struct **hailo_pcie_output_stream_params_t**

#include <hailort.h> PCIe output stream (device to host) parameters

Public Members

hailo_pcie_output_stream_params_t::EMPTY_STRUCT_PLACEHOLDER

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_stream_parameters_t

#include <hailort.h> Hailo stream parameters

Public Members*hailo_stream_interface_t* interface*hailo_stream_direction_t* direction*hailo_pcie_input_stream_params_t* pcie_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_eth_output_stream_params_t* eth_output_paramsunion *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[HAILO_MAX_STREAM_NAME_SIZE]

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

struct *hailo_input_vstream_params_by_name_t*

#include <hailort.h> Input virtual stream parameters

Public Members

char name[HAILO_MAX_STREAM_NAME_SIZE]

hailo_vstream_params_t params**struct** *hailo_output_vstream_params_by_name_t*

#include <hailort.h> Output virtual stream parameters

Public Members

char name[HAILO_MAX_STREAM_NAME_SIZE]

hailo_vstream_params_t params**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**[HAILO_MAX_STREAM_NAME_SIZE]

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**

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_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[HAILO_MAX_STREAM_NAME_SIZE]

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[HAILO_MAX_STREAM_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_configure_network_group_params_t

#include <hailort.h> Hailo configure parameters per network_group

Public Members

char name[HAILO_MAX_NETWORK_GROUP_NAME_SIZE]

uint16_t batch_size

hailo_power_mode_t power_mode

This parameter is only used in multi-context network_groups. In case of name mismatch, default value HAILO_DEFAULT_BATCH_SIZE is used

hailo_latency_measurement_flags_t latency

size_t stream_params_by_name_count

hailo_stream_parameters_by_name_t stream_params_by_name[HAILO_MAX_STREAMS_COUNT]

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[HAILO_MAX_NETWORK_GROUPS]

struct hailo_activate_network_group_params_t

#include <hailort.h> Hailo network_group parameters

Public Members

hailo_activate_network_group_params_t::EMPTY_STRUCT_PLACEHOLDER

struct hailo_network_group_info_t

#include <hailort.h> Hailo network group info

Public Members

char name[HAILO_MAX_NETWORK_GROUP_NAME_SIZE]

struct hailo_layer_name_t

#include <hailort.h> Hailo layer name

Public Members

char name[HAILO_MAX_STREAM_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 pcie_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**

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**

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

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_stream_raw_buffer_t

Public Members

void ***buffer**

size_t **size**

struct hailo_latency_measurement_result_t

Public Members

[float64_t](#) **avg_hw_latency_ms**

struct hailo_rate_limit_t

Public Members

char **stream_name**[HAILO_MAX_STREAM_NAME_SIZE]

uint32_t **rate**

10. HailoRT Python API Reference

`HailoHWObject` is the high level base class of the platform API. `PcieDevice` inherits from it and implements control and dataflow over PCIe.

10.1. hailo_platform.drivers.hw_object

Hailo hardware API

exception `hailo_platform.drivers.hw_object.HailoHWObjectException`

Bases: `hailo_platform.common.targets.inference_targets.InferenceTargetException`

Raised in any error related to Hailo hardware.

class `hailo_platform.drivers.hw_object.HailoHWObject`

Bases: `hailo_platform.common.targets.inference_targets.InferenceObject`

Abstract Hailo hardware device representation.

IS_NUMERIC = `True`

__init__()

Create the Hailo hardware object.

property `sorted_output_layer_names`

Getter for the property `sorted_output_names`. :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.

Returns Keys are device output names are 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.

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.

property `device_output_layers`

Get a list of the names of the device's outputs.

hef_loaded()

Return True if this object has loaded the model HEF to the hardware device.

outputs_count()

Return the amount of output tensors that are returned from the hardware device for every input image.

property `model_name`

Get the name of the current model.

Returns Model name.

Return type str

get_output_shapes()

Get the model output shapes, as returned to the user (without any hardware padding).

Returns Tuple of output shapes, sorted by the output names.

class `hailo_platform.drivers.hw_object.HailoChipObject(hw_arch='hailo8')`

Bases: `hailo_platform.drivers.hw_object.HailoHWObject`

Hailo hardware device representation

IS_NUMERIC = `True`

IS_HARDWARE = `True`

__init__(`hw_arch='hailo8'`)

Create the Hailo Chip hardware object.

Parameters `hw_arch`(str, optional) – Name of the device hardware architecture. Defaults to `hailo8`.

property control

`HailoControl`: 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.

get_all_input_layers_dtype()

Get the model inputs dtype.

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_layers_info()

Get the input layers info of the configured network group.

Returns list of `hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo` if there is exactly one configured network group, otherwise raises an exception.

get_output_layers_info(`expand_mux=True`)

Get the output layers info of the configured network group.

Parameters `expand_mux` (bool) – A Boolean flag that specifies whether or not the function should expand every mux layer object it gets to its underlying `hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo` objects.

Returns If there is exactly one configured network group, returns a list of `hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo` with information objects of all output layers. If `expand_mux` is set to `True`, the list will only contain demuxed layers.

get_input_vstream_infos()

get_output_vstream_infos()

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`

configure(`hef, configure_params_by_name={}`)

Configures target 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`. If not provided, default params will be applied

get_input_shape(*name=None*)

Get the input shape (not padded) of a network.

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.

Parameters *name* (str) – The name of the output.

Returns The index of the layer name in the output list.

Return type int

class hailo_platform.drivers.hw_object.**EthernetDevice**(*remote_ip, remote_control_port=22401, ...*)

Bases: [hailo_platform.drivers.hw_object.HailoChipObject](#)

Represents any Hailo hardware device that supports UDP control and dataflow.

NAME = 'udp'

__init__(*remote_ip, remote_control_port=22401, hw_arch='hailo8'*)

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.
- **hw_arch** (str, optional) – Name of the device hardware architecture. Defaults to hailo8.

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.

class hailo_platform.drivers.hw_object.**PcieDevice**(*hw_arch='hailo8', device_info=None*)

Bases: [hailo_platform.drivers.hw_object.HailoChipObject](#)

Hailo PCIe production device representation.

NAME = 'pcie'

__init__(*hw_arch='hailo8', device_info=None*)

Create the Hailo PCIe hardware object.

Parameters

- **hw_arch** (str, optional) – Name of the device hardware architecture. Defaults to hailo8.
- **device_info** ([hailo_platform.drivers.hailort.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.

Returns list of `hailo_platform.drivers.hailort.pyhailort.PcieDeviceInfo`

10.2. hailo_platform.drivers.hailort.pyhailort

class `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

Bases: `Exception`

class `hailo_platform.drivers.hailort.pyhailort.UdpRecvError`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.InvalidProtocolVersionException`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.HailoRTFirmwareControlFailedException`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.HailoRTInvalidFrameException`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.HailoRTUnsupportedOpcodeException`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.HailoRTTimeout`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.pyhailort.HailoRTStreamAborted`

Bases: `hailo_platform.drivers.hailort.pyhailort.HailoRTException`

class `hailo_platform.drivers.hailort.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 a hef file. If the source type is *bytes*, it is treated as a buffer. Any other type will raise a `ValueError`.

property *path*

HEF file path.

get_network_group_names()

Get the names of the network groups in this HEF.

get_all_layers_info(*network_group_name*="", *expand_mux*=True)

Get input and output layers information of a specific network group.

Parameters

- **network_group_name** (str) – The name of the network group to access.
- **expand_mux** (bool) – A Boolean flag that specifies whether or not the function should expand every mux layer object it gets to its underlying `hailo_platform.drivers.hailort.pyhailort.HailoLayerInfo` objects.

Returns The first item has the information objects of all input layers. The second item has the information objects of all output layers. If *expand_mux* is set to True, the second list will only contain demuxed layers.

Return type tuple of (list of hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo, list of hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo)

get_input_layers_info(*network_group_name=""*)

Get information about the input layers in a given network group. If the network group is not given, the first one is used.

get_output_layers_info(*network_group_name="", expand_mux=True*)

Get output layers information of a specific network group.

Parameters

- **network_group_name** (str) – The name of the network group to access.
- **expand_mux** (bool) – A Boolean flag that specifies whether or not the function should expand every mux layer object it gets to its underlying hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo objects.

Returns Information objects of all output layers. If expand_mux is set to True, the list will be sorted according to sorted_output_names and only contain demuxed layers, otherwise the list will not be sorted.

Return type list of hailo_platform.drivers.hailort._pyhailort.HailoLayerInfo

get_sorted_output_names(*network_group_name=""*)

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_stream_name_from_original_name(*original_name, network_group_name=""*)

Get stream 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.

Returns the matching stream name for the provided original name.

Return type str

get_original_names_from_stream_name(*stream_name, network_group_name=""*)

Get original names list from stream name for a specific network group.

Parameters

- **stream_name** (str) – The stream name.
- **network_group_name** (str, optional) – The name of the network group to access.

Returns all the matching original layers names for the provided stream name.

Return type list of str

class hailo_platform.drivers.hailort.pyhailort.**PcieDeviceInfo**(*bus, device, func, domain=None*)

Bases: hailo_platform.drivers.hailort._pyhailort.PcieDeviceInfo

Represents pcie device info, including domain, bus, device and function.

BOARD_LOCATION_HELP_STRING = 'Board location in the format of the command: "lspci -d 1e60: | cut -d\' \'

__init__(*self: hailo_platform.drivers.hailort.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.drivers.hailort.pyhailort.ConfiguredNetwork(configured_network, ...)
    Bases: object

    Represents a network group loaded to the device.

    __init__(configured_network, target, hef)
        Initialize self. See help(type(self)) for accurate signature.

    activate(network_group_params=None)
        Activate this network group in order to infer data through it.

        Parameters network_group_params (hailo_platform.drivers.hailort._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

    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 activa-
            tion. Defaults to HAILO_INFINITE.

        Raises HailoRTTimeout – In case of timeout.

    static create_params()
        Create activation params for network_group.

        Returns hailo_platform.drivers.hailort._pyhailort.
            ActivateNetworkGroupParams.

    property name

    get_all_layers_info(expand_mux=True)

    get_input_layers_info()

    get_output_layers_info(expand_mux=True)

    get_output_shapes()

    get_sorted_output_names()

    get_input_vstream_infos()

    get_output_vstream_infos()

    get_all_vstream_infos()

    get_udp_rates_dict(fps, max_supported_rate_bytes)

class hailo_platform.drivers.hailort.pyhailort.ActivatedNetworkContextManager(configured_network, ...)
    Bases: object

    A context manager that returns the activated network group upon enter.

    __init__(configured_network, activated_network, target, hef)
        Initialize self. See help(type(self)) for accurate signature.

class hailo_platform.drivers.hailort.pyhailort.ActivatedNetwork(configured_network, ...)
    Bases: object

    The network group that is currently activated for inference.

    __init__(configured_network, activated_network, target, hef)
        Initialize self. See help(type(self)) for accurate signature.

    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.drivers.hailort.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

FLOAT32 = FormatType.FLOAT32

UINT16 = FormatType.UINT16

UINT8 = FormatType.UINT8

__init__(self: hailo_platform.drivers.hailort.pyhailort.FormatType, arg0: int) → None

property name

(self: handle) -> str

class hailo_platform.drivers.hailort.pyhailort.**PowerMeasurementData**

Bases: pybind11_builtins.pybind11_object

__init__()

Initialize self. See help(type(self)) for accurate signature.

property average_time_value_milliseconds

float, Average time in milliseconds between samples

property average_value

float, The average value of the samples that were sampled

equals(self: hailo_platform.drivers.hailort.pyhailort.PowerMeasurementData, arg0: ...)

property max_value

float, The maximum 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.drivers.hailort.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

BUS_VOLTAGE = PowerMeasurementTypes.BUS_VOLTAGE

CURRENT = PowerMeasurementTypes.CURRENT

POWER = PowerMeasurementTypes.POWER

SHUNT_VOLTAGE = PowerMeasurementTypes.SHUNT_VOLTAGE

__init__(self: *hailo_platform.drivers.hailort_pyhailort.PowerMeasurementTypes*, arg0: int) → None

property name

(self: handle) -> str

class *hailo_platform.drivers.hailort_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

AVDD_H = DvmTypes.AVDD_H

MIPI_AVDD = DvmTypes.MIPI_AVDD

MIPI_AVDD_H = DvmTypes.MIPI_AVDD_H

OVERCURRENT_PROTECTION = DvmTypes.OVERCURRENT_PROTECTION

SDIO_VDD_IO = DvmTypes.SDIO_VDD_IO

USB_AVDD_IO = DvmTypes.USB_AVDD_IO

USB_AVDD_IO_HV = DvmTypes.USB_AVDD_IO_HV

VDD_CORE = DvmTypes.VDD_CORE

VDD_IO = DvmTypes.VDD_IO

VDD_TOP = DvmTypes.VDD_TOP

__init__(self: *hailo_platform.drivers.hailort_pyhailort.DvmTypes*, arg0: int) → None

property name

(self: handle) -> str

class hailo_platform.drivers.hailort.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

PERIOD_140us = SamplingPeriod.PERIOD_140us

PERIOD_204us = SamplingPeriod.PERIOD_204us

PERIOD_2116us = SamplingPeriod.PERIOD_2116us

PERIOD_332us = SamplingPeriod.PERIOD_332us

PERIOD_4156us = SamplingPeriod.PERIOD_4156us

PERIOD_588us = SamplingPeriod.PERIOD_588us

PERIOD_8244us = SamplingPeriod.PERIOD_8244us

__init__(self: hailo_platform.drivers.hailort.pyhailort.SamplingPeriod, arg0: int) → None

property name

(self: handle) -> str

class hailo_platform.drivers.hailort.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

AVERAGE_1024 = AveragingFactor.AVERAGE_1024

AVERAGE_128 = AveragingFactor.AVERAGE_128

```

AVERAGE_16 = AveragingFactor.AVERAGE_16
AVERAGE_256 = AveragingFactor.AVERAGE_256
AVERAGE_4 = AveragingFactor.AVERAGE_4
AVERAGE_512 = AveragingFactor.AVERAGE_512
AVERAGE_64 = AveragingFactor.AVERAGE_64
__init__(self: hailo_platform.drivers.hailort.pyhailort.AveragingFactor, arg0: int) → None
property name
    (self: handle) -> str
class hailo_platform.drivers.hailort.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
    RAW_12 = MipiDataTypeRx.RAW_12
    RAW_14 = MipiDataTypeRx.RAW_14
    RAW_6 = MipiDataTypeRx.RAW_6
    RAW_7 = MipiDataTypeRx.RAW_7
    RAW_8 = MipiDataTypeRx.RAW_8
    RGB_444 = MipiDataTypeRx.RGB_444
    RGB_555 = MipiDataTypeRx.RGB_555
    RGB_565 = MipiDataTypeRx.RGB_565
    RGB_666 = MipiDataTypeRx.RGB_666
    RGB_888 = MipiDataTypeRx.RGB_888
    __init__(self: hailo_platform.drivers.hailort.pyhailort.MipiDataTypeRx, arg0: int) → None
    property name
        (self: handle) -> str
class hailo_platform.drivers.hailort.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
PIXELS_PER_CLOCK_2 = MipiPixelsPerClock.PIXELS_PER_CLOCK_2
PIXELS_PER_CLOCK_4 = MipiPixelsPerClock.PIXELS_PER_CLOCK_4
__init__(self: hailo_platform.drivers.hailort.pyhailort.MipiPixelsPerClock, arg0: int) → None
property name
    (self: handle) -> str
class hailo_platform.drivers.hailort.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
        SELECTION_1040_TO_1200_MBPS = MipiClockSelection.SELECTION_1040_TO_1200_MBPS
        SELECTION_1200_TO_1350_MBPS = MipiClockSelection.SELECTION_1200_TO_1350_MBPS
        SELECTION_120_TO_160_MBPS = MipiClockSelection.SELECTION_120_TO_160_MBPS
        SELECTION_1350_TO_1500_MBPS = MipiClockSelection.SELECTION_1350_TO_1500_MBPS

```

```

SELECTION_1500_TO_1750_MBPS = MipiClockSelection.SELECTION_1500_TO_1750_MBPS
SELECTION_160_TO_200_MBPS = MipiClockSelection.SELECTION_160_TO_200_MBPS
SELECTION_1750_TO_2000_MBPS = MipiClockSelection.SELECTION_1750_TO_2000_MBPS
SELECTION_2000_TO_2250_MBPS = MipiClockSelection.SELECTION_2000_TO_2250_MBPS
SELECTION_200_TO_240_MBPS = MipiClockSelection.SELECTION_200_TO_240_MBPS
SELECTION_2250_TO_2500_MBPS = MipiClockSelection.SELECTION_2250_TO_2500_MBPS
SELECTION_240_TO_280_MBPS = MipiClockSelection.SELECTION_240_TO_280_MBPS
SELECTION_280_TO_320_MBPS = MipiClockSelection.SELECTION_280_TO_320_MBPS
SELECTION_320_TO_360_MBPS = MipiClockSelection.SELECTION_320_TO_360_MBPS
SELECTION_360_TO_400_MBPS = MipiClockSelection.SELECTION_360_TO_400_MBPS
SELECTION_400_TO_480_MBPS = MipiClockSelection.SELECTION_400_TO_480_MBPS
SELECTION_480_TO_560_MBPS = MipiClockSelection.SELECTION_480_TO_560_MBPS
SELECTION_560_TO_640_MBPS = MipiClockSelection.SELECTION_560_TO_640_MBPS
SELECTION_640_TO_720_MBPS = MipiClockSelection.SELECTION_640_TO_720_MBPS
SELECTION_720_TO_800_MBPS = MipiClockSelection.SELECTION_720_TO_800_MBPS
SELECTION_800_TO_880_MBPS = MipiClockSelection.SELECTION_800_TO_880_MBPS
SELECTION_80_TO_100_MBPS = MipiClockSelection.SELECTION_80_TO_100_MBPS
SELECTION_880_TO_1040_MBPS = MipiClockSelection.SELECTION_880_TO_1040_MBPS
SELECTION_AUTOMATIC = MipiClockSelection.SELECTION_AUTOMATIC

__init__(self: hailo_platform.drivers.hailort_pyhailort.MipiClockSelection, arg0: int) → None

property name
    (self: handle) -> str

```

```

class hailo_platform.drivers.hailort_pyhailort.MipiIspImageInOrder
    Bases: pybind11_builtins.pybind11_object

    Members:

    B_FIRST
    GB_FIRST
    GR_FIRST
    R_FIRST

    B_FIRST = MipiIspImageInOrder.B_FIRST
    GB_FIRST = MipiIspImageInOrder.GB_FIRST
    GR_FIRST = MipiIspImageInOrder.GR_FIRST
    R_FIRST = MipiIspImageInOrder.R_FIRST

    __init__(self: hailo_platform.drivers.hailort_pyhailort.MipiIspImageInOrder, arg0: int) → None

    property name
        (self: handle) -> str

class hailo_platform.drivers.hailort_pyhailort.MipiIspImageOutDataType
    Bases: pybind11_builtins.pybind11_object

    Members:

    RGB_888

```

YUV_422

RGB_888 = `MipiIspImageOutDataType.RGB_888`

YUV_422 = `MipiIspImageOutDataType.YUV_422`

`__init__(self: hailo_platform.drivers.hailort.pyhailort.MipiIspImageOutDataType, arg0: int) → None`

property name

(self: handle) -> str

class `hailo_platform.drivers.hailort.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`

LIGHT_FREQ_60_HZ = `IspLightFrequency.LIGHT_FREQ_60_HZ`

`__init__(self: hailo_platform.drivers.hailort.pyhailort.IspLightFrequency, arg0: int) → None`

property name

(self: handle) -> str

class `hailo_platform.drivers.hailort.pyhailort.Endianness`

Bases: `pybind11_builtins.pybind11_object`

Members:

BIG_ENDIAN

LITTLE_ENDIAN

BIG_ENDIAN = `Endianness.BIG_ENDIAN`

LITTLE_ENDIAN = `Endianness.LITTLE_ENDIAN`

`__init__(self: hailo_platform.drivers.hailort.pyhailort.Endianness, arg0: int) → None`

property name

(self: handle) -> str

class `hailo_platform.drivers.hailort.pyhailort.InputVStreamParams`

Bases: `object`

Parameters of an input virtual stream (host to device).

static `make_from_network_group(configured_network, quantized=True, format_type=None, ...)`

Create input virtual stream params from a 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`.

Returns The created virtual streams params. The keys are the streams names. The values are the params.

Return type dict

class `hailo_platform.drivers.hailort.pyhailort.OutputVStreamParams`

Bases: object

Parameters of an output virtual stream (device to host).

static `make_from_network_group(configured_network, quantized=True, format_type=None, ...)`

Create output virtual stream params from a 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 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 The created virtual streams params. The keys are the streams names. The values are the params.

Return type dict

class `hailo_platform.drivers.hailort.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`)

Initialize self. See `help(type(self))` for accurate signature.

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`

class `hailo_platform.drivers.hailort.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`)

Initialize self. See `help(type(self))` for accurate signature.

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](#)

class `hailo_platform.drivers.hailort.pyhailort.InputVStream`(*send_object*)

Bases: object

Represents a single virtual stream in the host to device direction.

__init__(*send_object*)

Initialize self. See help(type(self)) for accurate signature.

property **shape**

property **dtype**

send(*input_data*)

Send frames to inference.

Parameters **input_data** (`numpy.ndarray`) – Data to run inference on.

class `hailo_platform.drivers.hailort.pyhailort.OutputVStream`(*configured_network, ...*)

Bases: object

Represents a single output virtual stream in the device to host direction.

__init__(*configured_network, rcv_object, name, tf_nms_format=False, net_group_name=""*)

Initialize self. See help(type(self)) for accurate signature.

property **shape**

property **dtype**

recv()

Receive frames after inference.

Returns The output of the inference for a single frame. The returned tensor does not include the batch dimension.

Return type `numpy.ndarray`

10.3. hailo_platform.drivers.control_object

Control operations for the Hailo hardware device.

exception `hailo_platform.drivers.control_object.ControlObjectException`

Bases: Exception

Raised on illegal ControlObject operation.

exception `hailo_platform.drivers.control_object.FirmwareUpdateException`

Bases: Exception

class `hailo_platform.drivers.control_object.HailoControl`

Bases: object

Control object that sends control operations to a Hailo hardware device.

__init__()

Initializes a new HailoControl object.

abstract open()

Initializes the resources needed for using a control device.

abstract close()

Releases the resources that were allocated for the control device.

abstract device_dataflow_enable()

Enables neural network processing in the Hailo HW device. It should be called after writing configurations and before sending dataflow data.

abstract device_dataflow_disable()

Disables neural network processing in the Hailo HW device.

abstract device_reset_dataflow_state()

Sets the device to a clean state. When in this state, the device is ready to be configured and to send dataflow data.

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.

abstract chip_reset()

Resets the device (chip reset).

abstract 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

abstract 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.

class hailo_platform.drivers.control_object.HcpControl

Bases: [hailo_platform.drivers.control_object.HailoControl](#)

Control object that uses the HCP protocol for controlling the device.

WORD_SIZE = 4

__init__()

Initializes a new HailoControl object.

property device_id

Getter for the device_id.

open()

Initializes the resources needed for using a control device.

close()

Releases the resources that were allocated for the control device.

close_all_streams()

Close all input and output streams.

set_context_switch_breakpoint(*breakpoint_id*, *break_at_any_application_index*, *application_index*, ...)
continue_context_switch_breakpoint(*breakpoint_id*)
clear_context_switch_breakpoint(*breakpoint_id*)
config_context_switch_timestamp(*batch_index*)
remove_context_switch_timestamp_configuration()
config_ahb_to_axi(*use_64bit_data_only*)
device_dataflow_enable()

Enables neural network processing in the Hailo HW device. It should be called after writing configurations and before sending dataflow data.

device_dataflow_disable()

Disables neural network processing in the Hailo HW device.

device_reset_dataflow_state()

Sets the device to a clean state. When in this state, the device is ready to be configured and to send dataflow data.

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)

phy_operation(*operation_type*)
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.

power_measurement(*dvm=DvmTypes.AUTO*, *measurement_type=PowerMeasurementTypes.AUTO*)

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 be 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.

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(*delay=675, averaging_factor=AveragingFactor.AVERAGE_256, ...*)

Start performing a long power measurement.

Parameters

- **delay** (int) – Amount of time between each measurement interval (in milliseconds) This time period is sleep time of the core. The default value depends on the other default values, plus a factor of 20 percent.
- **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: $2 * \text{sampling_period} * \text{averaging_factor}$ [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(*index, dvm=DvmTypes.AUTO, ...*)

Set parameters for long power measurement on an Hailo chip.

Parameters

- **index** (int) – Index of the buffer on the firmware the data would be saved at.
- **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 be 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(*index, should_clear=True*)

Read measured power from a long power measurement

Parameters

- **index** (int) – Index of the buffer on the firmware the data would be saved at.
- **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 class `HailoIdentifyResponse` with Protocol version.

core_identify()

Gets the Core Hailo chip identification.

Returns class HailoIdentifyResponse with Protocol version.

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 component.

Parameters **should_activate** (bool) – Should be true to enable or false to disable.

get_throttling_state()

Get the current throttling state of temperature protection component.

Returns true if temperature throttling is enabled, false otherwise.

Return type bool

i2c_write(*slave, register_address, data*)

Write data to an I2C slave.

Parameters

- **slave** (hailo_platform.drivers.hailo_controller.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.drivers.hailo_controller.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(*is_first, section_index, start_offset, reset_data_size, sensor_type, total_data_size, ...*)

Store sensor configuration to Hailo chip flash memory.

Parameters

- **is_first** (int) – Erase the flash section for the first store command in the section.
- **section_index** (int) – Flash section index to write to.
- **start_offset** (int) – Offset to store the data in the section.
- **reset_data_size** (int) – Size of reset configuration.
- **sensor_type** ([SensorConfigTypes](#)) – Sensor type.
- **total_data_size** (int) – Total section data size.
- **config_height** (int) – Configuration resolution height.
- **config_width** (int) – Configuration resolution width.
- **config_fps** (int) – Configuration FPS.
- **config_buffer** (str) – Sensor configuration to be stored.
- **config_name** (bytes) – Sensor configuration name in bytes.

get_sensor_config(*section_index, offset, data_length*)

Get sensor configuration from Hailo chip flash memory.

Parameters

- **section_index** (int) – Flash section index to get the configuration from.
- **offset** – The start offset in the flash section.
- **data_length** – Size to read in bytes.

Returns Sensor configuration from Hailo chip flash memory.

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, should_hold_bus, ...*)

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.

reset_sensor(*section_index*)

Reset the sensor that is related to the section index config.

Parameters **section_index** (int) – Flash section index to reset.

wd_enable(*cpu_id*)

Enable firmware watchdog.

Parameters **cpu_id** (int) – 0 for App CPU, 1 for Core CPU.

wd_disable(*cpu_id*)

Disable firmware watchdog.

Parameters **cpu_id** (int) – 0 for App CPU, 1 for Core CPU.

wd_config(*cpu_id, wd_cycles, wd_mode*)

Configure a firmware watchdog.

Parameters

- **cpu_id** (int) – 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 previous system state.

Parameters **cpu_id** (int) – 0 for App CPU, 1 for Core CPU.

d2h_event_manager_set_host_info(*connection_type, host_port, host_ip_address*)

Set new host connection for d2h event manager use in the firmware.

Parameters

- **connection_type** (int) – 0 for UDP, 1 for PCIE.
- **host_port** (int) – host listening port (0 for PCIE).
- **host_ip_address** (int) – host ip, 0 – auto set IP address from control in case of UDP connection (0 for PCIE).

d2h_event_manager_send_host_info_event(*event_priority*)

Send an event with the host info of d2h event manager in the firmware.

Parameters **event_priority** (int) – 0 info, 1 for Critical.

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()

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

class `hailo_platform.drivers.control_object.UdpHcpControl`(*remote_ip, device=None, ...*)

Bases: `hailo_platform.drivers.control_object.HcpControl`

Control object that uses a HCP over UDP controller interface.

__init__(*remote_ip, device=None, remote_control_port=22401, retries=2, ...*)

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 useful for debugging).

set_udp_device(*device*)

class `hailo_platform.drivers.control_object.PcieHcpControl`(*device=None, device_info=None*)

Bases: `hailo_platform.drivers.control_object.HcpControl`

Control object that uses a HCP over PCIe controller interface.

__init__(*device=None, device_info=None*)

Initializes a new HailoPcieController object.

release()

set_pcie_device(*pcie_device*)

Prepare the pcie device to be used after creating it.

device_reset_dataflow_state()

Set device to a clean state. When in this state, the device is ready to be configured and to send dataflow data.

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.
- **opaque** (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.

10.4. hailo_platform.drivers.hailo_controller.i2c_slaves

`hailo_platform.drivers.hailo_controller.i2c_slaves.NO_I2C_SWITCH = 5`

Variable which defines that the I2C slave is not behind a switch.

exception `hailo_platform.drivers.hailo_controller.i2c_slaves.I2CSlavesException`

Bases: Exception

class `hailo_platform.drivers.hailo_controller.i2c_slaves.I2CSlave`(*name, bus_index, ...*)

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 slave 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 slave.

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.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_MIPI_AVDD`

Class which represents the MIPI AVDD I2C slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_USB_AVDD_IO`

Class which represents the USB AVDD IO slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_VDD_CORE`

Class which represents the V_CORE slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_VDD_TOP`

Class which represents the VDD TOP slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_MIPI_AVDD_H`

Class which represents the MIPI AVDD_H I2C slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_USB_AVDD_IO_HV`

Class which represents the DVM USB AVDD IO HV slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_VDD_IO`

Class which represents the DVM_VDDIO slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_AVDD_H`

Class which represents the DVM_AVDD_H slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_SDIO_VDD_IO`

Class which represents the DVM_SDIO_VDDIO slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_M_DOT_2_OVERCURREN_PROTECTION`

Class which represents the DVM_SDIO_VDDIO slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_I2S_CODEC`

Class which represents the I2S codec I2C slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_I2C_TO_GPIO`

Class which represents the I2C to gpio I2C slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_SWITCH`

Class which represents the I2C switch slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_TEMP_SENSOR_0`

Class which represents the I2C TEMP_sensor_0 slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_TEMP_SENSOR_1`

Class which represents the I2S TEMP_sensor_1 slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_EEPROM`

Class which represents the EEPROM I2C slave.

`hailo_platform.drivers.hailo_controller.i2c_slaves.I2C_SLAVE_RASPICAM`

Class which represents the raspicam I2C slave.

`hailo_platform.drivers.hailo_controller.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`.

10.5. 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.BadTCPParamError`

Bases: Exception

One of shell's tc command params is wrong.

exception `hailo_platform.tools.udp_rate_limiter.BadTCallError`

Bases: Exception

Shell's tc command has failed.

`hailo_platform.tools.udp_rate_limiter.get_max_supported_kbps(hw_arch='hailo8')`

class `hailo_platform.tools.udp_rate_limiter.RateLimiterWrapper(network_group, fps=1, ...)`

Bases: object

UDPRateLimiter wrapper enabling with statements.

`__init__(network_group, fps=1, fps_factor=1.0)`

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.

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)`

Initialize self. See help(type(self)) for accurate signature.

`set_rate_limit()`

`reset_rate_limit()`

static `calc_udp_rate(hef, network_group_name, fps, fps_factor=1, max_supported_kbps_rate=850000.0)`
 Calculates the proper UDP rate according to a HEF.

Parameters

- **hef** (str) – Path to a 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

class `hailo_platform.tools.udp_rate_limiter.UDPRateLimiterCLI(parser)`
 Bases: `hailo_platform.common.tools.cmd_utils.base_utils.HailortCliUtil`

CLI tool for UDP rate limitation.

__init__(*parser*)
 Initialize self. See `help(type(self))` for accurate signature.

10.6. hailo_platform.common.targets.inference_targets

class `hailo_platform.common.targets.inference_targets.InferenceObject(hw_arch=None)`
 Bases: `object`

Represents a target that can run models inference. The target can be either software based (eventually running on CPU/GPU), or Hailo hardware based.

Note: This class should not be used directly. Use only its inherited classes.

NAME = 'uninitialized'

IS_NUMERIC = False

IS_HARDWARE = False

IS_SIMULATION = False

__init__(*hw_arch=None*)
 Inference object constructor.

Parameters **hw_arch**(str, optional) – Name of the hardware architecture. Defaults to None.

use_device(**args, **kwargs*)
 A context manager that should wrap any usage of the target.

property name
 str: The name of this target. Valid values are defined by `InferenceObject`.

property is_numeric
 bool: Determines whether this target is working in numeric mode.

property is_hardware
 bool: Determines whether this target runs on a physical hardware device.

property is_simulation
 bool: Determines whether this target is used for HW simulation.

to_json()

Get a JSON representation of this object.

Returns A JSON dump.**Return type** str

10.7. hailo_platform.tools.firmware.sensor_config

Parses a configuration file that contains camera sensor configurations in the FW.

class hailo_platform.tools.firmware.sensor_config.SensorConfigTypes

Bases: enum.IntEnum

An enumeration.

SENSOR_GENERIC = 0**ONSEMI_AR0220AT** = 1**SENSOR_RASPICAM** = 2**ONSEMI_AS0149AT** = 3**HAILO8_ISP** = 2147483648

Python Module Index

h

`hailo_platform.common.targets.inference_targets`,
[113](#)

`hailo_platform.drivers`, [87](#)

`hailo_platform.drivers.control_object`, [101](#)

`hailo_platform.drivers.hailo_controller.i2c_slaves`,
[110](#)

`hailo_platform.drivers.hailort.pyhailort`, [90](#)

`hailo_platform.drivers.hw_object`, [87](#)

`hailo_platform.tools.firmware.sensor_config`,
[114](#)

`hailo_platform.tools.udp_rate_limiter`, [112](#)