



HailoRT - `cpp_agent` example documentation

Nadav Eden - Technical Account Manager

Aug. 2020

Contents

1	Intro	2
2	Perquisites	2
3	High Level Idea	2
4	Detailed	3
4.1	Instantiation	3
4.2	Configuration	4
4.3	Inference (Dataflow)	5
4.4	Summary Report	5
5	Building the Project	6

1 Intro

HailoRT is the C\C++ API library for interfacing the **Hailo8 device**. It includes all the needed API for **configuring and preparing the device** for running inference on a given neural-network (NN), and also the needed **API for the dataflow**. The cpp-agent example is a *suggested* implementation using C++, and a multi-thread approach. As it is an example, there are other alternative that might be better suited for a specific setup or system configuration.

2 Perquisites

We're using the following tool versions¹ to build this project on an x86 machine.

SW Tool	Version Used
CMake	3.0.0
Compiler	g++-9
ARM Compiler	aarch64-linux-gnu-g++
OS	Ubuntu 16.04
Kernel	4.15.0-96-generic

Also, at this point, you should already have a set of 4 JLF *or* HEF files, representing your model. If you do not have one, please either use a pre-made set or refer to the model preparation user guide.

3 High Level Idea

The cpp agent was build in order to be able to run any network that was compiled to JLFs\HEF. Moreover, it can also run inference on multiple devices simultaneously. By default, the program will generate a single frame, and push it a 100 times.

There is also a poor man's input API to direct the program to use specific set of JLFs (-j flag), or different number of images (-n flag).

¹It doesn't mean that other setups will not work, it was only tested with these versions

4 Detailed

The example is divided to 4 main parts:

1. Instantiation
2. Configuration
3. Inference (Dataflow)
4. Summary Report

4.1 Instantiation

The entry point in this example is the `main.cpp` file. Specifically, this example supports running inference on multiple devices. For this, we are running each agent on separate set of threads. In the snippet below, the agent is instantiated at line #57, and run at #11 (see below). the `run_inference` command (line #11) is a high-level wrapper, that will take care of the of all the device activities (Configuration, Inference, Report). For a single device case, one can choose to not instantiate the `example_device` in a new thread, it makes less sense doing so.

```

1 #include "example_device.hpp"
2 #include <iostream>
3 #include <thread>
4 #include <functional>
5
6 #define NOF_DEVICES (2)
7
8 void thread_wrapper(std::string &iface, std::string &jlf_dir, unsigned
   int &num_img)
9 {
10     example_device dev(iface, jlf_dir, num_img, 0, 0);
11     dev.run_inference();
12 }
13
14 int main(int argc, char **argv)
15 {
16     std::string jlf_dir = "./JLFs";
17     std::string iface = "eno2";
18     int opt;
19     unsigned int num_img = 100;
20     int debug = 0;
21     int write_log = 0;
22
23     while ((opt = getopt(argc, argv, "i:j:n:dl")) != -1)
24     {
25         switch (opt)
26         {
27             case 'j':
28                 jlf_dir = optarg;
29                 break;
30             case 'i':
31                 iface = optarg;

```

```

32         break;
33     case 'n':
34         num_img = atoi(optarg);
35         break;
36     case 'd':
37         debug = 1;
38         break;
39     case 'l':
40         write_log = 1;
41         break;
42     case '?':
43         fprintf(stderr, "Option -%c requires an argument.\n",
optopt);
44     default:
45         fprintf(stderr, "Usage: %s -i INTERFACE -j JLF-DIR [-n NUM-
IMAGES]\n\n", argv[0]);
46         fprintf(stderr, "        -i INTERFACE        The Ethernet
interface, defaults to 'eno2'\n");
47         fprintf(stderr, "        -j JLF-DIR          The JLFs directory,
defaults to './JLFs/'\n");
48         fprintf(stderr, "        -n NUM-IMAGES       The number of
images to process, defaults to 100\n");
49         fprintf(stderr, "        -d                  Read and print
debug registers from FW\n");
50         fprintf(stderr, "        -l                  Each receive thread
will write a log file\n");
51         exit(EXIT_FAILURE);
52     }
53 }
54 try
55 {
56     std::cout << "-I- TEST STARTS" << std::endl;
57     std::thread t0(thread_wrapper, std::ref(iface), std::ref(
jlf_dir), std::ref(num_img));
58     t0.join();
59     std::cout << "-I- TEST ENDED" << std::endl;
60 }
61 catch (const std::exception &e)
62 {
63     std::cout << e.what();
64 }
65 }

```

Listing 1: main.cpp

4.2 Configuration

The configuration part is built from these steps:

1. Scanning for Ethernet devices - `hailo_scan_ethernet_devices`
2. Creating a device - `hailo_create_ethernet_device`
3. Reading and loading the JLFs - `hailo_configure_device_from_jlf`
4. Activating the Data streams - `activate_[in|out]put_stream`

The first three parts are pretty much the same for each network, so we'll focus on the forth one - **Activating the Data streams**. By default, the `example_device::setup_device_for_inference` expects a type parameter - `T`, that is being used to derive the type of elements streaming into, and out of the device. The two options to use for `T` are:

1. `float32_t`
2. `uint8_t`

Once the streams are configured, a small banner is printed to identify the input and output stream names, and shapes. See the below example.

Figure 1: CenterNet-18-COCO Banner

```
-I-----
-I- Input[0]: input_layer (300, 300, 3)
-I- Output[1]: conv28 (80, 80, 2)
-I- Output[2]: conv29 (80, 80, 2)
-I- Output[3]: conv27 (80, 80, 80)
-I-----
```

4.3 Inference (Dataflow)

At this stage everything is ready for starting to stream images to the network. For that we use the `example_device::infer` function, which is type parametrized by `T`. The optional type are `float32_t` and `uint8_t`. This function will activate a new `std::thread` per input or output stream, which is also parametrized by `T`. The function will then run the threads, and wait for them to gracefully finish.

Note: The receive threads, implemented by the `example_device::_recv_thread` function do not do anything meaningful with the data.

4.4 Summary Report

If the test have finished gracefully, and no errors were emitted during the run, the `cpp_agent` will print the end of test banner. See Figure 2.

Figure 2: CenterNet-18-COCO End Banner

```

-I-----
-I- Total time:      0.0529536
-I- Average FPS:     37.7689
-I- Send data rate:  311.206 Mbit/s
-I- Recv[1] data rate: 14.75 Mbit/s
-I-----
-I- Recv[2] data rate: 14.75 Mbit/s
-I-----
-I- Recv[3] data rate: 590.14 Mbit/s
-I-----

```

5 Building the Project

This example project is built using cmake, but it is just an example, and other build systems should also work. For complete walk-through on cmake please refer to [CMake's website](#). In short, the configuration file for the project is called CMakeLists.txt and it holds all the information regarding file paths, external packages needed, etc. See below in Listing 3 the CMakeLists.txt of this project.

To build the project, execute the following commands, please note that the first command is only needed once.

```

1 CXX=/usr/bin/g++-9 cmake -H. -Bbuild
2 cmake --build build

```

Listing 2: Build project command

```

1 cmake_minimum_required(VERSION 3.0.0)
2 # Setting the ARCH if not provided
3 if (NOT DEFINED ARCH)
4     execute_process(COMMAND uname -m OUTPUT_VARIABLE arch_from_cmd)
5     string(STRIP ${arch_from_cmd} arch_from_cmd)
6     set(ARCH ${arch_from_cmd})
7 endif()
8
9 # Setting the CMAKE_C_COMPILER if not provided
10 if (${ARCH} STREQUAL "aarch64")
11     if (NOT DEFINED CMAKE_C_COMPILER)
12         set(CMAKE_C_COMPILER "/usr/bin/aarch64-linux-gnu-gcc")
13     endif()
14 else()
15     if (NOT DEFINED CMAKE_C_COMPILER)
16         message(STATUS "Setting compiler to g++-9")
17         set(CMAKE_C_COMPILER "/usr/bin/gcc-9")
18         set(CMAKE_CXX_COMPILER "/usr/bin/g++-9")
19     endif()
20 endif()
21
22 set(HAILORT_ROOT "/home/nadave/SDK/2.12.1/platform/hailort")

```

```
23
24 set(HAILORT_LIB "${HAILORT_ROOT}/lib/x86_64/libhailort.so.0.5.0")
25 set(HAILORT_INCLUDE_DIR "${HAILORT_ROOT}/include")
26 message(STATUS "Setting HailoRT to:" ${HAILORT_LIB})
27 set(COMPILER_OPTIONS_CPP -Werror -O3 -DNDEBUG )
28 include_directories(${HAILORT_INCLUDE_DIR} ".")
29
30 find_package(Threads)
31 find_package( OpenCV REQUIRED )
32 include_directories( ${OpenCV_INCLUDE_DIRS} )
33 foreach(target runme)
34     add_executable(${target} stream_and_info.cpp example_device.cpp
35         main.cpp)
36     target_compile_options(${target} PRIVATE ${COMPILER_OPTIONS_CPP})
37     target_link_libraries(${target} ${CMAKE_THREAD_LIBS_INIT})
38     target_link_libraries(${target} ${HAILORT_LIB})
39 endforeach(target)
```

Listing 3: CMakeLists.txt

Listings

1	main.cpp	3
2	Build project command	6
3	CMakeLists.txt	6

List of Figures

1	CenterNet-18-COCO Banner	5
2	CenterNet-18-COCO End Banner	6