QNN SDK

Download SDK : Qualcomm® Software Center

Prerequisites:

1. Create Raw file:

```
import numpy as np

x = np.random.randn(1,3,224,224).astype(np.float32)
# Write the raw data to the file
x.tofile("input.raw")
```

2. Create input list:

   input_list.txt Use this file for reference

3. Create docker file (Optional)

   Currently QNN SDK Supports ubuntu 22.04 version. If the cluster has different ubuntu version please make use of the below docker file

```
FROM ubuntu:22.04

# Update and upgrade the base image
RUN apt-get update -y && \
    apt-get upgrade -y && \
    apt-get install -y curl

# Install Python 3.10
RUN apt-get update && \
    apt-get install -y software-properties-common && \
    add-apt-repository ppa:deadsnakes/ppa && \
    apt-get update && \
    apt-get install -y python3.10 && \
    ln -s /usr/bin/python3.10 /usr/bin/python && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

```

```
# Install required system packages
RUN apt-get update && \
    apt-get install -y \
    python3-pip \
    python3-dev \
    libc++-dev \
    clang \
    libatomic1 && \
    apt-get clean && \
    rm -rf /var/lib/apt/lists/*

# Install ONNX-related tools
RUN pip3 install --no-cache-dir \
    onnx \
    onnxruntime \
    onnxsim

# Install additional Python packages
RUN pip3 install --no-cache-dir \
    numpy \
    pyyaml \
    pandas \
    packaging
```

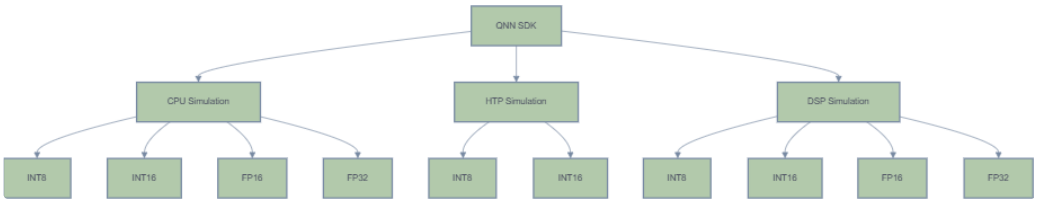
👉 **docker build :** docker build -t image_name:tag .

👉 **create docker container :** docker run -it -v local_dir_path:path_inside_container --name container_name image_name

👉 local_dir_path should contain the QNN SDK Folder that was downloaded earlier.

🛠️SDK Setup Commands: 🔗

```
export QNN_SDK_ROOT=/local/mnt/workspace/MCW_Workspace/QNN_SDK/qaisw-v2.17.0.231124161510_65373-auto/  - (change the directory to your peferred sdk path)
source $QNN_SDK_ROOT/bin/envsetup.sh
export PATH=/pkg/qct/software/llvm/build_tools/clang+llvm-8.0.0-x86_64-linux-gnu-ubuntu-14.04/bin:$PATH
export LD_LIBRARY_PATH=/pkg/qct/software/llvm/build_tools/clang+llvm-8.0.0-x86_64-linux-gnu-ubuntu-14.04/lib:$LD_LIBRARY_PATH
export CXX=clang++
echo $QNN_SDK_ROOT
export X86_CXX=clang++
export QNN_INCLUDE=${QNN_SDK_ROOT}/include/QNN
```



👀 Simulation

Commands used for simulation :

1. **qnn-onnx-converter :** Generates .cpp and .bin files

   `qnn-onnx-converter --input_network file_name.onnx --output_path file_name.cpp`

   Add the below flags for the respective datatypes

   👉 For FP 16 add `--float_bw 16`

   👉 For int 8 add `--input_list input.txt`

   👉 For int 16 add `--act_bw 16 --weight_bw 16 --input_list input.txt`

   Note : input.txt should contain the raw file path. (refer to prerequisites)

2. **qnn-model-lib-generator :** Generates .so file

   `qnn-model-lib-generator -c filename.cpp -b file_name.bin -o ./model_lib_out_fp32 -t x86_64-linux-clang`

3. **qnn-net-run :** Model's output is generated in a raw file

   `qnn-net-run --backend libQnnCpu.so --model model_lib_out_fp32/x86_64-linux-clang/libresnet18_fp32.so --input_list input.txt`

CPU Simulation(x86) :

```
# FP32
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_fp32.cpp
qnn-model-lib-generator -c resnet18_fp32.cpp -b resnet18_fp32.bin -o ./model_lib_out_fp32 -t x86_64-linux-clang
qnn-net-run --backend libQnnCpu.so --model model_lib_out_fp32/x86_64-linux-clang/libresnet18_fp32.so --input_list input.txt

# FP16
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_fp16.cpp --float_bw 16
qnn-model-lib-generator -c resnet18_fp16.cpp -b resnet18_fp16.bin -o ./model_lib_out_fp16 -t x86_64-linux-clang
qnn-net-run --backend libQnnCpu.so --model model_lib_out_fp16/x86_64-linux-clang/libresnet18_fp16.so --input_list input.txt
Error : qnn-net-run pid:2037
     0.4ms [ ERROR ] [QNN_CPU] OpConfig validation failed for Conv2d
[ ERROR ] QnnModel::addNode() adding node _conv1_Conv failed.
[ ERROR ] model.addNode(QNN_OPCONFIG_VERSION_1, "_conv1_Conv", "qti.aisw", "Conv2d", params__conv1_Conv, 4, inputs__conv1_Conv, 3, outputs__conv1_Conv, 1 ) expected MODEL_NO_ERROR, got MODEL_GRAPH_ERROR
[ ERROR ] addNode__conv1_Conv(resnet18_fp16) expected MODEL_NO_ERROR, got MODEL_GRAPH_ERROR
     2.4ms [ ERROR ] Failed in composeGraphs()
     2.4ms [ ERROR ] ComposeGraphs Failed with error = 1
Graph Prepare failure
HTP working

# int8
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_int8.cpp --input_list input.txt
qnn-model-lib-generator -c resnet18_int8.cpp -b resnet18_int8.bin -o model_lib_out_int8 -t x86_64-linux-clang
qnn-net-run --backend libQnnCpu.so --model model_lib_out_int8/x86_64-linux-clang/libresnet18_int8.so --input_list input.txt

# int16
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_int16.cpp --act_bw 16 --weight_bw 16 --input_list input.txt
```

```
qnn-model-lib-generator -c resnet18_int16.cpp -b resnet18_int16.bin -o model_lib_out_int16 -t x86_64-linux-clang
qnn-net-run --backend libQnnCpu.so --model model_lib_out_int16/x86_64-linux-clang/libresnet18_int16.so --input_list input.txt
HTP also not working
```

HTP Simulation(x86) :

Use the same commands as those for CPU Simulation. For the qnn-net-run, replace `libQnnCpu.so` with `libQnnHtp.so` The example given below is for int8. Change the command accordingly for int16.

```
# int8
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_int8.cpp --input_list input.txt
qnn-model-lib-generator -c resnet18_int8.cpp -b resnet18_int8.bin -o model_lib_out_int8 -t x86_64-linux-clang
qnn-net-run --backend libQnnHtp.so --model model_lib_out_int8/x86_64-linux-clang/libresnet18_int8.so --input_list input.txt
```

DSP Simulation(x86) :

The example given below is for int8. Change the command accordingly for int16, FP16 and FP32.

```
# int8
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_int8.cpp --input_list input.txt
qnn-model-lib-generator -c resnet18_int8.cpp -b resnet18_int8.bin -o model_lib_out_int8 -t x86_64-linux-clang
qnn-context-binary-generator --backend libQnnHtp.so --model model_lib_out_int8/x86_64-linux-clang/libresnet18_int8.so --binary_file model.serialized
```

Flashing SDK :

```
ftp 192.168.1.1
Name : root

cd /var/models/preethika/

put "/local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/bin/aarch64-qnx/qnn-net-run" qnn-net-run
put "/local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/bin/aarch64-qnx/qnn-profile-viewer" qnn-profile-viewer
put /local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/lib/aarch64-qnx/libQnnHtp.so libQnnHtp.so
put /local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/lib/aarch64-qnx/libQnnHtpV68tub.so libQnnHtpV68Stub.so
put "/local/mnt2/workspace2/stella/optrace_static_model/static_frozen_graph_log/static_frozen_graph_8Mb.serialized.bin" static_frozen_graph_8Mb.serialized.bin
put /local/mnt2/workspace2/stella/optrace_static_model/static_frozen_graph_log/input_new.txt input_list.txt
put /local/mnt2/workspace2/stella/optrace_static_model/static_frozen_graph_log/camera_front_main_image_0.raw camera_front_main_image_0.raw

cd /mnt/etc/images/dsp

put "/local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/lib/hexagon-v68/unsigned/libQnnHtpV68Skel.so" libQnnHtpV68Skel.so
put "/local/mnt2/workspace2/stella/qaisw-v2.24.0.240619123009_96268-auto/lib/aarch64-qnx/libQnnHtpNetRunExtensions.so" libQnnHtpNetRunExtensions.so
```

1. Two Targets were being used. **Makena** and **lemans**.
   For Makena use v68 (line no 09 and 16)

For Lemans use v73 (mostly used on the customer side)

2. Use ftp get to download files

   get remote_file_path local_file_path

In telnet :

```
telnet 192.168.1.1
root

# For int8

chmod 777 ./qnn-net-run
./qnn-net-run --version
./qnn-net-run --backend libQnnHtp.so --input_list input.txt --retrieve_context ser.bin --output_dir output_preethika
chmod 777 ./qnn-profile-viewer
./qnn-profile-viewer --input_log ./output_preethika/qnn-profiling-data_0.log
```

🔣 Calculate MSE :

1. Get the model output from ONNX Runtime
2. Obtain the model output from simulation (a raw file will be generated by qnn-net-run; just read the raw file)
3. Calculate the MSE between the two model outputs. The Resulting MSE Value should be 0

```
import onnxruntime as ort
import numpy as np

model_path = "resnet18.onnx"
sess = ort.InferenceSession(model_path)
input_name = sess.get_inputs()[0].name
input_data = np.fromfile("input.raw",np.float32).reshape(1, 3, 224, 224)
outputs = sess.run(None, {input_name: input_data})  # None returns all outputs
model_output = outputs[0]

#read the raw file
path = "output_int8/Result_0/_171.raw"
x = np.fromfile(path,np.float32).reshape(1,1000)

#Calculate MSE
MSE = np.square(np.subtract(model_output,x)).mean()
print("MSE",MSE)
```

📤 Generate Optrace:

```
# x86
qnn-onnx-converter --input_network resnet18.onnx --output_path resnet18_int8.cpp --input_list input.txt --debug
qnn-model-lib-generator -c resnet18_int8.cpp -b resnet18_int8.bin -o model_lib_out_int8 -t x86_64-linux-clang
qnn-context-binary-generator --backend libQnnHtp.so --model "/local/mnt/workspace/stella/developer/model_lib_out_int8/x86_64-linux-clang/libresnet18_int8.so" --binary_file model.serialized --profiling_option optrace --profiling_level detailed
# In telnet Terminal
./qnn-net-run --backend libQnnHtp.so --input_list input.txt --retrieve_context model.serialized.bin --output_dir output --profiling_option optrace --profiling_level detailed
# copy qnn-profiling-data.log to x86 using ftp get
```

```
get qnn-profiling-data.log telnet_path x86_path
# x86
qnn-profile-viewer --reader $QNN_SDK_ROOT/lib/x86_64-linux-clang/libQnnHtpOptraceProfilingReader.so --
input_log qnn-profiling-data.log --schematic
"/local/mnt/workspace/stella/developer/resnet18_int8_python_out.py" --output <chrometrace.json>
```

step 1 :

cd /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO (your working directory)

copy "/local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/QnnHtpDebug.conf" to your working directory QnnHtpDebug.conf

Step 2:

qnn-onnx-converter -i /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/qnn-v2.27/qnn-v2.27-release/model_add_log/model_add.onnx --input_list /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/qnn-v2.27/qnn-v2.27-release/model_add_log/input_list.txt --debug

Step 3:

qnn-model-lib-generator -c /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/model_add.cpp -b //local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/model_add.bin -o /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/model_add.cpp -l model_add -t x86_64-linux-clang

Step 4:

qnn-context-binary-generator --model /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/x86_64-linux-clang/libmodel_add.so --backend libQnnHtp.so --output_dir /local/mnt/workspace/stella/SCATTER_ADD_OPTRACE3/DEMO/ --profiling_level detailed --profiling_option optrace --binary_file model_add_8Mb.serialized --config_file /local/mnt/workspace/stella/qnn_automation_scripts/qnn_automation_scripts/config/HtpConfigFile8Mb.json

HtpConfigFile8Mb.json PerfSetting8Mb.conf

Step 5:

qnn-net-run --backend libQnnHtp.so --input_list <list.txt> --retrieve_context <ser_bin> --profiling_level detailed --profiling_option optrace

Step 6:

IMPORTANT: Copy the log file to your local x86

qnn-profile-viewer --reader $QNN_SDK_ROOT/lib/x86_64-linux-clang/libQnnHtpOptraceProfilingReader.so --input_log qnn-profiling-data.log --schematic "dir to pythonout.py" --output <chrometrace.json>