SIEMENS EDA

HLS4ML Flow in Catapult

SIEMENS

Support Center: support.sw.siemens.com
Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Contents

Licensing Requirement

Please note that a new license option is required to run the Catapult HLS4ML flow. Because this is a newly released option, most existing customers will not have a license for it.

If the flow fails with a licensing error, please ensure that you have the required AI license. Your Siemens EDA Account Manager will be able to assist you with evaluation or purchase.

Introduction

The intent of this document (and the accompanying example designs) is to introduce the HLS4ML Flow in Catapult. The HLS4ML Flow is the first dedicated Machine Learning design flow offered as part of the new Catapult.ai NN product line. This flow integrates hls4ml – an open-source Python package for designing neural network inferencing engines in HLS C++ – with enhanced Catapult features like power-optimized RTL, detailed per-layer PPA reporting and a full High-Level Verification (HLV) ecosystem for pre- and post- HLS design verification.

This document will give a brief introduction to a class of neural network designs (Convolutional Neural Networks), the hls4ml open-source package for generating C++ from a Python model and the HLS4ML Flow in Catapult for driving the entire design process from Python to RTL/gates.

Background

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are often used for object detection and classification. As such, they typically operate on image data (a 2-D image). A CNN can have many layers – an example of a object classifier is shown here:

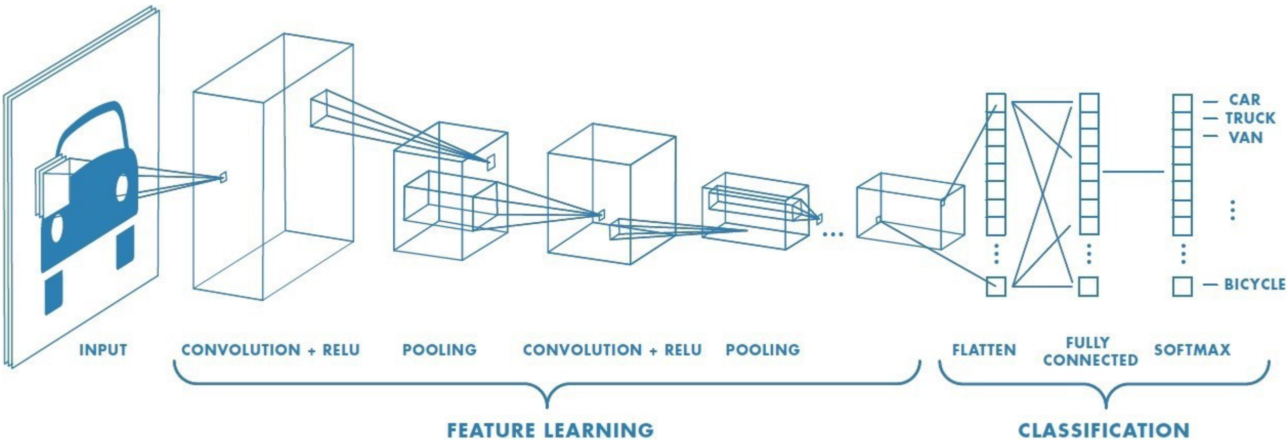


Figure 1 Object Classification CNN

A CNN is usually modeled in an ML framework like TensorFlow. In that framework, the model can be trained using large datasets. This training is usually done on CPU, GPU or on a cloud compute farm. The result is a neural network model with trained weight/bias values that can then be executed as in Inferencing engine. It is this inferencing model that users want to put into hardware – specifically hardware at the edge next to the sensor. It is this hardware that must meet the performance requirements of the application (latency to object classification) as well as the area and power requirements (small mobile applications). This is where Catapult HLS comes into the methodology.

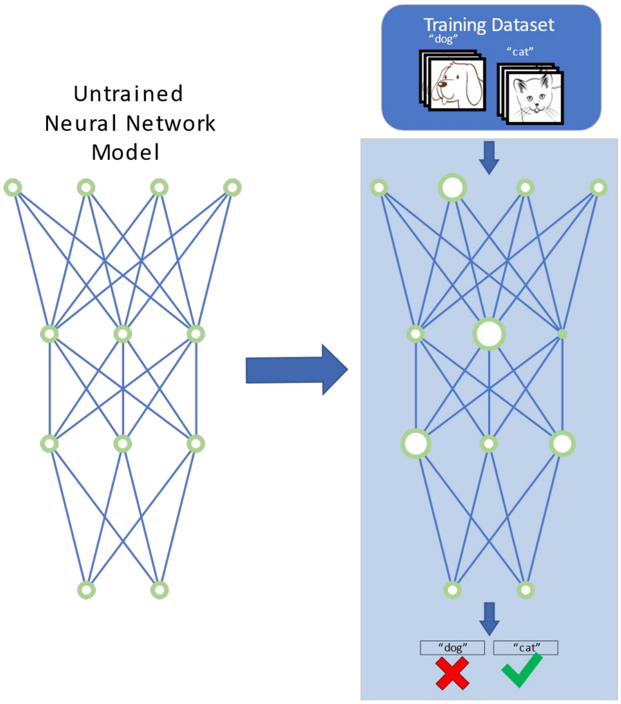


Figure 2 CNN Model Training

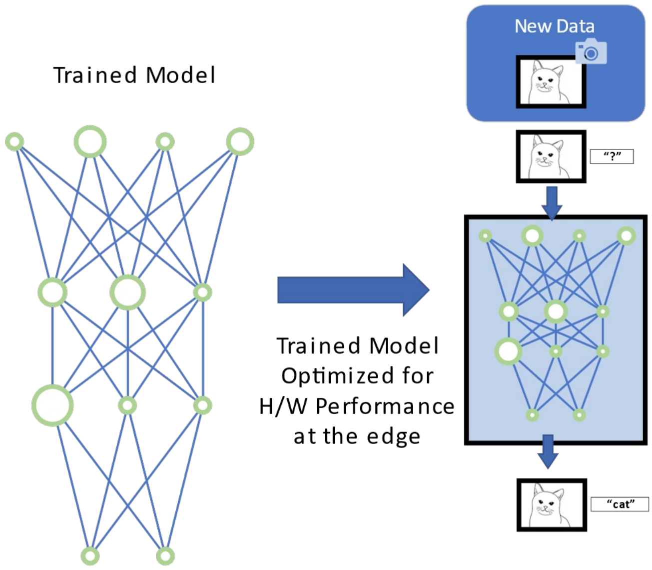


Figure 3 Inference Engine H/W Implementation

It is beyond the scope of this toolkit example to explain all of the aspects of Convolutional Neural Nets and machine learning in general. A good introductory tutorial on CNNs can be found here: https://cs231n.github.io/convolutional-networks/

HLS4ML

HLS4ML is an open-source package that consists of Python code that manages the integrations of TensorFlow, Keras, QKera, Onyx, PyTorch, etc as well as the code that generates the C++ for HLS output code and scripts for various HLS tools including Catapult. The following is the basic workflow for moving from a NN model in Python through HLS4ML and Catapult to generate H/W RTL.

Neural Network Model

The starting point is a NN model defined in one of the common ML frameworks like TensorFlow, PyTorch, Keras or ONNX. For example, the following is a simple QKeras network that performs conv2d:

```
model = tf.keras.Sequential()
model.add(QConv2DBatchnorm(filters=8, kernel_size=3,
                  padding='same', strides=2,
                  input_shape=(28,28,1),
                  kernel_quantizer=quantizers.quantized_bits(bits=8, integer=0),
                  bias_quantizer=quantizers.quantized_bits(bits=8, integer=0)))
```

A visualization of the network is:

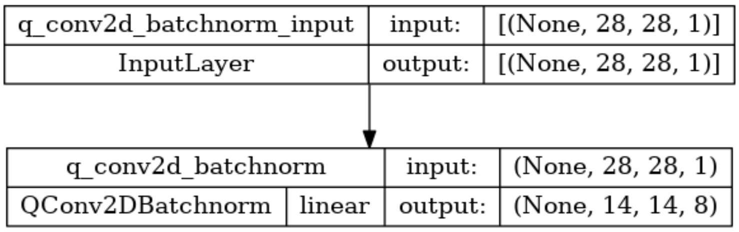


Figure 4 Network Graph using plot_model

From this point you can use the Python environment to load datasets, perform training and measure the accuracy of the trained network. Once you have a trained network model you can configure HLS4ML to generate the C++ implementation. HLS4ML has configuration options that point to the model, the testbench data (input feature maps and predicted outputs) and the target hardware requirements. Below is a portion of the configuration settings:

```
…
config_ccs = {}
config_ccs['Backend'] = 'Catapult'
config_ccs['ClockPeriod'] = 10
config_ccs['HLSConfig'] = {'Model': {'Precision': 'ac_fixed<16,8>', 'ReuseFactor':
args.reuse_factor}}
config_ccs['IOType'] = 'io_stream'
config_ccs['KerasH5'] = model_name+'_weights.h5'
config_ccs['KerasJson'] = model_name+'.json'
config_ccs['ProjectName'] = proj_name
config_ccs['ASICLibs'] = 'saed32rvt_tt0p78v125c_beh'
config_ccs['FIFO'] = 'hls4ml_lib.mgc_pipe_mem'
config_ccs['OutputDir'] = out_dir
config_ccs['InputData'] = 'tb_input_features.dat'
config_ccs['OutputPredictions'] = 'tb_output_predictions.dat'
…
hls_model_ccs = hls4ml.converters.keras_to_hls(config_ccs)
…
hls_model_ccs.build(csim=True, synth=True, cosim=False, validation=True, vsynth=False,
sw_opt=True)
```

From these setting you can see that the ASIC "saed32" technology is the target, with a fixed point precision (16,8), streaming input/output, 10ns clock period and the model is read from "model_name.json".
After the configuration dictionary is created, the HLS4ML model can be constructed (keras_to_hls).
Finally, the HLS4ML build() method is called to write out the C++ code, Catapult TCL script and then invoke Catapult in the background to run HLS. Details of the various configuration options and the build() method arguments are shown in the section Config Options.

For a tutorial on the basics of HLS4ML (targeting FPGA) you can view this tutorial: https://www.youtube.com/watch?v=FFUyRQukGvM

Python Environment

The HLS4ML package requires a properly configured Python environment in order to execute. Providing such an environment inside of the Catapult install tree is currently beyond the scope of Catapult HLS. To facilitate having a proper environment, Catapult provides a script that will create a new Python Virtual Environment (VENV) and populate it with the Python packages requires by HLS4ML. This script is automatically launched by Catapult when you attempt to use HLS4ML from within a Catapult session. The VENV will be created in the user's home directory under ~/ccs_venv. There exists an option in the HLS4ML Flow integration in Catapult that allows you to specify an alternate location:

```
flow package require /HLS4ML
flow package option set /HLS4ML/VENV <path to location for venv>
```

The Python packages installed include the following:

tensorflow	numpy	matplotlib	Scikit-learn
pandas	pytest	Pytest-cov	pydot
graphviz	jupyter	seaborn	sympy
calmjs	tabulate	qkeras	

Note that if you create your own VENV and it happens to contain HLS4ML, running HLS4ML from within a Catapult session will pick up the HLS4ML inside Catapult and not the one in your VENV.

HLS4ML Flow

The HLS4ML Flow integration in Catapult provides the framework to launch Python to execute HLS4ML to generate the C++. The flow is enabled using:

```
flow package require /HLS4ML
```

The Flow has minimal configuration settings:

- PYTHON – the pathname to the Python3 executable. By default, Catapult will use the Python3 shipped with Catapult
- VENV – the pathname to the Python Virtual Environment (VENV). The default is the user's ~/ccs_venv. If the VENV does not exist when the HLS4ML Flow attempt to launch Python3 the Flow will construct the VENV at the specified directory.

Since HLS4ML is primarily a "batch" mode environment (Python -> Gen C++ -> Launch HLS batch) the general use model is to have the Python NN model file executed using the command:

```
flow run /HLS4ML/gen_hls4ml <path_to_model_Python>
```

The expectation is that the Python model file will properly configure HLS4ML so that it generates the C++ design and the Catapult 'build_prj.tcl' command file. This means that the Python must call the HLS4ML build() method properly.
As an example, you can run the following example:

```
flow package require /HLS4ML
# Export the HLS4ML "simple" example to the directory somedir
project export -toolkit "Examples/HLS4ML/simple" somedir
# Move into somedir
cd somedir
# Run HLS4ML to generate the C++
flow run /HLS4ML/gen_hls4ml model_asic.py
```

The last part of the transcript should show:

```
#  ==========================================================================================
#  Compiling HLS C++ model
#  Writing HLS project
#  Copying NNET files to local firmware directory
#  Done
#  ==========================================================================================
#  Skipping HLS
#  - To run Catapult directly from the shell:
#      cd my-Catapult-test_asic1; catapult -file build_prj.tcl
#  - To run directly in the current Catapult session:
#      set_working_dir my-Catapult-test_asic1
#      dofile build_prj.tcl
#  0

solution.v1{10}>
```

After performing the set_working_dir / dofile commands as instructed, the result will be a synthesized design.

If you want to use the Catapult HLS4ML environment from a Python shell, you can do the following:

```
bash
export PYTHONPATH=$MGC_HOME/shared/pkgs/ccs_hls4ml/hls4ml
source $HOME/ccs_venv/bin/activate
python3
```

Config Options

The Catapult.ai NN model generation options include the following:

Option	Scope	Values	Description
IOType	Top configuration	**io_parallel** or **io_stream**	The I/O type for passing feature/weight data. Parallel uses C-Arrays, Stream uses ac_channels.
Strategy	Top configuration	**latency** or **resource**	Determines whether the code generation favors lower latency or lower area
ReuseFactor	Top configuration or per-layer	<integer>	Determines the level of parallelism. 1 is the most parallel (lowest latency), 2 is half that...
Precision	Top configuration or per-layer	<ac_fixed type>	Determines the data type precision for features, weights, biases
ClockPeriod	Top configuration	<integer>	Specifies the clock period for HLS and RTL synthesis in ns
Technology	Top configuration	**asic** or **fpga**	Determines the target technology
ASICLibs	Top configuration	<Catapult library name>	Determines the base technology library to load (asic mode only)
Part / XilinxPart	Top configuration	<Xilinx Part>	Determines the AMD/Xilinx Part to load (fpga mode only)
FIFO	Top configuration	<Catapult FIFO lib.mod	Determines the FIFO interconnect component to

		name>	use between layers

The build() method allows the following options for launching Catapult in batch mode:

Option	Values	Description
csim	**True** or **False**	Enables running SCVerify on the C++ right after 'go compile'
synth	**True** or **False**	If True, runs Catapult through 'go extract' or 'go power'. If False, stops HLS at 'go assembly'
cosim	**True** or **False**	Enables running SCVerify after RTL generation to validate the RTL and C++ behavior match
vhdl / **verilog**	**True** or **False**	Enables the various netlist outputs
ran_frame	<integer>	If no training datasets provided, use N random data frames for SCVerify
sw_opt	**True** or **False**	Enables performing RTL power estimation
power	**True** or **False**	Enabled performing RTL power optimization
bup	**True** or **False**	Enables building the project in a bottom-up fashion
da	**True** or **False**	Enabled invoking Design Analyzer while running HLS

Running the Examples

There are five HLS4ML based example designs shipped with Catapult:

- Simple – just a simple conv2d network
- MNIST – the MNIST model implemented using conv2d layers
- MNIST_Dense – the MNIST model implemented with dense layers
- SeperableConv2D – an example of the SeperableConv2D layer

To run any of the examples, from the Start Page in the Catapult GUI, click on "Examples" and browse the navigation tree on the left to locate the HLS4ML folder. Expand that out and select one of the five examples. There are multiple variations for each example (that highlight features like reuse_factor and asic vs fpga). Select the specific example script to run and click "Launch Project".

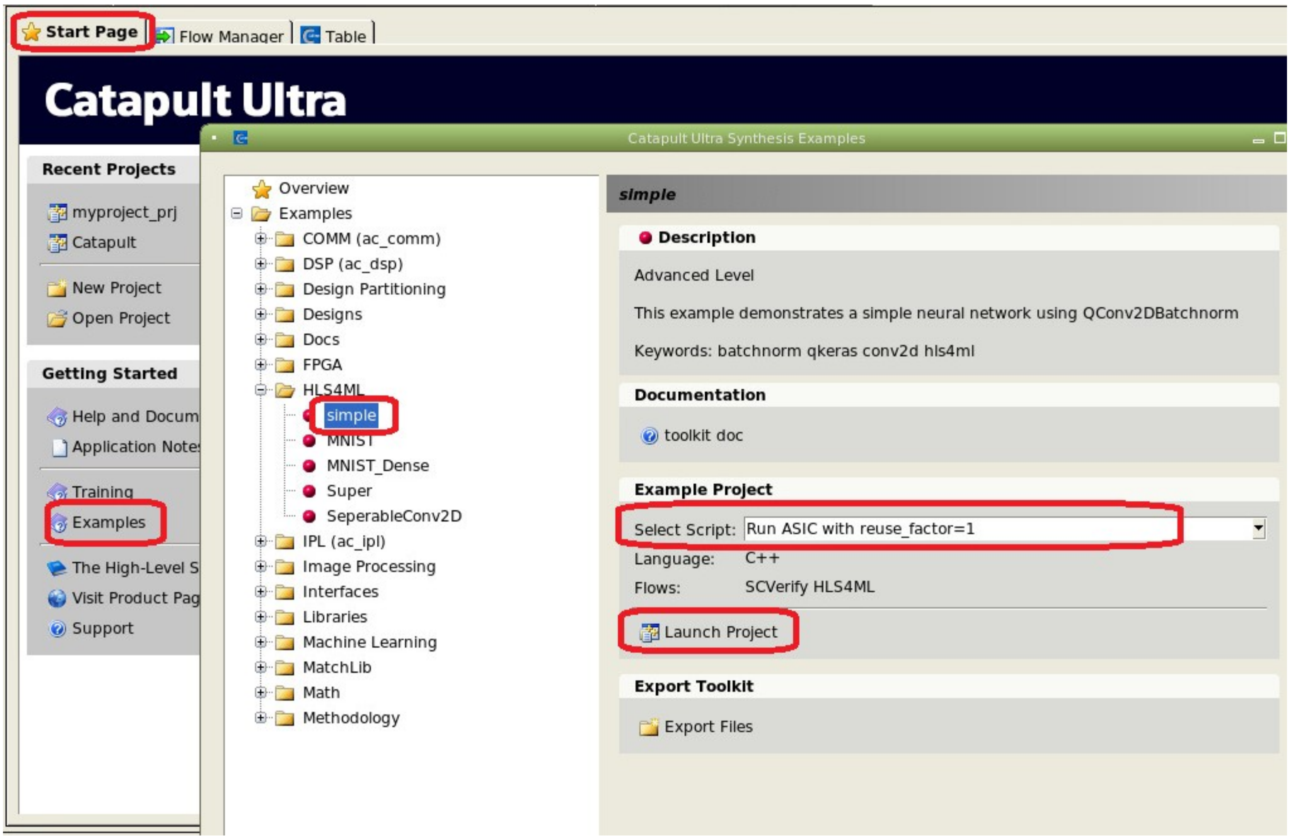


Figure 5 Launching a HLS4ML Toolkit Example

After the script has finished running the transcript show the results per layer:

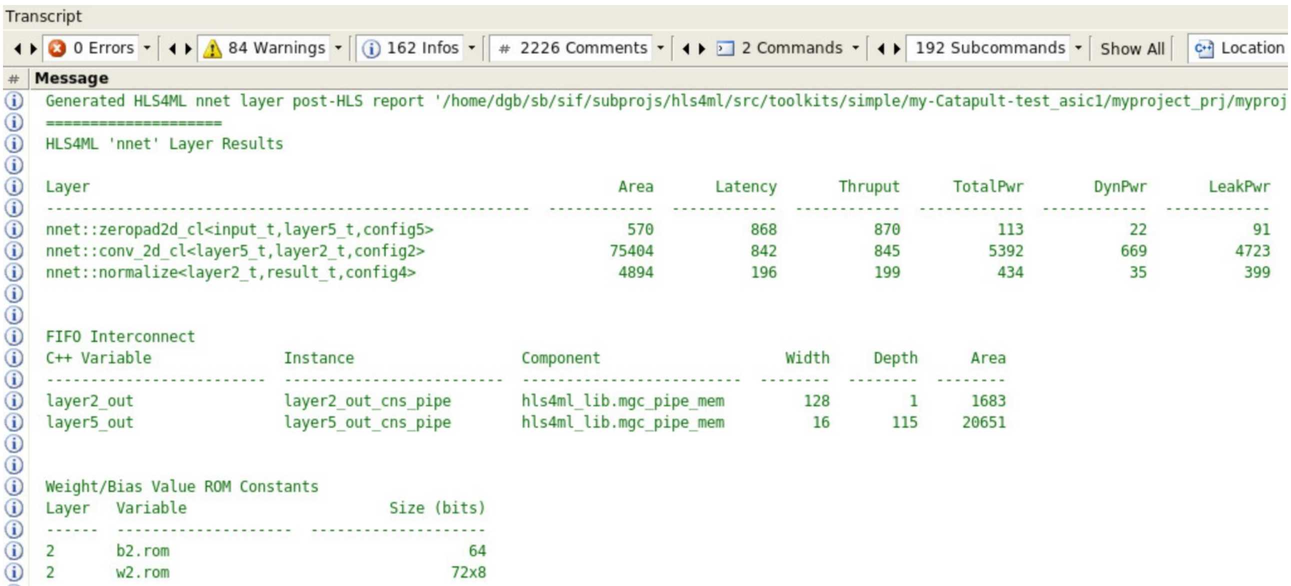


Figure 6 Transcript after HLS4ML Synthesis

Some of the examples leverage the Power Analysis features built into Catapult Ultra. Those examples will show power numbers in the report.

The Catapult Project Files pane will also contain the final reports from the HLS4ML flow:

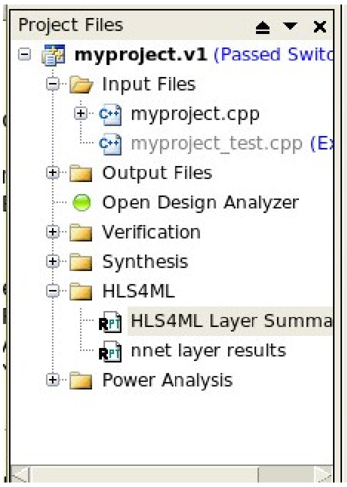
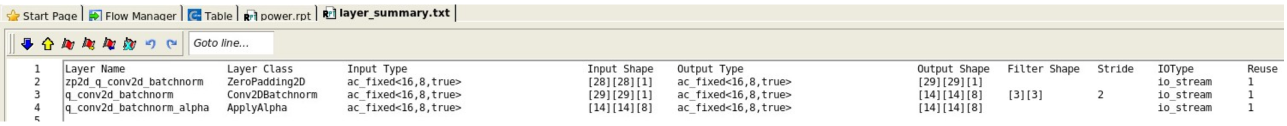


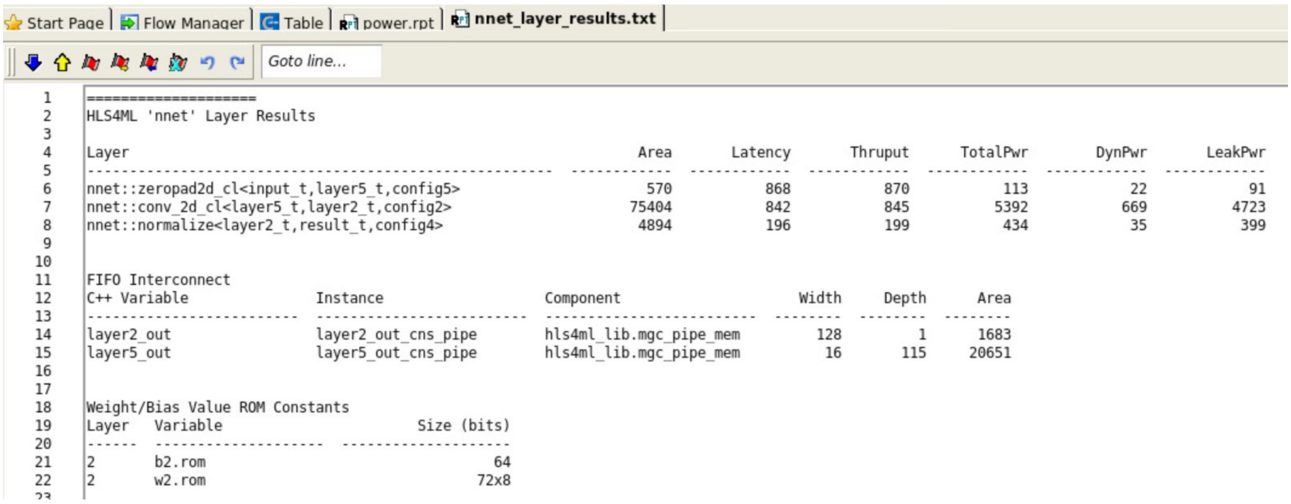
Figure 7 HLS4ML Flow Reports

The "HLS4ML Layer Summary" report shows the Python description of each layer:



```
Layer Name                 Layer Class        Input Type             Input Shape     Output Type            Output Shape    Filter Shape  Stride  IOType     Reuse
zp2d_q_conv2d_batchnorm    ZeroPadding2D      ac_fixed<16,8,true>    [28][28][1]     ac_fixed<16,8,true>    [29][29][1]                           io_stream  1
q_conv2d_batchnorm         Conv2DBatchnorm    ac_fixed<16,8,true>    [29][29][1]     ac_fixed<16,8,true>    [14][14][8]     [3][3]        2       io_stream  1
q_conv2d_batchnorm_alpha   ApplyAlpha         ac_fixed<16,8,true>    [14][14][8]     ac_fixed<16,8,true>    [14][14][8]                           io_stream  1
```

While the "nnet_layer results" report shows the PPA per layer:



```
====================
HLS4ML 'nnet' Layer Results

Layer                                                        Area      Latency      Thruput     TotalPwr       DynPwr      LeakPwr
------------------------------------------------------  ----------  -----------  -----------  -----------  -----------  -----------
nnet::zeropad2d_cl<input_t,layer5_t,config5>                   570          868          870          113           22           91
nnet::conv_2d_cl<layer5_t,layer2_t,config2>                  75404          842          845         5392          669         4723
nnet::normalize<layer2_t,result_t,config4>                    4894          196          199          434           35          399


FIFO Interconnect
C++ Variable               Instance                   Component                   Width     Depth      Area
-------------------------  -------------------------  -------------------------  -------  --------  --------
layer2_out                 layer2_out_cns_pipe        hls4ml_lib.mgc_pipe_mem        128         1      1683
layer5_out                 layer5_out_cns_pipe        hls4ml_lib.mgc_pipe_mem         16       115     20651


Weight/Bias Value ROM Constants
Layer   Variable                 Size (bits)
------  -------------------  -------------------
2       b2.rom                                64
2       w2.rom                              72x8
```

Figure 8 HLS Synthesis PPA Results

Running Jupyter Notebooks

You can run Catapult.ai NN from within a Juypter notebook. First, make sure your Python Virtual Environment has been created:

```
sh $MGC_HOME/shared/pkgs/ccs_hls4ml/create_env.sh $MGC_HOME/bin/python3 $HOME/ccs_venv
```

where $HOME/ccs_venv is the path where you want the virtual environment placed.

Then start a bash shell followed by the Jupyter notebook (make sure you are in the directory containing the notebook file(s) *.ipynb):

```
bash
export PYTHONPATH=$MGC_HOME/shared/pkgs/ccs_hls4ml/hls4ml
jupyter notebook --ip="127.0.0.1" --browser=firefox
```

This should start a notebook server and open a Firefox web browser on the directory.

Note: If the browser does not show any file contents, it is probably a browser cache issue – just hit Ctrl-Shift-R to clear it.

Once the browser shows the file tree, double-click on the notebook file:

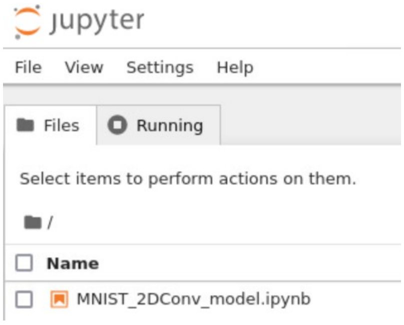


Figure 9 Jupyter Notebook file tree

Once the notebook is loaded use "Play" button to step through each cell of the notebook:

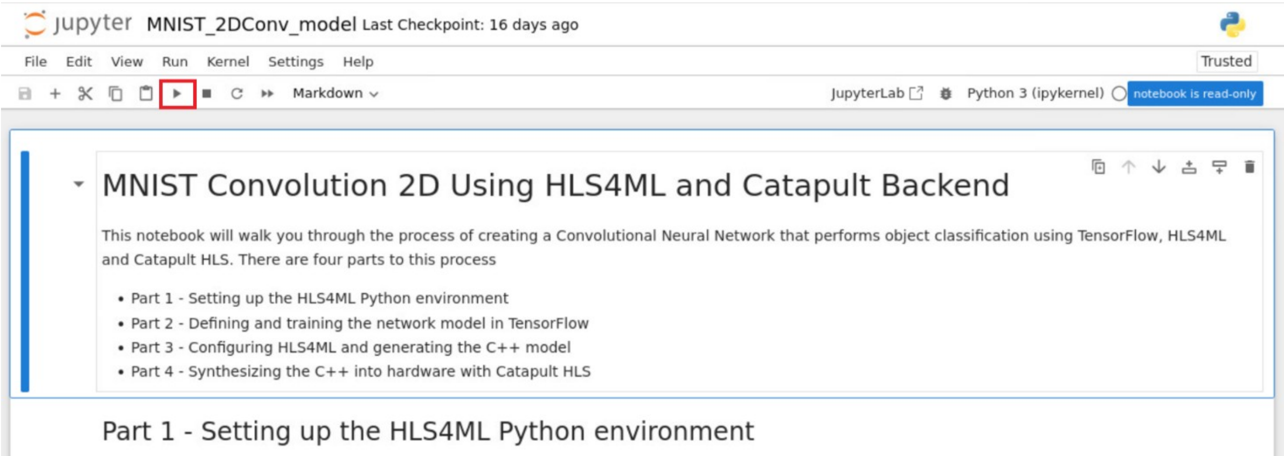


Figure 10 Playing a notebook

If you need to restart the python kernel, use the Kernel menu pick:

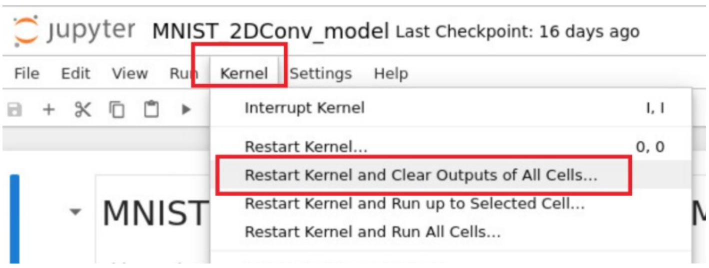


Figure 11 Resetting the Kernel and Notebook