How to create hand-written design:

Once you have gone through the HLS4ML Tutorial and have a working Vivado and HLS4ML setup, it is now time to start comparing an HLS4ML benchmark to hand-written SystemVerilog code. This document will take you through the initial steps of how to get the HLS4ML project into Vivado and how to compare it to your hand-written design.

1. Create HLS design of benchmark
    a. Either train the benchmark or load in the model from a .json and .h5 file. If you are training the model directly, follow the examples in the HLS4ML tutorial. If you have the trained weights you can use this to create a Keras model directly and import the weights.
    Googling how to create a Keras model based on the type of file you have can be very helpful here!!
    b. You want to be able to run the following commands to understand the structure for your model. Example:

```
In [2]: from tensorflow.keras.models import Sequential, model_from_json

In [3]: json_file = open('encoder_8x8_c8_S2_tele.json', 'r')
        loaded_model_json = json_file.read()

In [4]: loaded_model = model_from_json(loaded_model_json)

        loaded_model.load_weights('encoder_8x8_c8_S2_tele.h5')
        loaded_model.get_weights()
        loaded_model.summary()

2023-10-11 17:51:43.072633: I tensorflow/compiler/jit/xla_cpu_device.cc:41] Not creating XLA devices, tf_xla_enable_
xla_devices not set
2023-10-11 17:51:43.092476: W tensorflow/stream_executor/platform/default/dso_loader.cc:60] Could not load dynamic l
ibrary 'libcuda.so.1'; dlerror: libcuda.so.1: cannot open shared object file: No such file or directory
2023-10-11 17:51:43.092508: W tensorflow/stream_executor/cuda/cuda_driver.cc:326] failed call to cuInit: UNKNOWN ERR
OR (303)
2023-10-11 17:51:43.092536: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] kernel driver does not appear
to be running on this host (ubuntu): /proc/driver/nvidia/version does not exist
2023-10-11 17:51:43.100865: I tensorflow/compiler/jit/xla_gpu_device.cc:99] Not creating XLA devices, tf_xla_enable_
xla_devices not set

Model: "encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 8, 8, 1)]         0
_________________________________________________________________
conv2d (Conv2D)              (None, 4, 4, 8)           80
_________________________________________________________________
flatten (Flatten)            (None, 128)               0
_________________________________________________________________
encoded_vector (Dense)       (None, 16)                2064
=================================================================
Total params: 2,144
Trainable params: 2,144
Non-trainable params: 0
_________________________________________________________________
```

    Now I know the layers of my model and have a working Keras model!
    c. Following the flow shown in the HLS4ML tutorial, create the proper HLS config file.s
        i. Do you have a conv2d layer? If so, ‘io_type’ needs to be set to ‘io_stream’.
        ii. Should your strategy be ‘resource’ or ‘latency’? Most likely ‘resource’.
        iii. What do you want your reuse factor to be?

iv. Example config:

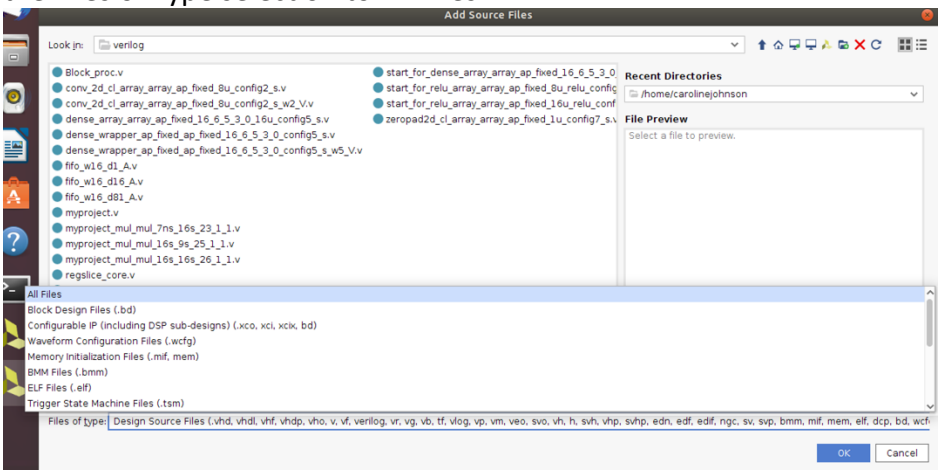
```
In [7]: cfg

Out[7]: {'OutputDir': 'basic',
         'ProjectName': 'myproject',
         'Backend': 'Vivado',
         'XilinxPart': 'xcu250-figd2104-2L-e',
         'Board': None,
         'ClockPeriod': 5,
         'IOType': 'io_stream',
         'HLSConfig': {'Model': {'Precision': 'ap_fixed<16,6>',
           'ReuseFactor': 1,
           'Strategy': 'Latency'},
          'LayerName': {'input_1': {'Precision': {'result': 'ap_fixed<16,6>'},
            'Strategy': 'Resource'},
           'conv2d': {'Precision': {'weight': 'ap_fixed<16,6>',
             'bias': 'ap_fixed<16,6>',
             'result': 'ap_fixed<16,6>'},
            'ReuseFactor': 1,
            'Strategy': 'Resource'},
           'conv2d_relu': {'Precision': 'ap_fixed<16,6>',
            'ReuseFactor': 1,
            'table_size': 1024,
            'table_t': 'ap_fixed<18,8>',
            'Strategy': 'Resource'},
           'encoded_vector': {'Precision': {'weight': 'ap_fixed<16,6>',
             'bias': 'ap_fixed<16,6>',
             'result': 'ap_fixed<16,6>'},
            'ReuseFactor': 1,
            'Strategy': 'Resource'},
           'encoded_vector_relu': {'Precision': 'ap_fixed<16,6>',
            'ReuseFactor': 1,
            'table_size': 1024,
            'table_t': 'ap_fixed<18,8>',
            'Strategy': 'Resource'}}},
         'KerasModel': <tensorflow.python.keras.engine.functional.Functional at 0x7f54d82c05e0>,
         'Stamp': '0eaBb151'}
```

d. Run hls_model.build()
   i. If it fails: check your config!!! You are most likely running out of resources. Try increasing the reuse factor per layer. Look into other similar models done in the HLS4ML tutorial and see if one of your configuration parameters is different from them.
   ii. If you do not get an output of estimated resources and timing, it did not finish running. Do not trust these results but this tells you it finished.

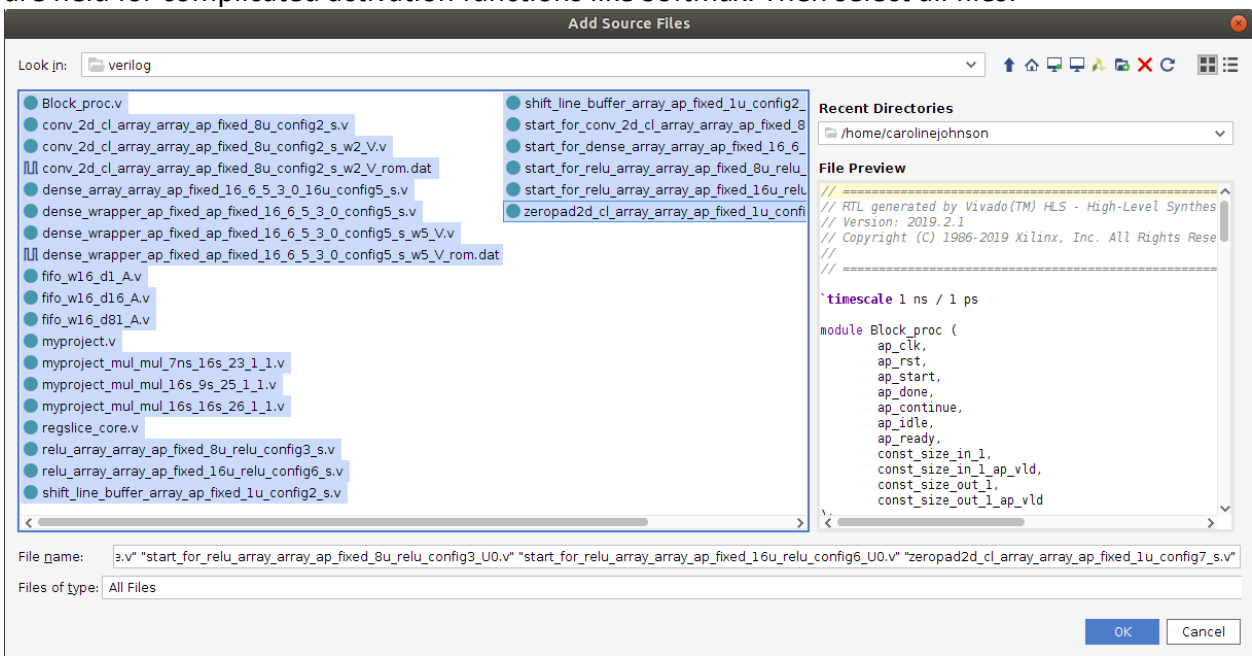
```
In [8]: hls_model.build(csim=False)
        INFO: [VLOG 209-307] Generating Verilog RTL for myproject.
        ***** C/RTL SYNTHESIS COMPLETED IN 0h5m8s *****
        INFO: [HLS 200-112] Total elapsed time: 314.69 seconds; peak allocated memory: 537.193 MB.
        INFO: [Common 17-206] Exiting vivado_hls at Wed Oct 11 17:57:40 2023...

Out[8]: {'EstimatedClockPeriod': '4.299',
         'BestLatency': '568',
         'WorstLatency': '569',
         'IntervalMin': '163',
         'IntervalMax': '568',
         'BRAM_18K': '467',
         'DSP48E': '1048',
         'FF': '21470',
         'LUT': '26768',
         'URAM': '0',
         'AvailableBRAM_18K': '5376',
         'AvailableDSP48E': '12288',
         'AvailableFF': '3456000',
         'AvailableLUT': '1728000',
         'AvailableURAM': '1280'}
```

2. Understand the layers produced by the benchmark
   a. From the layers we saw with the loaded_model.summary() command, understand the computations done in each layer. Create a flow chart and understand how many filters each layer has and what is being done in each.
3. Create a Vivado project of HLS produced Verilog files and write an example testbench.

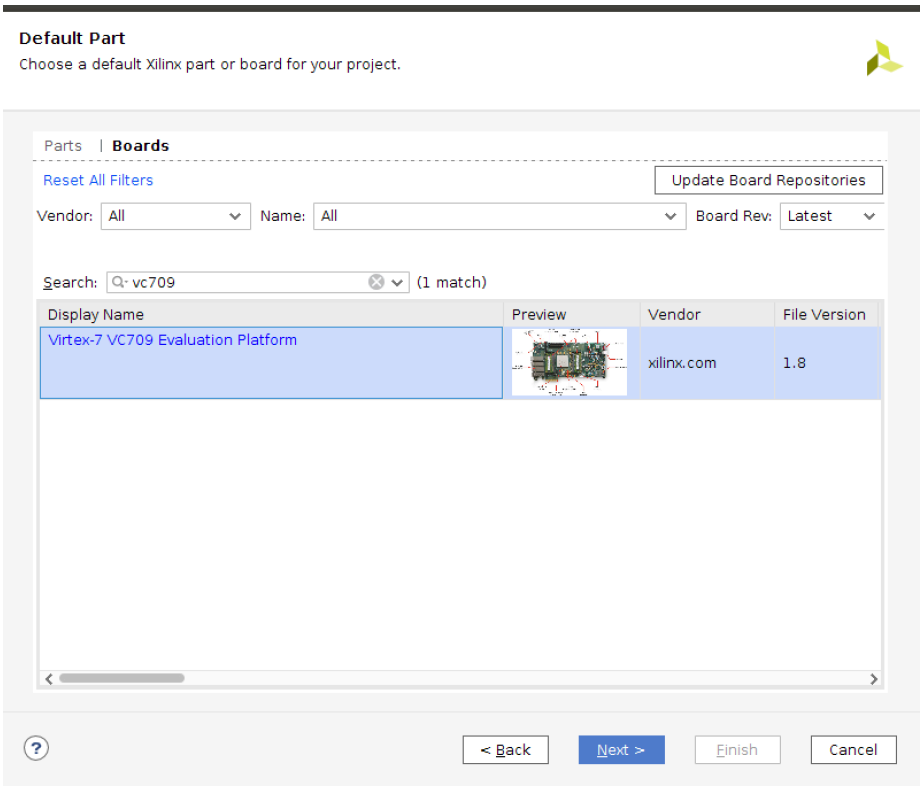
a. Open Vivado → Create Project → Pick a directory for it → RTL Project → Hit Add Files, go to the directory where your HLS model was written to, mine was called “basic”.
So go to “/basic/myproject_prj/solution1/impl/verilog”. These are all the produced Verilog files for the benchmark. BEFORE SELECTING THE FILES, change the Files of Type selection to All Files



Without this, you will not include any .dat files which are how the lookup tables are held for complicated activation functions like Softmax. Then select all files:



And hit ok → Next
Write clock constraint file, not required but you will want it eventually. You can create a file name “top.xdc” and fill it in later. It should contain a command like the following:
create_clock -period 4.4 -name ap_clk -add [get_ports ap_clk]
This sets the clock for synthesis and implementation.
Hit next, choose VC709 for the board.



Done!

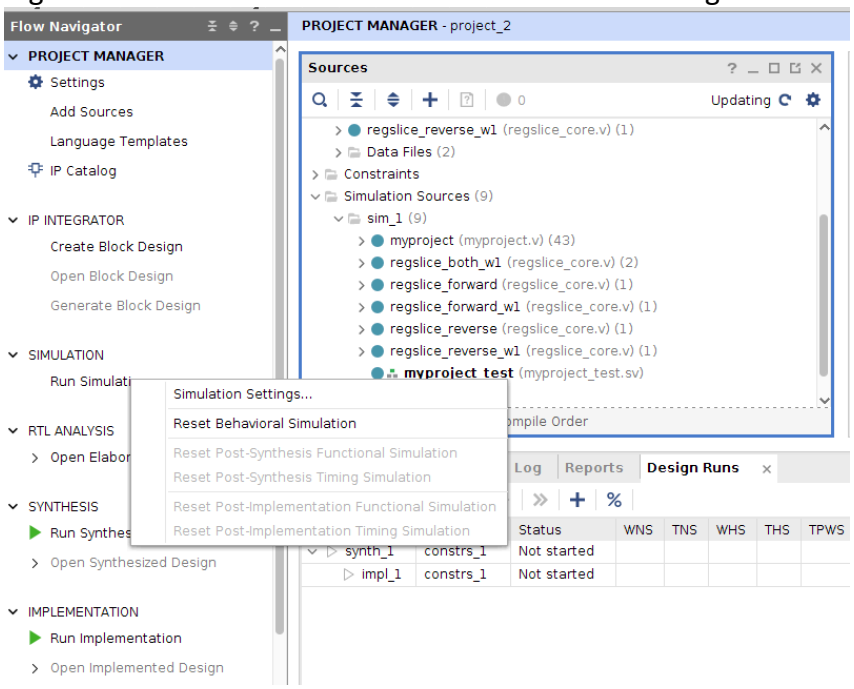
b. Make sure myproject.v (or you may have changed this from the default name) is selected as the top level file.

c. Now we need to create a testbech to test the code. Go to File → Add Sources → Add or create simulation sources → Create File → Name it what you want, for example “myproject_test.sv” (Also need to change File Type to SystemVerilog from default Verilog) → Ok → Finish → Ok → Yes. You do not need to change the Module definitions.

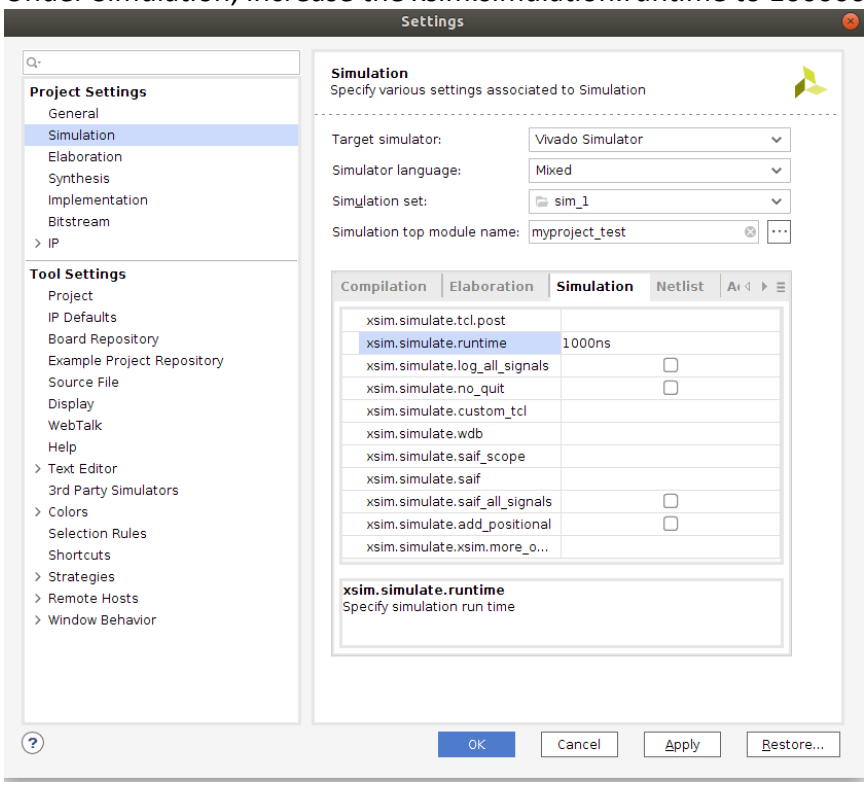
d. Create a basic testbench. HLS produced code will generally have a lot of handshake signals but if you set the input ones to 1 right away it should be okay. The reset is active low. There is an example testbench in this same directory, use that as reference. The input can be set to a random number for now, more testing will be done later.

e. Under simulation sources make sure your testbench or “myproject_test.sv” is set as the top-level module there.

f. Right click Run Simulation and select Simulation Settings…



g. Under Simulation, increase the xsim.simulation.runtime to 100000 ns.



Hit Apply and then Ok. If you run into an issue where you are adding more clock cycles to your testbench but your simulation results are not changing, it most likely means you are not giving the simulation enough runtime. Therefore, you

can either change the default period for the clock you defined in your testbench or increase the runtime like we did here.

h. You should now have a working testbench and simulation. Look around the different submodules and their output and make sure each is getting an output that is not 0. If it is 0, should it be 0? Probably not. There may be an issue with your weights or the output is not correct for the HLS code.

4. Start handwriting layers.
    a. What layers are not already written?
    b. For layers that are written, are the parameters the same?
    c. Write the layers you need.
    d. Convert weights and biases into .txt files with the Python script in this same directory. Upload them as a package in the file they are needed in. Use this package: https://github.com/francof2a/fxpmath.
5. Check that the models match.
    a. For one example input, do the outputs of each layer match?
    b. If you add another input, do the outputs still match?

Common problems:

1. Your weights are not being brought in in the order you think they are.
    a. To check ordering of computation – fix the weights.
    If you set all of the weights to be zero with this approach, then the output of the layer should just be the bias for that filter. This way you can check that the values you are sending in are the same. Then add in just one weight and zero out the bias, see if it still matches. You can check that the weights are correctly changing by printing them out like this:

```
In [14]: loaded_model.summary()
         loaded_model.get_weights()

Out[14]: [array([[[[ 0.06020275,  0.16404137, -0.19657356, -0.12554844,
                     0.20385617, -0.08980857,  0.05838475, -0.23521315]],

                  [[ 0.10153291,  0.22388497,  0.0111694 , -0.01896974,
                    -0.00668848, -0.25167084, -0.02478935,  0.11082622]],

                  [[ 0.03196576, -0.21574187,  0.11355463, -0.06249049,
                     0.15765509,  0.14154029,  0.18967474,  0.21455032]]],


                 [[[ 0.25337768, -0.2008658 , -0.16569339,  0.21263725,
                     0.18212259,  0.03575799,  0.02900943, -0.11591563]],

                  [[ 0.26877695,  0.14122525,  0.10640493, -0.09821402,
                     0.07125762, -0.01236323,  0.22774458,  0.20871371]],

                  [[ 0.07250181, -0.21728864, -0.17330101,  0.26439387,
                    -0.07735014, -0.11799052,  0.05874896,  0.11940765]]],
```

b. Setting the size of the array for weights and biases can be tricky, see example below:

```
In [5]: x_test = np.array([(None, 9, 9, 1)])

In [6]: test = np.array([[[[6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1],
                        [6,148,72,35,0,33.6,0.627,50,1]]]],
        dtype="float32")

In [7]: test
Out[7]: array([[[[  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ],
            [  6.   , 148.   ,  72.   ,  35.   ,   0.   ,  33.6  ,
              0.627,  50.   ,   1.   ]]]], dtype=float32)
```

c. Pull out the weights array that is in the HLS simulation and check if your values are the same.

2. HLS reset is active-low.
3. Sometimes the Verilog produced by HLS makes it so the final output is zero. Check internal signals to see if the final output is not being written to.
4. If hls_model.write() is falling, this means a parameter is likely off in your configuration and the FPGA is running out of resources. Is your CNN io_type set to io_stream? Increase the reuse factor on all layers and try again?

```
For reading values out of package files utilize the following command:
#### import data16_10::*;
#### Example can be seen in /2DConv-Stride1/denseLatencyParameterized.sv
```

Questions? Contact Caroline at carolinejjohnson99@gmail.com or 414-702-3798.