

Hardware-accelerated CNN-based data selection for DUNE

Yeon-jae Jwa, Columbia University

DUNE FD DAQ DS/PP WG Meeting

Mar. 2, 2021

Introduction (1)

- CNN-based data selection has been demonstrated to show excellent performance on DUNE FD Monte Carlo simulations, classifying images from the collection plane of a single APA (over a full drift) as containing
 - A single High Energy (HE) interaction (including neutron-antineutron oscillation, proton decay, atmospheric neutrino interaction, or cosmic ray muon)
 - A single Low Energy (LE) interaction (including a supernova neutrino interaction)
 - Electronics Noise and radiological Backgrounds (NB) only.
- High efficiency and low mis-ID rate has been demonstrated, as well as fast inference on GPU, capable of keeping up with total detector rate.
- See [docdb-11311](#) and [IEEE-paper](#).

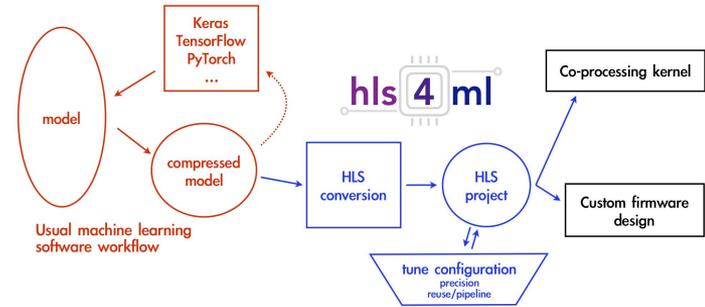
Introduction (2)

- Inference on GPUs is still not ideal:
 - Power consumption is a concern for a potential future application for DUNE.
- In the past few months, we have been investigating alternate hardware architectures for acceleration of ML algorithms:
 - FPGAs: Much more power-aware platform than GPU.
 - FPGA resources available in the current FD DAQ design could be leveraged for more advanced DS.
 -
- FPGAs are more being widely used in high energy physics, especially in triggering/data selection (See, for example, [Phil Harris's Talk at FPGA 2021](#)).

Introduction (3)

- I will be showing results from the training and testing of six (6) distinct CNN's
- These CNNs have different number of layers and parameters, and are trained with and without “quantization” ([arxiv:2006.10159](https://arxiv.org/abs/2006.10159)), in an attempt to utilize fewer resources on FPGA
 - CNN 01
 - Quantized CNN 01
 - CNN 02
 - Quantized CNN 02
 - Downsized CNN 02
 - Quantized Downsized CNN 02
- I will be showing results in terms of **network accuracy** for **implementations of each network on CPU/GPU vs FPGA**, with the latter utilizing hls4ml to translate each network into interpretable language for FPGA

Note: hls4ml tools

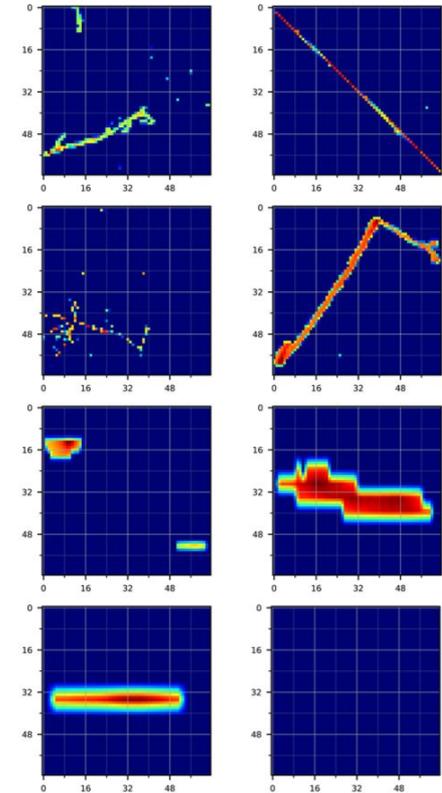


- High Level Synthesis: a process where an algorithmic description for a task is implemented on a hardware.
- *hls4ml* ([official doc](#)): Designed tool developed by particle physicists and computer scientists to enable implementations of ML algorithms on FPGAs.
- Various network architectures with 2D convolutional layers can be tested on Python API using DUNE simulated input images.
- Quantization-aware training ([arxiv:2006.10159](#)) through network quantization for a given CNN architecture is supported in the Python API.
- hls simulation & implementation on the target FPGA is supported.

CNN input image preparation

- The raw collection plane readout of one APA from DUNE simulated events are zero-suppressed.
- Contiguous rectangular regions over threshold ADC are selected as 2D region-of-interests (ROIs).
- The ROIs are resized to 64x64.

	NB (noise/background)	LE (low-energy interaction)	HE (high-energy interaction)
Training set size	12,023	12,050	10,137
Testing set size	4,027	3,970	3,417



64x64 ROIs for HE, LE, NB figure excerpted from (Fig. 3. [IEEE-paper](#))

CNN 01

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	320
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 8)	262152
dense_1 (Dense)	(None, 3)	27
Total params: 262,499		
Trainable params: 262,499		
Non-trainable params: 0		

Keras Accuracy: 0.950

hls4ml Accuracy: 0.784

Keras	NB	LE	HE
trueNB	99.45%	0.55%	0%
trueLE	3.83%	94.23%	1.94%
trueHE	3.37%	6.14%	90.49%

hls sim.	NB	LE	HE
trueNB	98.14%	1.84%	0.02%
trueLE	6.57%	89.1%	4.33%
trueHE	19.37%	37.72%	42.90%

Quantized CNN 01

Model: "sequential"

Layer (type)	Output Shape	Param #
q_conv2d (QConv2D)	(None, 64, 64, 32)	320
relu1 (QActivation)	(None, 64, 64, 32)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
dropout (Dropout)	(None, 32, 32, 32)	0
flatten (Flatten)	(None, 32768)	0
q_dense (QDense)	(None, 8)	262152
relu5 (QActivation)	(None, 8)	0
q_dense_1 (QDense)	(None, 3)	27
softmax (Activation)	(None, 3)	0

Total params: 262,499
Trainable params: 262,499
Non-trainable params: 0

Keras Accuracy: 0.932

hls4ml Accuracy: 0.932

Keras	NB	LE	HE
trueNB	99.3%	0.7%	0%
trueLE	4.33%	92.9%	2.77%
trueHE	5.09%	8.34%	86.5%

hls sim.	NB	LE	HE
trueNB	99.3%	0.7%	0%
trueLE	4.33%	93.12%	2.54%
trueHE	5.26%	8.78%	85.95%

CNN 02

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	320
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 8)	131080
dense_1 (Dense)	(None, 3)	27

Total params: 149,923

Trainable params: 149,923

Non-trainable params: 0

Keras Accuracy: 0.945

hls4ml Accuracy: 0.607

Keras	NB	LE	HE
trueNB	99.5%	0.50%	0%
trueLE	3.98%	93.25%	2.77%
trueHE	3.25%	6.58%	90.17%

hls sim.	NB	LE	HE
trueNB	98.06%	0.25%	1.91%
trueLE	22.75%	10.6%	66.65%
trueHE	21.54%	3.72%	74.74%

Quantized CNN 02

Model: "sequential"

Layer (type)	Output Shape	Param #
q_conv2d (QConv2D)	(None, 64, 64, 32)	320
relu1 (QActivation)	(None, 64, 64, 32)	0
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
q_conv2d_1 (QConv2D)	(None, 32, 32, 64)	18496
relu2 (QActivation)	(None, 32, 32, 64)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
q_dense (QDense)	(None, 8)	131080
relu5 (QActivation)	(None, 8)	0
q_dense_1 (QDense)	(None, 3)	27
softmax (Activation)	(None, 3)	0

Total params: 149,923

Trainable params: 149,923

Non-trainable params: 0

Keras Accuracy: 0.952

hls4ml Accuracy: 0.946

Keras	NB	LE	HE
trueNB	99.73%	0.27%	0%
trueLE	4.16%	95.06%	0.78%
trueHE	3.78%	8.08%	88.15%

hls sim.	NB	LE	HE
trueNB	99.53%	0.47%	0%
trueLE	3.98%	95.79%	1.23%
trueHE	3.60%	5.77%	90.64%

Downsized CNN 02

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 4)	40
max_pooling2d (MaxPooling2D)	(None, 16, 16, 4)	0
conv2d_1 (Conv2D)	(None, 16, 16, 8)	296
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 8)	0
dropout (Dropout)	(None, 4, 4, 8)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 8)	1032
dense_1 (Dense)	(None, 3)	27

Total params: 1,395

Trainable params: 1,395

Non-trainable params: 0

Keras Accuracy: 0.950

hls4ml Accuracy: 0.791

Keras	NB	LE	HE
trueNB	99.48%	0.52%	0%
trueLE	3.70%	94.38%	1.92%
trueHE	3.01%	6.50%	90.49%

hls sim.	NB	LE	HE
trueNB	99.53%	0.47%	0%
trueLE	4.94%	93.12%	1.94%
trueHE	21.22%	40.12%	38.66%

Downsized Quantized CNN 02

Model: "sequential"

Layer (type)	Output Shape	Param #
q_conv2d (QConv2D)	(None, 64, 64, 4)	40
relu1 (QActivation)	(None, 64, 64, 4)	0
max_pooling2d (MaxPooling2D)	(None, 16, 16, 4)	0
q_conv2d_1 (QConv2D)	(None, 16, 16, 8)	296
relu2 (QActivation)	(None, 16, 16, 8)	0
max_pooling2d_1 (MaxPooling2D)	(None, 4, 4, 8)	0
dropout (Dropout)	(None, 4, 4, 8)	0
flatten (Flatten)	(None, 128)	0
q_dense (QDense)	(None, 8)	1032
relu5 (QActivation)	(None, 8)	0
q_dense_1 (QDense)	(None, 3)	27
softmax (Activation)	(None, 3)	0

Total params: 1,395

Trainable params: 1,395

Non-trainable params: 0

Keras Accuracy: 0.946

hls4ml Accuracy: 0.945

Keras	NB	LE	HE
trueNB	99.48%	0.52%	0%
trueLE	3.90%	94.79%	1.31%
trueHE	3.31%	8.08%	88.61%

hls sim.	NB	LE	HE
trueNB	99.48%	0.52%	0%
trueLE	3.93%	95.19%	0.88%
trueHE	3.31%	8.93%	87.77%

Downsized CNN 02 Resource Usage

Xilinx FPGA (xc7k115-flvb2104-2-i)

== Performance Estimates

+ Timing:

* Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.375 ns	0.62 ns

+ Latency:

* Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
4680	4680	23.400 us	23.400 us	4364	4364	dataflow

BRAM (Block RAM) : stores data

DSP (digital signal processor) : computes multiplication and arithmetic

FF (flip-flops) : registers data in time with the clock pulse

LUT (look-up table) : performs logic function

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	63	-	1961	3996	-
Instance	112	1372	76571	57229	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	175	1372	78538	61293	0
Available SLR	2160	2760	663360	331680	0
Utilization SLR (%)	8	49	11	18	100
Available	4320	5520	1326720	663360	0
Utilization (%)	4	24	5	9	0

Downsized Quantized CNN 02 Resource Usage

Xilinx FPGA (xc7k115-flvb2104-2-i)

== Performance Estimates

+ Timing:

* Summary:

Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.375 ns	0.62 ns

+ Latency:

* Summary:

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
4679	4679	23.395 us	23.395 us	4364	4364	dataflow

BRAM (Block RAM) : stores data

DSP (digital signal processor) : computes multiplication and arithmetic

FF (flip-flops) : registers data in time with the clock pulse

LUT (look-up table) : performs logic function

== Utilization Estimates

* Summary:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	32	-
FIFO	55	-	1853	3816	-
Instance	48	884	61781	57309	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	6	-	-
Total	103	884	63640	61193	0
Available SLR	2160	2760	663360	331680	0
Utilization SLR (%)	4	32	9	18	100
Available	4320	5520	1326720	663360	0
Utilization (%)	2	16	4	9	0

Performance summary of networks

Network	Number of trainable params.	Accuracy from CPU inference	Accuracy from hls inference	Latency on Xilinx FPGA (xcku115-flvb2104-2-i)	Resource Utilization on Xilinx FPGA (xcku115-flvb2104-2-i)			
					BRAM (%)	DSP (%)	LL (%)	LUT (%)
CNN 01	242,499	94.95%	78.46%	NA	NA			
Q CNN 01	242,499	93.24%	93.16%	NA	NA			
CNN 02	149,923	94.53%	60.66%	NA	NA			
Q CNN 02	149,923	95.21%	94.64%	NA	NA			
Downsized CNN 02	1,395	95.01%	79.08%	23.4 μ s	4%	24%	5%	9%
Downsized Q CNN 02	1,395	94.59%	94.48%	23.395 μ s	2%	16%	4%	9%

- Overall ~94-95% accuracy could be achieved in simple networks.
- Quantization-aware training (QAT) does not cause accuracy loss in hls inference.
- Downsized networks with reduced number of trainable parameters can be implemented in the target FPGA (with significant resource margins).

Summary

- Simple ML networks with only single or double 2D convolution layers can achieve ~95% tagging accuracy for noise/background, low energy, and high energy simulations (raw ADC values).
- Quantized networks (QCNN 01, QCNN 02, downsized QCNN 02) did not show accuracy drop in hls simulation.
- Significant downsizing (~100x reduction of trainable parameters) enabled the hls synthesis on our target FPGA, allows reaching 23 μ s inference and promising resource utilization; the inference accuracy was maintained.
- Further network architecture optimization, possibly in combination with quantization aware pruning ([arxiv:2102.11289](https://arxiv.org/abs/2102.11289)), can be studied in near future.

Back up

Classification performance from GPU study

TABLE II

GPU INFERENCE RESULTS USING METHOD 1, OBTAINED WITH A WITH VGG16B NETWORK (TRAINING FOR 2 EPOCHS AND LEARNING RATE SET TO 2×10^{-4}).

Sample	Train Size	Test Size	Accuracy (%)			Inference Time (ms)
			ϵ_{NB}	ϵ_{LE}	ϵ_{HE}	
NB	51,100	99,000	91.45	8.49	0.06	27.7±8.6
LE	44,900	29,800	3.17	96.83	0	
HE	52,828	67,178	6.03	3.48	90.48	

TABLE IV

GPU INFERENCE RESULTS USING METHOD 2, OBTAINED WITH THE CNN_S NETWORK (TRAINING FOR 48 EPOCHS AND LEARNING RATE SET TO 2×10^{-3}).

Sample	Train Size	Test Size	Accuracy (%)			Inference Time (ms)
			ϵ_{NB}	ϵ_{LE}	ϵ_{HE}	
NB	12,023	4,027	99.53	0.47	0.12	1.6±0.1
LE	12,050	3,970	4.01	94.48	1.51	
HE	10,137	3,417	3.63	6.15	90.22	

CNN implementation through hls4ml

- hls4ml installation & set-up ([github repo.](#))
- Define network

```
def CNN_02_DS(input_shape, num_classes, data_format):  
  
    if data_format == 'channels_first':  
        img_chnl, img_rows, img_cols = input_shape  
    else:  
        img_rows, img_cols, img_chnl = input_shape  
    model = Sequential()  
    model.add(Conv2D(4, kernel_size=(3, 3), padding='same', activation='relu', input_shape=input_shape, data_format=d  
    model.add(MaxPooling2D(pool_size=(4, 4)))  
    model.add(Conv2D(8, (3, 3), padding='same', activation='relu'))  
    model.add(MaxPooling2D(pool_size=(4, 4)))  
    model.add(Dropout(0.25))  
    model.add(Flatten())  
    model.add(Dense(8, activation='relu'))  
    model.add(Dense(num_classes, activation='softmax'))  
  
    return model
```

CNN implementation through hls4ml

- Train and save model using Keras

```
train = False

if (train):
    filepath='./model_cnn_02_ds/weights-{epoch:02d}-{val_accuracy:.3f}.ckpt'
    checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', verbose=1, save_best_only=True, mode='max')
    callbacks_list = [checkpoint]
    model.fit(X_train, y_train, batch_size=batch_size, epochs=70, verbose=1, callbacks=callbacks_list, validation_data=(X_val, y_val))
    model.save('./model_cnn_02_ds/cnn_02_ds.h5')
    save_model(model, 'cnn_02_ds', './model_cnn_02_ds')
    print('INFO: Evaluation...')
    score = model.evaluate(X_test, y_test, verbose=1)
    print('INFO: Test loss:', score[0])
    print('INFO: Test accuracy:', score[1])
    print('INFO: Accuracy per class (Background = 0, Low-Energy=1, High-Energy = 2)')
```

```
best_model = tf.keras.models.load_model('./model_cnn_02_ds/weights-63-0.953.ckpt')

y_prob = best_model.predict(X_test)

y_test = np.argmax(y_test, axis=1) # Convert one-hot to index

y_pred = best_model.predict_classes(X_test)
pred_csv_file = open('label_prob_cnn_02_ds.csv', 'w+')

for i in range(y_prob.shape[0]):
    pred_csv_file.write('%d %f %f %f\n' %(y_test[i], y_prob[i][0], y_prob[i][1], y_prob[i][2]))

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	0.94	0.99	0.97	4027
1	0.94	0.94	0.94	3970
2	0.98	0.90	0.94	3417
accuracy			0.95	11414
macro avg	0.95	0.95	0.95	11414
weighted avg	0.95	0.95	0.95	11414

CNN implementation through hls4ml

- Converse Keras model to hls model

```
import hls4ml

config = hls4ml.utils.config_from_keras_model(model, granularity='name')

print("-----")
print("Configuration")

config['Model'] = {}
config['Model']['Precision'] = 'ap_fixed<16,6>'
config['Model']['ReuseFactor'] = 1

print("-----")
hls_model = convert_from_keras_model_custom(best_model,
                                           hls_config=config,
                                           output_dir='model_cnn_02_ds/hls4ml_prj',
                                           fpga_part='xcvu115-flvb2104-2-i')
```

Interpreting Sequential

Topology:

Layer name: conv2d, layer type: Conv2D

-> Activation (relu), layer name: conv2d

Layer name: max_pooling2d, layer type: MaxPooling2D

Layer name: conv2d_1, layer type: Conv2D

-> Activation (relu), layer name: conv2d_1

Layer name: max_pooling2d_1, layer type: MaxPooling2D

Layer name: dense, layer type: Dense

-> Activation (relu), layer name: dense

Layer name: dense_1, layer type: Dense

-> Activation (softmax), layer name: dense_1

Configuration

Interpreting Sequential

Input shape: [64, 64, 1]

Topology:

Layer name: conv2d, layer type: Conv2D, current shape: [[None, 64, 64, 1]]

Layer name: max_pooling2d, layer type: MaxPooling2D, current shape: [[None, 64, 64, 4]]

Layer name: conv2d_1, layer type: Conv2D, current shape: [[None, 16, 16, 4]]

Layer name: max_pooling2d_1, layer type: MaxPooling2D, current shape: [[None, 16, 16, 8]]

Layer name: dense, layer type: Dense, current shape: [[None, 4, 4, 8]]

Layer name: dense_1, layer type: Dense, current shape: [[None, 8]]

Creating HLS model

```
from hls4ml.converters.keras_to_hls import keras_to_hls

def convert_from_keras_model_custom(model, output_dir='my-hls-test', project_name='myproject',
                                   fpga_part='xcvu115-flvb2104-2-i', clock_period=5, hls_config={}):
    config = hls4ml.converters.create_vivado_config(output_dir=output_dir,
                                                    project_name=project_name, fpga_part=fpga_part, clock_period=clock_period)
    config['KerasModel'] = model
    config['IOType'] = 'io_stream'
    model_config = hls_config.get('Model', None)
    if model_config is not None:
        if not all(k in model_config for k in ('Precision', 'ReuseFactor')):
            raise Exception('Precision and ReuseFactor must be provided in the hls_config')
    else:
        model_config = {}
        model_config['Precision'] = 'ap_fixed<16,6>'
        model_config['ReuseFactor'] = '1'
    config['HLSConfig']['Model'] = model_config

    if 'LayerName' in hls_config:
        config['HLSConfig']['LayerName'] = hls_config['LayerName']

    if 'LayerType' in hls_config:
        config['HLSConfig']['LayerType'] = hls_config['LayerType']

    if 'Optimizers' in hls_config:
        config['HLSConfig']['Optimizers'] = hls_config['Optimizers']

    return keras_to_hls(config)
```

CNN implementation through hls4ml

- Compile hls model, predict the accuracy

```
hls_model.compile()
```

```
Writing HLS project  
Done
```

```
y_hls = hls_model.predict(X_test)
```

```
print("Keras Accuracy: {}".format(accuracy_score(y_test, np.argmax(y_prob,axis=1))))  
print("hls4ml Accuracy: {}".format(accuracy_score(y_test, np.argmax(y_hls,axis=1))))
```

```
Keras Accuracy: 0.9501489398983705  
hls4ml Accuracy: 0.7907832486420185
```

CNN implementation through hls4ml

- Building hls synthesis, RTL implementation (vivado_hls -f build_prj.tcl)

```
os.environ['PATH'] = '/scratch/Xilinx/Vivado/2019.2/bin:' + os.environ['PATH']  
hls_model.build(csim=False)
```

```
{'EstimatedClockPeriod': '4.375',  
  'BestLatency': '4680',  
  'WorstLatency': '4680',  
  'IntervalMin': '4364',  
  'IntervalMax': '4364',  
  'BRAM_18K': '175',  
  'DSP48E': '1372',  
  'FF': '78538',  
  'LUT': '61293',  
  'URAM': '0',  
  'AvailableBRAM_18K': '4320',  
  'AvailableDSP48E': '5520',  
  'AvailableFF': '1326720',  
  'AvailableLUT': '663360',  
  'AvailableURAM': '0'}
```

CNN implementation through hls4ml

- Read vivado report on RTL implementation

```
hls4ml.report.read_vivado_report('model_cnn_02_ds/hls4ml_prj/')
```

```
Found 1 solution(s) in model_cnn_02_ds/hls4ml_prj//myproject_prj.  
Reports for solution "solution1":
```

```
C simulation report not found.
```

```
SYNTHESIS REPORT:
```

```
=====  
== Vivado HLS Report for 'myproject'  
=====
```

```
* Date:          Fri Feb 26 22:54:05 2021
```

```
* Version:       2019.2 (Build 2704478 on Wed Nov 06 22:10:23 MST 2019)  
* Project:       myproject_prj  
* Solution:      solution1  
* Product family: kintexu  
* Target device: xcku115-flvb2104-2-i
```

```
=====  
== Performance Estimates  
=====
```

```
+ Timing:
```

```
* Summary:
```

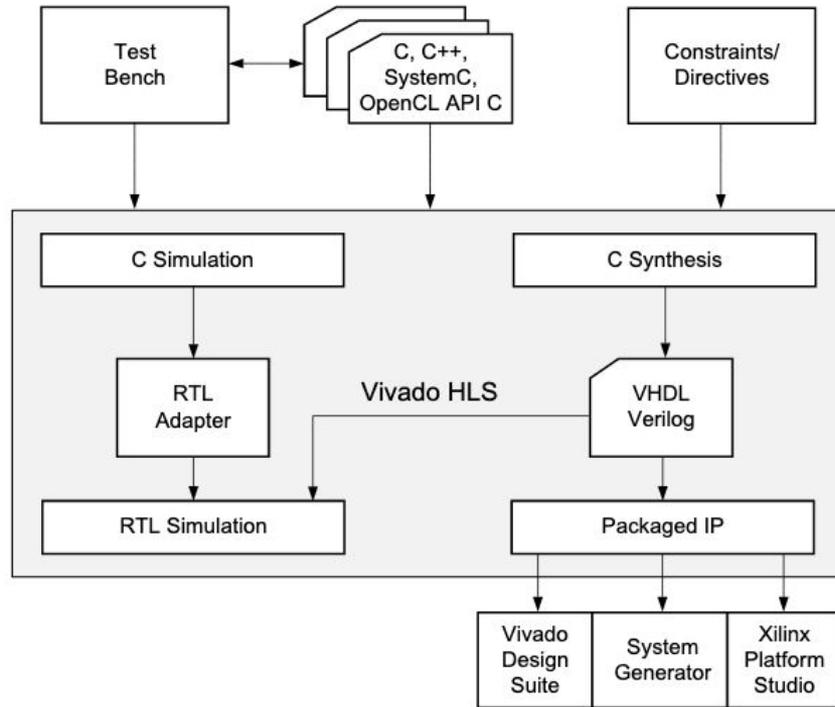
Clock	Target	Estimated	Uncertainty
ap_clk	5.00 ns	4.375 ns	0.62 ns

```
+ Latency:
```

```
* Summary:
```

Latency (cycles)		Latency (absolute)		Interval		Pipeline
min	max	min	max	min	max	Type
4680	4680	23.400 us	23.400 us	4364	4364	dataflow

Vivado HLS implementation flow



X14309

Figure 1-4: Vivado HLS Design Flow

QKeras:

Post-Training Quantization vs. Quantization-aware training (QAT)

- Network Quantization :
 - Reducing the precision of the calculations for weights, bias, inputs.
 - FPGA uses fixed-point representation.
 - Allow optimization of resource usage; crucial for fast machine learning.
- Post-training quantization :
 - Weights and activations can be quantized after the training.
 - Possible degradation of accuracies.
- Quantization-aware training (QKeras) :
 - *Layers* in Keras are replaced with *QLayers* in QKeras, specifying *kernel_quantizer* with desired fixed-point precision.
 - The span of weights in each layer can be profiled prior of training, to decide suitable precision.
 - Achieve better accuracy than post-training quantization.

0101.1010011101
← integer ← fractional →