UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Universidad
Nacional de San Luis

The Abdus Salam
International Centre
for Theoretical Physics

FPGA for the Acceleration of Machine Learning Algorithms

Romina Molina

Joint ICTP-IAEA School on FPGA-based SoC
and its Applications for Nuclear and Related Instrumentation — (smr 3562)

January – February 2021

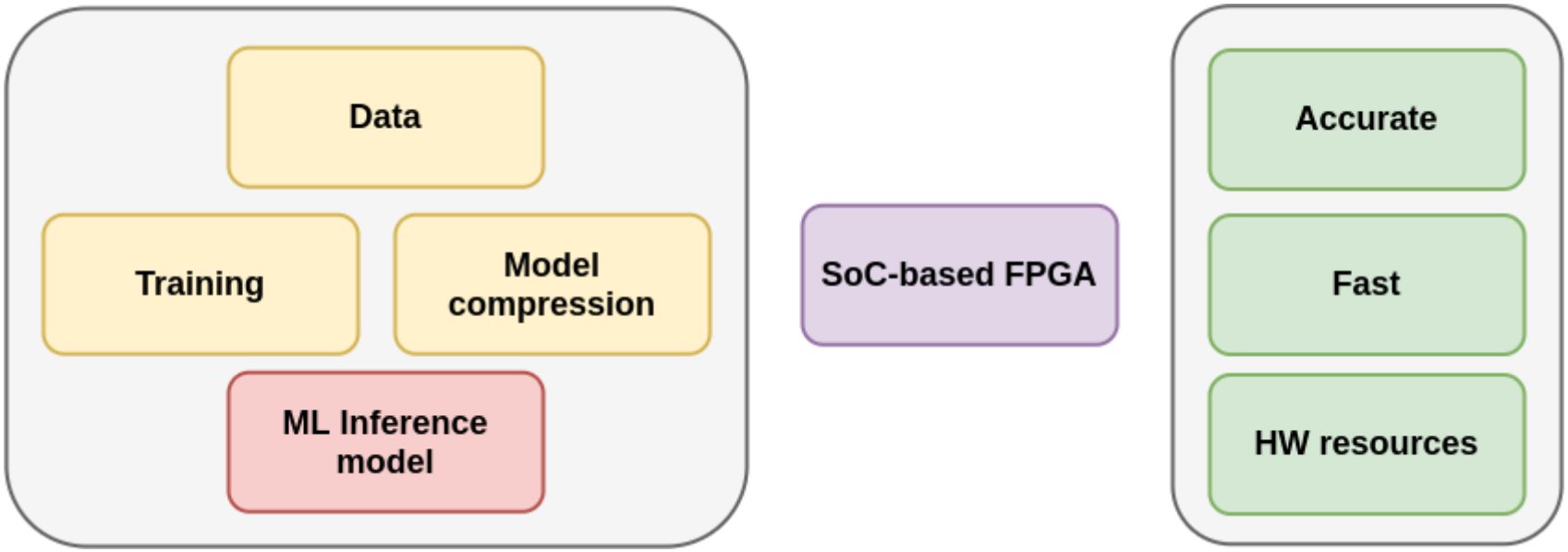
Outline

- Introduction
- Machine Learning (ML)
- SoC-based FPGA
- Acceleration of ML Inference
- High-Level Synthesis for ML (hls4ml)

Introduction

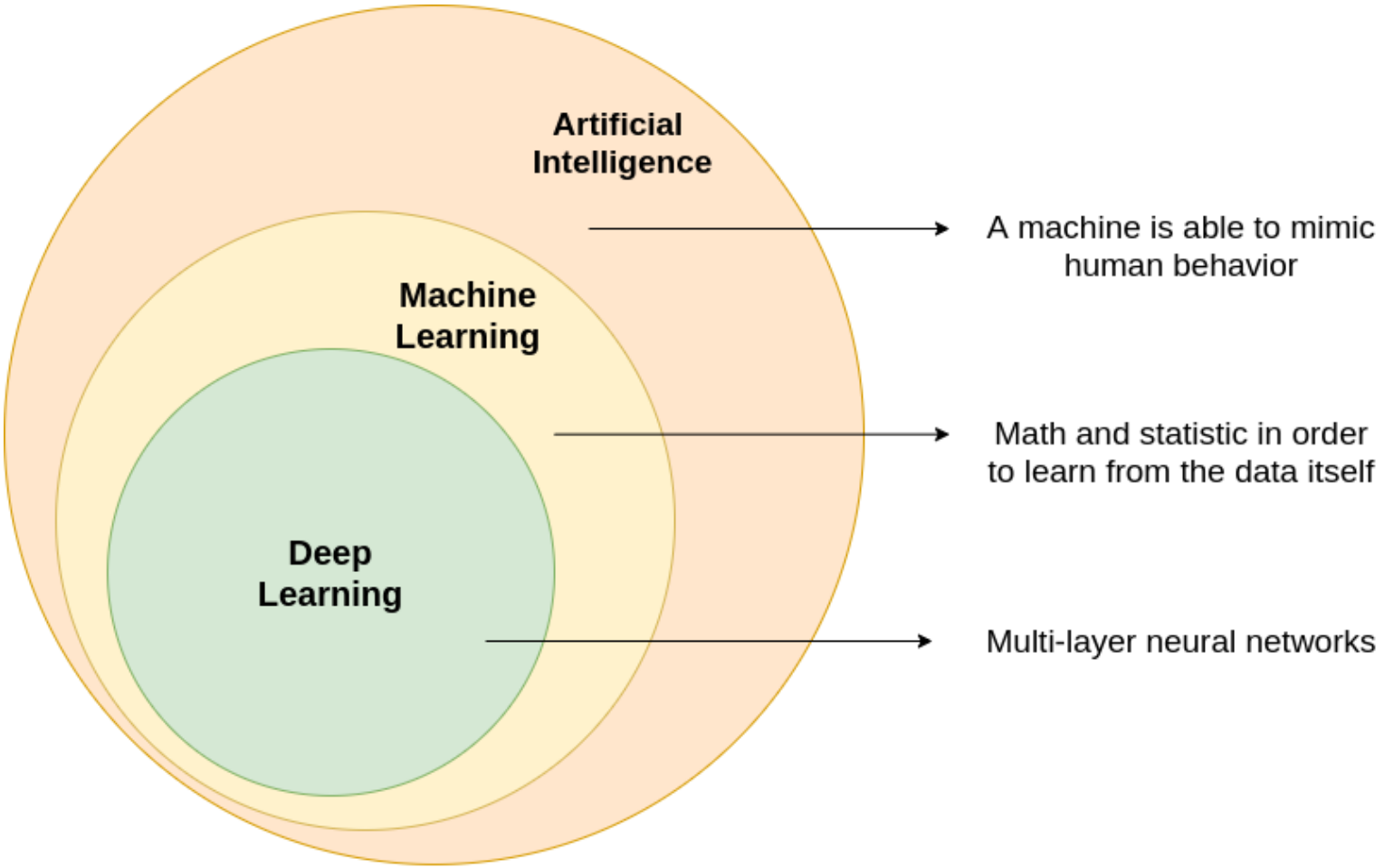
Introduction

Machine learning and SoC



Machine Learning

Machine Learning

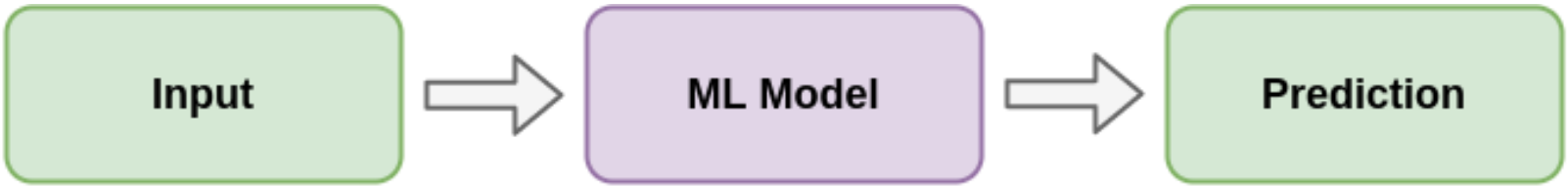


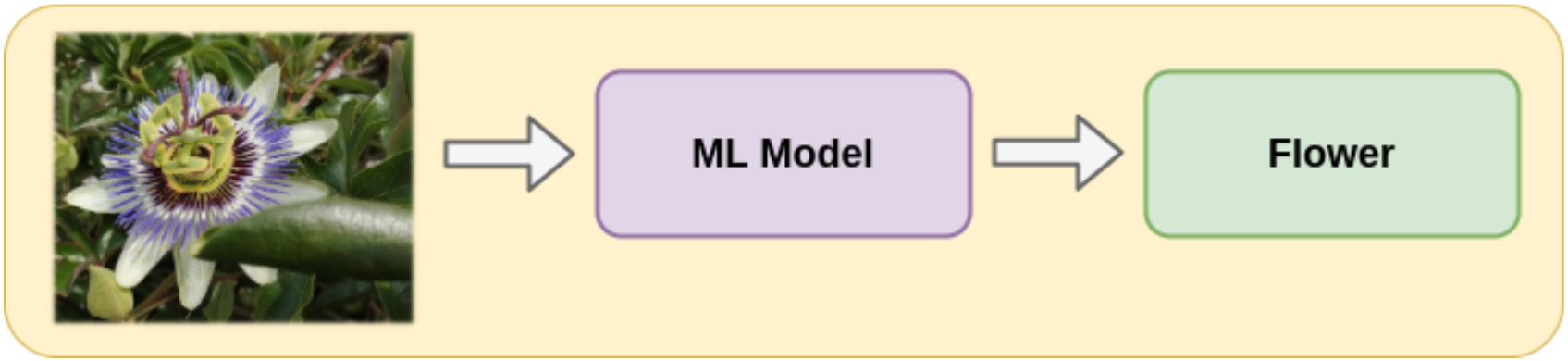
Artificial Intelligence, Machine Learning, Deep Learning

Machine Learning

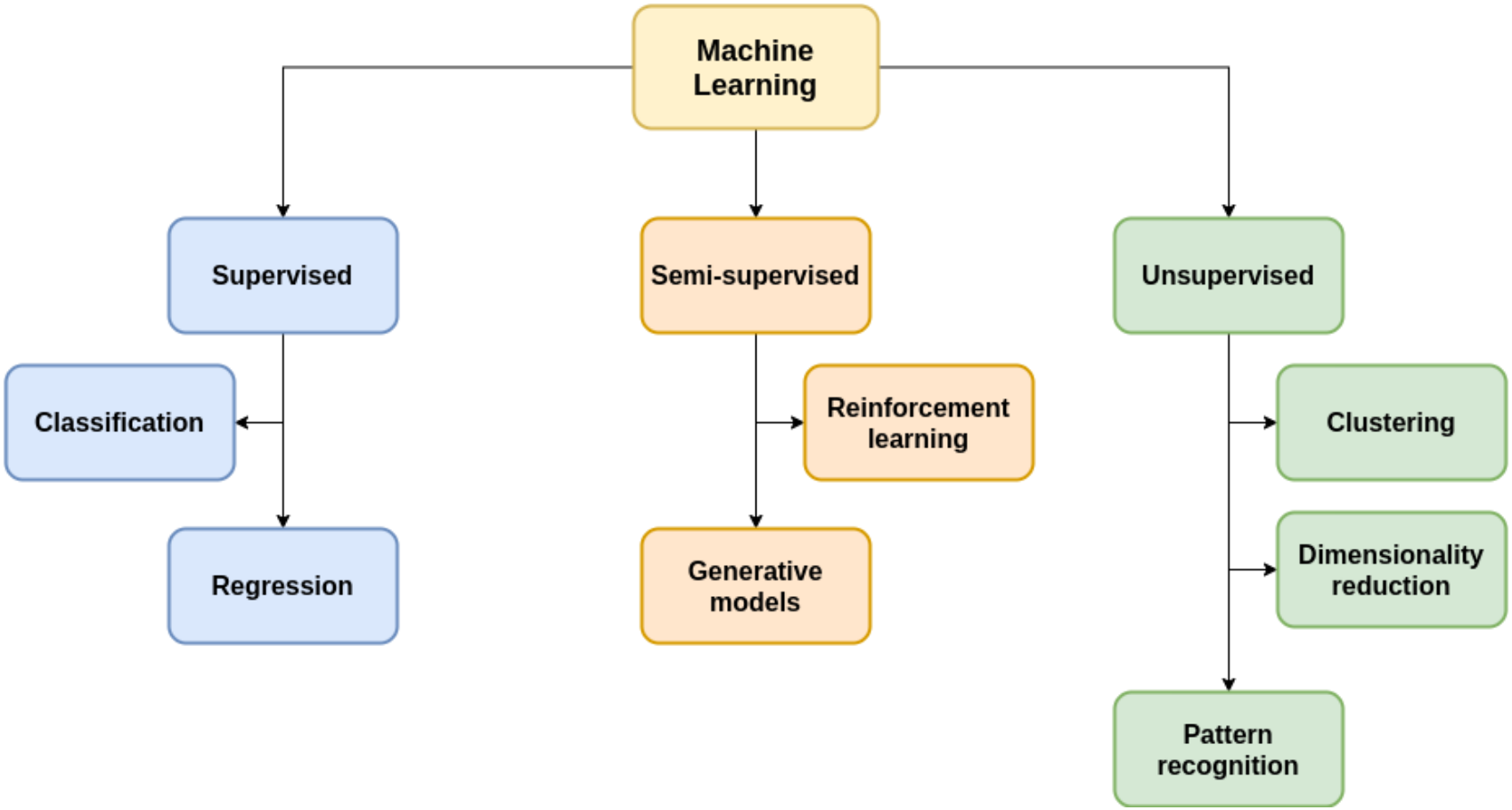
"Learning can be defined as the process of estimating associations between inputs, outputs, and parameters of a system using a limited number of observations" (Cherkassky et al. 2007)

Machine Learning



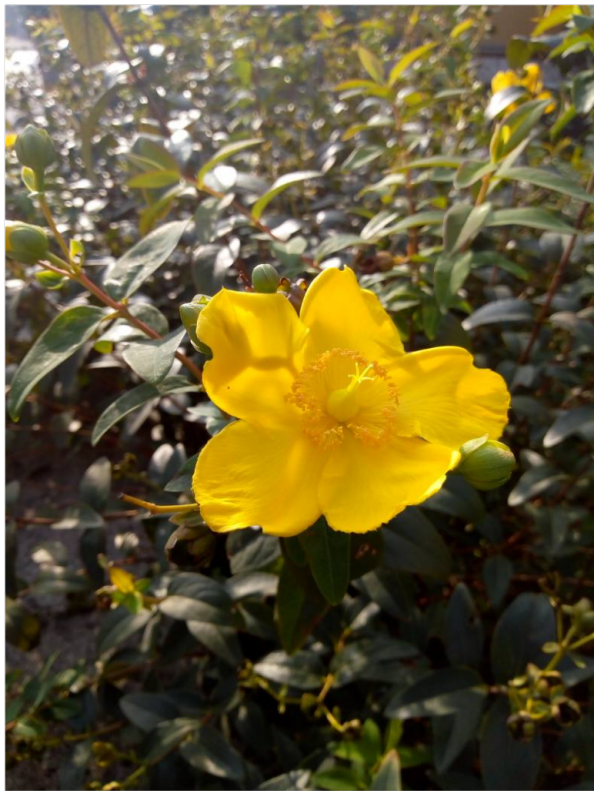


Machine Learning



Machine Learning

Sea



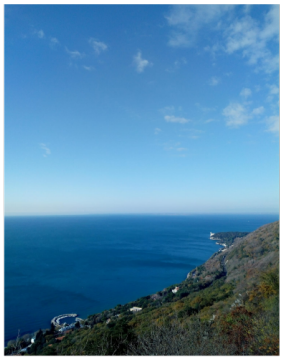
Flower

Supervised Learning

Machine Learning

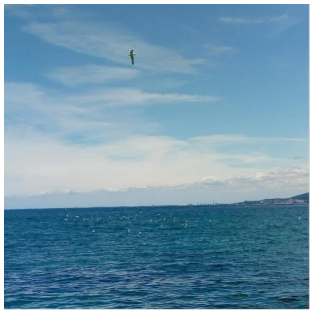


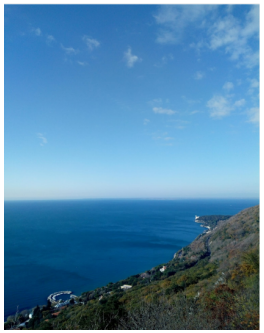




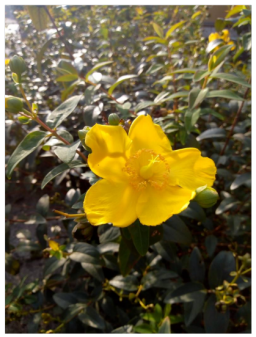

Unsupervised Learning

Machine Learning





Sea



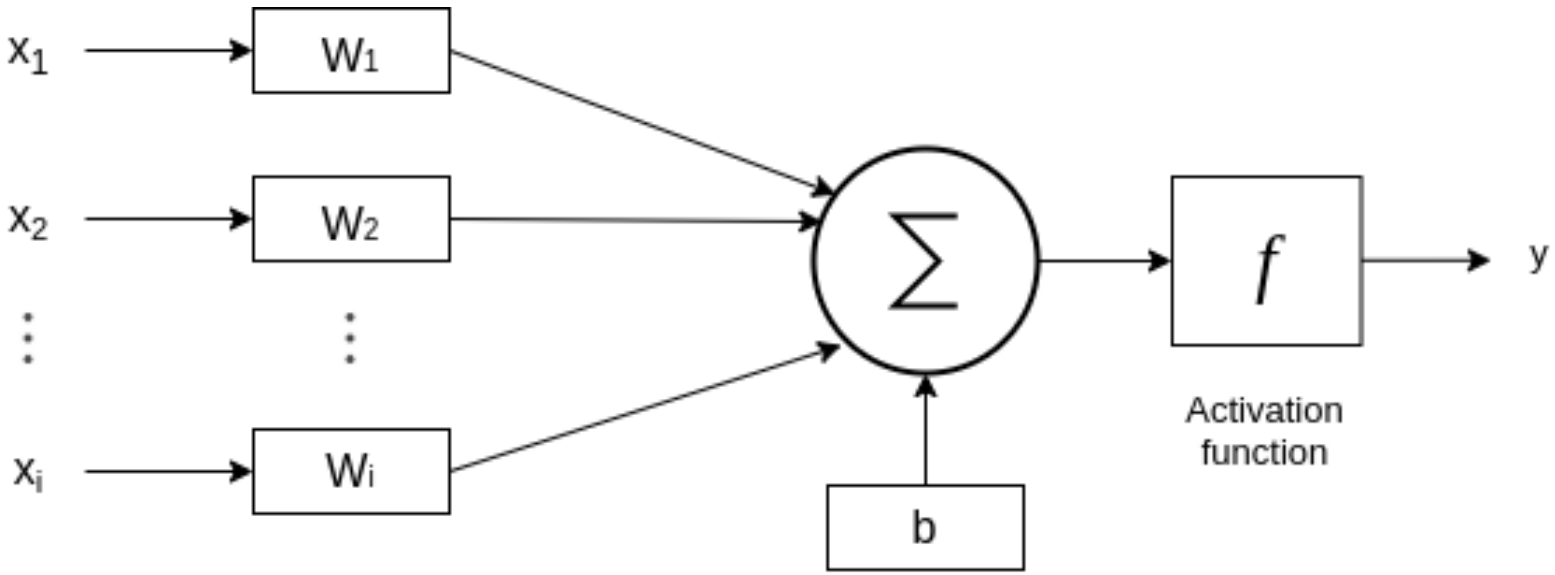
Flower

Unsupervised Learning

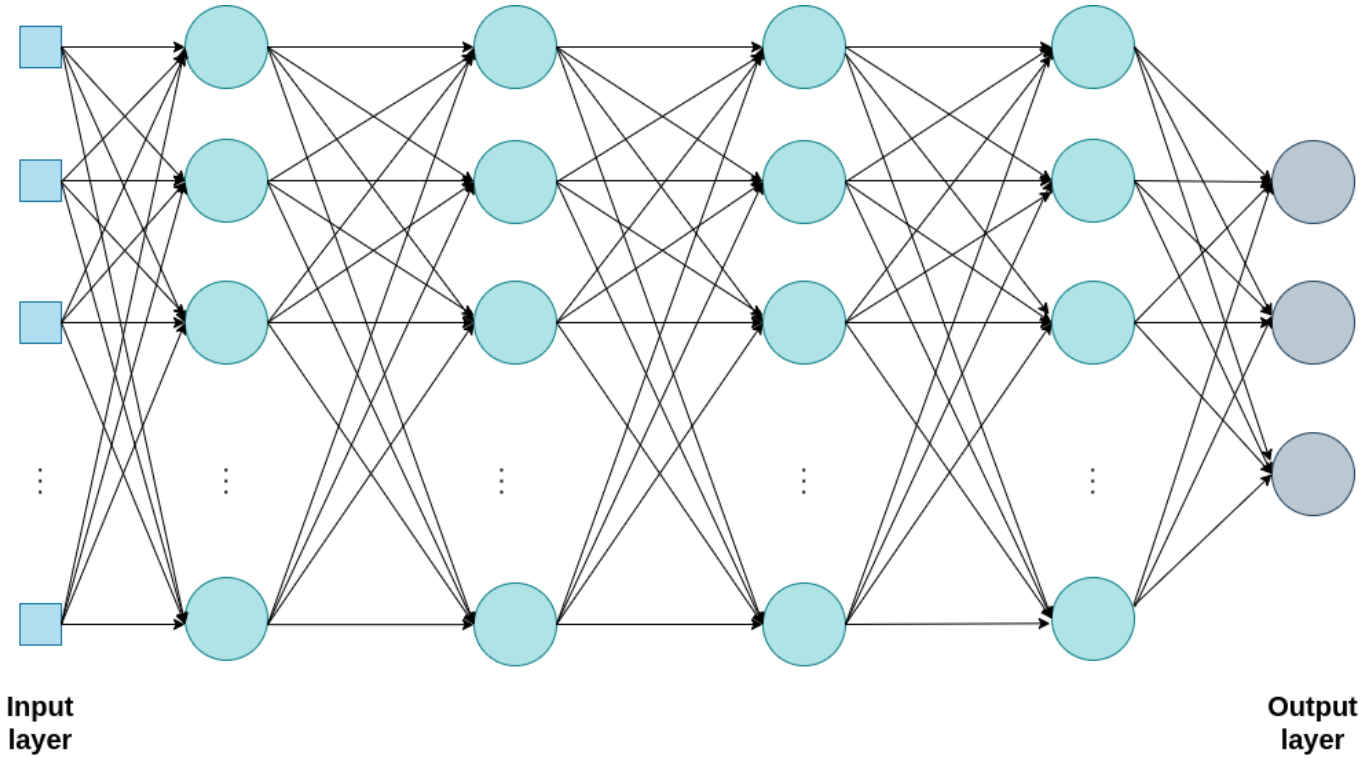
Machine Learning

An artificial neural network (ANN) is composed of neuron (or node) interconnections arranged in different layers.

$$y = f(b + \sum w_i x_i) \tag{1}$$

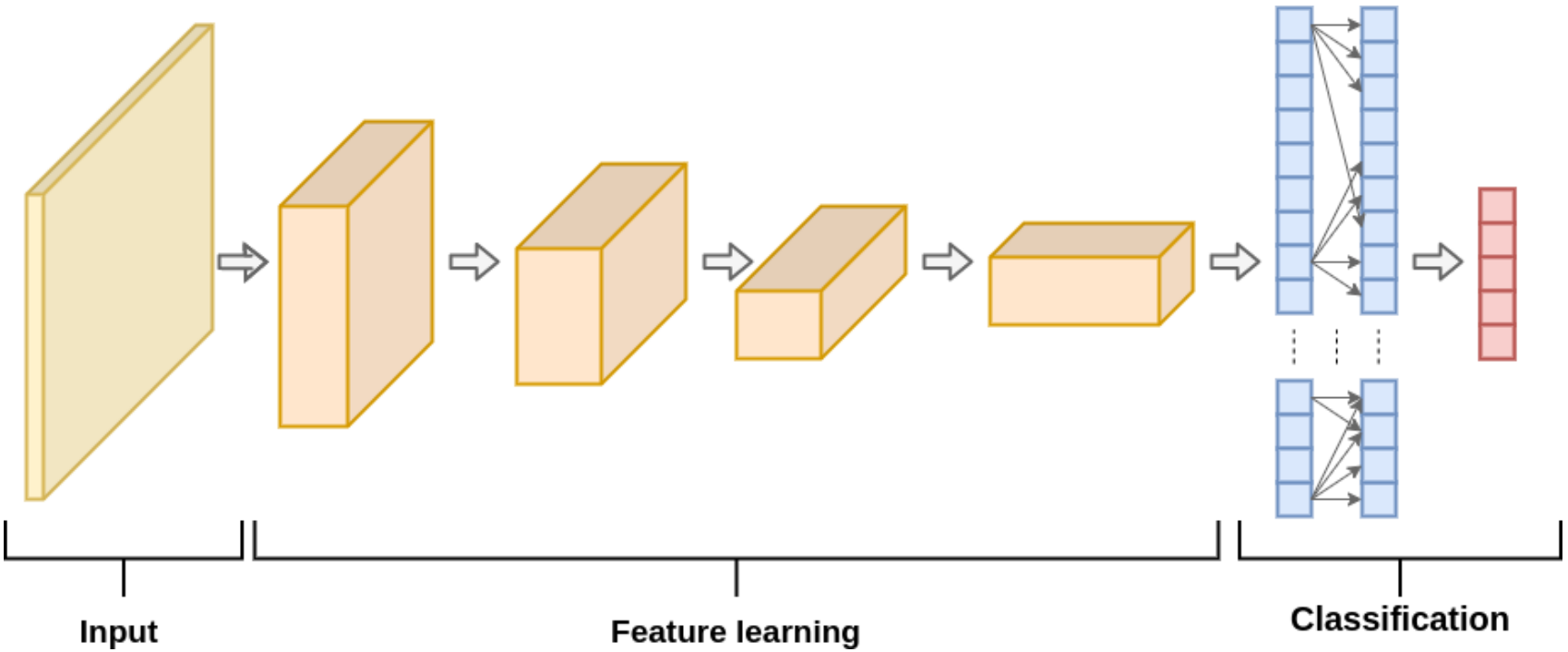


Multi-Layer Perceptron

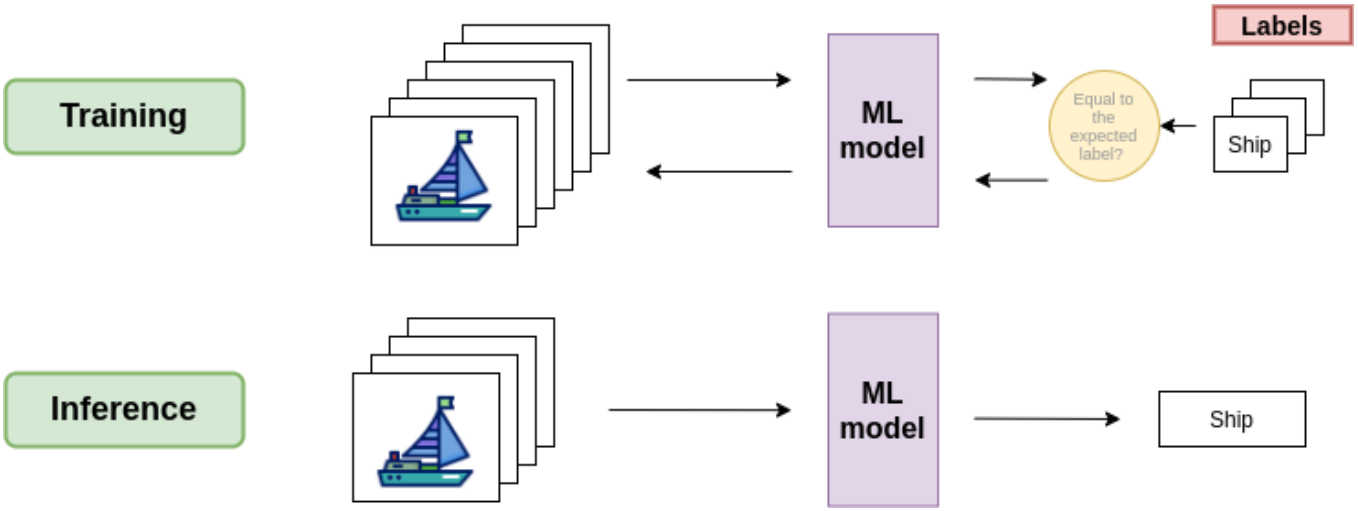


MLP architecture

Convolutional Neural Networks (CNN)



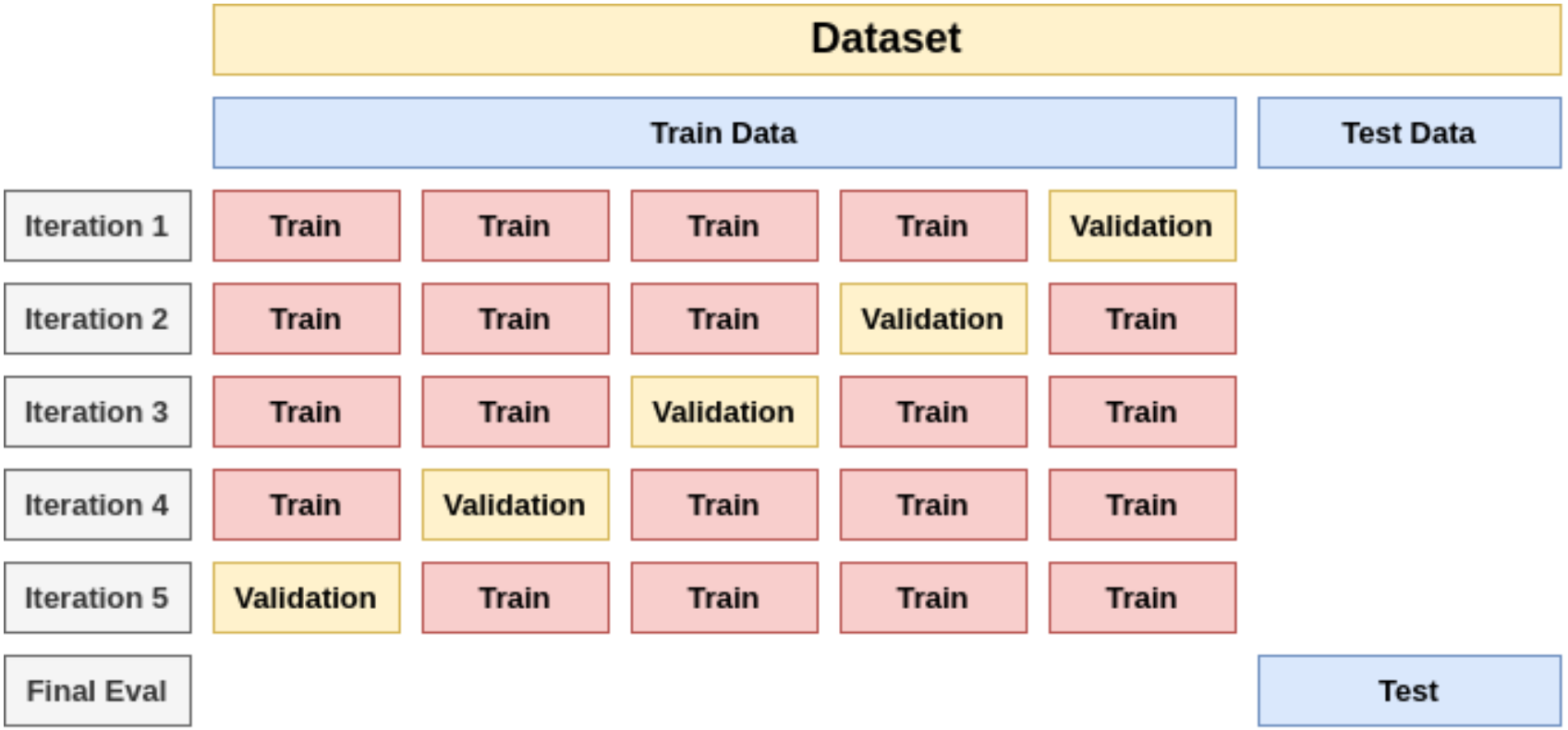
Machine Learning



Training and inference

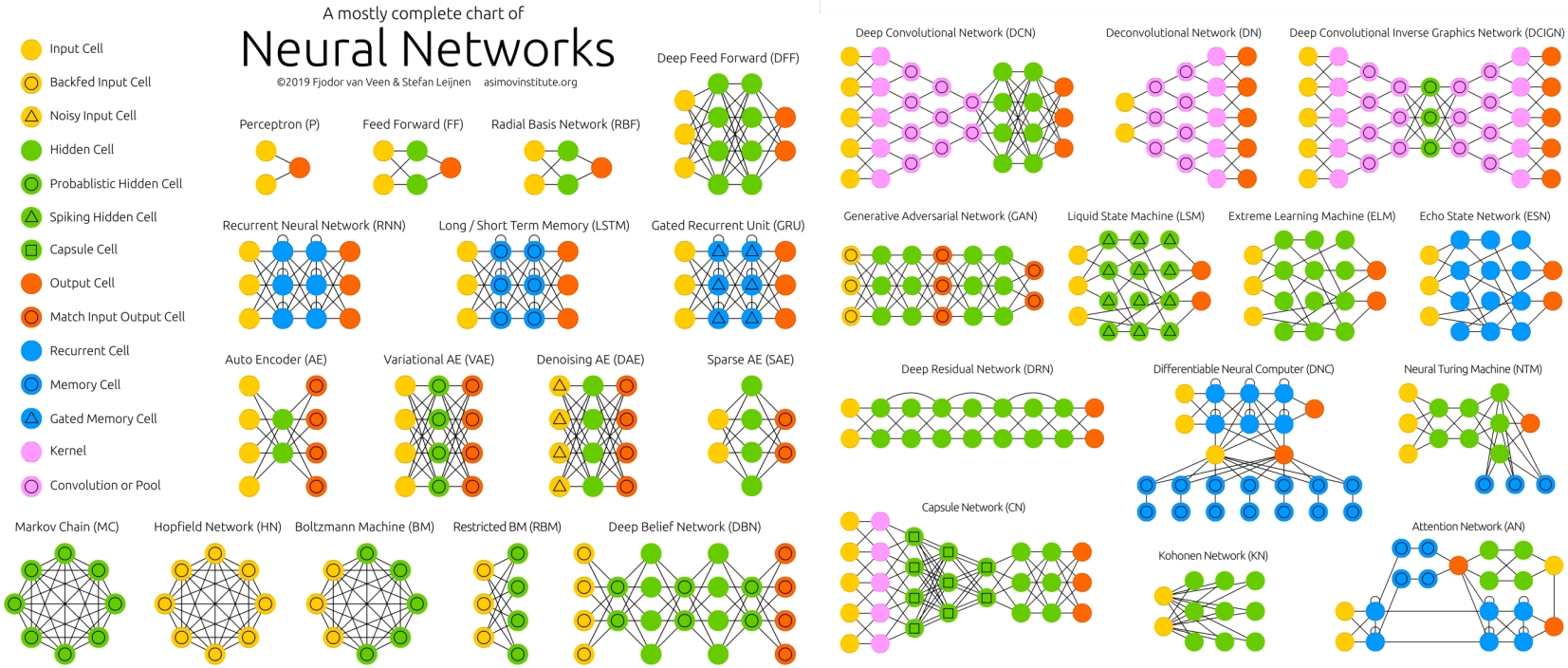
In a classifier, an input is mapped into a specific class.
Supervised training step to recognize patterns: the network compares its actual output with the desired output. The difference between these two values is adjusted with backpropagation.

Machine Learning



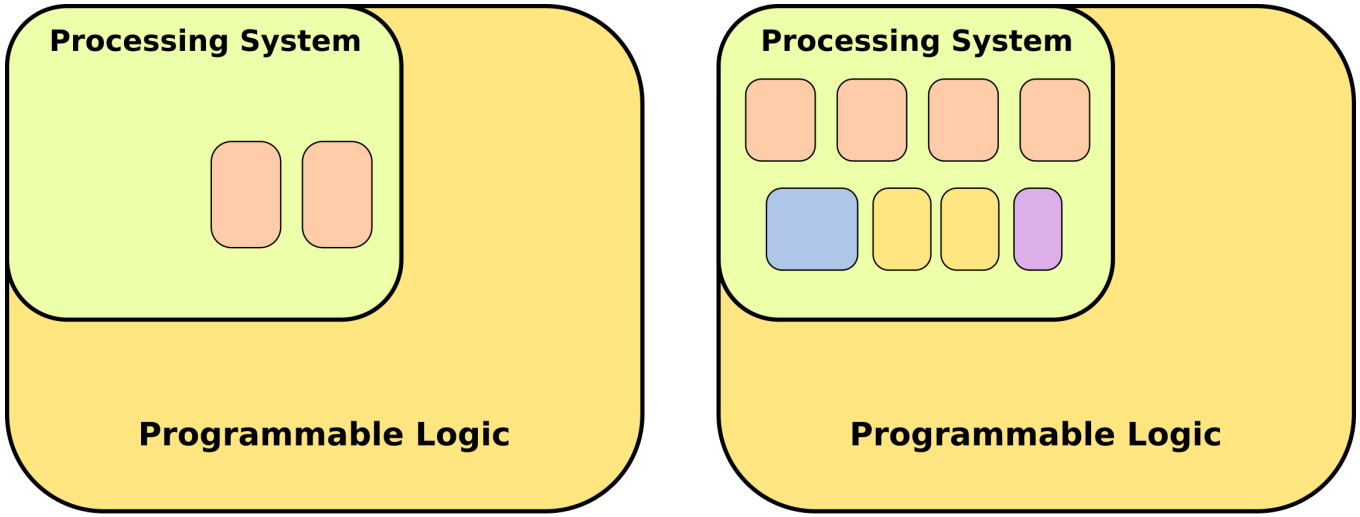
K-fold cross-validation

Machine Learning



SoC-based FPGA

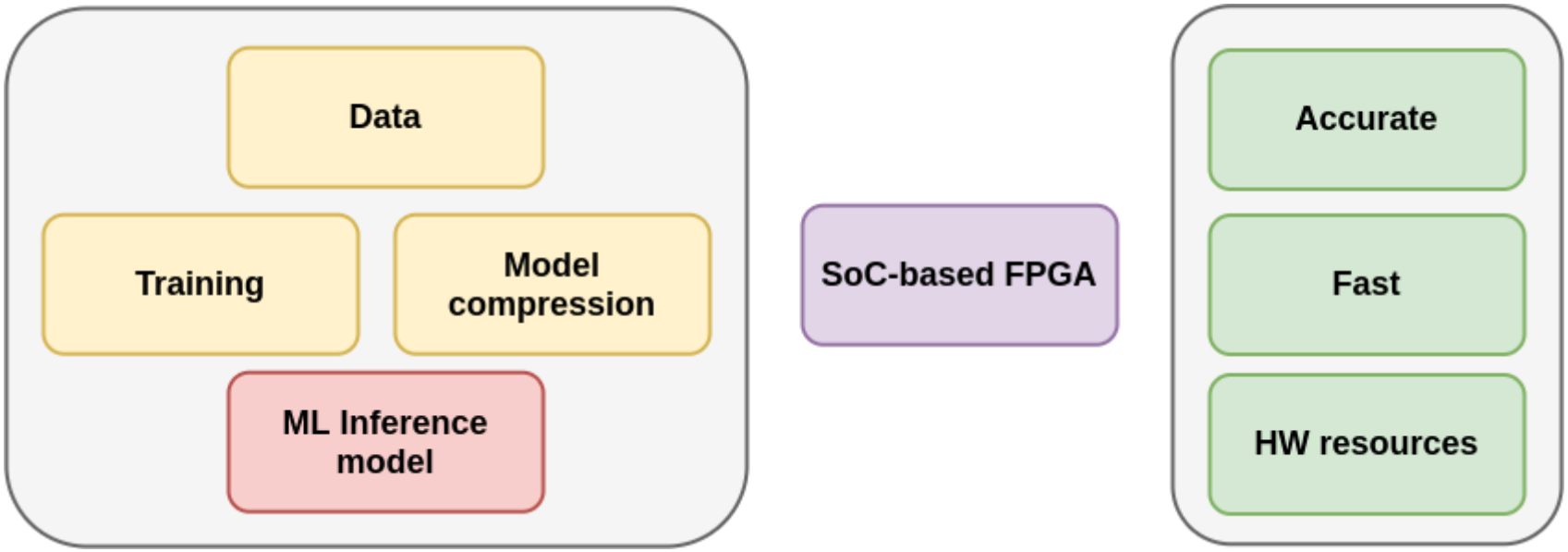
SoC-based FPGA



High level comparison of Zynq-7000 SoC and Zynq UltraScale+ MPSoC

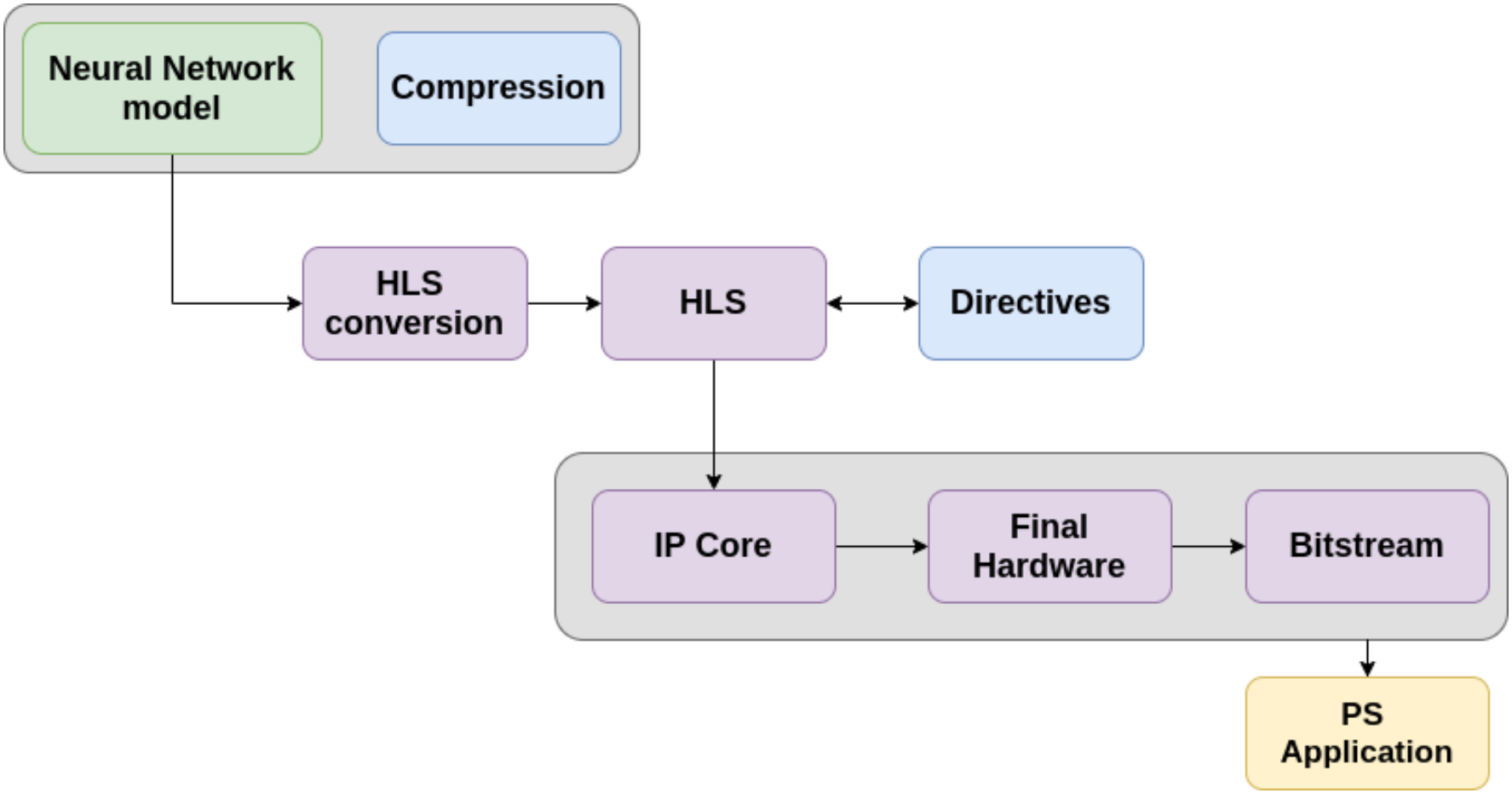
SoC-based FPGA

Machine learning and SoC



Acceleration of Machine Learning Inference

Acceleration of Machine Learning Inference



Acceleration of Machine Learning Inference

Considerations to map inference into FPGAs

- Clock frequency
- Bandwidth off-chip memory
- DSP, LUT, BRAM, FF
- Fixed point
- Power consumption

Acceleration of Machine Learning Inference

- Low-precision arithmetic to reduce power consumption and increase throughput.
- Reduce memory footprint – NN model can be deployed into on-chip memory, avoiding DDR access and bottlenecks.
- Model compression techniques [1]

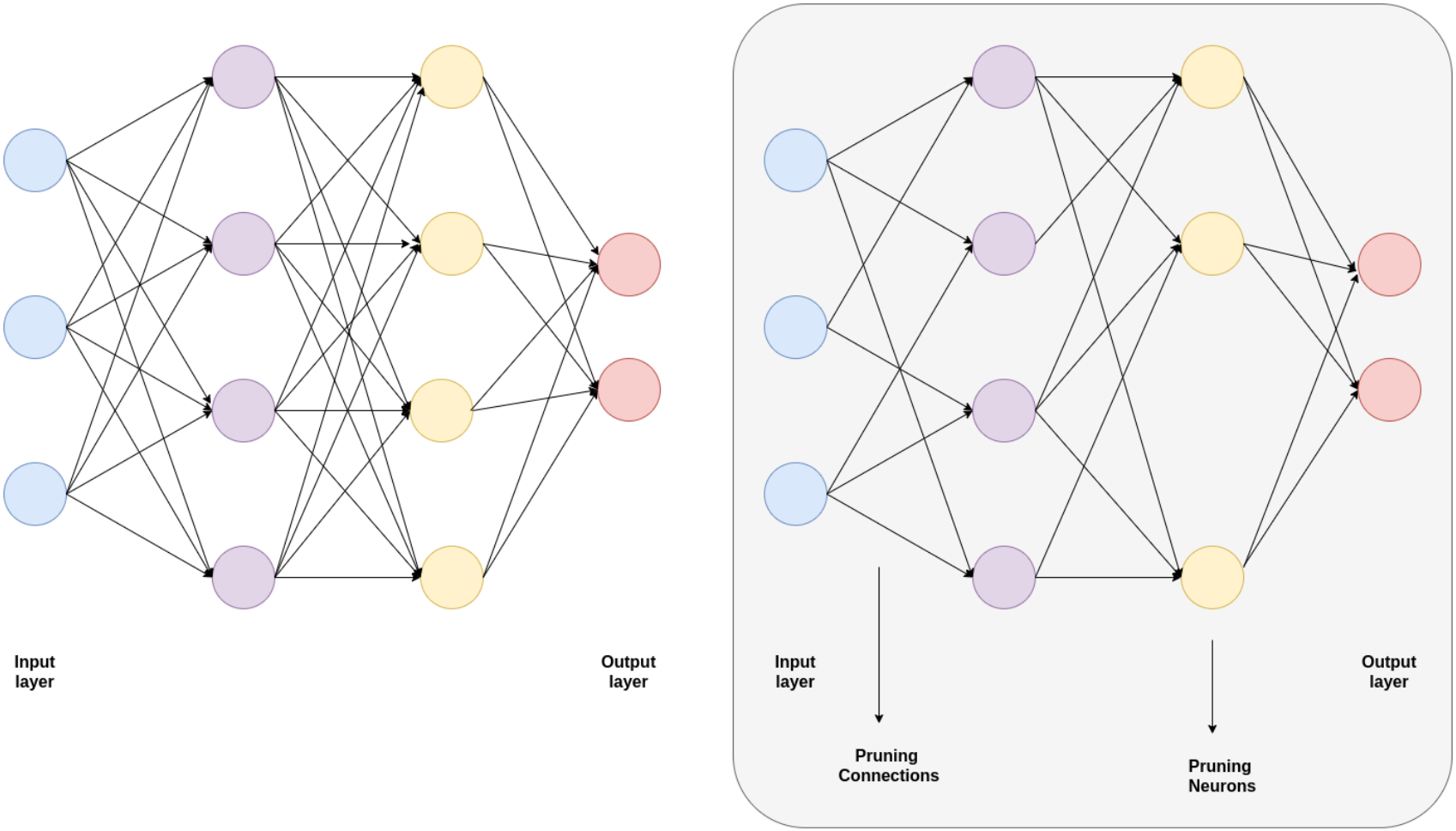
Acceleration of Machine Learning Inference

Model compression

- Quantization (Q) and pruning (P) (train from scratch and pre-trained model)
  - Q: Reduce number of bits to represent weights and bias
  - P: Remove connections and/or neurons
- Low-rank factorization (train from scratch and pre-trained model)
- Compact convolutional filters (train from scratch)
- Knowledge destillation (train from scratch)

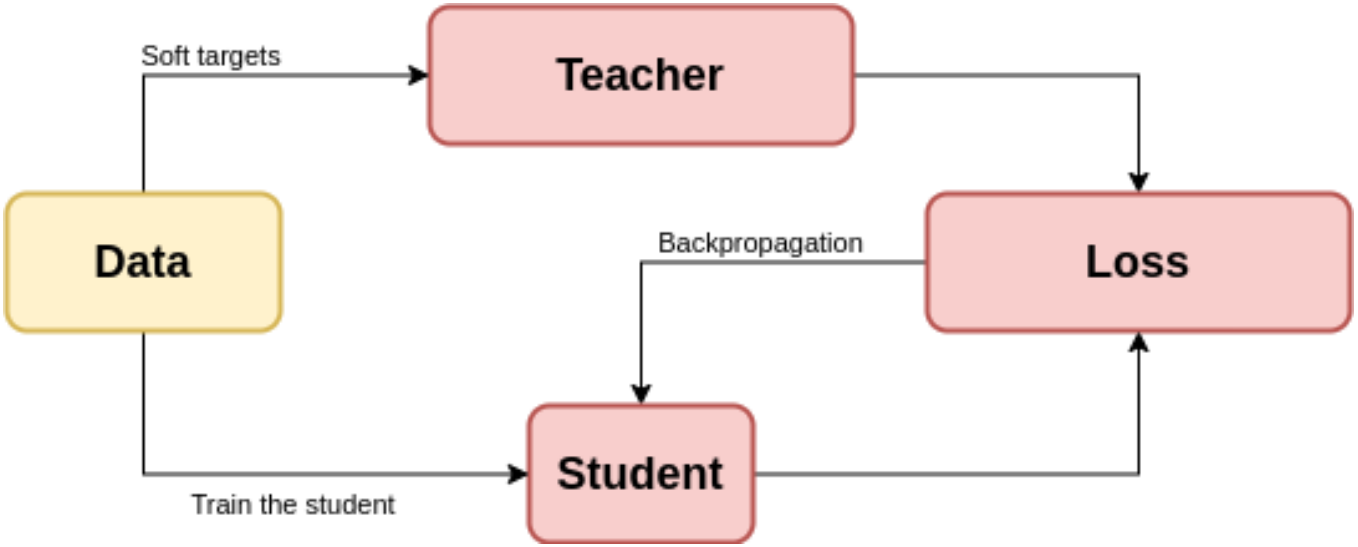
Acceleration of Machine Learning Inference

Model compression: Pruning



Left: Before pruning - Right: after pruning

Model compression: Knowledge distillation



High level diagram for knowledge distillation.

High-Level Synthesis

Hardware design - High-Level Synthesis

Vivado HLS:

- It provides the facility to create RTL from a high level of abstraction.
- It allows the optimization of the input code using directives to:
  - Reduce latency
  - Improve performance and throughput
  - Reduce resource utilization

Without directives, Vivado HLS will look minimize latency and improve concurrency

Acceleration of Machine Learning Inference

Hardware design - High-Level Synthesis

- **Minimize latency:** UNROLL, LOOP_FLATTEN, LOOP_MERGE.
- **Minimize throughput:** DATAFLOW, PIPELINE.
- **Improve bottleneck:** RESOURCE, ARRAY_PARTITION, ARRAY_RESHAPE.

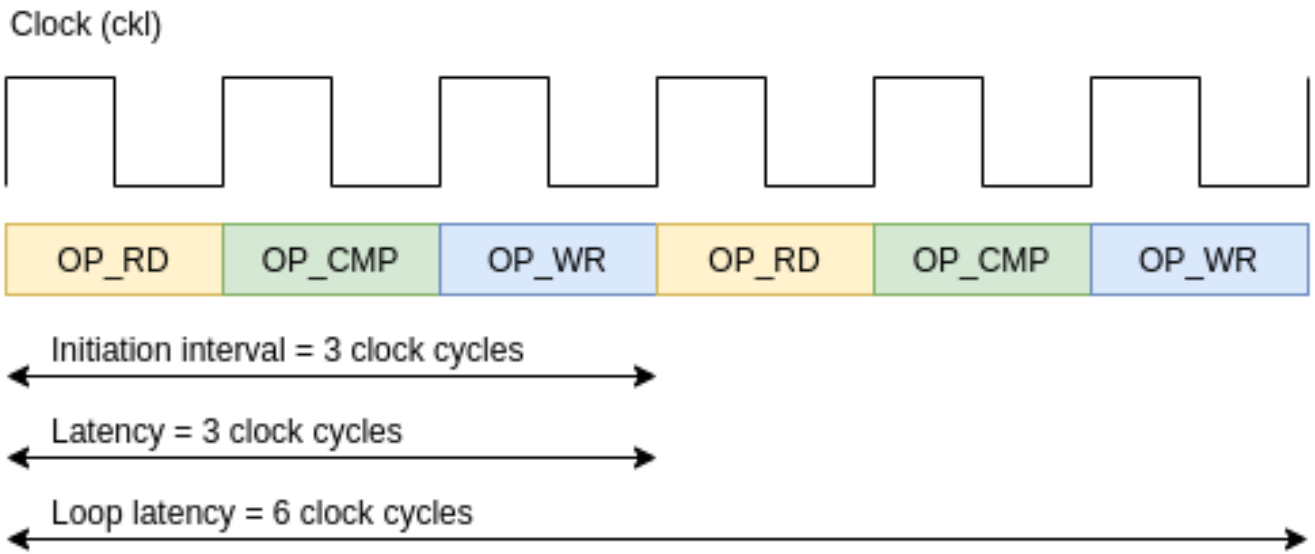
Hardware design - High-Level Synthesis - Directives

```
Loop_1: for(i=1; i<3; i++){
    OP_RD;
    OP_CMP;
    OP_WR;
}
```

OP_RD — Read

OP_CMP — Computation

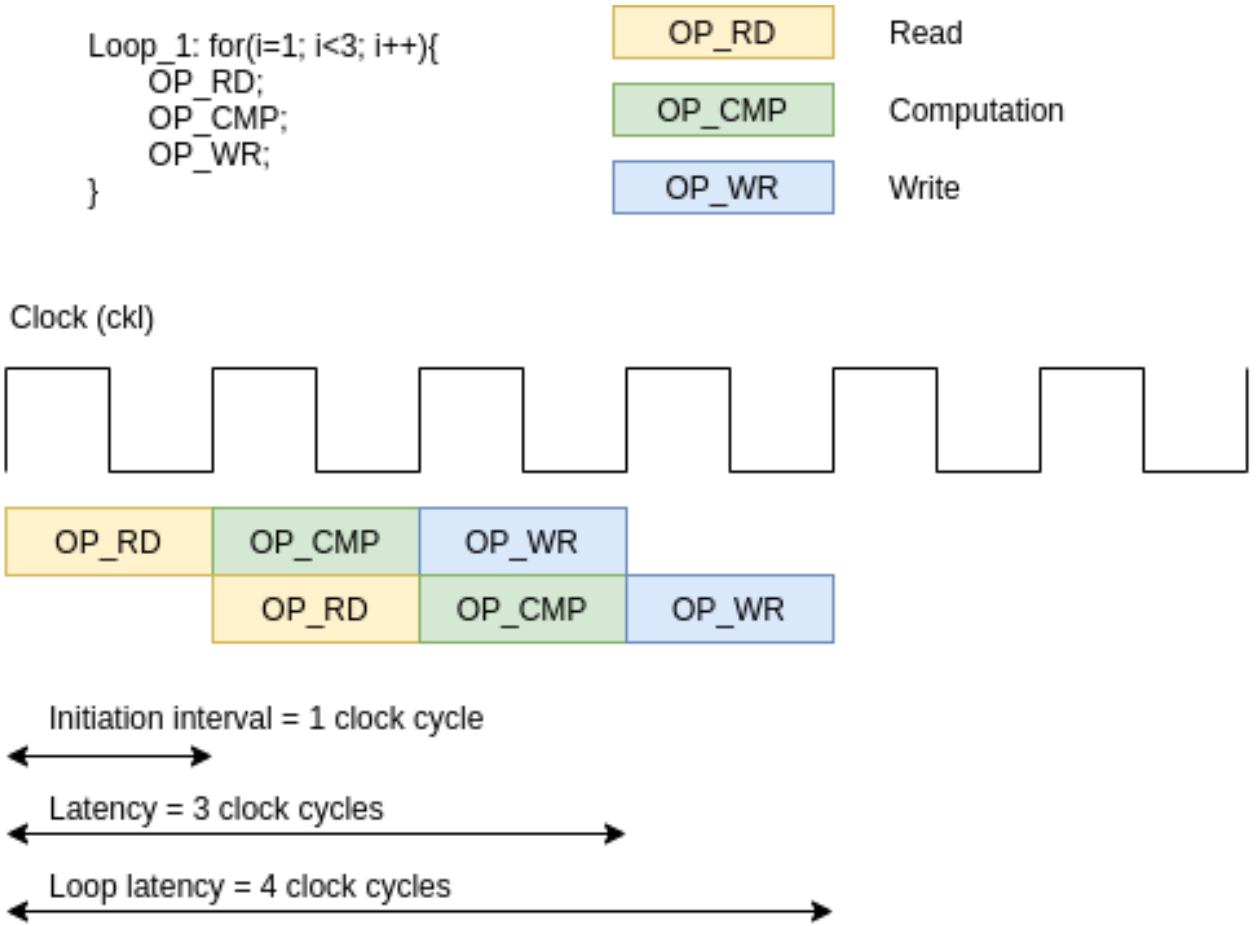
OP_WR — Write



Loop example

Hardware design - High-Level Synthesis - Directives

```
Loop_1: for(i=1; i<3; i++){
    OP_RD;
    OP_CMP;
    OP_WR;
}
```

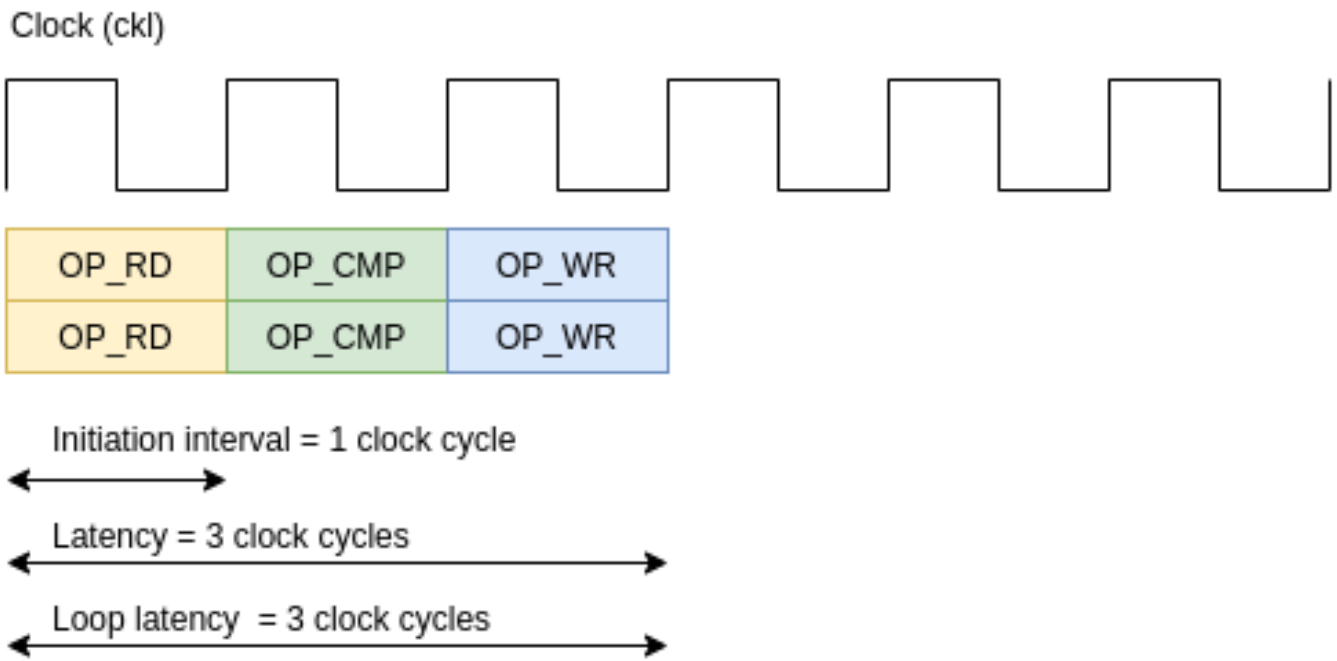


Loop + Pipeline directive

Hardware design - High-Level Synthesis - Directives

```
Loop_1: for(i=1; i<3; i++){
    OP_RD;
    OP_CMP;
    OP_WR;
}
```

OP_RD	Read
OP_CMP	Computation
OP_WR	Write



Loop + Unroll directive

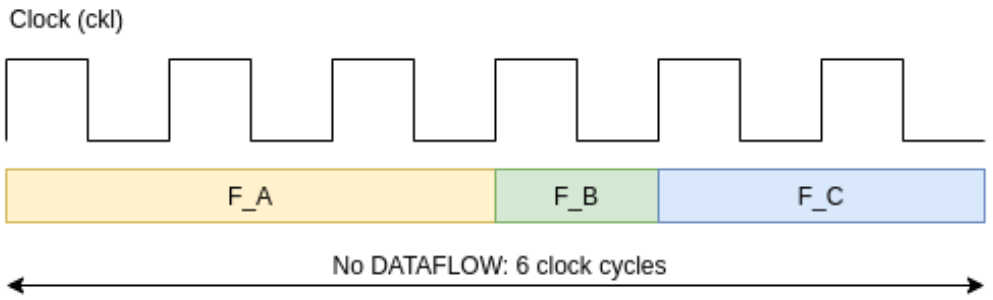
Hardware design - High-Level Synthesis - Directives

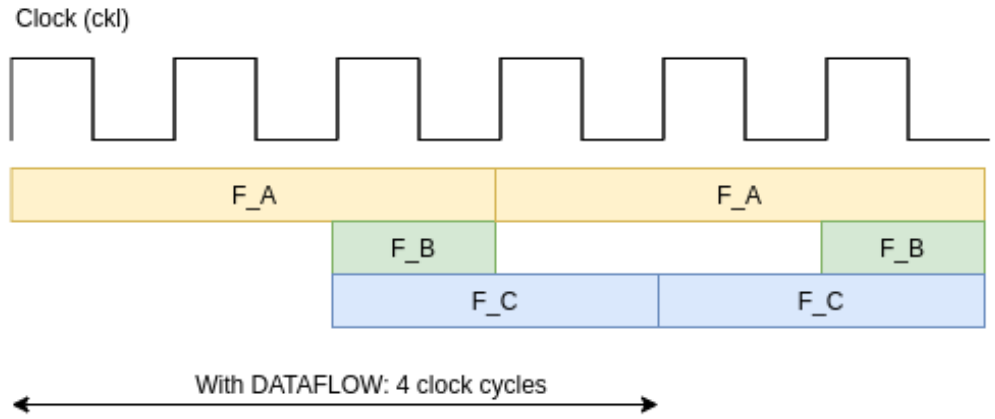
```
void function(a, b, c, d) {
    f_A(a, b, i1);
    f_B(c, i1, i2);
    f_C(i2, d);

return d;
}
```

F_A Function f_A(a, b, i1)

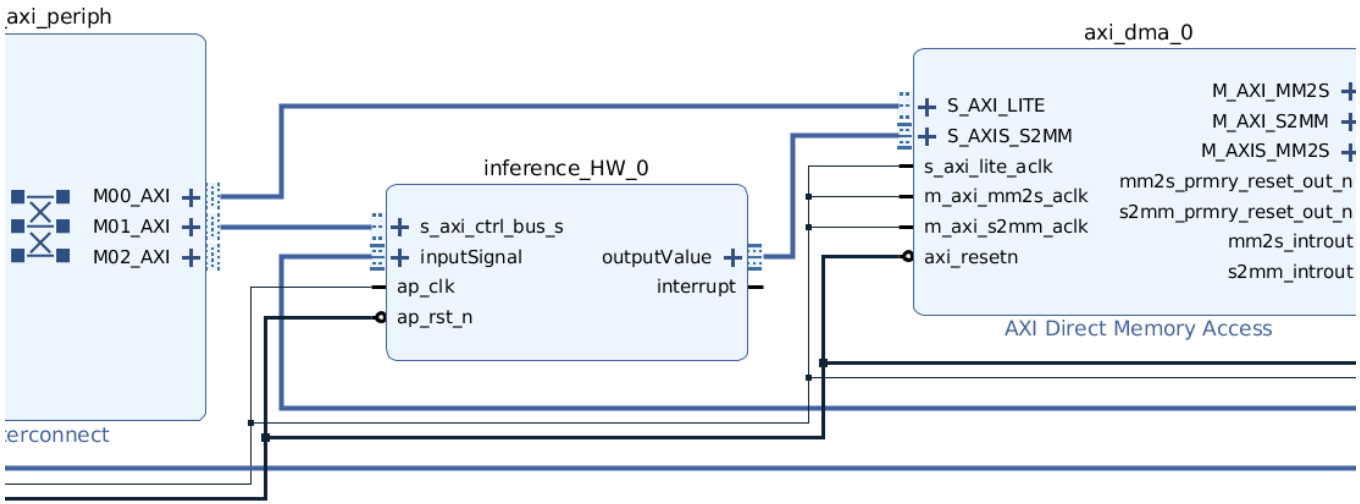
F_B Function f_B(c, i1, i2)

F_C Function f_C(i2, d)





Acceleration of Machine Learning Inference



Inference IP core

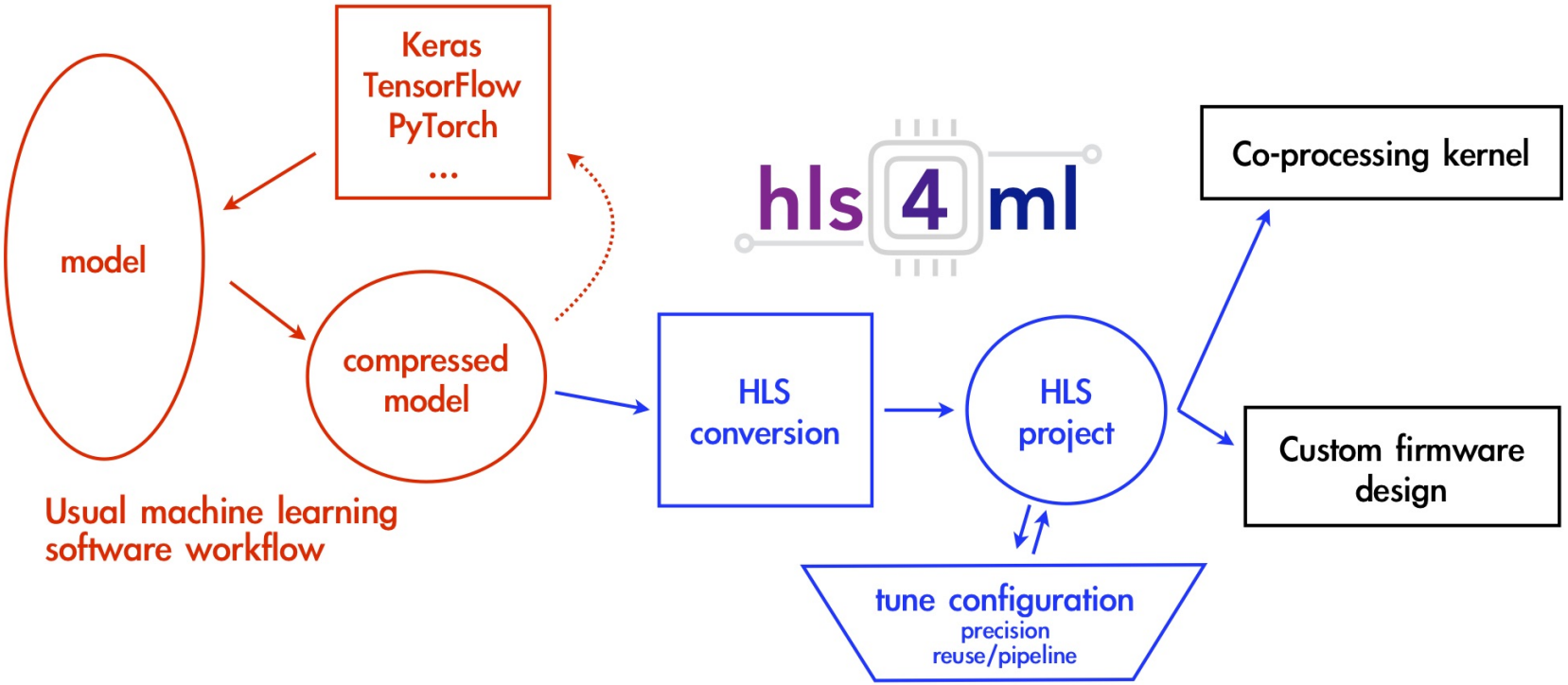
High-Level Synthesis for ML (hls4ml)

High level synthesis for machine learning (hls4ml)

- Package for ML inference in FPGAs using HLS. (Duarte et. al)
- "Fast inference of deep neural networks (DNN) in FPGAs for particle physics" Duarte et al. [2]
- GitHub: https://github.com/fastmachinelearning/hls4ml-tutorial
- https://fastmachinelearning.org/hls4ml/

High-Level Synthesis for ML (hls4ml)

hls4ml



Design flow for hls4ml [2]

High-Level Synthesis for ML (hls4ml)

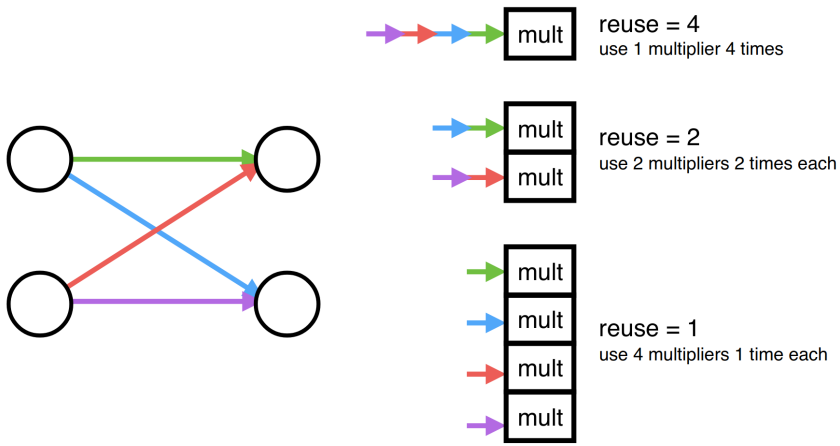
hls4ml

- HLS to create IP Core.
- Keras, TensorFlow, Pytorch.
- On-chip data structures.
- Quantization through ap_fixed in HLS.
- Trade-off between resoure utilization and latency/throughput.
- Integration with other tools.

High-Level Synthesis for ML (hls4ml)

Pipelining to speed up the process by accepting new inputs after an initiation interval.

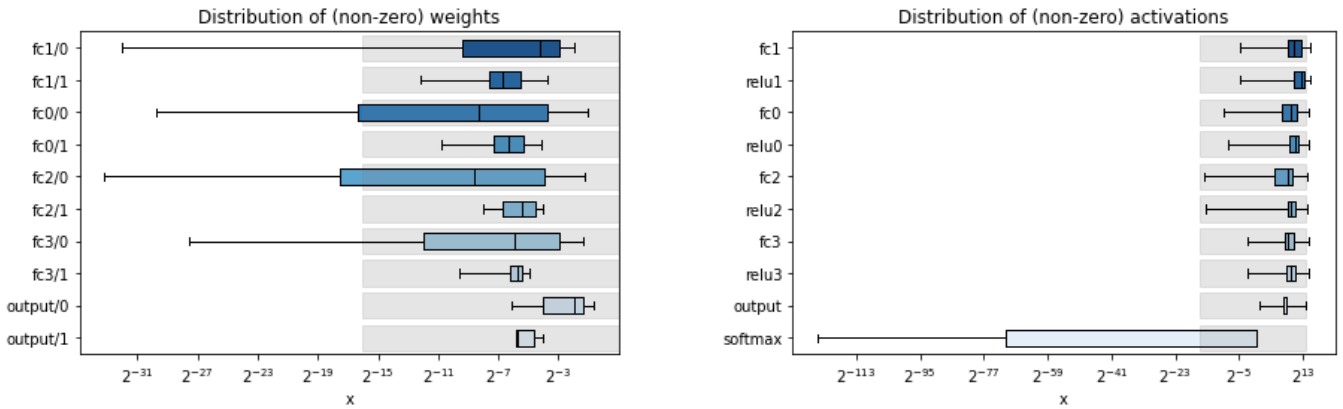
- Size/Compression
- Precision
- Dataflow/Resource Reuse
- Quantization Aware Training: Qkeras [3]



Reuse factor [2]

Profiling

Method: hls4ml.model.profiling.numerical



Profiling to adjust precision

High-Level Synthesis for ML (hls4ml)

How we start with the tool?

First, we have to download packages and dependencies. Then, we need to decide between:

- Using command line
- Using Jupyter Notebook

High-Level Synthesis for ML (hls4ml)

Using command line:

- Configuration file (.yml). In this case, the file has the name **model-config.yml**
- Files required: .json y .h5

```
# particles_keras_config.yml

# File json
KerasJson: ../model/model_architecture.json

# File h5
KerasH5:   ../model/model_weights.h5

#InputData:  ../model/modelInput.dat
#OutputPredictions:  ../modelPredictions.dat

OutputDir: particleHW
ProjectName: particlesIdentification
XilinxPart: xc7z020-clg484-1
ClockPeriod: 10
Backend: Vivado

IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,8>
    ReuseFactor: 1
#  LayerType:
#    Dense:
#      ReuseFactor: 2
#      Strategy: Resource
#      Compression: True
# xczu9eg-ffvb1156-2-e  xc7z020clg484-1
```

High-Level Synthesis for ML (hls4ml)

Commands for terminal execution

- hls4ml convert -c model-config.yml
- hls4ml build -p ProjectName -a
- vivado_hls -f ProjectName.tcl "csim=1 synth=1 cosim=0 export=0"

Following the information in the previous image, **ProjectName** was replaced by **particlesIdentification**:

- hls4ml convert -c model-config.yml
- hls4ml build -p particlesIdentification -a
- vivado_hls -f particlesIdentification.tcl "csim=1 synth=1 cosim=0 export=0"

Using Jupyter Notebook:

```
1 from tensorflow.keras.models import load_model
2 from sklearn.metrics import accuracy_score
3 model = load_model('model_keras_MLP.h5')
4 model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 60)                3900
_________________________________________________________________
relu1 (Activation)           (None, 60)                0
_________________________________________________________________
fc0 (Dense)                  (None, 40)                2440
_________________________________________________________________
relu0 (Activation)           (None, 40)                0
_________________________________________________________________
fc2 (Dense)                  (None, 30)                1230
_________________________________________________________________
relu2 (Activation)           (None, 30)                0
_________________________________________________________________
fc3 (Dense)                  (None, 10)                310
```

Model summary

High-Level Synthesis for ML (hls4ml)

Using Jupyter Notebook:

```
import hls4ml
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                                       hls_config=config,
                                                       output_dir='model1/MLP',
                                                       fpga_part='xczu9eg-ffvb1156-2-e')


hls_model.compile()
```

High-Level Synthesis for ML (hls4ml)

Generated code

```
63
64      layer3_t layer3_out[N_LAYER_3];
65      #pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
66      nnet::dense_latency<input2_t, layer3_t, config3>(input1, layer3_out, w3, b3);
67
68      layer5_t layer5_out[N_LAYER_3];
69      #pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
70      nnet::relu<layer3_t, layer5_t, relu_config5>(layer3_out, layer5_out);
71
72      layer6_t layer6_out[N_LAYER_6];
73      #pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
74      nnet::dense_latency<layer5_t, layer6_t, config6>(layer5_out, layer6_out, w6, b6);
75
76      layer8_t layer8_out[N_LAYER_6];
77      #pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
78      nnet::relu<layer6_t, layer8_t, relu_config8>(layer6_out, layer8_out);
79
80      layer9_t layer9_out[N_LAYER_9];
81      #pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
82      nnet::dense_latency<layer8_t, layer9_t, config9>(layer8_out, layer9_out, w9, b9);
83
84      layer11_t layer11_out[N_LAYER_9];
85      #pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
86      nnet::relu<layer9_t, layer11_t, relu_config11>(layer9_out, layer11_out);
87
88      layer12_t layer12_out[N_LAYER_12];
89      #pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
90      nnet::dense_latency<layer11_t, layer12_t, config12>(layer11_out, layer12_out, w12, b12);
91
92      layer14_t layer14_out[N_LAYER_12];
93      #pragma HLS ARRAY_PARTITION variable=layer14_out complete dim=0
94      nnet::relu<layer12_t, layer14_t, relu_config14>(layer12_out, layer14_out);
95
96      layer15_t layer15_out[N_LAYER_15];
97      #pragma HLS ARRAY_PARTITION variable=layer15_out complete dim=0
98      nnet::dense_latency<layer14_t, layer15_t, config15>(layer14_out, layer15_out, w15, b15);
99
100     nnet::softmax<layer15_t, result_t, softmax_config17>(layer15_out, layer17_out);
101
```

High-Level Synthesis for ML (hls4ml)

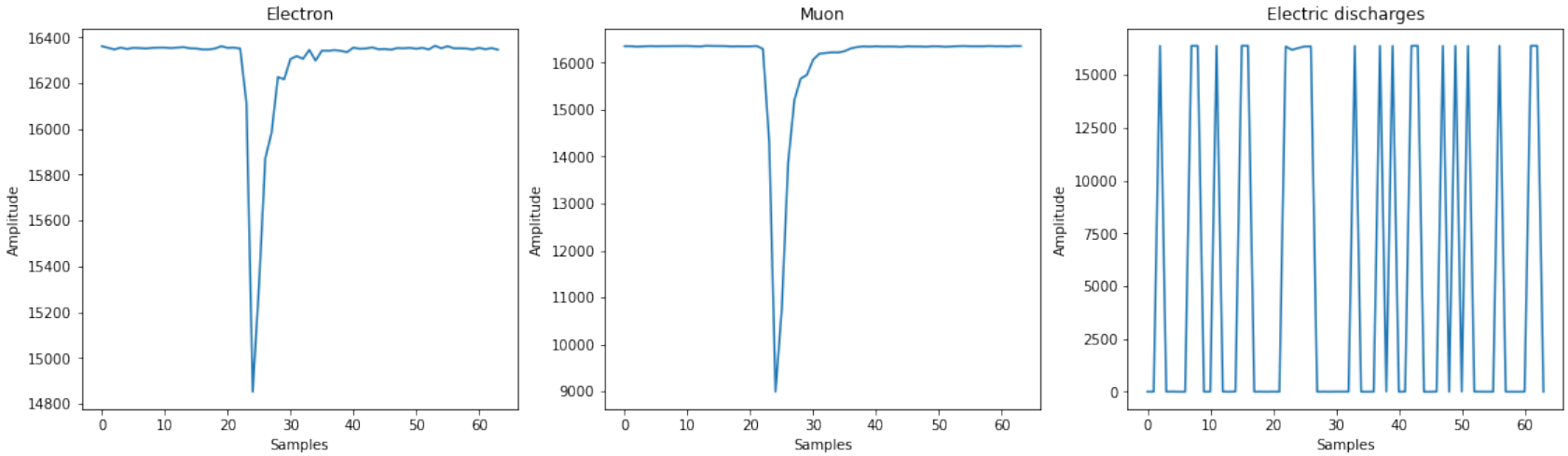
Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors

High-Level Synthesis for ML (hls4ml)

Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors [4]

- Water Cherenkov detector (WCD) at the Escuela de Ciencias Físicas y Matemáticas in Universidad de San Carlos de Guatemala (ECFM-USAC).
- Signal: sampled at 125 MHz, 14-bit resolution.
- Feature extraction in the incoming signal to obtain signal classification.

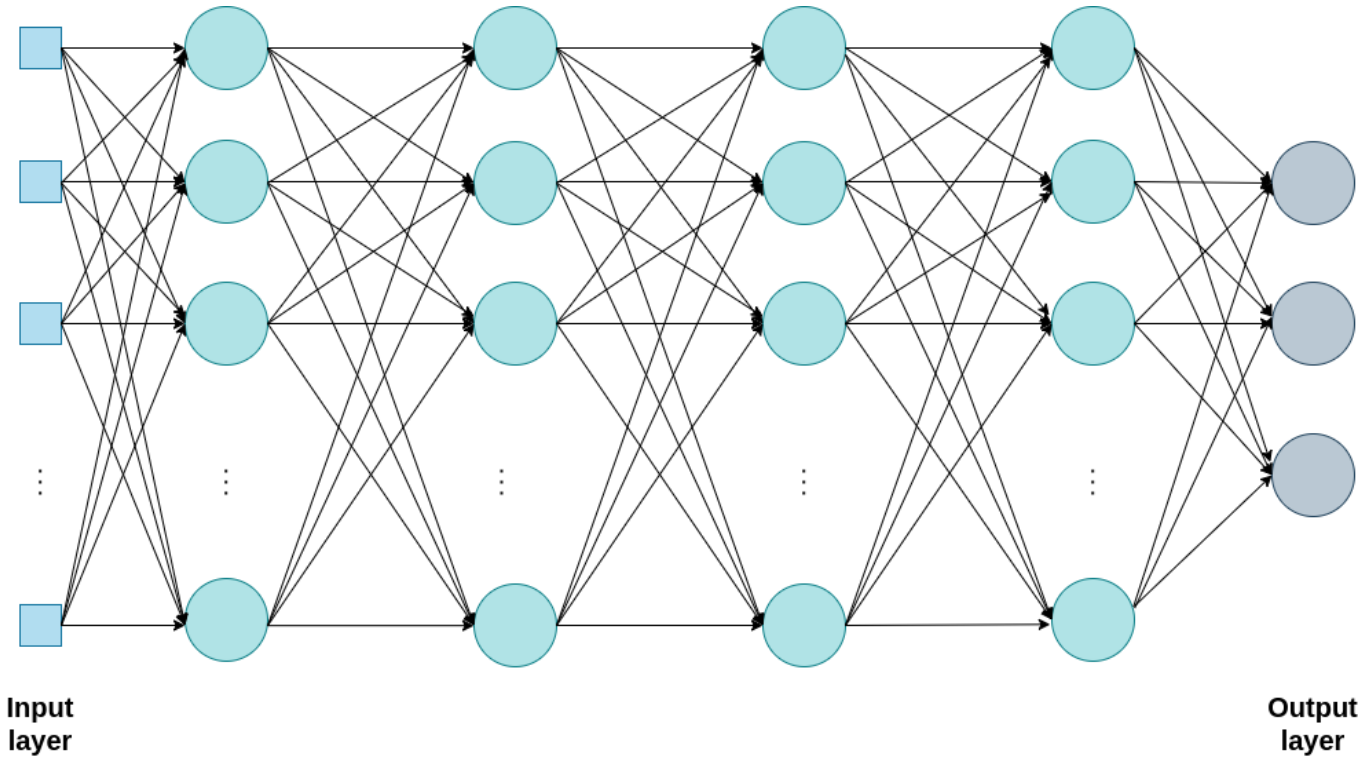
Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors



Different types of signals: electron, muon and electric discharges

High-Level Synthesis for ML (hls4ml)

Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors



MLP architecture

Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors

```
: 1 model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 60)                3900
_________________________________________________________________
relu1 (Activation)           (None, 60)                0
_________________________________________________________________
fc0 (Dense)                  (None, 40)                2440
_________________________________________________________________
relu0 (Activation)           (None, 40)                0
_________________________________________________________________
fc2 (Dense)                  (None, 30)                1230
_________________________________________________________________
relu2 (Activation)           (None, 30)                0
_________________________________________________________________
fc3 (Dense)                  (None, 10)                310
_________________________________________________________________
relu3 (Activation)           (None, 10)                0
_________________________________________________________________
output (Dense)               (None, 3)                 33
_________________________________________________________________
softmax (Activation)         (None, 3)                 0
=================================================================
Total params: 7,913
Trainable params: 7,913
Non-trainable params: 0
_________________________________________________________________
```

Model summary

Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors

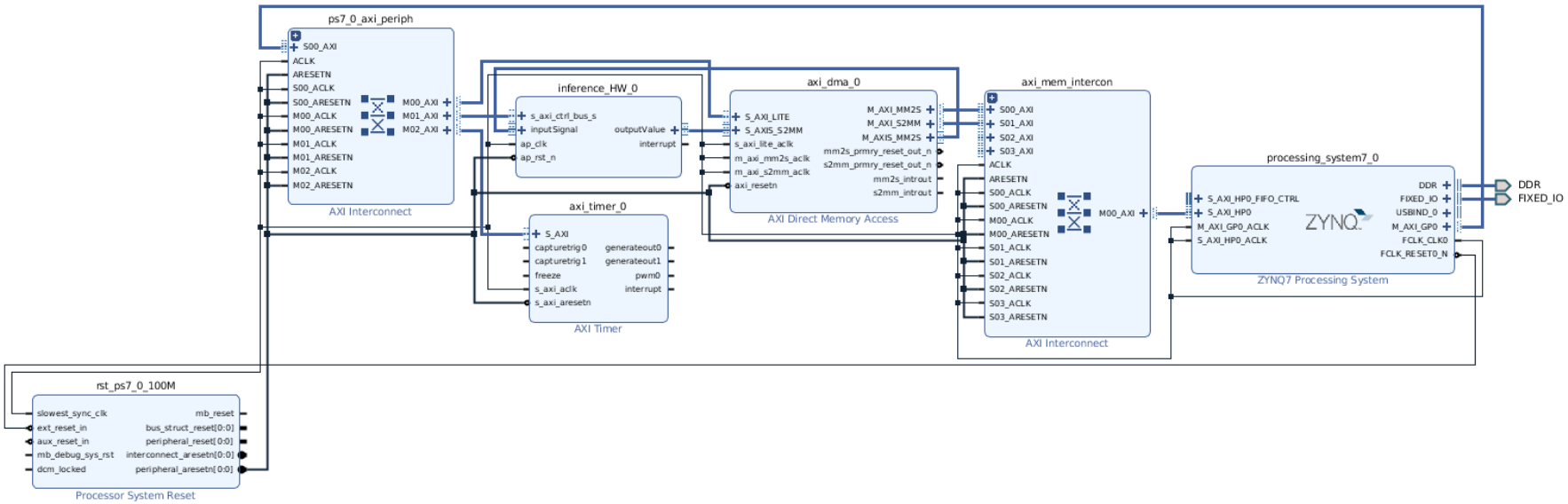
HLS reports comparison. Solutions 1 and 5 without directives. Solutions 2 and 6 with directives applied by hls4mland Softmax as activation function. Solutions 3 and 7 with directives applied by hls4ml, PIPELINE to improve the interval,without Softmax and with a reuse factor of 1 for all the layers. Solutions 4 and 8 with directives applied by hls4ml, PIPELINEto improve the interval, without Softmax and with a reuse factor of 8 for all the dense layers

Solution	Directives	Estimated Clock [ns]	Clock Cycles	Inference Clock Cycles	Interval	BRAM	DSP	FF	LUT
ZU9EG									
1	No	4.653	36,917	36,848	36,917	23	2	2407	5732
2	Yes + Softmax	4.653	18,526	18,457	18,526	2	1245	26,192	180,066
3	Yes + NS + RF: 1	4.251	84	19	64	0	1235	27221	167,158
4	Yes + NS + RF: 8	4.993	115	50	64	0	155	38,571	141,443
XC7Z020									
5	No	6.508	91,777	91,707	91,777	23	2	4313	6952
6	Yes + Softmax	6.508	40,063	39,993	40,063	2	1245	188,626	171,599
7	Yes + NS + RF: 1	4.350	121	55	64	0	1235	189,059	159,351
8	Yes + NS + RF: 8	5.561	143	77	64	0	155	76,286	118,936

High-Level Synthesis for ML (hls4ml)

Case of study: Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors

Hardware creation with Vivado IP Integrator



References

1 Duarte, J.; Han, S.; Harris, P.; Jindariani, S.; Kreinar, E.; Kreis, B.; Ngadiuba, J.; Pierini, M.; Rivera, R.; Tran, N.; et al. Fast inference of deep neural networks in FPGAs for particle physics. J. Instrum. 2018, 13, P07027, doi:10.1088/1748-0221/13/07/p07027.

2 Cheng, Y.; Wang, D.; Zhou, P.; Zhang, T. A Survey of Model Compression and Acceleration for Deep Neural Networks. arXiv 2017, arXiv:1710.09282

3 Coelho, J.; Kuusela, A.; Zhuang, H.; Aarrestad, T.; Loncar, V.; Ngadiuba, J.; Pierini, M.; Summers, S. Ultra Low-latency, Low-area Inference Accelerators using Heterogeneous Deep Quantization with QKeras and hls4ml. arXiv 2020, arXiv:2006.10159.

4 Garcia, L.G.; Molina, R.S.; Crespo, M.L.; Carrato, S.; Ramponi, G.; Cicuttin, A.; Morales, I.R.; Perez, H. Muon–Electron Pulse Shape Discrimination for Water Cherenkov Detectors Based on FPGA/SoC. Electronics 2021, 10, 224. https://doi.org/10.3390/electronics10030224

5 Vivado Design Suite User Guide - High-Level Synthesis - UG902 (v2019.1) July 12, 2019

Mg. Romina S. Molina

ICTP
rmolina@ictp.it

Università degli Studi di Trieste
rominasoledad.molina@phd.units.it

National University of San Luis:
rsmolina@unsl.edu.ar

Questions?

Thanks!