Deep Learning Inference with FPGA

Corso INFN “Tecniche Di Machine Learning Con Dispositivi FPGA per Gli Esperimenti Di Fisica Delle Particelle”

Riccardo Travaglini - 02/11/2022

Disclaimer:
This presentation is intended only for personal use of this course participants;
Do not distribute - dot not modify
Use only for education purposes

The ML/AI big picture

- Training and Inference
- Several techniques:
    - BDT, Random forest, DNN, CNN, RNN, …
- Need to accelerate processing and get a lower power consumption

- GPU are best suited for training
- FPGA only used for inference acceleration - best for:
    - Low latency (can be also fixed and reproducible)
    - Low power (not true for floating point arithmetic)
    - Supported architectures (depends on the tool you are using)
        - mainly based on NN: DNN, CNN, RNN, Autoencoders, … (I can be outdated!)

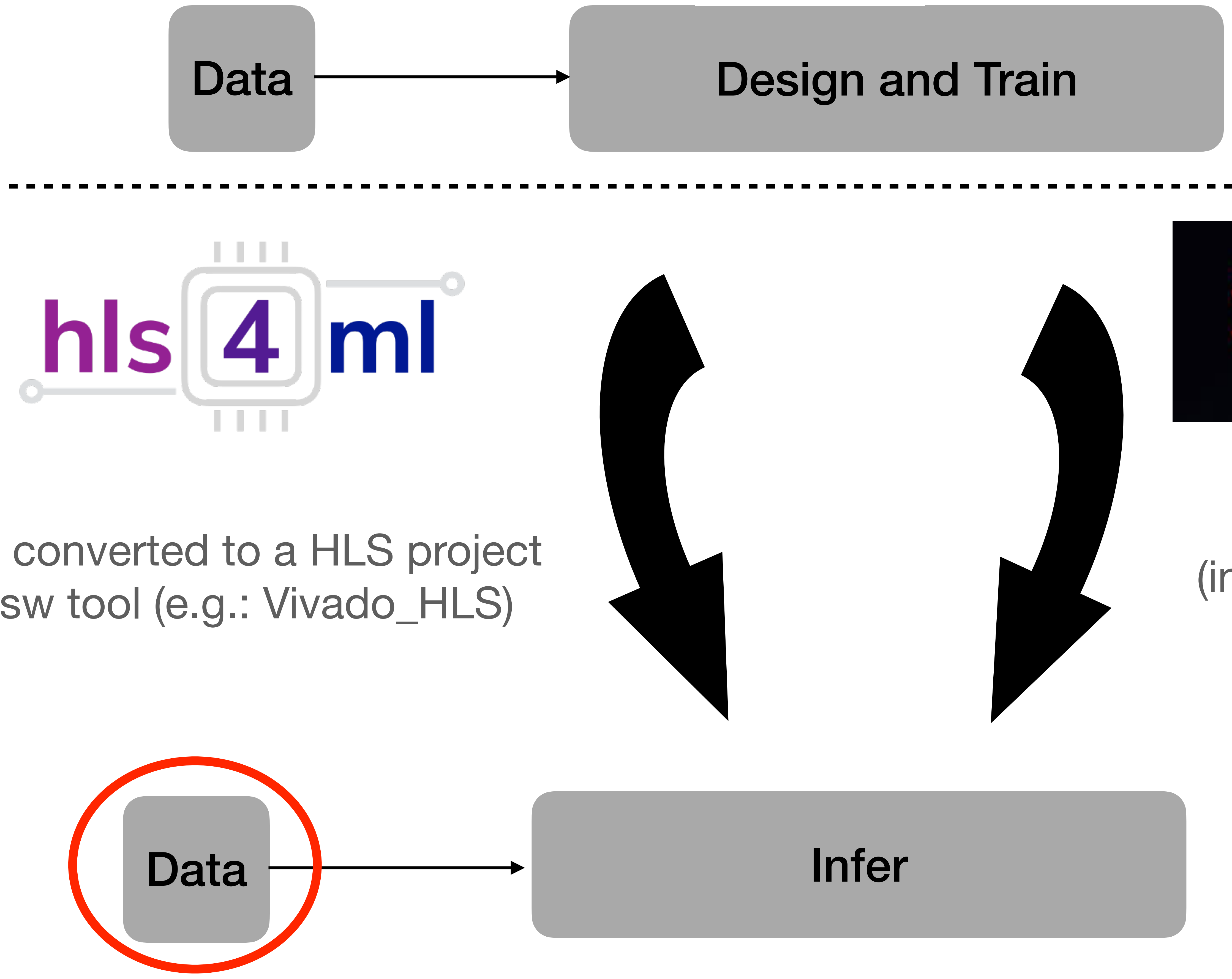
From algorithm to FPGA

Two typical alternative flows

"FPGA agnostic"

TensorFlow

PyTorch



High Level Synthesis Approach

hls4ml

Trained DNN is converted to a HLS project for a specific sw tool (e.g.: Vivado_HLS)

XILINX VITIS AI

IP cores based Approach

Trained DNN is converted to model (instruction list, graph, ..) to be executed on a specific co-processor Implemented on Fabric

"FPGA dependant"

High Level Synthesis approach

- Given a chosen ML framework: <100% can be supported for HLS translation
- Synthesis and P&R must be performed
    - Usually takes longer on big FPGA needed for AI/ML
- The model (f.i. the CNN) must fit into the FPGA resources (Multi-FPGA requires tools still in development)
- Requires (mostly custom) interface with the FPGA I/O for data
- Weights can be hardcoded into the FPGA
- Model performance can be estimated; they are measurable and reproducible

IP core based approach

- Given a chosen ML framework: <100% can be supported for HLS translation
- Synthesis and P&R not needed
    - Platform with IP cores already available or can be done once (“overlay")
- Replica of the model can be executed in parallel just like “threads” (depends on the IP core “occupancy”)
- Model computed like executing instruction on a co-processor
- Data and weights dynamically loaded to the IP-core (typically IO interface through software), limited by supported formats (e.g. batch limitations for CNN inputs)
- Can be driven by CPU: platform/overlay available
- Performance can be difficult to estimate - profiling tools can be available

FPGA Constraints impact

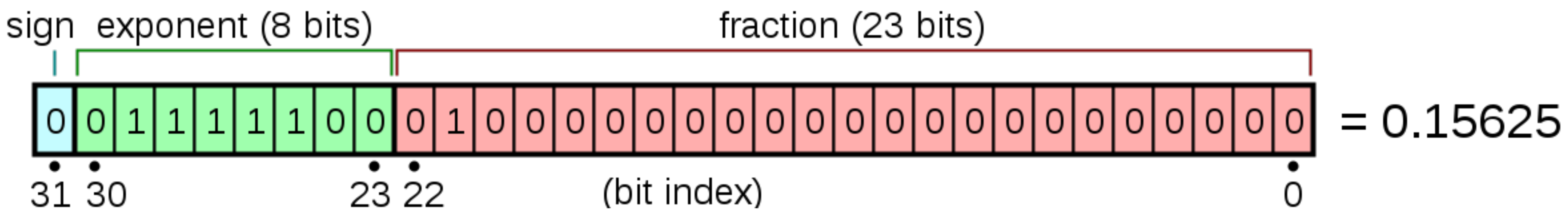
FPGA have limited resources

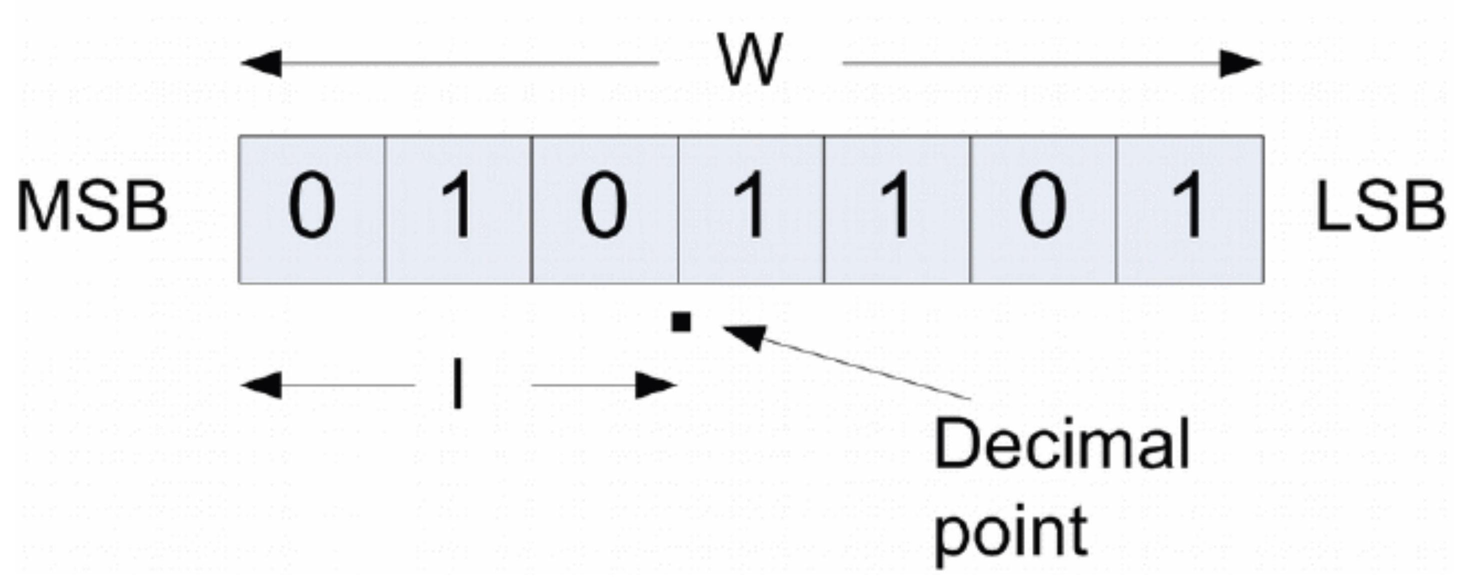
- Floating-point (single precision too) adder and multiplier are resource hungry
- Nets with high fan-out have bad impact on timing (slow max. Clock frequency)

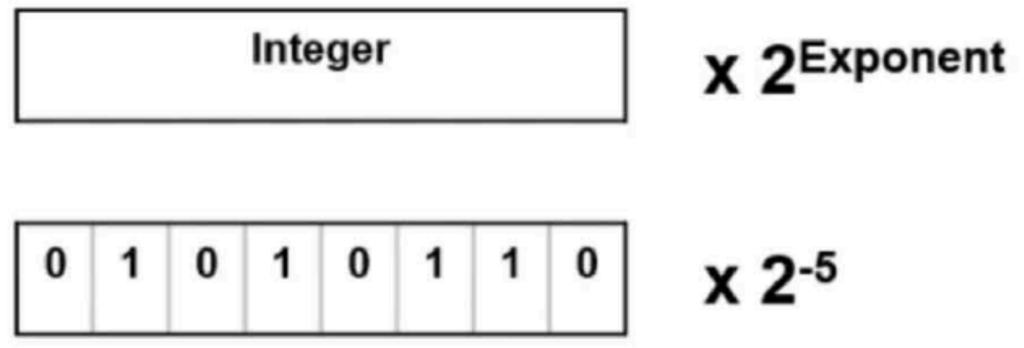
Number representation

Floating - fixed point - integer - int8

- Positional notation: bit i contributes $2^{\pm I}$
- Signed/unsigned
- Most used in FPGA: arbitrary length fixed point, int8
- TensorFlow Lite Int8 with exponent (i.e. scale) ; a common offset can be provided







Exponent is shared (fixed) for a set of variables

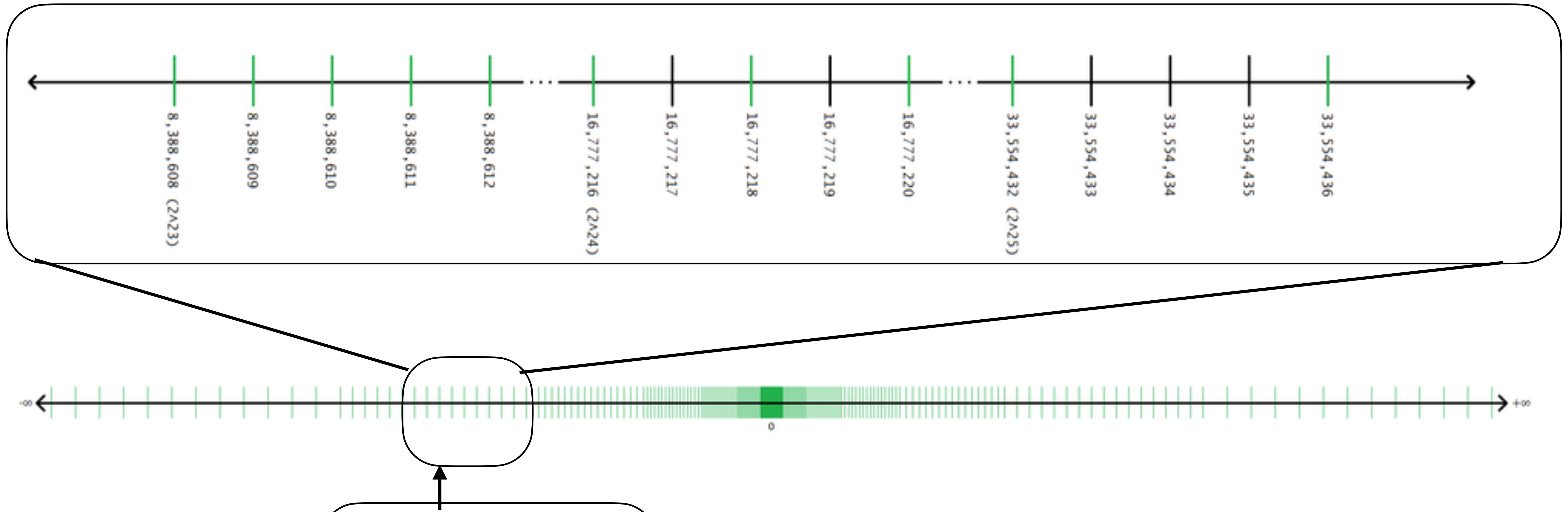
Saturation and precision

Quantization

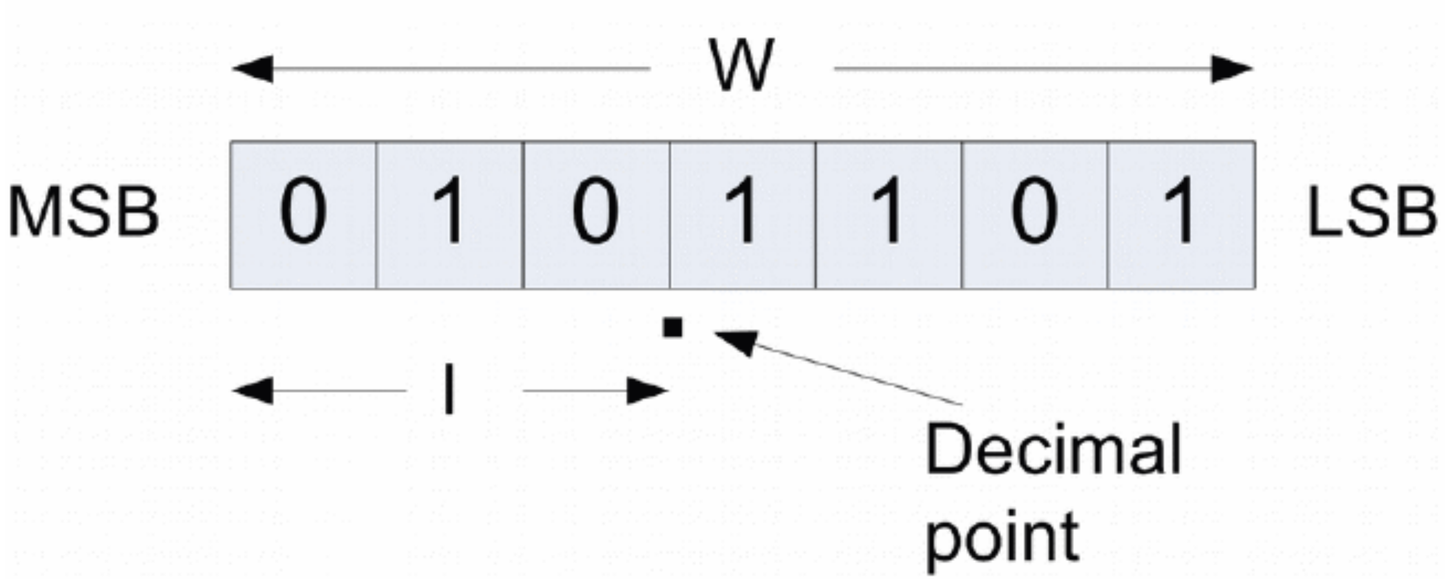
- Fixex point with integer I bits is limited to ~ 2^I

- I-bits unsigned integer limited to 2^I-1

- Signed range from ~-2^{I-1} to ~ 2^{I-1}



Floating points have variable precision
Fixed point and integer have constant precision



Quantization

- Design in full precision (typically FP32) e quantize in inference (possibly scaling)
- Quantization-aware training
- Quantization can be global, per layer (NN), per channel (CNN)

- Several tools available
- See hands-on for some examples

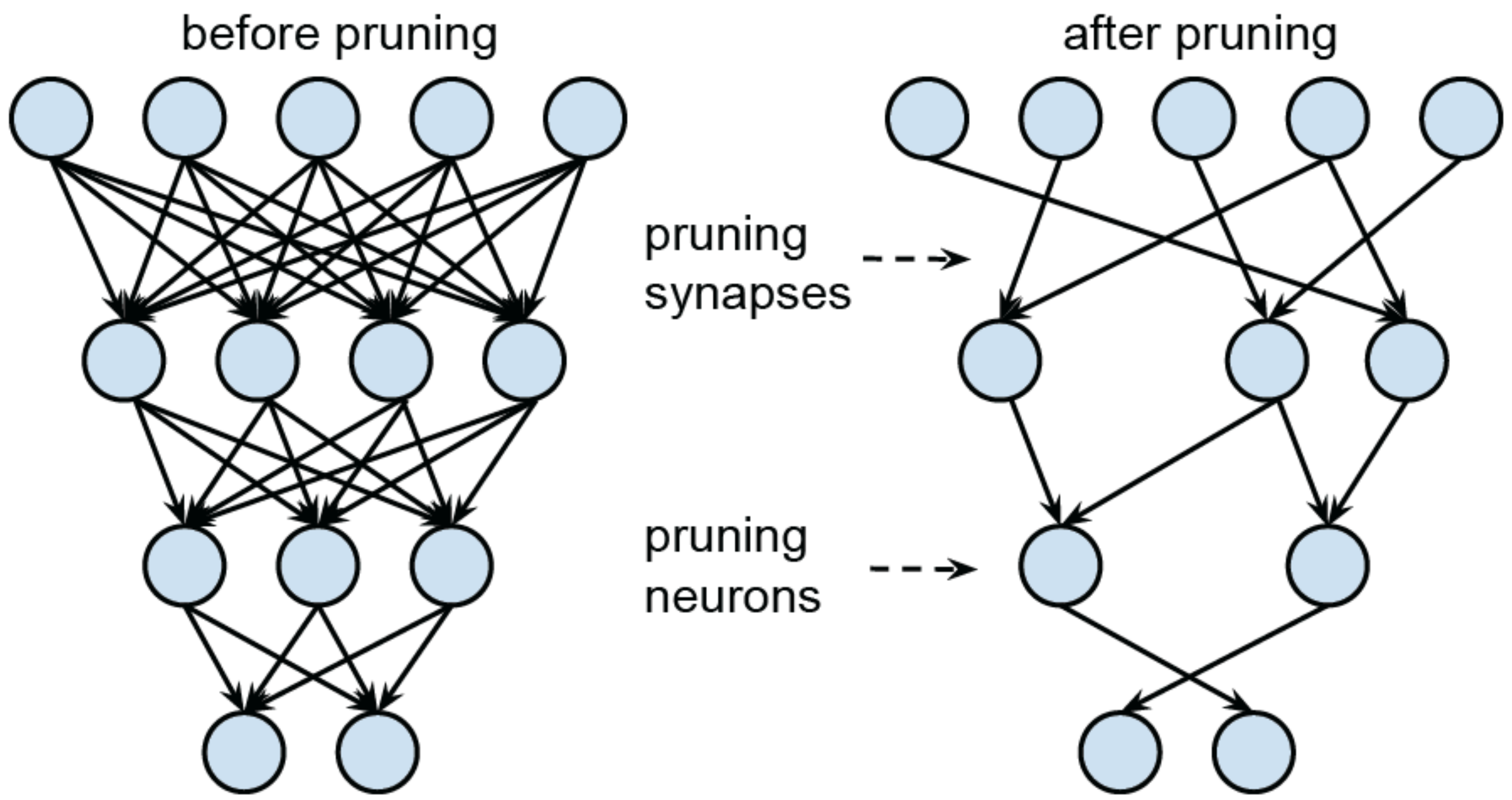
FPGA Constraints impact

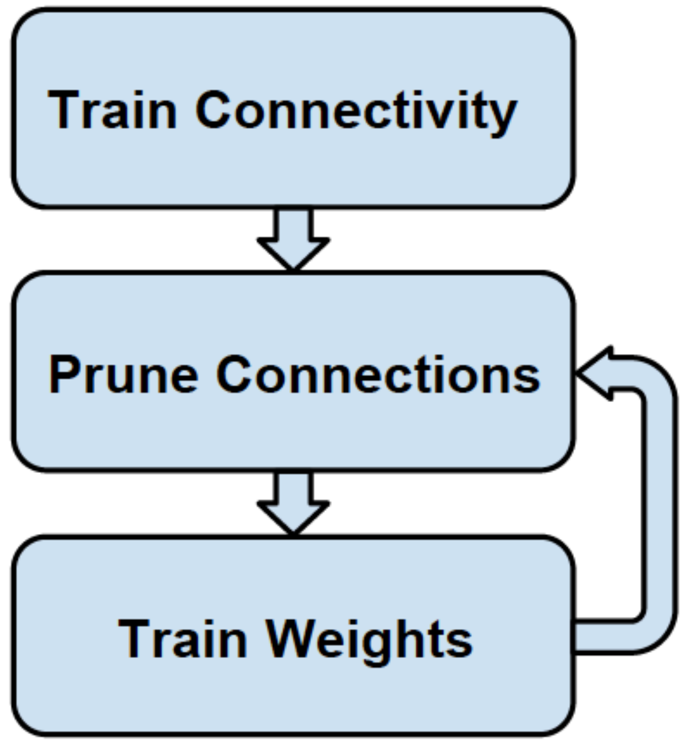
FPGA have limited resources

- Floating-point (single precision too) adder and multiplier are resource hungry
- Nets with high fan-out have bad impact on timing (slow max. Clock frequency)

Pruning

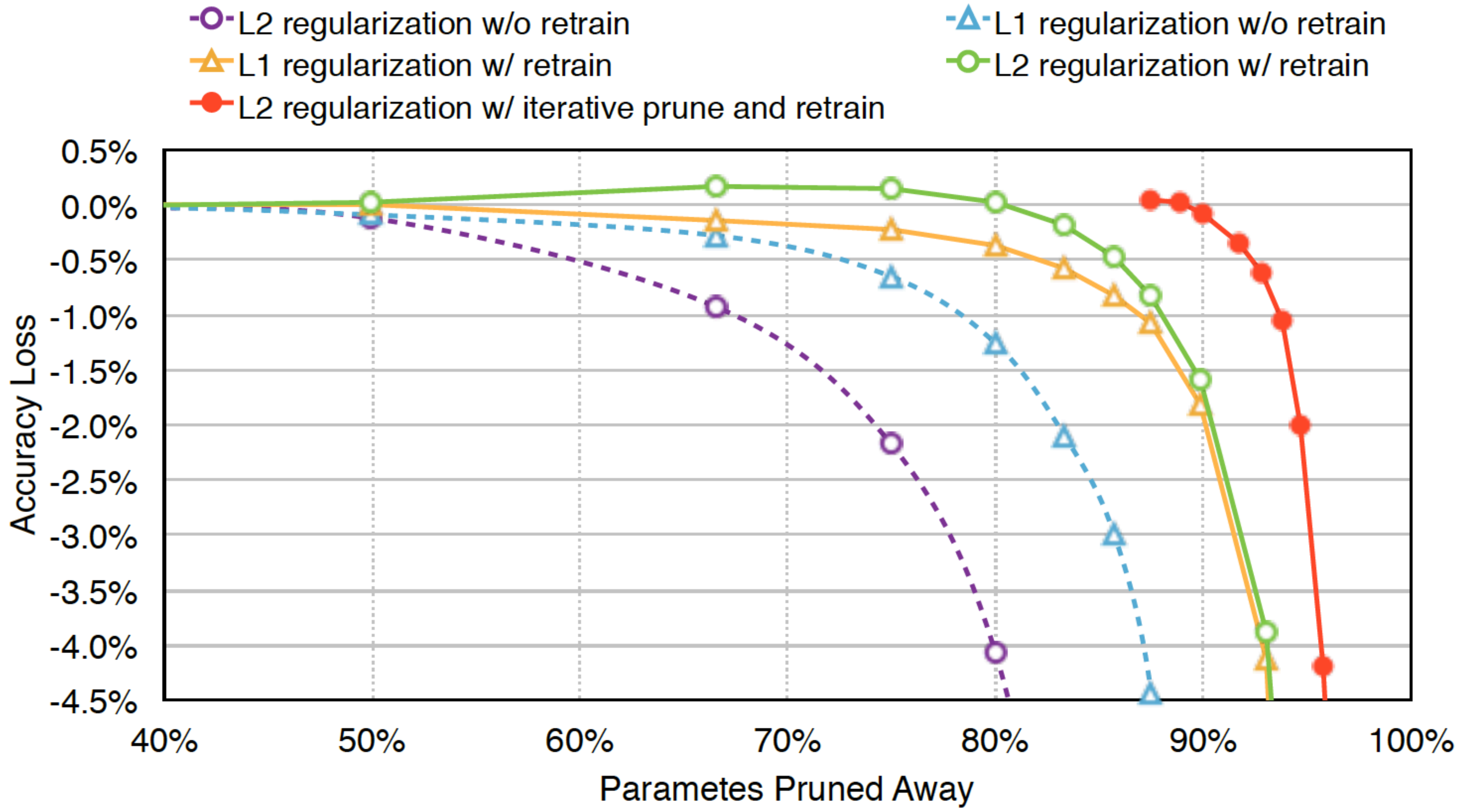
Resource optimization




Train Connectivity

Prune Connections

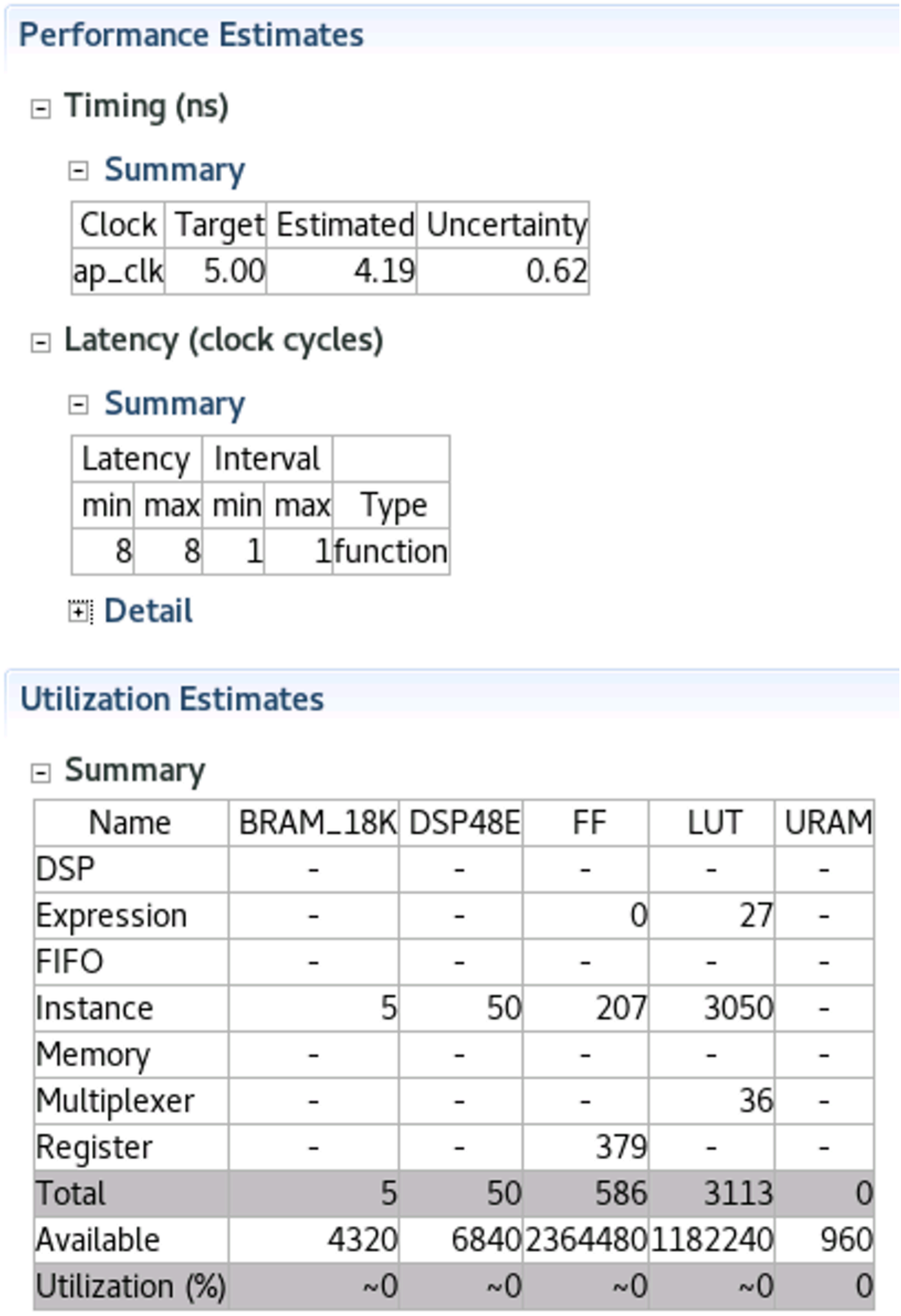
Train Weights



Learning both Weights and Connections for Efficient Neural Networks
Song Han, Jeff Pool, John Tran, William J. Dally
arXiv:1506.02626 [cs.NE]

Resource and performance estimation

- HLS-based



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	5.00	4.19	0.62

⊟ Latency (clock cycles)

⊟ Summary

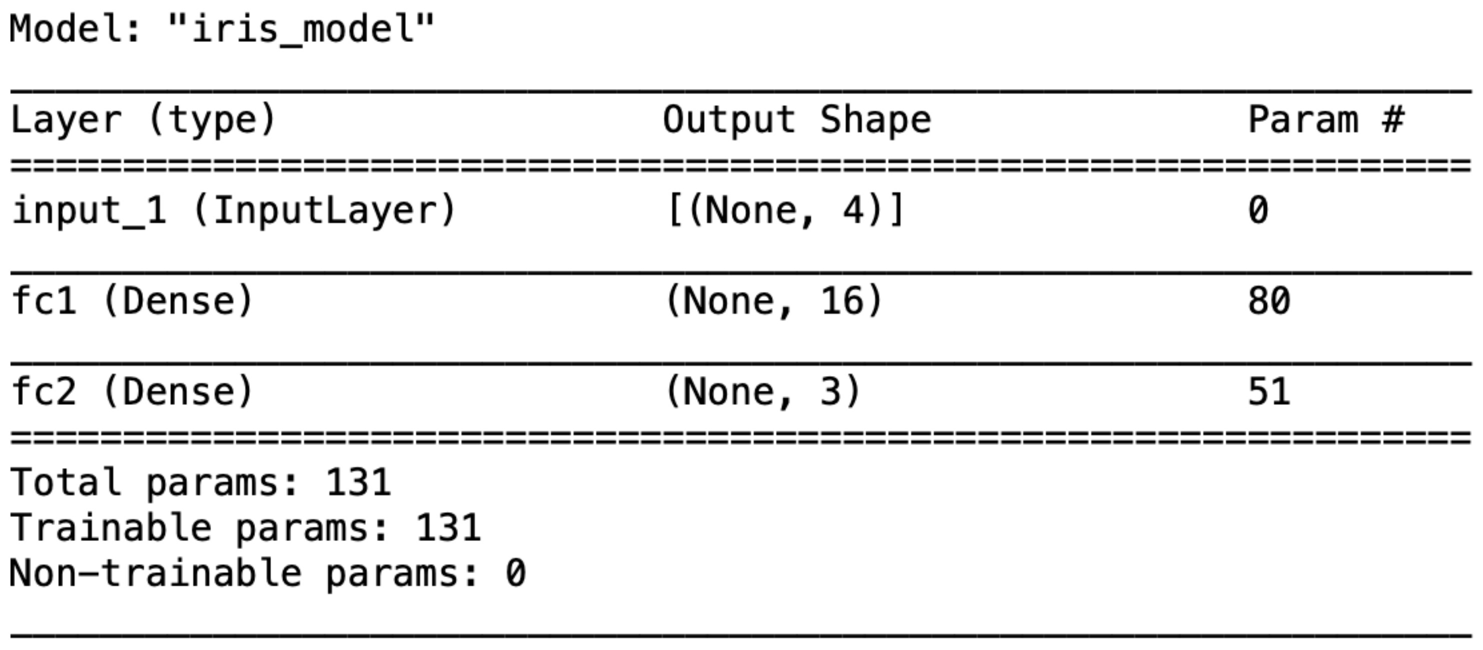
Latency		Interval		
min	max	min	max	Type
8	8	1	1	function

⊞ Detail

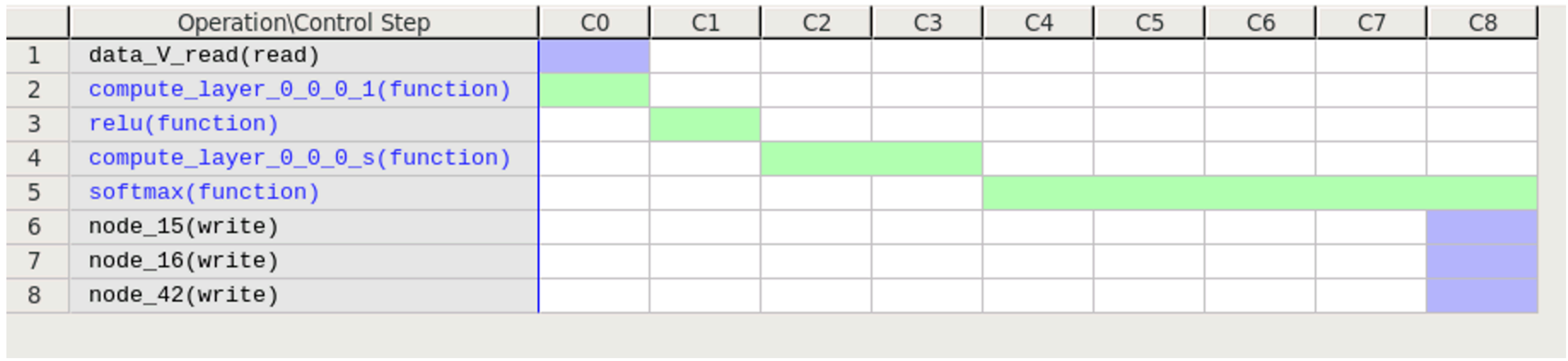
Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	-	-	-	-
Expression	-	-	0	27	-
FIFO	-	-	-	-	-
Instance	5	50	207	3050	-
Memory	-	-	-	-	-
Multiplexer	-	-	-	36	-
Register	-	-	379	-	-
Total	5	50	586	3113	0
Available	4320	6840	2364480	1182240	960
Utilization (%)	~0	~0	~0	~0	0



```
Model: "iris_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 4)]               0
_________________________________________________________________
fc1 (Dense)                  (None, 16)                80
_________________________________________________________________
fc2 (Dense)                  (None, 3)                 51
=================================================================
Total params: 131
Trainable params: 131
Non-trainable params: 0
_________________________________________________________________
```



	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
1	data_V_read(read)	🟦								
2	compute_layer_0_0_0_1(function)	🟩								
3	relu(function)		🟩							
4	compute_layer_0_0_0_s(function)			🟩	🟩					
5	softmax(function)					🟩	🟩	🟩	🟩	🟩
6	node_15(write)									🟦
7	node_16(write)									🟦
8	node_42(write)									🟦

- IP based

Resource utilization and clock frequency are fixed by design

Performance difficult to estimate

Implementation

Data strategy:

- Model standalone
    - How to send/receive data to/from the board?
    - How to interface NN block with the FPGA I/O?
    - I think FPGA expert is mandatory
- Model as a co-processors
    - Bitfile available (either as a dynamically loaded “overlay")

Running inference for co-processors

- Data can be managed in C/C++, Open-CL or Python with the CPU
- API provided to:
    - Load the bitfile (overlay)
    - Get tensors (size and “reference”)
    - Move data through DMA (connecting also to high speed memories into the FPGA boards)
    - Load “n" threads
    - Control execution
    - Access output data