Hands-on set-up

Systems@ETH Zürich

- The interactive part is done using Python notebooks

- If you haven't shared your GitHub username already, please fill in https://forms.gle/EorUm4Lj7LXsbR84A, so that access can be granted

- Open http://tutorials.fastmachinelearning.org/ in your web browser
  - Authenticate with your GitHub account (login if necessary)

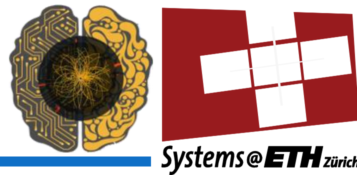
- If you have Vivado install yourself, you might prefer to work locally, see 'conda' section at: https://github.com/fastmachinelearning/hls4ml-tutorial

tutorial

Fast Machine Learning for Science 2025 @ ETH Zurich

Benjamin Ramhorst et al. for the **hls4ml** team

Introduction

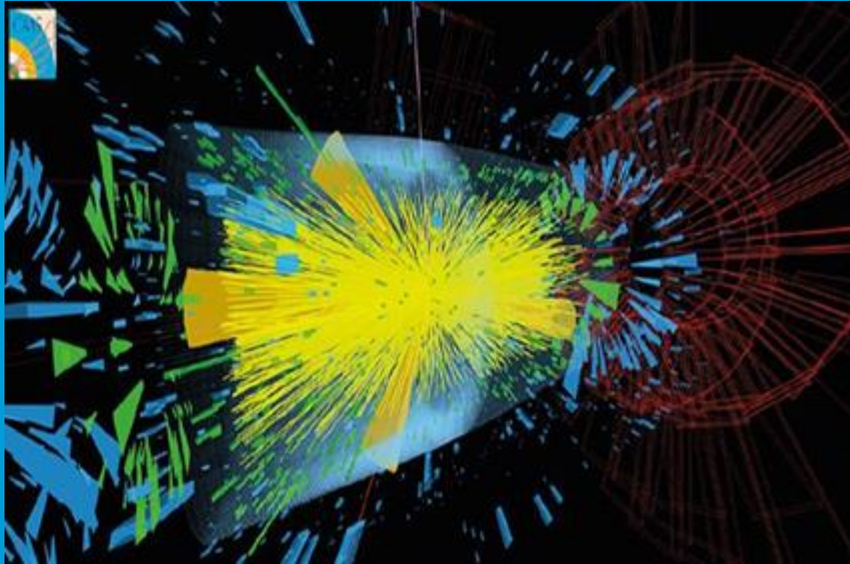


- **hls4ml** is a Python package for translating neural networks to FPGA firmware for inference with extremely low latency on FPGAs

- In this session you will get hands on experience with the **hls4ml** package

- We'll learn how to:
  - Translate high-level models into synthesizable FPGA code
  - Explore the different handles provided by the tool to optimize the inference
  - Make our inference more computationally efficient with quantization

LHC Triggering



Systems@ETH Zürich

- Extreme collision frequency of **40 MHz** → extreme data rates ~**100 TB/s**
- Most collision "events" don't produce interesting physics

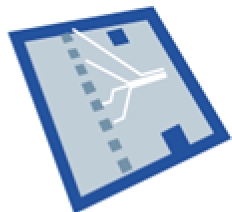
"Triggering" = filter events to reduce data rates to manageable levels

LHC Experiment Data Flow

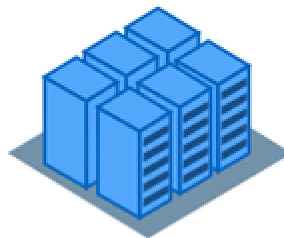
- L1 trigger: Incoming data rates of **100s TB/s:**

40 MHz
pp collisions

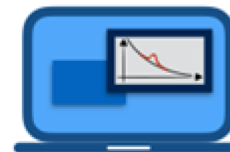
L1 Trigger



High-Level
Trigger



Offline
Computing



DATA FLOW

LHC Experiment Data Flow

- Deploy ML algorithms very early, avoiding off-line computation and storage
  - Challenge: Strict latency constraints **~10us** (incl. data movement, pre-processing, etc.)

40 MHz pp collisions

L1 Trigger

High-Level Trigger

Offline Computing

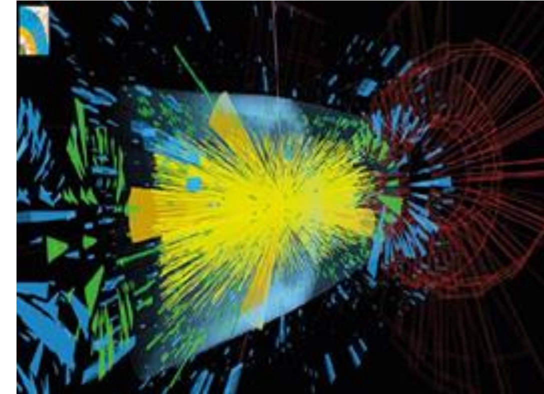
DATA FLOW

The latency - visualised

~1-3 seconds

~50ms

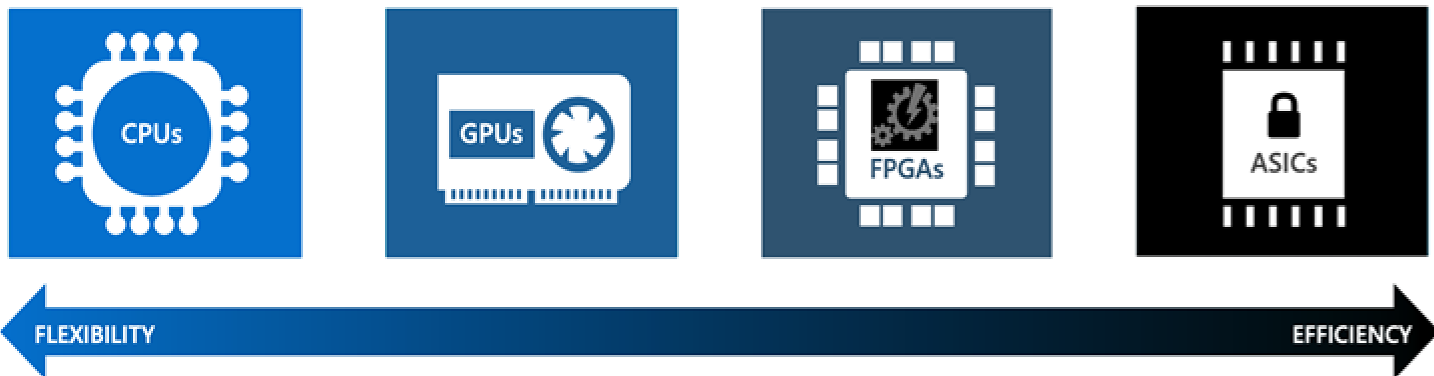
~500 ns



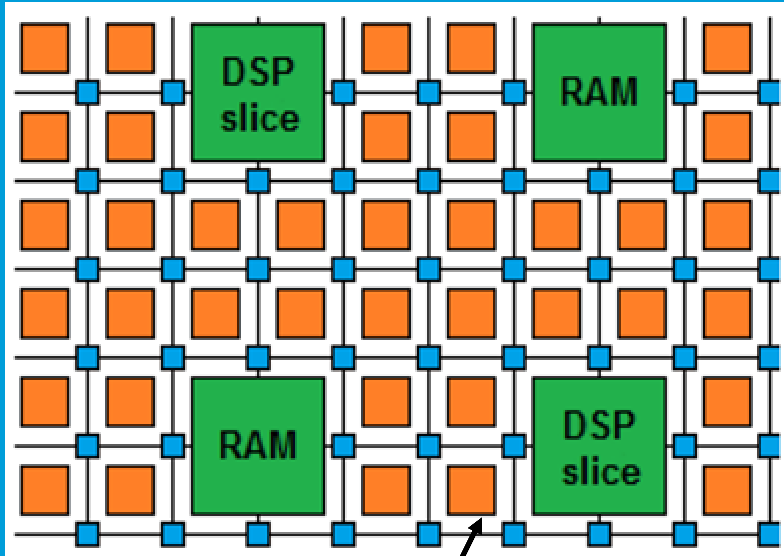
Why FPGAs?

Systems@ETH zürich

- ❑ Custom hardware acceleration, precisions and memory management:
  - ❑ E.g., GPUs offer very little support for arbitrary (e.g., 2-bit) precisions
- ❑ Data-flow architecture with no scheduling or control overheads:
  - ❑ E.g., A CPU has an operating system, a GPU has a workload scheduler etc.



What are FPGAs?



- **Logic cells (Look–up Tables)** perform arbitrary functions on small bit width inputs
- These can be used for Boolean operations, arithmetic, small memories

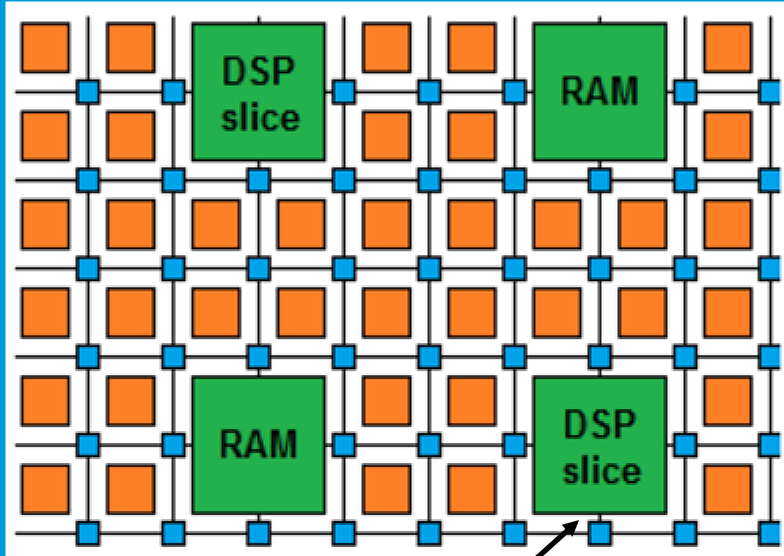
e.g., look-up table for NOT (A OR B)

A	B	LUT
0	0	1
1	0	0
0	1	0
1	1	0

- **Flip-Flops (registers)** data in time with the clock pulse

Systems@ETH Zürich

What are FPGAs?



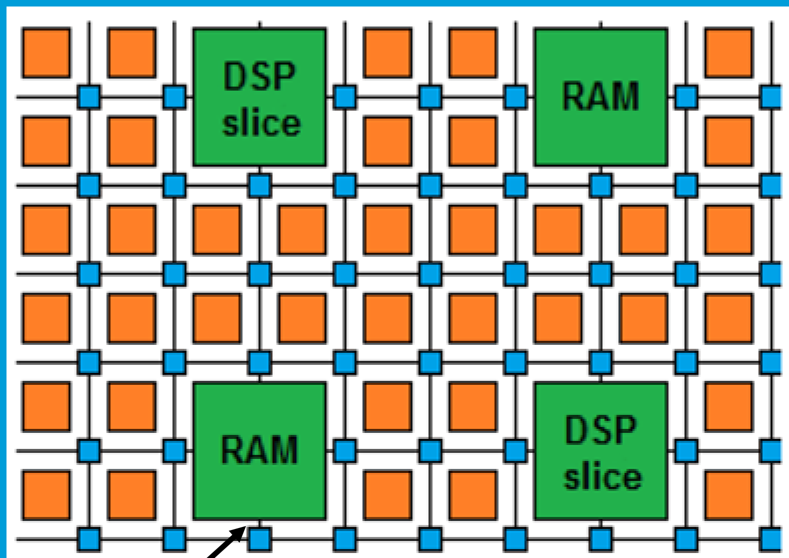
DSPs

- **DSPs (Digital Signal Processors)** are specialized units for multiplication and arithmetic
- Faster and more effici... for these types of op...
- And for neural networks, DSPs are often the most scarce:
  - Even high-end FPGAs will have ”only” around 10,000 DSPs, corresponding to 10,000 multiplications with DSPs at any given moment

Today, we will also see how to optimize the usage of DPS

Systems@ETH Zürich

What are FPGAs?



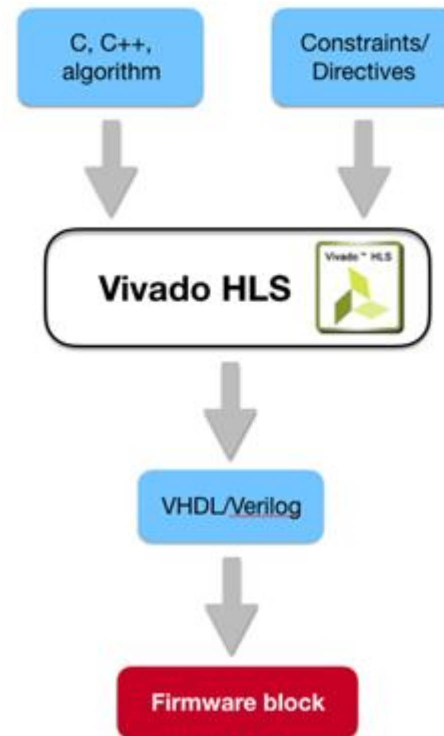
RAM

Systems@ETH Zürich

- **BRAMs** are small, fast memories
  - Access data in one clock cycle
- A big FPGA has nearly 40MB of BRAM, chained together as needed (bandwidth)
  - Even suitable for "larger" models, such as ResNet
- Recent accelerator cards also come equipped with off-chip HBM memory (up to 800 GBps)

How are FPGAs programmed?

- **Hardware Description Languages**
  - HDLs are programming languages which describe electronic circuits
- **High Level Synthesis**
  - Compile from C/C++ to VHDL
  - Pre-processor directives and constraints used to optimize the design
  - Drastic decrease in firmware development time!
- Today we'll use **AMD/Xilinx Vitis HLS**



HLS examples

Systems@ETH zürich

```
for(int i = 0; i < 16; i++) {
    #pragma HLS UNROLL 4
    c[i] = a[i] + b[i];
}
```

- Loop parallelised 4 times in each clock cycle
- Total block execution: 16 / 4 = 4cc

```
float weights[16];
#pragma HLS ARRAY_RESHAPE variable=weights block factor=block_factor
#pragma HLS RESOURCE variable=weights core=ROM_2P_BRAM
```

- Reshapes the weight matrix according to a specific block factor
- Stores the resulting array in on-chip ROM

Jargon

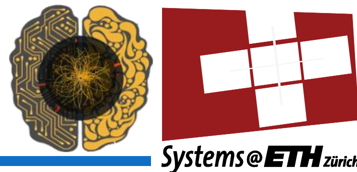
Systems@ETH Zürich

- **LUT** - Look Up Table aka 'logic' - generic boolean functions on small bitwidth inputs. Combine many to build the algorithm
- **FF** - Flip Flops - control the flow of data with the clock pulse. Used to build the pipeline and achieve high throughput
- **DSP** - Digital Signal Processor - performs multiplication and other arithmetic in the FPGA
- **BRAM** - Block RAM - hardened RAM resource. More efficient memories than using LUTs for more than a few elements
- **HLS** - High Level Synthesis - compiler for C, C++, SystemC into FPGA IP cores
- **HDL** - Hardware Description Language - low level language, such as Verilog or VHDL for describing circuits
- **Latency** - time between starting processing and receiving the result
- **II** - Initiation Interval - time from accepting first input to accepting next input (visualize, cars on a production line)

Latency vs initiation interval

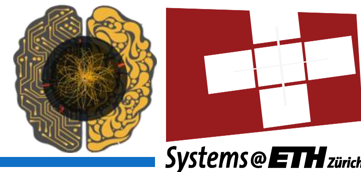
Latency

What is hls4ml today?



- A generic framework for FPGA acceleration of neural networks:
  - **Front-end agnostic**: Keras, PyTorch, (Q)ONNX
  - **Back-end agnostic:** Vivado HLS, Vitis HLS, Intel HLS, oneAPI etc.
  - **Many supported layers:** Dense, Conv, Recurrent, Graph etc.
  - **High configurability:** Tune precision, reuse factor, custom layers etc.
  - **An active, open-source community:** Many collaborators from many different fields and institutions

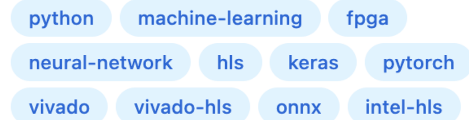
What is hls4ml today?



About

Machine learning on FPGAs using HLS

fastmachinelearning.org/hls4ml



- Readme
- Apache-2.0 license
- Contributing
- Cite this repository
- Activity
- Custom properties
- 1.6k stars
- 53 watching
- 478 forks

Fast inference of deep neural networks in FPGAs for particle physics — [PDF] iop.org

J Duarte, S Han, P Harris, S Jindariani, E Kreinar, B Kreis, J Ngadiuba, M Pierini, R Rivera…

Journal of instrumentation, 2018 • iopscience.iop.org

Abstract

Recent results at the Large Hadron Collider (LHC) have pointed to enhanced physics capabilities through the improvement of the real-time event processing techniques. Machine learning methods are ubiquitous and have proven to be very powerful in LHC physics, and particle physics as a whole. However, exploration of the use of such techniques in low-latency, low-power FPGA (Field Programmable Gate Array) hardware has only just begun. FPGA-based trigger and data acquisition systems have extremely

MEHR ANZEIGEN

Speichern   Zitieren   Zitiert von: 546   Ähnliche Artikel   Alle 12 Versionen   Web of Science: 249

hls4ml use cases

REAL-TIME SEMANTIC SEGMENTATION ON FPGAS FOR AUTONOMOUS VEHICLES WITH HLS4ML

Nicolò Ghielmetti,* Vladimir Loncar,† Maurizio Pierini, Marcel Roed,‡ Sioni Summers
European Organization for Nuclear Research (CERN)
CH-1211 Geneva 23, Switzerland

…and many more in science, quantum, biomedical etc.

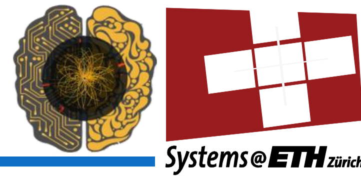
SoC-based implementation of 1D Convolutional Neural Network for 3-Channel ECG Arrhythmia Classification via HLS4ML

Feroz Ahmad, Saima Zafar, *Senior Member, IEEE*

Machine Learning-based Deep Packet Inspection at Line Rate for RDMA on FPGAs

Maximilian J. Heer* Benjamin Ramhorst* Gustavo Alonso

high level synthesis for machine learning

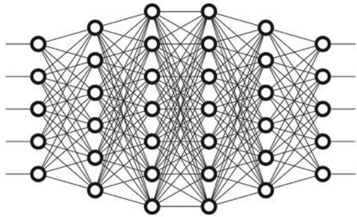


User-provided

hls4ml optimization

hls4ml model optimizers

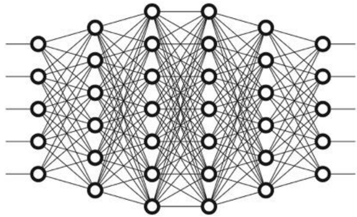
hls4ml kernels



hls4ml is completely front-end agnostic

high level synthesis for machine learning

User-provided



```
Backend: Vitis
ClockPeriod: 4
IOType: io_parallel
Part: xcu55c-fsvh2892-2L-e
Model:
    Precision: ap_fixed<16, 8>
    ReuseFactor: 1
    Strategy: Latency
    TraceOutput: false
```

optimization

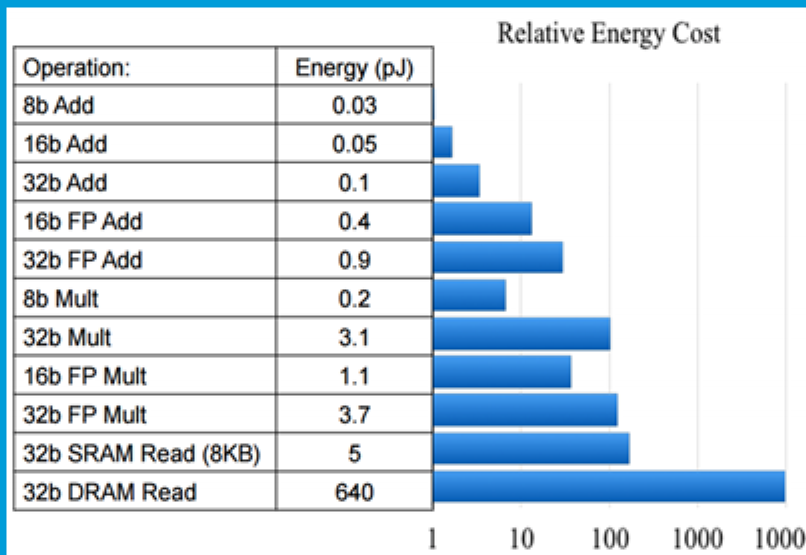
model optimizers

kernels

```
Backend: Vitis
ClockPeriod: 4
IOType: io_parallel
Part: xcu55c-fsvh2892-2L-e
Model:
    Precision: ap_fixed<16, 8>
    ReuseFactor: 1
    Strategy: Latency
    TraceOutput: false
```

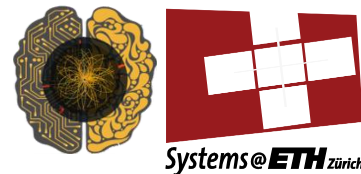
hls4ml is highly configurable and extendable though the Extension API

Efficient inference: quantisation



Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

[Horowitz @ ISSCC'14]



- ❑ Floating point operations are expensive
- ❑ On FPGAs, we can use fixed-point precision
  - ❑ Implemented using integer logic, so very fast
  - ❑ Acts like "limited-precision" floating-point, so need to ensure sufficient bits

0101.110000

integer fractional

- ❑ **hls4ml** includes support for both post-training quantisation and quantisation-aware training

Example flow

User-provided

```
Backend: Vitis
ClockPeriod: 4
IOType: io_parallel
Part: xcu55c-fsvh2892-2L-e
Model:
    Precision: ap_fixed<16, 8>
    ReuseFactor: 1
    Strategy: Latency
    TraceOutput: false
```

hls4ml optimization

HGQ

Hardware-aware pruning

hls4ml model optimizers

Conv 2D

Batch Norm

Scaled Conv 2D

hls4ml kernels

```
dense_latency<CONFIG_T>()
dense_resource<CONFIG_T>()
```

```
winograd_conv2d<CONFIG_T>()
im2col_conv2d<CONFIG_T>()
```

```
leaky_relu<CONFIG_T>()
```

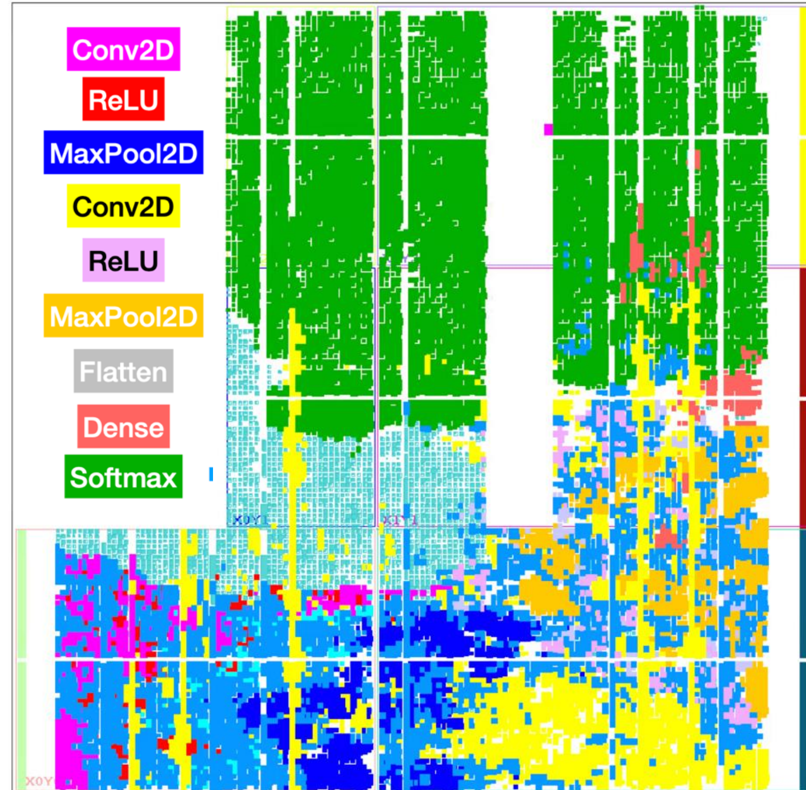
Many commonly used architectures supported!

And finally…

- **hls4ml** passes the FPGA project to commercial FPGA tools, completely abstracting aways the complexity behind simulation, synthesis and place-and-route:

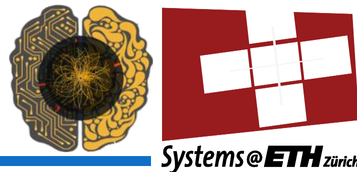
```
def build(
    csim=True, synth=True, cosim=False,
    validation=False, export=False
):
```

- **Backend-agnostic:** support for all major FPGA boards and languages:
  - Vivado HLS, Vitis HLS (Xilinx / AMD)
  - Intel HLS, Intel oneAPI (Intel / Altera)
  - Siemens Catapult HLS



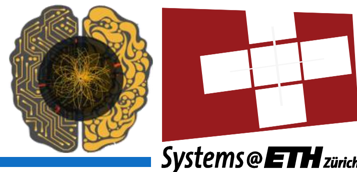
Systems@ETH zürich

Today's tutorial



- **Part 1:**
  - Get started with **hls4ml**: train a basic model and run the conversion, simulation & C-synthesis steps
- **Part 2:**
  - Learn how to tune inference performance with quantization & ReuseFactor
- **Part 3:**
  - Train using QKeras "quantization-aware training" and study impact on FPGA metrics

What's not covered today?



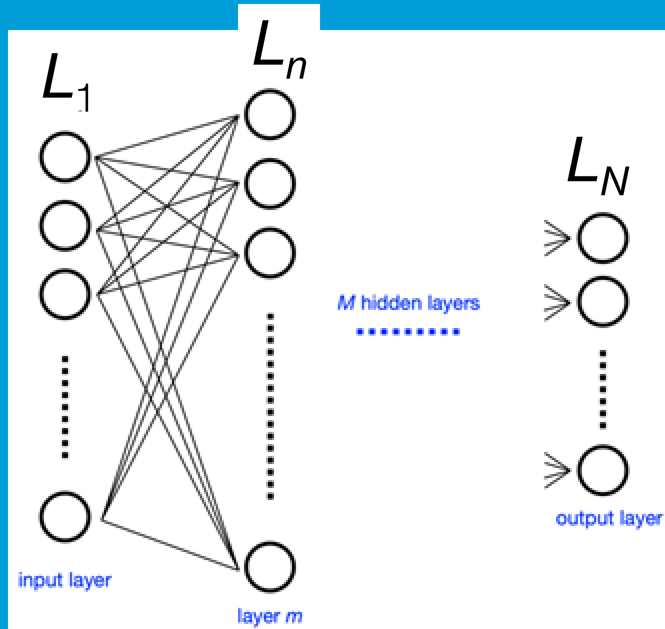
- **High-granularity quantisation:** heterogenous layer quantisation for ultra-low latency, low-area neural networks **(covered in the next session)**

- **Convolutional neural networks**
  - Notebooks available on GitHub, however, synthesis takes long

- **What comes after hls4ml…** you would need to integrate the 'IP core' into a larger design
  - For a custom board, you'd need to do this by hand (e.g. CMS L1 Trigger)
  - For more off-the-shelf boards, integration with system-on-chip or host CPU can be more straightforward, using tools such as XRT
  - If you are interested in deploying applications on FPGA, **check out the Coyote tutorial in the next session**

Part 1: Model Conversion

Neural network inference



Systems@ETH Zürich

$$\mathbf{x}_n = g_n(\mathbf{W}_{n,n-1}\mathbf{x}_{n-1} + \mathbf{b}_n)$$

logic cells OR precomputed and stored in BRAMs

DSPs

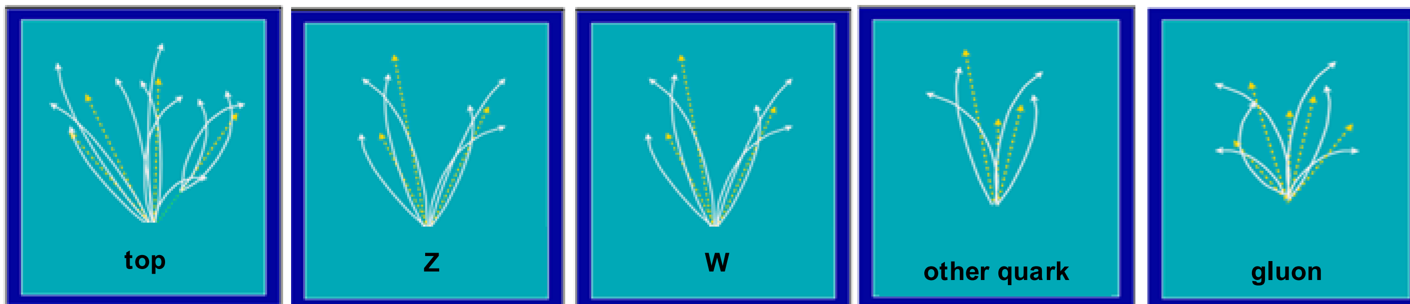
logic cells

How many resources? DSPs, LUTs, FFs?

Does the model fit in the latency requirements?

Physics use case: jet tagging

- Study a multi-classification task to be implemented on FPGA: discrimination between highly energetic (boosted) q, g, W, Z, t initiated jets



t→bW→bqq 3-prong jet

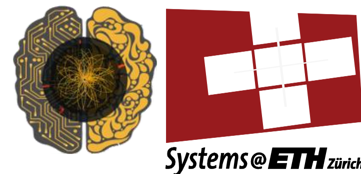
Z→qq 2-prong jet

W→qq 2-prong jet

Reconstructed as one massive jet with substructure

q/g background no substructure and/or mass ~ 0

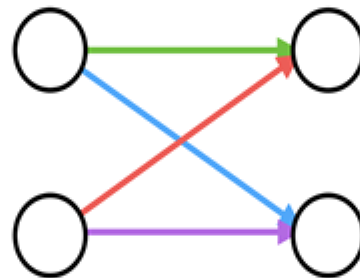
Part 2: Advanced configuration

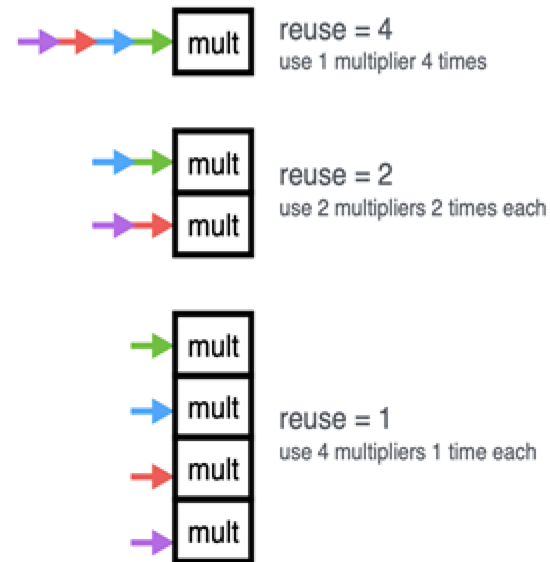


Efficient inference: parallelisation

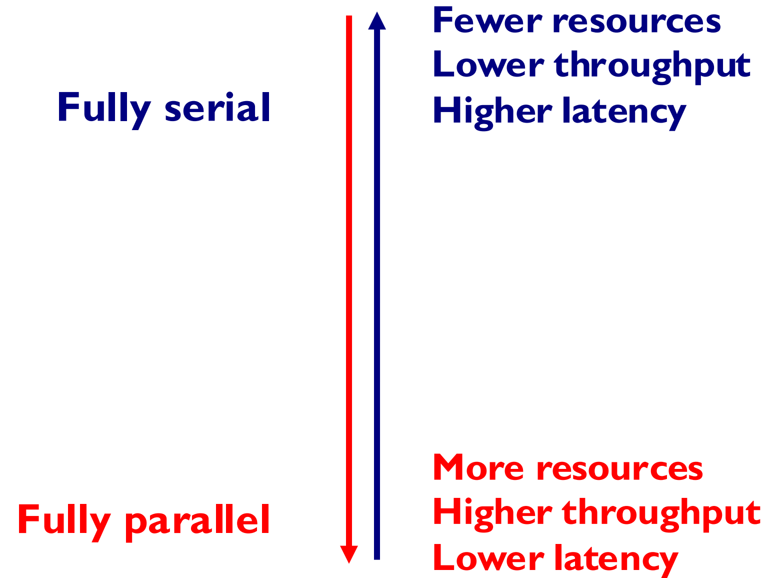
- Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer

- **Configure the "reuse factor" = number of times a multiplier is re-used to do a computation**

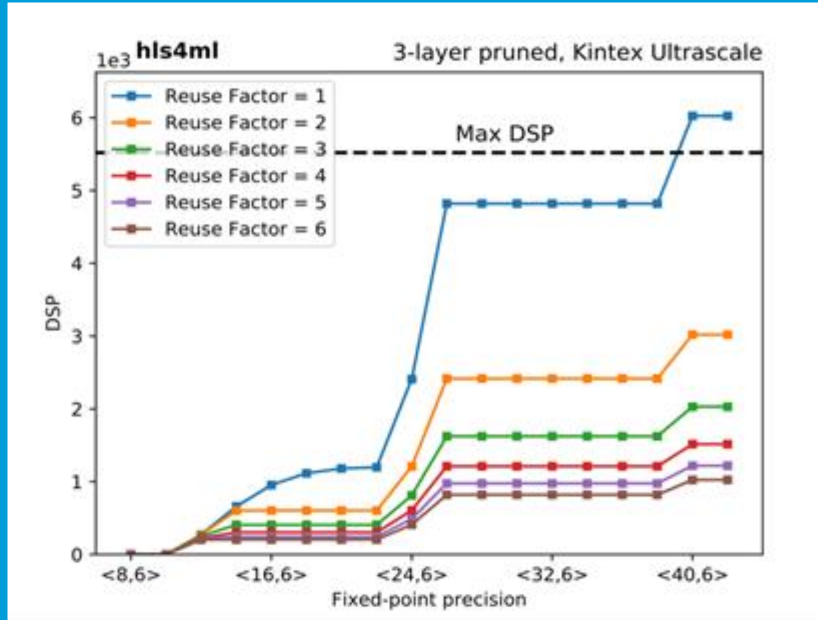




Efficient inference: parallelisation



Parallelisation: DSP usage



Systems@ETH Zürich

More resources

Fully parallel
Each mult. used 1x

Each mult. used 2x

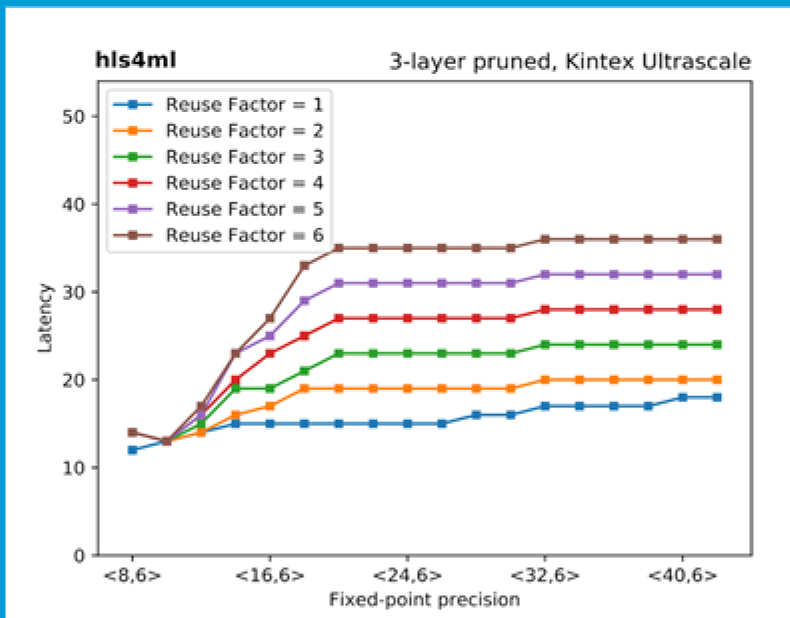
Each mult. used 3x

⋮

Longer latency

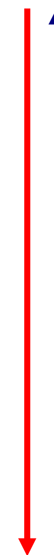
Systems@ETH Zürich

Parallelisation: Latency



Less resources

~ 175 ns

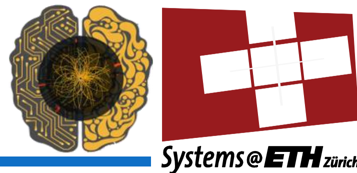


~ 75 ns

Lower latency

Part 3: Quantisation

Quantisation-aware training



- **hls4ml** allows us to use different data types everywhere, we saw how to tune that in Part 2

- Now, we will also try quantization-aware training with QKeras

- Quantisation-aware training enables training models with very low precision:
  - Out-performs post-training quantisation significantly
  - At a high level, it performs the forward pass with reduced precision and the backward pass in floating point precision
  - Possible to achieve very precisions (binary and ternary models)

QKeras

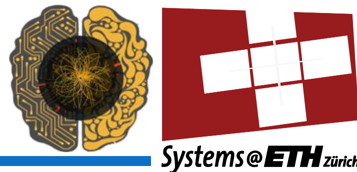
Systems@ETH Zürich

- QKeras is a library to train models with quantization during training
  - Developed & maintained by Google

- Easy to use, drop-in replacements for Keras layers
  - e.g. Dense → QDense, Conv2D → QConv2D
  - Use 'quantizers' to specify how many bits to use where
  - Can achieve good performance with very few bits

- Stable support for QKeras-trained models to **hls4ml**
  - The number of bits used in training is automatically parsed for conversion & inference

Summary



- After this session you've gained some hands-on experience with **hls4ml**
  - Translated neural networks to FPGA firmware, run simulation and synthesis
  - Tuned network inference performance with precision and ReuseFactor
  - Trained a quantized model using QKeras, and use the same model for inference with hls4ml
- The tutorials to run locally are at: https://github.com/fastmachinelearning/hls4ml-tutorial
- Use **hls4ml** locally: `pip install hls4ml`

Questions?

Special thanks to the Fast Machine Learning Community for the on-going efforts in hls4ml development and its accompanying tutorials. The materials used today are based on: https://github.com/fastmachinelearning/hls4ml-tutorial and were used and edited with permission of the authors.