HLS4ML CONCEPTS AND APPLICATIONS

NEMER CHIEDDE

Supervisor: Emmanuel MONNIER

Co-supervisor: Georges AAD

CONTENT

- *Introduction*
- *High Level Synthesis for Machine Learning workflow*
- *Package Architecture supports*
- *HLS4ML with RNNs and optimizations*
- *HLS / VHDL differences and optimizations*
- *User perspectives*

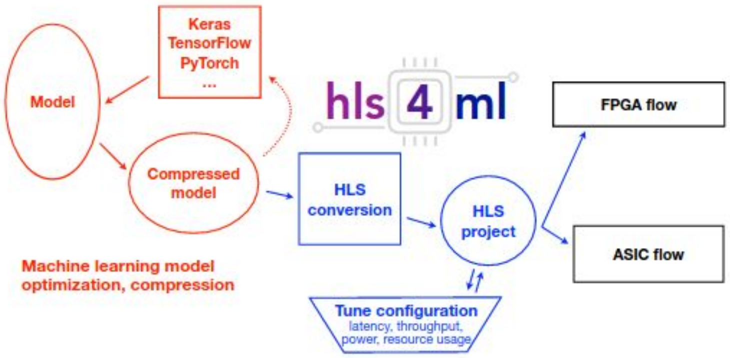
Introduction

- HLS4ML is an open source software designed to facilitate the implementation of AI algorithms on FPGAs.
- Performs automatically the task of translating a trained NN, specified by the model's architecture, weights, and bias, firmware for a specific hardware
- Comes with implementation of common ingredients (layers, activation functions, binary NN, ...)
- Available for Vivado (Xilinx) and recently for Quartus (Intel)
  - I implemented the RNNs for Quartus
  - I Included some scripts and tools for hls conversion and inspection
- **RNN** for Quartus is now supported (link)
- HLS4ML code available on Github (link)
- HLS4ML can be found on this documentation (link)

HLS4ML WORKFLOW

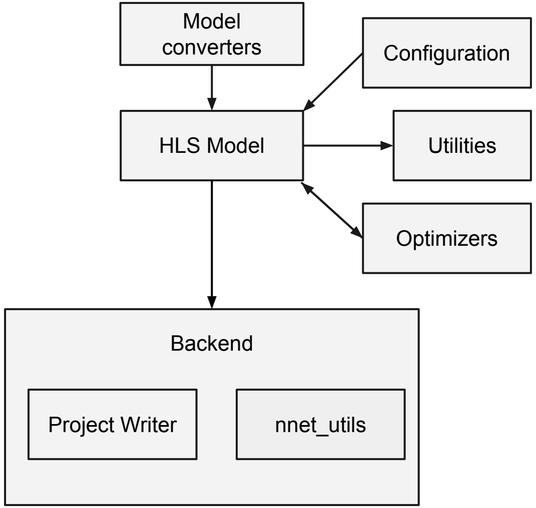
- **HLS4ML** is a **python package**
- **Create optimal digital design**: balance available resources with achieving the target power, latency, ….

- **Red section** describes the usual steps needed to design a NN for a specific task
  - Definition of the model structure
  - Compression drop zero weights to reduce the resource usage



- **Blue section** translates a model into an HLS design that can later be synthesized and implemented on an FPGA or ASIC, example:
  - Run fully in parallel or concurrently, control the inference latency versus utilization of FPGA resources
  - Quantization reduce precision of the calculations
  - Activations functions used as LUT don’t need calculations reducing the resource used
- **Black section** is the options to export the HLS project generated by HLS4ML

HLS4ML PACKAGE ARCHITECTURE



- Internal structure is divided in six principles packages
- Model converters:
  - Convert the NN model into a common internal representation of the network graph
  - Support Keras, QKeras, TensorFlow, PyTorch, and ONNX model formats
- Optimizers:
  - Modify the network graph to target a more lightweight, faster inference.
  - Reduce operations at runtime
  - Examples:
    - 1x1 convolution uses optimized HLS code implementation reducing latency
    - Automatic removal of Softmax if it is present in the last layer (avoiding redundancy)

HLS4ML ADAPTABILITY

- Utilities:
  - Provides a set of utilities to aid the configuration process, so the model object can be inspected
  - Can display the NN graph with the user configuration
- Configuration:
  - Provides a number of configurable parameters which can help the user explore and customize
    - **“precision”:** Bitwith size for weights, recurrent weights and bais (*ap_fixed*<16, 6> *by default*)
    - **“table_size”:** Lookup table (1024 by default)
    - **“ReuseFactor”:** Lower reuse factor reduces latency and higher resources (1 by default)
    - **“Strategy”:** resources or latency (not added yet for quartus)
    - **“io_type”:** parallel, stream (Parallel by default)
    - **“clock_period”:** Clock frequency (5 ns or 200 MHz by default)
    - **“backend”:** Possible to use Vivado, VivadoAccelerator and Quartus (Vivado by default)
    - **“part”:** FPGA part (xcku115-flvb2104-2-i by default)
  - Each layer and activation type is implemented as a separate configurable module customized to perform that specific operation

HLS4ML MODEL

- Backend:
  - Used to export the model into a given specific language, such as Intel HLS, Vivado HLS

  - Possibility that the CPU runs the conversion and obtains the numerical results

  - Project writers:
    - Overwrite HLS and generate hardware modules

  - Nnet utils:
    - Apply the neural networks requested on the NN model
    - Support multi features and return sequences
    - NN architectures: Fully Connected NNs (Dense), Convolutional NNs (1D, 2D), Recurrent RNN (GRU, Simple-RNN, LSTM)
    - Activations and normalization are added on nnets (pooling and batch normalization)

- HLS Model
  - File generated to produce an IP core

HLS DEFINITION

- High Level Synthesis is an automated design procedure
- Converts the algorithmic description of a system into the corresponding hardware circuit.
- The synthesis tool generates the technical detail
  - Creates a Register Level Transfer (RTL) implementation from C++ (link)
- Developed HLS from scratch for our specific use case
  - Already found to have a good energy resolutions (link)
  - Used as comparateur for hls4ml RNN implementation

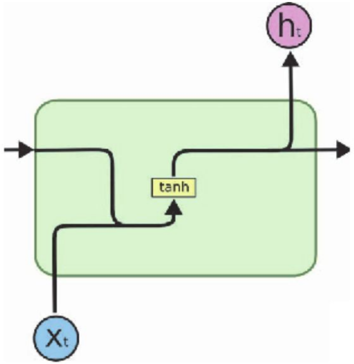
HLS4ML SYNTHESIS TOOLS AND VALIDATION

- From HLS Model it is possible to do:
  - **C simulation:** Numerical results
  - **HLS Synthesis:** Numerical results and RTL synthesis
  - **Full Synthesis:** Place and route, time analysis, numerical results and RTL synthesis
  - **Export:** Export IP

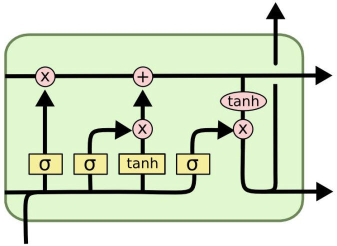
Support of RNN models on HLS4ML

- HLS4ML didn't support RNNs nor Quartus, so I implemented the LSTM and Vanilla-RNN and some optimizations
  - LSTM cell: more complex, highest accuracy, more resources used.
    - Two activation functions (Hyperbolic tangent and sigmoid)
  - Vanilla-RNN or Simple-RNN cell: simplest, less resources used, less performing than LSTM.
    - One activation function (ReLU)

Simple-RNN

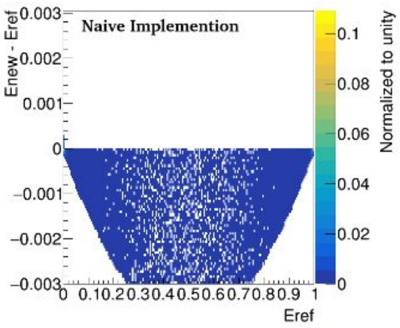


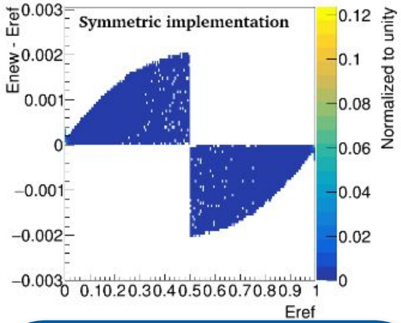
LSTM

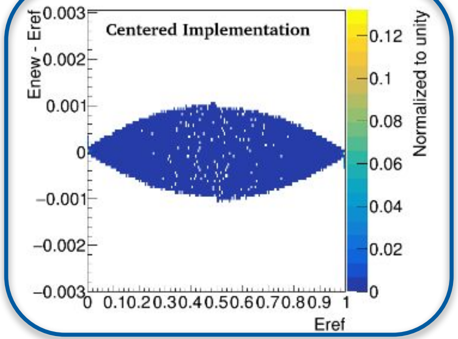


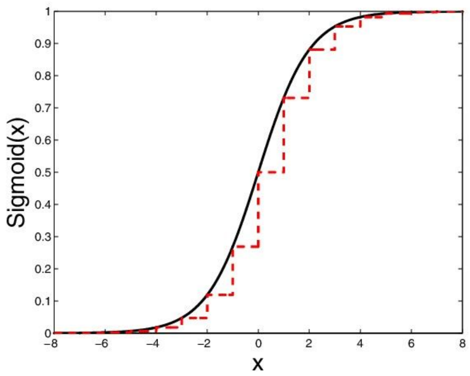
Symmetric optimization LUT applied

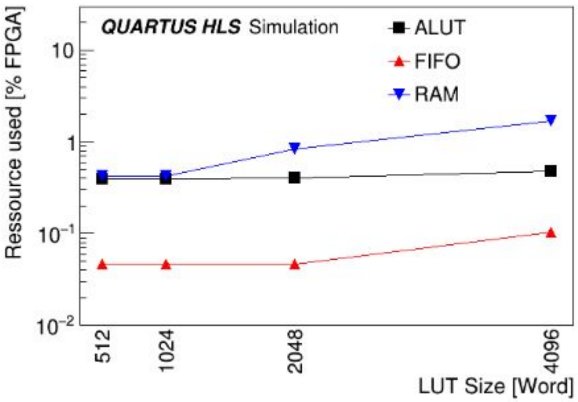
- To improve resource usage, it is possible to apply half the table size usage using the symmetric softsign, sigmoid and tanh properties (e.g. sig(-x)=-sig(x))
- Already implemented in hls and **already available on master**
- Optimizing the LUT's gives an higher precision for the same resources usage value
- Sigmoid and hyperbolic tangent LUT output strictly between -1 and 1:
  - no need for integer bits









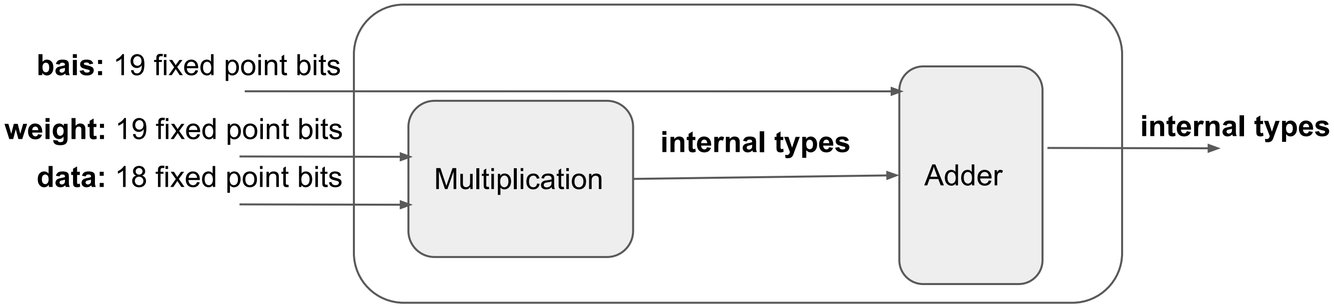


Internal Fixed Point effects

- The internal fixed point representation affects directly the performance of the output and the resource used (*Etienne's thesis*)
    - Especially for DSP, which is the most used when applying multiplications

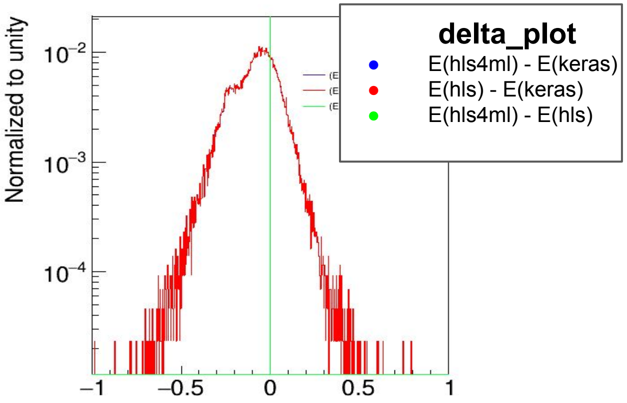
- For the FPGA type statix 10, the precision DSP blocks for 2 variable are:
    - Fixed-point complex 18 x 19 multiplication

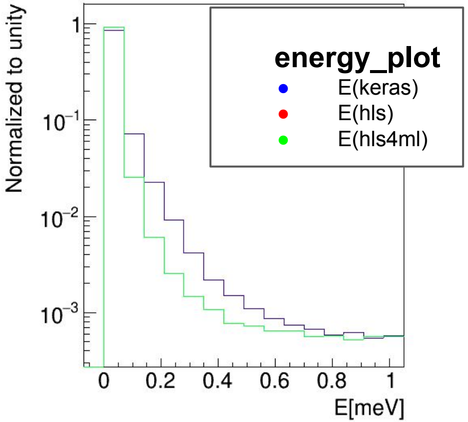
- After the multiplication, the 37 bits generated can be casted now in a different internal fixed point representation and not more in 19 bits.
    - Other math calculations, like adder, can obtain more precision using a different internal representation fixed point after the first multiplication



Actual LSTM comparison between HLS and HLS4ML

- HLS output already found to have a good energy resolutions
- **No difference between hls reference and hls4ml**

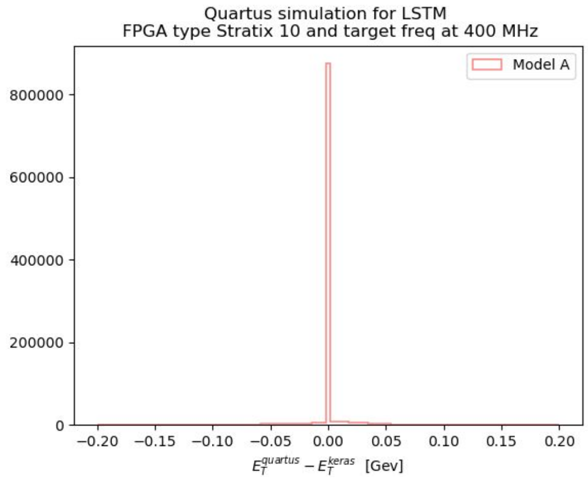


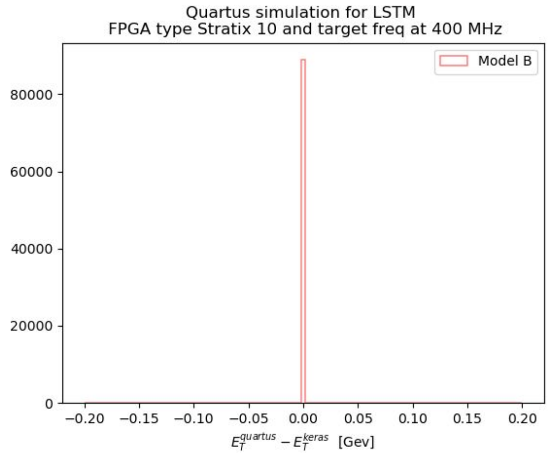


Stratix10 1SG280HU2F50E2VG	LUTs	FFs	RAMs	DSPs	Latency (min, max,avg)	II (min, max, avg)	Target frequency
HLS (Etienne PhD)	8%	6%	4%	13%	325,353,339	1, 1, 1	400 MHz
HLS4ML with no optimizations	9 %	7%	14%	22%	322, 346, 322	1, 1, 1	400 MHz
HLS4ML with LUT optimization	9 %	6%	8%	22%	322, 346, 322	1, 1, 1	400 MHz
HLS4ML with presented optimizations	8%	6%	4%	13%	320,326,325	1, 1, 1	400 MHz

Examples of others Recurrent Neural Networks

LSTM





Model A

Layer (type)	Output Shape	Param #
lstm_6 (LSTM)	(None, 4)	96
dense_7 (Dense)	(None, 8)	40
dense_8 (Dense)	(None, 1)	9

Total params: 145
Trainable params: 145
Non-trainable params: 0

- The **activation** and **recurrent activation** function of **LSTM** was **Tanh** and **Sigmoid**. For the both **Dense** was **Relu**

Model B

Layer (type)	Output Shape	Param #
lstm_4 (LSTM)	(None, 4)	96
dense_4 (Dense)	(None, 1)	5

Total params: 101
Trainable params: 101
Non-trainable params: 0

- The **activation** and **recurrent activation** function of **LSTM** was **Sigmoid** and **Tanh**. **Dense** was **Relu**

VHDL IMPLEMENTATION

- Manually setting the placement constraints allows us to reduce the timing issue
  - Impossible to replace on HLS

- Duplicate the recurrent kernel cell reduce the mean path between the weights and the DSPs

- Synchronization inside of the cell at the critical path on the RTL code

Type	Number of network	Multiplexing	Number of channel	ALM	DSP	Memory	Fmax
Specifications	X	X	384	30%	70%	30%	Mult * 40 MHz
without placement	28	14	392	18.2%	66.1%	15.8%	501 MHz
with placement	28	14	392	18.2%	66.1%	15.8%	531 MHz

- Incremental compilation
  - Fixing some crucial FPGA areas and using incremental compilation, it's possible to reach higher frequencies

Type	Number of network	Multiplexing	Number of channel	ALM	DSP	Memory	Fmax
Specifications	X	X	384	30%	70%	30%	Mult * 40 MHz
first compilation	28	14	392	18.2%	66.1%	15.8%	531 MHz
incremental compilation	28	14	392	18.2%	66.1%	15.8%	561 MHz

General status and perspectives

- Software package for translation of trained neural networks into synthesizable FPGA firmware
- RNN are now available on github master for Quartus
- HLS4ML can be used for quick start and optimization of NN params. However, final VHDL optimization is needed if hard constraint have to be met on latency and resource usage.