www.fpgafais.com

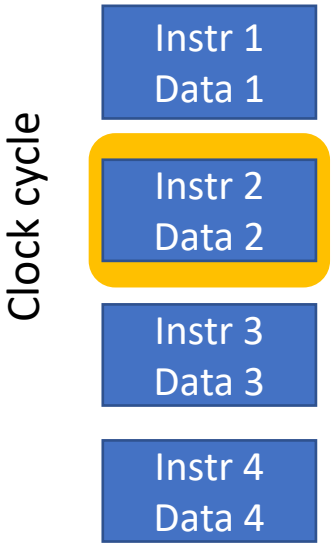
Neural Networks inference on FPGA-based platforms

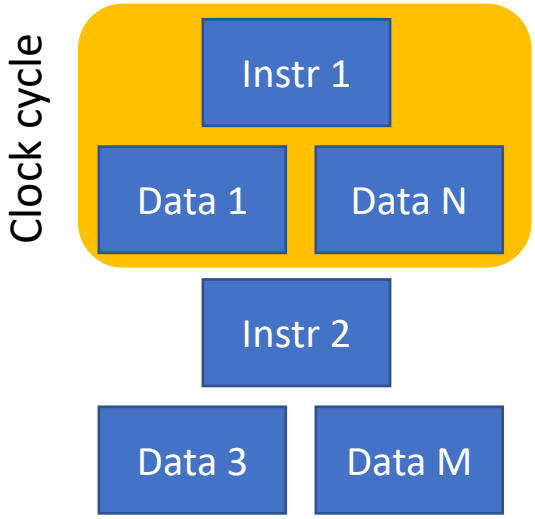
Grzegorz Korcyl

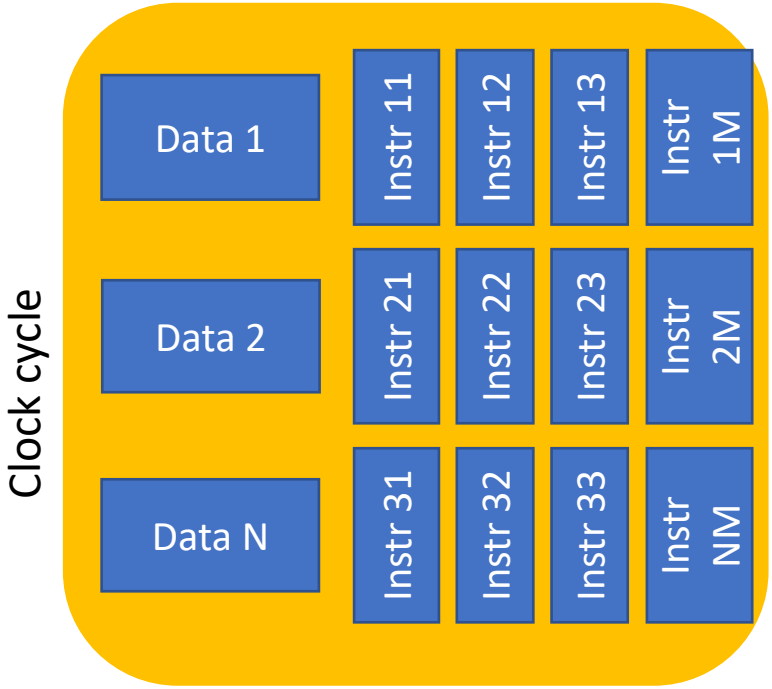
Department of Information Technologies
Jagiellonian University, Cracow

15 September 2022
WMLQ 2022

CPU vs GPU vs FPGA







- CPU
  - Single Instruction Single Data per core
  - Fixed instruction set
  - Multiple cores
  - High clock freq.
  - Operating system
- GPU
  - Single Instruction Multiple Data
  - Fixed instruction set
  - High clock freq.
  - Memory access
  - Accelerates CPU
- FPGA
  - Flexible architecture
  - Massive parallelism
  - Streamlined processing
  - Low clock freq.
  - Instant memory access
  - Standalone platforms

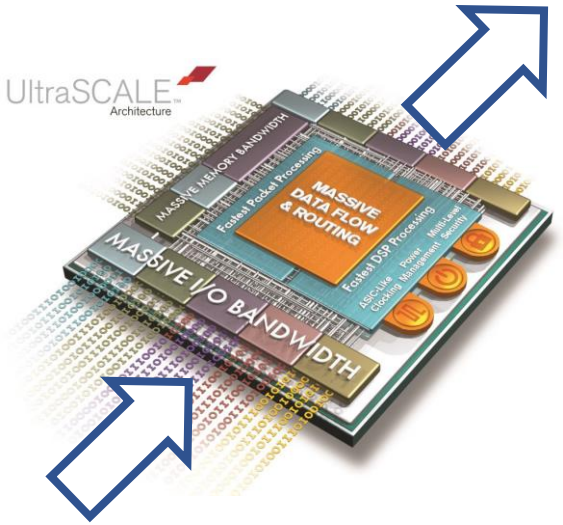
Different approach

Instead of adapting the program to a given architecture

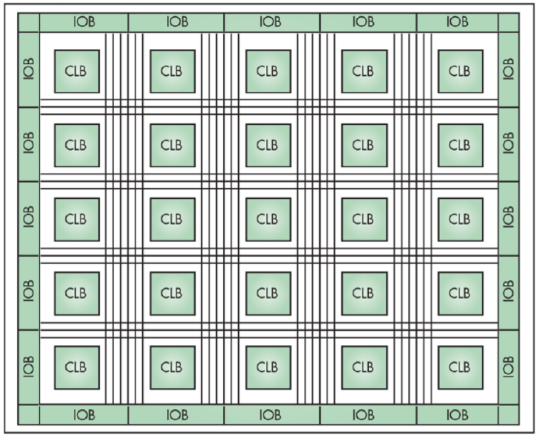
Let’s design the architecture that performs the task in the most efficient way

What are FPGAs

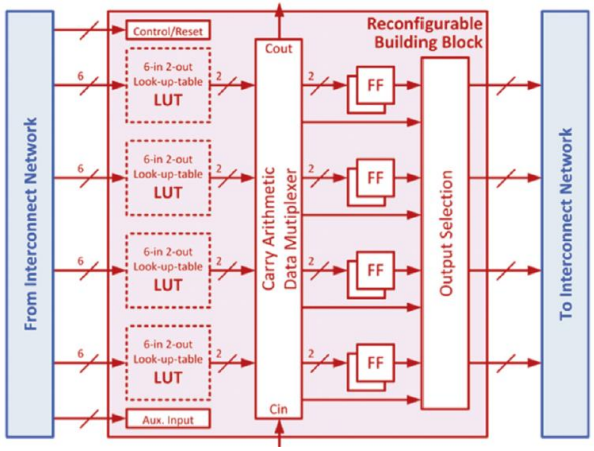
- Field Programmable Gate Arrays
  - Devices for processing digital data streams
  - Adaptable computing resources
  - Reconfigurable at any time

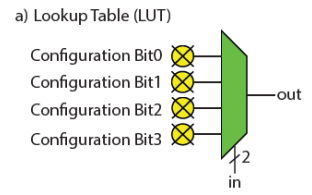


Arrays of Configurable Logic Blocks



Basic Configurable Logic Block



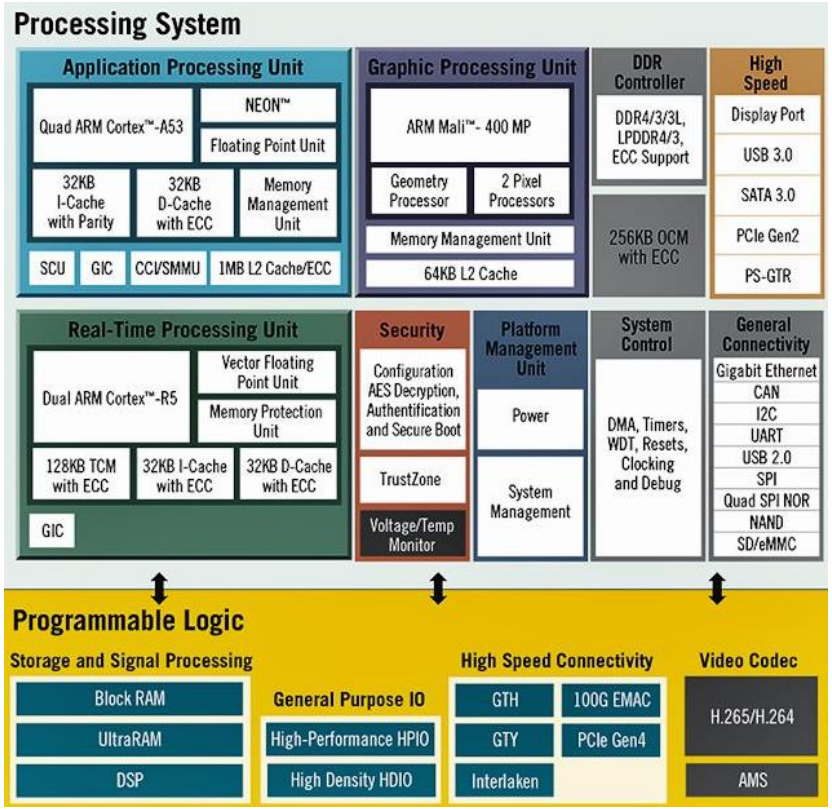


R. Kastner, J. Matai, S. Neuendorffer „Parallel Programming for FPGAs"

www.electronicdesign.com

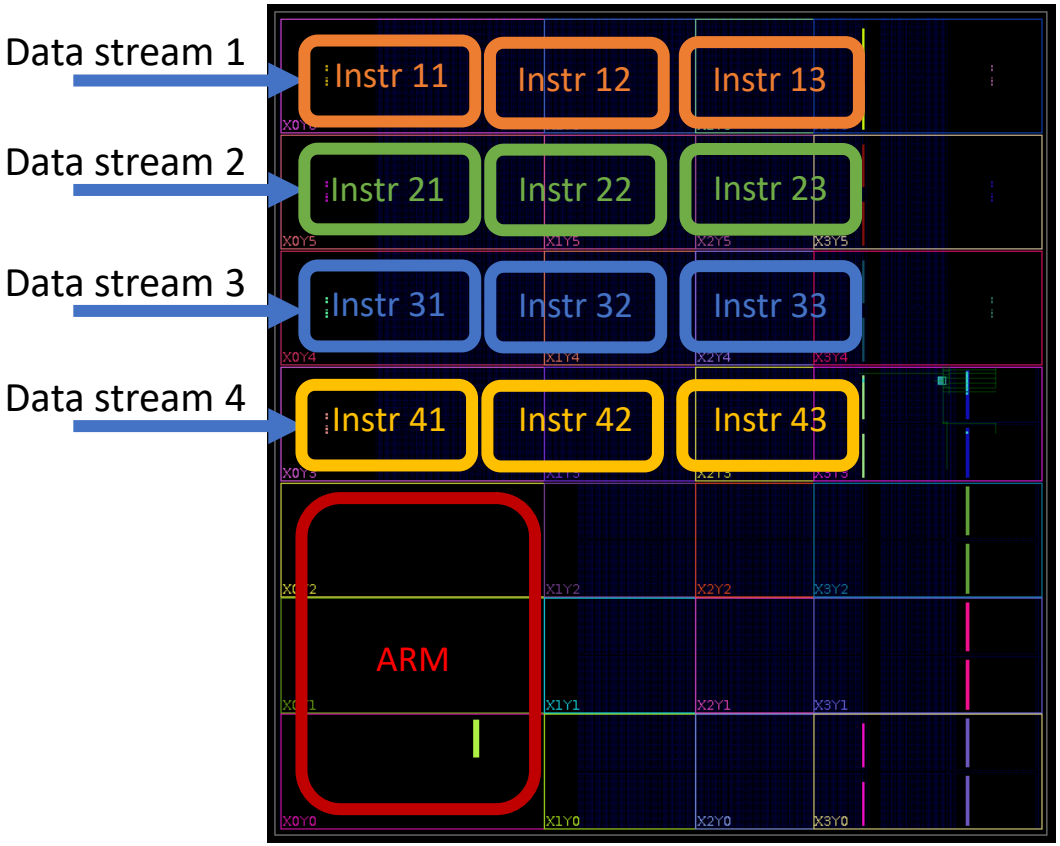
What are FPGAs

- Much more than just CLBs:
  - Memory blocks
  - DSP block (hard multipliers)
  - Multigigabit transceivers
  - Clock managers
  - Hard protocols and codecs
  - Ext. memory controllers
  - ADC/DAC
- Complete System-On-Chip:
  - PowerPC/ARM
  - Ext. Memory controllers
  - Multiple I/O controllers
  - Fast interconnect



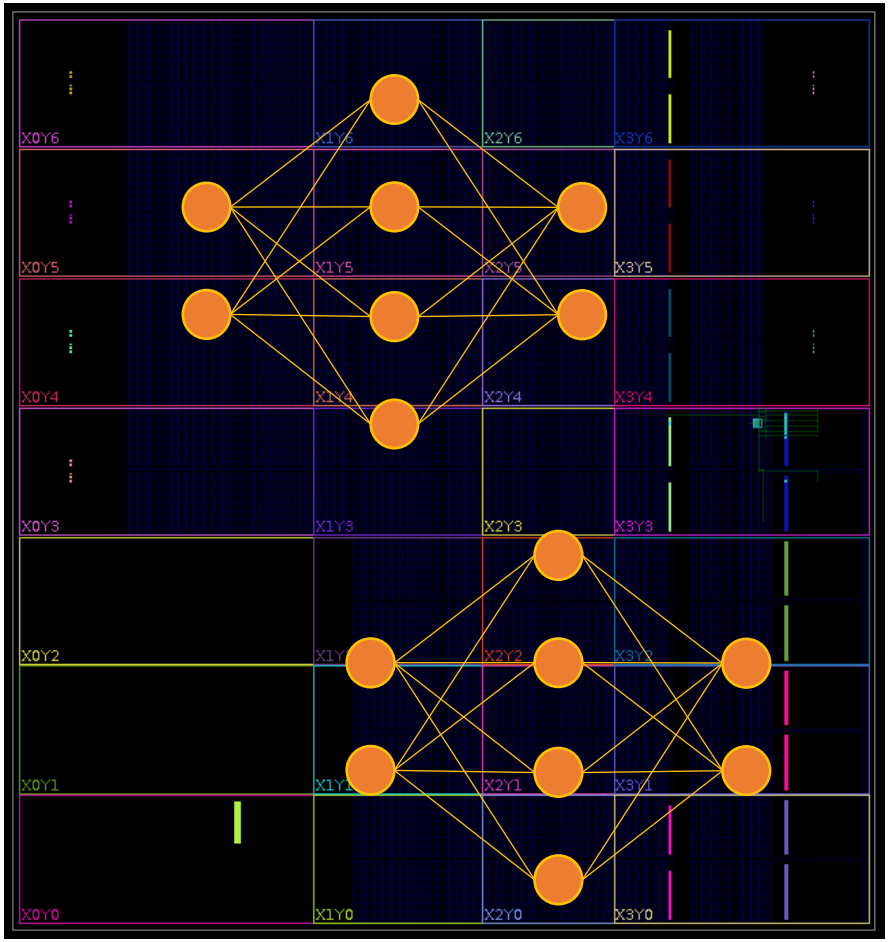
Xilinx Zynq MPSoC - infrastructure Xilinx.com

Natural parallelism and streamlined processing

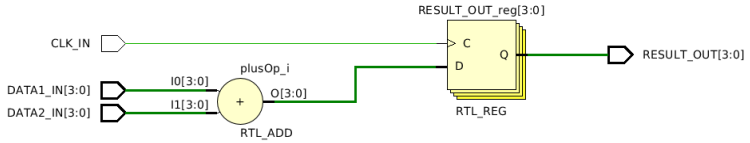


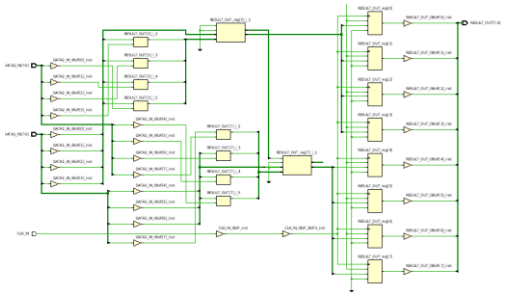
Neural Networks

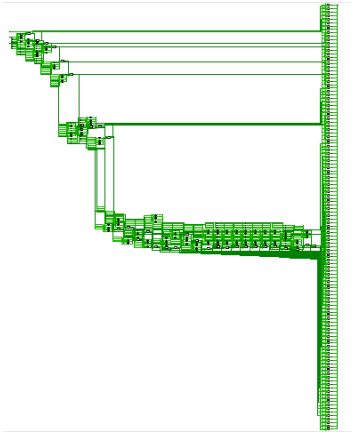
- Massive parallelization
  - Accelerated computing time
- Deterministic Latency
  - Exact time of the result in a processing pipeline
- Optimized data types
  - Better FLOPs/Watt ratio

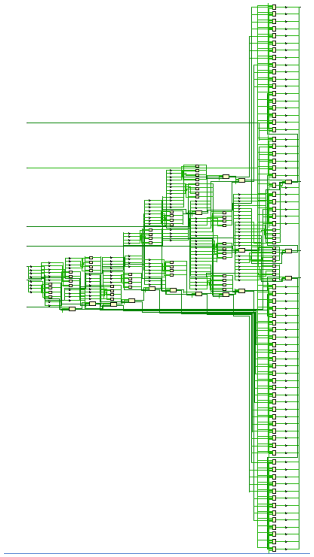


Flexible data types









Primitive Statistics

Primitive type	Count
FLOP_LATCH	8
LUT	8
CARRY	2
IO	25
CLK	1

Primitive Statistics

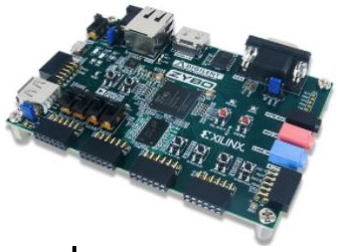
Primitive type	Count
FLOP_LATCH	124
LUT	124
CARRY	31
IO	373
CLK	1

Primitive Statistics

Primitive type	Count
FLOP_LATCH	1024
LUT	1024
CARRY	256
IO	3073
CLK	1

Device selection

- **Low-end devices** -> cheap, small form-factor, ultra-low power
  - 3.7k LUT, 5 BRAM, 10 DSP
- **High-end devices** -> ultra expensive, large form-factor, low power
  - 4M LUT, 3.7k BRAM + 1.2k URAM, 12.2k DSP, 128 MGT (32 Gbps)
- Everything in between



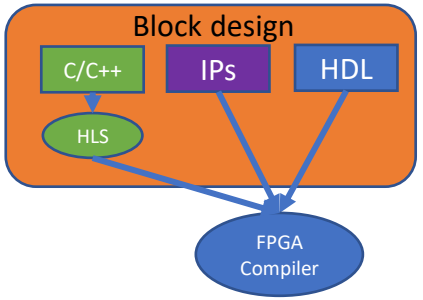
- **Edge**
  - Ready to use, standalone platforms, large selection of add-on cards
- **Cloud**
  - Accelerator cards, host – PCIe card

How to use these resources?

- Hardware Description Languages: Verilog/VHDL
  - Control over each flip-flop and clock cycle
  - Difficult, time consuming development and debugging cycles
- High-level tools
  - C++ to HDL converters (High-Level Synthesis)
  - Loss of performance, gain in solution-to-market
  - Accelerated libraries
  - Complete system builders
- However, HDL always in the end

```
-- ==============================================================
-- RTL generated by Vivado(TM) HLS - High-Level Synthesis from C, C++ and SystemC
-- Version: 2018.3
-- Copyright (C) 1986-2018 Xilinx, Inc. All Rights Reserved.
--
-- ==============================================================

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity adder is
port (
    ap_start : IN STD_LOGIC;
    ap_done : OUT STD_LOGIC;
    ap_idle : OUT STD_LOGIC;
    ap_ready : OUT STD_LOGIC;
    data1 : IN STD_LOGIC_VECTOR (31 downto 0);
    data2 : IN STD_LOGIC_VECTOR (31 downto 0);
    ap_return : OUT STD_LOGIC_VECTOR (31 downto 0) );
end;


architecture behav of adder is
    attribute CORE_GENERATION_INFO : STRING;
    attribute CORE_GENERATION_INFO of behav : architecture is
    "adder,hls_ip_2018_3,{HLS_INPUT_TYPE=cxx,HLS_INPUT_FLOAT=0,HLS_INPUT_FIXED=0,HLS_INPU
    constant ap_const_logic_1 : STD_LOGIC := '1';
    constant ap_const_logic_0 : STD_LOGIC := '0';
    constant ap_const_boolean_1 : BOOLEAN := true;



begin



    ap_done <= ap_start;
    ap_idle <= ap_const_logic_1;
    ap_ready <= ap_start;
    ap_return <= std_logic_vector(unsigned(data2) + unsigned(data1));
end behav;
```

```
#include "adder.h"

int adder(int data1, int data2) {
  return data1 + data2;
}
```

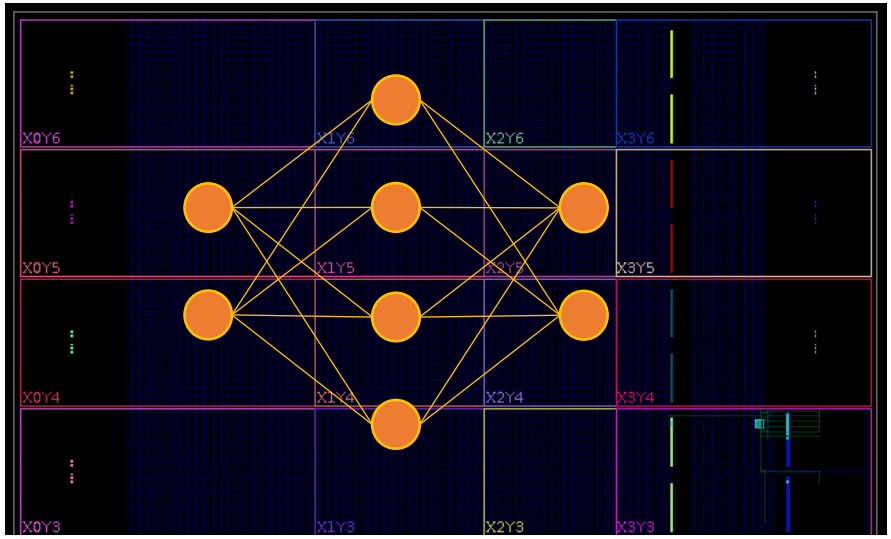


Neural Networks on FPGAs

- Full model
  - Pros:
    - Fits entire model into programmable resources
    - No supervisor
    - Direct processing
    - Lowest, fixed latency
    - Any data type
    - No ext. memory req.
  - Cons:
    - High resource consumption
    - Limited memory capacity
    - Limited types of layers
- Decomposed model
  - Pros:
    - Any model can be compiled
    - High-level model optimizers
    - Moderate resource consumption
    - Subset of data types support
  - Cons:
    - Requires supervisor
    - Requires ext. memory
    - Subset of data types support

Full model

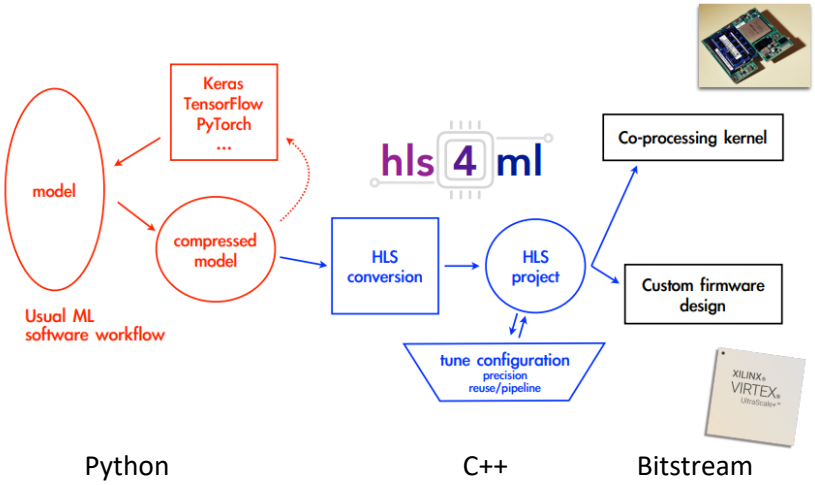
- Decomposition of the entire model into series of matrix operations
- Construction of pipelined sequence of operation blocks (layers)
  - Highest throughput, lowest latency
- Implementation of operations on LUT and DSP, weights stored in FF
  - Very high resource consumption, resources limits



Full model

- Integration of HDL logic generation with Python
  - NN model converted to C++ code
  - High Level Synthesis used to convert C++ into HDL
  - Full control over data types
  - Moderate control over resource consumption vs latency

```
import hls4ml
config = hls4ml.utils.fetch_example_model('KERAS_3layer.json')
hls_model = hls4ml.converters.keras_to_hls(config)
hls_model.build()
```



J. Duarte et al., "Fast inference of deep neural networks in FPGAs for particle physics", JINST 13 P07027 (2018), arXiv:1804.06913.

Full model

- Application in hardware trigger for CMS experiment in CERN
  - Trigger: fast feature extraction for decision making
  - Hardware level: sustain incoming 40 MHz collision rate, true real-time processing

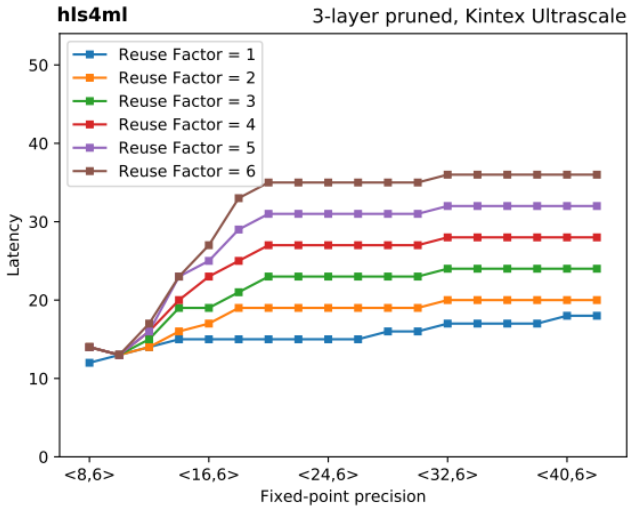
  - CPU/GPU not viable: fast but not real-time
  - FPGA over ASIC: reprogrammability

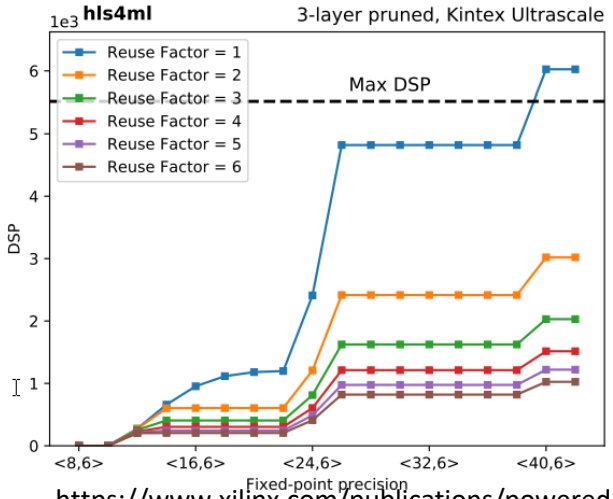
Figure 2: Compared to alternative devices such as GPUs and ASICs, FPGAs are the only viable choice for the event trigger processing because they provide extremely low latency. While the large numerical processing capability of GPUs is attractive, these technologies are optimized for high throughput, not low latency.

1GHz -> 1 ns

Artificial Intelligence Accelerates Dark Matter Search

Integrating Inference Acceleration with Sensor Pre-processing in Xilinx FPGAs Delivers Performance Unachievable by GPUs and CPUs

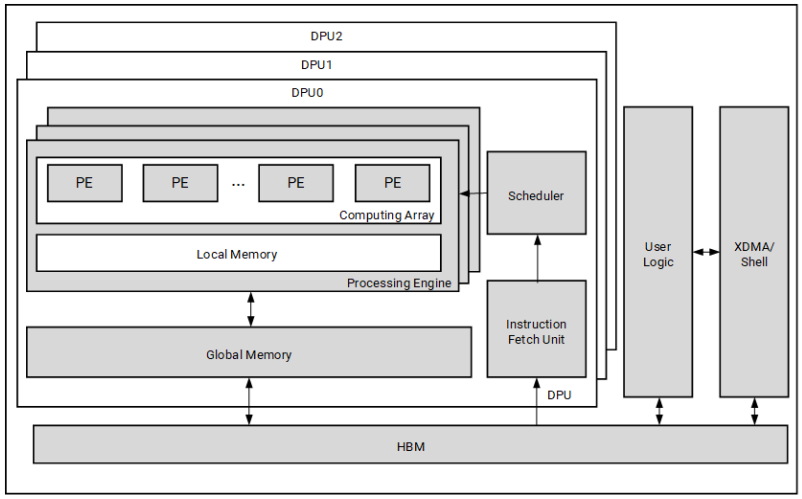




https://www.xilinx.com/publications/powered-by-xilinx/cerncasestudy-final.pdf

Decomposed model

- Model decomposition into a set of sequential instructions dedicated processor
- Processor – Deep Learning Processing Unit (DPU)
  - Configurable entity to instantiate in programmable resources
  - Instruction set optimized for NN inference
  - Instructions and weights stored in external memory



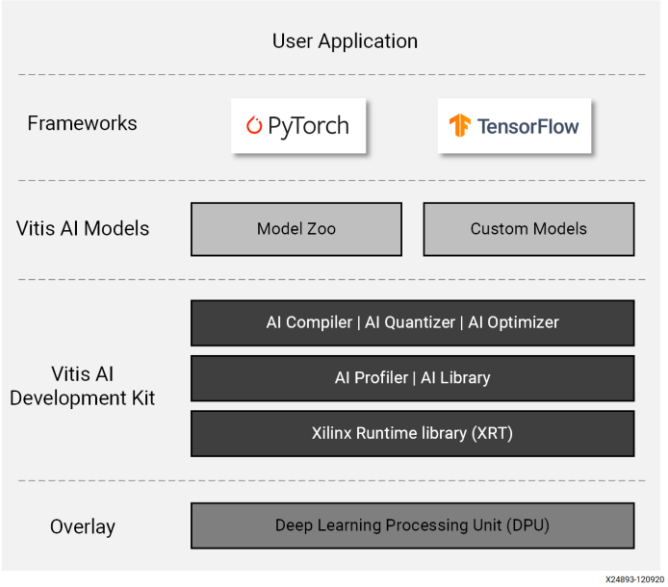
https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Alveo-U50LV/U55C-Card-DPUCAHX8H

Requires supervisor!
Breaks true real-time chain

Host processor:
- CPU on PCIe based platforms
- ARM cores in SoC platforms

Decomposed model

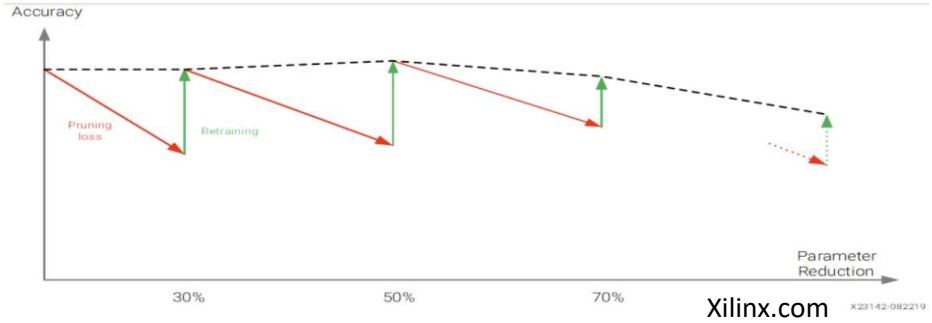
- AMD/Xilinx Vitis AI stack
  - Integrated with ML frameworks
  - Model optimizers:
    - Quantization
    - Pruning
  - Model compiler into DPU instructions
  - DPU logic components to manual instantiations
  - Ready to use bitfiles with DPU instances



https://docs.xilinx.com/r/en-US/ug1414-vitis-ai/Alveo-U50LV/U55C-Card-DPUCAHX8H

Decomposed model

- Model optimizations:
  - Iterative, offline processes: apply change -> retrain



Xilinx.com

  - Quantization
    - Model size reduction (in MB) while maintaining performance
    - Limited data types selection based on DPU configuration
      - E.g. INT8: 4x size reduction, 0.1% accuracy drop (ResNet 50)
  - Pruning
    - Model nodes removal while maintaining performance
      - E.g. ResNet50, 46% less parameters, 1% accuracy drop
      - E.g. Custom CNN for FMNIST: 90% less parameters, 2% accuracy drop

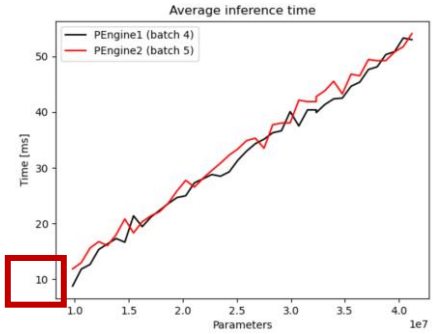
G. Kozioł – UJ Master thesis, under development

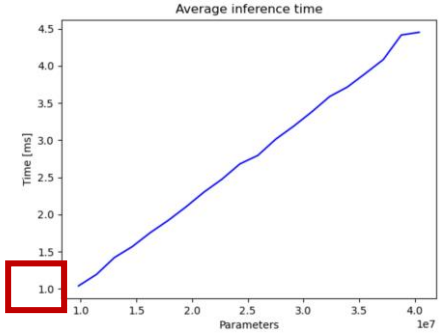
Decomposed model

- Performance evaluation
  - ResNet50 model with additional layers
  - 2 DPU bitstreams evaluated: throughput and latency optimized

Throughput optimised:

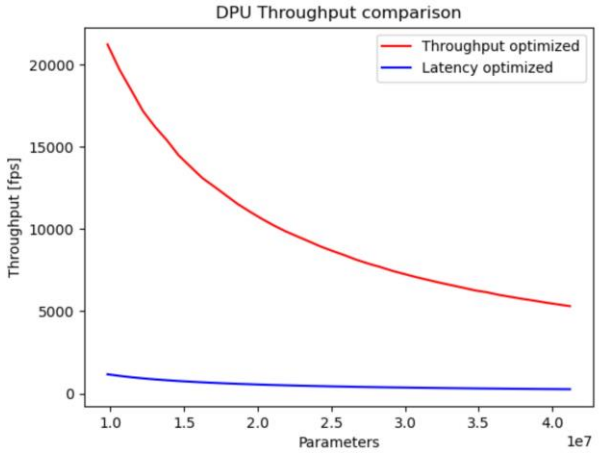
Large data batches mixed and processed by all DPU instances

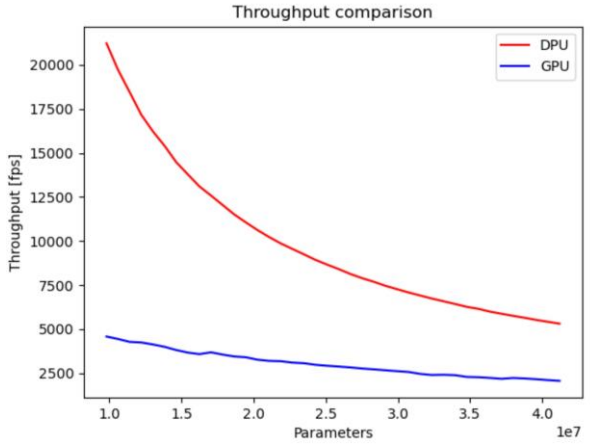




Latency optimised:

Single inference employs all resources, long inference switching time





GPU: Nvidia RTX 2080

G. Kozioł – UJ Master thesis, under development

Why inference only?

- FPGAs work great when:
  - Operations can be pipelined
  - Required memory can fit into embedded resources
  - Memory access is sequential and continuous
  - Input data comes from built in trasceivers
  - Arithmetics are simple (+, -, *)
- GPUs have superior performance in NN training

Summary

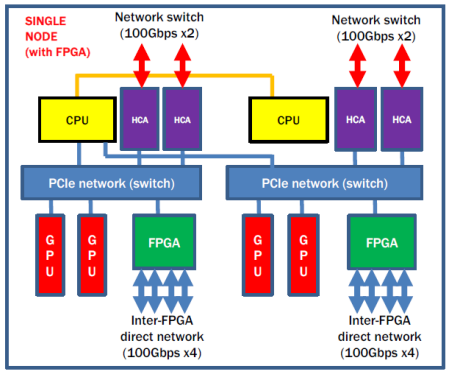
- FPGAs have unique set of features for NN
  - Real-time data processing
  - Ultra-low latency or power applications
  - Sensor Fusion
  - Adaptation in time – reprogramability
- Growing set of high-level development tools
  - Ready to use hardware platforms
  - Accelerated libraries
  - System builders
  - Python integration

Is it hard to use FPGAs?

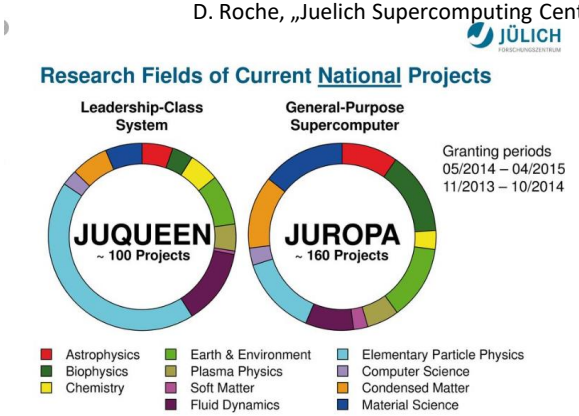
```python
import pynq

devices = pynq.Device.devices
for i in range(len(devices)):
    print("{}) {}".format(i, devices[i].name))

ol = pynq.Overlay("binary_container_1.xclbin")

kernel = ol.function_low_1

in1 = pynq.allocate((1024,), 'u4', target=ol.HBM0)
in2 = pynq.allocate((1024,), 'u4', target=ol.HBM1)
in3 = pynq.allocate((1024,), 'u4', target=ol.HBM2)
out = pynq.allocate((1024,), 'u4', target=ol.HBM3)

in1.sync_to_device()
in2.sync_to_device()
in3.sync_to_device()

kernel.call(in1, in2, in3, out, 1024)

out.sync_from_device()

ol.free()
```

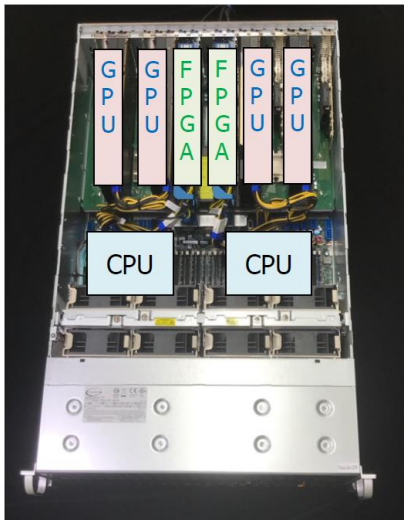
HPC Example

- Cygnus – Center for Computational Sciences, Tsukuba, Japan



D. Roche, „Juelich Supercomputing Centre"





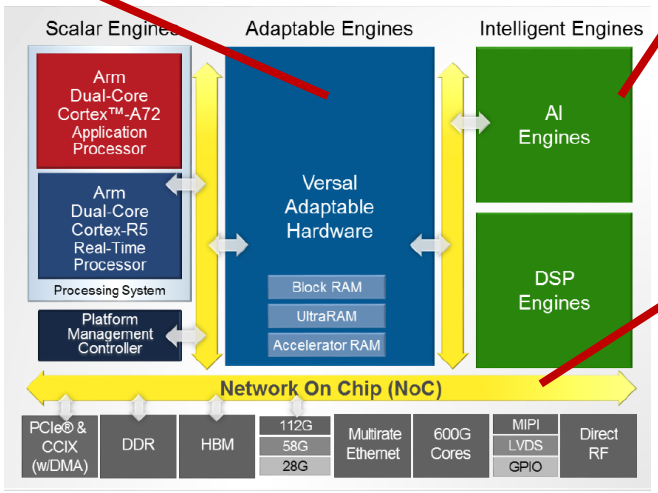
T. Boku, „Japanese Supercomputer development and hybrid accelerated supercomputing"

Xilinx.com

What is next to come

- 7nm Versal Architecture

General CLB resources:
- 4x larger blocks (32 LUTs)
- Less routing
- Higher frequencies



Interconnected vector SIMD:
- 1GHz frequency
- 512b floating point vector
- Local/shared memory
- Streamlined designs

High bandwidth interconnect

Xilinx.com

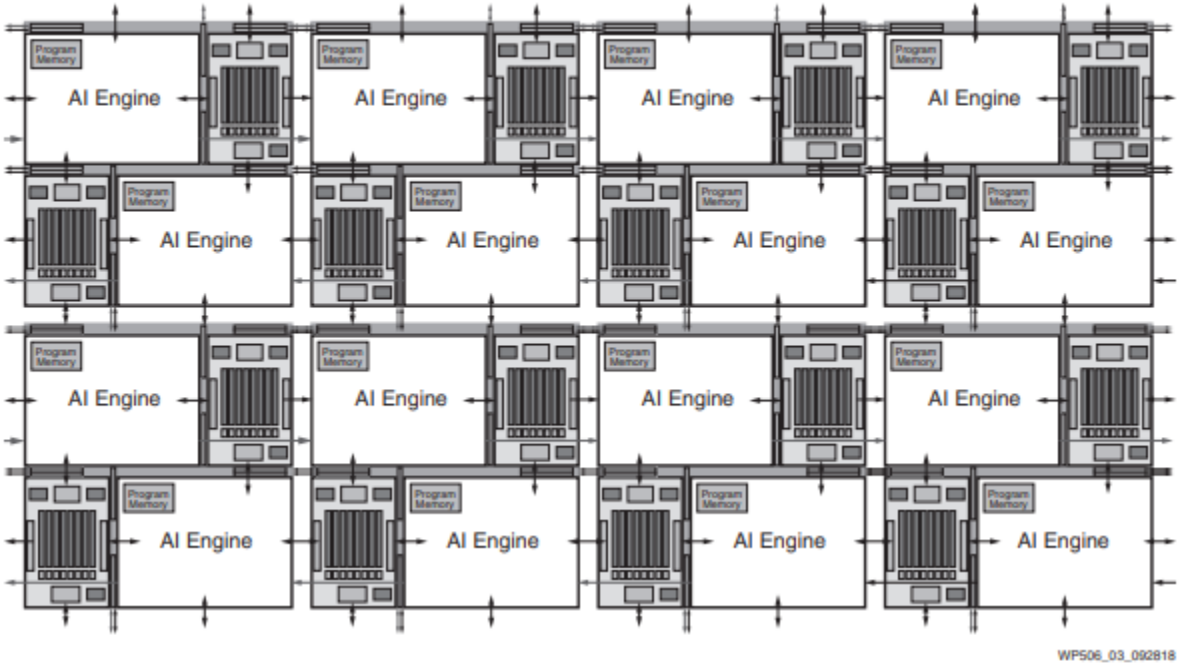


Figure 3: **AI Engine Array**

Each AI Engine tile includes vector processors for both fixed and floating-point operations, a scalar processor, dedicated program and data memory, dedicated AXI data movement channels, and support for DMA and locks. AI Engines are a single instruction multiple data (SIMD); and very long instruction word (VLIW), providing up to 6-way instruction parallelism, including two/three scalar operations, two vector load and one write operation, and one fixed or floating-point vector operation, every clock cycle.

Optimized for real-time DSP and AI/ML computation, the AI Engine array provides deterministic timing through a combination of dedicated data and instruction memories, DMA, locks, and