Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

ICSC
Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

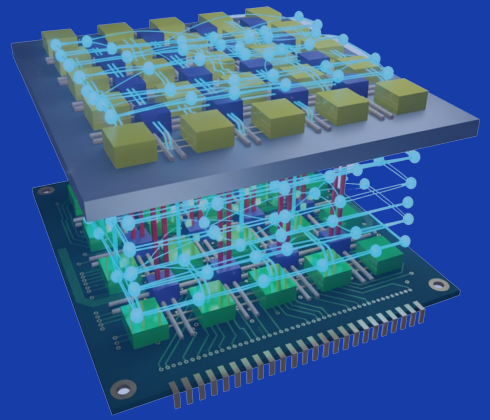
Centro Nazionale di Ricerca in HPC, Big Data e Quantum Computing

2nd AI-INFN Advanced Hackathon - Pavia, 26 November 2025

Giulio Bianchini

Introduction to Field Programmable Gate Array (FPGA) and Machine Learning on FPGA

Outline



I. Context

II. Where Are FPGAs Used?

III. What is an FPGA?

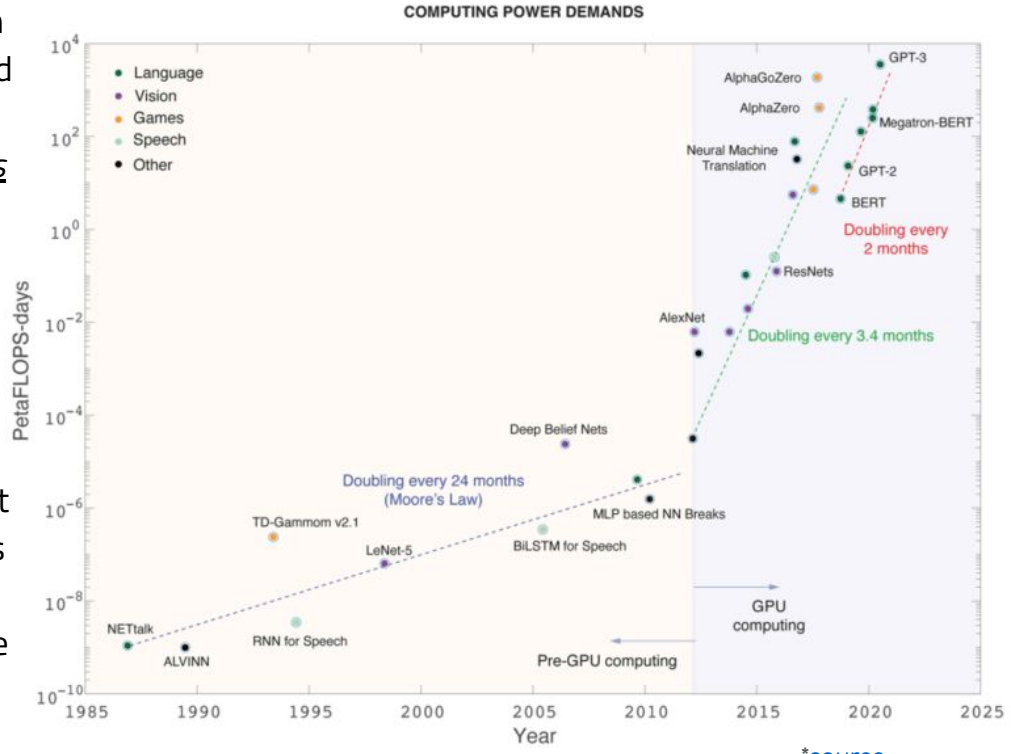
IV. FPGA Ecosystem

V. FPGA Development

VI. Machine Learning Inference on FPGA

Context

- **Increasing demand for computing power**: Especially in Machine Learning (ML) and in software (algorithms) and hardware (specialized devices). *Until 2012, computing power demand doubled every 24 months; recently this has shortened to approximately every two months**
- **Compute demand requires hardware advancements**: Growth compute demand requires new hardware solutions - Interest in FPGAs as a promising solution
- **Inference phase challenges**: In production environments, the inference accounts for 90% of compute cost**. Energy consumption is a critical aspect (it is estimates that a single query for ChatGPT requires 2.9W***)
- **Overcome traditional architectures**: Explore innovative computing architecture to provide alternative less energy-hungry yet performance solution to overcome traditional architectures (GPU, CPU)



*source
**source
***source

Emerging Architectures

Compute demand is exploding and traditional CPUs and GPUs alone can't keep up with the need for **more performance at lower energy cost**.

A Heterogeneous Future of Computing

ARM / RISC-V

Energy-efficient RISC architecture increasingly dominant in HPC, mobile, and servers

Used in modern HPC clusters

Ideal for parallel workloads and edge deployment

FPGA

Right balance between energy efficiency and specialization

Programmable like software

Efficient like ASICs

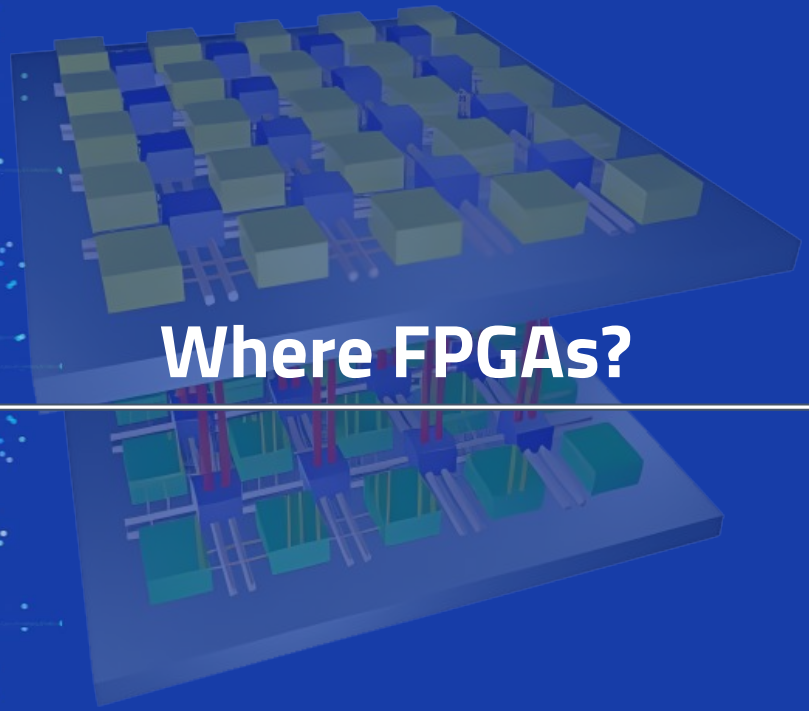
Scalable like ARM/RISC-V systems

AI Accelerators (ASICs)

Purpose-built silicon for ML workloads (ex. Google TPU v4)

Extremely high throughput and efficiency, but limited flexibility

Best fit for large-scale training and massive inference workloads



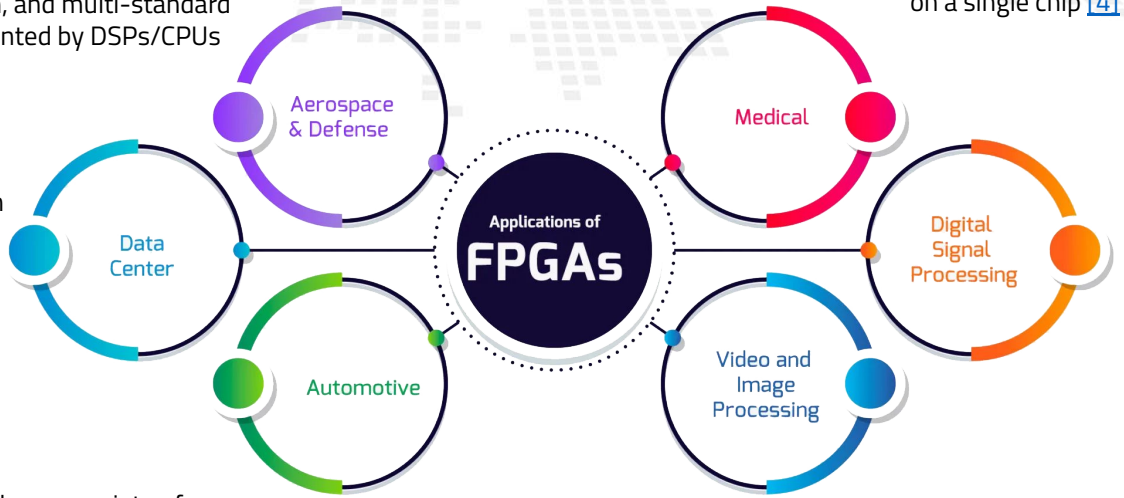
Where FPGAs?

FPGAs in industries: a few examples

Modern SDR platforms use FPGAs for high-speed data acquisition, digital filtering, modulation/demodulation, and multi-standard channelization, complemented by DSPs/CPUs for control tasks [5]

FPGA-based MRI processing modules capable of performing real-time filtering and image reconstruction on a single chip [4]

The "Catapult" project, which later evolved into "AzureBoost," is one of the first large-scale examples of FPGA usage in the cloud [1].



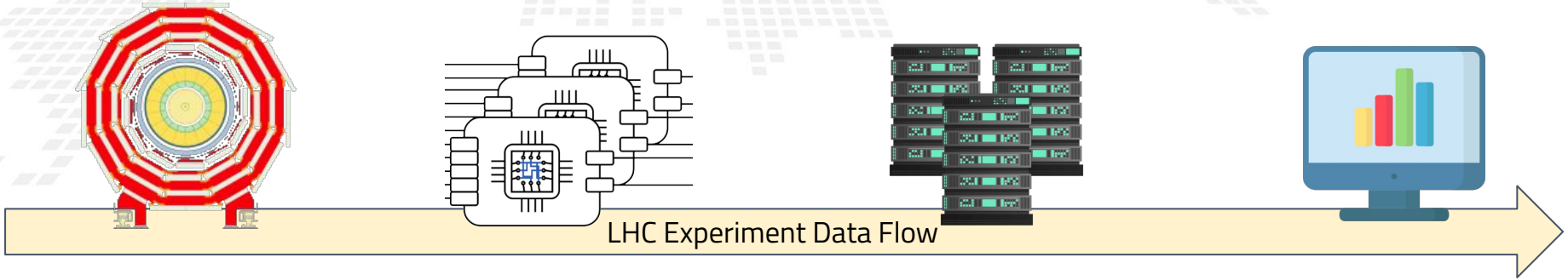
NASA JPL implemented the digital signal processors for an airborne precipitation-measurement radar entirely on FPGAs, enabling onboard pulse compression, filtering, and FFT processing with low power consumption [2]

Autonomous vehicles rely on a variety of sensors, including LiDAR, radar, cameras, and ultrasonic sensors, to perceive their surroundings and FPGAs widely used to process the massive amounts of data generated in real time [3]

CTAccel partnered with Xilinx to deliver an FPGA-based image/video processing accelerator (for the cloud) in production since ~2016 [6]

FPGAs in scientific applications: a few examples

FPGAs are widely used in scientific applications requiring flexible, high-speed, low-latency computing, for example in particle-physics detectors where they perform real-time event selection



Detector

produces
data at high rate
- Raw data rate: hundreds of terabytes per second.
- Impossible to store or analyze everything offline.

L1 Trigger (Level-1 Trigger)

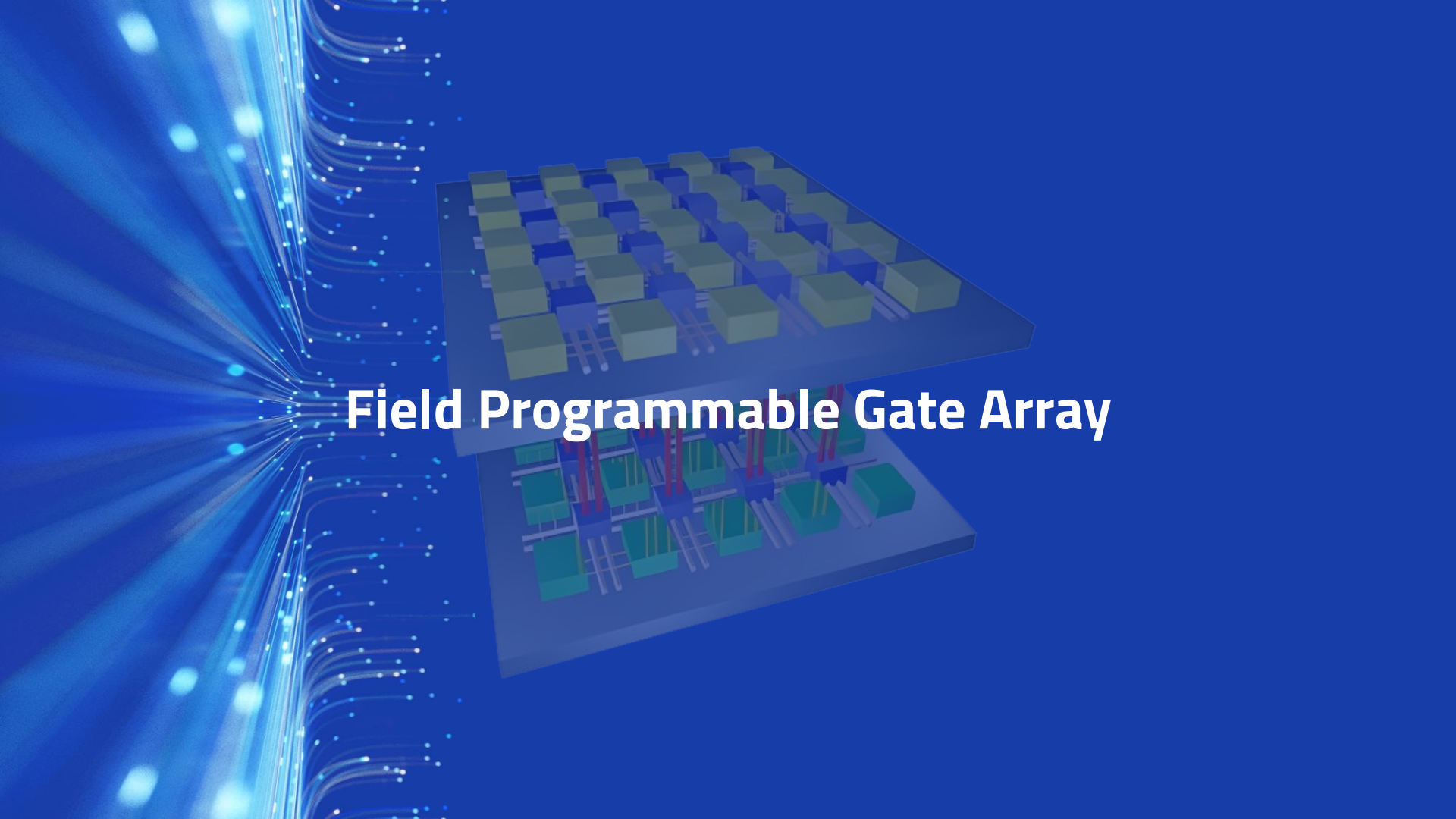
Implemented entirely in **FPGAs**
- filter relevant collision events
- Decisions in (O(μs))

High-Level Trigger (HLT)

Computing farm
- Runs on a large cluster of CPUs/GPUs
- Less time constraint

Offline processing

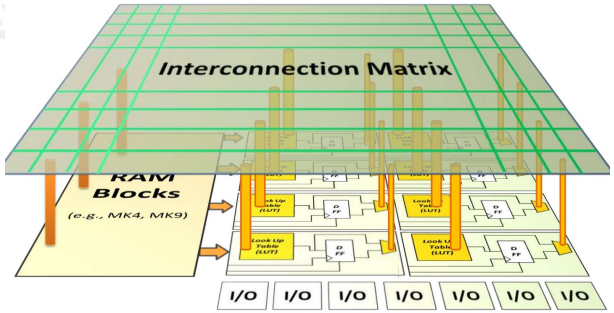
- Stored to long-term disk/tape for deep physics analysis.
- No time constraint

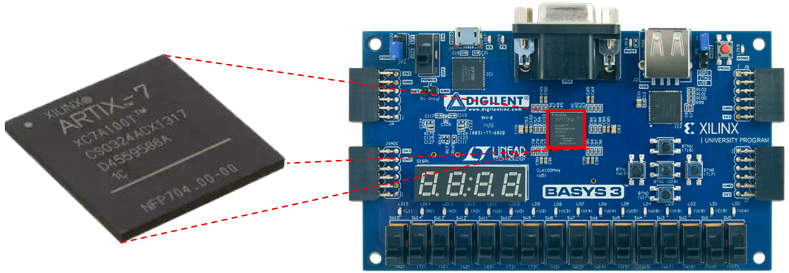

Field Programmable Gate Array

FPGA - Basic FPGA Architecture

An FPGA is a configurable hardware device that can be programmed to instantiate any form of digital logic.

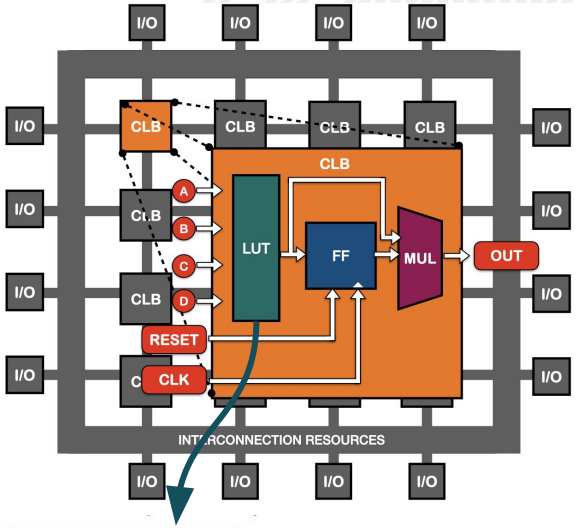
- **Matrix of Configurable Logic Blocks (CLBs)**
  - These blocks can implement arbitrary logic functions, and they are the basic building elements.
- **Programmable interconnect network**
  - routing matrix of switch boxes and wiring channels
- **I/O blocks at periphery**
  - I/O blocks handle communication with the outside world
- **Additional resources**:
  - BRAM (memory)
  - DSP slices
  - ...

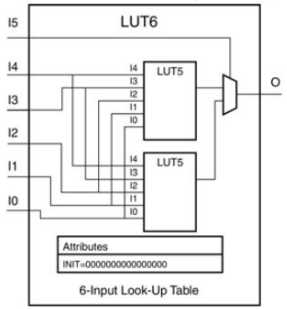




Basys3, example of development board

What is inside a CLB?





Example: 6-input LUT ⇒ implements any Boolean function of 6 inputs

- **Lookup Tables (LUTs)**
  - A small configurable piece of memory that stores the truth table of a logic function.
- **Flip-Flops / Registers**
  - Clocked storage elements that hold data between cycles, enabling pipelining and sequential logic.
- **(MUXes) Multiplexers**
  - Configurable switches that select between multiple signals, shaping data paths and combining LUT outputs.
- **Carry chains**:
  - Dedicated fast paths for propagating carries, enabling efficient addition, subtraction, and counters.

And the additional resources?

Beyond LUTs and routing: the heterogeneous FPGA ecosystem

DSP Blocks (Digital Signal Processing Units)

- Hardware-optimized arithmetic units (e.g., $A \times B + C$)
- Very high-speed Multiply–Accumulate (MAC) operations
- Essential for filters, linear algebra, and **ML kernels**

BRAM (Block RAM)

- Dedicated 18 Kb / 36 Kb on-chip memory blocks
- Faster and denser than LUT-based "distributed RAM"
- Used for buffers, FIFOs, feature maps, small caches

URAM (UltraRAM)

- 288 Kb high-density memory blocks (newer devices)
- Cascadable for deep, wide memory structures
- Ideal for large buffers or dataflow pipelines

Hard Processors (SoC – Processing System)

- Embedded ARM / RISC-V cores @ GHz speed
- Used for OS, drivers, and high-level orchestration

Soft-Core Processors (in FPGA fabric)

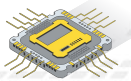
- Synthesized CPUs: MicroBlaze, Nios II, RISC-V
- Fully customizable for embedded control tasks

High-Speed Transceivers

- Support PCIe, 10G/40G/100G Ethernet, etc.
- Enable high-bandwidth external communication

and even more...

How FPGA works? (CPU vs GPU vs FPGA vs ASIC)



Central Processing Unit

CPU

Graphics Processing Unit

GPU

Field Programmable Gate Array

FPGA

Application-Specific Integrated Circuit

ASIC

Flexibility, programming abstraction ⟵ ⟶ Performance, energy efficiency

- **Executes one (or a few) instructions at a time**
- Optimized for flexibility and control flow
- Great for general-purpose workloads

- **Thousands of lightweight threads running in parallel**
- Hardware is massively parallel
- Ideal for dense numeric workloads (ML training, graphics)

- The algorithm is compiled into a custom digital circuit
- LUTs, DSPs, BRAM, routing physically wired for the task
- **No instructions and no threads, the hardware is the algorithm**
- True hardware parallelism and pipelining
- Fully reconfigurable for a different design

- A custom chip built specifically for one workload
- **Highest performance and efficiency but fixed forever after fabrication**
- Lowest latency and power consumption
- Long development time, high cost

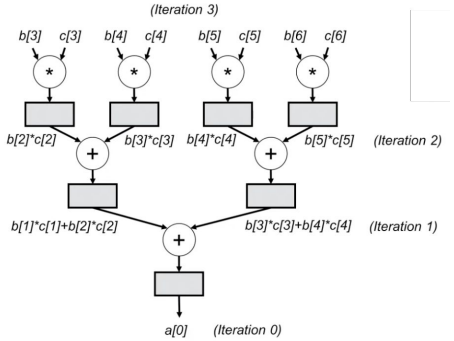
FPGA Parallelism

Parallelism is built directly into the fabric: millions of identical, programmable logic cells operating simultaneously

FPGA's architecture is inherently designed for parallelism!

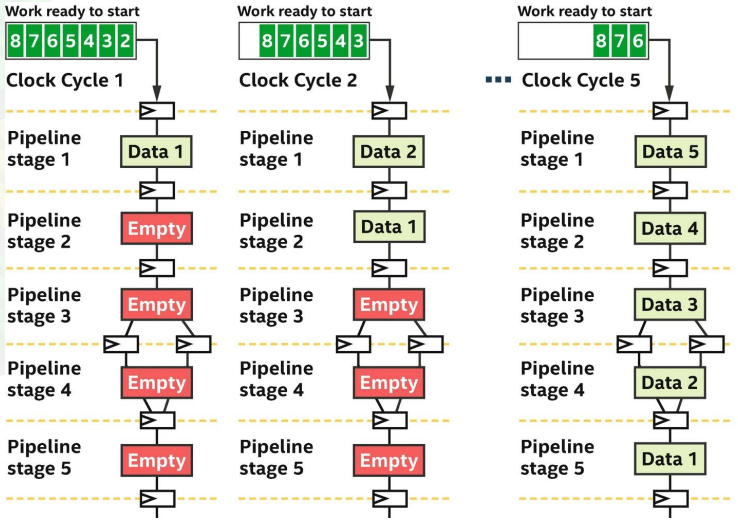
Pipeline Parallelism

Pipeline parallelism breaks computation into stages that operate concurrently on dedicated hardware allowing multiple data elements to be processed simultaneously



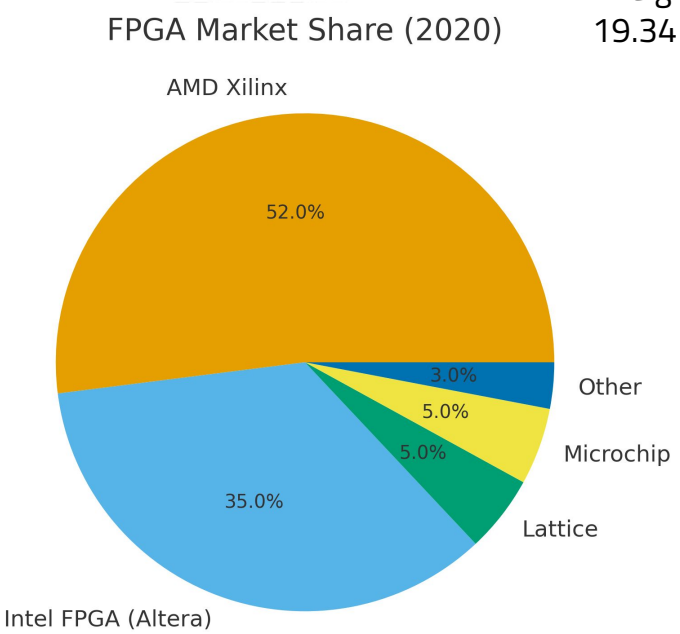
Example of a pipelined loop computation at cycle 4

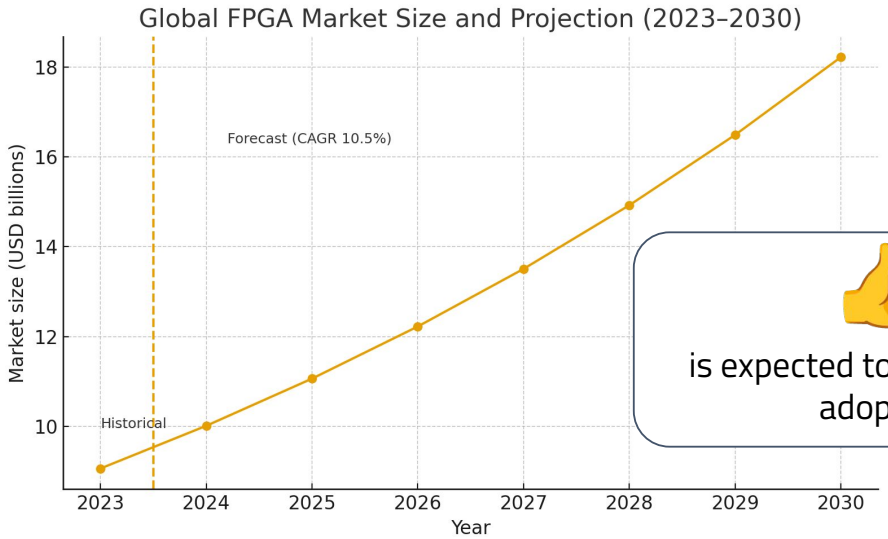
```
for (i=0; i < 1000; i++) {
    a[i] = b[i]*c[i]+ b[i+1]*c[i+1]
         + b[i+2]*c[i+2] +b[i+3]*c[i+3];
}
```



FPGA Vendors

The global FPGA market is expected to rise from USD 11.73 billion in 2025 to USD 19.34 billion by 2030 (CAGR 10.5%), driven primarily by the growing use of AI and IoT





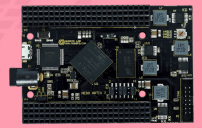
is expected to see growing adoption

Source: https://www.nextmsc.com/report/field-programmable-gate-array-fpga-market

FPGA families

Every vendor provides a portfolio of FPGA families, each designed to target specific use cases such as low power, embedded processing, high-speed networking, or compute acceleration

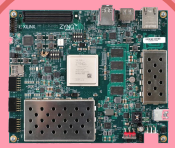
AMD XILINX



Artix-7

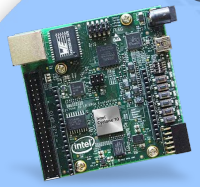
Spartan-7

Kintex-7

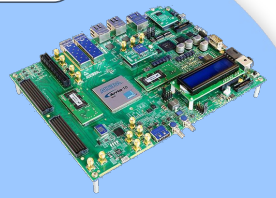


UltraScale+

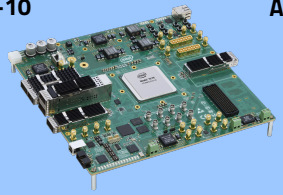
intel ALTERA



Cyclone-10

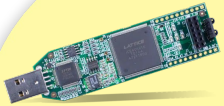


Arria-10



Stratix-10

Lattice Semiconductor Corporation



ice-40

ECP5

FPGA - Alveo

The **Alveo boards are high-performance reconfigurable acceleration cards** designed to accelerate workloads in data centers. They are **based on the Xilinx UltraScale/UltraScale+ architecture** and they are designed to work with any server, any workload, and any set of applications to deliver significant performance improvements over CPU-only systems

Alveo U55C

FPGA specs	
Architecture	UltraScale+
LookUp Tables	1,000,000
Flip-Flops	2,000,000
DSP Slices	3,840
Block RAM	36 MB
Card specs	
Form Factor	Half-Height, Half-Length Single Slot Low-Profile
HBM2 Memory	16 GB
HBM2 Bandwidth	460 GB/s
Network Interface	1 x QSFP28 (100GbE)
Clock Precision	IEEE 1588
PCI Express	PCIe Gen4 x 16
Thermal Solution	Passive
Power (TDP)	150W

ok very powerful… but how do I use this ~5000$ beast?

FPGA Development Flow - Hardware Description Language

HDL flow

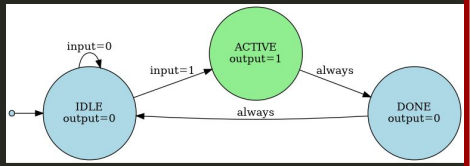
1. Hardware design
2. RTL synthesis
3. Netlist synthesis
4. Implementation

steps performed by Vivado/Quartus

Firmware

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity simple_fsm is
    Port ( clk    : in  STD_LOGIC;
           reset  : in  STD_LOGIC;
           input  : in  STD_LOGIC;
           output: out STD_LOGIC);
end simple_fsm;

architecture Behavioral of simple_fsm is
    type state_type is (IDLE, ACTIVE, DONE);
    signal current_state, next_state : state_type;
begin
    -- State register
    process(clk, reset)
    begin
        if reset = '1' then
            current_state <= IDLE;
        elsif rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    -- Next state logic
    process(current_state, input)
    begin
        case current_state is
            when IDLE =>
                if input = '1' then
                    next_state <= ACTIVE;
                else
                    next_state <= IDLE;
                end if;
            when ACTIVE =>
                next_state <= DONE;
            when DONE =>
                next_state <= IDLE;
        end case;
    end process;

    -- Output logic
    output <= '1' when current_state = ACTIVE else '0';

end Behavioral
```

Example of Finite State Machine

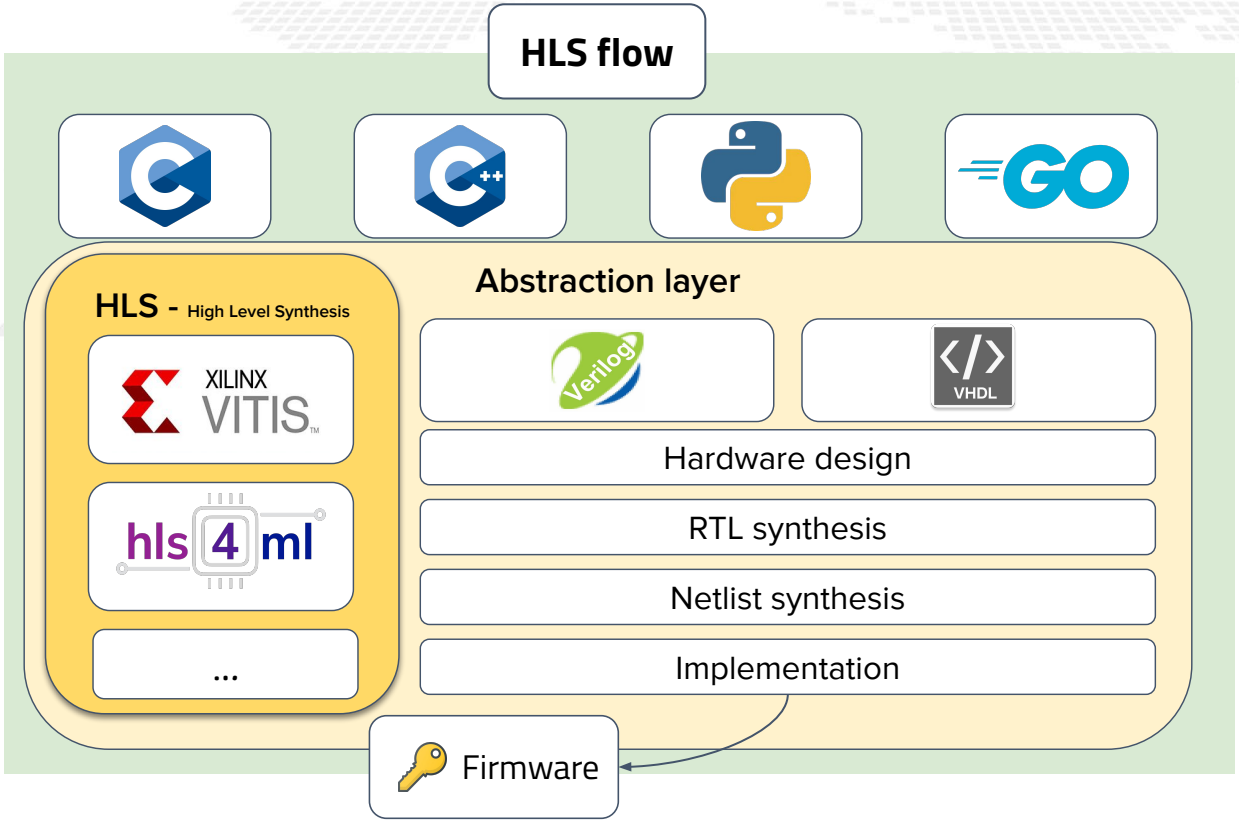


The hardware design step is not only writing HDL code ...

- Writing constraints
- Simulation setup
- IP integration
- High-level block design

Hardware-level steps that make this flow complex

FPGA Development Flow - High Level Synthesis



High-level thinking, hardware-level performance.

```
#include "ap_int.h"

// Simple array multiplier function
void array_multiplier(
    int a[100],         // Input array A
    int b[100],         // Input array B
    int result[100],    // Output array
    int size            // Array size
) {
    #pragma HLS INTERFACE mode=m_axi port=a bundle=gmem0
    #pragma HLS INTERFACE mode=m_axi port=b bundle=gmem1
    #pragma HLS INTERFACE mode=m_axi port=result bundle=gmem2
    #pragma HLS INTERFACE mode=s_axilite port=size
    #pragma HLS INTERFACE mode=s_axilite port=return

    // Main computation loop
    MULT_LOOP:
    for(int i = 0; i < size; i++) {
        #pragma HLS PIPELINE II=1
        result[i] = a[i] * b[i];
    }
}
```

The algorithm is written in a high-level language, and an HLS tool ('middleware') generates the corresponding RTL finally used to generate the firmware.

FPGA Development Flow

HDL flow

Verilog | VHDL

Hardware design

RTL synthesis

Netlist synthesis

Implementation

HLS flow

C | C++ | Python | GO

HLS - High Level Synthesis

XILINX VITIS

hls4ml

...

Abstraction layer

Verilog | VHDL

Hardware design

RTL synthesis

Netlist synthesis

Implementation

Firmware

✓ HDL - Pro

Full Design Control: fine-grained control of timing, registers, interfaces

Peak Performance: best QoR when carefully hand-optimized

Protocol Flexibility: ideal for custom interfaces and control logic

✗ HDL – Cons

Low Abstraction: more code and slower development

Steep Learning Curve: requires cycle-accurate hardware mindset

Slow Iteration: algorithm changes require heavy low-level code edits

✓ HLS - Pro

High Abstraction: faster development and easier algorithm exploration

Rapid Debug: fast C-level simulation and easier testing

Efficient for Compute: great for ML and image processing datapaths

✗ HLS – Cons

Indirect Control: hardware structure depends on tool and pragmas

Needs Fine-Tuning: performance may suffer without proper directives

Control Logic Challenges: manually written low-level logic can be clearer and more reliable

FPGA Development Flow

HDL flow

TODAY

Verilog

Netlist synthesis

Implementation

HLS flow

1

Abstraction layer

HLS - High Level Synthesis

XILINX VITIS

hls4ml

...

2

Verilog

VHDL

Hardware design

RTL synthesis

Netlist synthesis

Implementation

3

Firmware

FPGA

keywords to remember

FPGA: Blank, reconfigurable hardware that becomes a custom circuit for a specific task

CLB (Configurable Logic Block): Basic building block combining LUTs, FFs, carry chains, muxes

LUT (Lookup Table): Small memory implementing any Boolean function of N inputs

Flip-Flop / Register: Clocked storage for pipelining and sequential logic

BRAM (Block RAM): On-chip dedicated memory for buffers, FIFOs, feature maps

DSPs: High-speed arithmetic block for MACs, filters, linear algebra, ML

Pipelining: Parallel execution of computation stages, producing one output per cycle

Parallelism: Hardware-level concurrency — many operations happening at once

HLS (High-Level Synthesis): High-level design flow that generates RTL automatically

Bitstream/Firmware: Configuration file that programs the FPGA with the generated hardware

Machine Learning (Inference) on FPGA

ML Inference on FPGAs: Why It Matters

training ≠ inference

Fixed Computation Graph

The model architecture is frozen after computation

No backpropagation needed

FPGA can implement the exact dataflow graph in hardware

Specialized, optimized pipeline

Inference specific requirements

Low batch size

Edge/embedded applications process single inputs

Real-time systems can't wait to accumulate batches

GPUs are optimized for large batches

FPGAs maintain high throughput even at batch=1

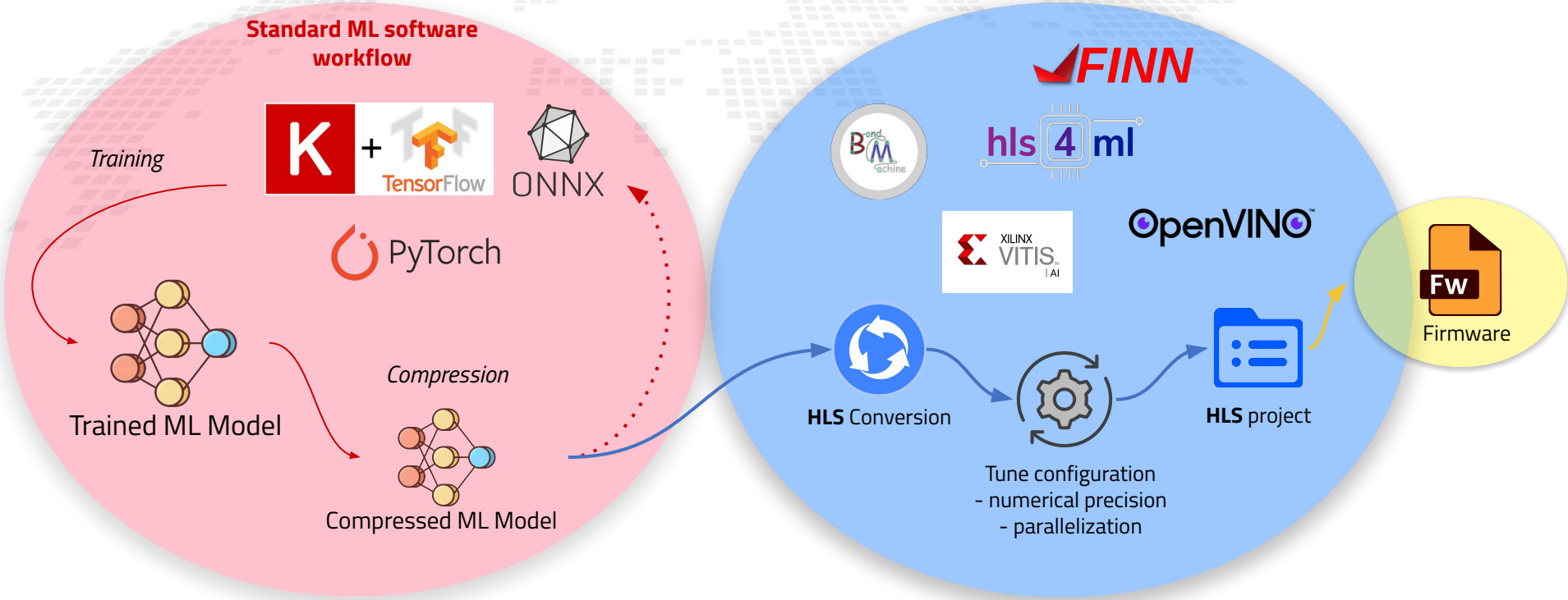
Latency vs Throughput Trade-off

Training: compute-intensive workload where you want to maximize performance on large-scale data

Inference: minimize latency (respond quickly to single input)

The FPGA pipelined architecture allows consistent low-latency predictions

ML on FPGa - High Level Flow

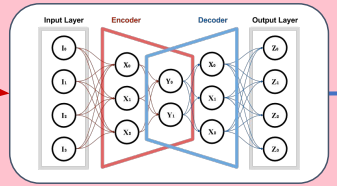


ML on FPGa - High Level Flow

Standard ML software workflow

K + TensorFlow

Training



Trained autoencoder NN

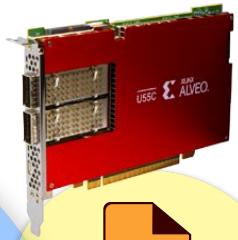
TODAY

hls4ml

HLS Conversion

Tune configuration
- numerical precision
- parallelization

HLS project

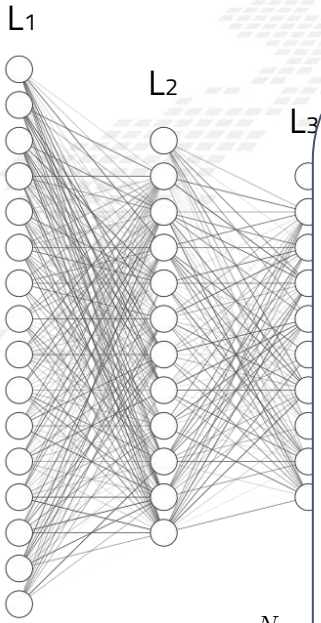


Fw

Firmware

Already built ~ 15 firmwares thanks to AI-INFN

Neural Network Inference on FPGA

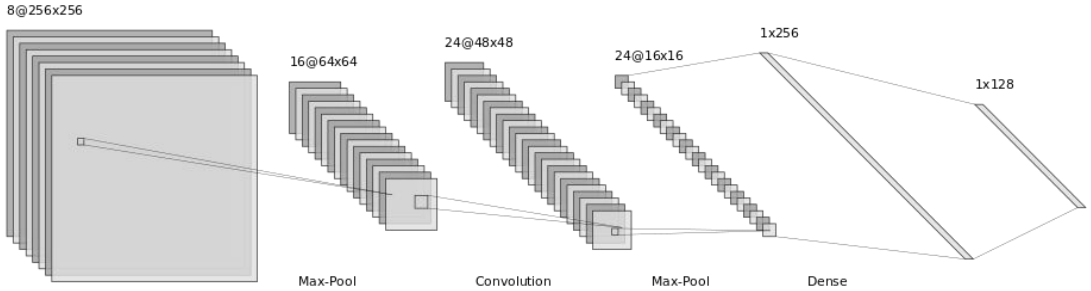


$$[N_{\text{multiplications}} = \sum_{n=2}^{N} L$$

L_n *the number of neurons in layer*

N *total number of layers*

Consider a scenario where the neural network model is extremely large…



When a model exceeds the FPGA's capacity, what techniques can we apply to shrink the resource requirements?

…e resources?

Resource used

…s constant parameters)

…all constants stored

…ons)

…s (adders)

Optimizations - quantization

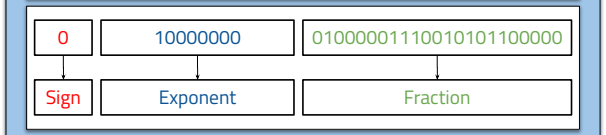
change numerical precision

The same floating point number can be represented in different ways

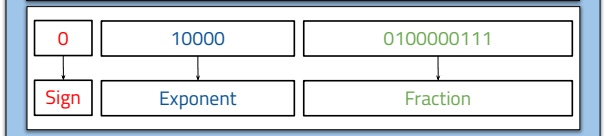
IEEE754

2.541

32 bit (single precision)



16 bit (half precision)



AP_FIXED<16,8>

Total bits **W** = 16
Integer bits: **I** = 8 (7 integer magnitude bits + 1 sign bit)
Fractional bits: **F** = W - I = 8
Step size: 2^{-8}

2.541

scale: 2.541 x 2^{8}

650.496 → **650**

0 0000010 10001010

Sign	Integer	Fraction
	2	138/256 ≈ 0.5391

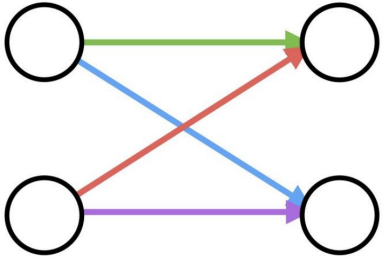
*Quantization error ≈ −0.00194 (≈ 0.076%)

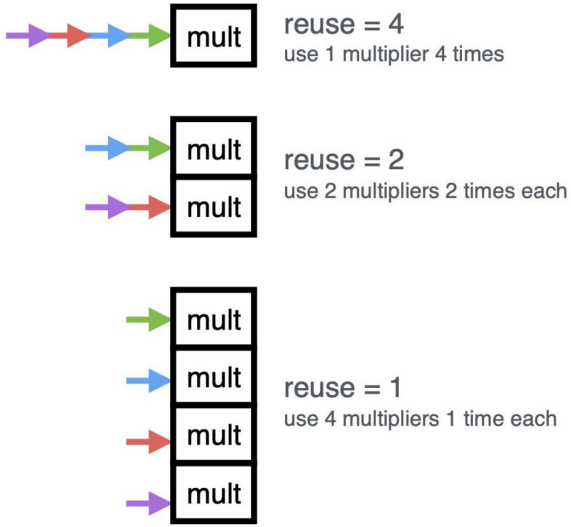
- ✓ Reduced memory usage
- ✓ Increased computational speed
- ✓ Lower power consumption
- ✗ Reduced accuracy
- ✗ Increased rounding errors
- ✗ Limited range

Optimizations

parallelization

In **hls4ml**, the reuse factor determines how much to parallelize operations in a hidden layer





More resources

Higher throughput

Lower latency

Fewer resources

Lower throughput

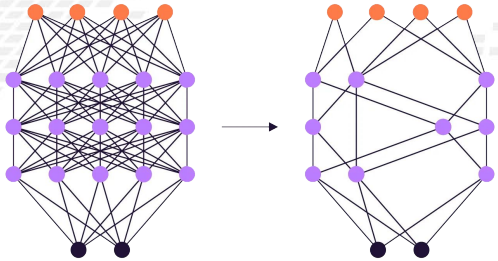
Higher latency

How strongly does the reuse factor drive the hardware design?

Compression

reduce size of a network

Pruning



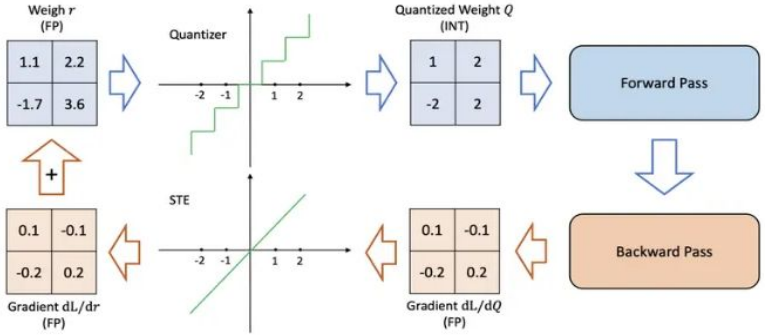
Removes weights or neurons that contribute little to the output → Creates a sparser network by eliminating low-importance connections

Reduces model size and inference cost ⇢ May require fine-tuning after pruning to recover accuracy

HOW?

- Post training → Prune the model *after* it has been fully trained
- During training → Gradually prune weights *while* the model trains

Quantization aware training (QAT)



QKeras

QKeras implements the "Quantized" version of the Keras layers (**Dense** → **QDense**, **Conv2D** → **QConv2D**, ..)

Allows precise control of bit-width for weights, activations, and biases.

HLS4ML support QKeras-trained models, that means the number of bits used in training is also used in inference

Conclusions

- **FPGAs offer a balance between flexibility and efficiency**
  - software-programmable and capable of implementing highly optimized hardware pipelines
- **Parallelism and pipelining are native features**
  - enabling extremely low latency and predictable timing
- **FPGA resources fits very well Neural network mapping**
  - run as custom hardware datapaths rather than instruction-based programs
- **HLS tools accelerate development**
  - making it easier to translate high-level ML algorithms into efficient low-level code
- **The FPGA ecosystem is growing**
  - vendor support, many device families, and data-center-grade acceleration cards like Alveo
- **FPGAs are not meant to replace GPUs**
  - but highly competitive for inference where energy efficiency, determinism, and low latency are crucial aspects

Thanks