FPGA Architecture for Deep Learning

FCCM’23 Tutorial

Organizers

Vaughn Betz
Prof. of ECE
U Toronto

Andrew Boutros
PhD Student @ U Toronto
ML Systems Architect @ MangoBoost

Contributors

Aman Arora
PhD Student
U Texas Austin

Sayedramin Rasoulinezhad
PhD Student
U Sydney

Lizy K. John
Prof. of ECE
U Texas Austin

Philip Leong
Prof. of Computer Systems
U Sydney

Raise your hand if you have used an FPGA for accelerating a deep learning (DL) workload

Raise your hand if you have used a new DL-optimized FPGA

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

Keep it interactive ✋

Come say hi during breaks 🤝

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

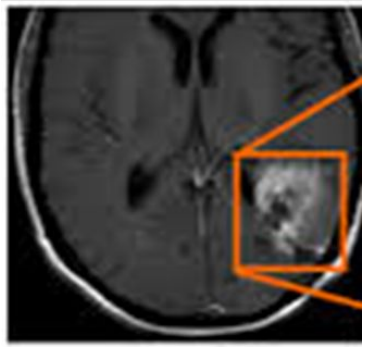
FPGA Architecture Through a DL Lens

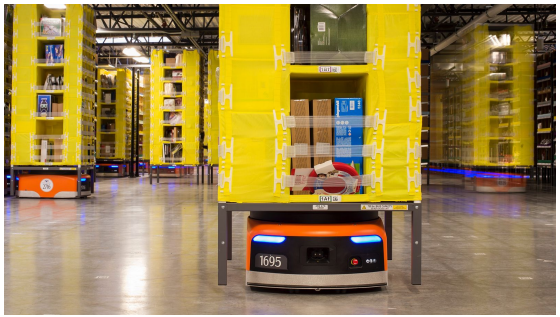
FPGA Architecture Through a DL Lens

FPGA Architecture, DL Implications and Opportunities

(Vaughn)

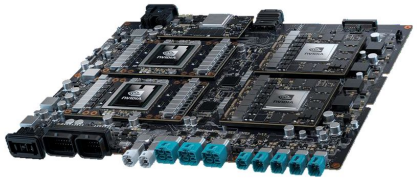
Deep Learning Inference → Becoming Ubiquitous





Energy Efficiency and Latency Matter

Chevy Bolt
~6 kW in city



Nvidia Drive AGX
Pegasus (2022)
750W

Low inference latency
crucial for safety!

Key metric is
Perf / W / $

Power is **~30%** of cost

~6% of global electricity demand by 2030 [1]

Low inference latency enables cascade of AI algorithms + networking!

[1] N. Jones. "How to Stop Data Centres from Gobbling up the World's Electricity". In: Nature 2018

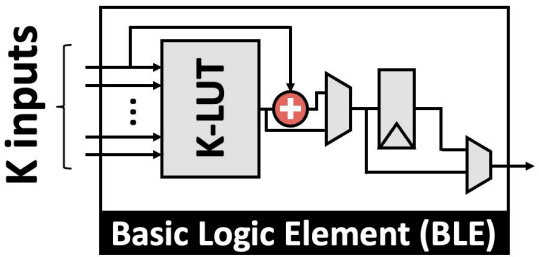
FPGA Architecture (through a DL Lens) from 300 m

1. What are the key building blocks of FPGAs?

2. How do they create strengths & challenges for Deep Learning?

3. Opportunities to create DL-optimized architectures

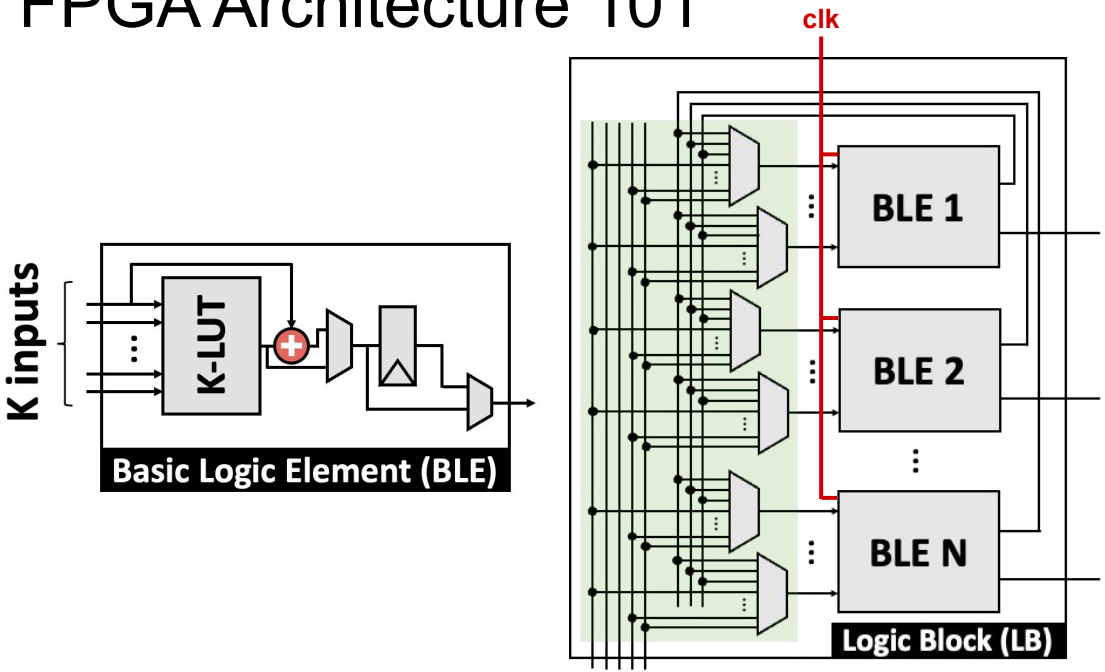
FPGA Architecture 101



Any function of K or fewer inputs, or a 1 bit adder

A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

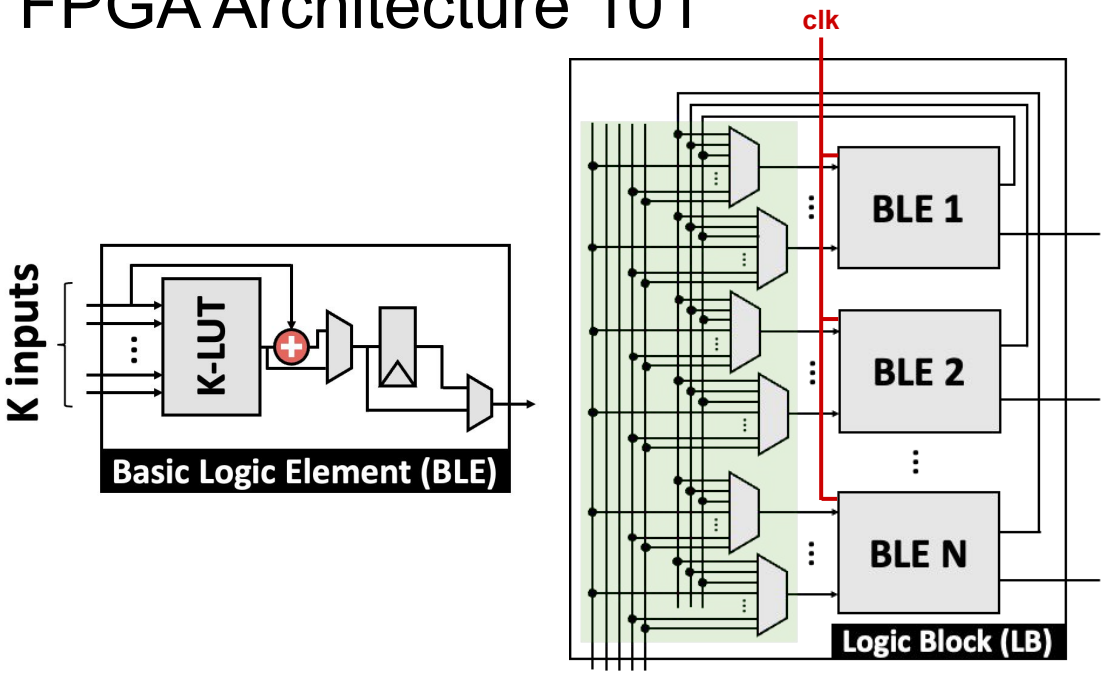
FPGA Architecture 101



Group BLEs into larger logic blocks with local interconnect

A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

FPGA Architecture 101

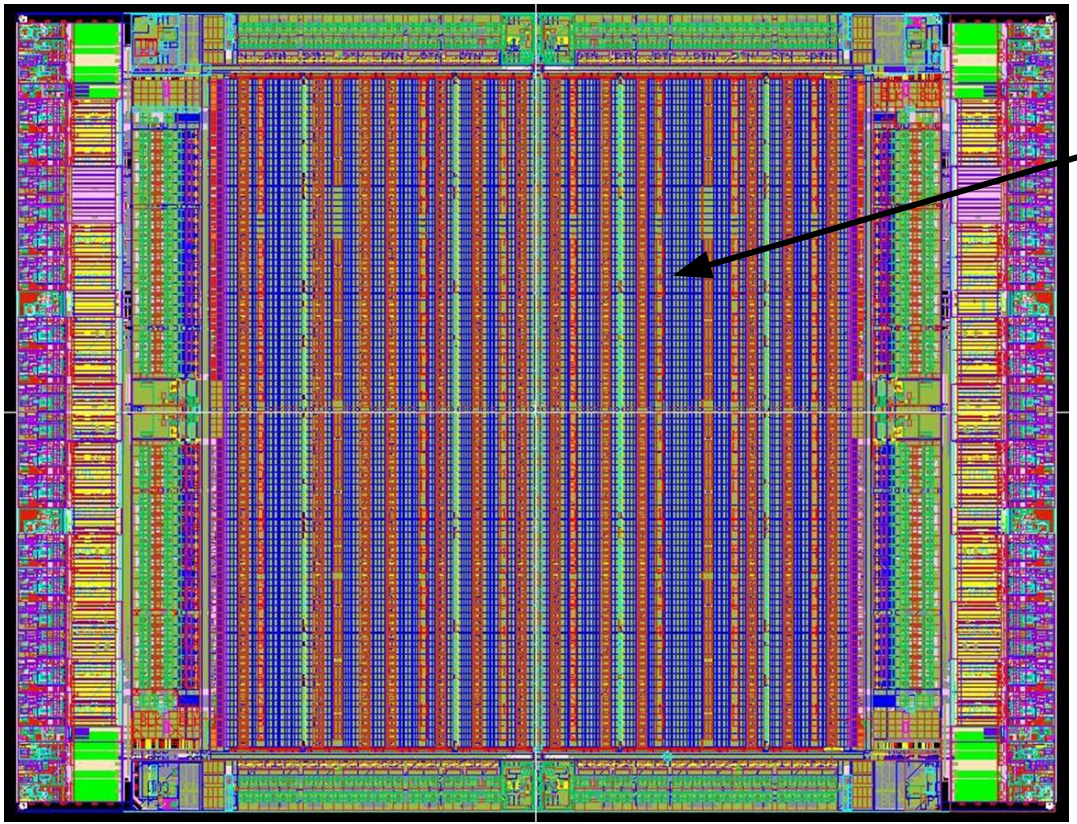


Create array of logic blocks surrounded by pre-fabricated wire segments and SRAM-controlled routing muxes

A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

Many Logic Blocks … Spatial Computing

Layout plot: Altera Stratix IV GX 230

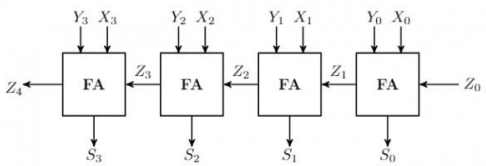


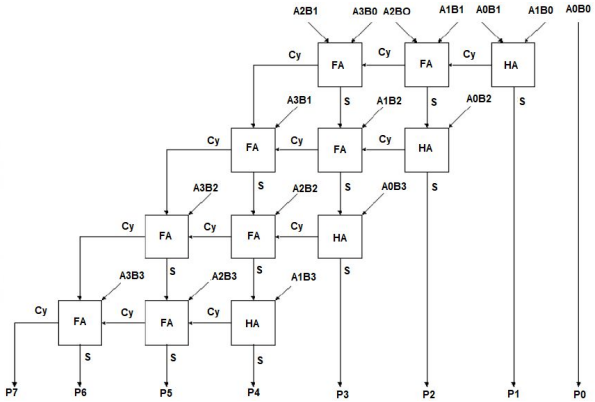
Blue: Logic Blocks (and Programmable Routing)

Prog. Logic & Routing **Strength** 1: Variable Precision

Can program to realize hardware of *any bit width*

- N-bit adder: ~N LEs
- N-bit multiplier: ~N^2 LEs

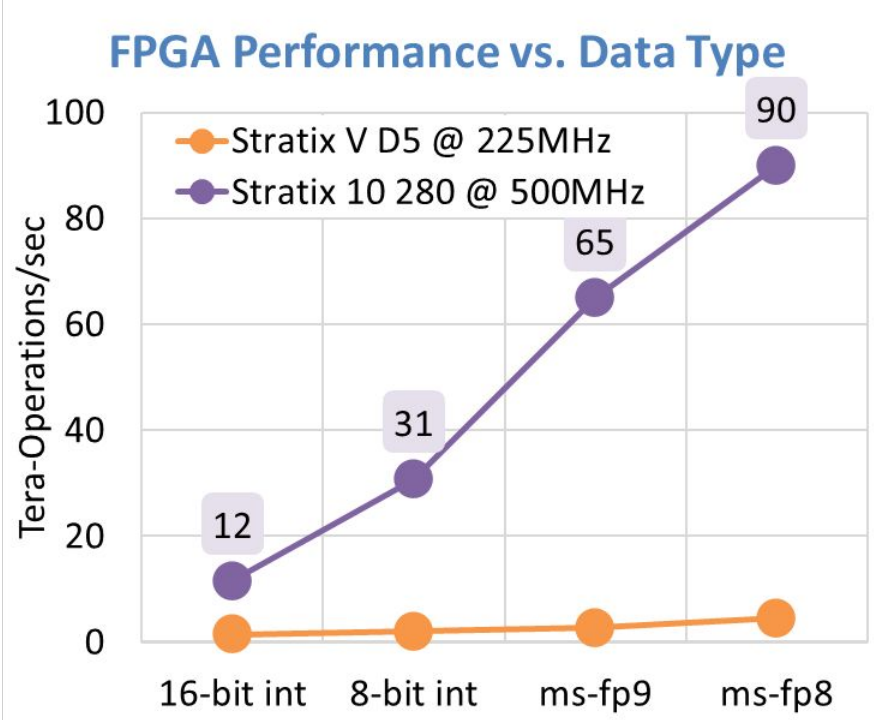


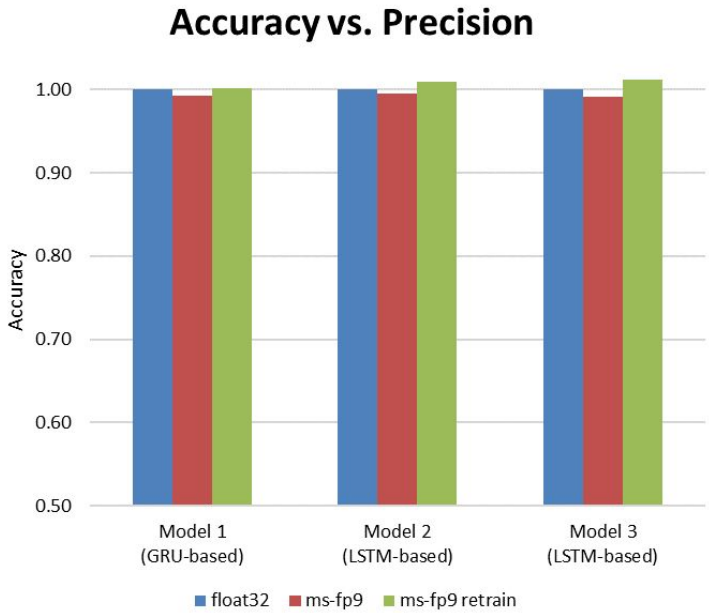


- DL tolerant of low precision
- No one best precision for all networks and all layers
- Use lowest precision that meets accuracy needs for each network / layer

→ No need to pick from a limited group of precisions or numeric formats

Leveraging Variable Precision: Microsoft Brainwave





Small, custom floating point: 7x performance

No accuracy loss at (retrained) custom 9-bit floating point

Figures from [E. Chung et al, "Accelerating Persistent Neural Networks at Datacenter Scale," Hot Chips 2017]

Prog. Logic & Routing **Strength** 2: Spatial Compute Energy

Can reprogram FPGA to implement exact hardware needed by network

- Programmable routing: directly wire data from one unit to another
- Programmable logic: perform only necessary operation, w/o instruction stream
  → Large power / efficiency gains possible

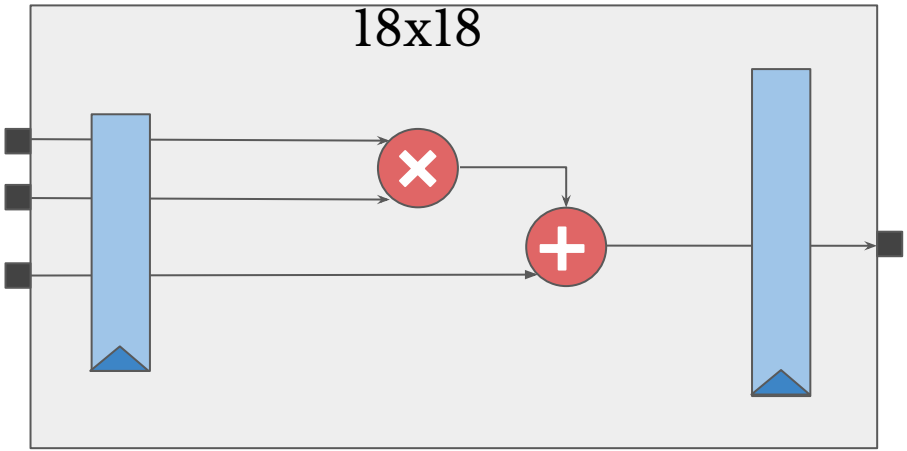
45 nm CPU energy breakdown [from M. Horowitz, "Computing's Energy Problem," *ISSCC,* 2014].

Prog. Logic & Routing **Weakness**: Area & Delay Overhead

- Programmability not free!
- LEs and programmable routing larger & slower than gates & wires
  - Average: ~25 - 30x larger and ~3x slower!
- How to mitigate?
  - Implement common functions in **hardened** blocks
  - ***Less programmable*** but built with gates (like an ASIC)
  - Example: DSP blocks for larger multiply-accumulate
- **Opportunity 1**
  - Can we make LEs themselves more efficient for DL operations?

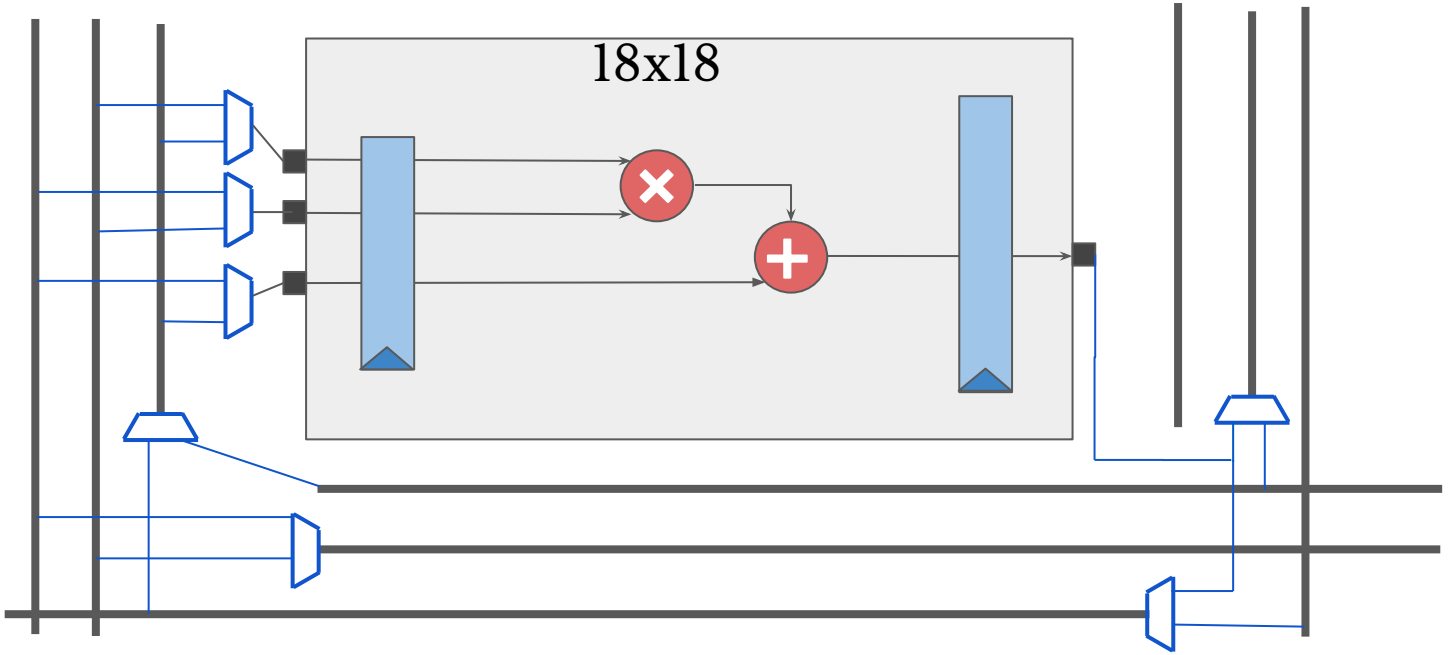
A. Boutros, S. Yazdanshenas and V. Betz, "You Can't Improve What You Don't Measure: FPGA vs. ASIC Efficiency Gaps for Convolutional Neural Network Inference," ACM TRETS, Dec. 2018, pp. 20:1 – 20:23.

Hard Block Example: DSP Blocks



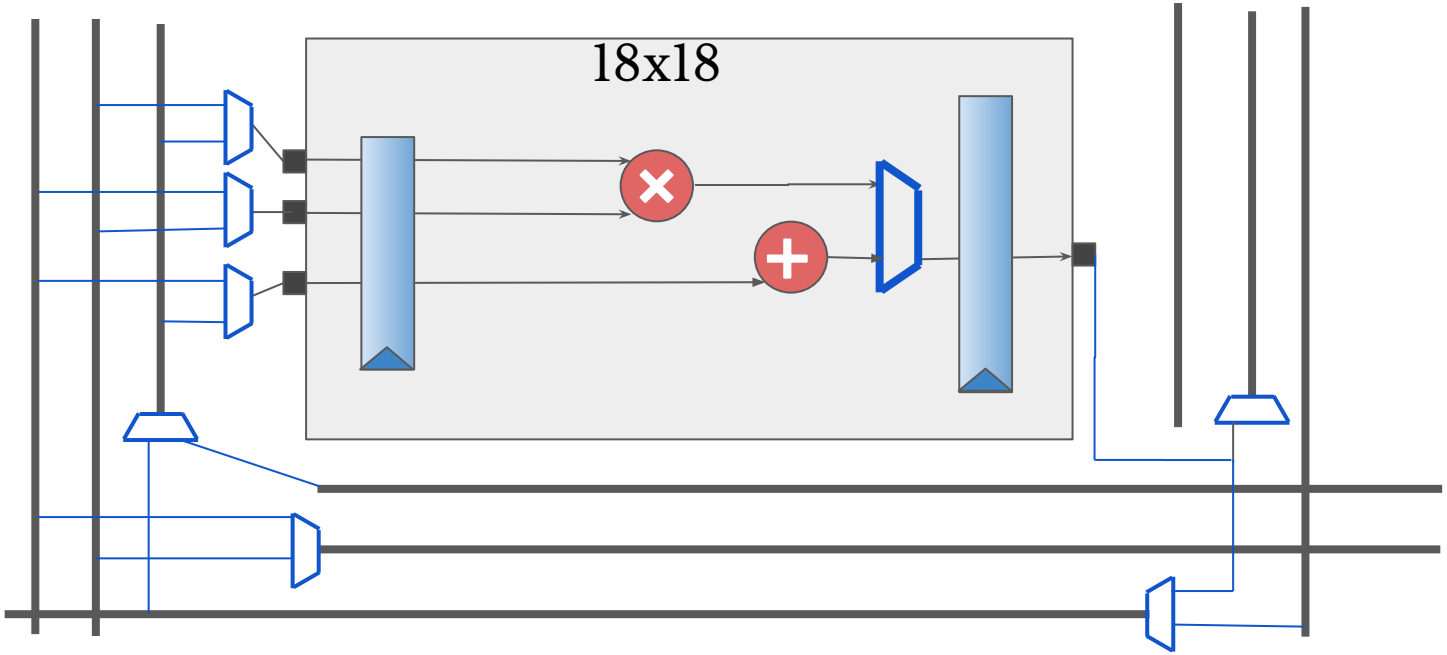
25x more dense & 3x faster?

Hard Blocks: Programmable Routing Impact



- New block → Needs muxes to/from programmable routing wires
- Column of blocks → Another channel of programmable routing
- Programmable Routing Area ∝ $\text{Block}_{\text{inputs}}$ + $\text{Block}_{\text{outputs}}$

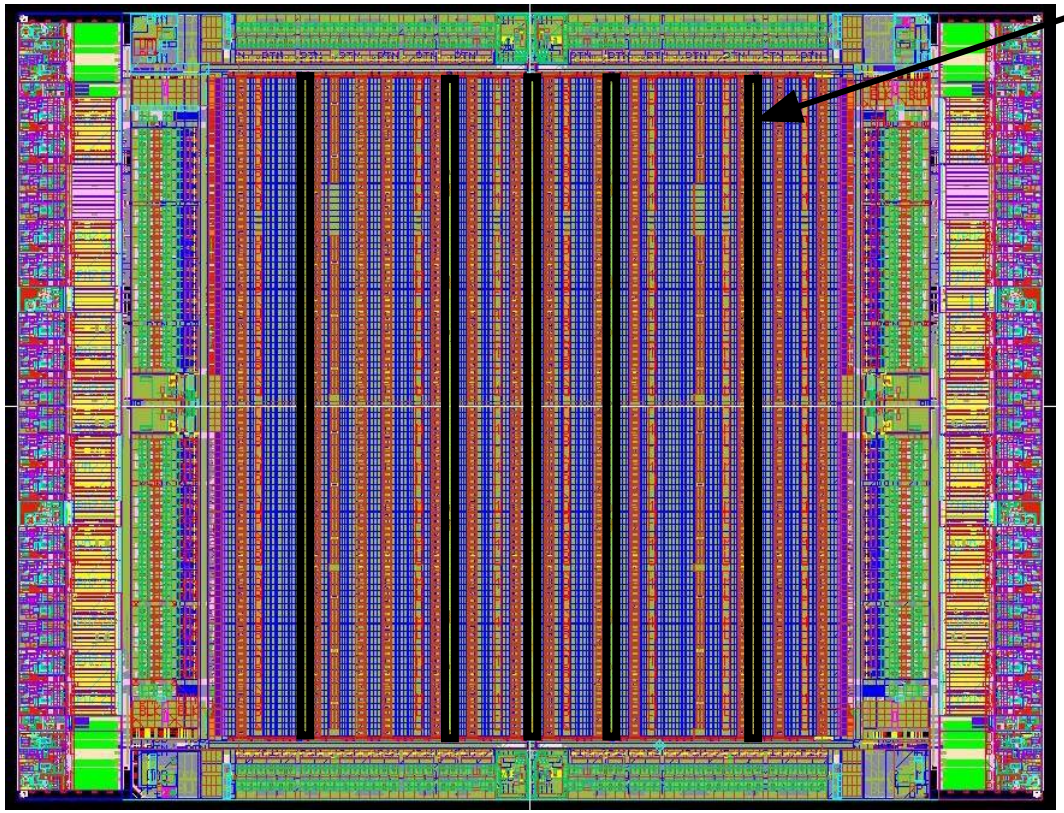
Hard Blocks: Add Low-Cost Programmability



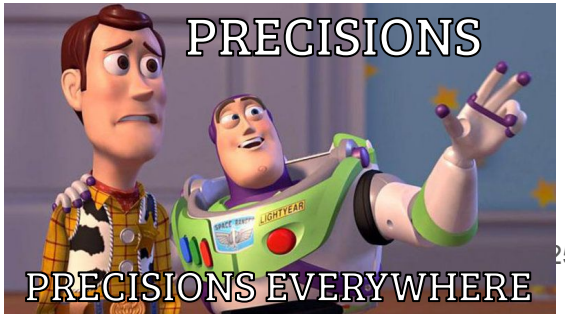
- MAC → multiply or MAC
- Register inputs & outputs → Optional registering
- *18-bit FIR filters*: 6X - 8X density & 2X - 2.5X speed vs. LE-implementation

A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

Strength 3: Massive Parallelism

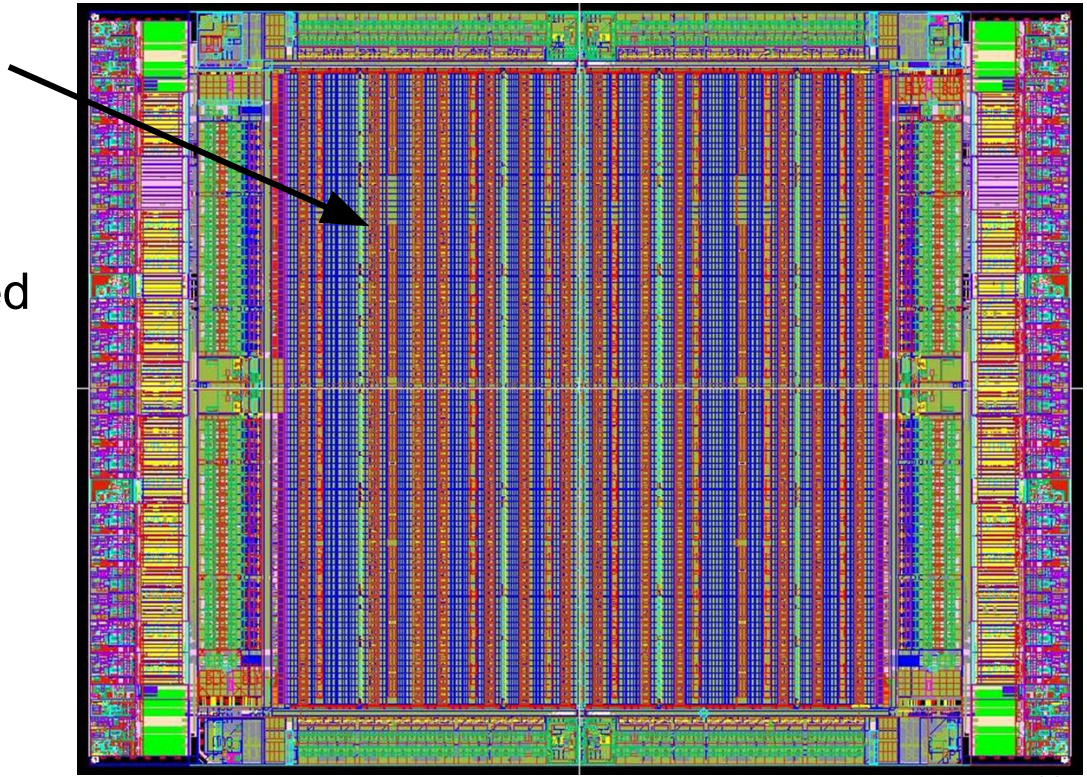


- Tens of thousands of multipliers in recent devices
- Designed for (mostly wireless) signal processing
- **Opportunity 2**: New hard blocks?



Block RAM

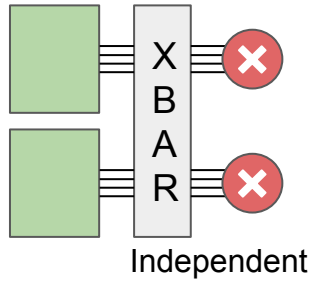
- Thousands of independent RAM blocks, spatially distributed
- Best size?
  - Trade-off: larger blocks lead to lower area/bit
  - Smaller blocks let you fit more RAM blocks & bandwidth in chip
  - ~20 kb / block a common choice

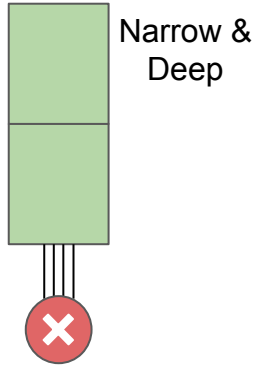


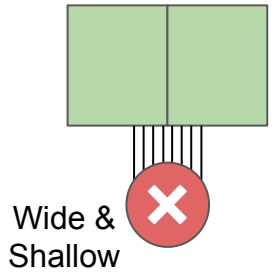
Strength 4: Flexible Memory → Low Latency

Huge flexibility in combining RAMs with programmable logic & routing

- Different for each layer
- Custom scatter/gather can exploit sparsity







Massive bandwidth

- ~Pb/s of on-chip bandwidth, *split into 10,000+ components*

Can keep compute units fed with little or no batching if most/all data on chip

- GPUs batch multiple inputs to amortize weight re-loading → latency increase

Challenge: very large networks need off-chip memory → weakens advantage

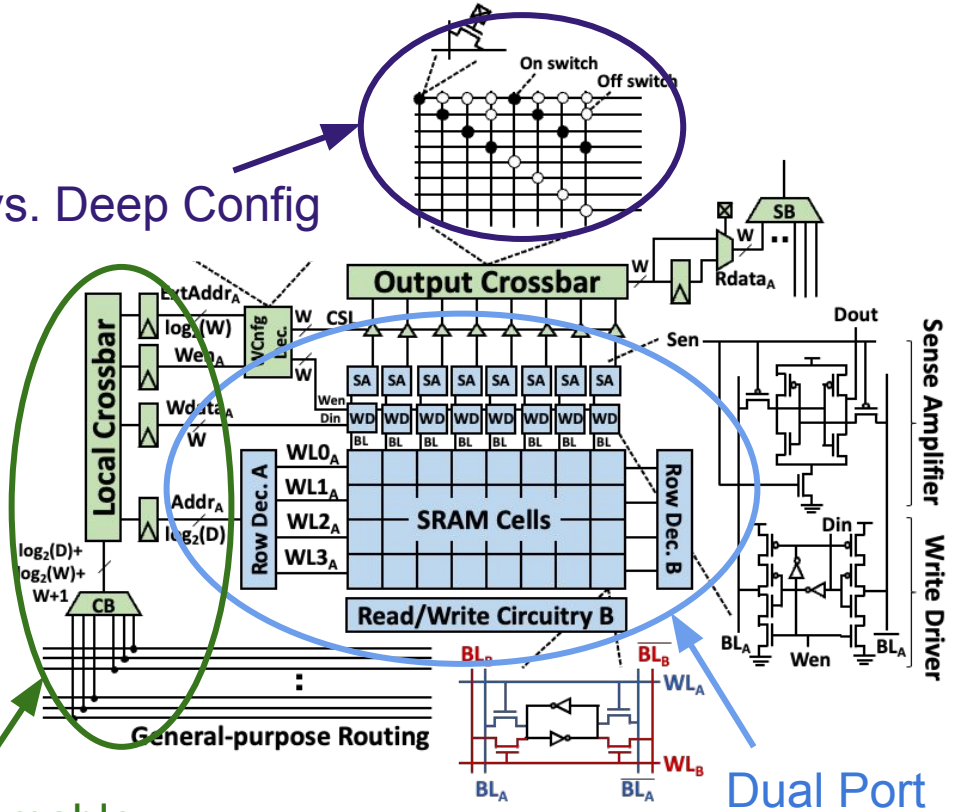
Block RAM: Under the Hood

Unique feature: RAM configurable width/depth

- Increases flexibility
- E.g. *18k words x 1b* or *512 words x 36b*

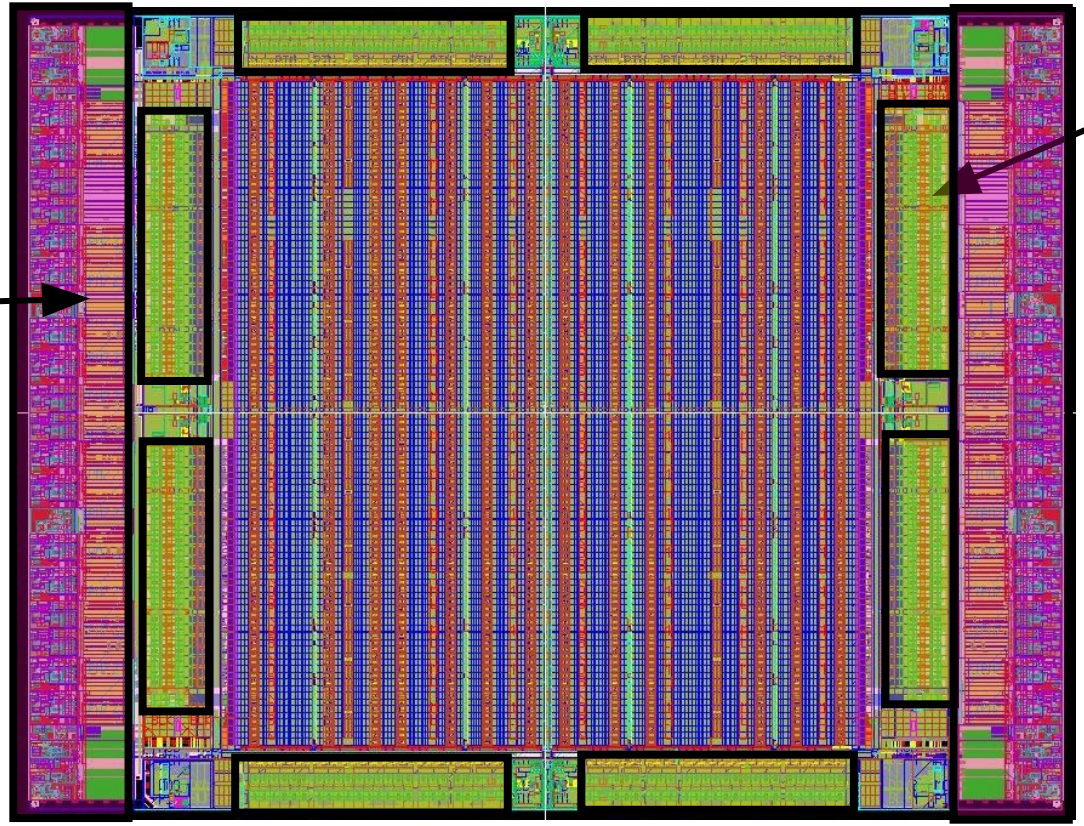
Opportunity 3:

- Tens of thousands of RAMs
- Can connect to anything
- Can we add in-memory processing cheaply?



A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

Don’t Forget I/O !



Programmable Serial I/O (PCIe,...)

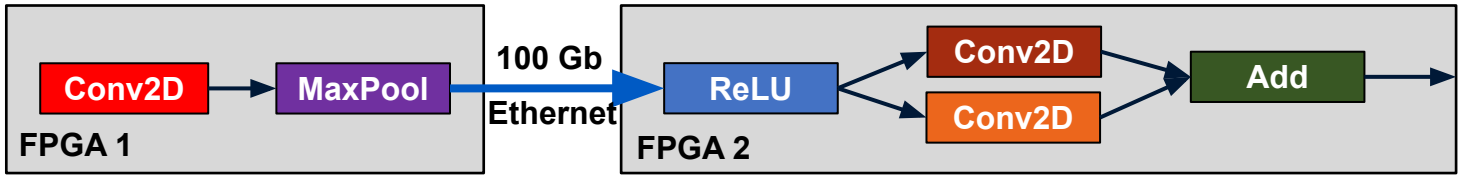
Programmable I/O (DDR, …)

Strength 5: Low Latency, Highly Flexible I/O

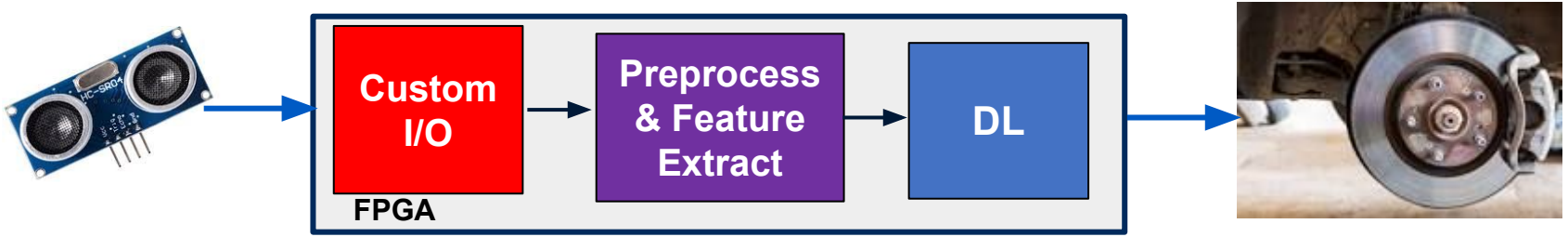
HW accelerated, low latency

Myriad I/O options: DDR5, PCIe, Ethernet, custom standards, …

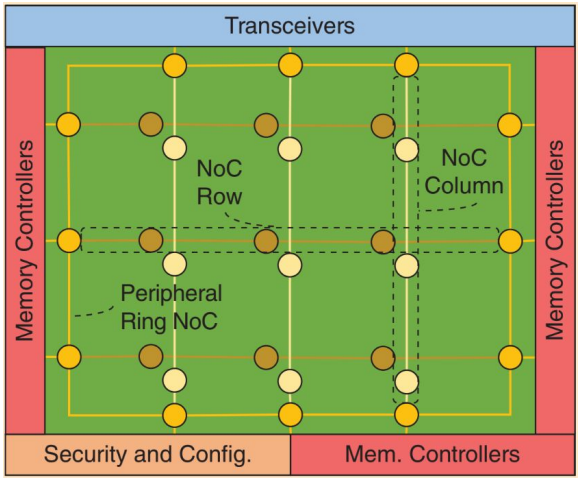
Datacenter: scale in space



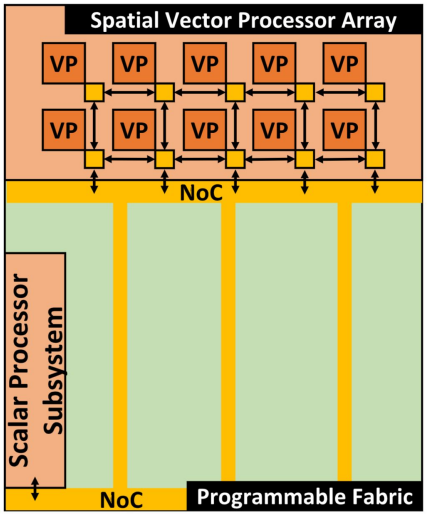
Embedded: low latency & custom I/O



Recent Developments: NoCs, Embedded Accelerators



Achronix Speedster 7t



Xilinx Versal

Efficient system-level interconnect for (high bandwidth) I/O to prog. fabric

Opportunity 4: easier to integrate coarse-grained / novel accelerators

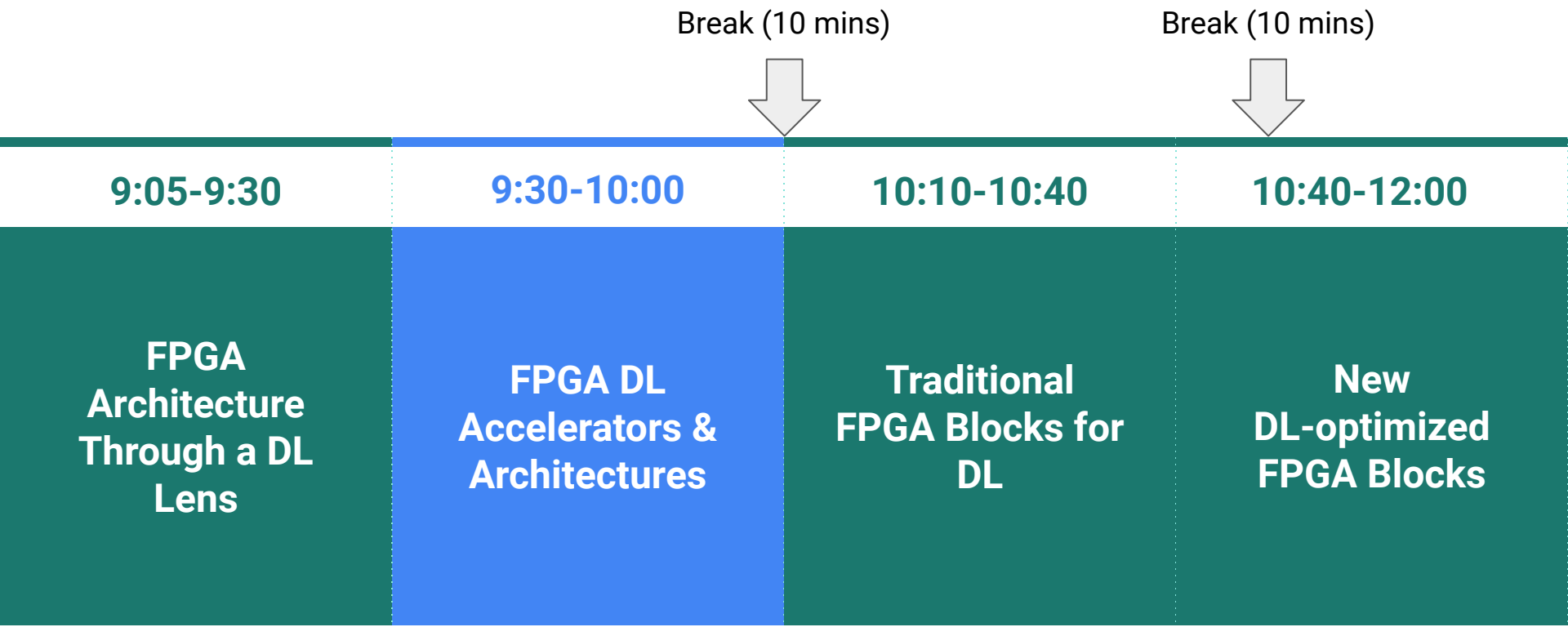
FPGA Architecture Through a DL Lens

Strengths & Weaknesses vs. DL Application Attributes
(Vaughn)

Deep Learning Attributes and FPGAs

Characteristic		
Precision	Low	High
Sparse Weights?	Yes: Efficient with custom memory & hw scatter-gather	No: still efficient, but less opportunity for customization
Latency Constraint	Tight	Loose: batching can help GPU efficiency
Network Size	Moderate: on-chip memory stores much of network	Very large: Off-chip memory interfaces most important
Phase	Inference	Training
Network Changes	Rare	Frequent: Some accelerator styles will reduce developer productivity

Schedule

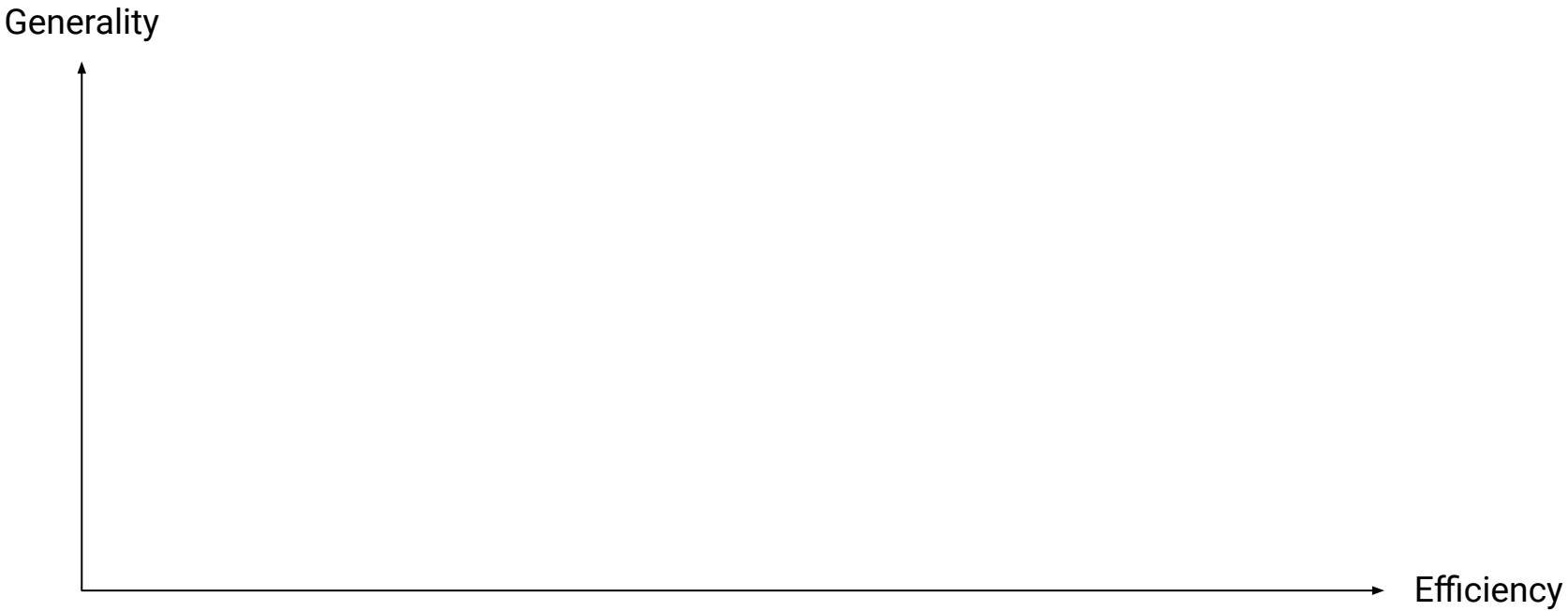

Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

FPGA DL Accelerators and Architectures

FPGA DL Accelerators and Architectures

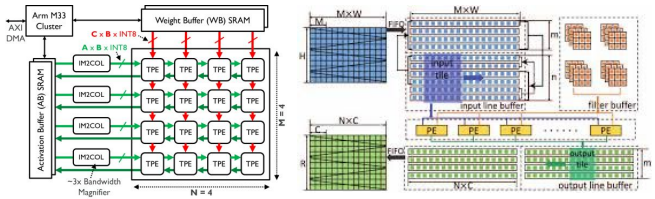
Styles of Accelerating DL using FPGAs
(Andrew)

DL Acceleration Styles on FPGAs



DL Acceleration Styles on FPGAs

Generality

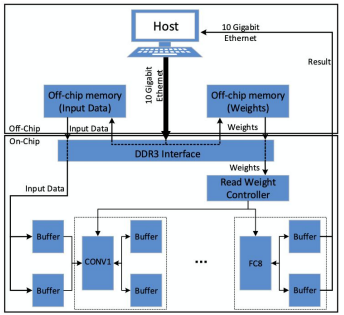


Offload specific operations from host to FPGA (e.g. GEMM, Convolution)

Layer/Operation Engines

Efficiency

DL Acceleration Styles on FPGAs



Accelerate a specific NN completely on FPGA

Model-Specific Accelerators

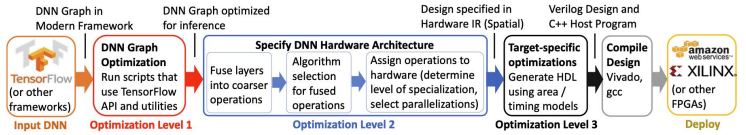
Generality

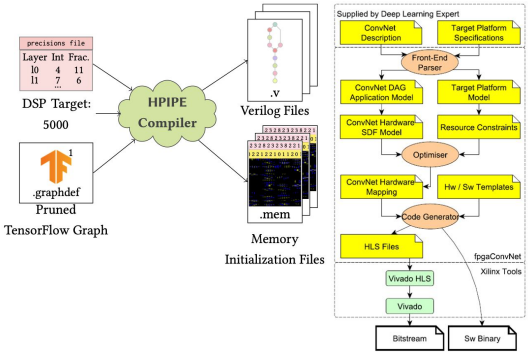
Layer/Operation Engines

Efficiency

H. Li et al, “A High Performance FPGA-based Accelerator for Large-Scale Convolutional Neural Networks“, FPL 2016

DL Acceleration Styles on FPGAs





Domain-specific compilers
DL Framework → FPGAs

Custom HW Generators

Model-Specific Accelerators

Generality

Layer/Operation Engines

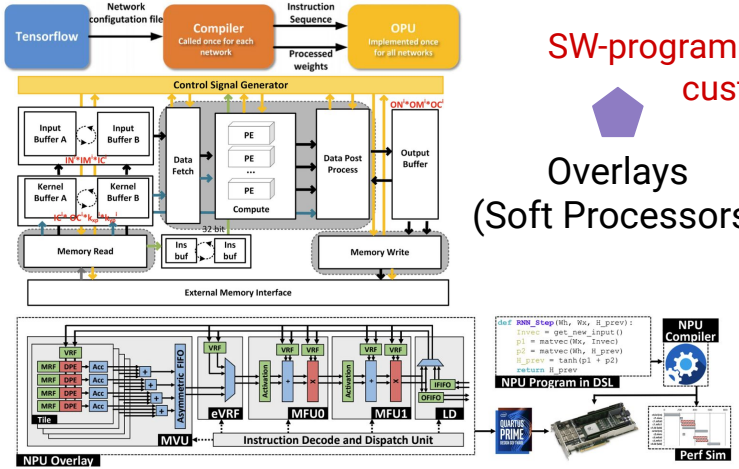
Efficiency

X. Zhang et al, “DNNBuilder: An automated tool for building high-performance DNN hardware accelerators for FPGAs”, ICCAD, 2018

M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

S. Hadjis et al, “TensorFlow to Cloud FPGAs: Tradeoffs for Accelerating Deep Neural Networks“, FPL 2019

DL Acceleration Styles on FPGAs



SW-programmable processors with custom ISA & HW

Overlays (Soft Processors)

Custom HW Generators

Model-Specific Accelerators

Layer/Operation Engines

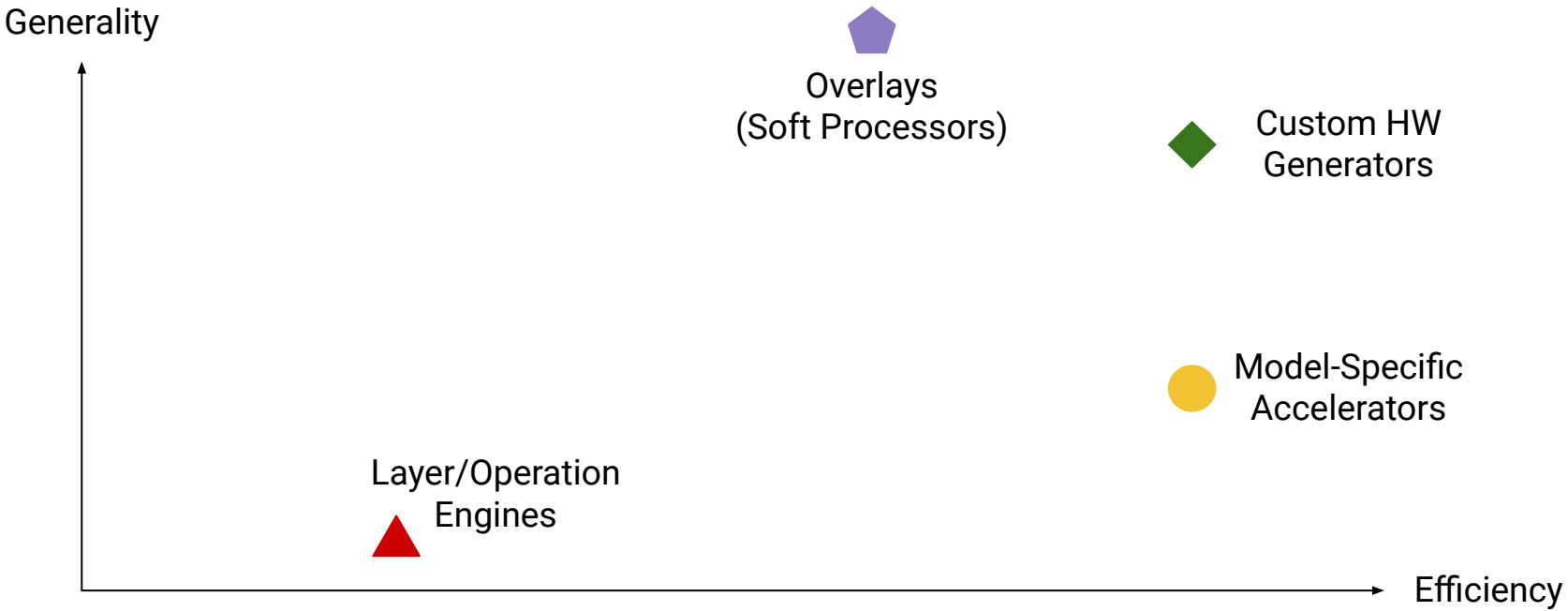
Generality

Efficiency

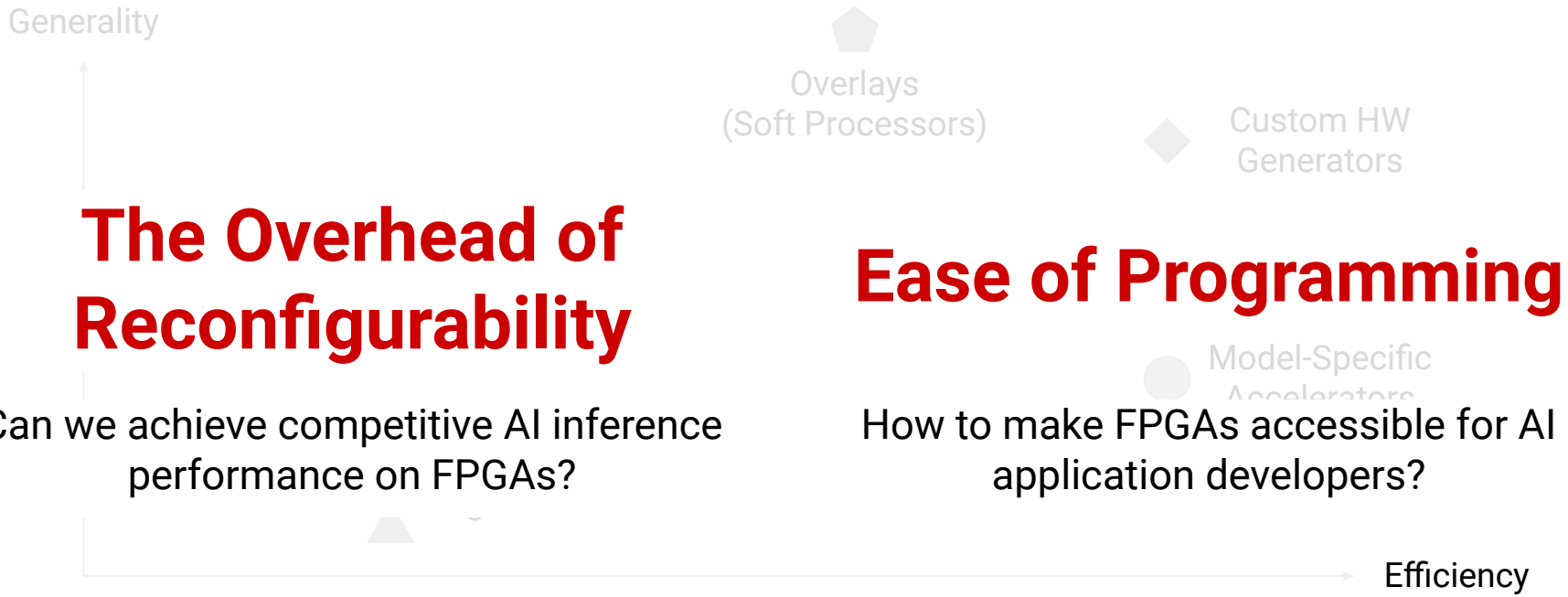
A. Boutros et al, “Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs“, FPT 2020

Y. Yu et al, “OPU: An FPGA-Based Overlay Processor for Convolutional Neural Networks“, TVLSI 2020

Two Key Challenges for Accelerating DL using FPGAs …



Two Key Challenges for Accelerating DL using FPGAs …



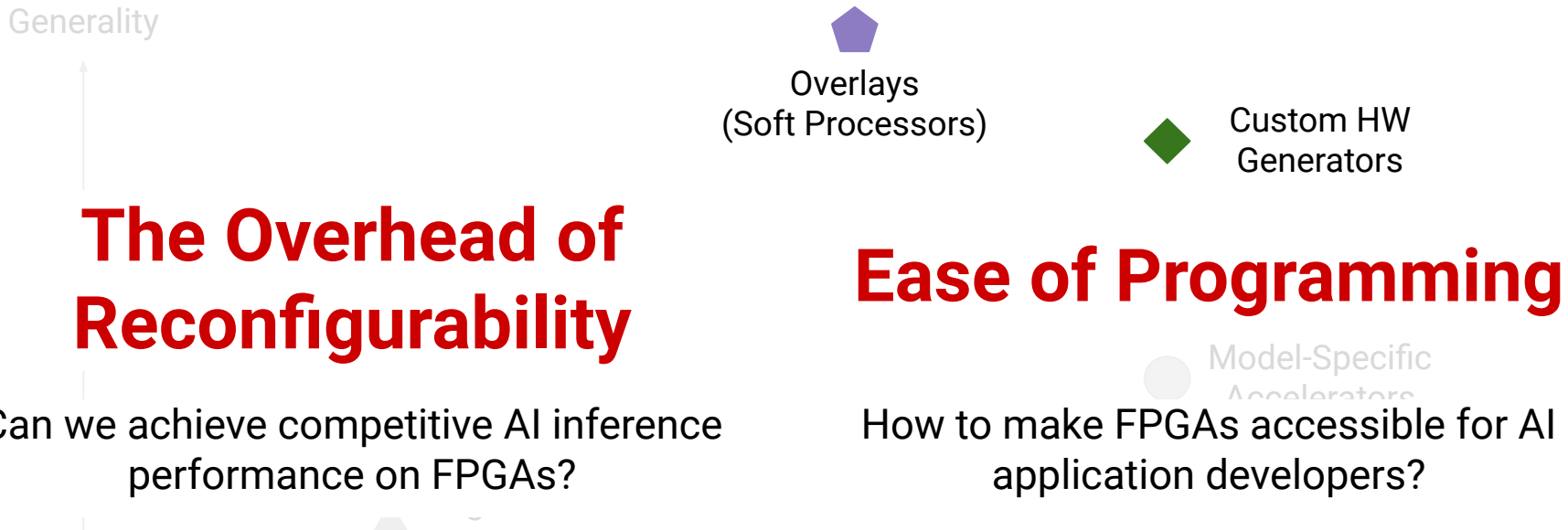
The Overhead of Reconfigurability

Can we achieve competitive AI inference performance on FPGAs?

Ease of Programming

How to make FPGAs accessible for AI application developers?

Two Key Challenges for Accelerating DL using FPGAs …



The Overhead of Reconfigurability

Can we achieve competitive AI inference performance on FPGAs?

Ease of Programming

How to make FPGAs accessible for AI application developers?

Will show two examples from these two design styles & how they can address these concerns

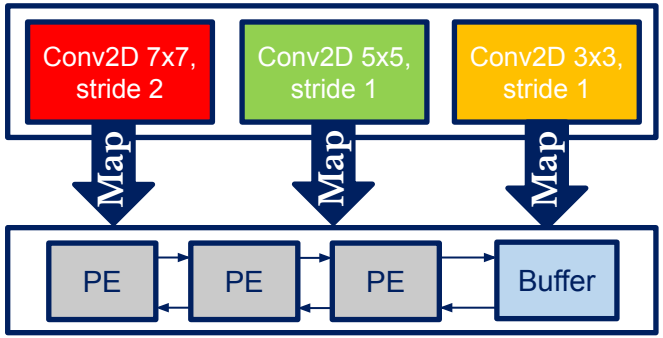
Design Approach 1

Custom HW Generators (HPIPE)

Design Philosophy

Commonly → Temporal mapping on PE arrays

- Sequential processing of layers
- PEs handle any layer → **lower efficiency**

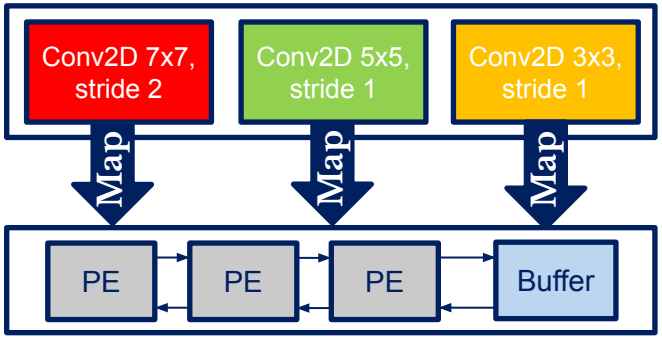


M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

Design Philosophy

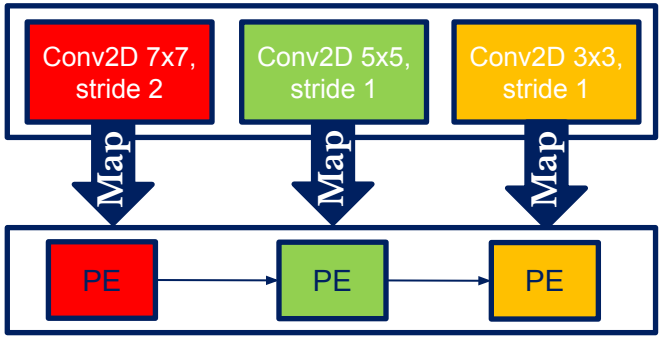
Commonly → Temporal mapping on PE arrays

- Sequential processing of layers
- PEs handle any layer → **lower efficiency**



Instead → Spatial mapping to specialized units

- Per-layer custom HW → **higher efficiency**
- Exploit pipeline parallelism



M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

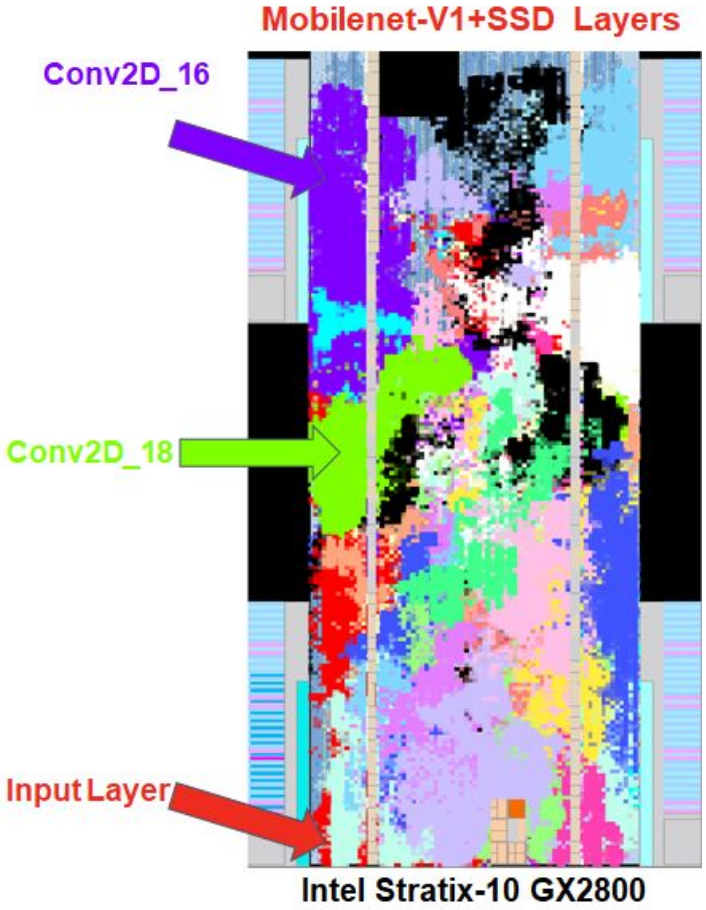
Design Philosophy

Commonly → Temporal mapping on PE arrays

- Sequential processing of layers
- PEs handle any layer → **lower efficiency**

Instead → Spatial mapping to specialized units

- Per-layer custom HW → **higher efficiency**
- Exploit pipeline parallelism



M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

Design Philosophy

Commonly → Temporal mapping on PE arrays

- Sequential processing of layers
- PEs handle any layer → **lower efficiency**

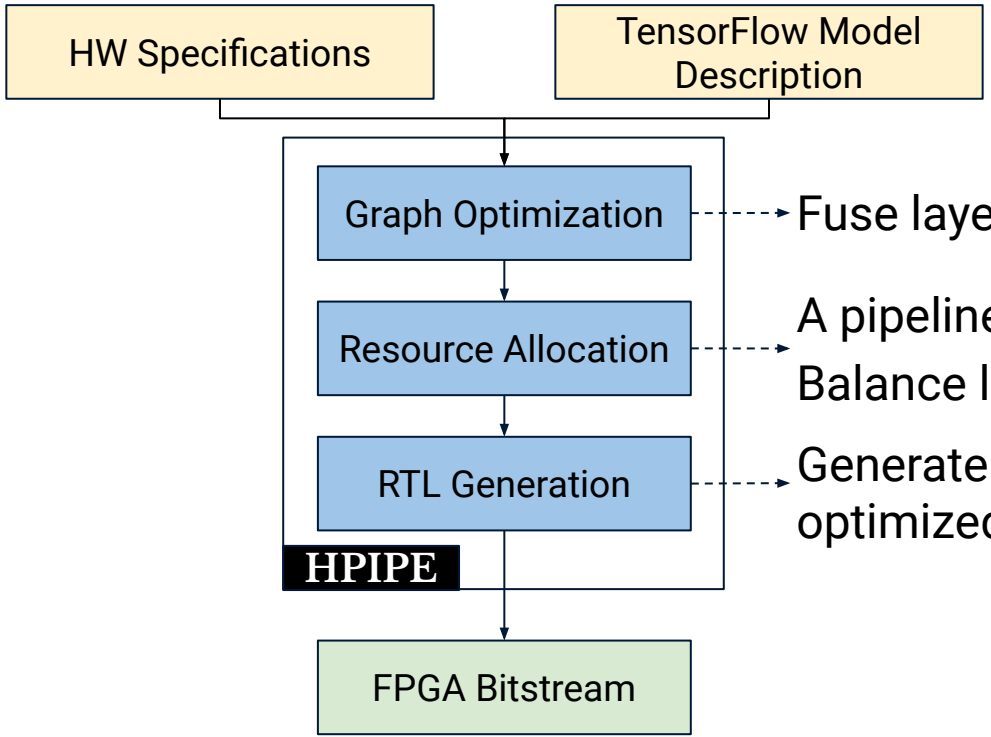
Instead → Spatial mapping to specialized units

- Per-layer custom HW → **higher efficiency**
- Exploit pipeline parallelism

Very efficient but … requires a new implementation for each model!

M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

Auto Generation of Custom CNN HW (HPIPE)



Fuse layers for more efficient implementations

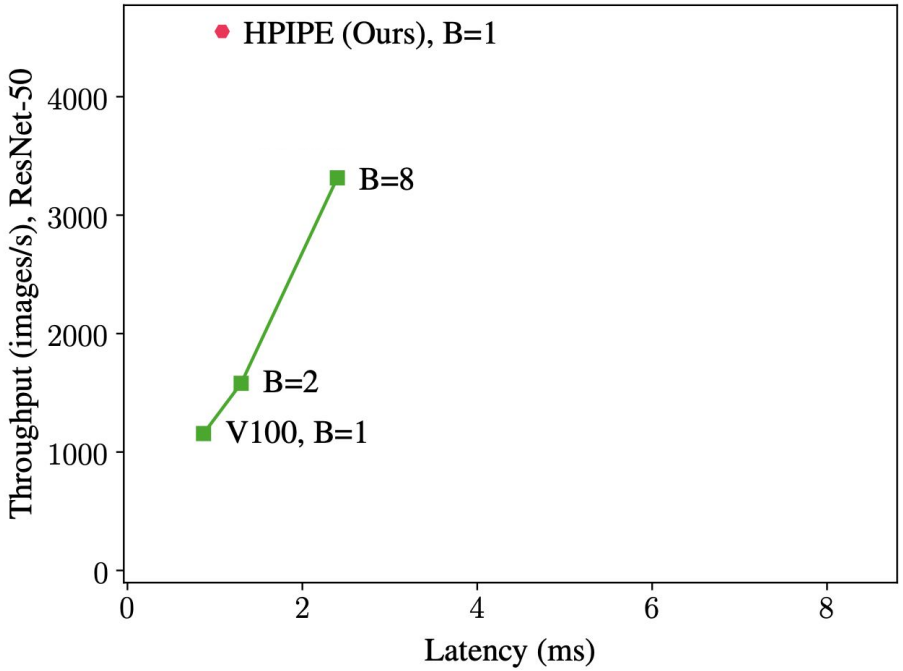
A pipeline is as slow as its slowest stage
Balance layer throughput to increase efficiency

Generate HW implementation from highly optimized layer templates

M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

ResNet-50 Results (Conventional Stratix 10 FPGA)

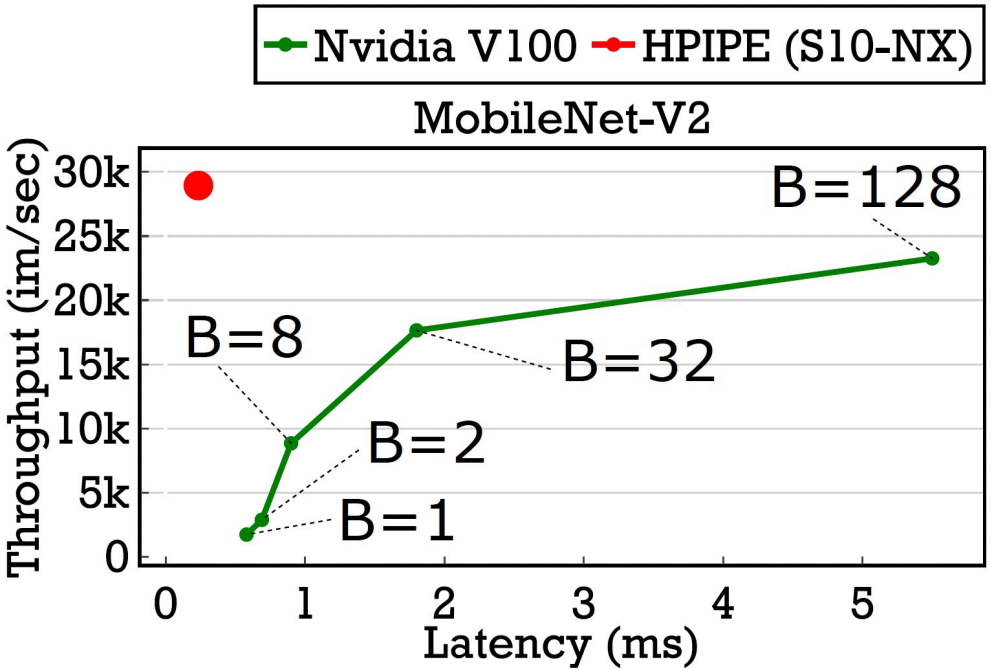
- **4x** higher batch-1 throughput vs. V100 GPU at similar (low) latency
- **1.4x** higher batch-8 throughput vs. V100 GPU at **2x** lower latency



M. Hall et al, “From TensorFlow Graphs to LUTs and Wires: Automated Sparse and Physically Aware CNN Hardware Generation“, FPT 2020

MobileNet-V2 Results (AI-Optimized Stratix 10 FPGA)

- **17x** higher batch-1 throughput vs. V100 GPU at lower latency
- **1.3x** higher batch-128 throughput vs. V100 GPU at **3x** lower latency

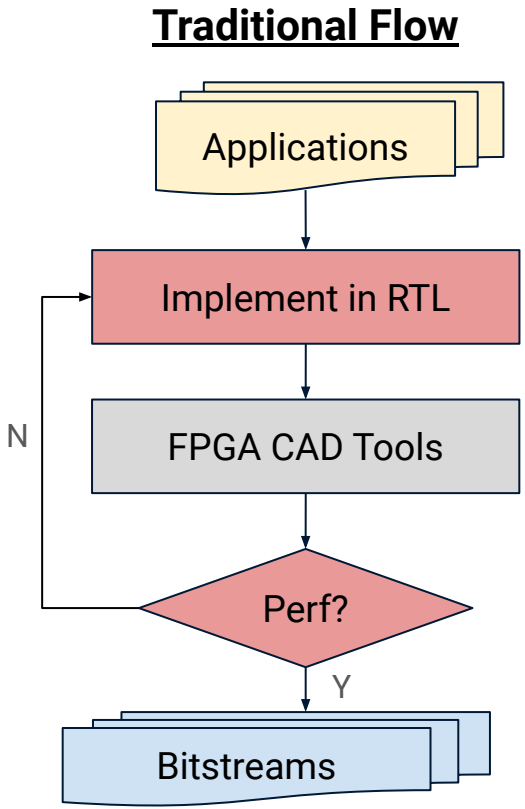


M. Stan et al, “HPIPE NX: Boosting CNN Inference Acceleration Performance with AI-Optimized FPGAs“, Under Review

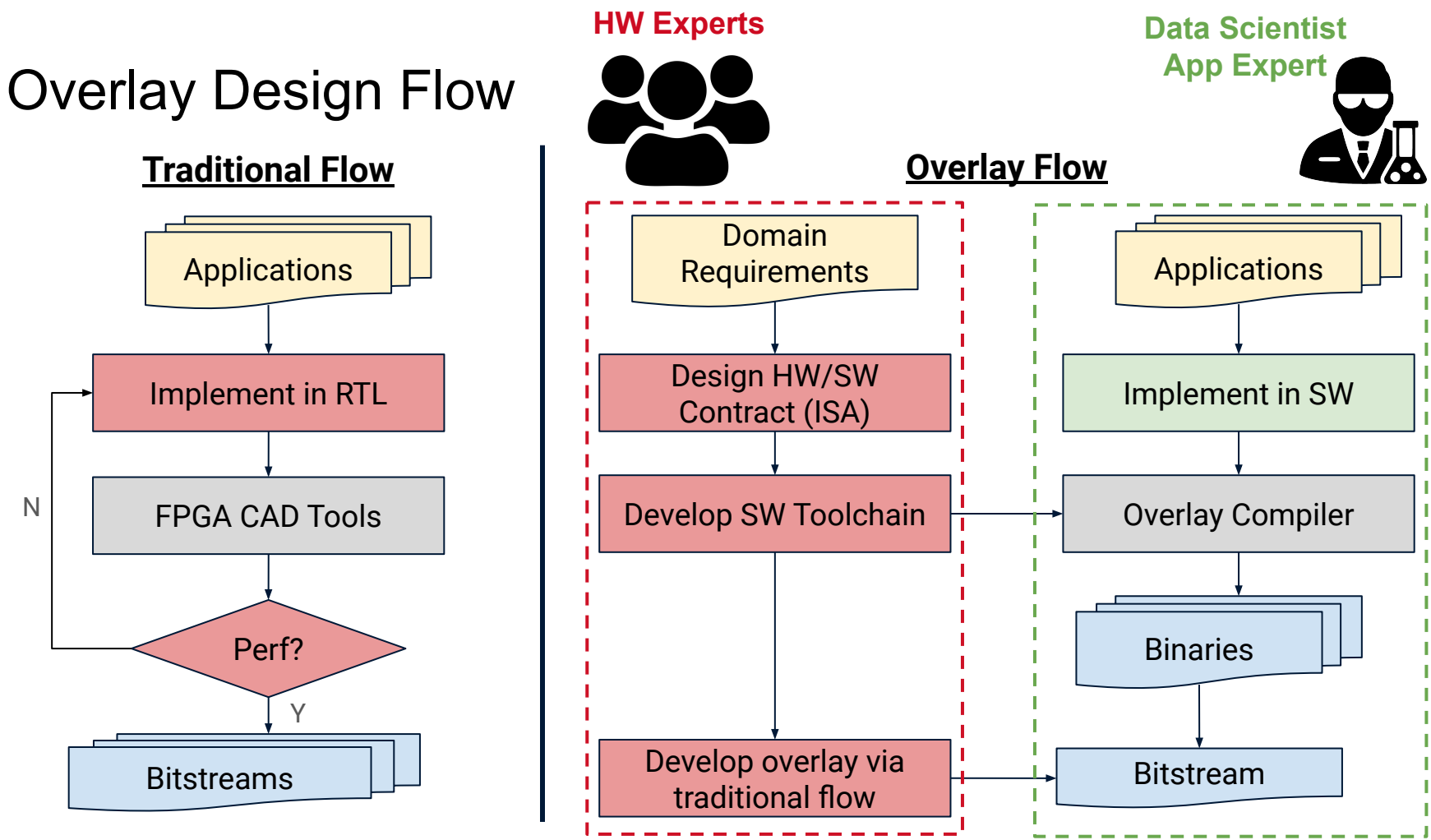
Design Approach 2

FPGA Overlays
(NPU)

Overlay Design Flow

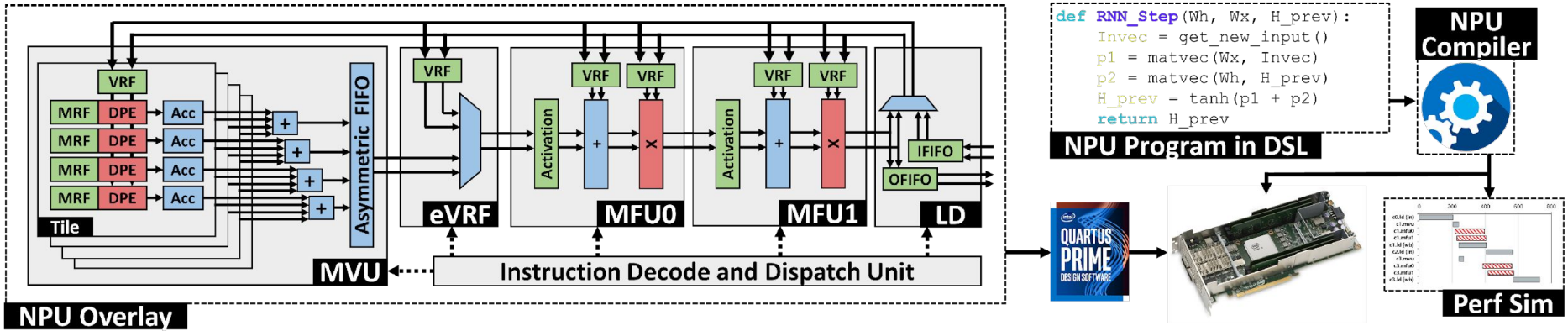


Overlay Design Flow



Neural Processing Unit (NPU)

- Very Long Instruction Word (VLIW) soft processor - 5 coarse grained stages
- Amortize control → Single instruction executes **45,000** operations
- Customize memory subsystem → Exploit tremendous on-chip memory BW
- Targeting memory-bound models (MLPs, RNNs, GRUs, LSTMs)

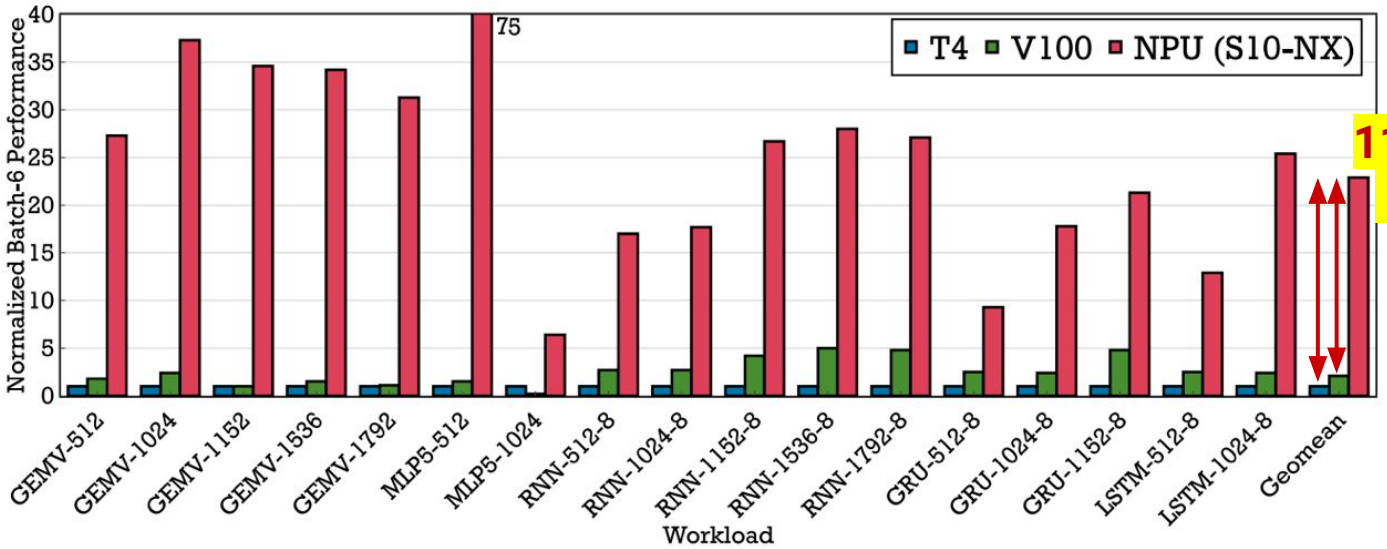


A. Boutros et al, “Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs“, FPT 2020

Results vs. Same-Generation DL-Optimized GPUs

	Nvidia V100	Nvidia T4	Intel S10 NX
On-chip Memory	16 MB	10 MB	16 MB
Process Tech.	TSMC 12nm	TSMC 12nm	Intel 14nm
Die Size (mm^2)	815	545	< 500

AI-Optimized FPGA with Tensor Blocks (More details later!)



11x vs. V100
23x vs. T4

A. Boutros et al, "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs", FPT 2020

Can current FPGAs achieve good DL inference performance?

Can current FPGAs achieve good DL inference performance?

YES!

Automatic custom HW generation → HPIPE

Software-programmable Overlays → NPU

Can current FPGAs achieve good DL inference performance?

YES!

Automatic custom HW generation → HPIPE

Software-programmable Overlays → NPU

Can we make current FPGAs easier to use for DL application developers?

Can current FPGAs achieve good DL inference performance?

YES!

Automatic custom HW generation → HPIPE

Software-programmable Overlays → NPU

Can we make current FPGAs easier to use for DL application developers?

YES!

Tensorflow to LUTs & wires → compile new bitstream for each model

Program purely in software → run instructions on a single optimized bitstream

Can current FPGAs achieve good DL inference performance?

YES!

Automatic custom HW generation → HPIPE

Software-programmable Overlays → NPU

Can we make current FPGAs easier to use for DL application developers?

YES!

Tensorflow to LUTs & wires → compile new bitstream for each model

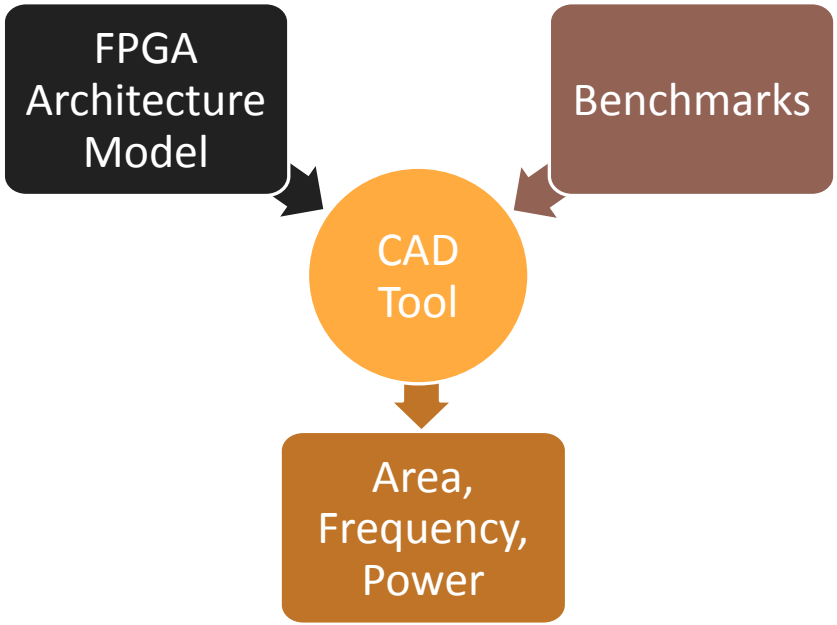
Program purely in software → run instructions on a single optimized bitstream

Both performance and ease-of-use can also be improved by enhancing underlying FPGA architecture for DL use cases …

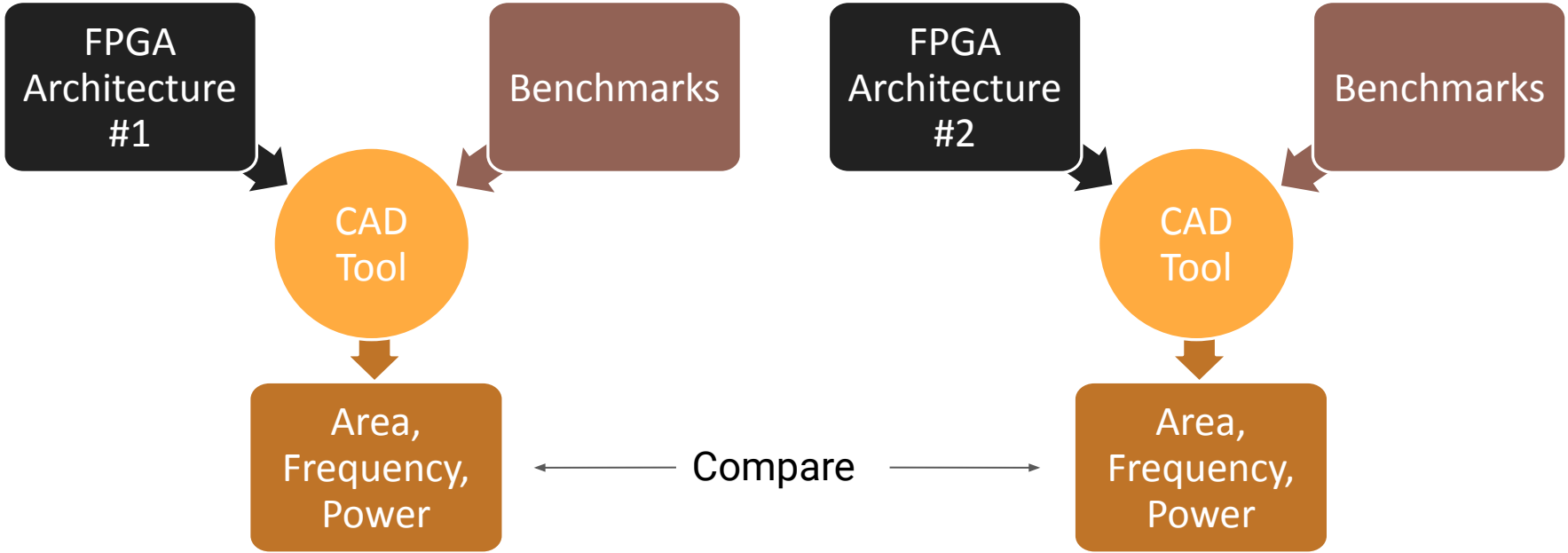
FPGA DL Accelerators and Architectures

Architecture Exploration of DL-Optimized FPGAs
(Vaughn)

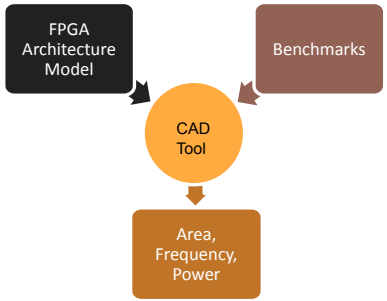
FPGA Architecture Research

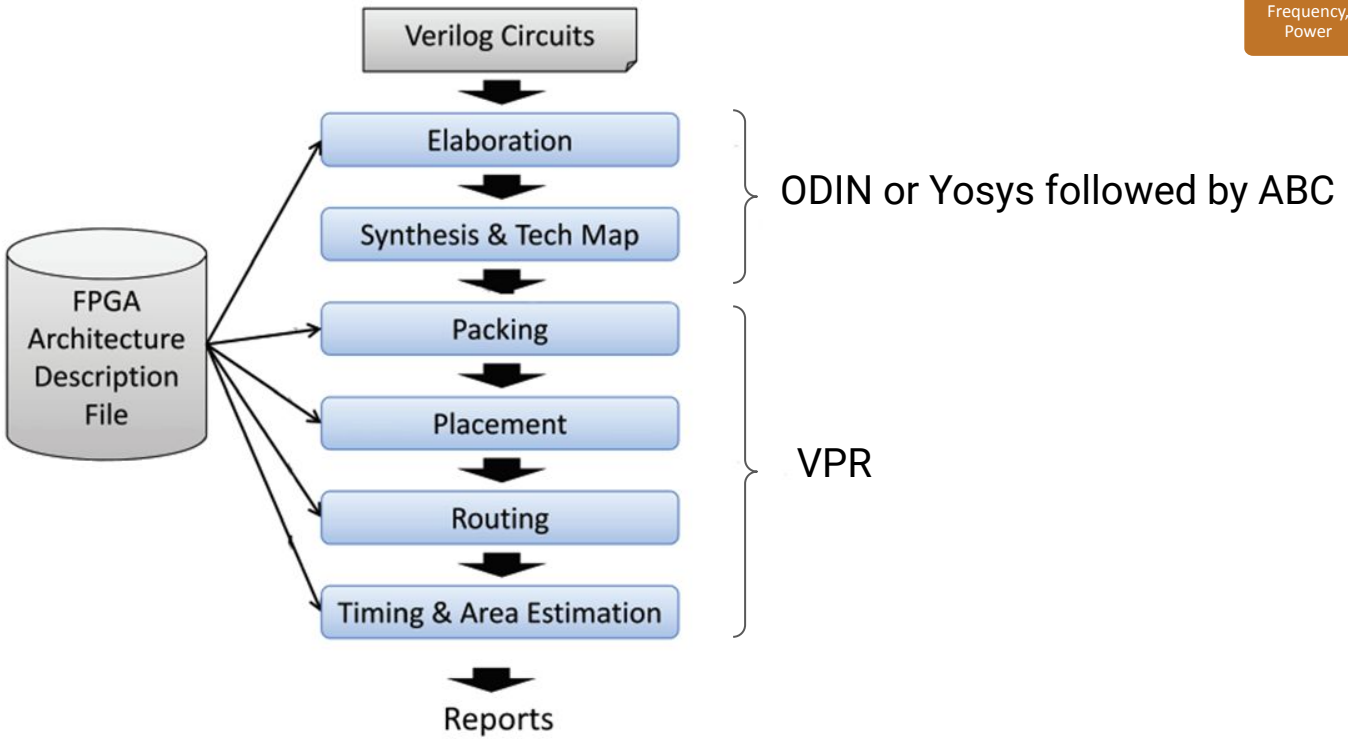


FPGA Architecture Research

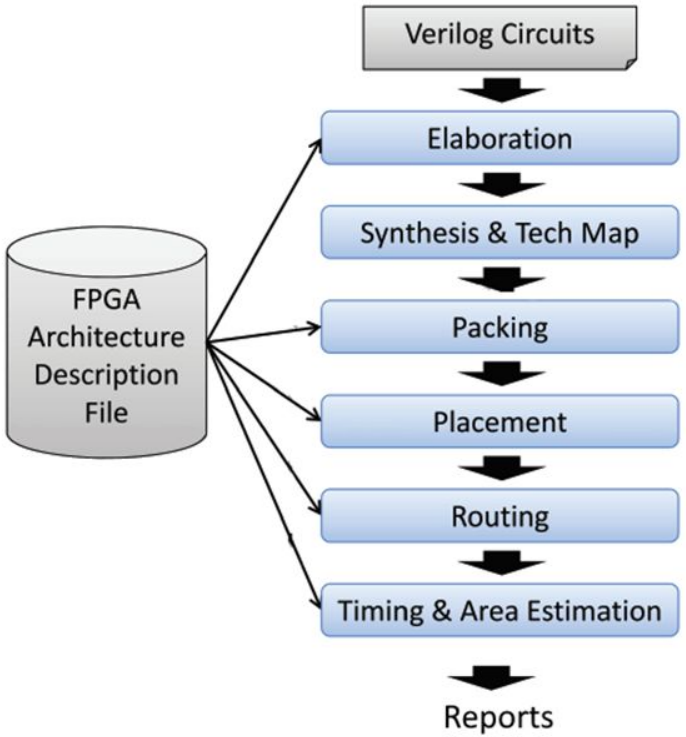


VTR (Verilog to Routing)





VTR (Verilog to Routing)



Think Xilinx/AMD Vivado or Altera/Intel Quartus

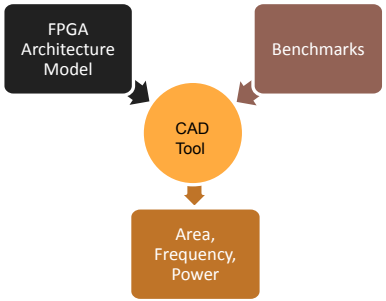
But…

For an FPGA described in the file

And

No bitstream generation

FPGA Architecture Model



Blocks

- Number and type of blocks
- Layout of blocks on the FPGA

Routing

- Distribution of wire segments
- Types of switches

Configuration circuitry

- If you need to experiment with it

FPGA Architecture Model in VTR

```
<tiles>
  <tile name="io" capacity="8" area="0">
  </tile>
  <tile name="clb" area="53894">
  </tile>
  <tile name="memory" height="2" area="548000">
  </tile>
  <tile name="mult_36" height="6" area="396000">
  </tile>
</tiles>
```

```
<device>
    <sizing R_minW_nmos="8926" R_minW_pmos="16067"/>
    <area grid_logic_tile_area="0"/>
    <chan_width_distr>
        <x distr="uniform" peak="1.000000"/>
        <y distr="uniform" peak="1.000000"/>
    </chan_width_distr>
    <switch_block type="wilton" fs="3"/>
    <connection_block input_switch_name="ipin_cblock"/>
</device>
```

```
<layout>
    <auto_layout aspect_ratio="1.0">
      <perimeter type="io" priority="100"/>
      <fill type="clb" priority="10"/>
      <col type="mult_36" startx="4" starty="1" repeatx="8" priority="20"/>
      <col type="memory" startx="2" starty="1" repeatx="8" priority="20"/>
    </auto_layout>
</layout>
```

```
<pb_type name="clb">
    <pb_type name="fle" num_pb="10">
        <input name="in" num_pins="6"/>
        <input name="cin" num_pins="1"/>
        <output name="out" num_pins="2"/>
        <output name="cout" num_pins="1"/>
        <clock name="clk" num_pins="1"/>
        <!-- n1_lut6 -->
        <mode name="n1_lut6">
        </mode>
        <!-- n1_lut6 -->
    </pb_type>
</pb_type>
```

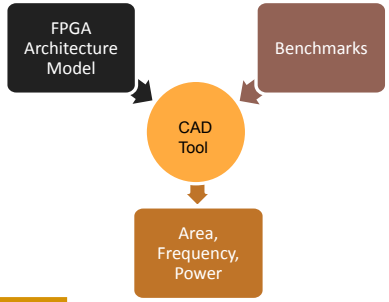
How do you create an FPGA architecture model?

Start from already existing ones in VTR

Capture the architecture attributes from existing FPGAs

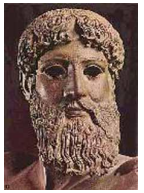
Model it yourself using CAD tools like COFFE

FPGA Benchmark Suites



Benchmark Suite	Medium-Large	Hetero-genous	Open-source CAD	DL-specific
MCNC20	☐	☐	✔	☐
UMass RCG	✔	-	☐	☐
Groundhog	-	✔	-	☐
ERCBench	-	✔	☐	☐
VTR	☐	✔	✔	☐
Titan	✔	✔	☐	☐
Koios	✔	✔	✔	✔

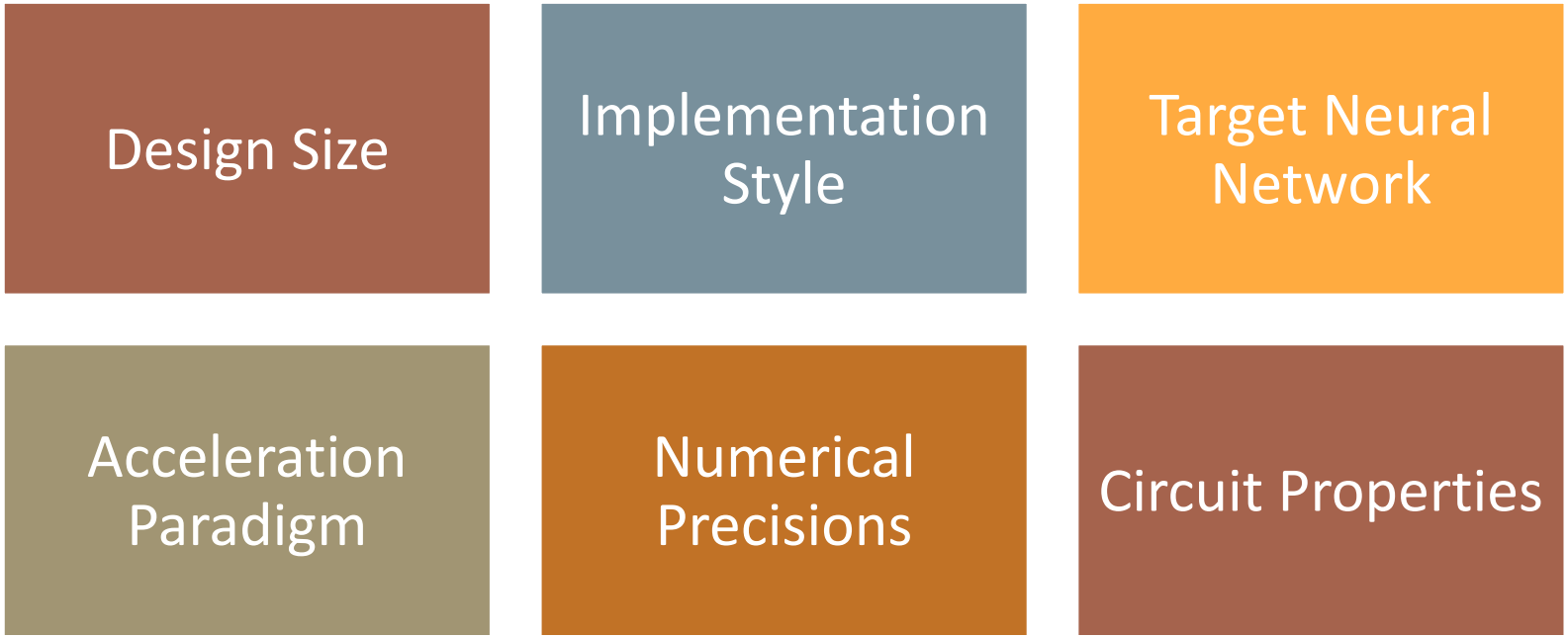
Koios – The Titan of Intelligence



- A DL-specific benchmark suite for FPGA research
- 40 benchmarks that cover a diverse representative space
- Open-source and works with VTR
- Contains original designs, and designs re-created from prior works
- Suitable for DL-specific FPGA architecture exploration and CAD research

Arora et al., Koios: A Deep Learning Benchmark Suite for FPGA Architecture and CAD Research, FPL'21

The Koios Benchmark Suite



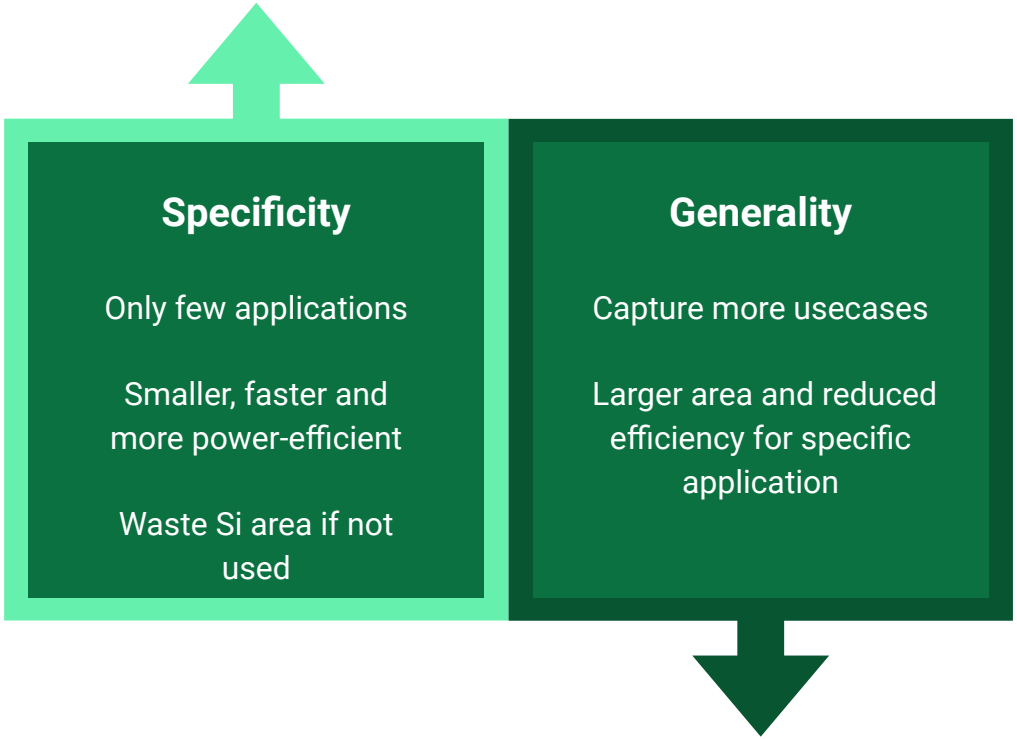
Case Study - Let’s add a new block for DL

How much FPGA die area should be dedicated to it?

What is the impact on programmable routing?

What functionality should be hardened?

How flexible should that block be?



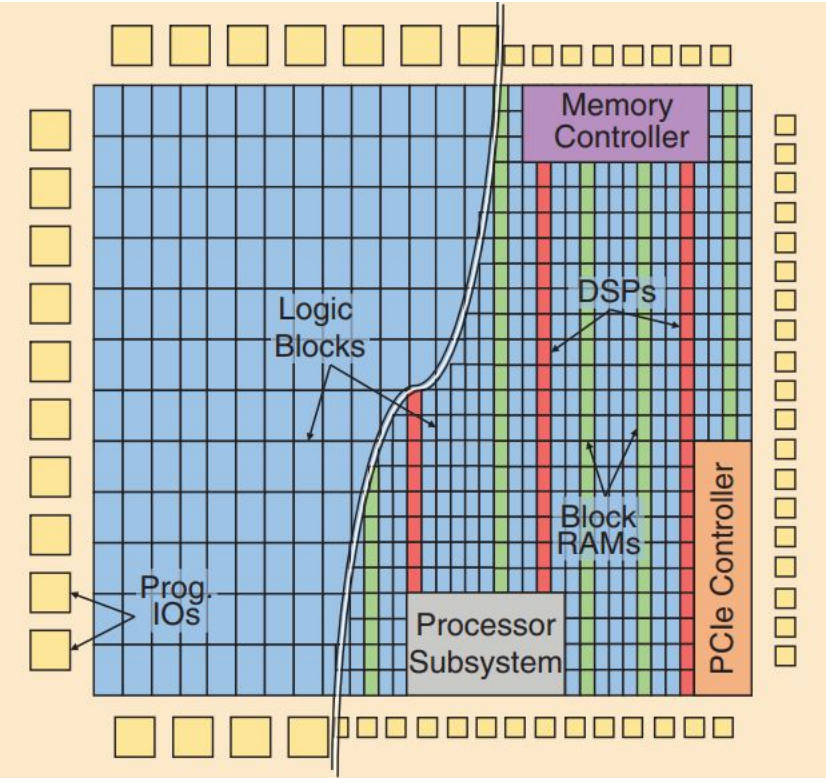
Case Study - Let’s add a new block for DL

Write RTL model	Use COFFE	Use VTR
Functionality, including programmable modes	Implements core with standard cells & programmable routing with full custom → Speed & Area → VTR-compatible model of block	Add new block to a full VTR architecture Use a benchmark suite like Koios Area/timing/routability for new architecture

FPGA DL Accelerators and Architectures

Taxonomy of DL-optimized FPGA Architectures
(Vaughn)

What can we improve?



A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

We are changing the architecture of the FPGA itself

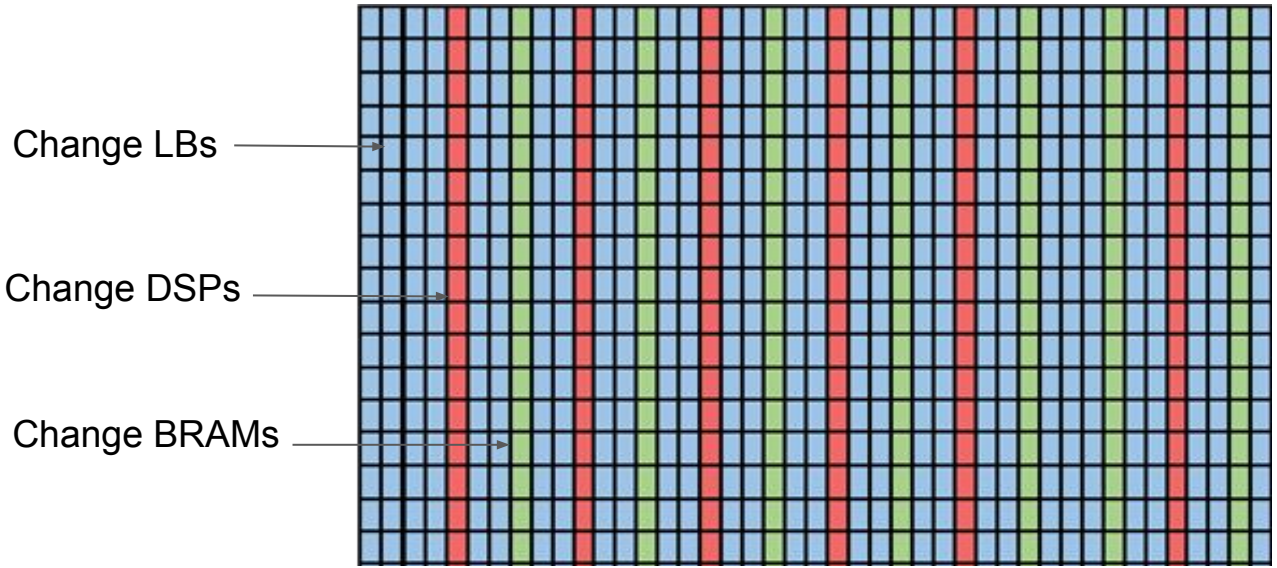
E.g. changing the size of a LUT in a logic block

Not the design configured/programmed into the FPGA

E.g. designing a Brainwave like accelerator for an existing FPGA

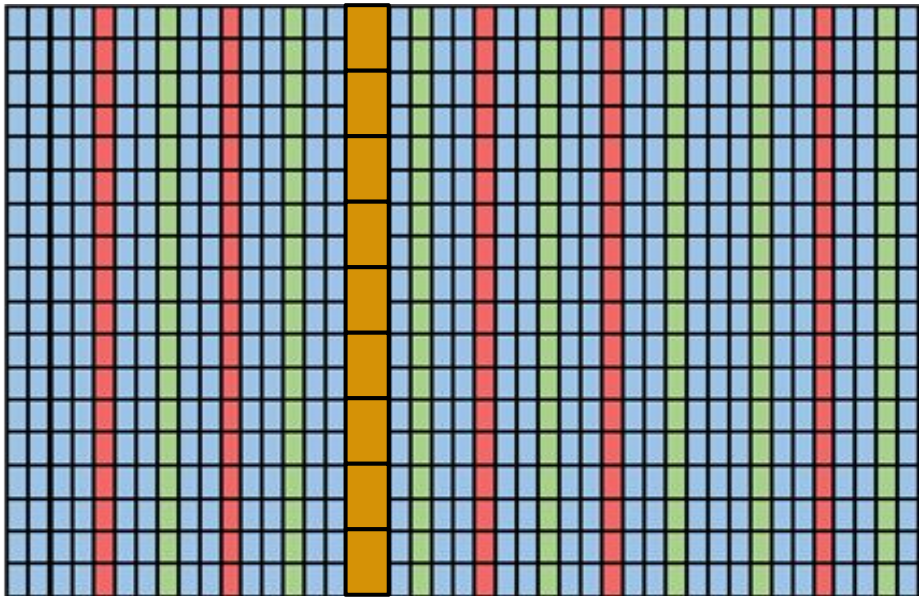
What can we change?

Change existing blocks



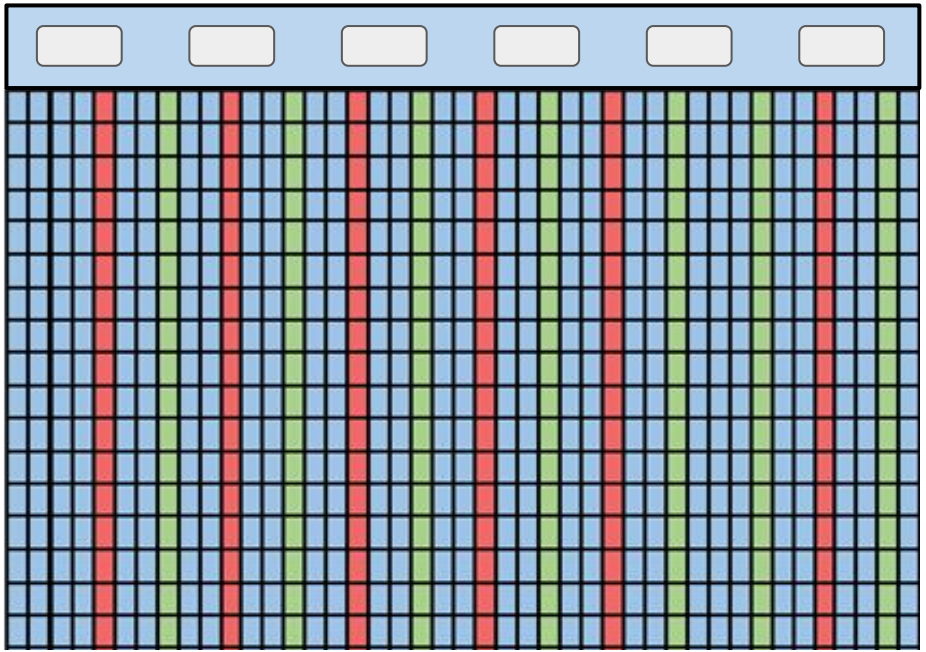
What can we change?

Add new in-fabric blocks



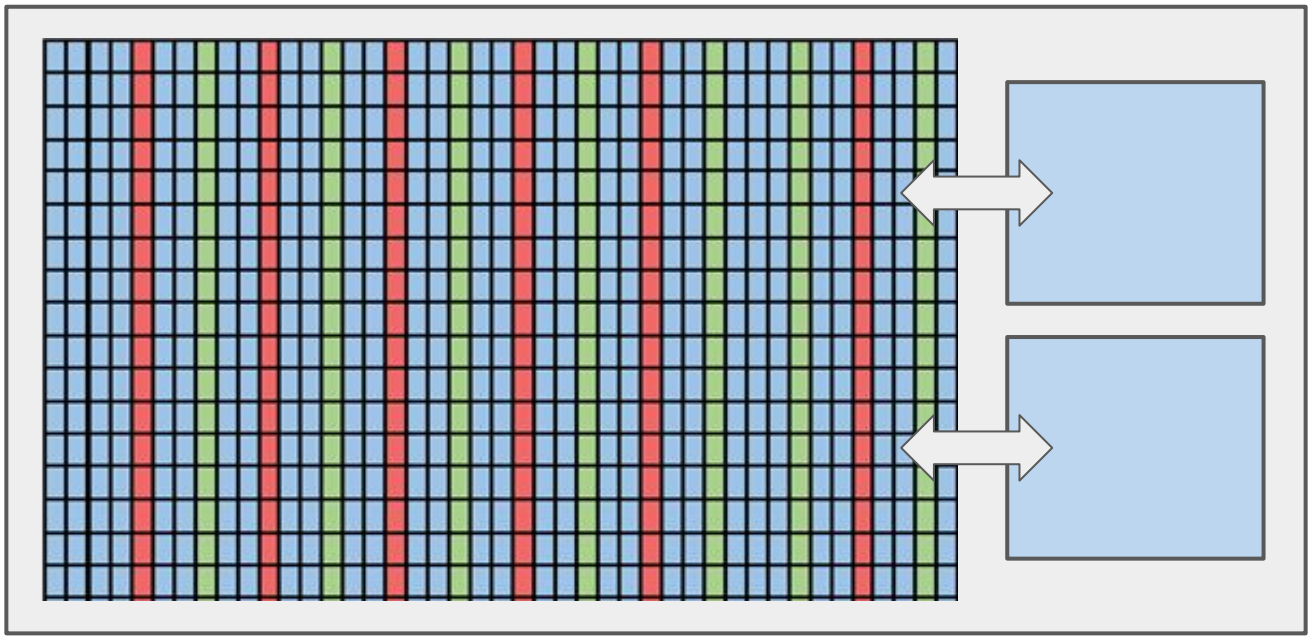
What can we change?

Add new blocks outside the fabric

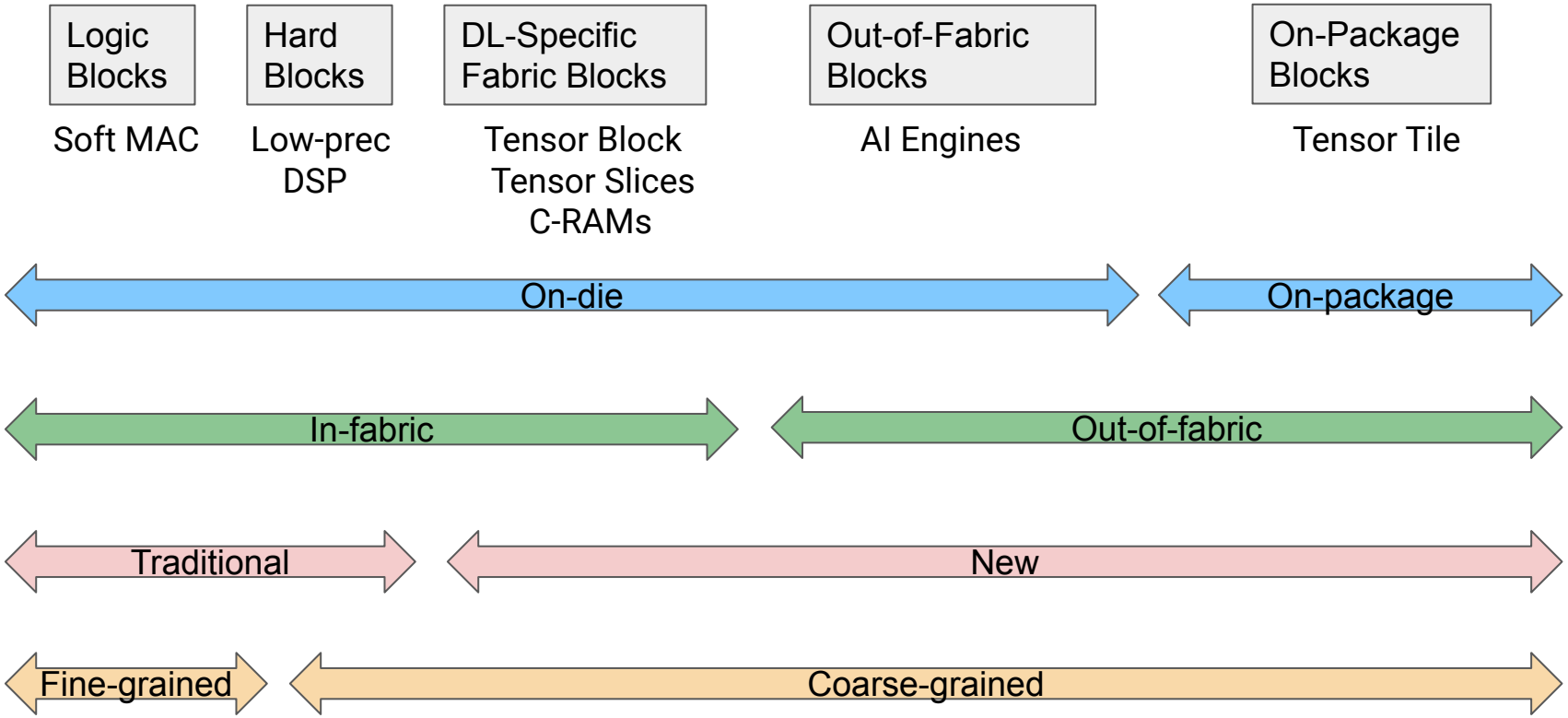


What can we change?

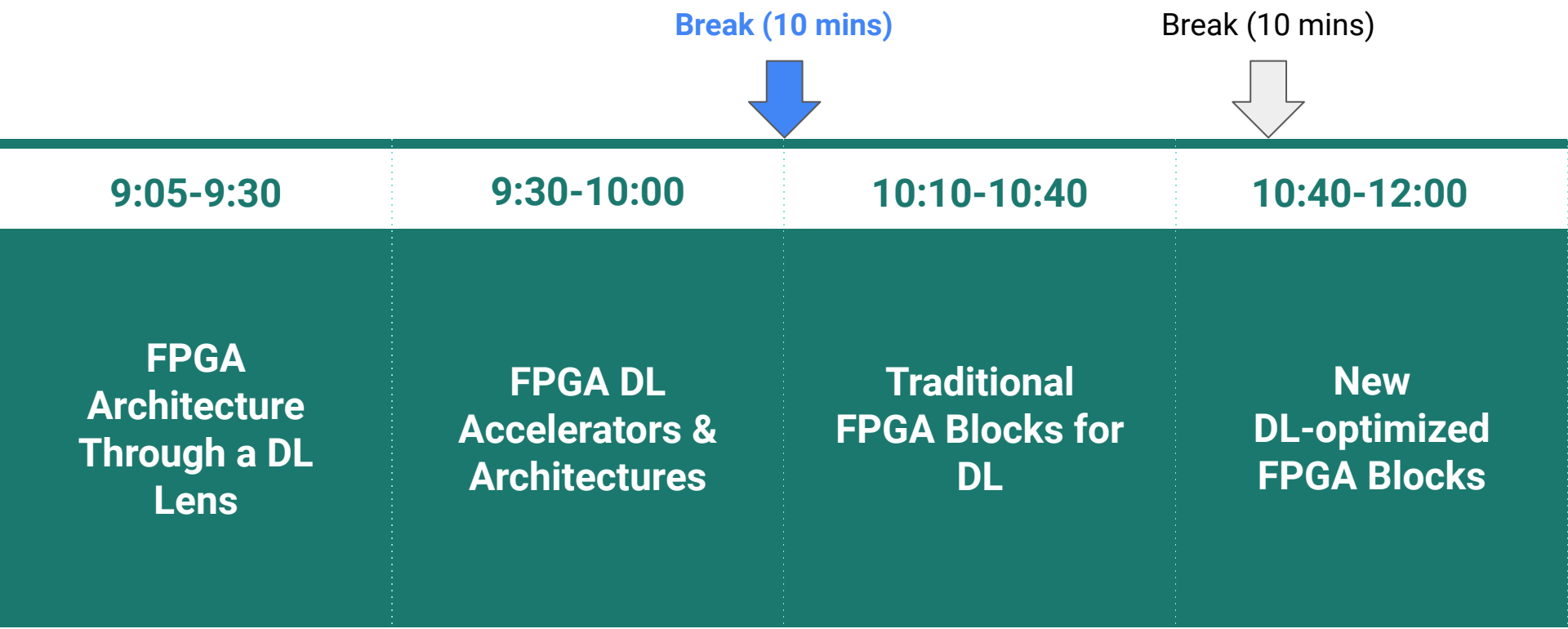
Add new chiplets within a package



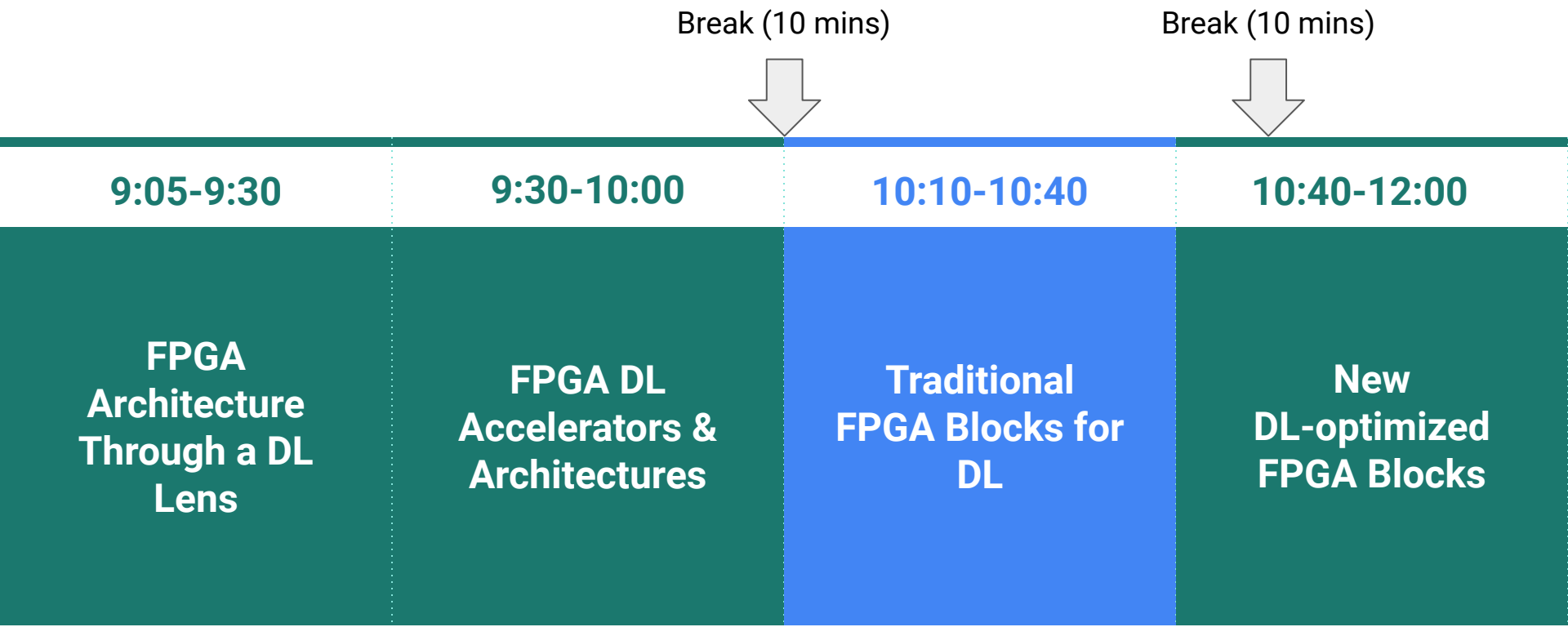
Taxonomy


Logic Blocks
Hard Blocks
DL-Specific Fabric Blocks
Out-of-Fabric Blocks
On-Package Blocks
Soft MAC
Low-prec DSP
Tensor Block
Tensor Slices
C-RAMs
AI Engines
Tensor Tile
On-die
On-package
In-fabric
Out-of-fabric
Traditional
New
Fine-grained
Coarse-grained

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

Traditional FPGA Blocks for DL

Traditional FPGA Blocks for DL

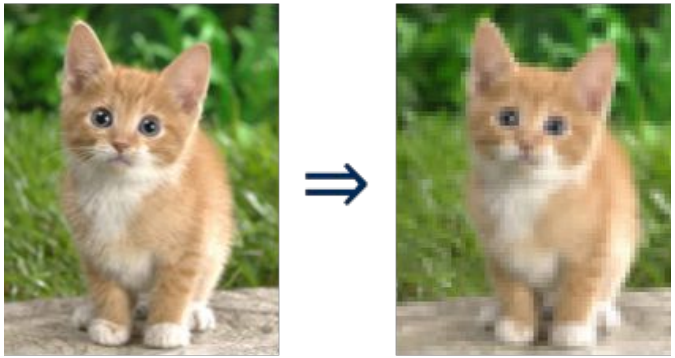
Logic Blocks
(Andrew)

Good News for FPGAs … Low-Precision DL Inference

DL is resilient against noise/approximations
→ Use low precision MACs for inference

1.63254756272 ⇒ 1.633

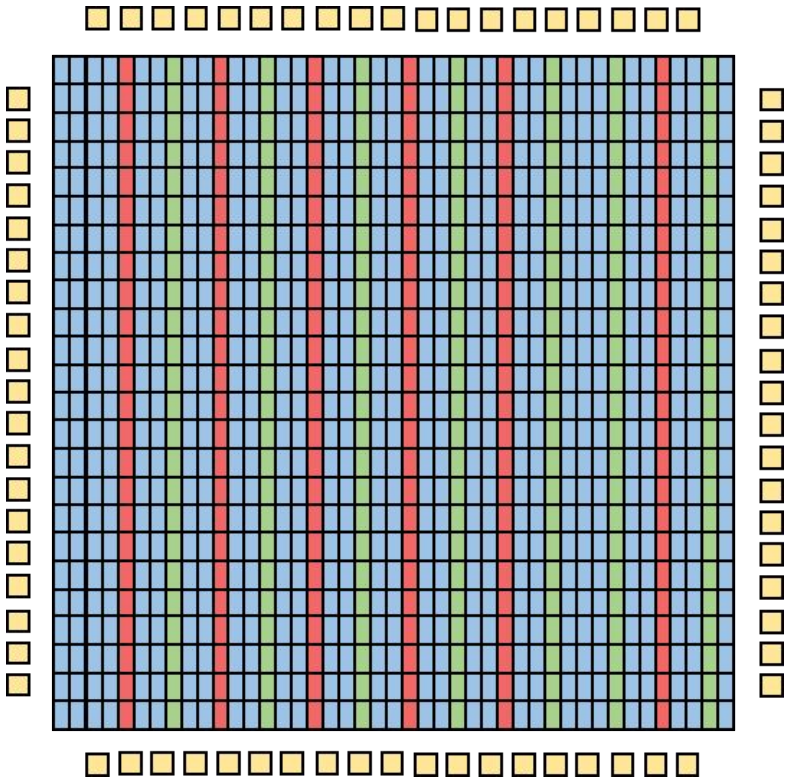
“It is a cat anyway”



Many techniques to enable INT8/INT4 calculations with no accuracy loss
Can sacrifice a bit of accuracy by going down to ternary/binary networks

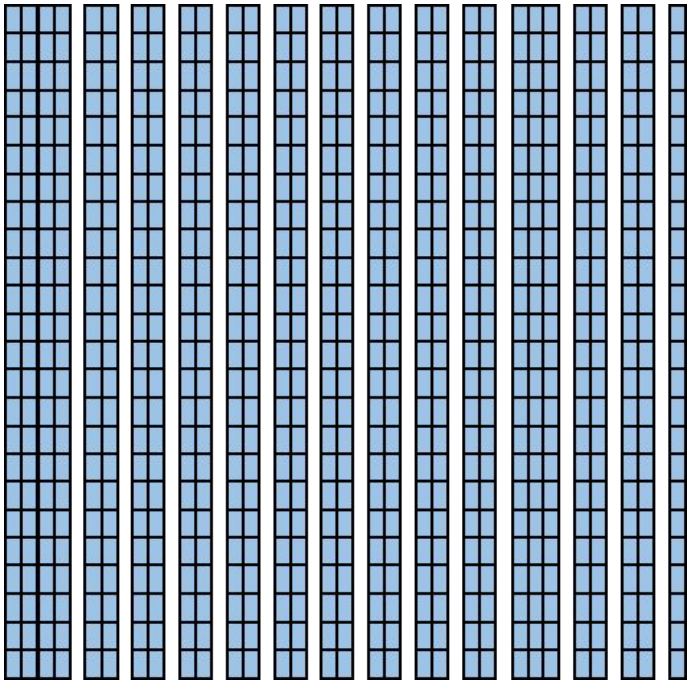
Good news for FPGAs → Can implement custom precisions efficiently!

Looking at Conventional FPGA Architectures …



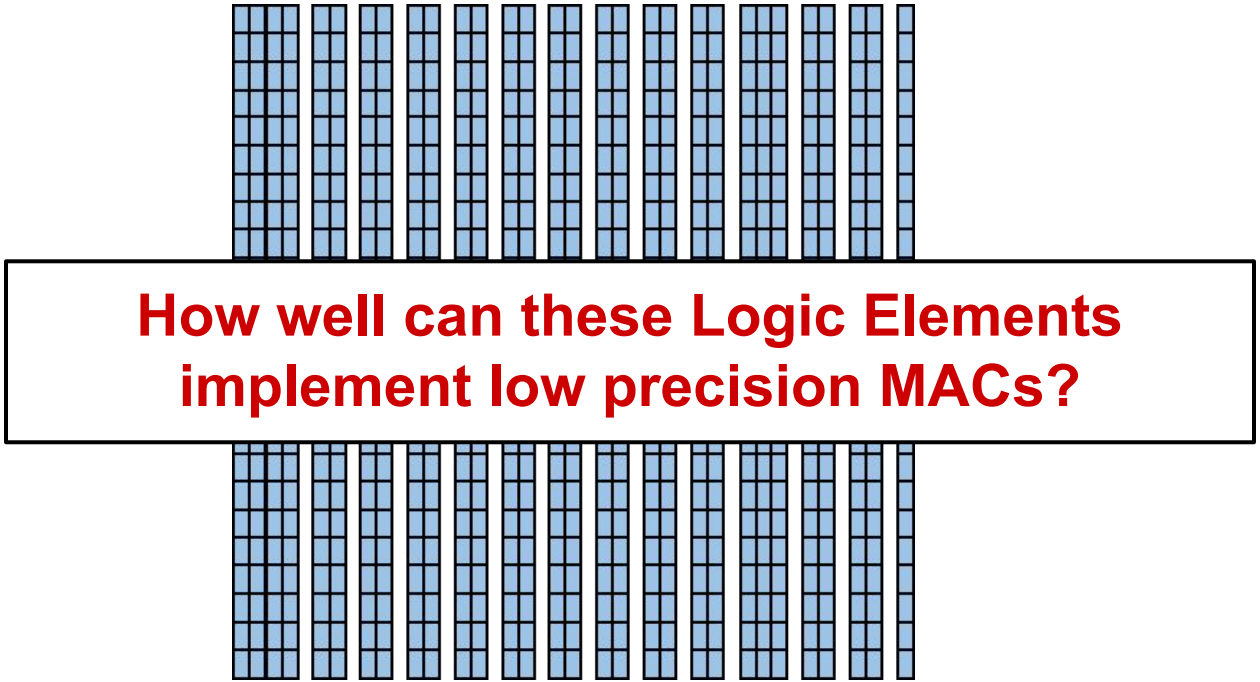
A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

… LEs are the most common blocks in an FPGA



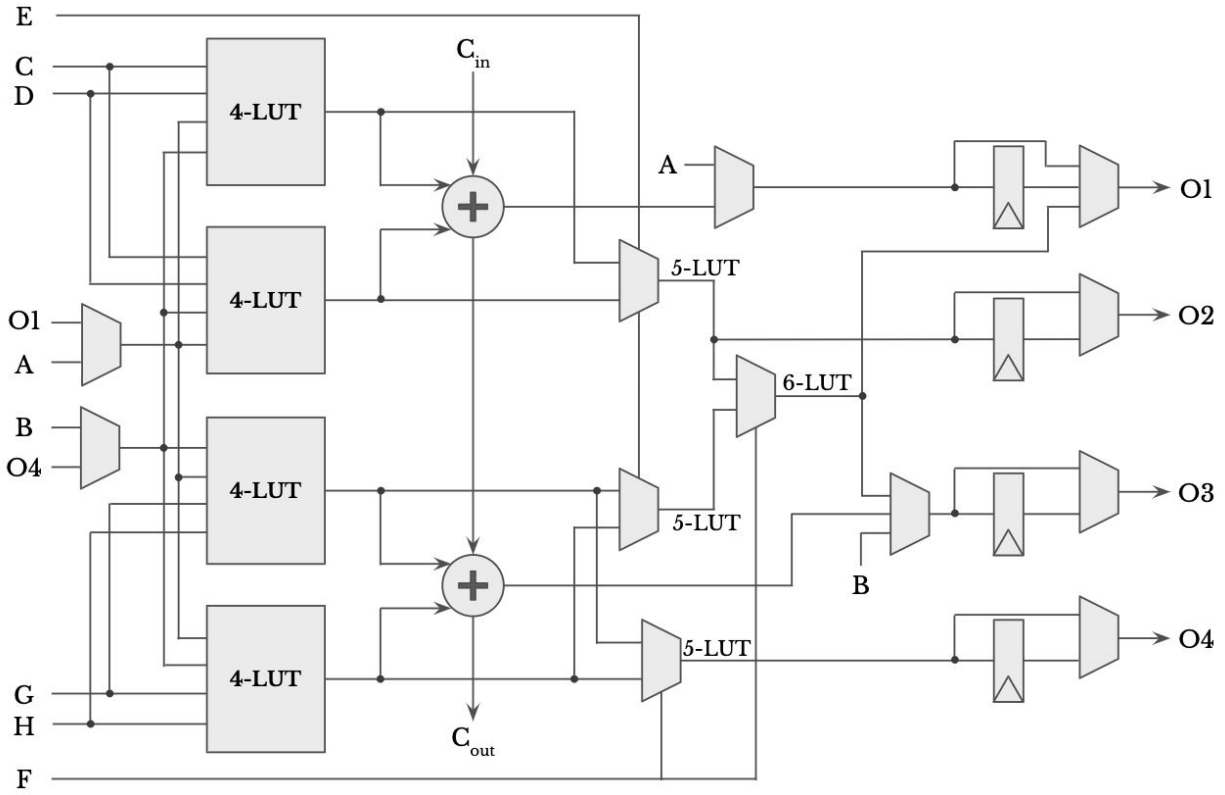
A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

… LEs are the most common blocks in an FPGA



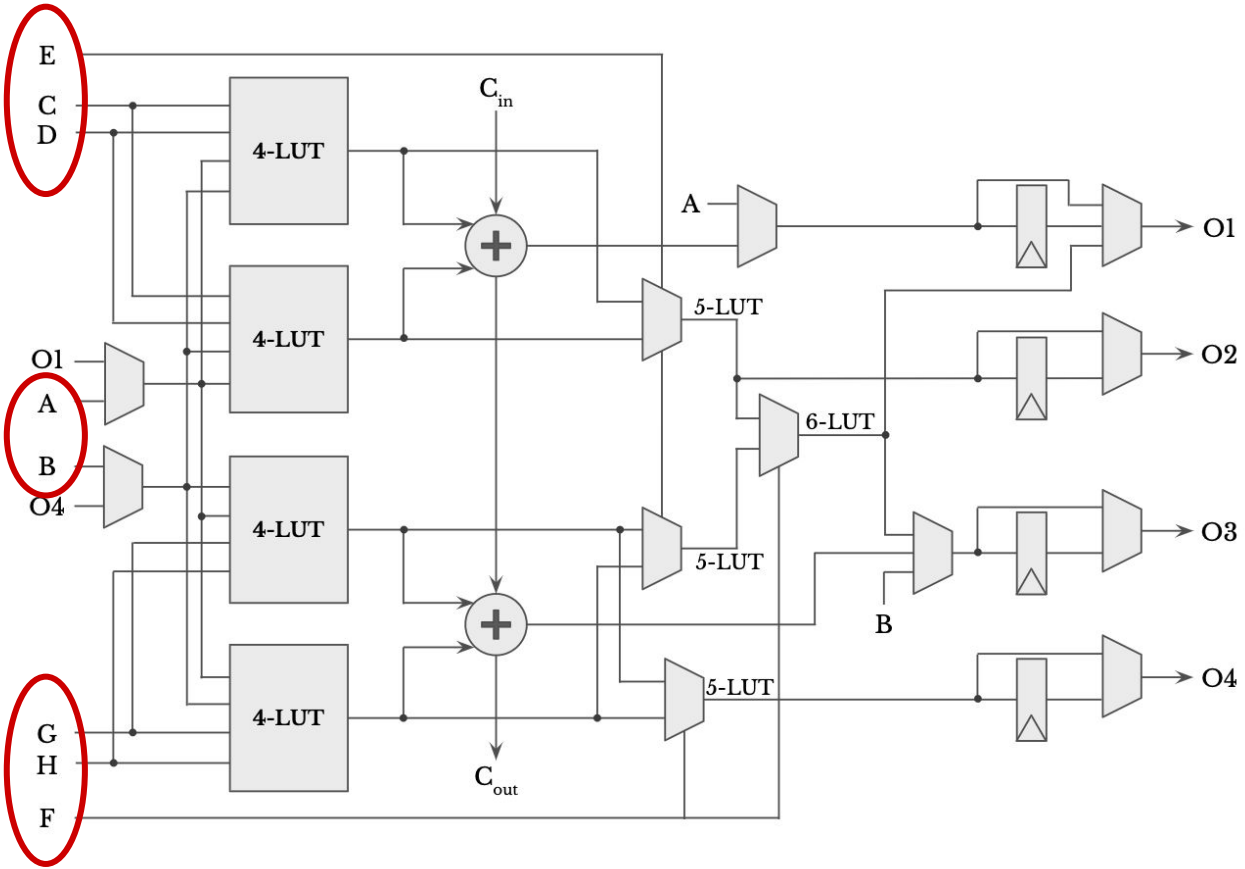
A. Boutros and V. Betz, "FPGA Architecture: Principles and Progression." IEEE Circuits and Systems Magazine 21.2 (2021): 4-29

Deeper Look into a Modern FPGA Logic Element



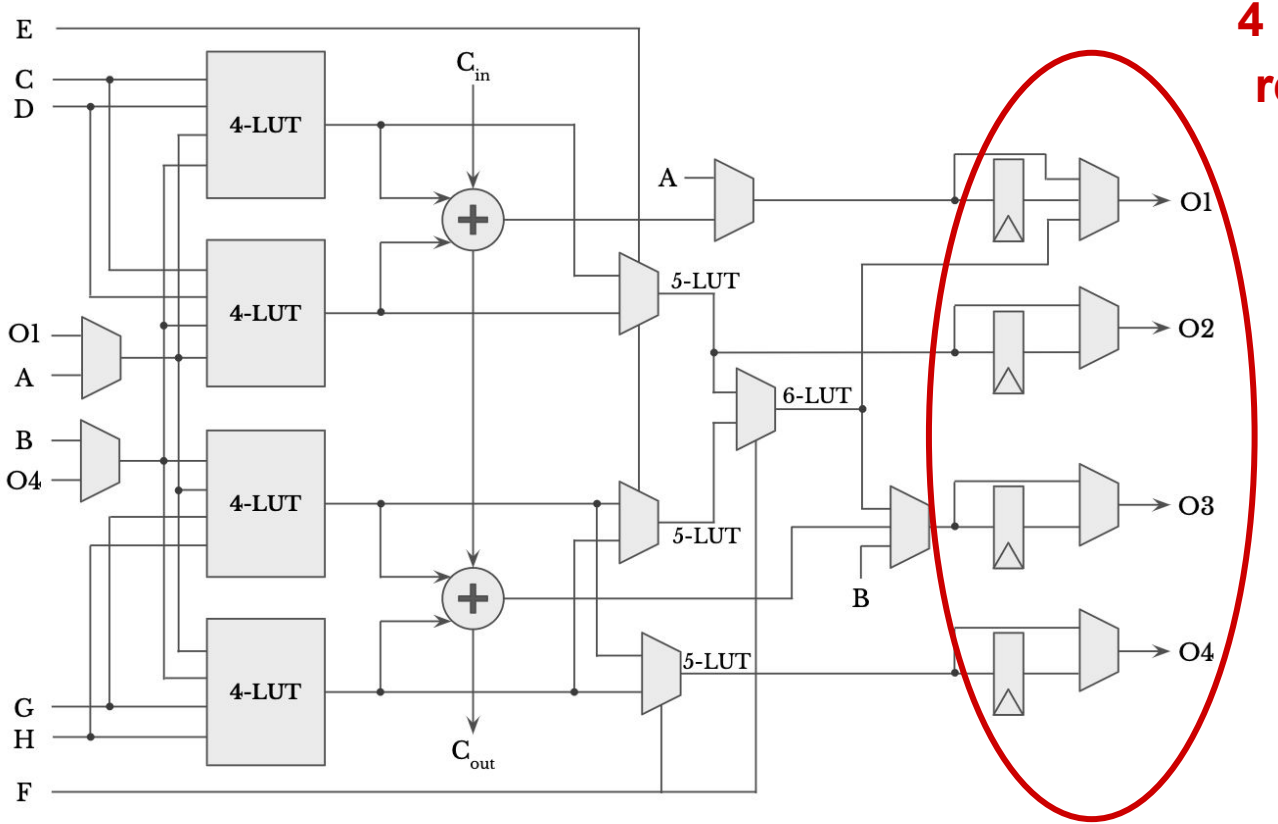
Deeper Look into a Modern FPGA Logic Element

8 distinct inputs A→H



Deeper Look into a Modern FPGA Logic Element

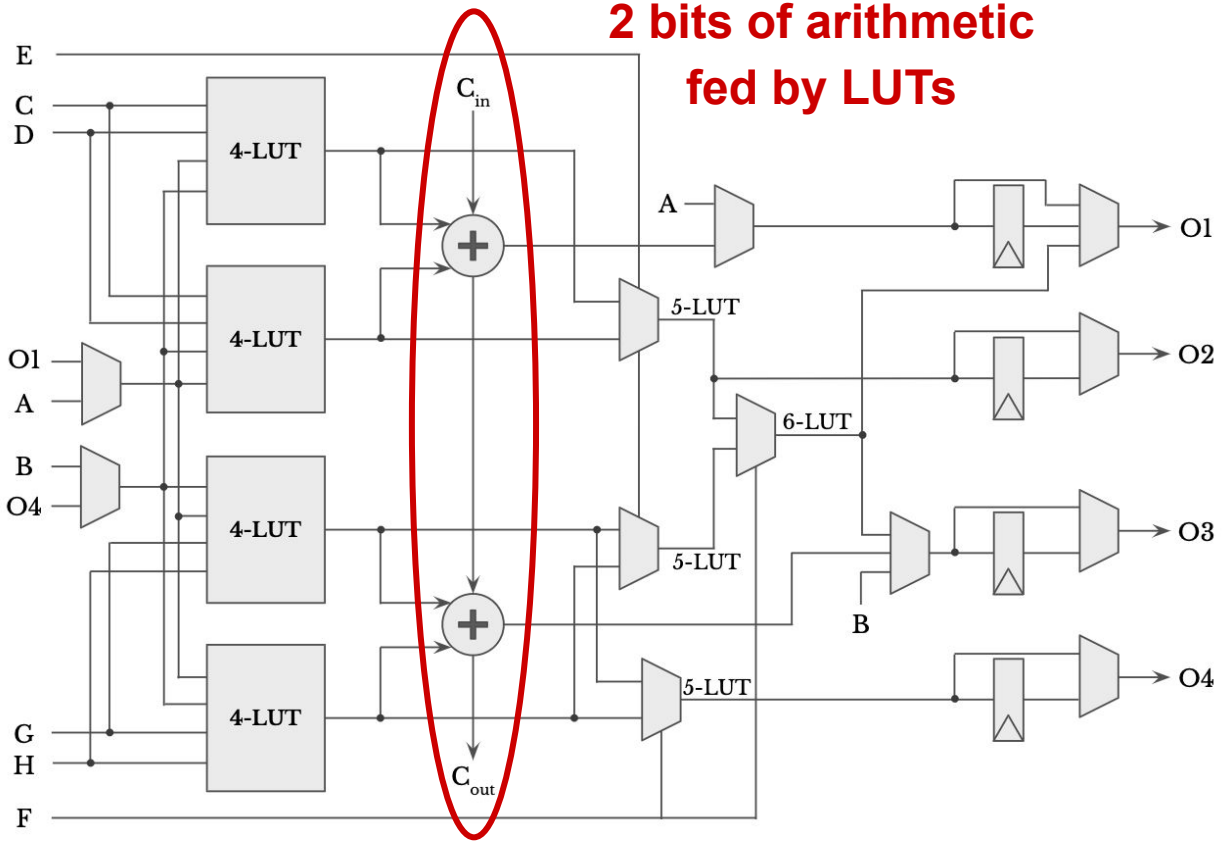
8 distinct inputs A→H



4 optionally registered outputs O1→O4

Deeper Look into a Modern FPGA Logic Element

8 distinct inputs A→H



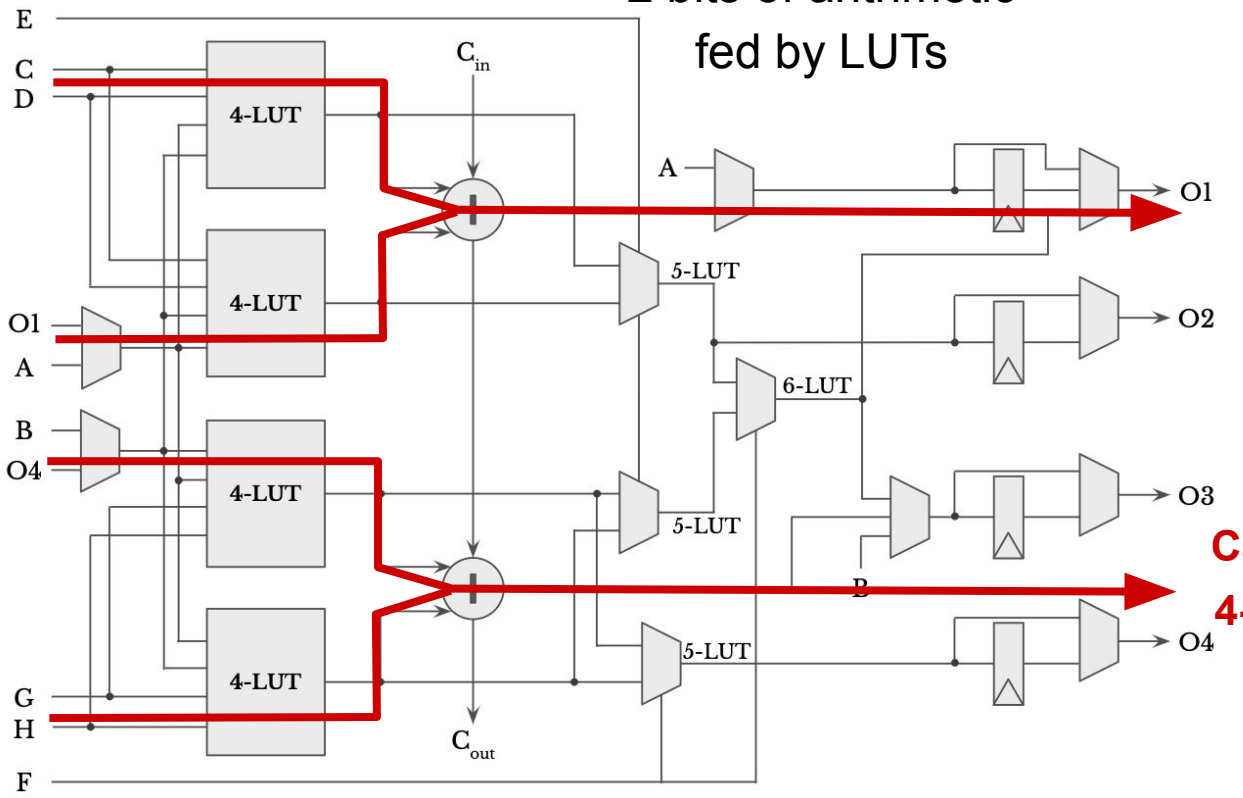
4 optionally registered outputs O1→O4

Deeper Look into a Modern FPGA Logic Element

2 bits of arithmetic fed by LUTs

4 optionally registered outputs O1→O4

8 distinct inputs A→H



Can implement:

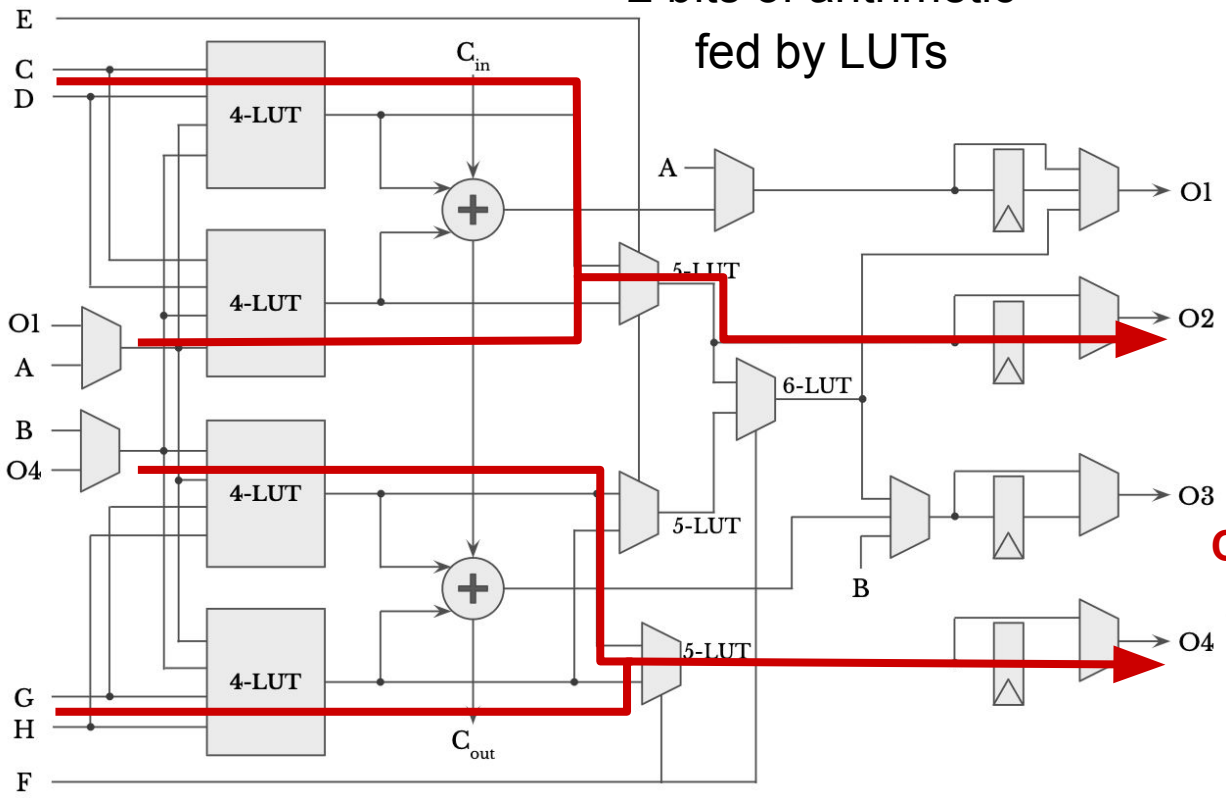
4-in logic→Add

Deeper Look into a Modern FPGA Logic Element

2 bits of arithmetic fed by LUTs

4 optionally registered outputs O1→O4

8 distinct inputs A→H



Can implement:

4-in logic→Add

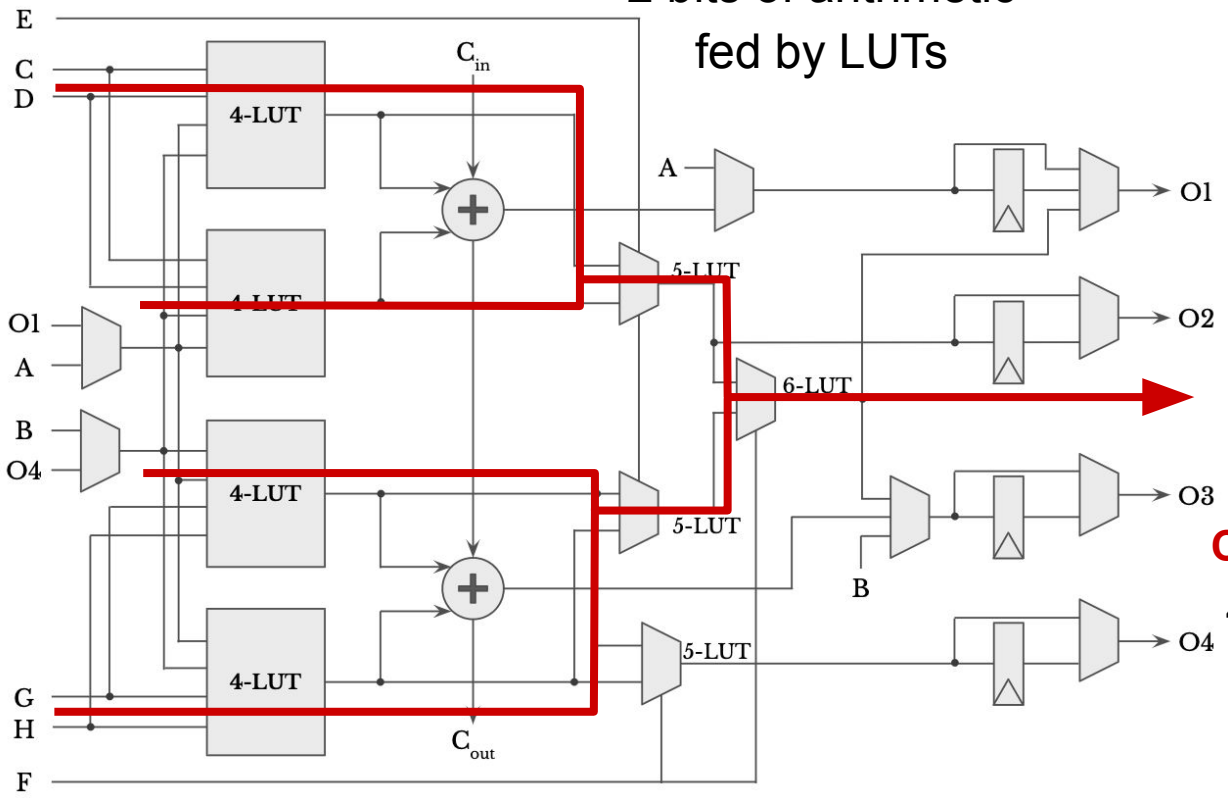
2x 5-in logic

Deeper Look into a Modern FPGA Logic Element

2 bits of arithmetic fed by LUTs

4 optionally registered outputs O1→O4

8 distinct inputs A→H



Can implement:

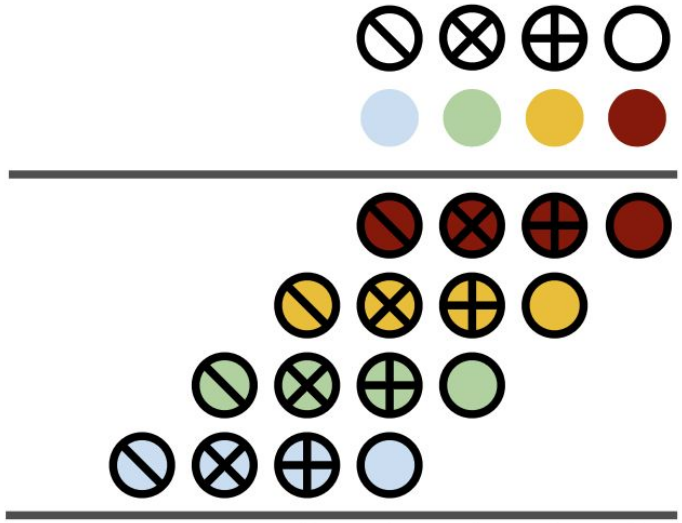
4-in logic→Add

2x 5-in logic

1x 6-in logic

Back to Grade 3 Maths …

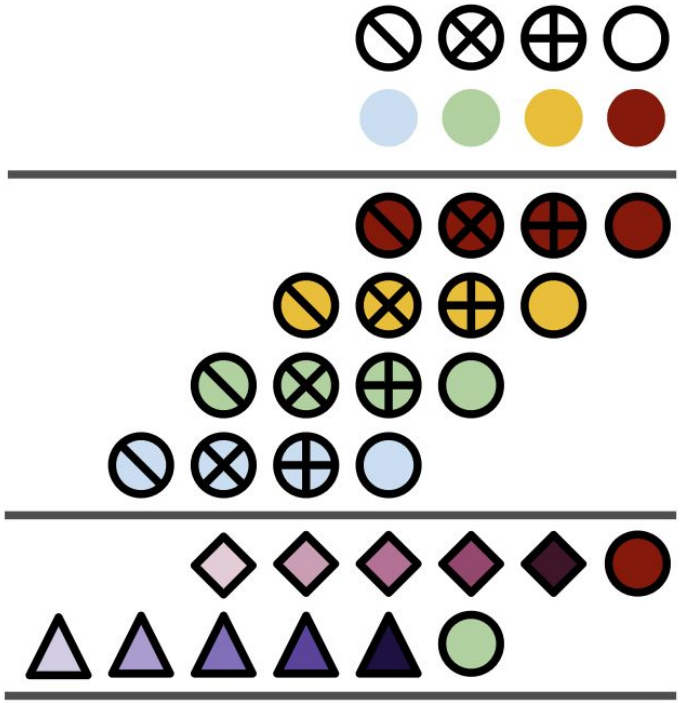
Back to Grade 3 Maths …



Inputs

Generate Partial Products (AND Gates)

Back to Grade 3 Maths …

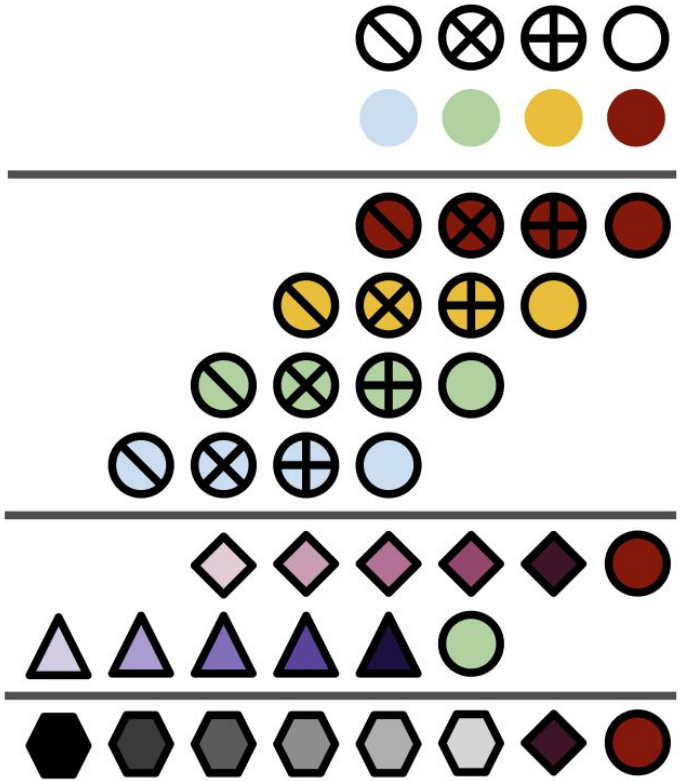


Inputs

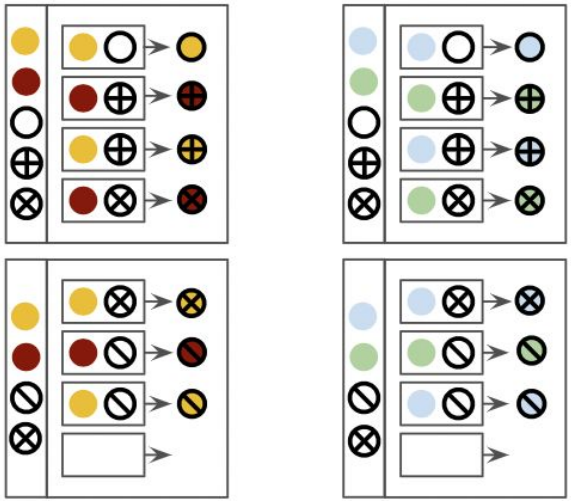
Generate Partial Products (AND Gates)

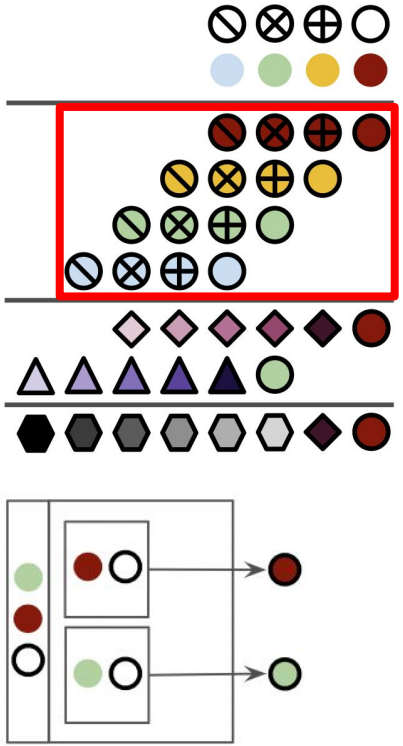
Reduce Partial Products

Back to Grade 3 Maths …

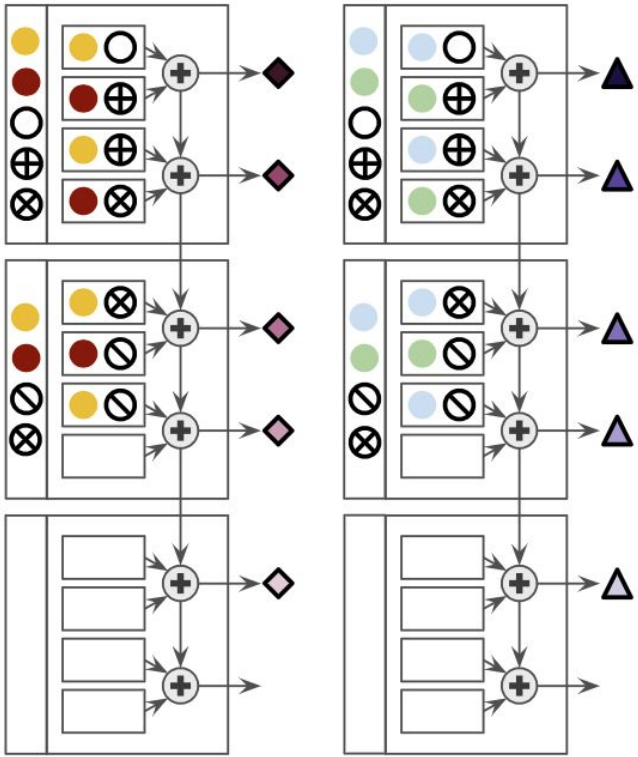


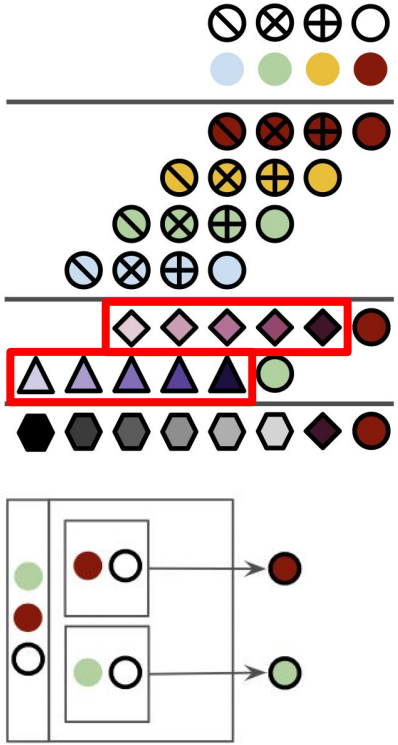
How is it mapped?



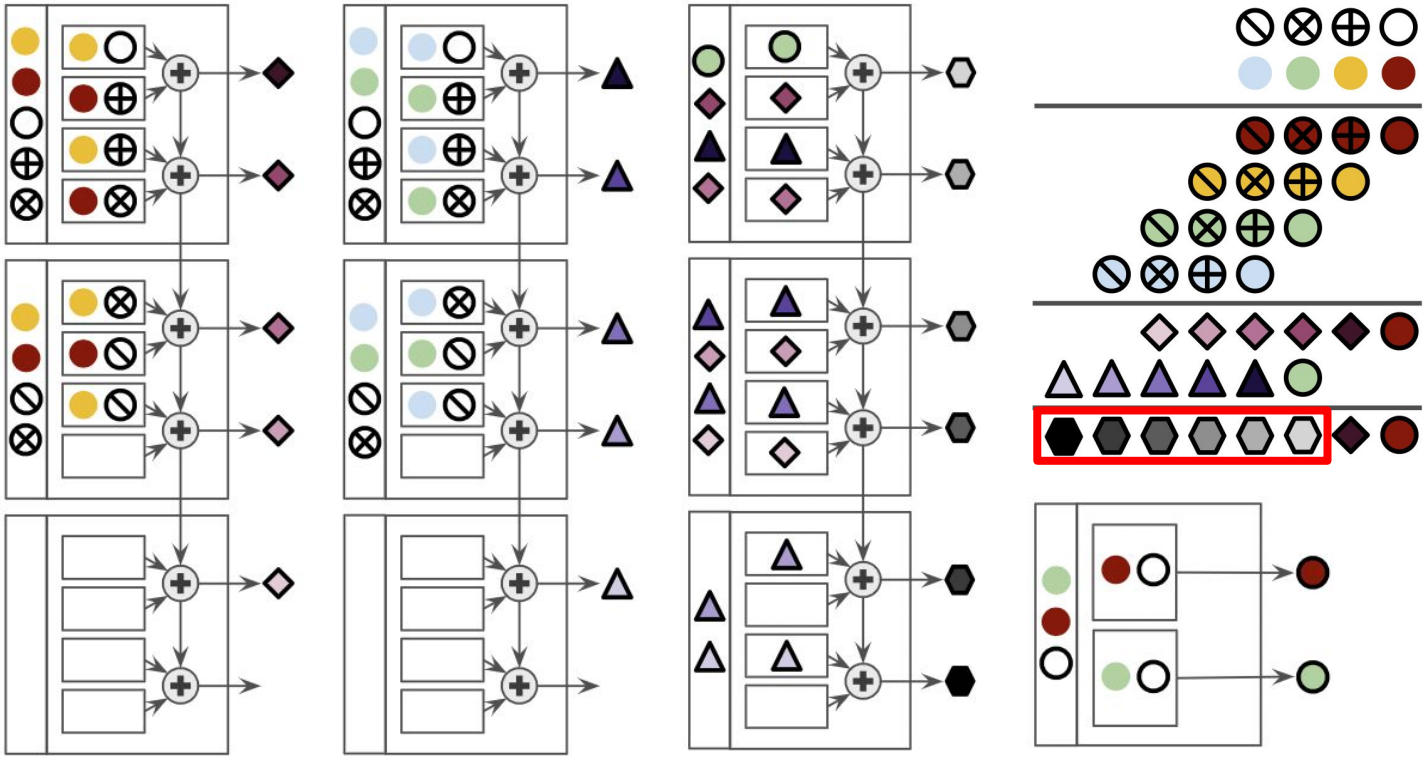


How is it mapped?





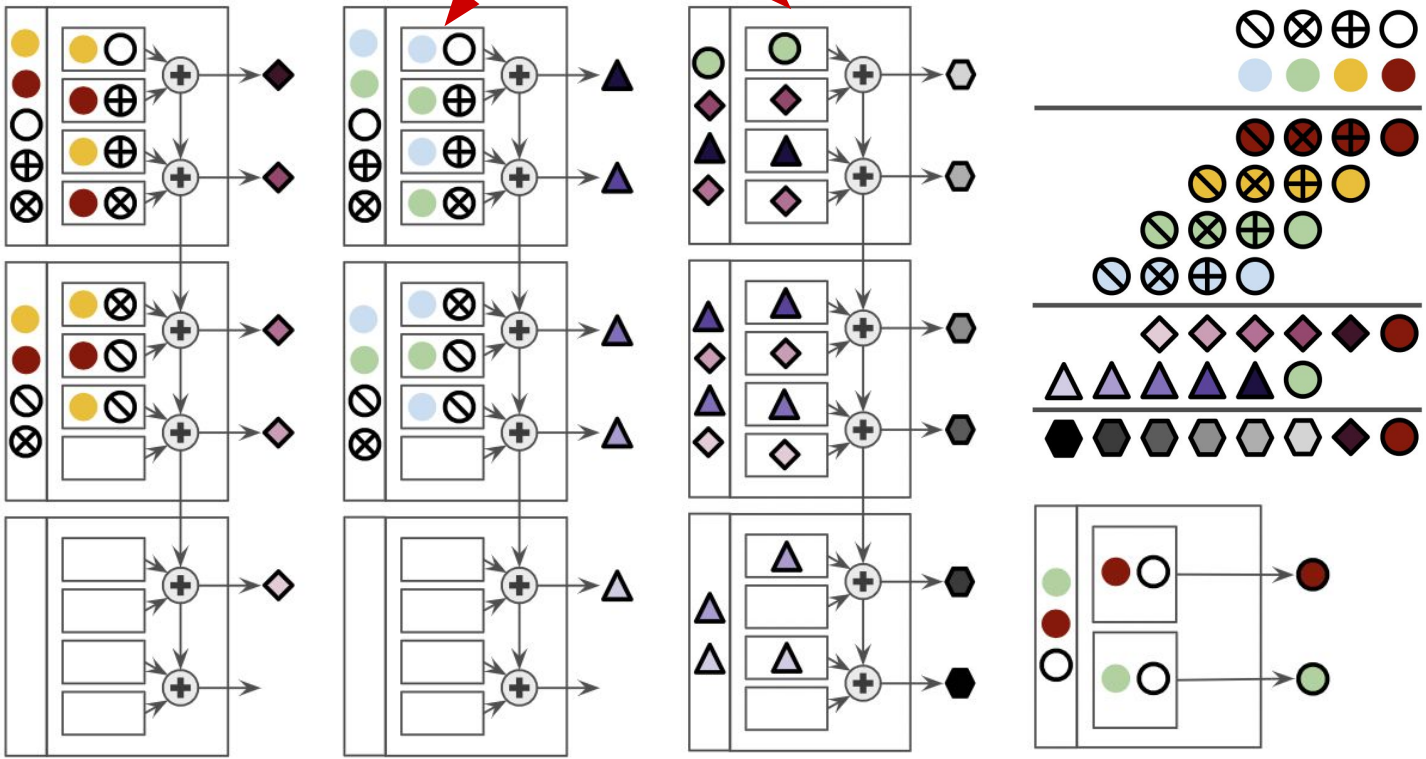
How is it mapped?



How is it mapped?

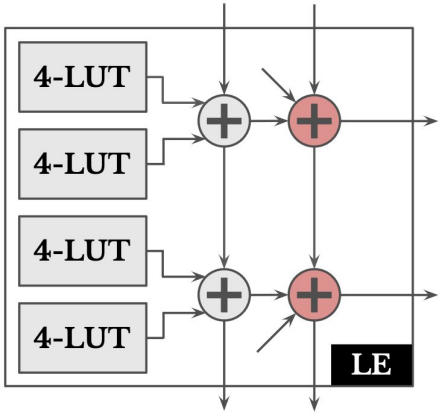
4-input LUT used to implement 2-input AND

Some LEs only used for adders



3 Suggested Ideas

Idea 1: Add a 2nd carry chain

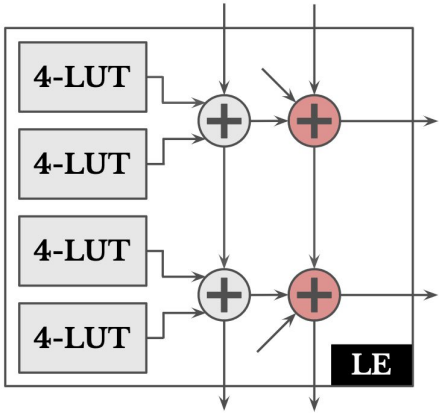


More efficient adder compressor trees

M. Eldafrawy and others, "FPGA Logic Block Architectures for Efficient Deep Learning Inference", TRETS, 2020

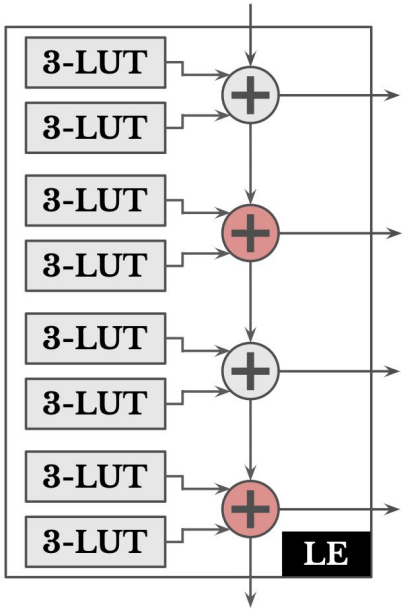
3 Suggested Ideas

Idea 1: Add a 2nd carry chain



More efficient adder compressor trees

Idea 2: Add more adders in the chain

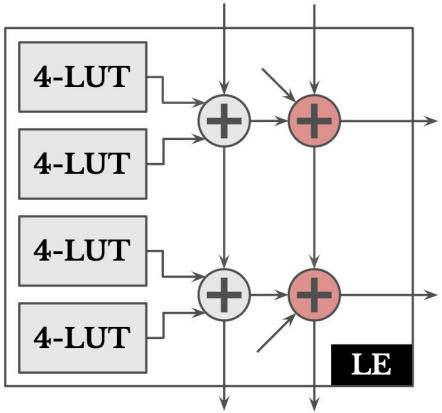


Denser arithmetic in general

M. Eldafrawy and others, "FPGA Logic Block Architectures for Efficient Deep Learning Inference", TRETS, 2020

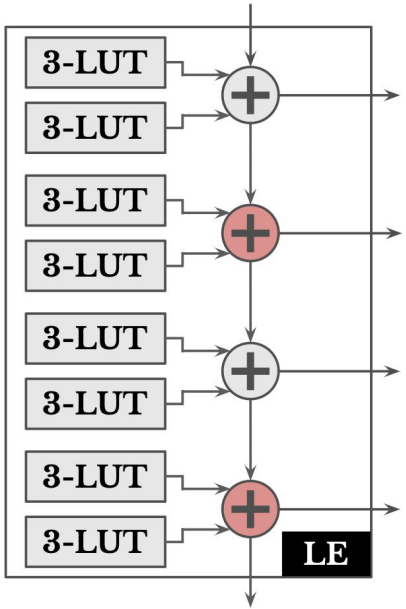
3 Suggested Ideas

Idea 1: Add a 2nd carry chain



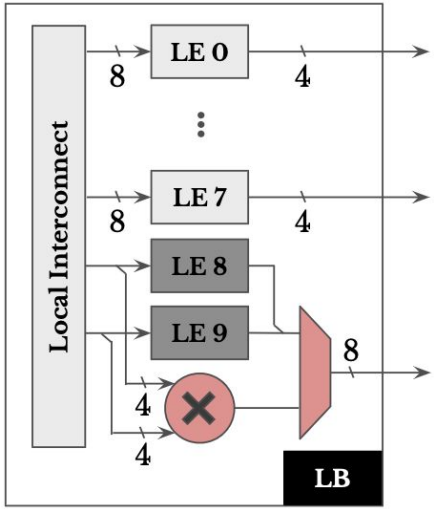
More efficient adder compressor trees

Idea 2: Add more adders in the chain



Denser arithmetic in general

Idea 3: Add shadow multipliers to LE clusters (reuse ports)



Denser multiplies vs. implementing in LEs

M. Eldafrawy and others, "FPGA Logic Block Architectures for Efficient Deep Learning Inference", TRETS, 2020

Results

Low Cost Change: 4-bit chain (Idea 2)

1.5x denser + 10% faster

Also benefits other non-DL applications

3% die area increase

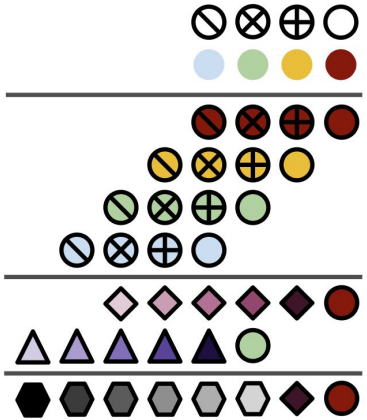
High Gain Change: 9-bit Shadow Multipliers (Idea 3)

2.4x denser + 17% faster

12% die area increase

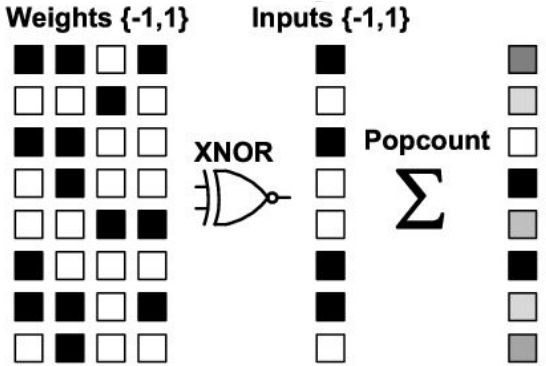
M. Eldafrawy and others, "FPGA Logic Block Architectures for Efficient Deep Learning Inference", TRETS, 2020

It’s all about Adders …

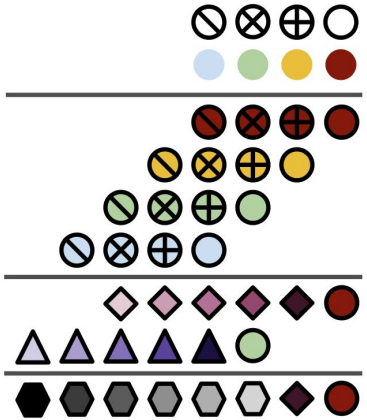


In MACs
Sum partial products to get final output

In Binary NNs
Sum XNOR outputs to get final output

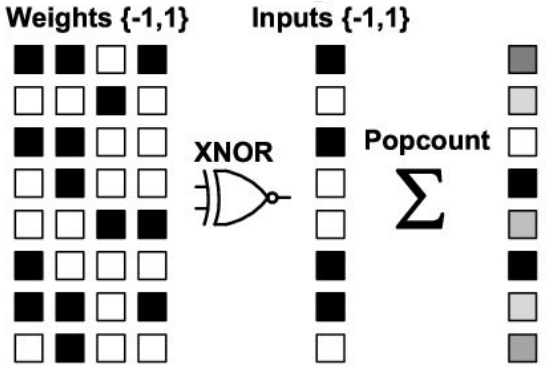


It’s all about Adders …



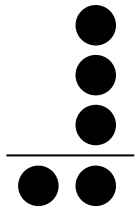
In MACs
Sum partial products to get final output

In Binary NNs
Sum XNOR outputs to get final output

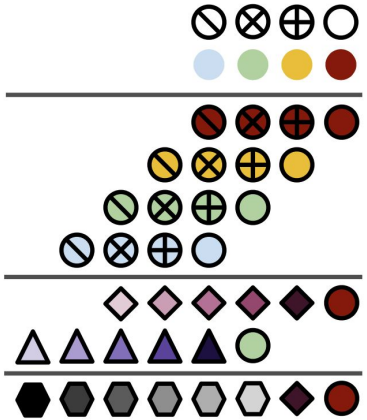


In the previous ideas, we add more full adders to FPGA LEs …

Full adders: 3 in → 2 bits (1 same significance + 1 higher order)

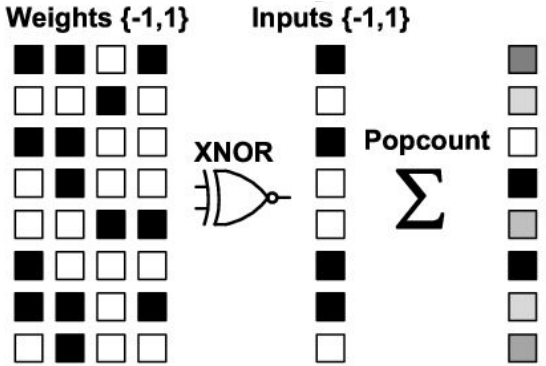


It's all about Adders …



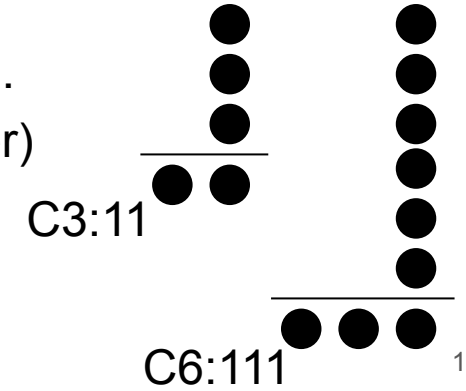
In MACs
Sum partial products to get final output

In Binary NNs
Sum XNOR outputs to get final output



In the previous ideas, we add more full adders to FPGA LEs …
Full adders: 3 in → 2 bits (1 same significance + 1 higher order)

In many cases, we need to add more than 3 bits
Compressors: N in → M bits



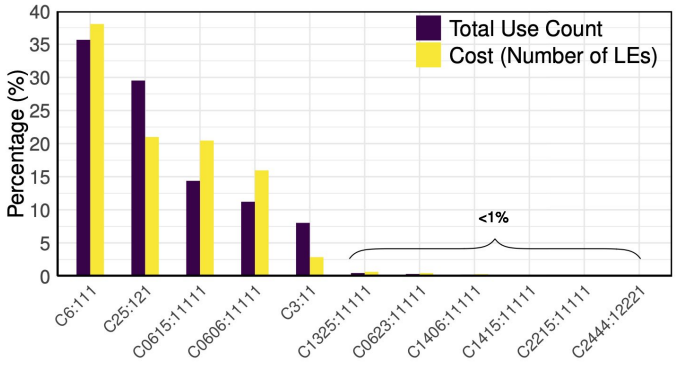
Compressors are important …

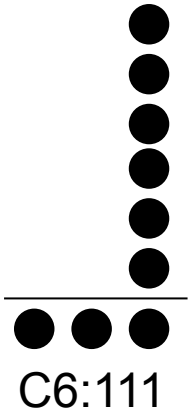
Generalized parallel counters/adders (i.e. compressors) are not efficient when mapped to FPGA LEs

Across many microbenchmarks

>35% of compressors are C6:111

Most expensive on FPGAs

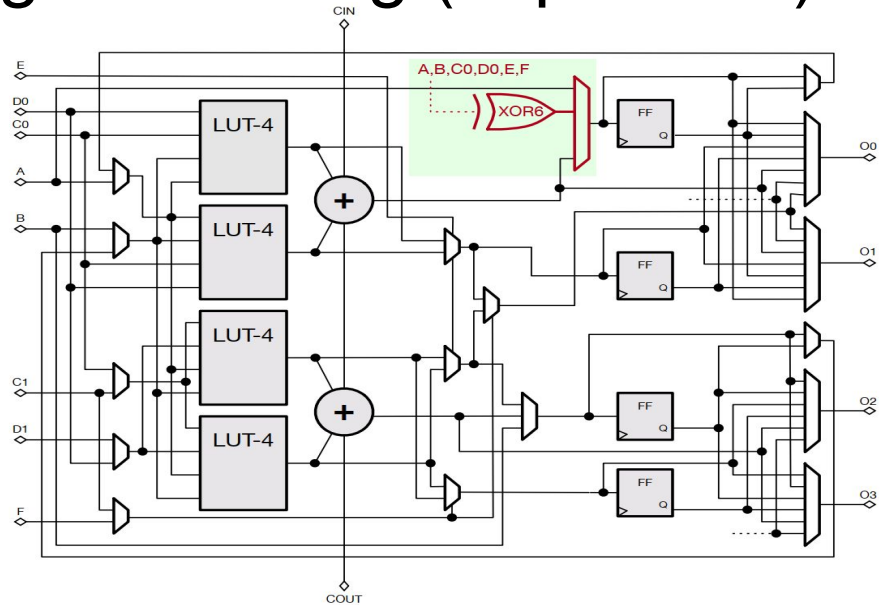




Can we improve efficiency with minimal additions to FPGA LEs?

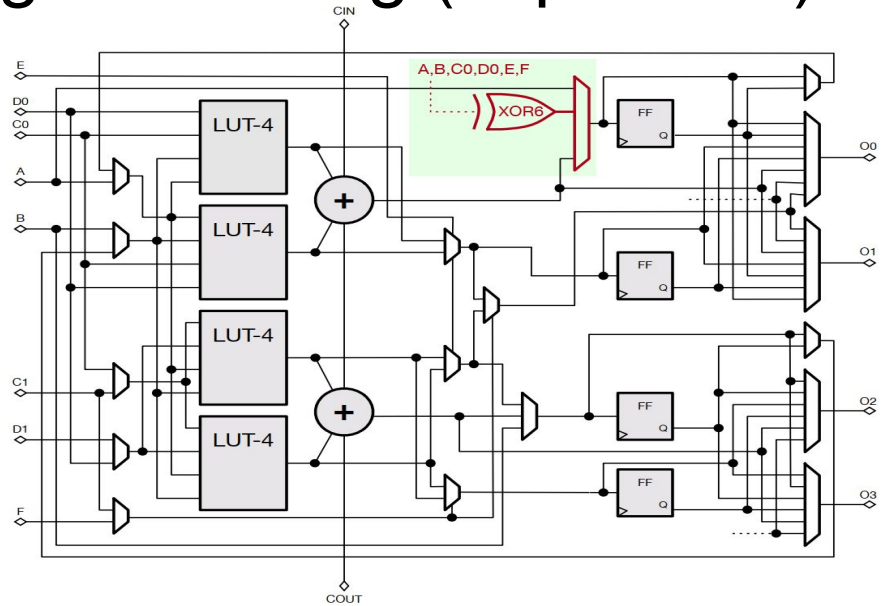
S. Rasoulinezhad and others, "LUXOR: An FPGA Logic Cell Architecture for Efficient Compressor Tree Implementations", FPGA, 2020

6-Input XOR gate sharing (expensive) LE inputs



S. Rasoulinezhad and others, "LUXOR: An FPGA Logic Cell Architecture for Efficient Compressor Tree Implementations", FPGA, 2020

6-Input XOR gate sharing (expensive) LE inputs



< 0.5% area overhead

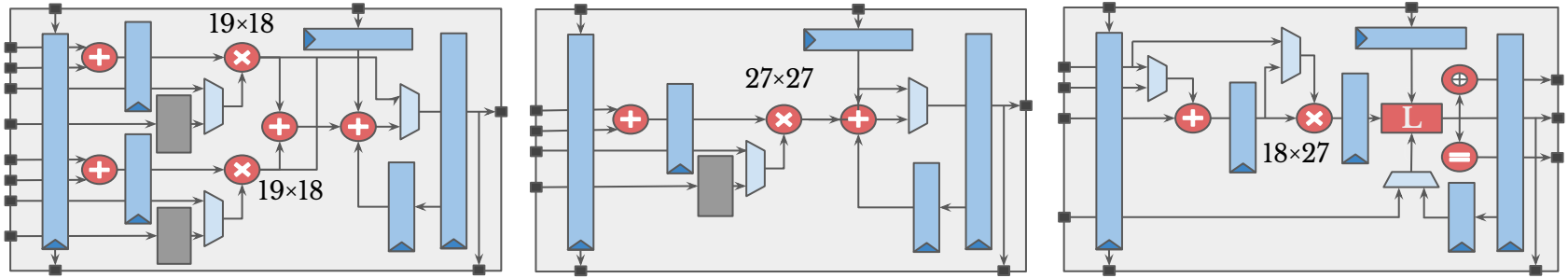
Up to **36%** denser compressor implementations

S. Rasoulinezhad and others, "LUXOR: An FPGA Logic Cell Architecture for Efficient Compressor Tree Implementations", FPGA, 2020

Traditional FPGA Blocks for DL

DSP Blocks
(Andrew)

Traditional FPGA DSPs



Intel DSP blocks (19x18 and 27x27 modes)

Xilinx DSP block

- Optimized for wireless communication & filtering
  - Intel → 2x18b or 1x27b multiplication
  - Xilinx → 1x 18x27
- Can do better if we care about low-precision MACs

Key Design Goals

- Backward compatibility
  - Usable for other applications

- Ideally no effect on block frequency
  - No negative impact on non-DL designs

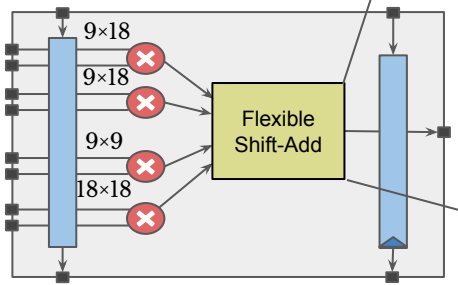
- Do not add (relatively expensive) routing ports
  - Avoid large area cost
  - Avoid potential routing hotspots

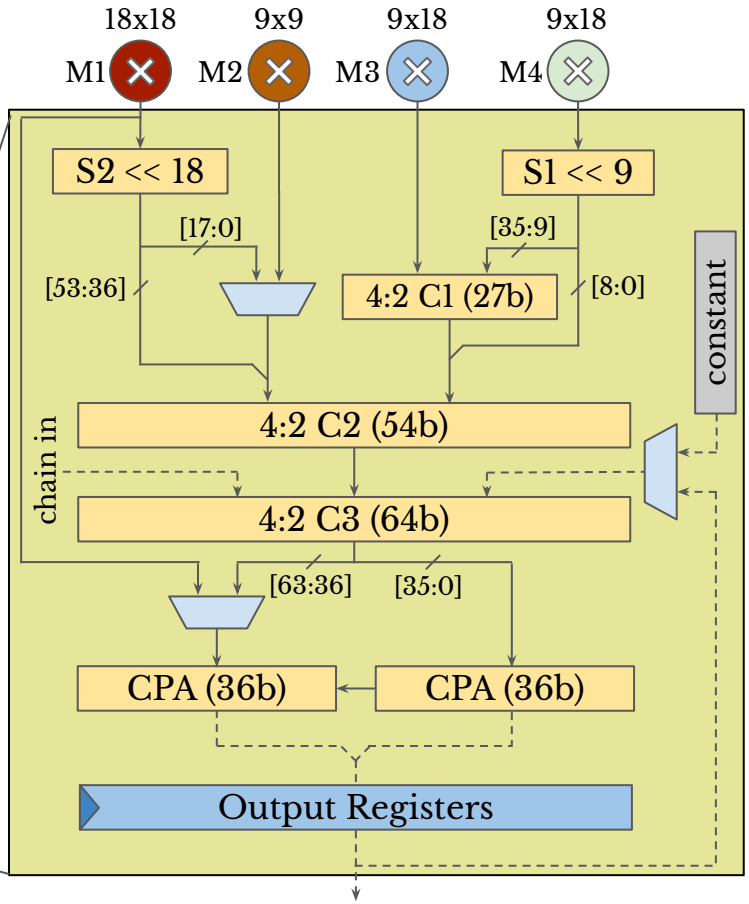
More Fracturability …

- Support traditional modes (27×27,18x18)
- Add new low-precision modes
  - 4x 9b multiply/MAC
  - 8x 4b multiply/MAC

Keeping it low-cost is key …

- No additional routing ports
- Max reuse of existing multiplier arrays





A. Boutros et al., "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," FPL18

Low Area Overhead

Block area overhead:

12% DSP Block Area

which is equivalent to

~0.6% FPGA Core Area

in DSP-rich devices

… and runs at the same frequency

A. Boutros et al., "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," FPL18

Application-level Results

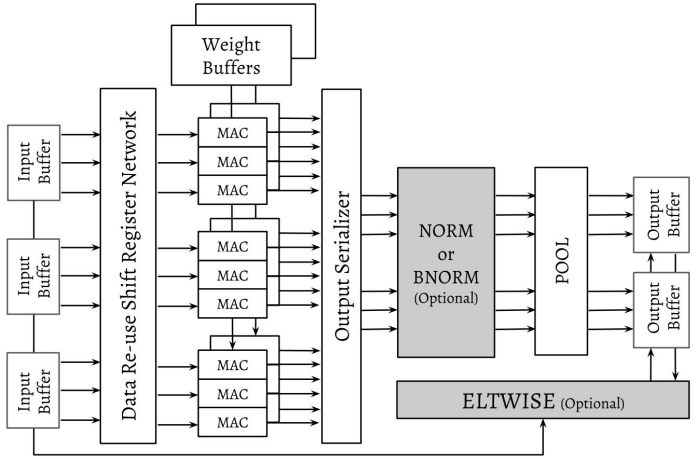
3 CNNs on 2 accelerator architectures

1.3x Higher Perf. **15%** Less Logic Util.

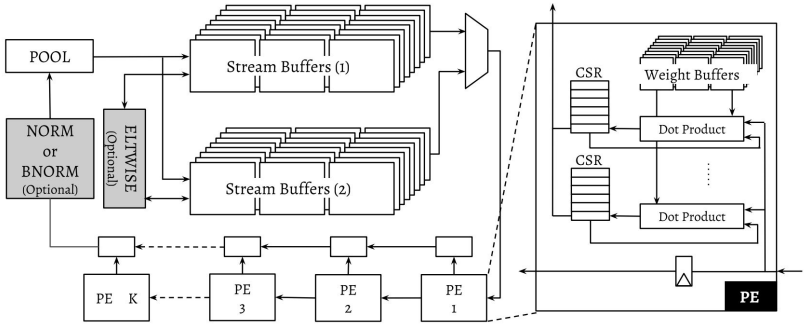
in case of 8-bit precision

1.6x Higher Perf. **30%** Less Logic Util.

in case of 4-bit precision



ASU CNN Accelerator

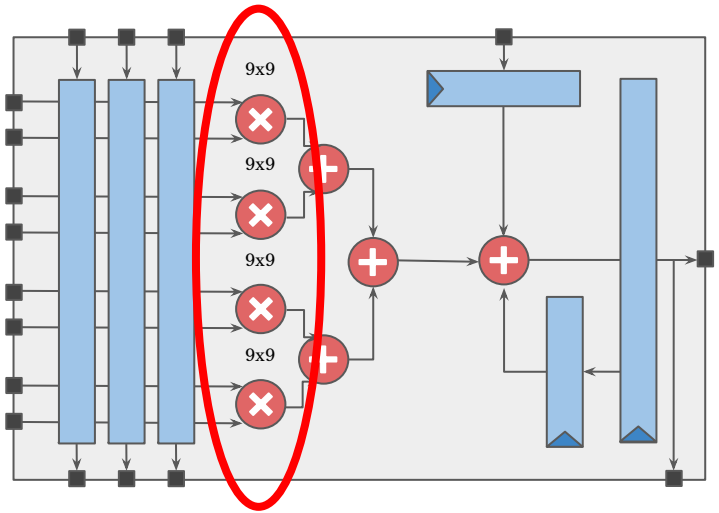


Intel DLA

A. Boutros et al., "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," FPL18

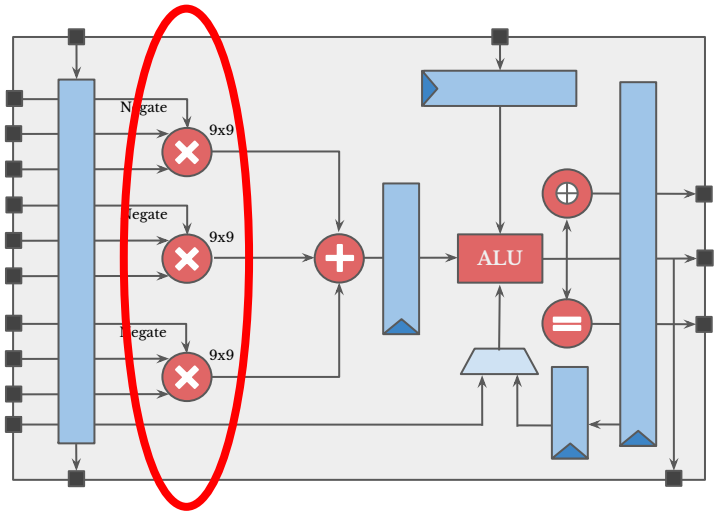
Industrial Adoption

4x9b, 2x18b, 1x27b



Intel Agilex DSP

27bx24b, 3x9b

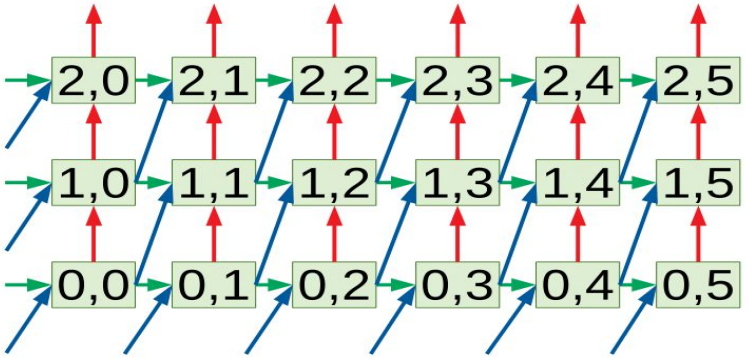


Xilinx Versal DSP

… both support similar INT9 mode!

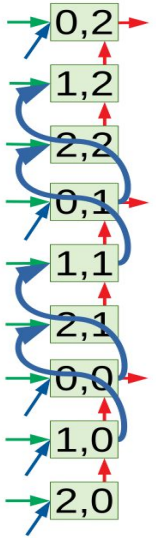
Going beyond Precision

- **Special dedicated links**
  - Semi-2D DSP-to-DSP interconnect



2D Systolic Array

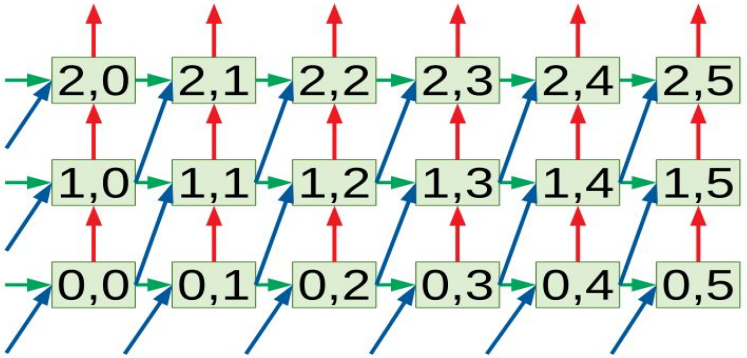
Transform into columnar structure



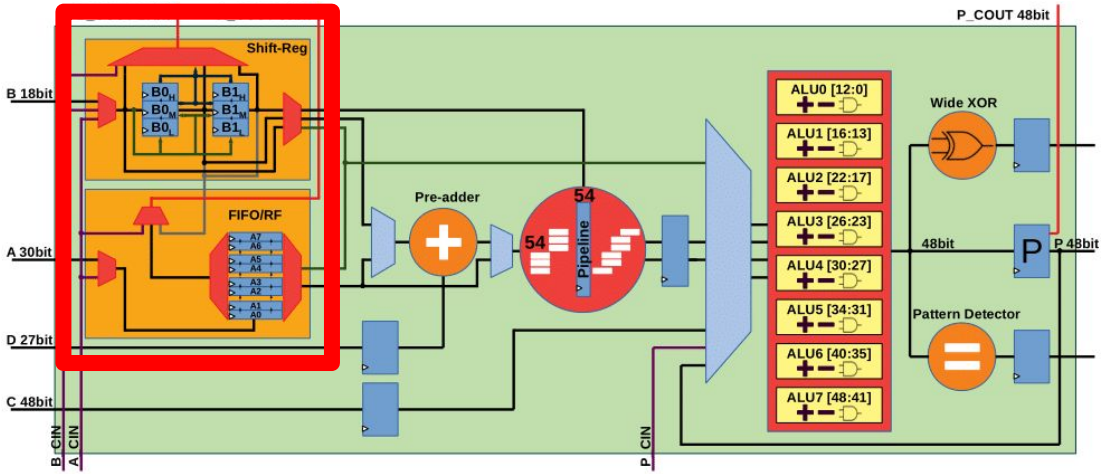
S. Rasoulinezhad et al., "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," FCCM19

Going beyond Precision

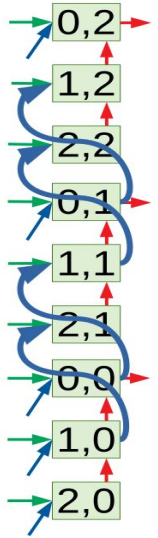
- **Special dedicated links**
  - Semi-2D DSP-to-DSP interconnect
- **Better localization of data**
  - Embedded RF to reuse data



2D Systolic Array

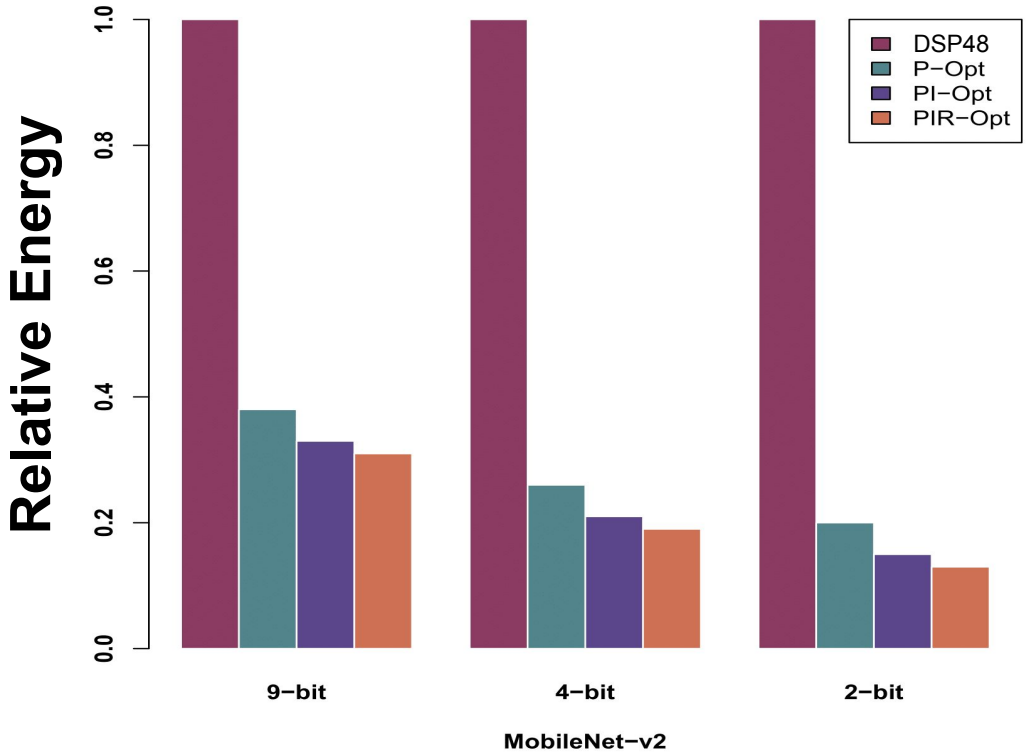


Transform into columnar structure



S. Rasoulinezhad et al., "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," FCCM19

Huge Energy Savings



S. Rasoulinezhad et al., "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," FCCM19

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

New DL-optimized FPGA Blocks

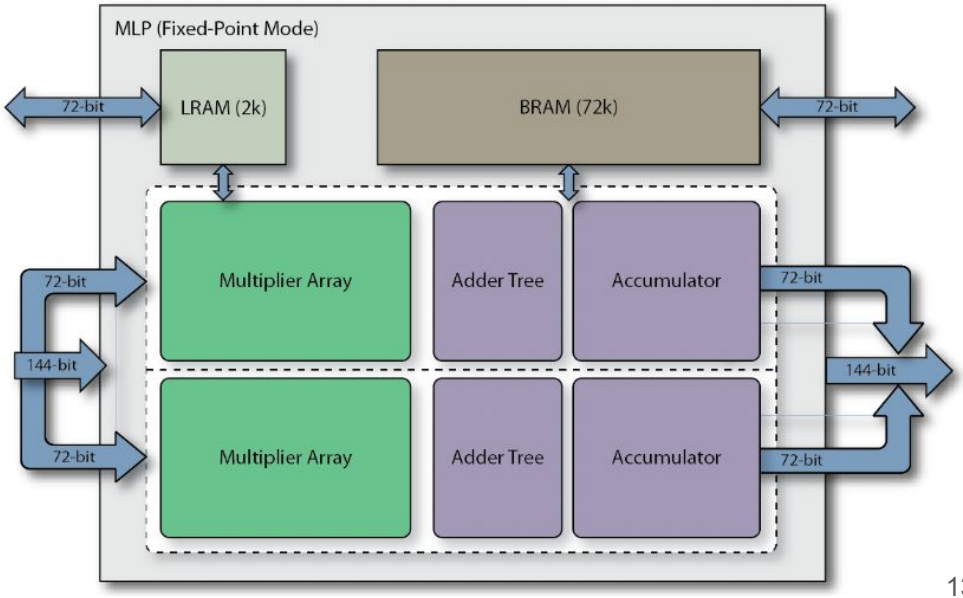
New DL-optimized FPGA Blocks

DL-Specific Fabric Blocks: Commercial Tensor Blocks (Vaughn)

Achronix Speedster MLP

- Small & medium int & fp formats
  - Decomposable multipliers
- Input limit: provide **extra inputs from closely coupled BRAM**
- Enables 16x 8-bit multiples
- Or 32x 4-bit multiplies

	Formats
Integer	int3, int4, int6, int8, int16
Floating Point	fp3, fp4, fp6, fp8, fp16, bfloat16, fp24.



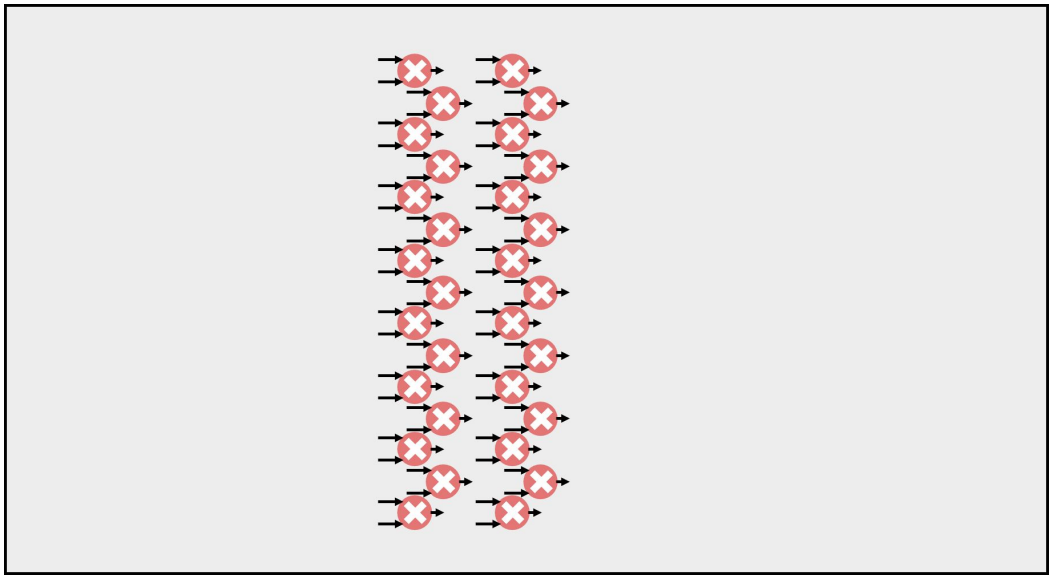
Intel Stratix 10 NX:Tensor Block

30x int8 multipliers instead of **2x int18** multipliers

- Or **60x int4** multipliers
- Also block floating point bfp16 and bfp12
  (~int8/int4 with 10-element shared exponent)

Focusing on this mode

30x int8

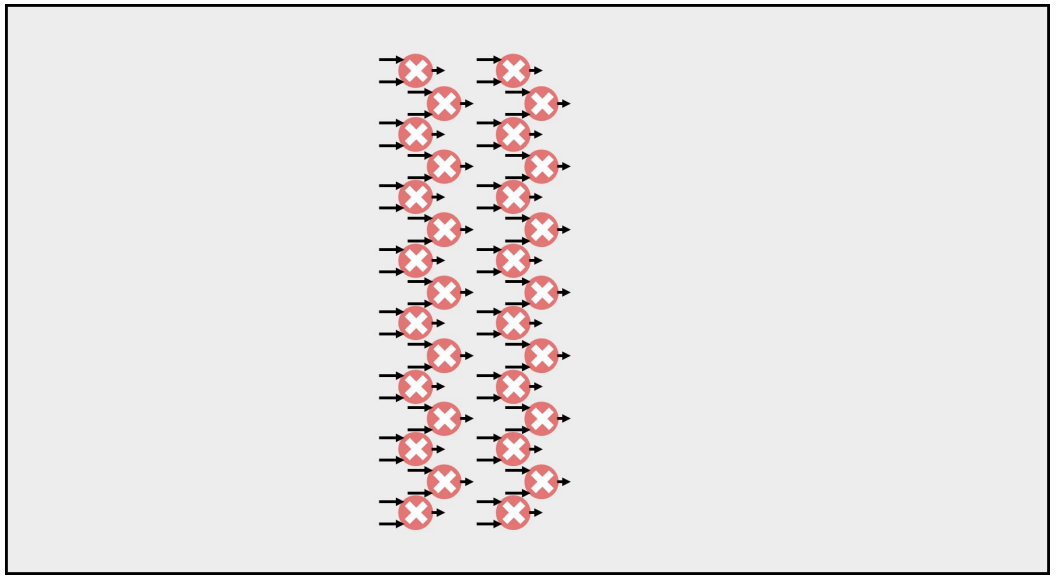


Tensor Block int8

30x int8 multipliers instead of **2x int18** multipliers

480 inputs

DSP block:
96 inputs



480 outputs

DSP block:
72 outputs

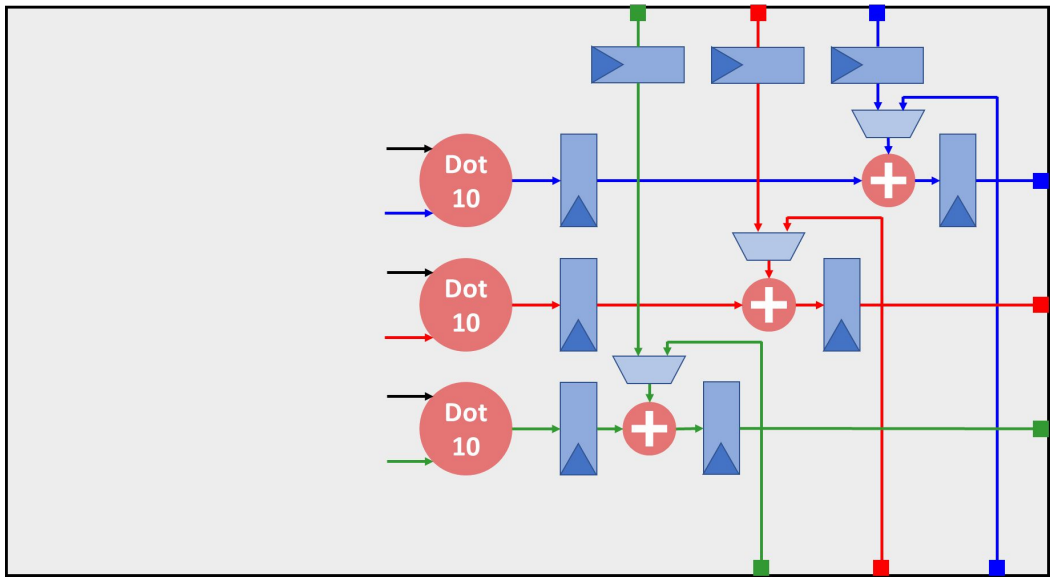
Tensor Block int8

30x int8 multipliers instead of **2x int18** multipliers

Limit Outputs: Arrange multipliers as **3x dot-10** engines + accumulators

480 inputs

Dedicated Cascades: Cheap



72 outputs

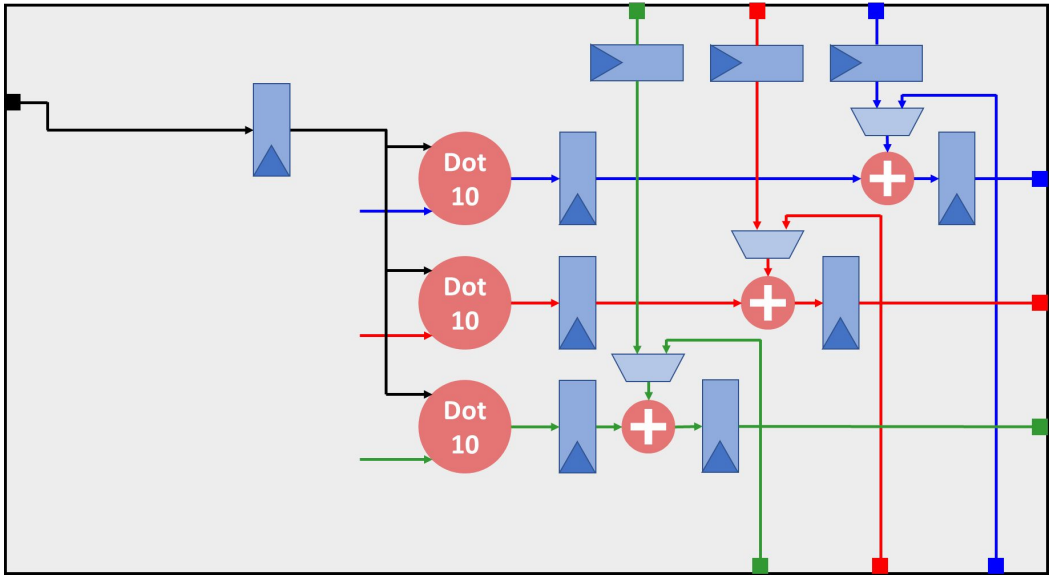
Tensor Block int8

30x int8 multipliers instead of **2x int18** multipliers

Limit Outputs: Arrange multipliers as **3x dot-10** engines + accumulators

Limit Inputs: **Broadcast** one set of inputs to all dot-10 engines

320 inputs



72 outputs

Tensor Block int8

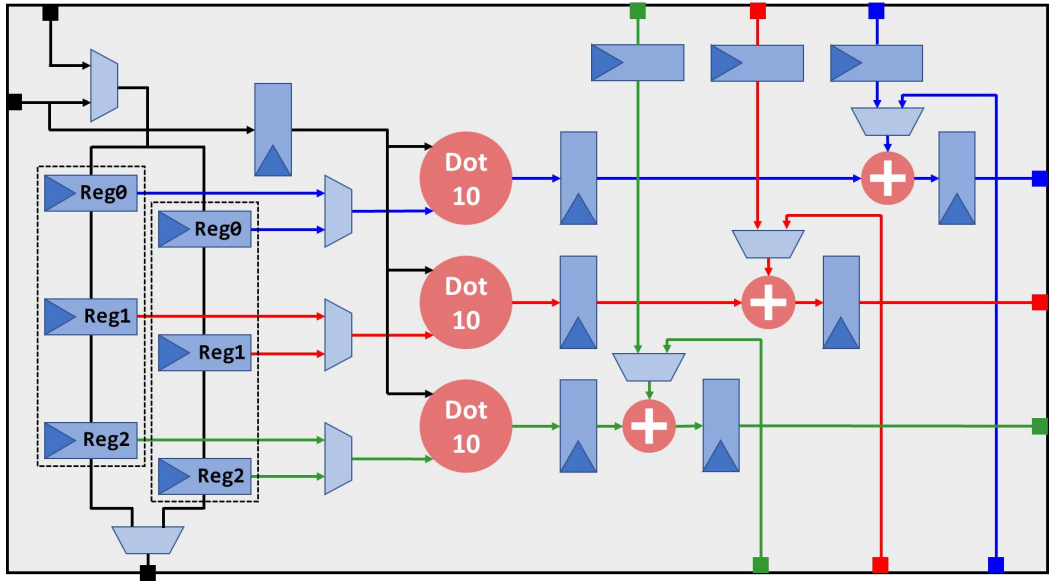
30x int8 multipliers instead of **2x int18** multipliers

Limit Outputs: Arrange multipliers as **3x dot-10** engines + accumulators

Limit Inputs: **Broadcast** one set of inputs to all dot-10 engines

Limit Inputs: **Ping-pong input reuse chain** loaded from the block above

80 inputs



72 outputs

Tensor Block int8

15x peak int8 TOPS but significant I/O constraints

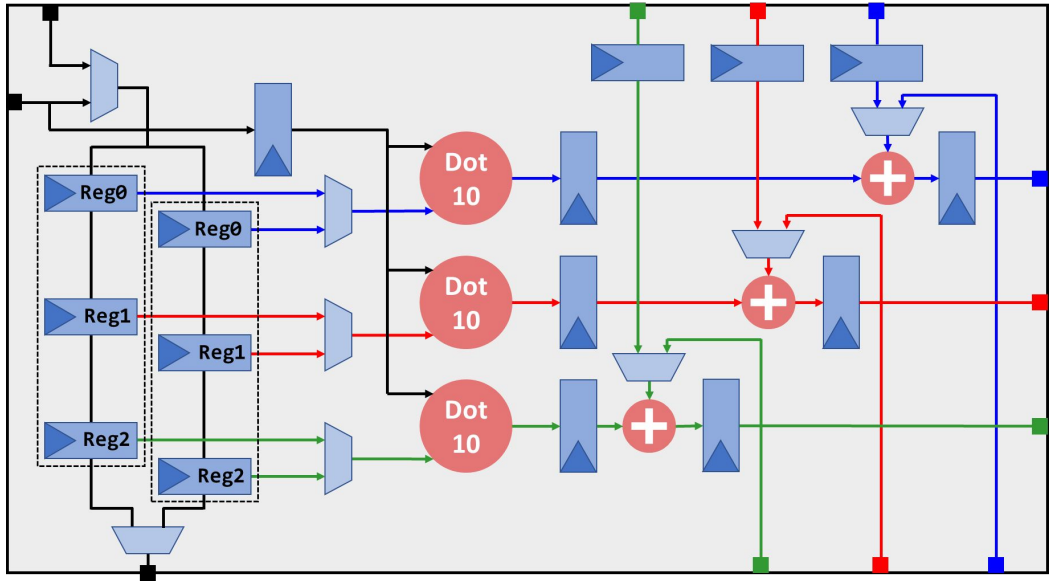
30x int8 multipliers instead of **2x int18** multipliers

Limit Outputs: Arrange multipliers as **3x dot-10** engines + accumulators

Limit Inputs: **Broadcast** one set of inputs to all dot-10 engines

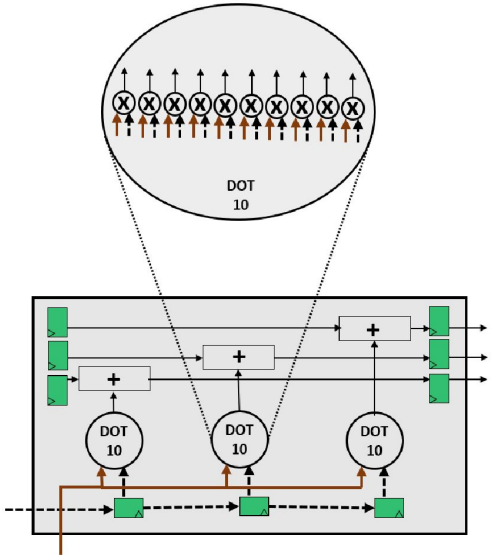
Limit Inputs: **Ping-pong input reuse chain** loaded from the block above

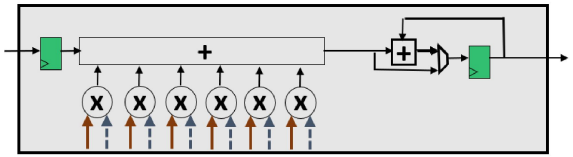
80 inputs

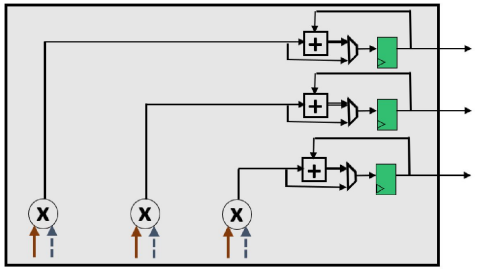


72 outputs

Tensor Block: 3 Modes to Give Interconnect Options





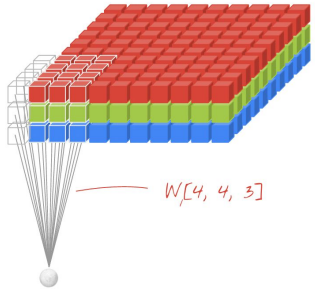


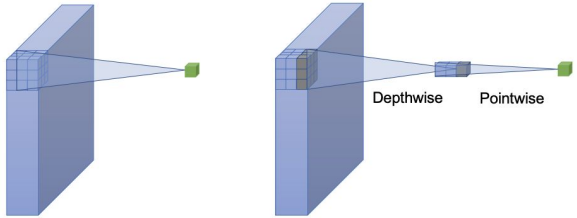
Tensor Mode: **30x** int8
Broadcast & preload inputs
Three dot-10

Vector Mode: **6x** int8
No input restriction
one dot-6

Scalar Mode: **3x** int8
No input or output restrictions

Can CNNs Exploit S10 Tensor Blocks? → HPIPE





MobileNetV1 - V3: Multiple convolution types

Yes!

Need all modes

Dense weights

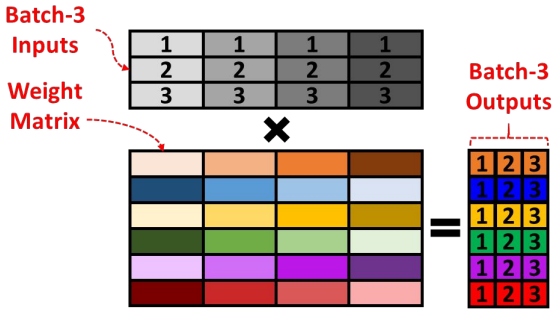
- **Tensor mode**, preload activations, broadcast weights
  - Except depthwise conv: **scalar mode**
- **5x speedup** vs. DSP blocks
- Less than 15x peak, but well above any other reported results

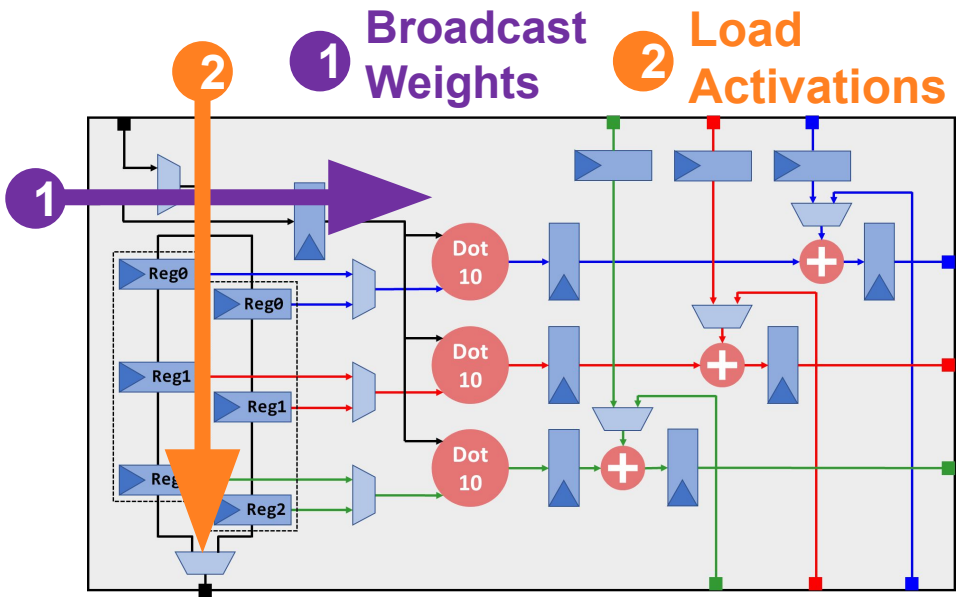
Sparse weights

- **Vector mode**
  - **1.9x speedup** vs. DSP blocks

M. Stan, et al, “HPIPE NX: Boosting CNN Inference Acceleration Performance with AI-Optimized FPGAs,” *FPT*, Dec. 2022.

Can RNNs & LSTMs Exploit S10 Tensor Blocks? → NPU





Yes!

Need some batching

- Preload activations, in batches of 3
- Broadcast weights
- **3.5x speedup** vs. DSP blocks

A. Boutros, et al "Beyond Peak Performance: Comparing The Real Performance of AI-Optimized FPGAs and GPUs," *FPT*, 2020.

Tensor Block Takeaway

Large gains possible with a more “coarse-grained” block

Have to minimize/re-use/restructure interconnect

- Not trivial to use
- Can’t just recompile your RTL/HLS → **restructure your computation**

New DL-optimized FPGA Blocks

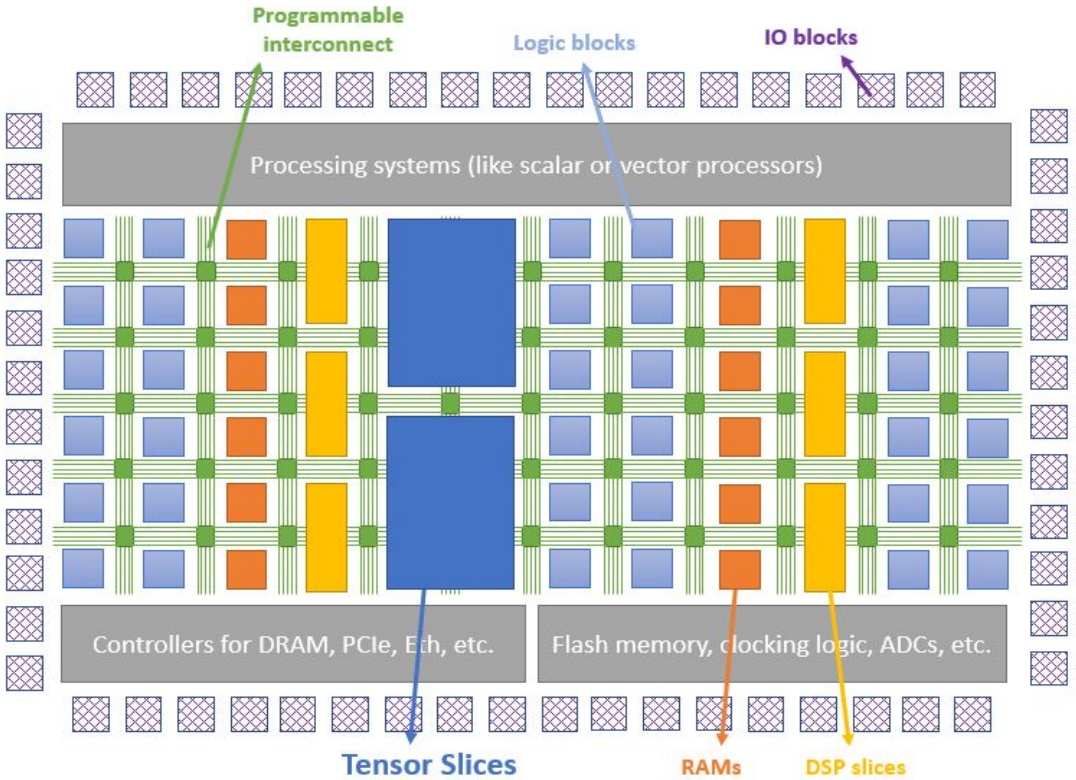
DL-Specific Fabric Blocks: Academic Tensor Blocks (Vaughn)

Tensor Slices

Tensor/matrix operations are at the heart of Deep Learning

Matrix multiplier using Logic Blocks and DSP Slices is inefficient (~4x slower and ~10x larger than an ASIC)

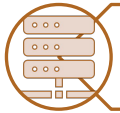
Can we perform matrix multiplication on an FPGA more efficiently?



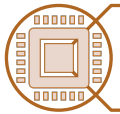
Arora et al. “Tensor Slices to the Rescue: Supercharging ML Acceleration on FPGAs”, ISFPGA 2021
Arora *et al.*, “Tensor Slices: FPGA Building Blocks For The Deep Learning Era,” ACM TRETS 2022

Why add Tensor Slices?

Compute density. Pack more compute in the same area footprint.

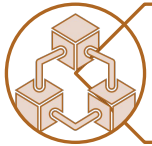


Reduce routing wire usage

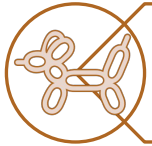


Reduce area and increase frequency for implementing ML designs

Coarse grained. Faster compilation.

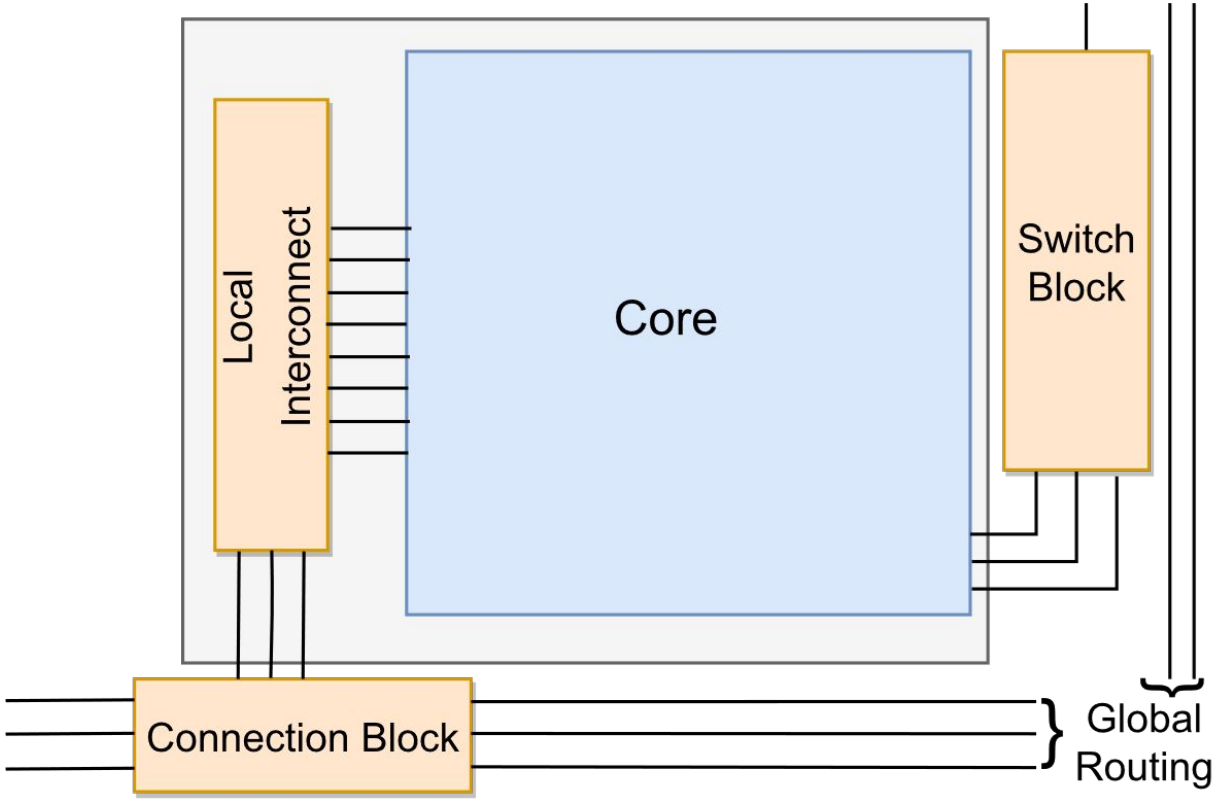


Update tools, provide libraries, etc.

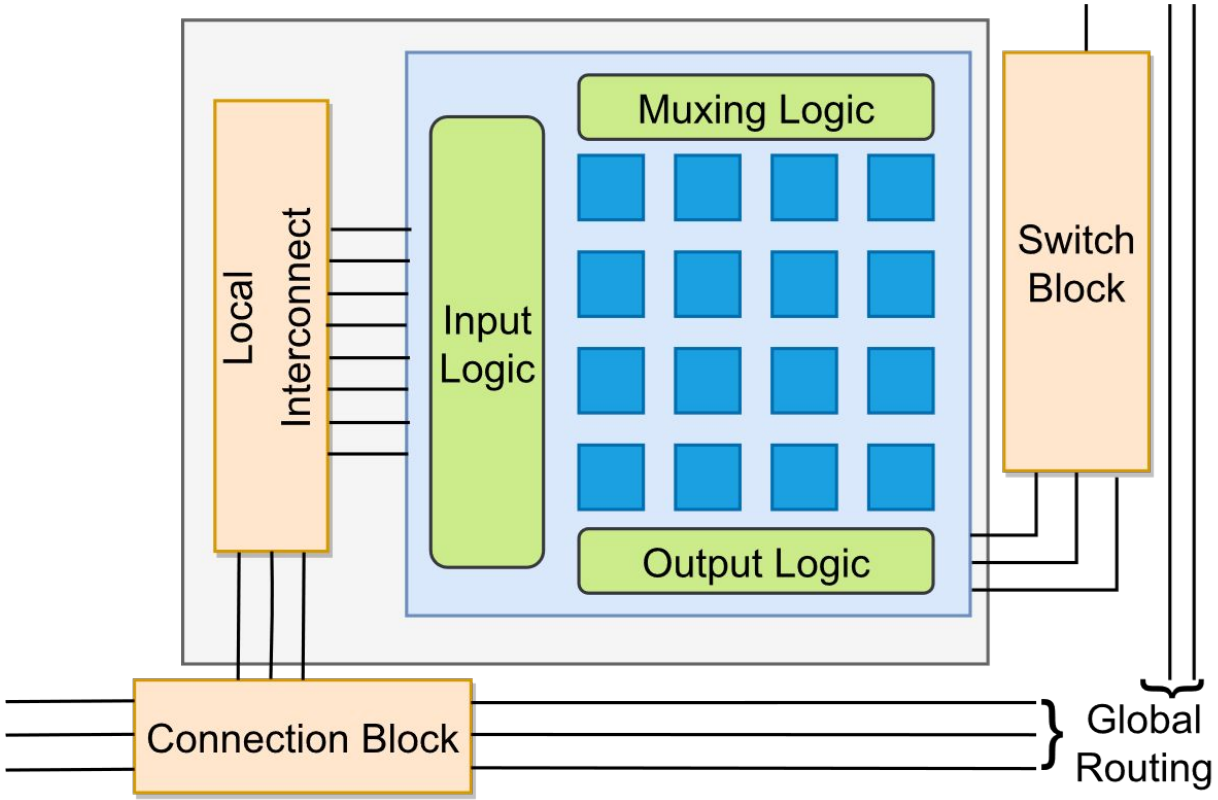


Less generic/flexible than a typical FPGA. But worth it because of so many ML applications.

Tensor Slice: High level diagram



Tensor Slice: High level diagram



Tensor Slice: Design Space

Architecture
- Systolic
- Dot-product based

How to lay them out in the FPGA
- Along columns
- Grouped together

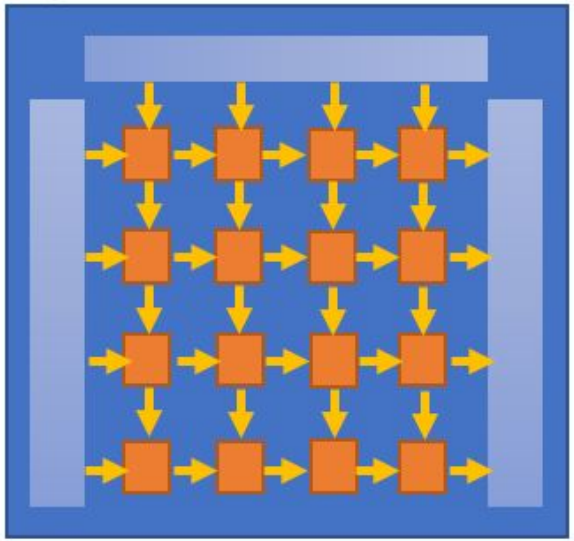
Operations to support
- Matrix matrix multiplication
- Matrix vector multiplication
- Element wise operations

Size (Number of PEs)
- 2x2, 4x4, 8x8, 16x16
- Something else

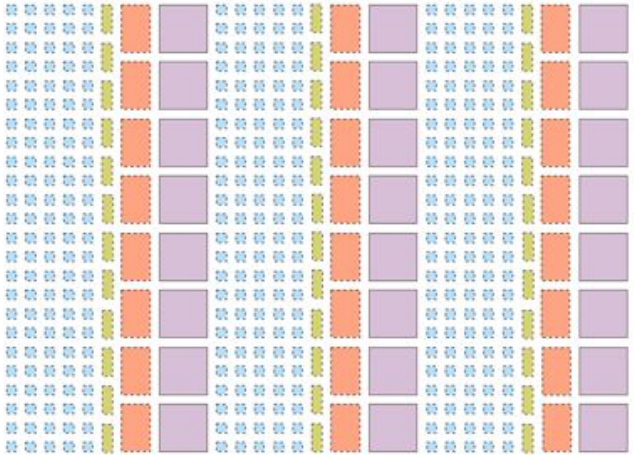
FPGA area to spend on them
- 5%, 10%, 20%, 30%,…
- Replace all DSP Slices with Tensor Slices

Precisions to support
- Integer (int4, int8, int16)
- Floating point (bf16, fp16)

Tensor Slice: Architecture and Layout

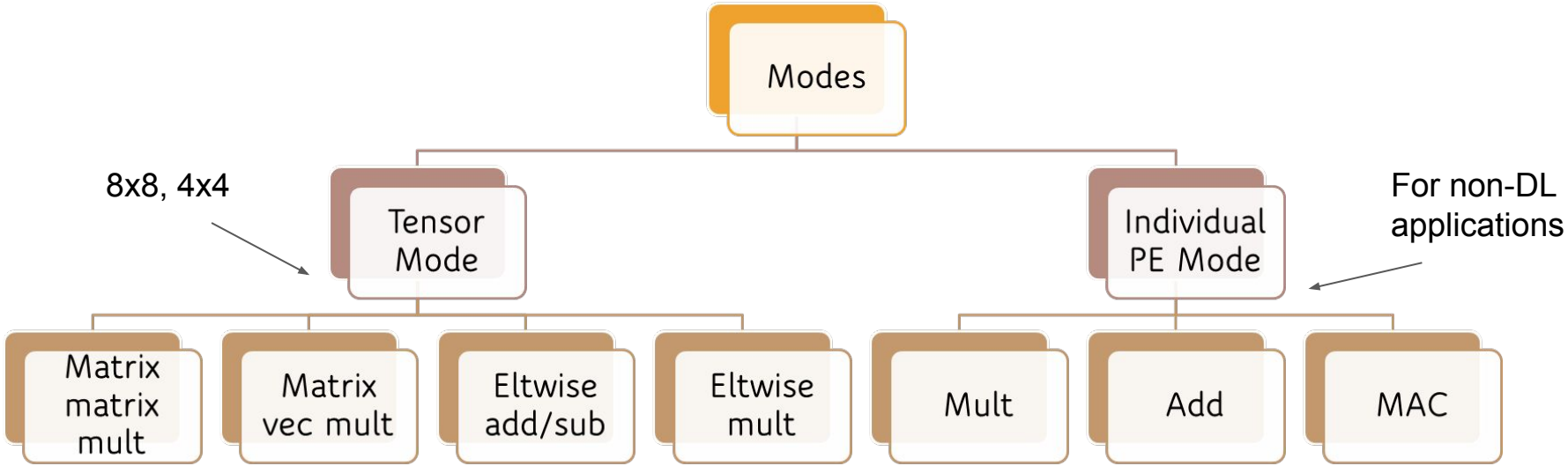


Systolic Architecture



Arranged in columns

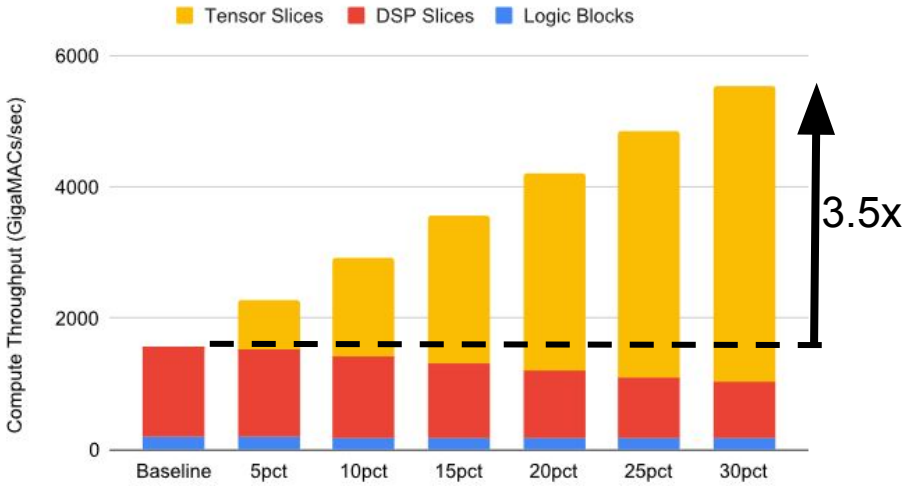
Tensor Slices: Modes, Sizes, Precision



Precisions: int8, int16, fp16, bf16

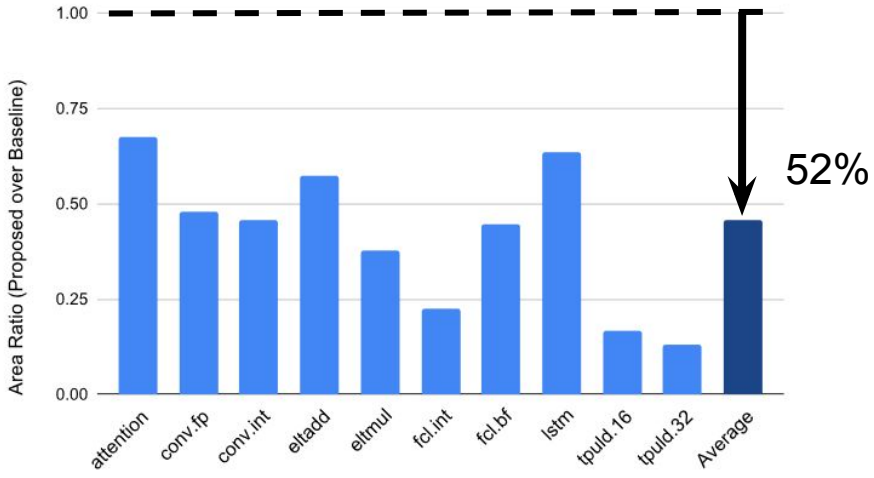
Tensor Slice: Compute Throughput and Area

Peak Compute vs. Tensor Slice Die Area



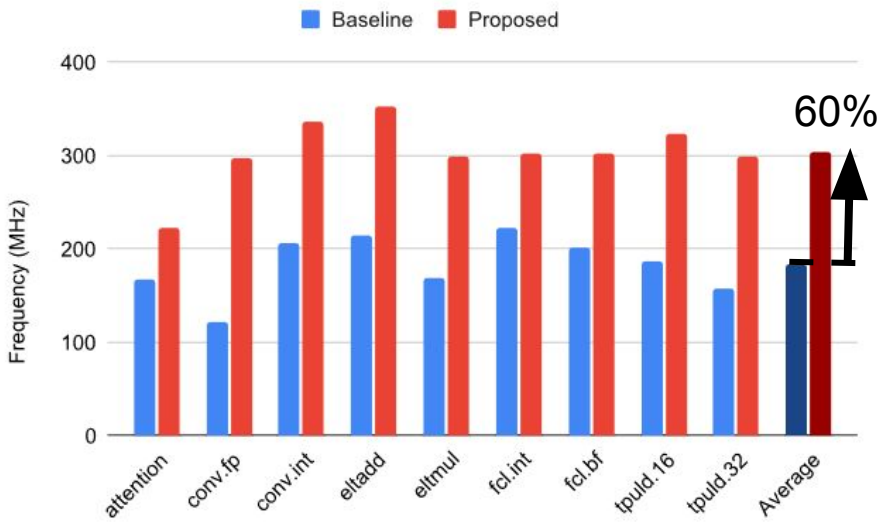
Die Area Devoted to Tensor Slices

Used Device Area

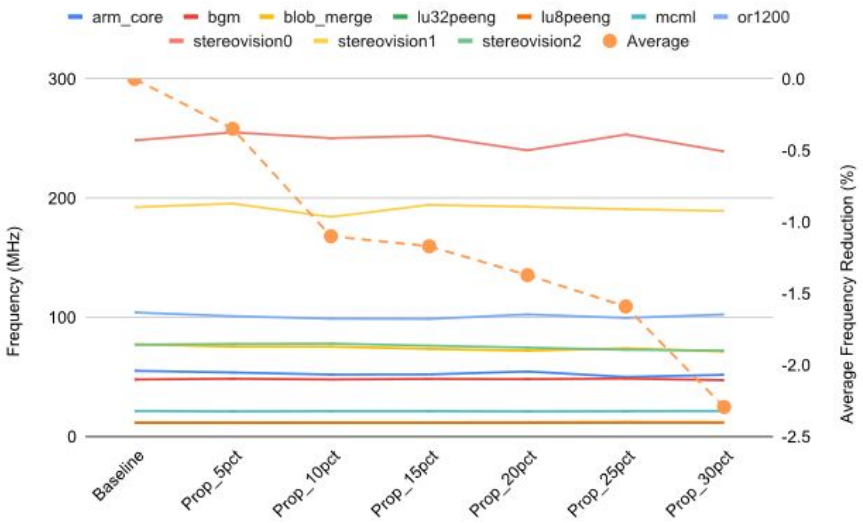


Koios DL benchmark

Tensor Slice: Frequency

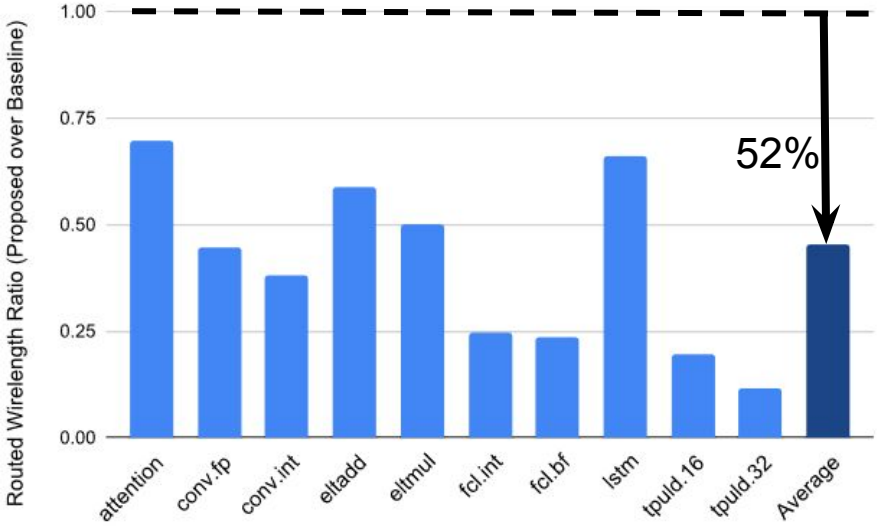


(a) DL benchmarks

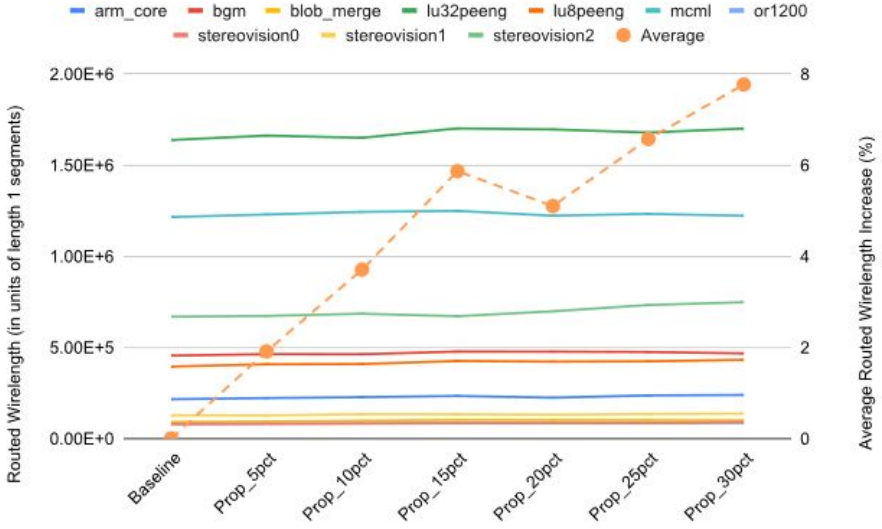


(b) Non-DL benchmarks

Tensor Slice: Interconnect

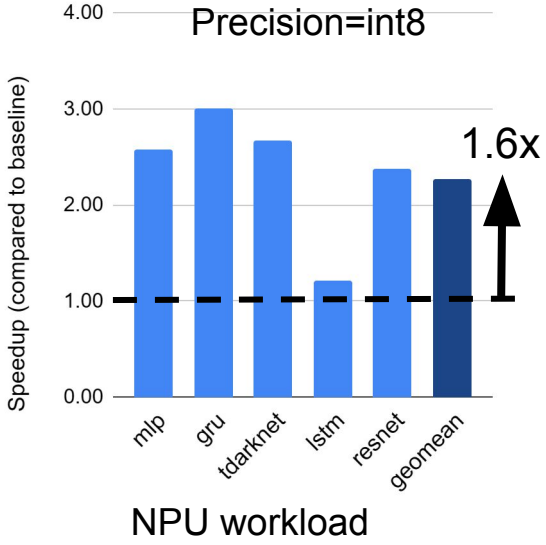


(a) DL benchmarks

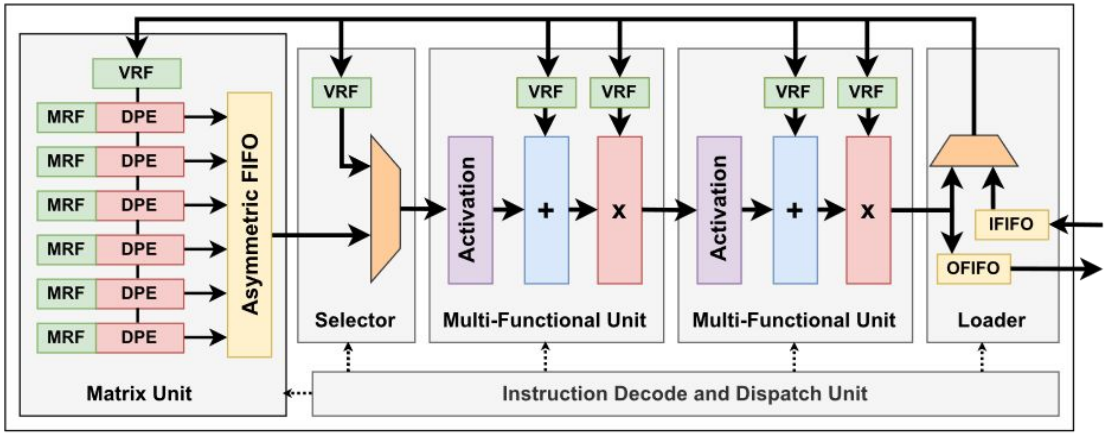


(b) Non-DL benchmarks

Tensor Slice: Deep Neural Network Overlay



NPU workload



NPU-like Accelerator

Takeaway: An FPGA with Tensor Slices can achieve significant speedups on realistic DL networks, compared to commercial FPGA.

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

Schedule


Break (10 mins)
Break (10 mins)
9:05-9:30
9:30-10:00
10:10-10:40
10:40-12:00
FPGA Architecture Through a DL Lens
FPGA DL Accelerators & Architectures
Traditional FPGA Blocks for DL
New DL-optimized FPGA Blocks

New DL-optimized FPGA Blocks

DL-Specific Fabric Blocks: Compute RAMs (Vaughn)

Compute-In-Memory

Also called Processing-In-Memory (PIM)

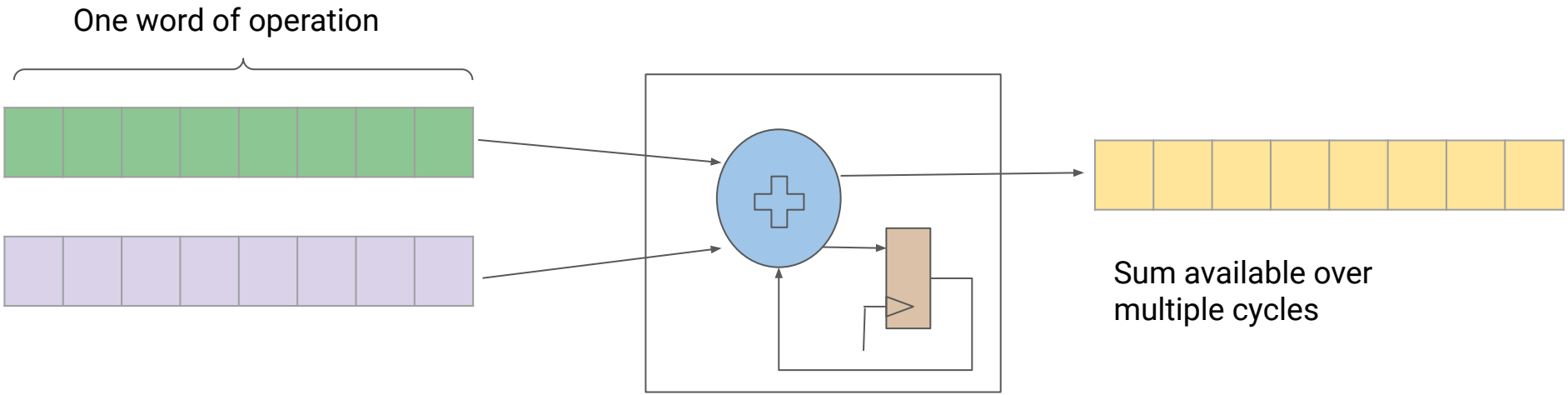
Bring computation closer to the storage

Reduces data movement, hence reducing energy and latency

Many flavors have been proposed:

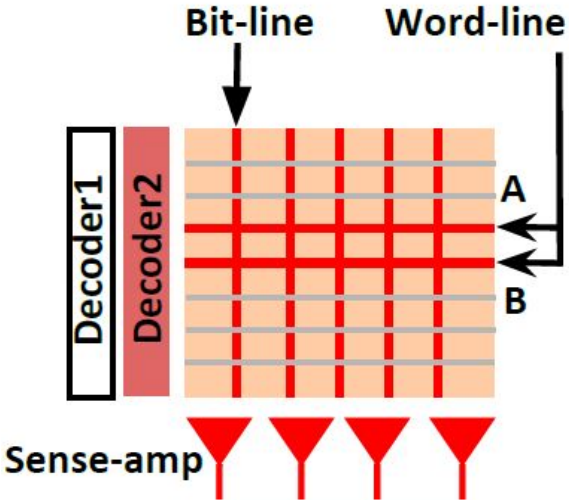
- ReRAM based
- **SRAM based** ← Add compute to the Block RAMs on FPGA
- DRAM based
- 3D stacking based

Bit-Serial Computing



Precision agnostic! Great for DL!

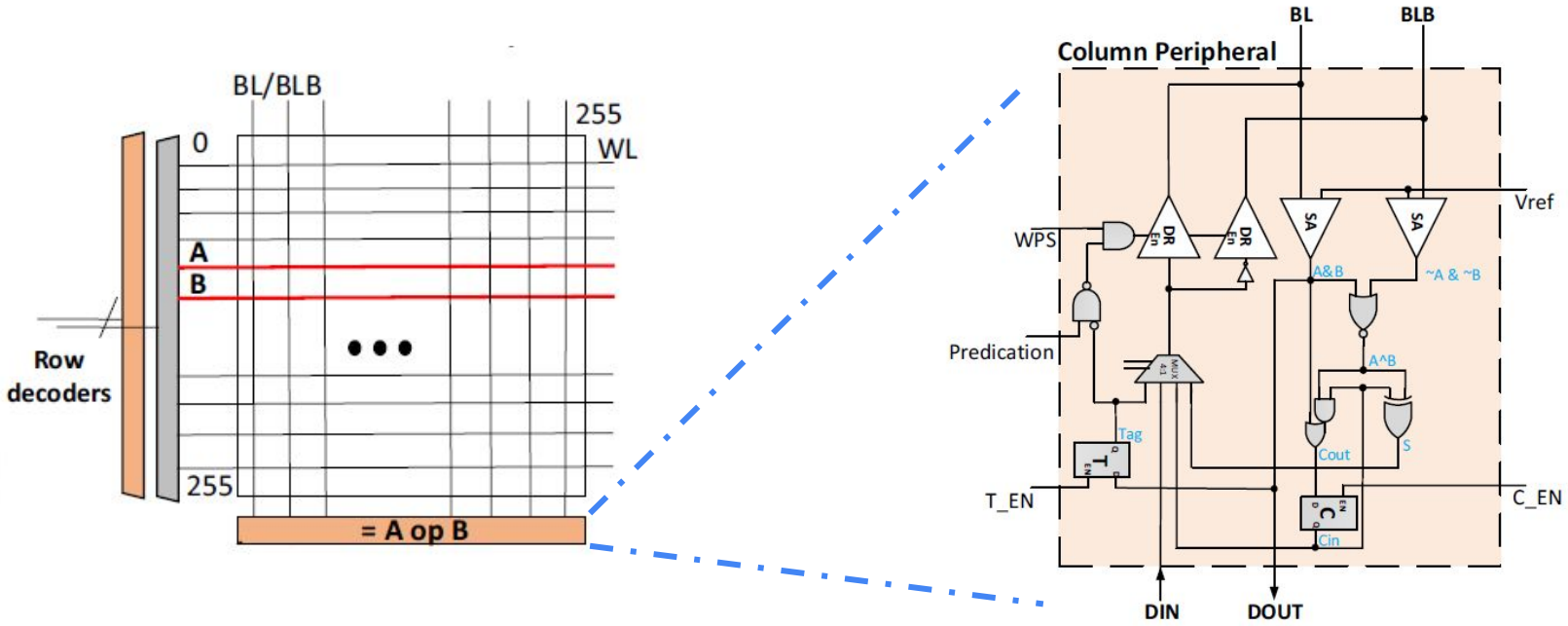
Bit-Line Computing



A AND B

Logic-in-Memory [*JSSC'16*]
Compute Caches [*HPCA'17*]

Adding Processing Elements inside a SRAM

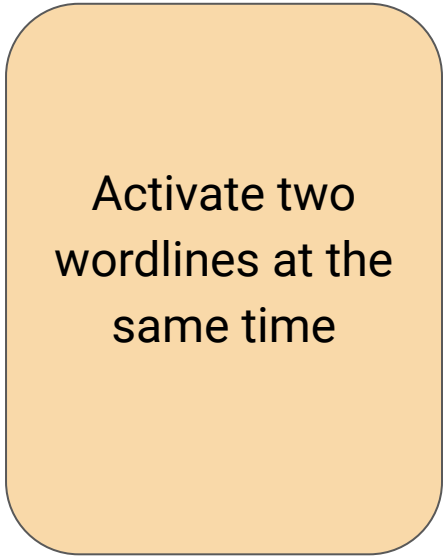


Neural Cache [*ISCA'18*]

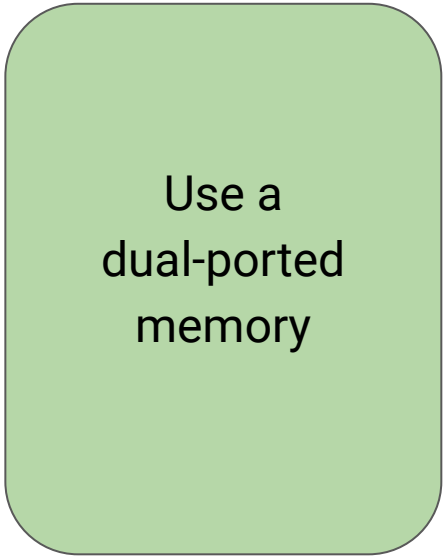
What’s the main principle here?

Get two bits (one from each operand), add them, write result back to the RAM

Two approaches:

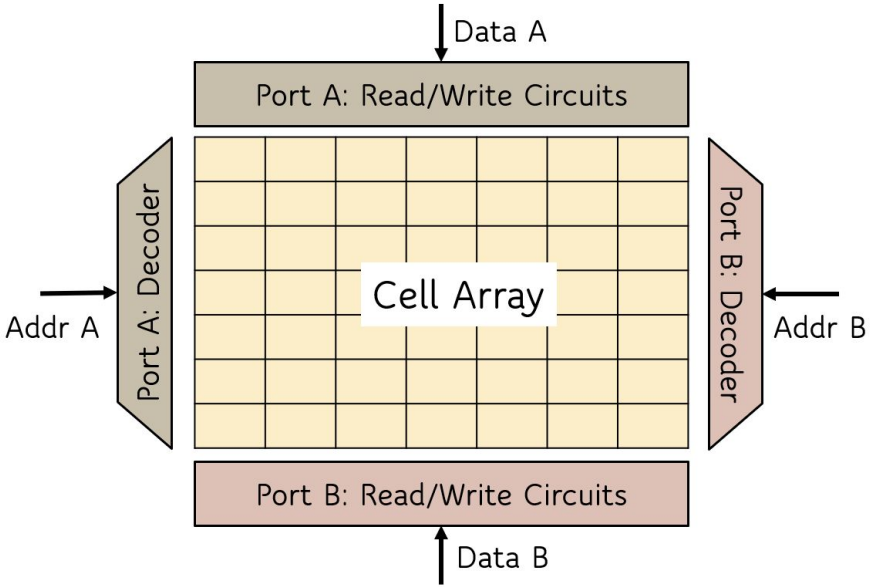


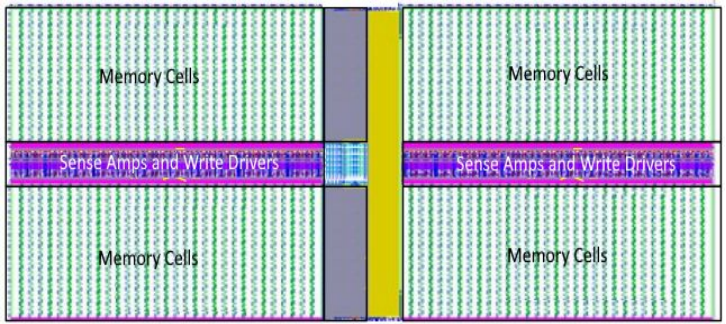
Robustness 😟



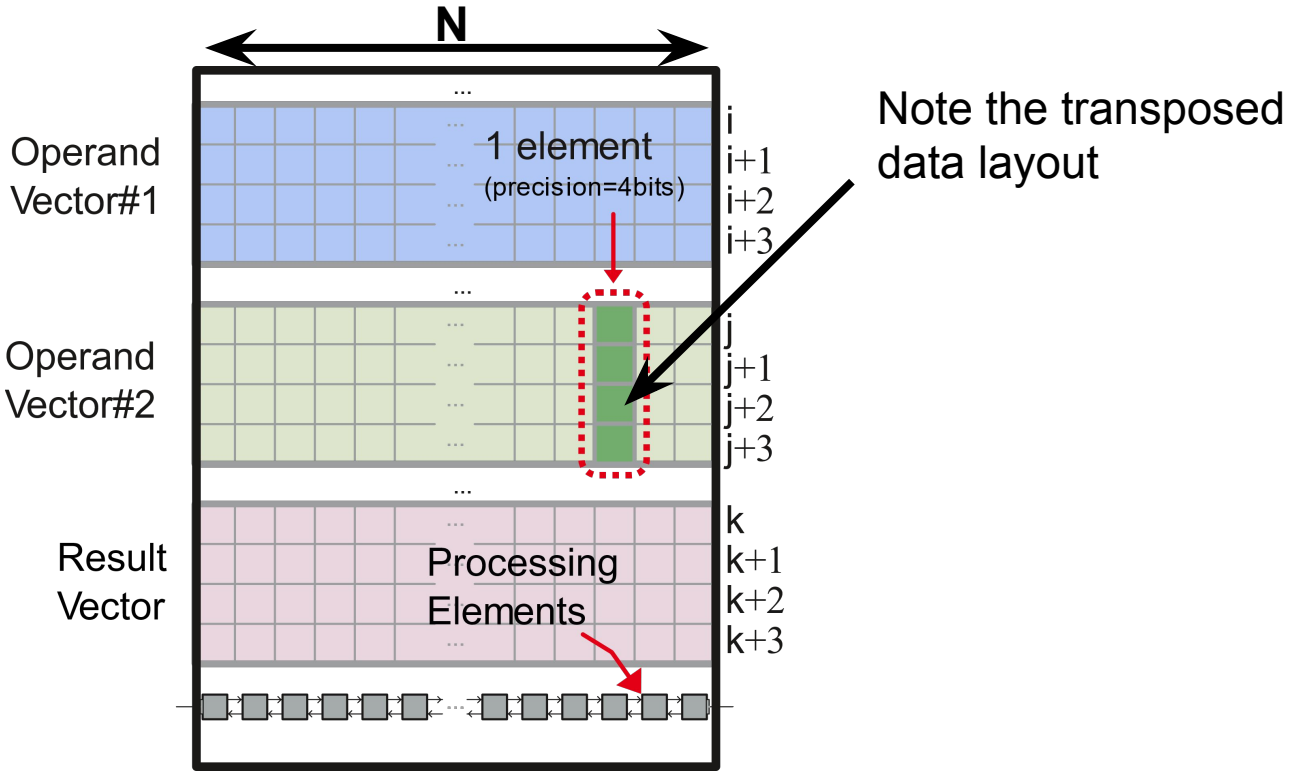
Area😟

Block RAMs on FPGAs are already dual ported! :)

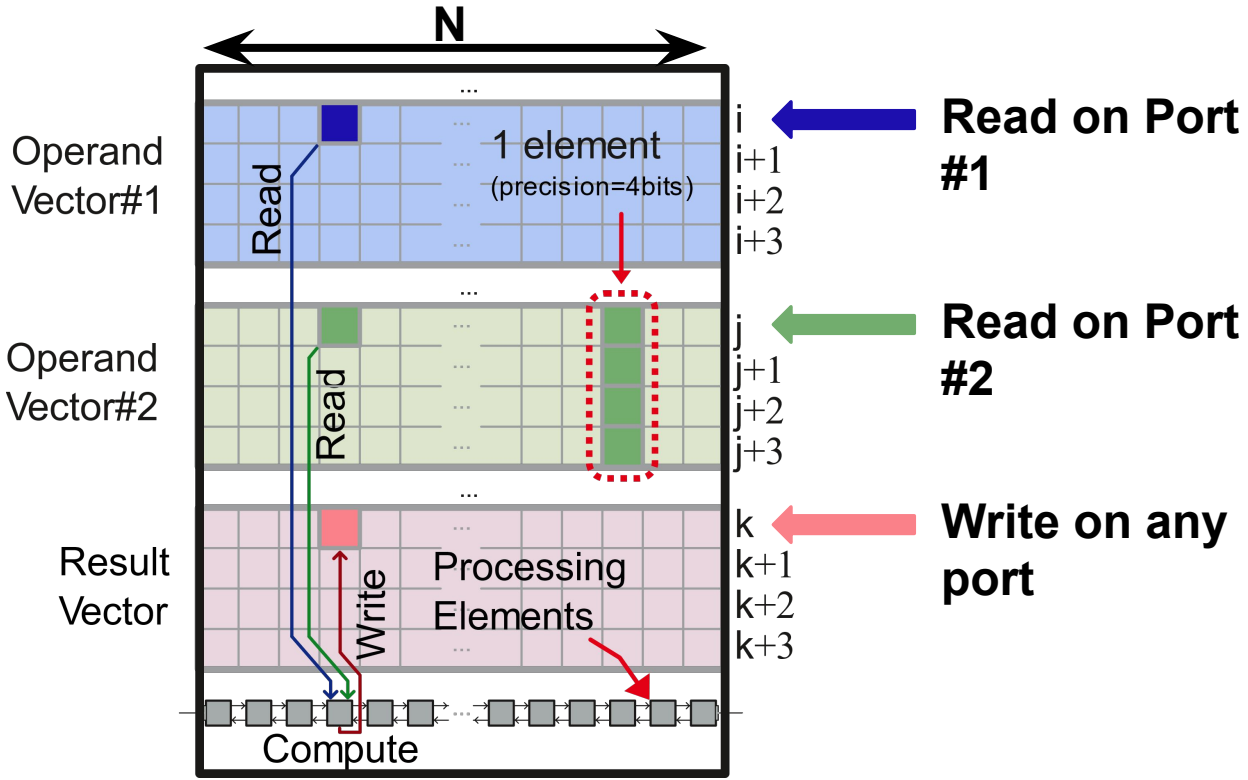




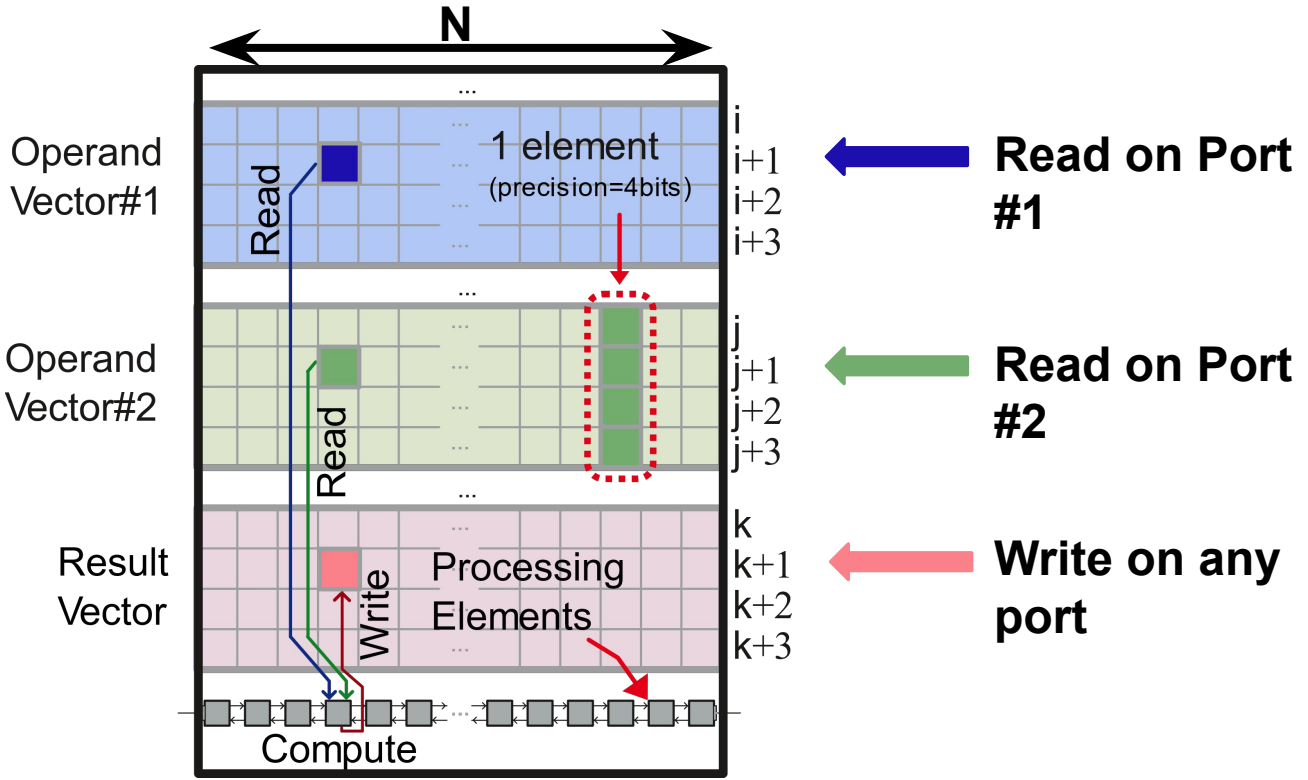
High Level Operation



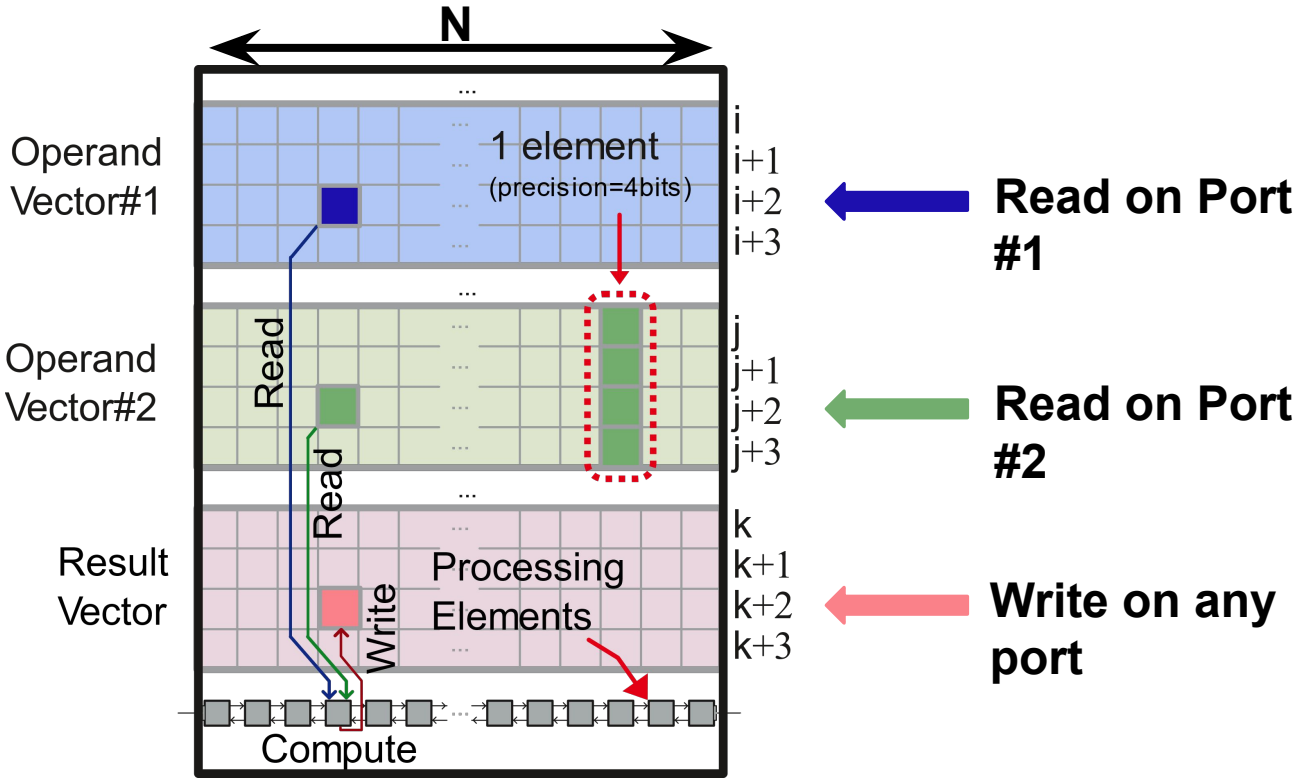
High Level Operation



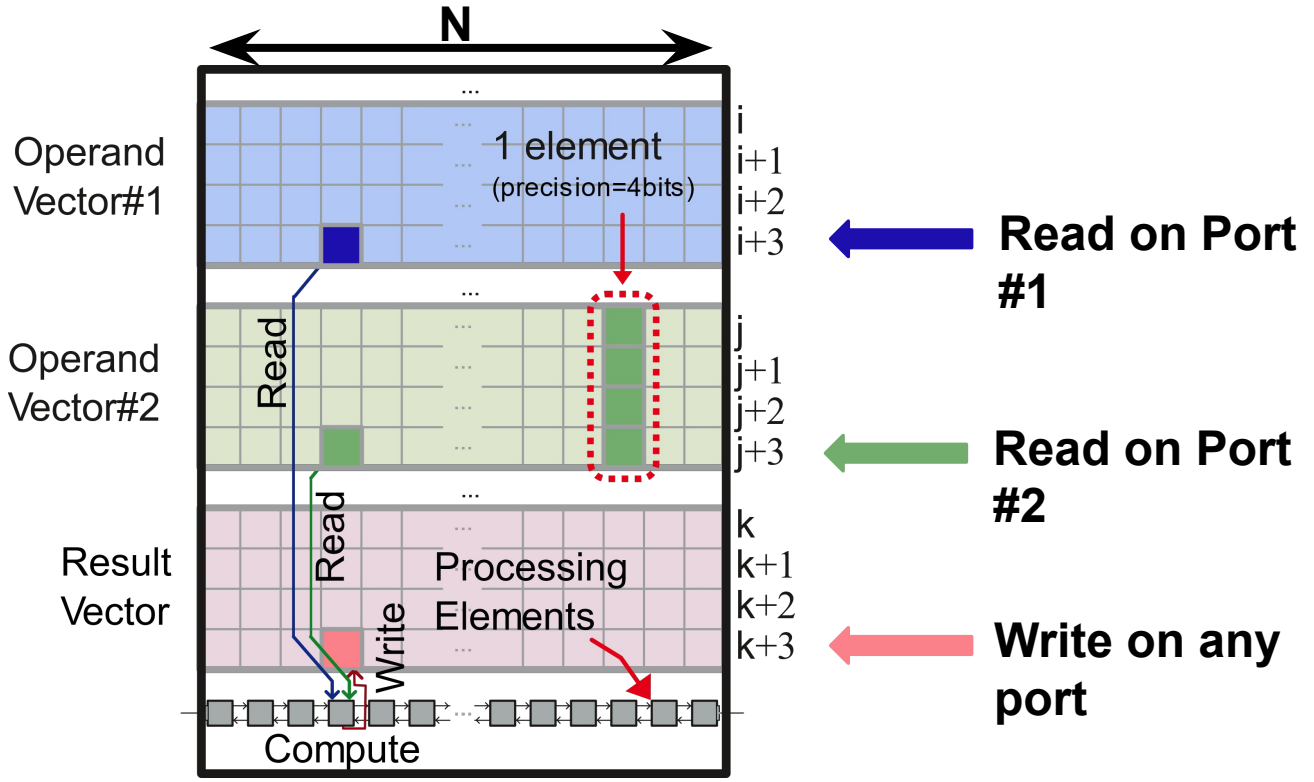
CoMeFa RAMs: High Level Operation



CoMeFa RAMs: High Level Operation



CoMeFa RAMs: High Level Operation



Design Space

Getting 2 operands in 1 cycle

- Activating two wordlines
- Use dual ported memory

Architecture of a PE

- Operations (add, logical, etc)
- Predication
- Configurability

Number of PEs and SAs

- PEs = SAs = Number of bitlines
- PEs = SAs = Number of data lines
- Or something in between

Loading and unloading data

- Transpose in soft logic
- Use RAM with transposable cells
- Transpose in DRAM controller

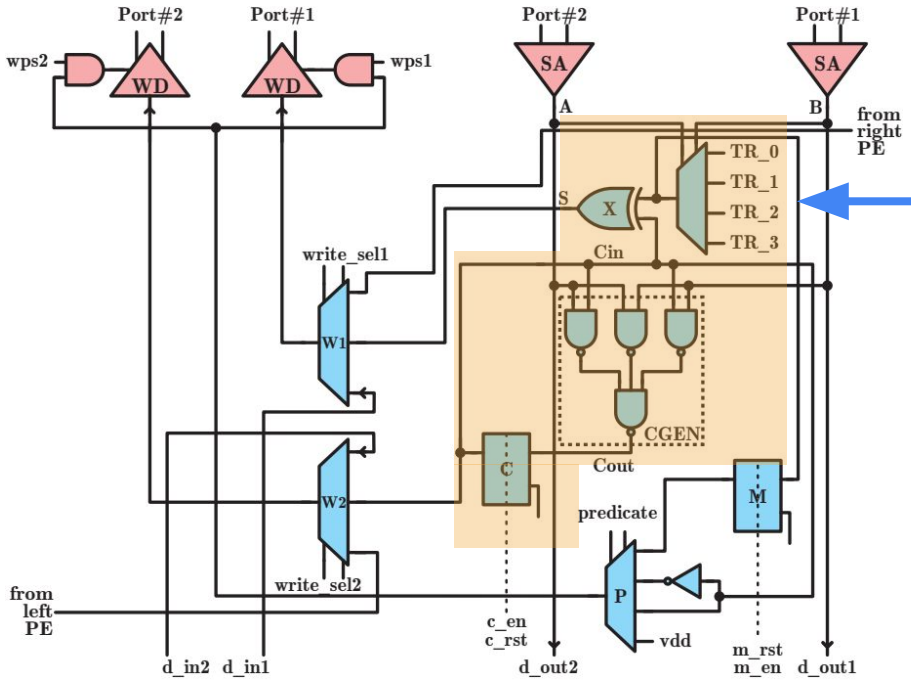
Programming the RAM

- Workload specific FSM
- Stored program

Signaling instructions to the RAM

- Write to a special address
- Repurpose a signal on the interface

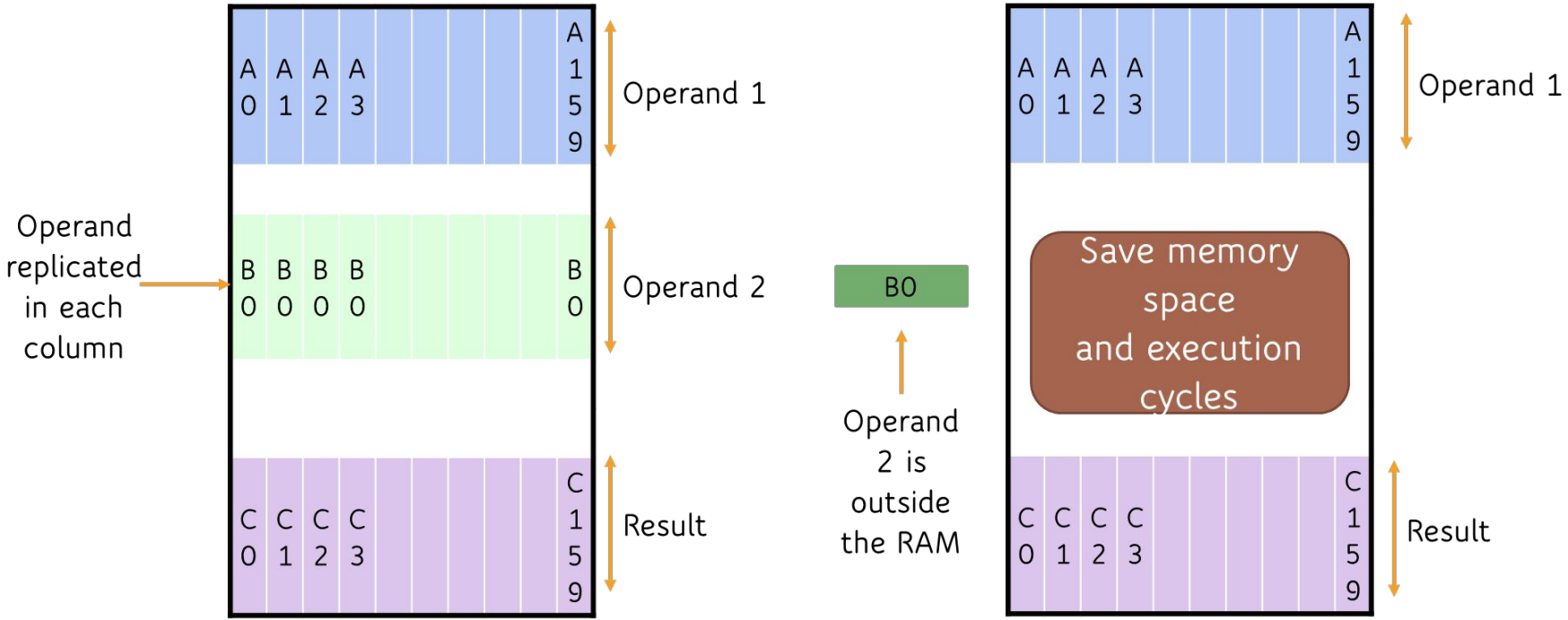
Processing Element



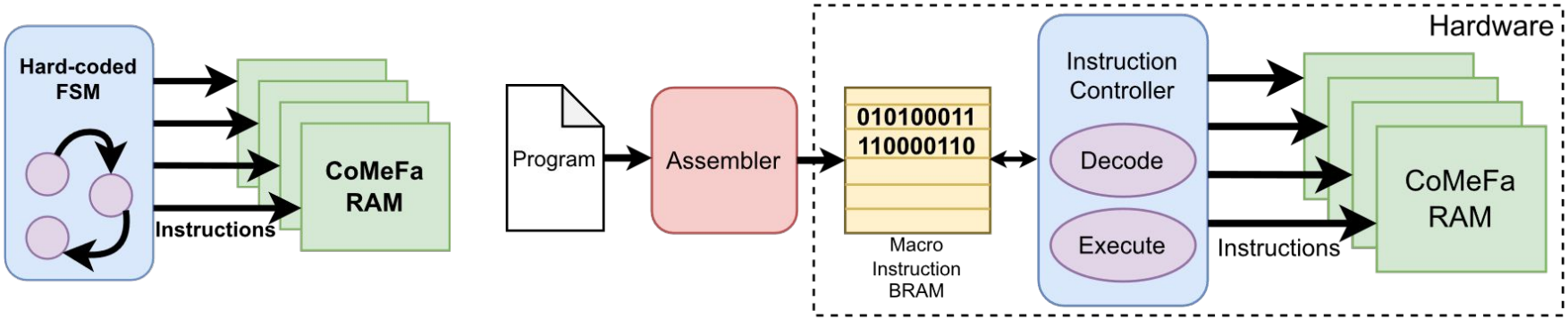
Mux to enable any function of 2 inputs

Basically a dynamic "LUT"

One Operand Outside RAM (OOOR) Operations



Programming the RAM



Harder to program
Higher performance

Easier to program
Slightly reduced performance

Observation

The BRAM has now become a SIMD processor

Observation: Enhanced “effective” bandwidth

Internal (physical) geometry of the BRAM is more squarish than the “external” (logical) geometry

Example: Consider 16 Kilobit RAM

Logical geometries available: 512x32, 1024x16, 2048x8, 4906x4,...

Physical geometry: 128x128 (128 word lines, 128 bitlines)

Why is this done?

Physical layout issues (pitch matching), ECC, Routing interface limitations

Can access more number of bits inside the RAM (assuming we have enough sense amps)

Overhead

PEs=SAs=# Bitlines
Activate two wordlines
Less-configurable PE

PEs=SAs=# Bitlines
Use dual-ported'ness
More-configurable PE

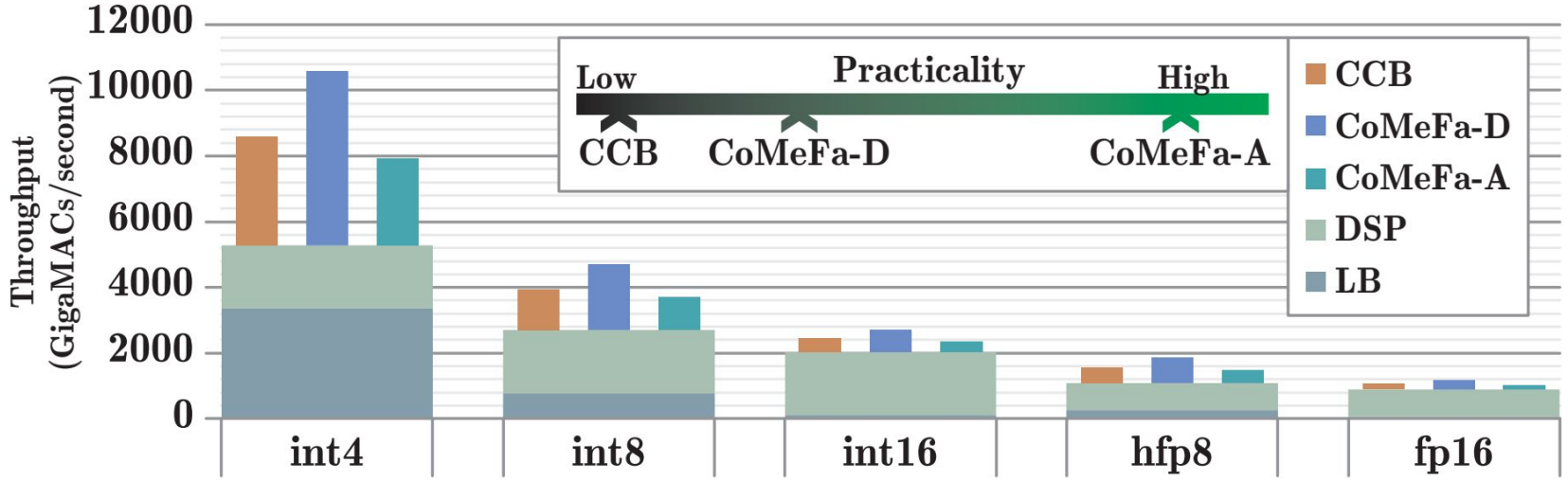
PEs=SAs=# Datalines
Re-use (cycle) SAs
Use dual-ported'ness
More-configurable PE

	CCB [1]	CoMeFa-D [2]	CoMeFa-A [2]
Clock duration	60%	25%	125%
Area (block)	16.8%	25.4%	8.1%
Area (chip)	2.5%	3.8%	1.2%

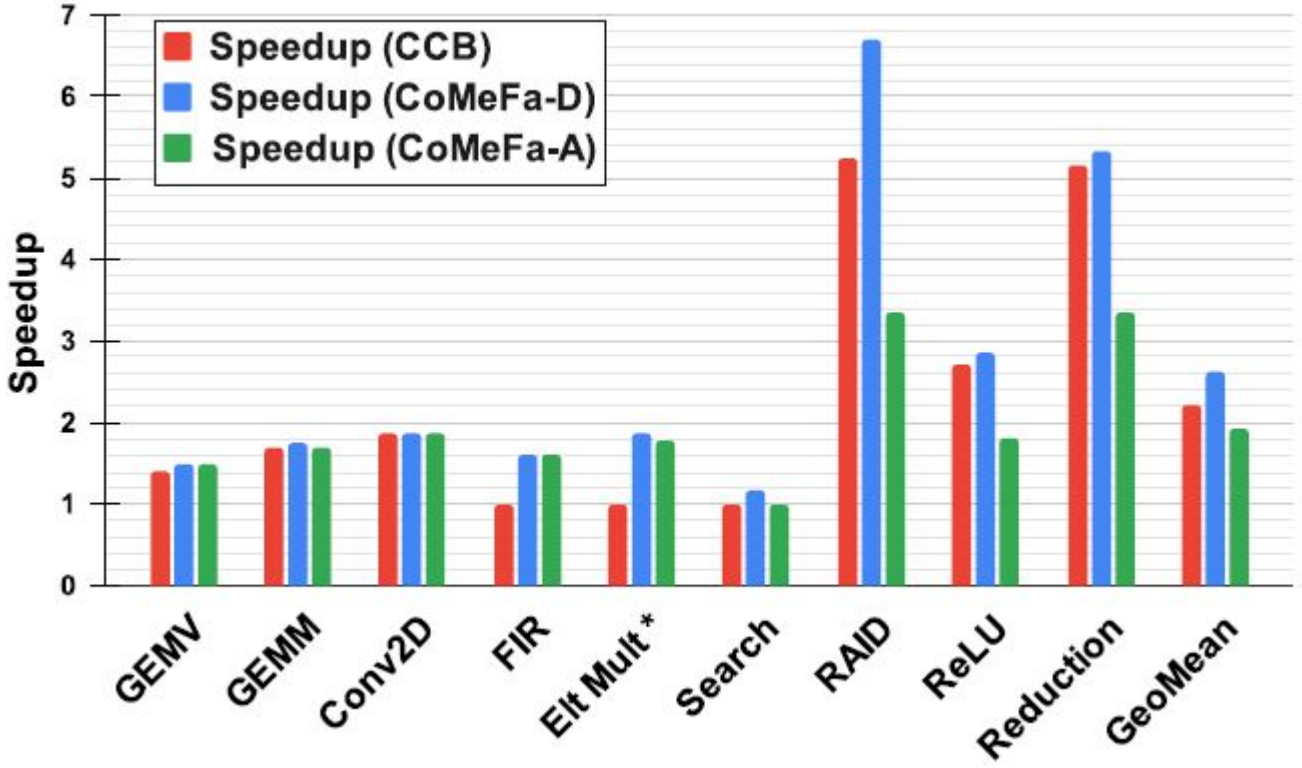
[1] X. Wang et al., "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," FCCM 2021

[2] Arora et al., "CoMeFa: Compute-in-Memory Blocks for FPGAs," FCCM 2022

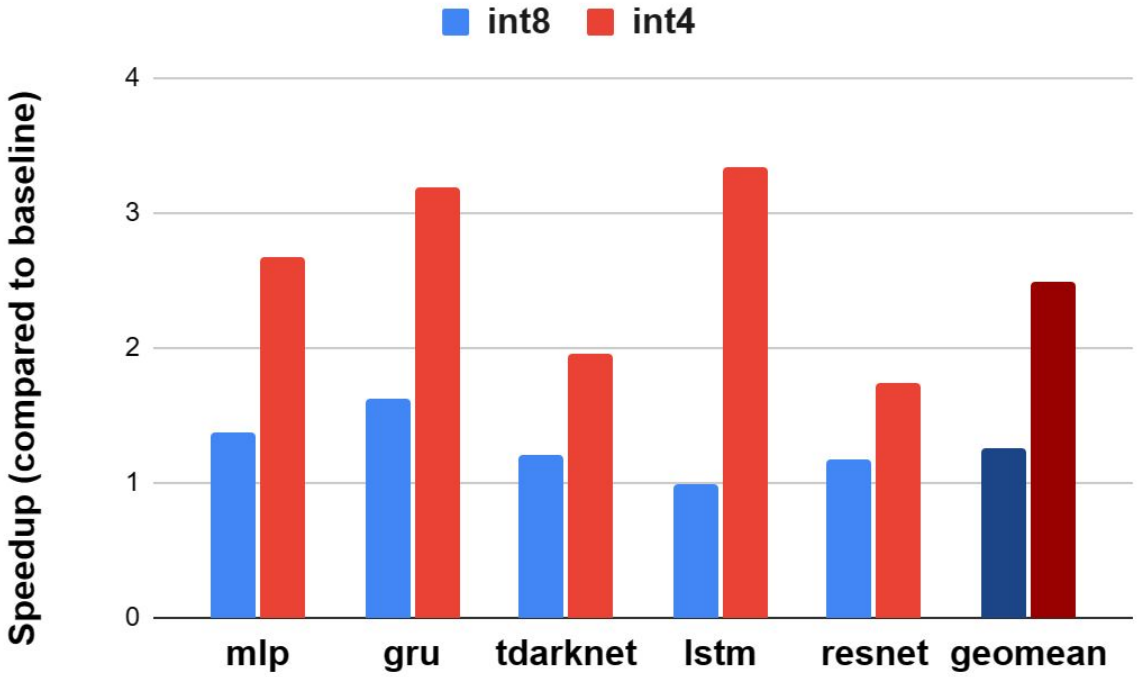
Compute Throughput (Mid-Size, Arria 10-Like)



Speedup - Microbenchmarks (8- to 20-bit precision)



Speedup - DNN Overlay (NPU-Like)



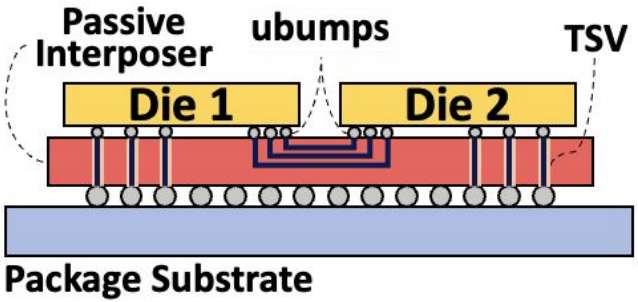
~2.5x speed-up at 4-bit, but only ~1.2x at 8-bit

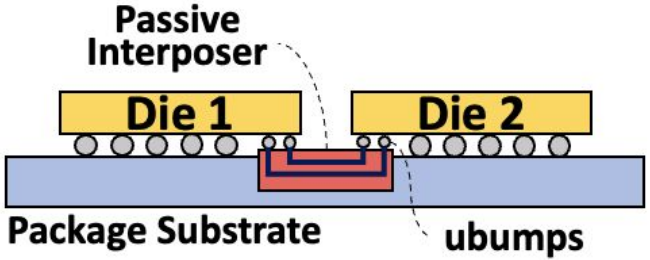
N^2 cycles for N-bit serial multiplication

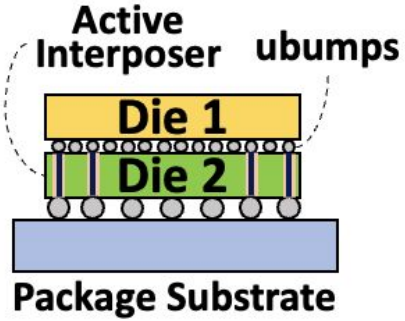
New DL-optimized FPGA Blocks

Out-of-Fabric Blocks
(Andrew)

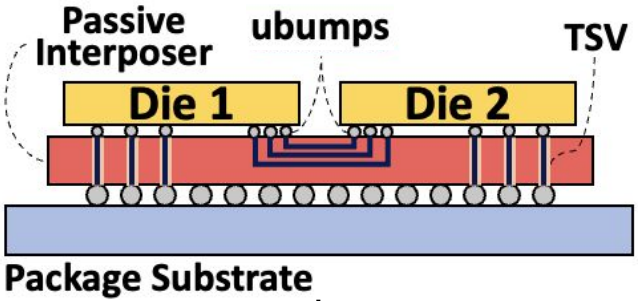
Interposers



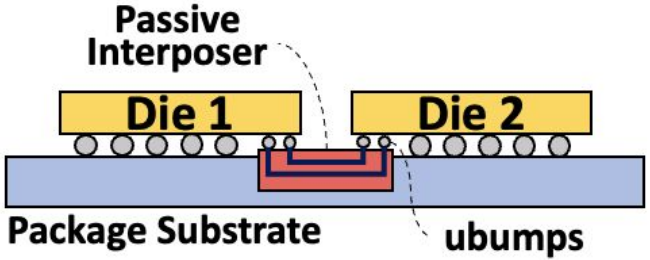




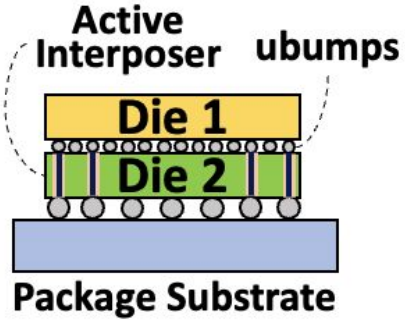
Interposers



This is how Xilinx creates large multi-die FPGAs



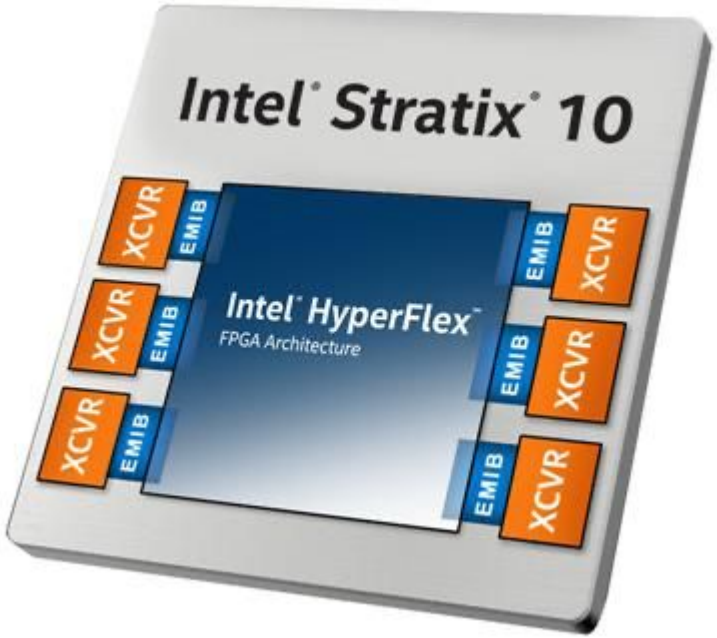
This is how Intel integrates FPGAs with transceiver and HBM *"chiplets"*



Future
www.crossroadsfpga.org

AI Targeted Chiplets

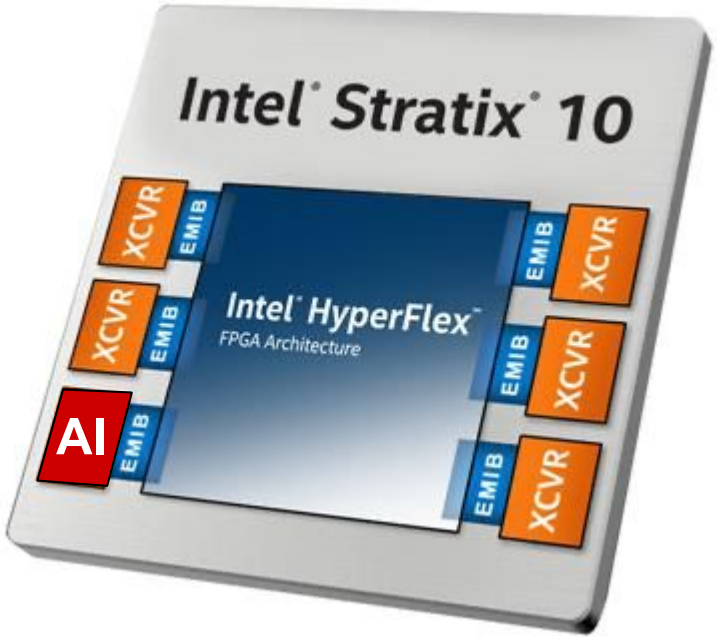
Stratix 10 uses interposer technology to integrate FPGA with transceiver chiplets



AI Targeted Chiplets

Stratix 10 uses interposer technology to integrate FPGA with transceiver chiplets

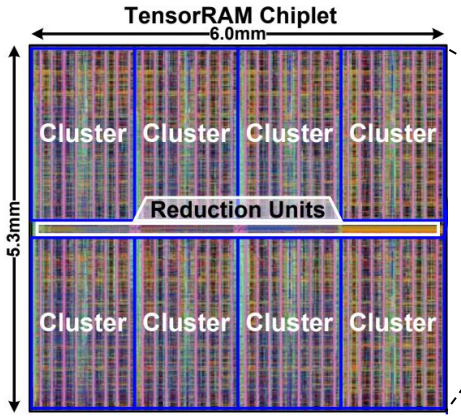
What if we swap some/all with AI chiplets?

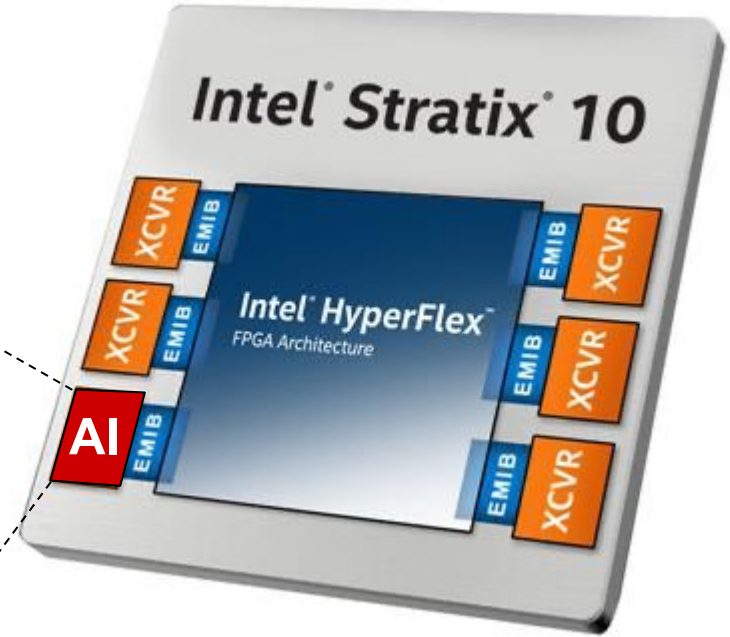


AI Targeted Chiplets

Stratix 10 uses interposer technology to integrate FPGA with transceiver chiplets

What if we swap some/all with AI chiplets?





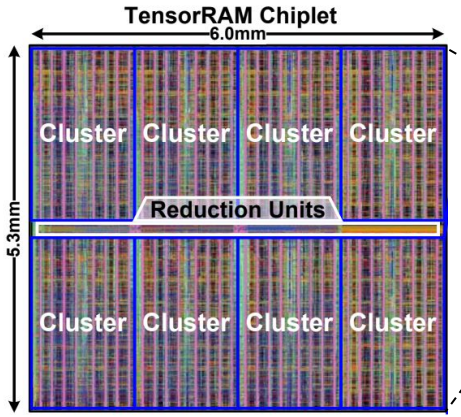
E. Nurvitadhi et al, "In-Package Domain-Specific ASICs for Intel Stratix 10 FPGAs: A Case Study of Accelerating Deep Learning using TensorTile ASIC", FPL, 2018
E. Nurvitadhi et al, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs", FCCM, 2019

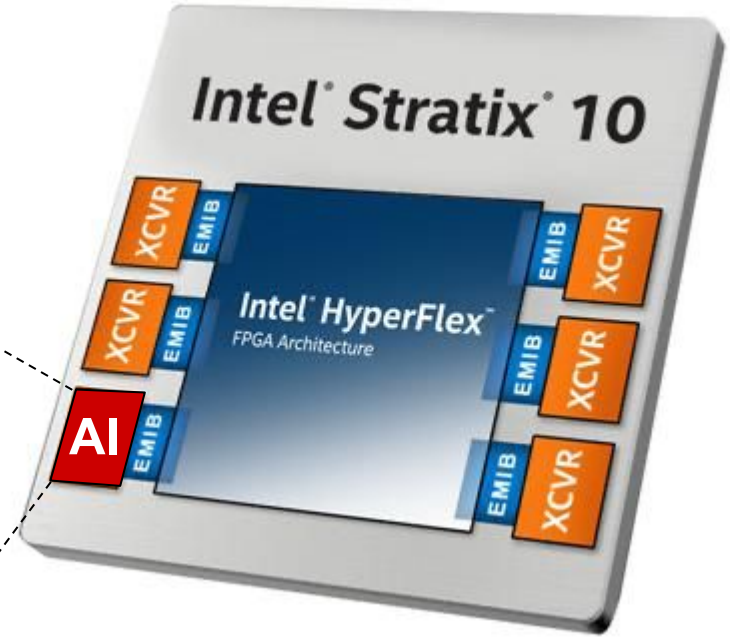
AI Targeted Chiplets

Stratix 10 uses interposer technology to integrate FPGA with transceiver chiplets

What if we swap some/all with AI chiplets?

Batch-1 Inference of Sequence Models
16x lower latency
34x higher energy efficiency vs. Volta GPU





E. Nurvitadhi et al, "In-Package Domain-Specific ASICs for Intel Stratix 10 FPGAs: A Case Study of Accelerating Deep Learning using TensorTile ASIC", FPL, 2018
E. Nurvitadhi et al, "Why Compete When You Can Work Together: FPGA-ASIC Integration for Persistent RNNs", FCCM, 2019

Xilinx Versal: Overview

It is getting **harder to design & close timing** for large FPGA systems

&

Not all applications benefit from the **bit-level flexibility** of FPGAs

Xilinx Versal: Overview

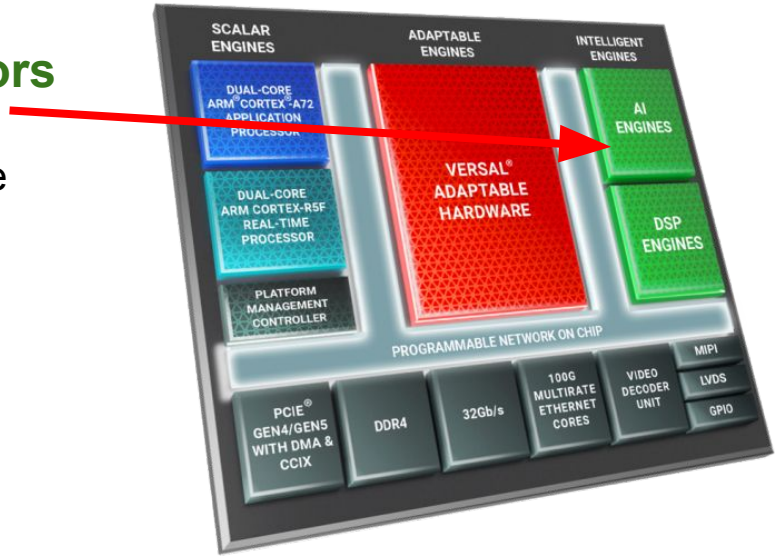
It is getting **harder to design & close timing** for large FPGA systems

&

Not all applications benefit from the **bit-level flexibility** of FPGAs

(1) Array of Specialized Vector Processors

Efficiently execute parallel workloads on SW-programmable cores with programmable bus-based routing between them



Xilinx Versal: Overview

It is getting **harder to design & close timing** for large FPGA systems

&

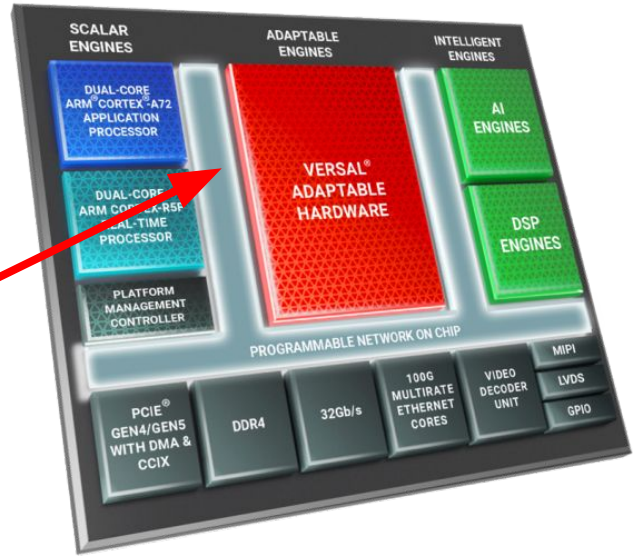
Not all applications benefit from the **bit-level flexibility** of FPGAs

(1) Array of Specialized Vector Processors

Efficiently execute parallel workloads on SW-programmable cores with programmable bus-based routing between them

(2) System-level Packet-switched NoC

Decouple compute & communication for easier system integration



Xilinx Versal: Overview

It is getting **harder to design & close timing** for large FPGA systems

&

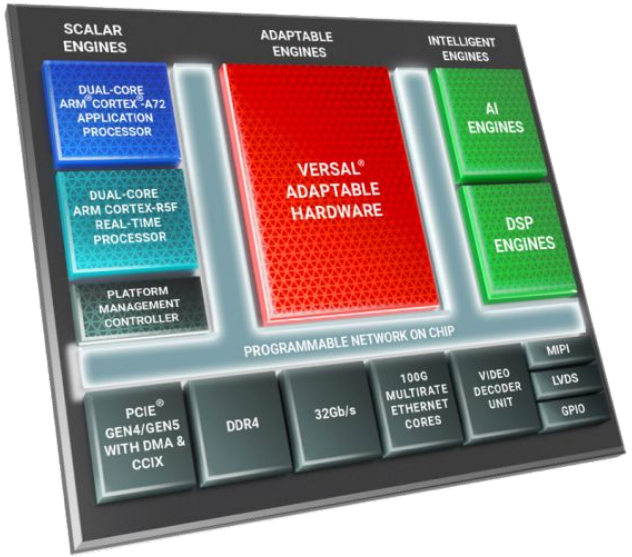
Not all applications benefit from the **bit-level flexibility** of FPGAs

(1) Array of Specialized Vector Processors

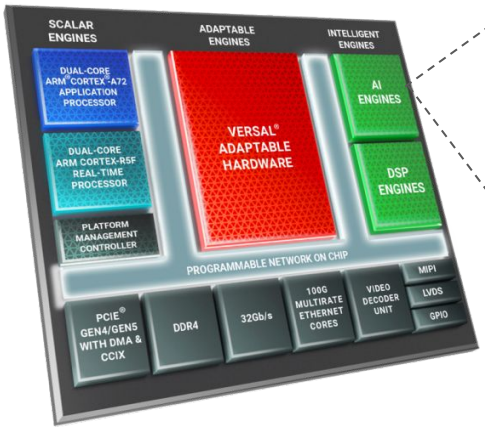
Efficiently execute parallel workloads on SW-programmable cores with programmable bus-based routing between them

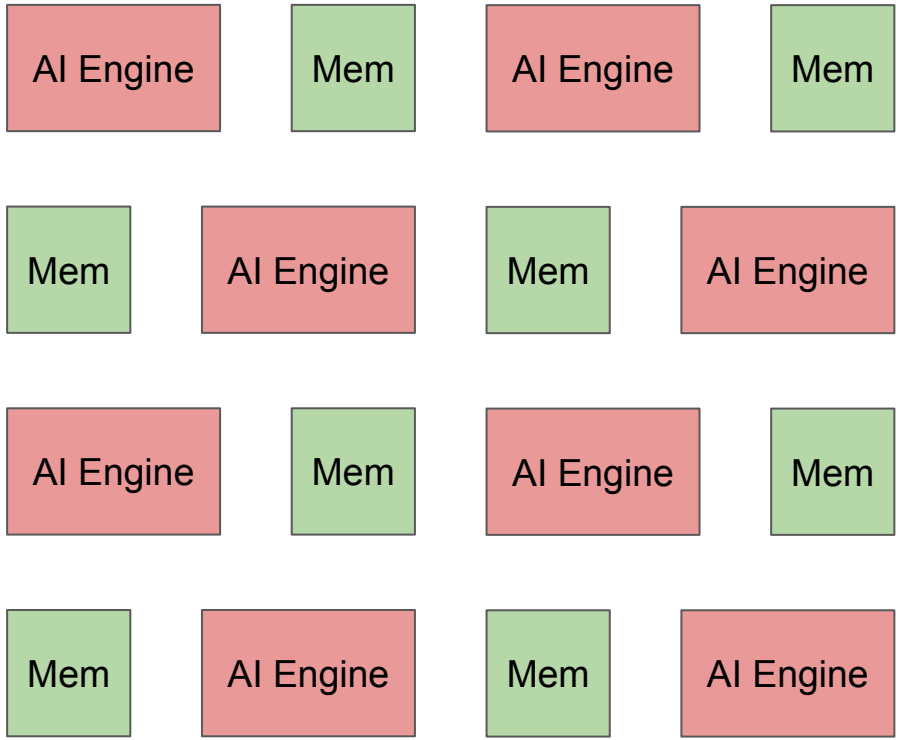
(2) System-level Packet-switched NoC

Decouple compute & communication for easier system integration



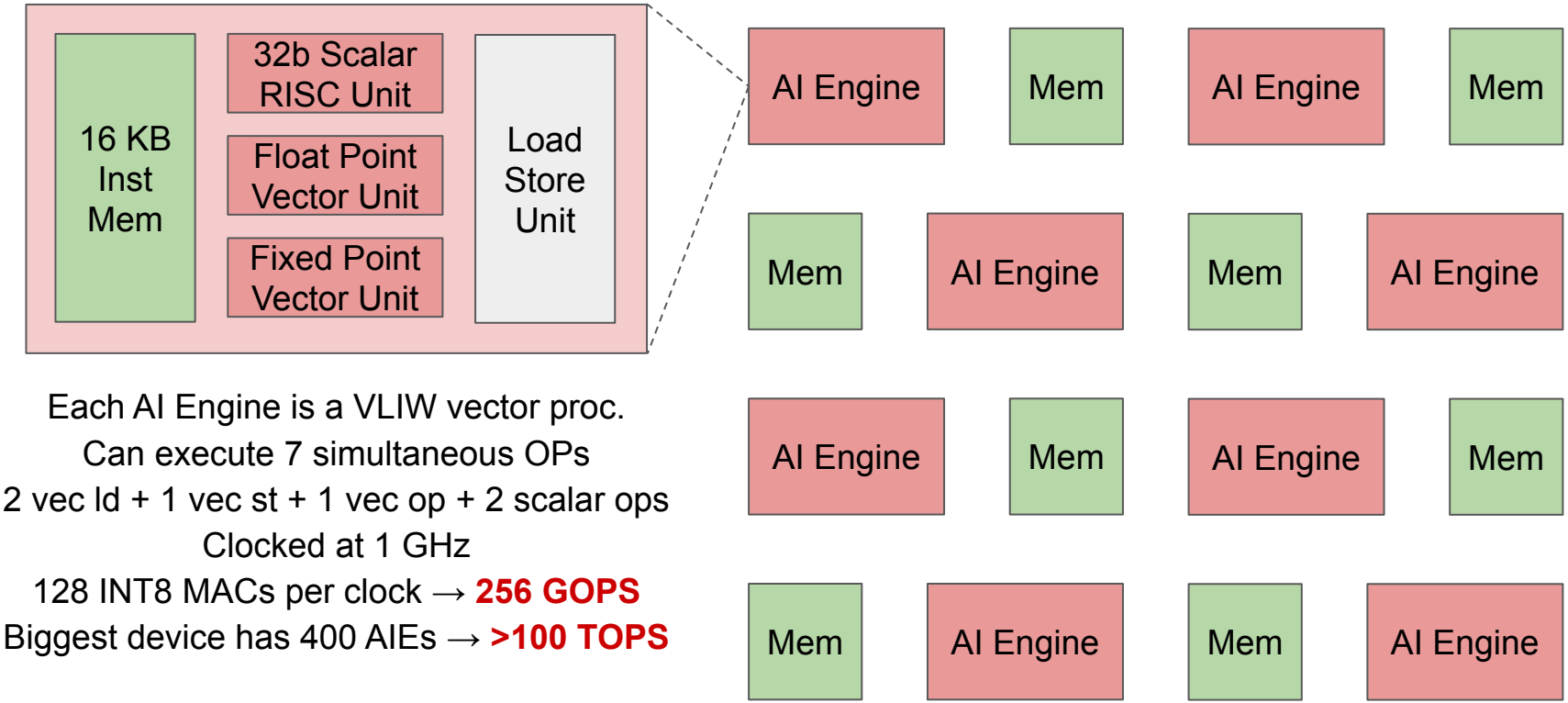
Xilinx Versal: AI Engines





S. Neuendorffer and others, "The Evolution of Domain-Specific Computing for Deep Learning" IEEE Circuits and Systems Magazine 21.2 (2021): 75-96

Xilinx Versal: AI Engines

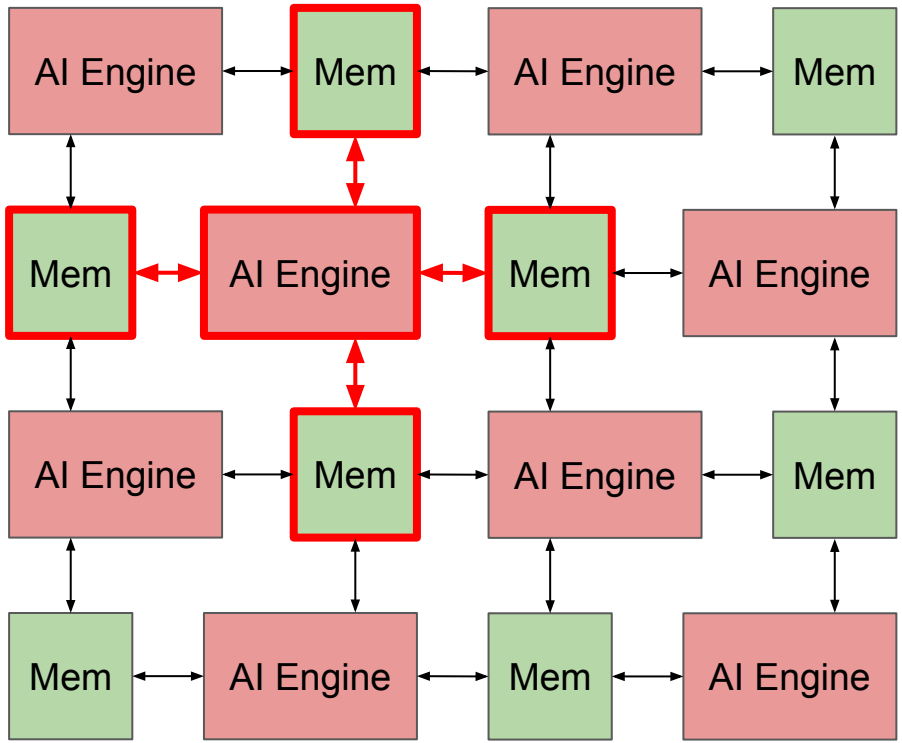


Each AI Engine is a VLIW vector proc.
Can execute 7 simultaneous OPs
2 vec ld + 1 vec st + 1 vec op + 2 scalar ops
Clocked at 1 GHz
128 INT8 MACs per clock → **256 GOPS**
Biggest device has 400 AIEs → **>100 TOPS**

S. Neuendorffer and others, "The Evolution of Domain-Specific Computing for Deep Learning" IEEE Circuits and Systems Magazine 21.2 (2021): 75-96

Xilinx Versal: AI Engines

Each AI Engine can read/write directly to its 4 adjacent memory blocks (NSEW)

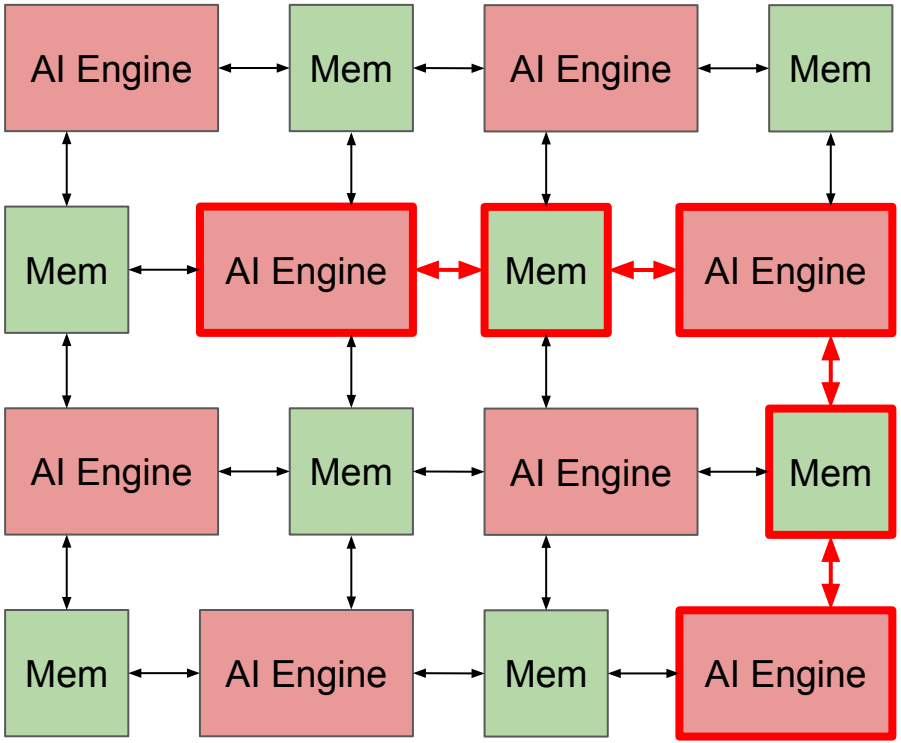


S. Neuendorffer and others, "The Evolution of Domain-Specific Computing for Deep Learning" IEEE Circuits and Systems Magazine 21.2 (2021): 75-96

Xilinx Versal: AI Engines

Each AI Engine can read/write directly to its 4 adjacent memory blocks (NSEW)

Hardware locks for sync between AIEs
→ Memory block can act as ping-pong buffer between two pipelined AIEs



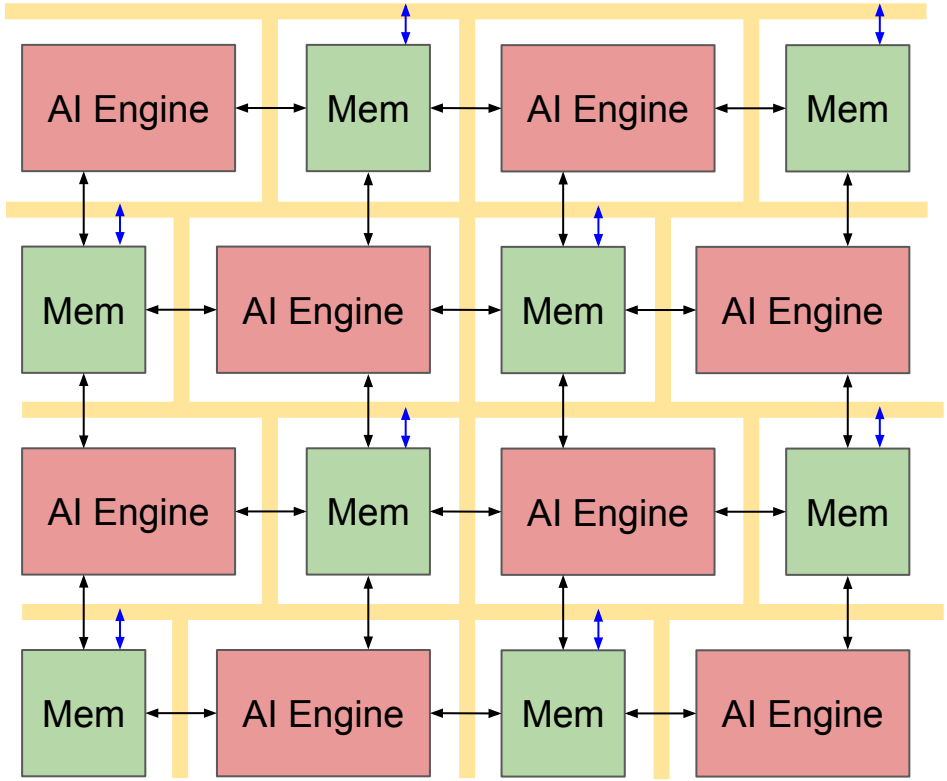
S. Neuendorffer and others, "The Evolution of Domain-Specific Computing for Deep Learning" IEEE Circuits and Systems Magazine 21.2 (2021): 75-96

Xilinx Versal: AI Engines

Each AI Engine can read/write directly to its 4 adjacent memory blocks (NSEW)

Hardware locks for sync between AIEs
→ Memory block can act as ping-pong buffer between two pipelined AIEs

Bus-based reconfigurable routing
→ AIE can can read/write data from/to the memory of any other AIE
→ Allows efficient broadcast / multicast



S. Neuendorffer and others, "The Evolution of Domain-Specific Computing for Deep Learning" IEEE Circuits and Systems Magazine 21.2 (2021): 75-96

Xilinx Versal: Overview

It is getting **harder to design & close timing** for large FPGA systems

&

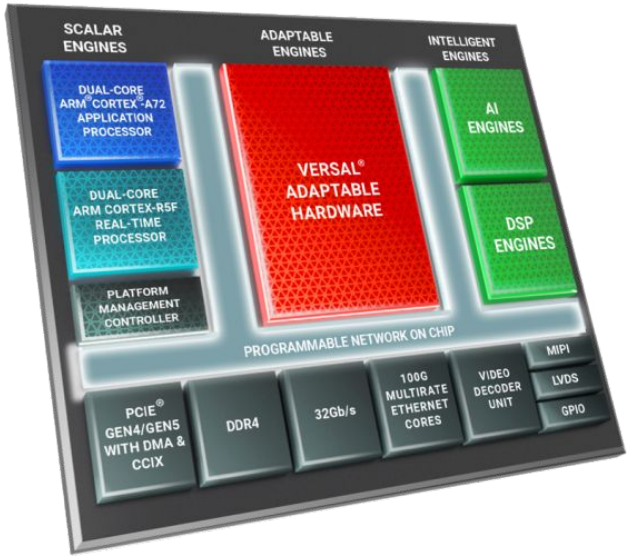
Not all applications benefit from the **bit-level flexibility** of FPGAs

(1) Array of Specialized Vector Processors

Efficiently execute parallel workloads on SW-programmable cores with programmable bus-based routing between them

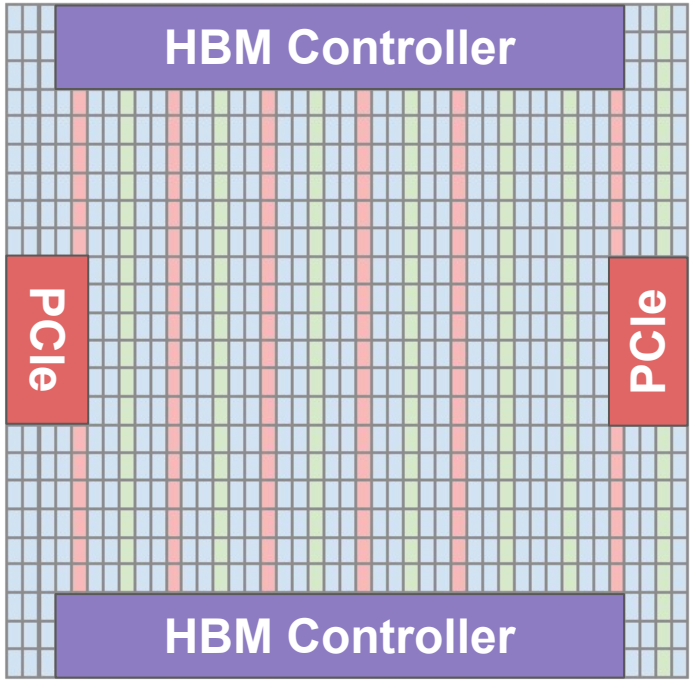
(2) System-level Packet-switched NoC

Decouple compute & communication for easier system integration



Why NoCs for FPGAs?

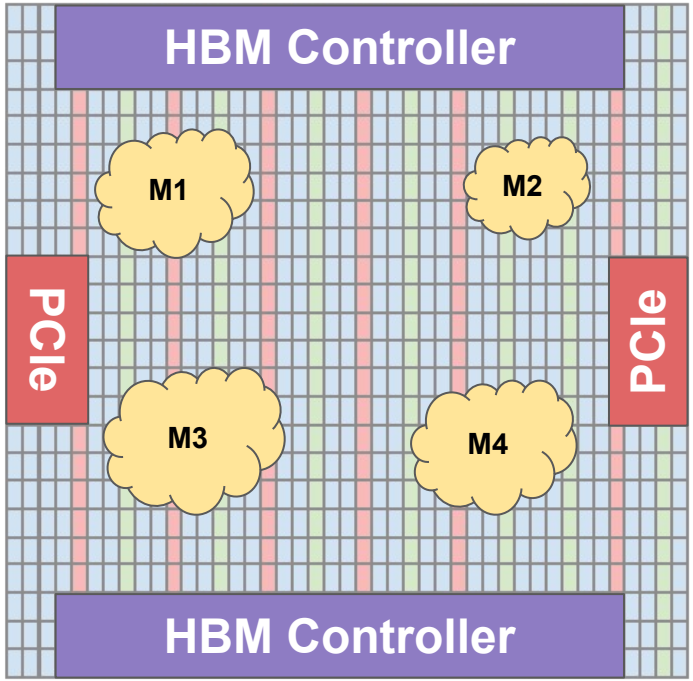
Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet



Why NoCs for FPGAs?

Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet

Large FPGA systems consist of many modules

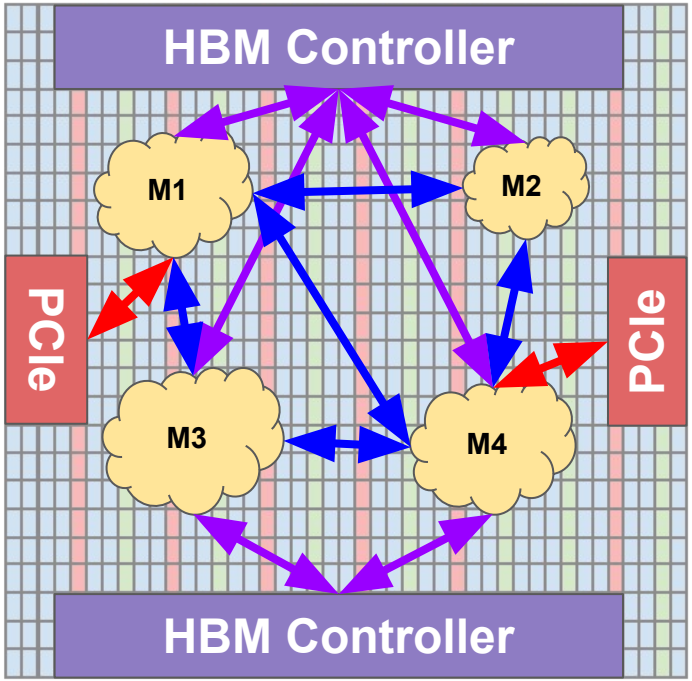


Why NoCs for FPGAs?

Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet

Large FPGA systems consist of many modules

Tremendous on-chip communication
→ between modules, modules ↔ hard blocks



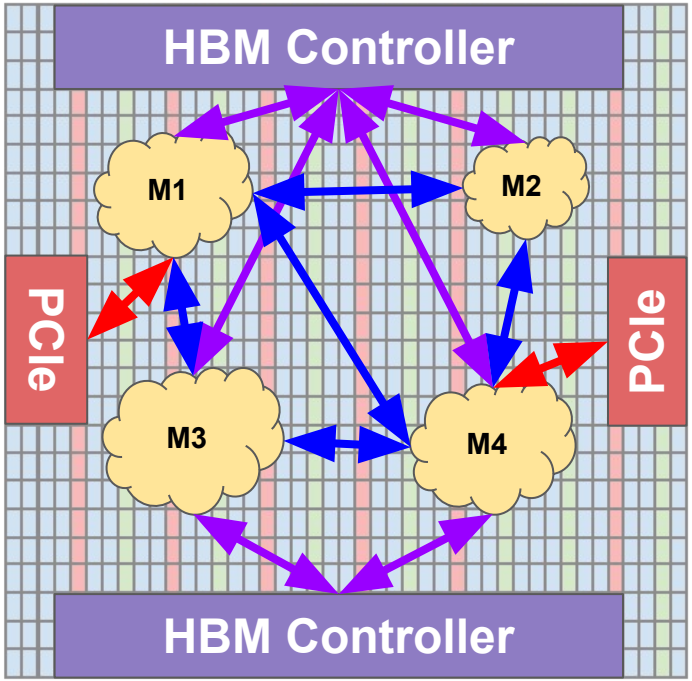
Why NoCs for FPGAs?

Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet

Large FPGA systems consist of many modules

Tremendous on-chip communication
→ between modules, modules ↔ hard blocks

Closing timing is a nightmare!
→ esp. with long CAD runtimes



Why NoCs for FPGAs?

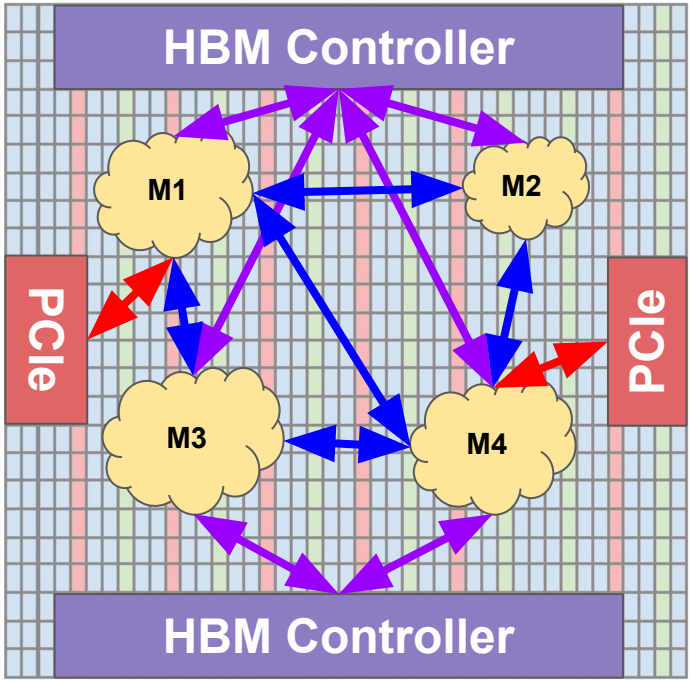
Can't harden efficient busses because of the FPGA's reconfigurability!!

Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet

Large FPGA systems consist of many modules

Tremendous on-chip communication
→ between modules, modules ↔ hard blocks

Closing timing is a nightmare!
→ esp. with long CAD runtimes



Why NoCs for FPGAs?

Modern FPGAs with many high-BW interfaces
→ HBM/DDR, PCIe, Ethernet

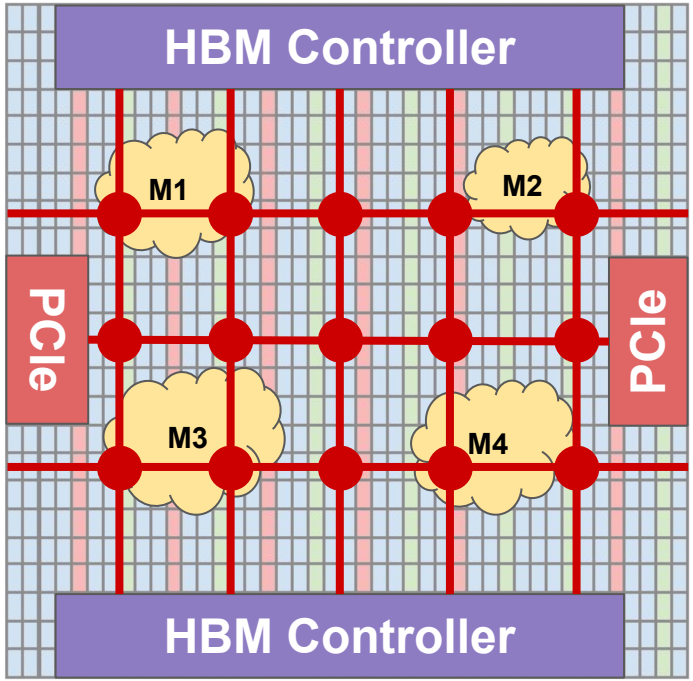
Large FPGA systems consist of many modules

Tremendous on-chip communication
→ between modules, modules ↔ hard blocks

Closing timing is a nightmare!
→ esp. with long CAD runtimes

NoCs to the rescue!

Easier timing closure
Faster system integration
More efficient communication



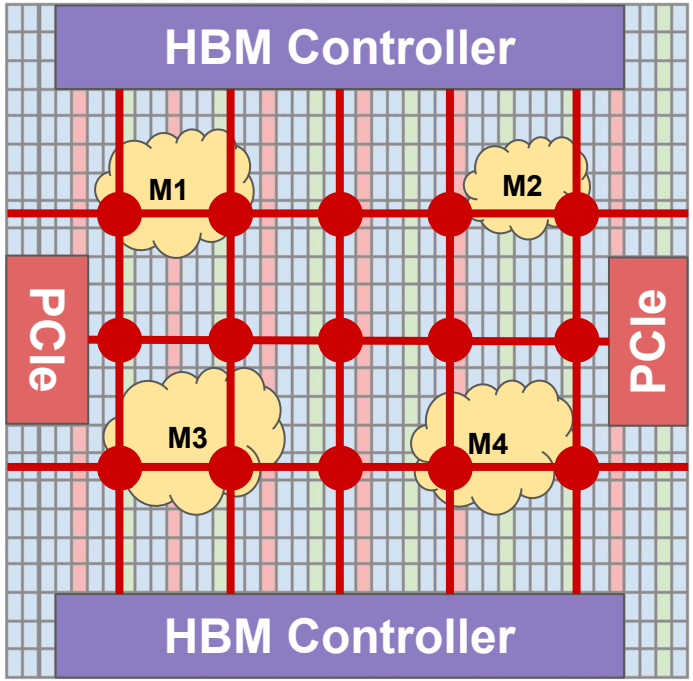
Why NoCs for FPGAs?

Many architecture questions …

What are NoC specifications?
How to connect to programmable routing?
Soft vs. Hard links?
Cost of embedding a hard NoC?
How can applications benefit from it?

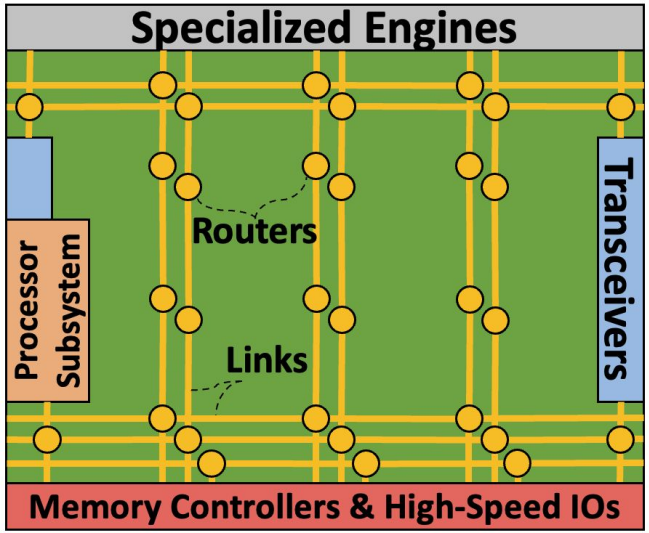
NoCs to the rescue!

Easier timing closure
Faster system integration
More efficient communication



M. Abdelfattah and others, "Design and applications for embedded networks-on-chip on FPGAs", Transactions on Computers (TC), 2016

Xilinx Versal NoC



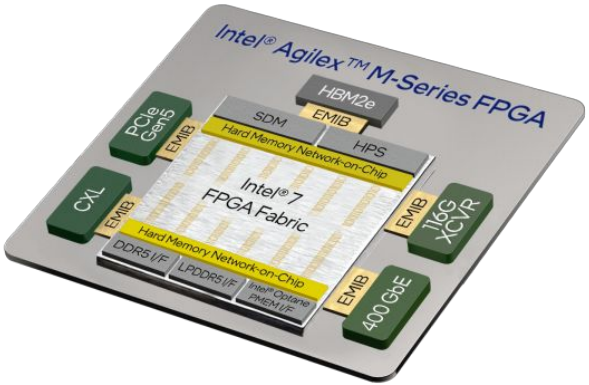
- 128b NoC links @ 1GHz → match DDR channel bandwidth
- Modified mesh topology (rows squished to top & bottom) → match FPGA column layout
- **Only** way to access external memory from the FPGA fabric
- 10s -100s fabric ports to FPGA logic presented as standard AXI interfaces

I. Swarbrick and others, "Versal network-on-chip (NoC)" IEEE *Symposium on High-Performance Interconnects (HOTI)*, 2019

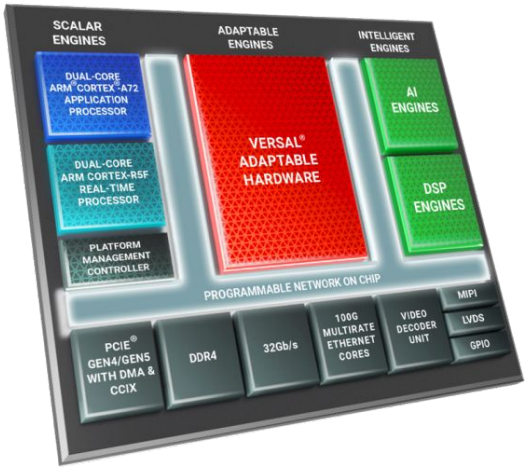
New DL-optimized FPGA Blocks

Beyond-FPGA Devices (Andrew)

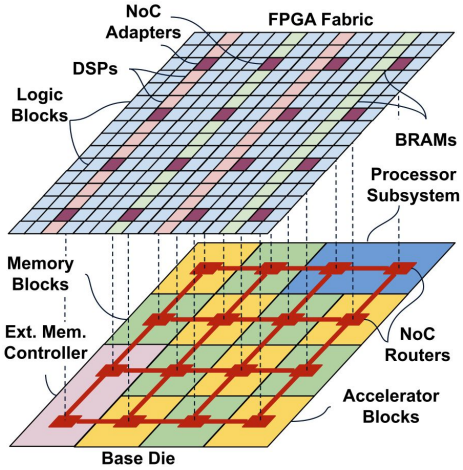
The Rise of “Beyond-FPGA” Devices



Intel FPGA System-in-Package (Chiplets)



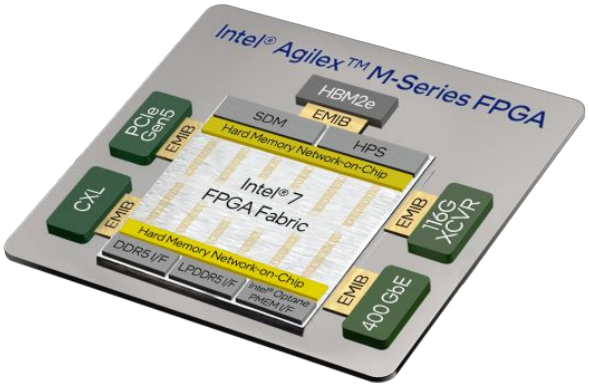
Xilinx Versal ACAP



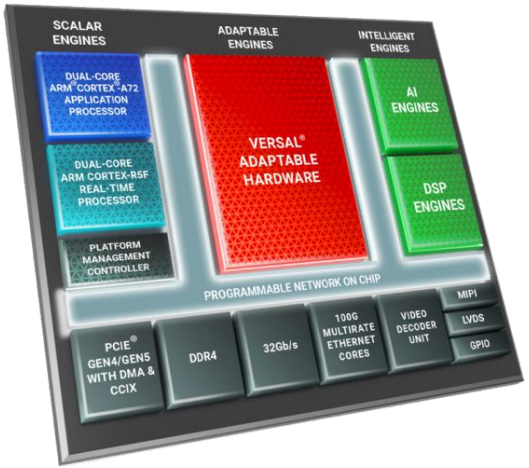
Future 3D-Integrated Devices

The Rise of “Beyond-FPGA” Devices

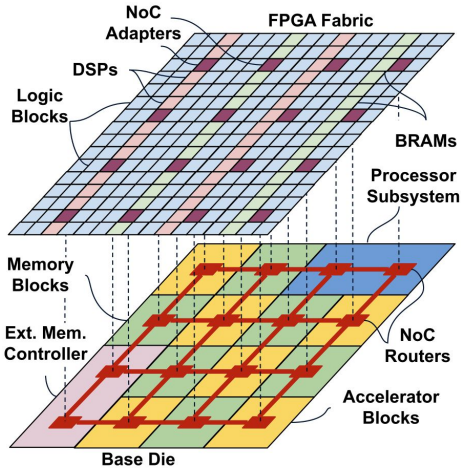
Reconfigurable **A**cceleration **D**evices (**RADs**)



Intel FPGA System-in-Package (Chiplets)



Xilinx Versal ACAP



Future 3D-Integrated Devices

New Territories … New Evaluation Tools!

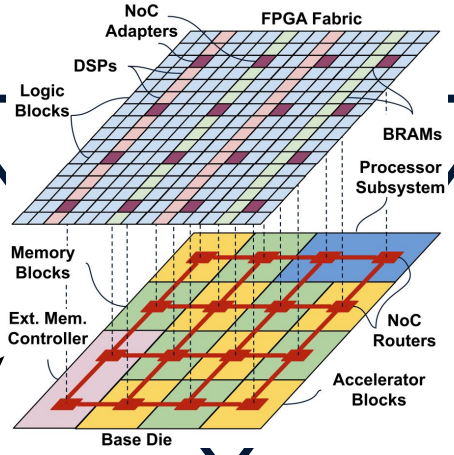
Routing Fabric?

Logic Blocks?

Hard Blocks?

FPGA CAD Algorithms?

FPGA Fabric



Coarse-Grain Accelerator Blocks

Functionality?

Granularity?

Quantity?

Programming Model?

New RADs

(Reconfigurable Acceleration Devices)

System-Level Interconnect

NoC(s) spec?

Abstraction?

NoC Placement?

New Territories … New Evaluation Tools!

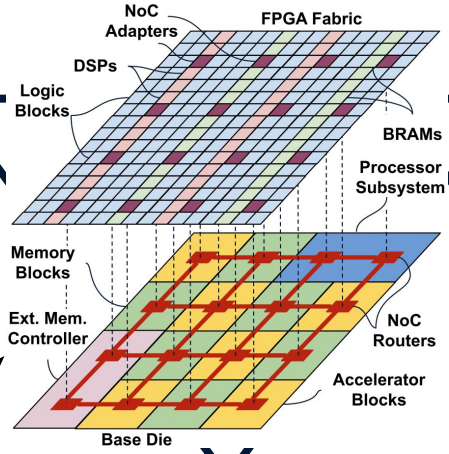
Routing Fabric?

Logic Blocks?

Hard Blocks?

FPGA CAD Algorithms?

FPGA Fabric



Coarse-Grain Accelerator Blocks

Functionality?

Granularity?

Quantity?

Programming Model?

New RADs

(Reconfigurable Acceleration Devices)

Huge Design Space to Explore … Need new tools!

System-Level Interconnect

NoC(s) spec?

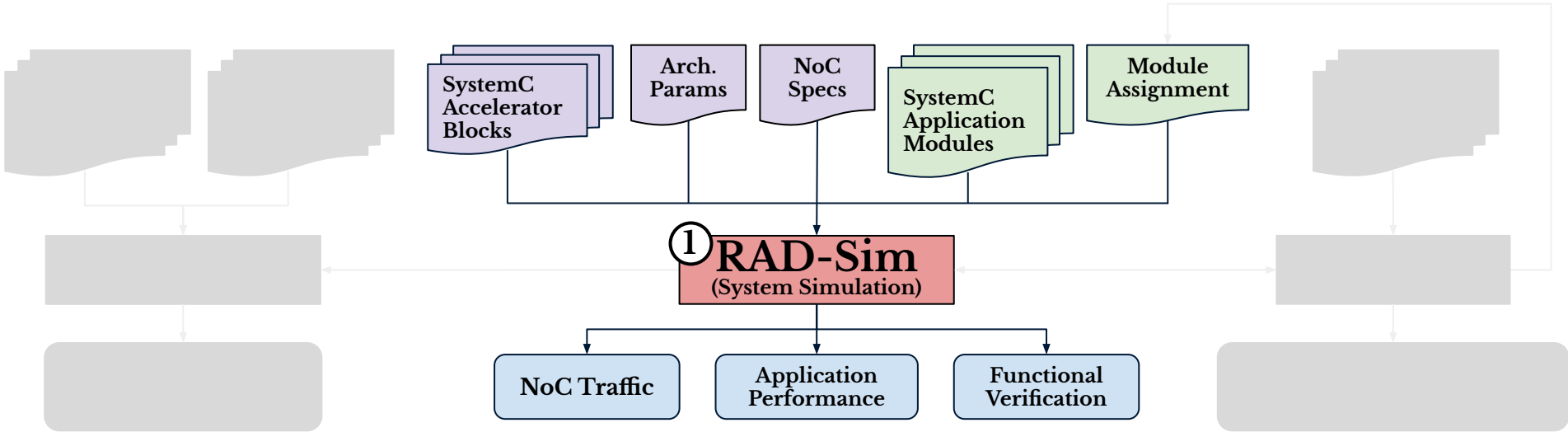
Abstraction?

NoC Placement?

The RAD Flow

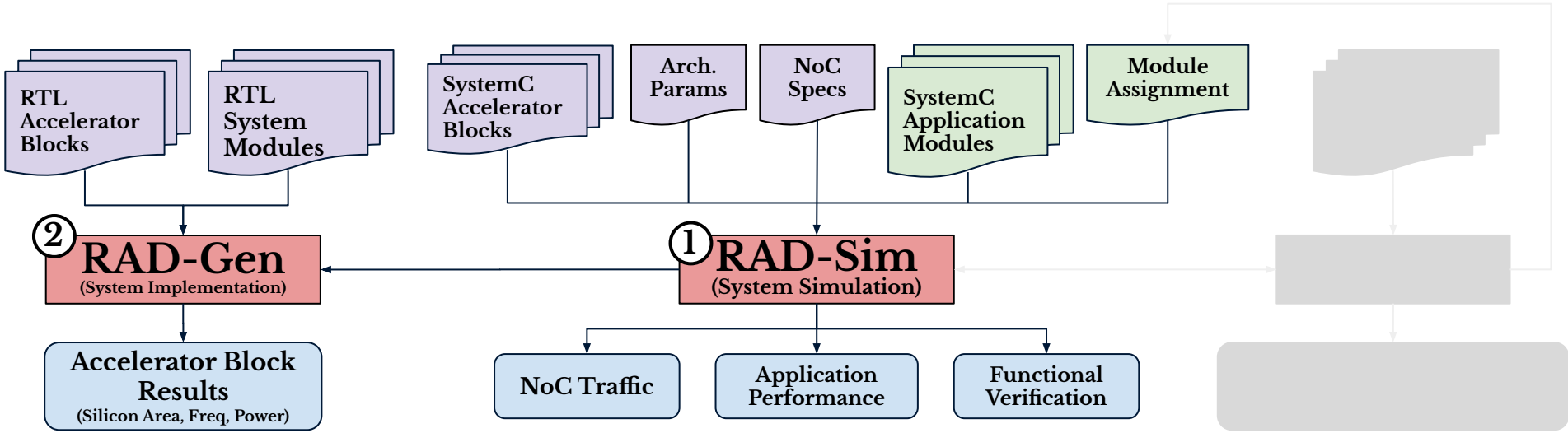
A. Boutros and others, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices", IEEE Access (2022)
Github Repo: https://github.com/andrewboutros/rad-flow

The RAD Flow



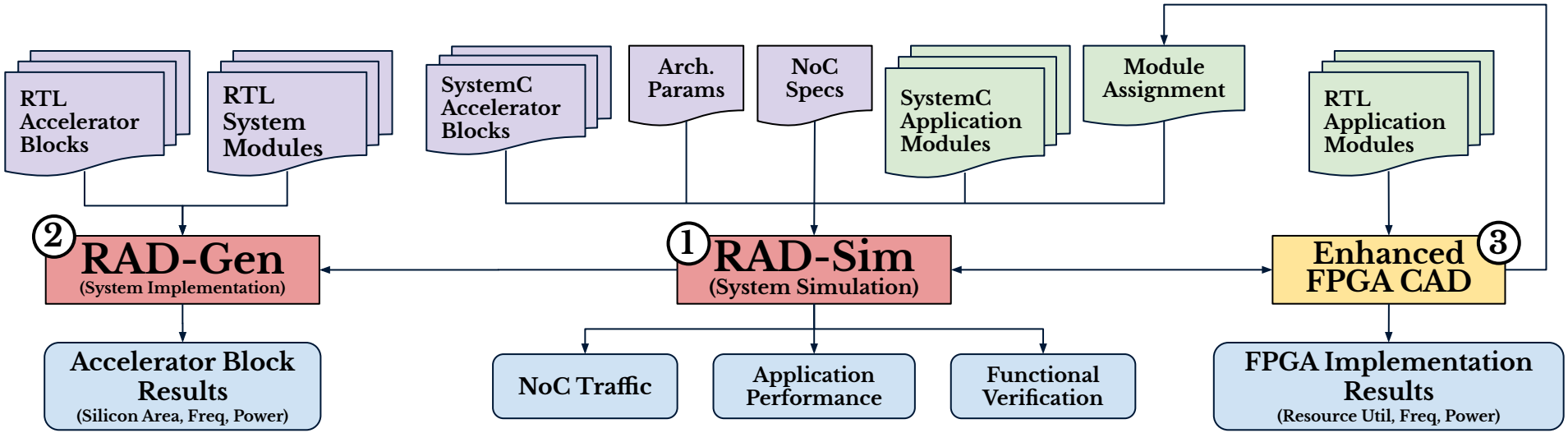
A. Boutros and others, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices", IEEE Access (2022)
Github Repo: https://github.com/andrewboutros/rad-flow

The RAD Flow



A. Boutros and others, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices", IEEE Access (2022)
Github Repo: https://github.com/andrewboutros/rad-flow

The RAD Flow



A. Boutros and others, "Architecture and Application Co-Design for Beyond-FPGA Reconfigurable Acceleration Devices", IEEE Access (2022)
Github Repo: https://github.com/andrewboutros/rad-flow

Summary

- FPGA architecture has always evolved to meet the needs of key markets … DL is a big one!

- FPGA architecture has always evolved to meet the needs of key markets … DL is a big one!
- Traditional blocks optimized for DL (logic blocks, DSPS)
  - Maintain FPGA generality
  - Achieve considerable gains at minimal cost

- FPGA architecture has always evolved to meet the needs of key markets … DL is a big one!
- Traditional blocks optimized for DL (logic blocks, DSPS)
  - Maintain FPGA generality
  - Achieve considerable gains at minimal cost
- New DL-targeted blocks (tensor blocks, compute-in-BRAMs)
  - New class of specialized FPGAs for DL
  - Higher gains at a higher cost

- FPGA architecture has always evolved to meet the needs of key markets … DL is a big one!
- Traditional blocks optimized for DL (logic blocks, DSPS)
  - Maintain FPGA generality
  - Achieve considerable gains at minimal cost
- New DL-targeted blocks (tensor blocks, compute-in-BRAMs)
  - New class of specialized FPGAs for DL
  - Higher gains at a higher cost
- Heterogeneous reconfigurable devices
  - Monolithic → NoCs + coarse-grained accelerators
  - 2.5D Integration → DL chiplets
  - 3D Integration?

Thanks!

References

VTR: https://github.com/verilog-to-routing/vtr-verilog-to-routing

COFFE: https://github.com/vaughnbetz/COFFE

Koios: https://tinyurl.com/vtrkoios

EB benchmark framework:

E. Roorda, S. Rasoulinezhad, P. H. W. Leong, and S. J. E. Wilton, “FPGA Architecture Exploration for DNN Acceleration”, ACM Transactions on Reconfigurable Technology and Systems, Vol. 15, No. 3, May 2022. [Online]. Available: https://doi.org/10.1145/3503465

References

- A. Boutros et al., "Embracing Diversity: Enhanced DSP Blocks for Low-Precision Deep Learning on FPGAs," FPL18, doi:10.1109/FPL.2018.00014.

- S. Rasoulinezhad et al., "PIR-DSP: An FPGA DSP Block Architecture for Multi-precision Deep Neural Networks," *FCCM19*, doi:10.1109/FCCM.2019.00015. https://github.com/raminrasoulinezhad/PIR-DSP

References

● Aman Arora *et al*., “Tensor Slices: FPGA Building Blocks For The Deep Learning Era,” ACM TRETS (Dec 2022), doi.org/10.1145/3529650.

● A. Boutros *et al*., "Beyond Peak Performance: Comparing the Real Performance of AI-Optimized FPGAs and GPUs," *International Conference on Field-Programmable Technology (ICFPT)*, 2020, pp. 10-19, doi:10.1109/ICFPT51103.2020.00011.

● Seyedramin Rasoulinezhad *et al*., “Rethinking embedded blocks for machine learning applications,” *ACM TRETS*, (Nov 2021) doi:10.1145/3491234. http://github.com/raminrasoulinezhad/MLBlocks

References

Neural Cache

C. Eckert *et al*., "Neural Cache: Bit-Serial In-Cache Acceleration of Deep Neural Networks," *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, 2018, pp. 383-396, doi: 10.1109/ISCA.2018.00040.

CCB

X. Wang et al., "Compute-Capable Block RAMs for Efficient Deep Learning Acceleration on FPGAs," 2021 IEEE 29th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2021, pp. 88-96, doi: 10.1109/FCCM51124.2021.00018.

CoMeFa

A. Arora et al., "CoMeFa: Compute-in-Memory Blocks for FPGAs," 2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), 2022, pp. 1-9, doi: 10.1109/FCCM53951.2022.9786179.