Compute-Efficient Neural-Network Acceleration

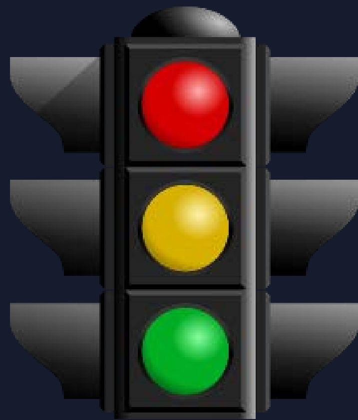
Ephrem Wu, Xiaoqian Zhang, David Berman, Inkeun Cho, John Thendean

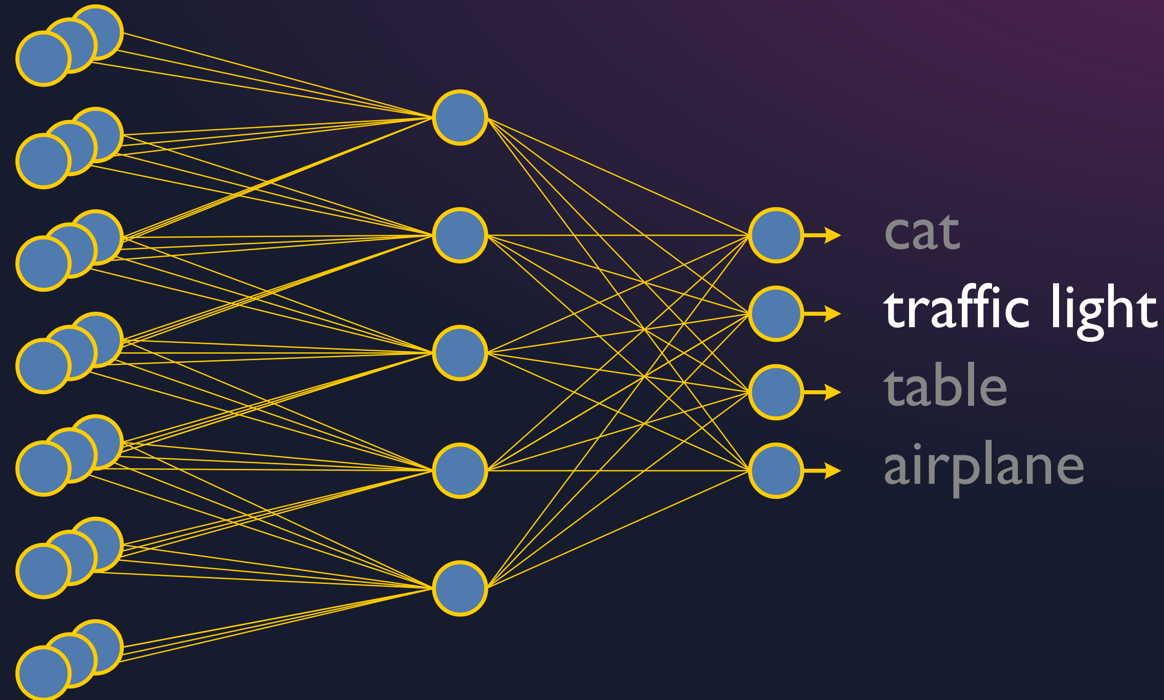
FPGA 2019
2/24/2019

© Copyright 2019 Xilinx

Goal

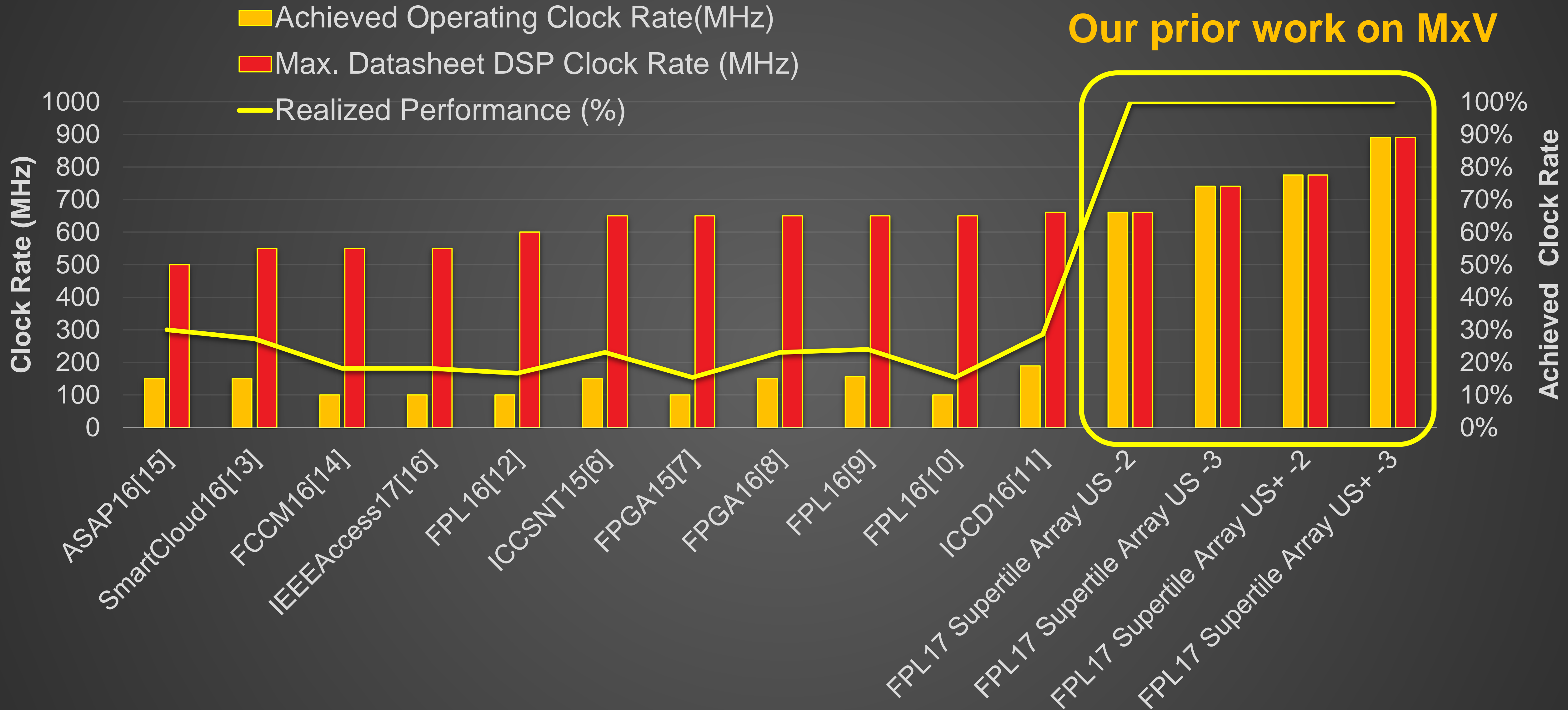
Build a compute-efficient and reconfigurable neural-net inference accelerator





© Copyright 2019 Xilinx

We Don't Have to Leave DSP Performance Behind



References from FPL17

© Copyright 2019 Xilinx

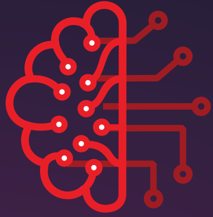
XILINX

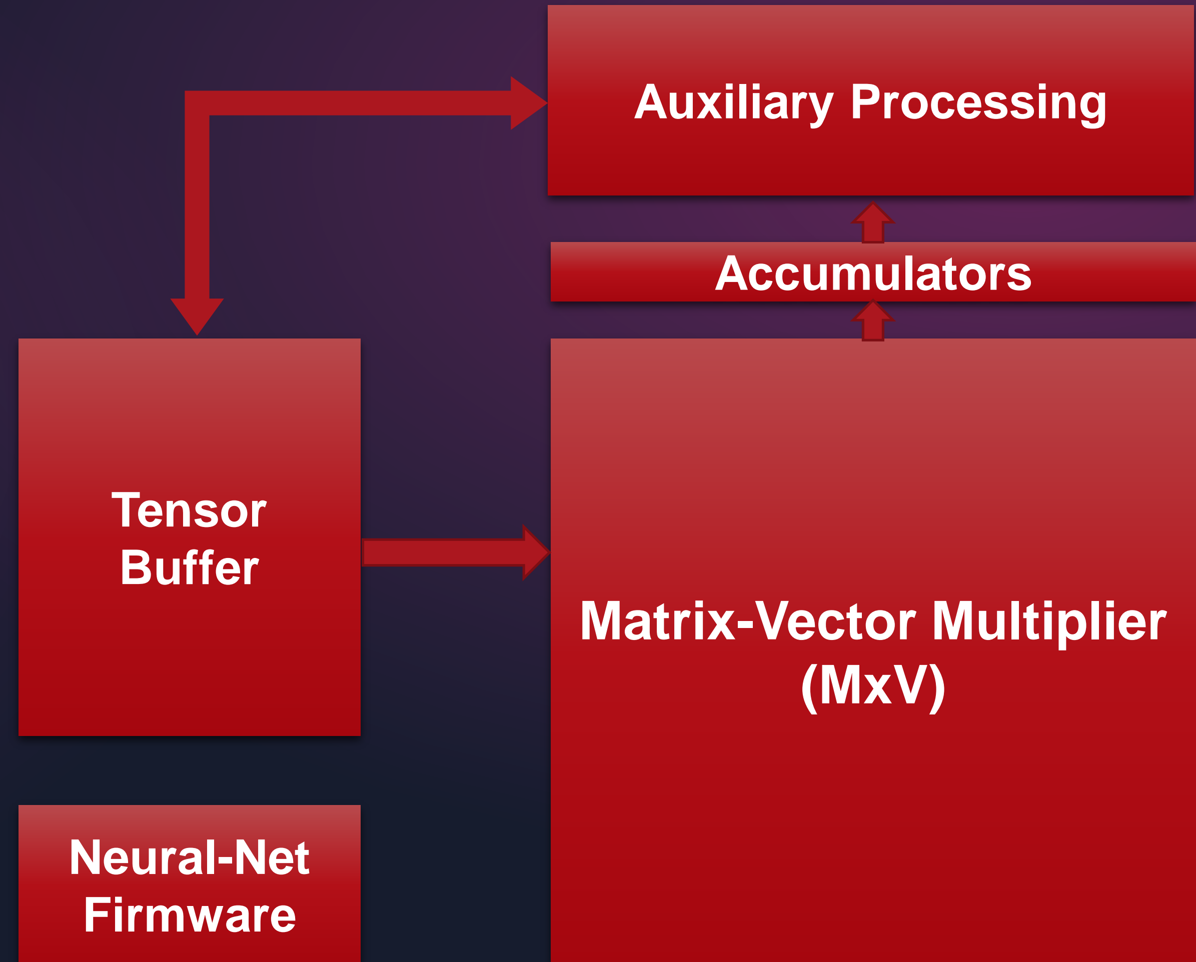
Key Design Principles

1. Make compute fast.
2. Keep compute busy.
3. Simplify implementation.

© Copyright 2019 Xilinx

A Domain-Specific Architecture for Neural Networks

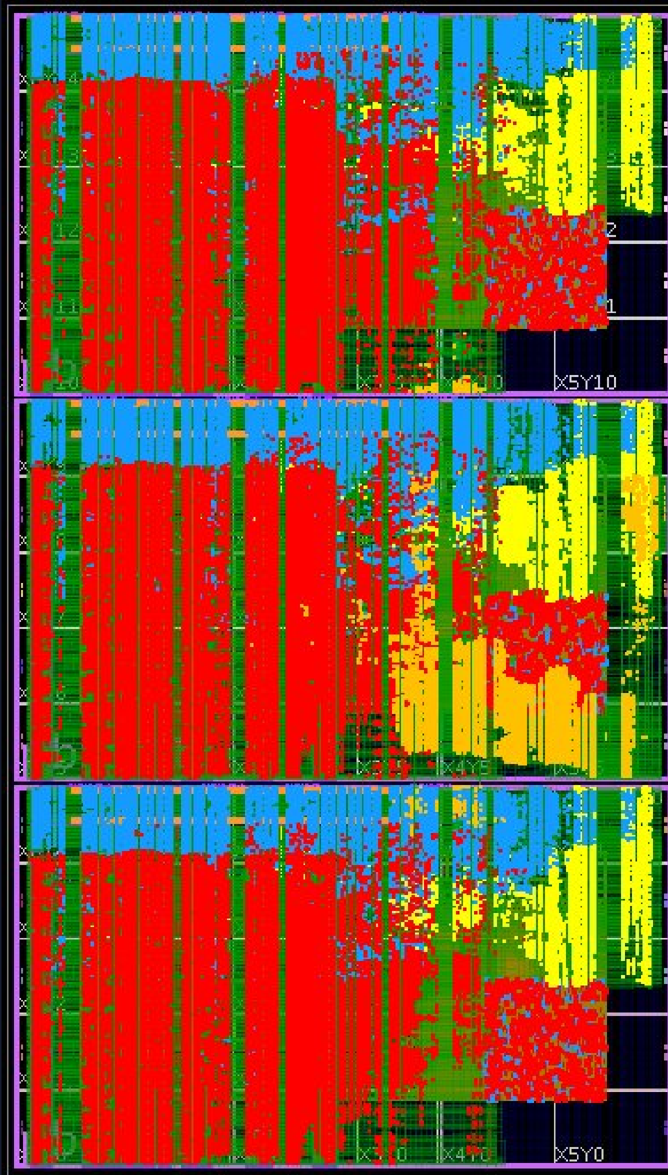




Domain-specific architectures called for by Hennessey and Patterson. See "A New Golden Age for Computer Architecture, " Communications of the ACM, February 2019, Vol. 62 No. 2, Pages 48-60.

© Copyright 2019 Xilinx

Compute-Efficient Neural Processing



VU9P Layout

> **Case Study: GoogLeNet v1 Inference**
  >> 3 parallel GoogLeNets with their own weights
  >> Aggregate 3046 images/sec, 3.3 ms latency

> **Platform: VCU1525 board with VU9P-2 FPGA**

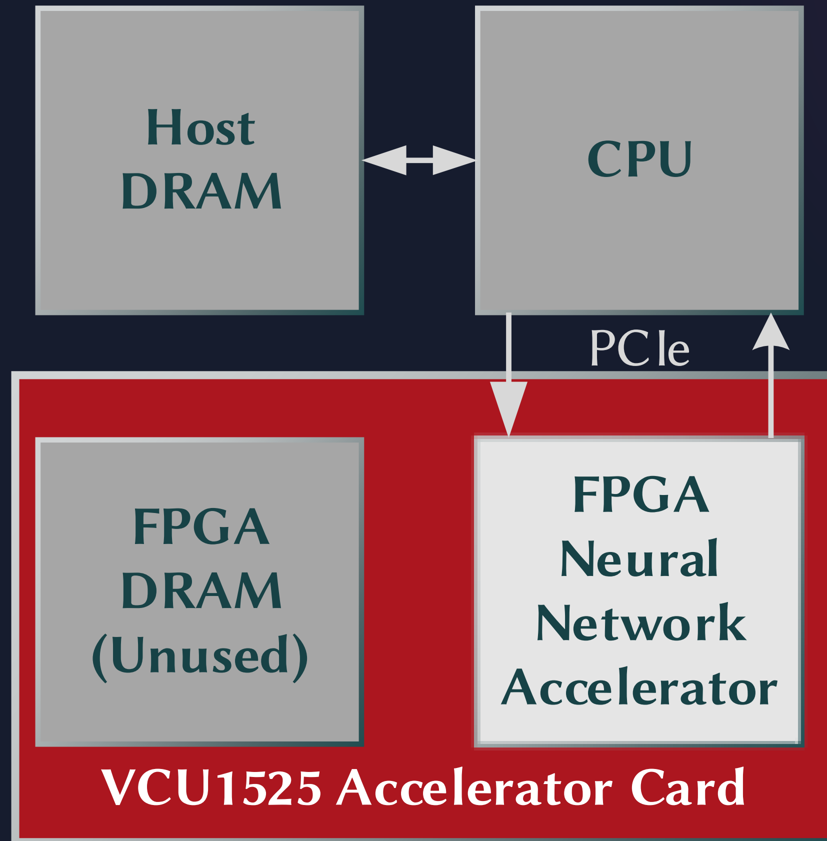
> **Compute**
  >> DSP supertile arrays running at 720 MHz
  >> Consumes only 56% DSP48 tiles
  >> DSP cycles 95% utilized
  >> Per-tensor block floating-point, 8-/16-bit significands

> **Memory**
  >> No external DRAM on accelerator card used
  >> All tensors stored in UltraRAM & BRAM
  >> 1/2 DSP clock rate

© Copyright 2019 Xilinx

System and Accelerator Block Diagrams

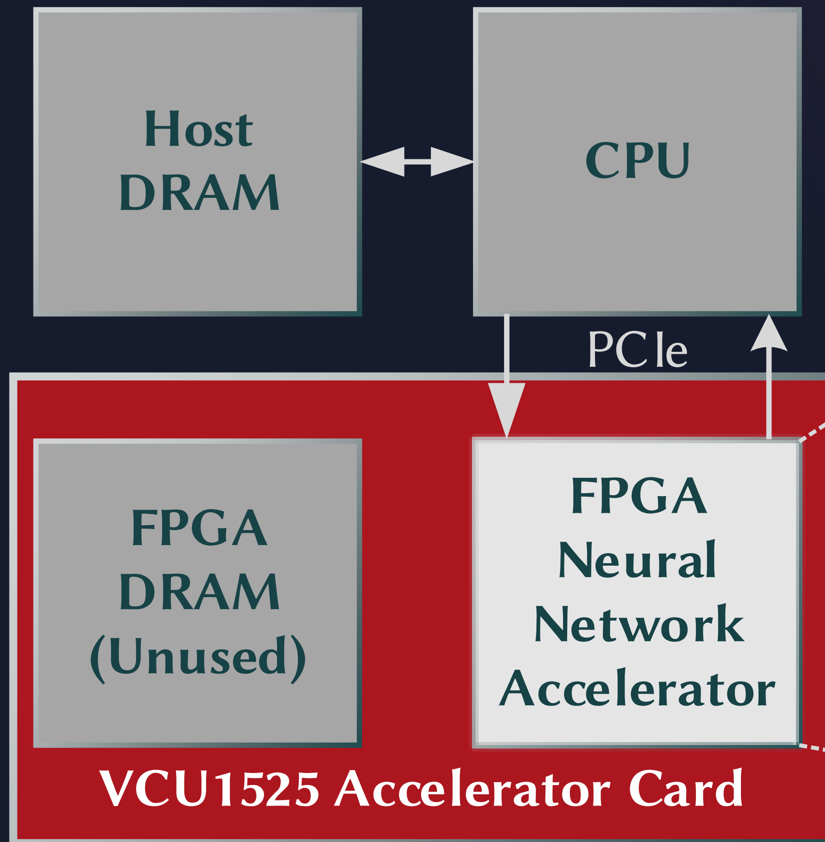
System



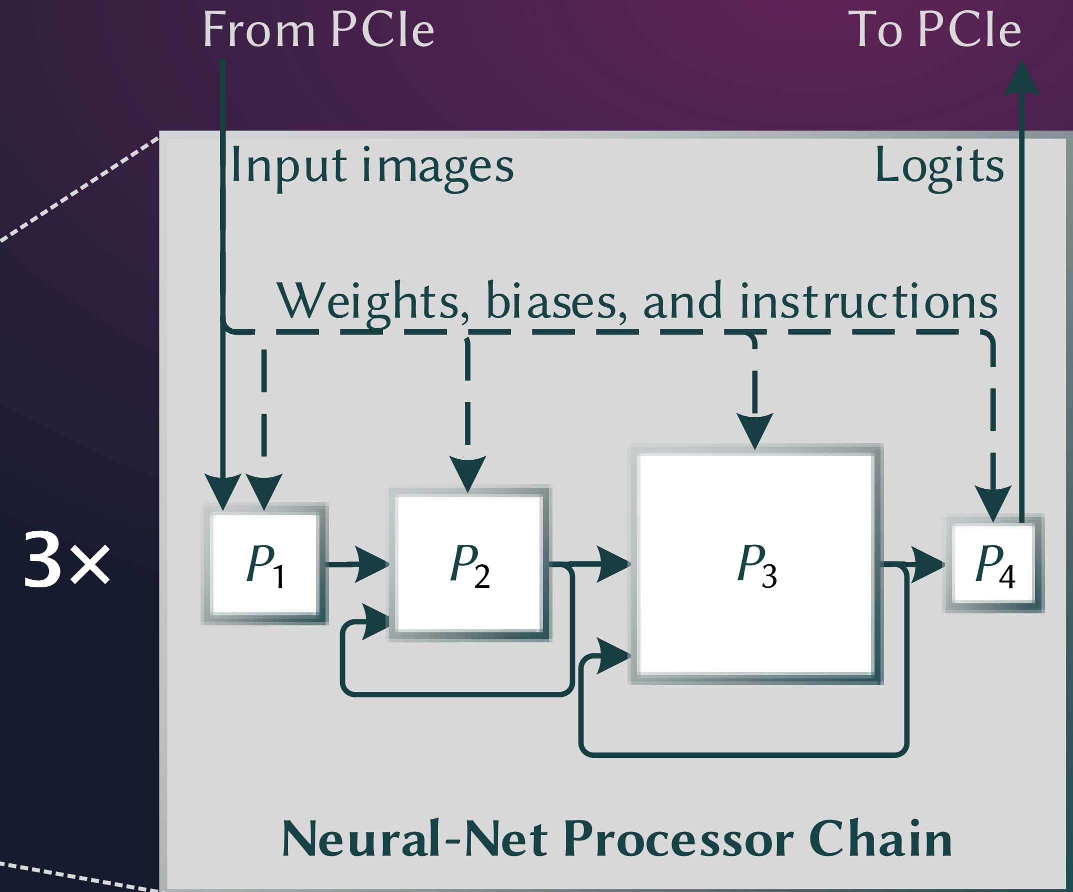
© Copyright 2019 Xilinx

System and Accelerator Block Diagrams

System



Accelerator



© Copyright 2019 Xilinx

XILINX

Apples-to-Apples Accelerator Comparison

- Throughput per device doesn't tell the whole story
- Differences: FPGAs, total compute, data movement, numerical formats, and clock rates
- Normalize results
  - Achieved Clock Rate: Actual DSP clock compared to datasheet F_{max}
  - Compute Efficiency: % available DSP cycles that do useful work

Make compute fast

Keep compute busy

Overall Efficiency = % DSP Fmax × Compute Efficiency

© Copyright 2019 Xilinx

GoogLeNet Inference Comparison

FPGA Implementation	Realized Clock Rate (% of Peak)	Compute Efficiency	Overall Efficiency
Ngo, 2016	15.4%	15.0%	2.3%
Venieris, 2017	19.2%	66.6%	12.8%
Shen, 2017	26.1%	93.8%	24.5%
Huang, 2018	30.8%	81.4%	25.1%
Gokhale, 2017	38.4%	91.0%	34.9%
This work	92.9%	95.5%	88.7%

The higher the clock rate, the harder it is to reach high compute efficiency

Bang for the buck from each DSP resource

© Copyright 2019 Xilinx

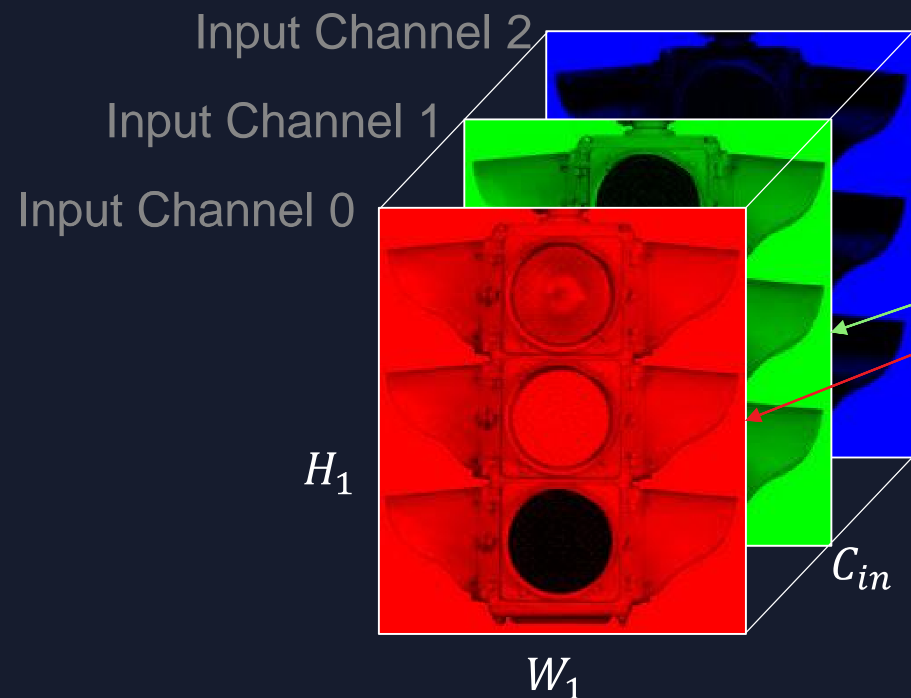
XILINX

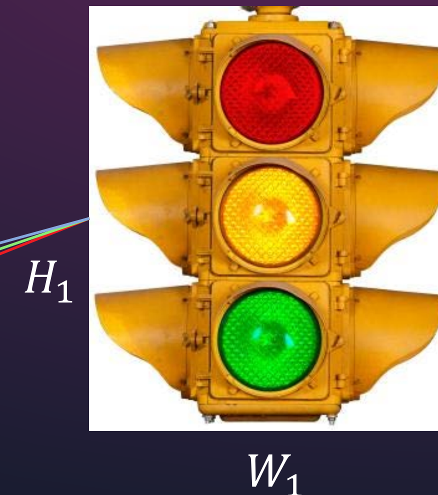
Convolution:
From Tensors View to Matrix View

© Copyright 2019 Xilinx

2D Convolution: Tensor View

Input Tensor $\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

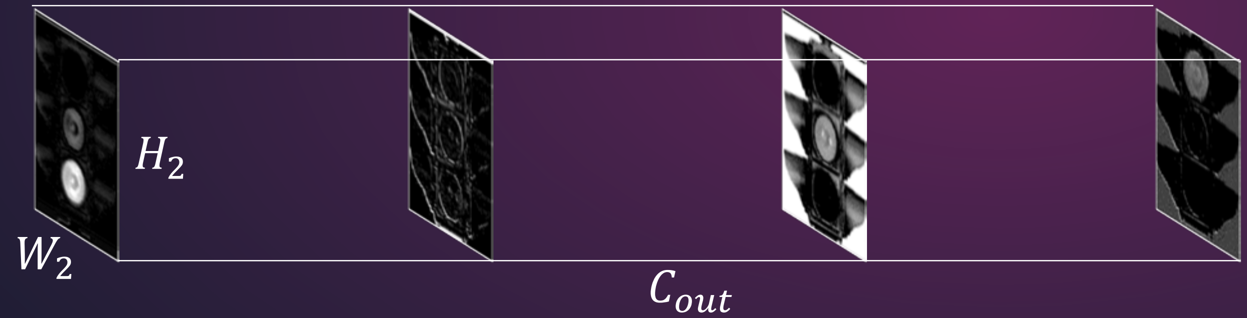




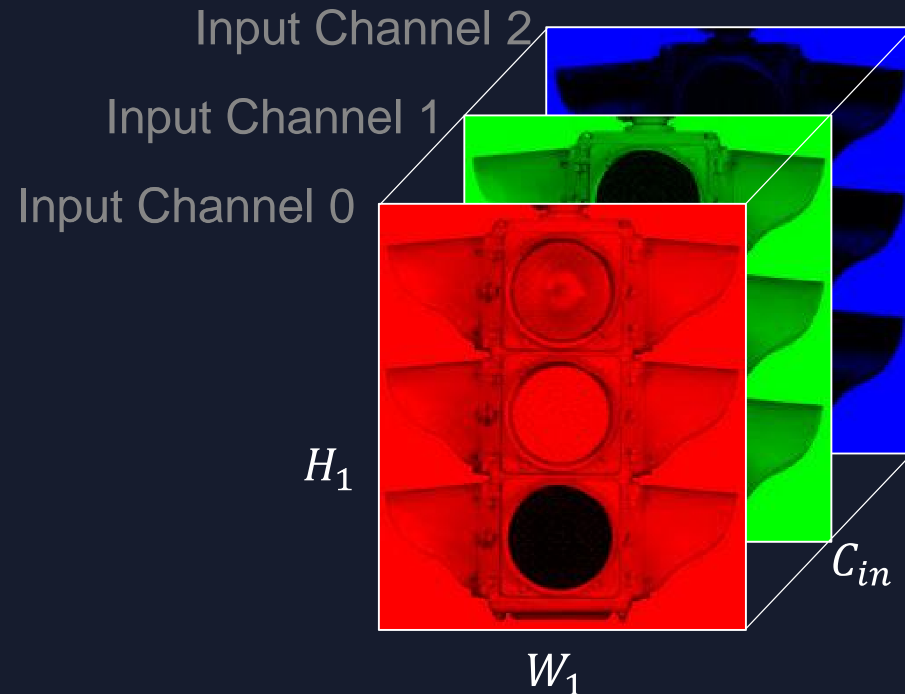
© Copyright 2019 Xilinx

2D Convolution: Tensor View

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



Input Tensor $\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

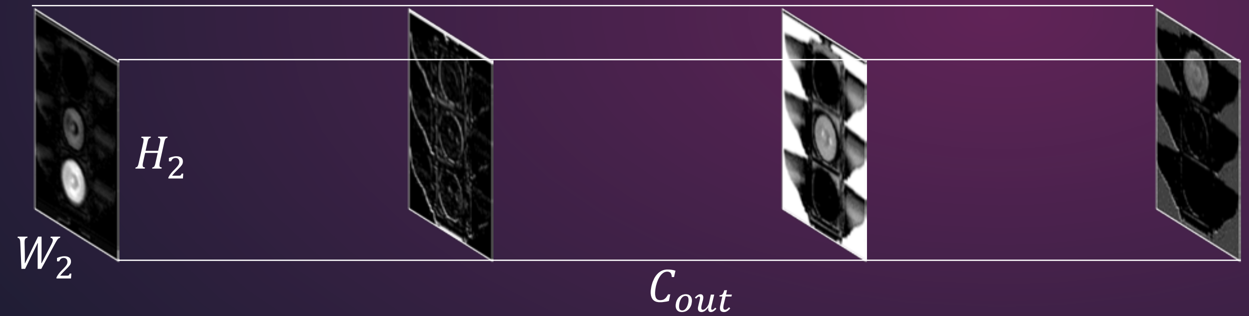


© Copyright 2019 Xilinx

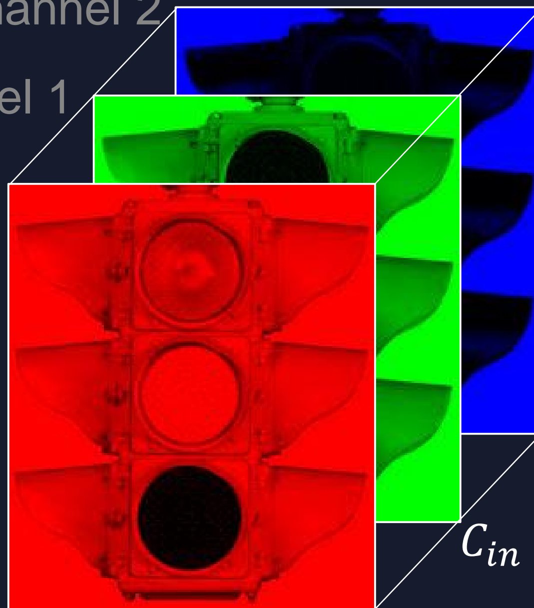
XILINX

2D Convolution: Tensor View

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



Input Tensor $\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$

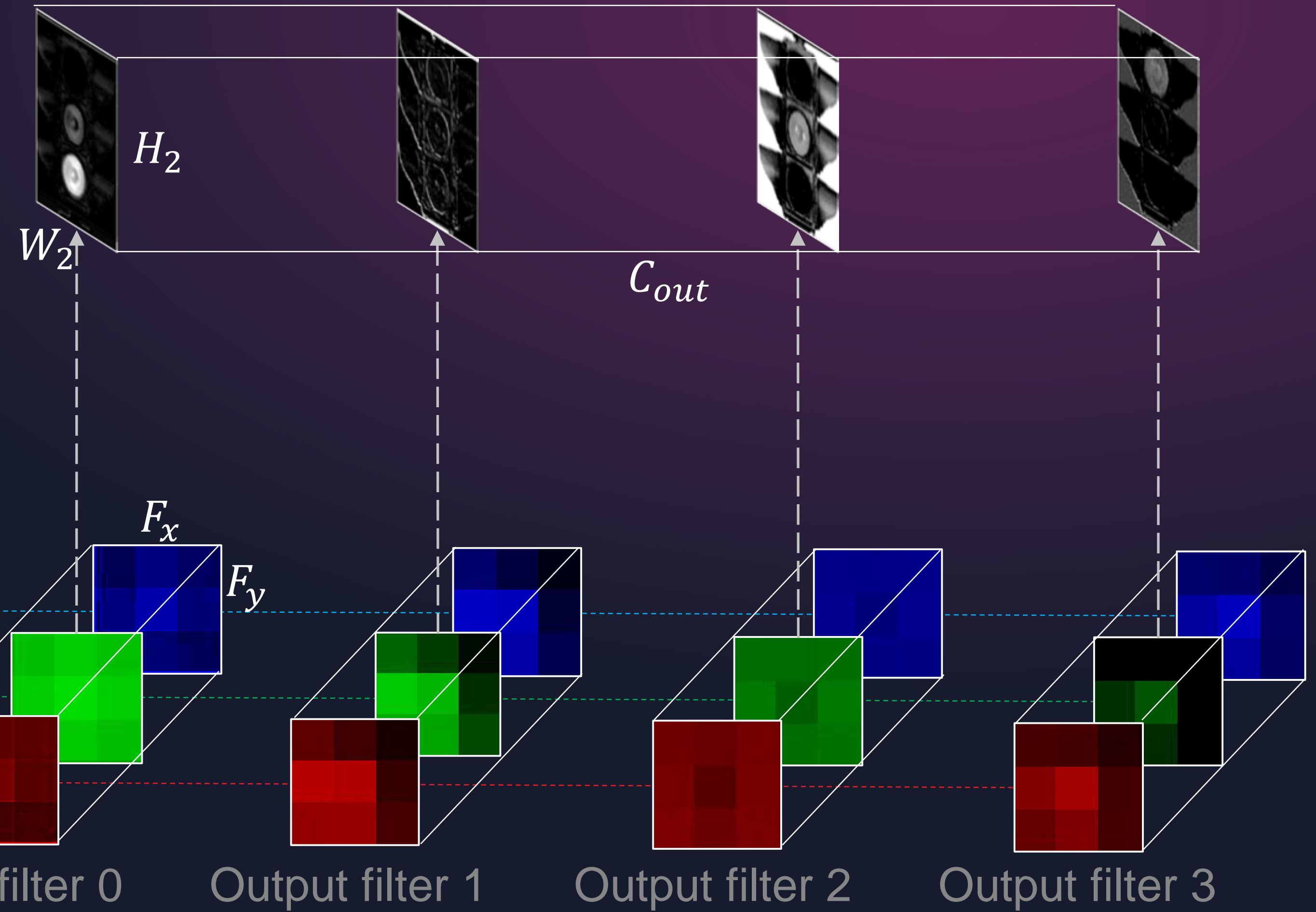


- **Although input and output tensors have three axes, convolution is 2D, not 3D.**
- **There's a 2D convolution filter mask for every input-output channel pair (3 × 4 = 12 in this example).**

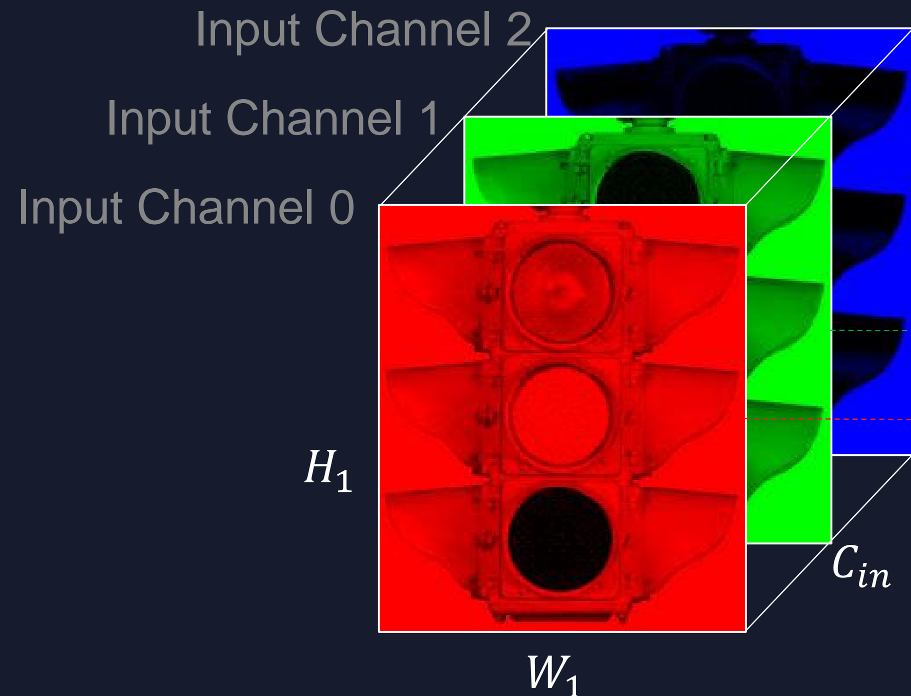
© Copyright 2019 Xilinx

2D Convolution: Tensor View

Output Tensor $\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



Input Tensor $\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$



Filter Weight Tensor $\mathcal{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}}$

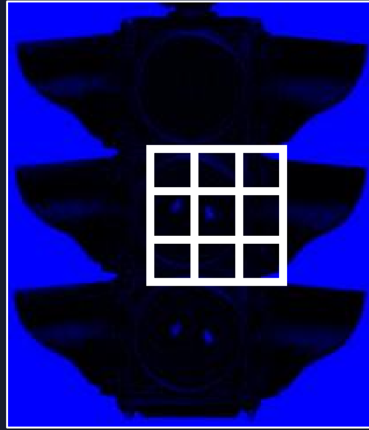
© Copyright 2019 Xilinx

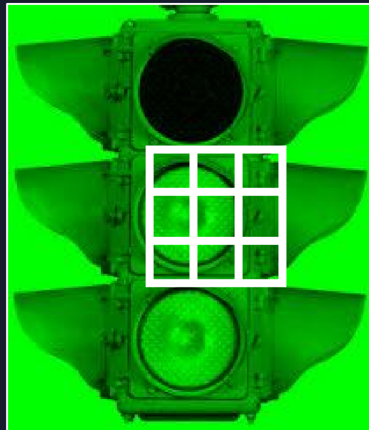
XILINX

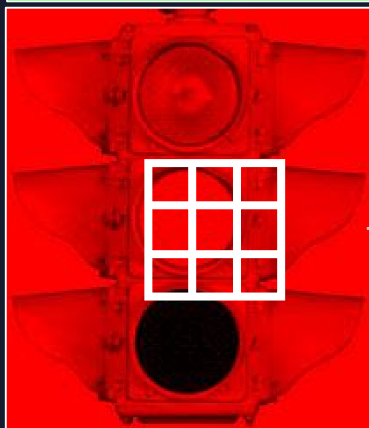
2D Convolution: From Tensor to Flat View

Input Image

Input Channels







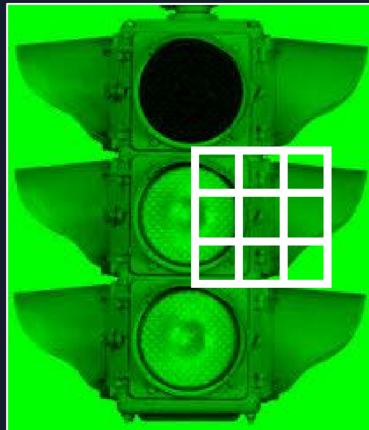
Parallel convolution sliding windows, one per input channel

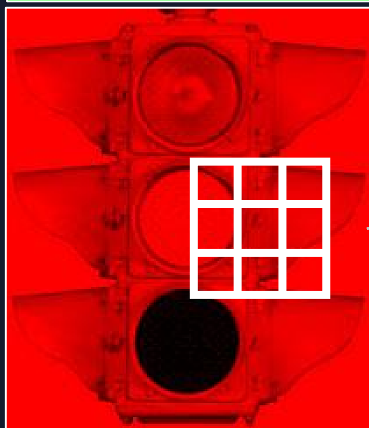
© Copyright 2019 Xilinx

2D Convolution: Flat View

Input Image

Input Channels





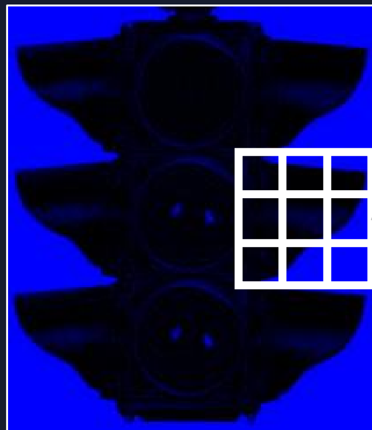
Parallel convolution sliding windows, one per input channel

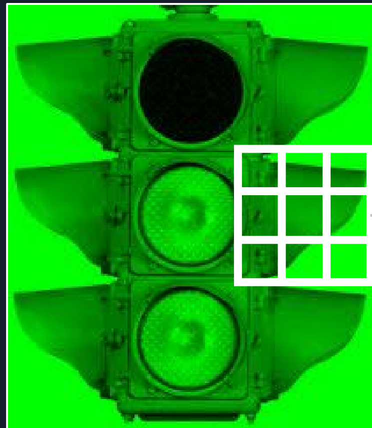
© Copyright 2019 Xilinx

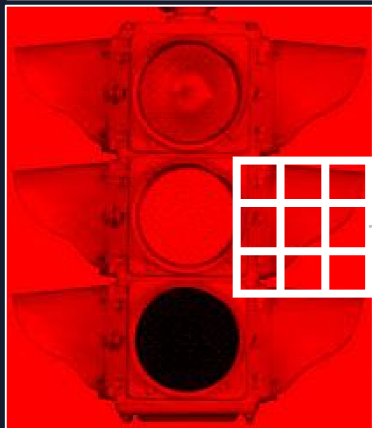
2D Convolution: Flat View

Input Image

Input Channels







Parallel convolution sliding windows, one per input channel

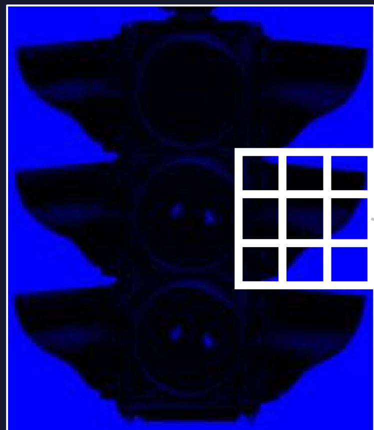
© Copyright 2019 Xilinx

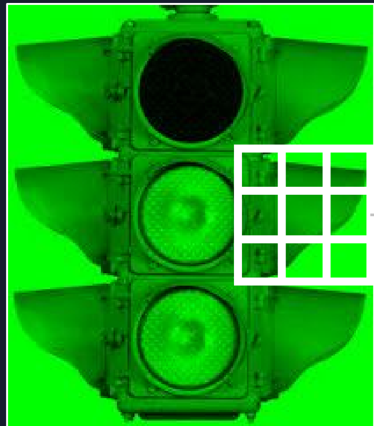
2D Convolution: Flat Silicon View

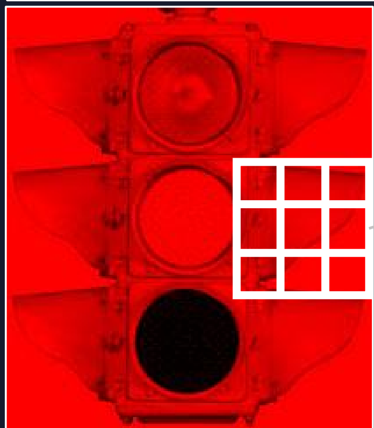
- Input channel broadcast
- Per-channel element-wise multiplication with filter weights

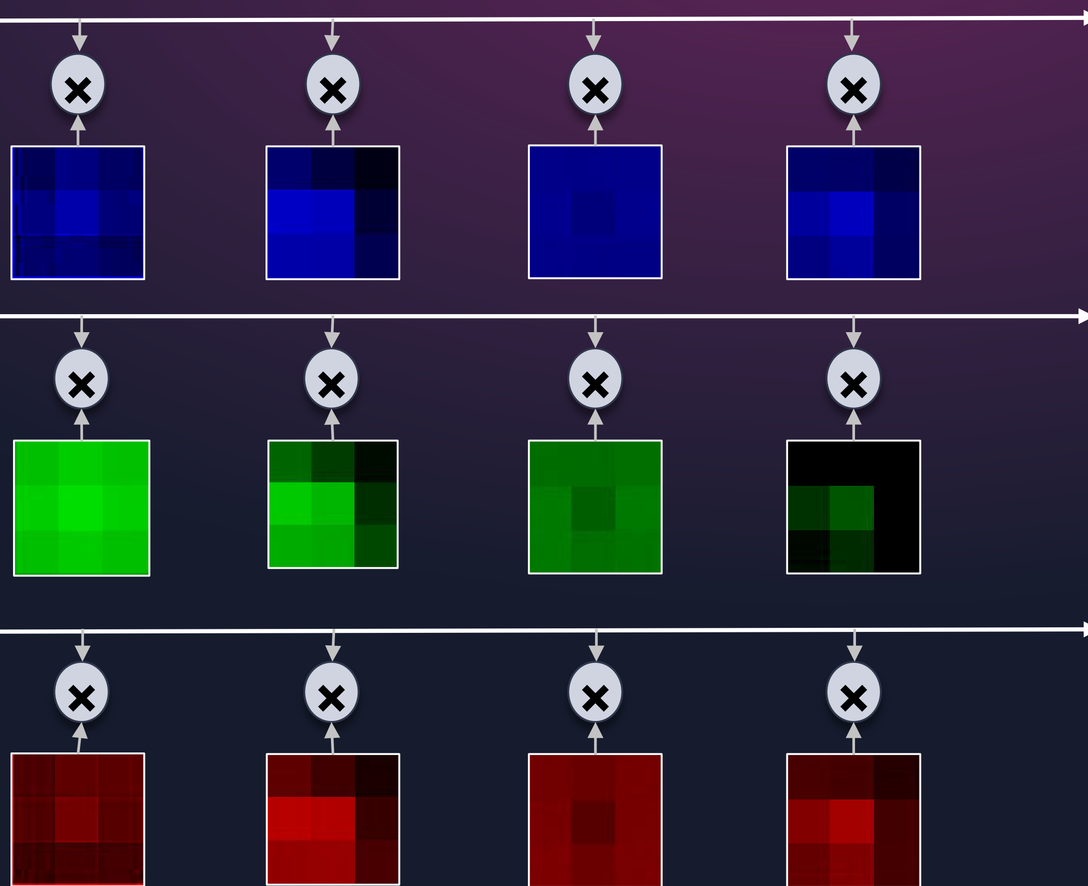
Input Image

Input Channels









© Copyright 2019 Xilinx

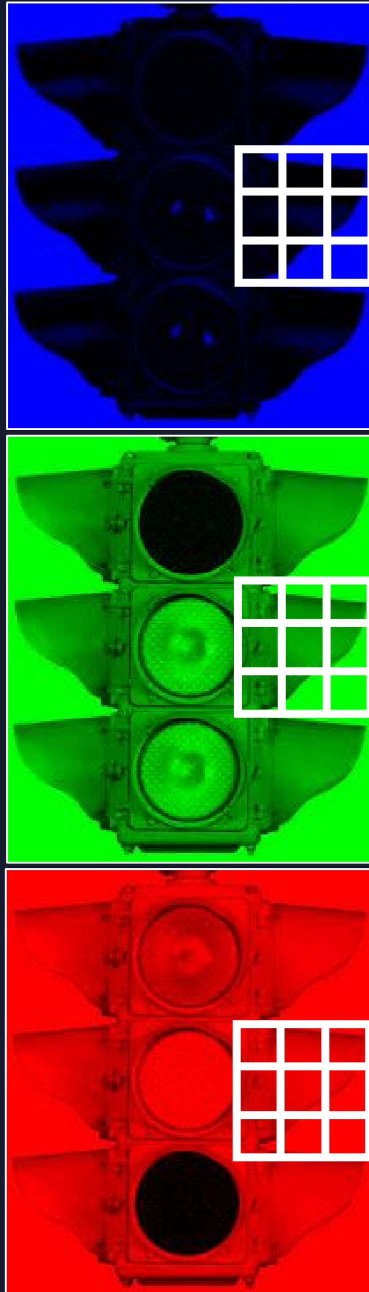
XILINX

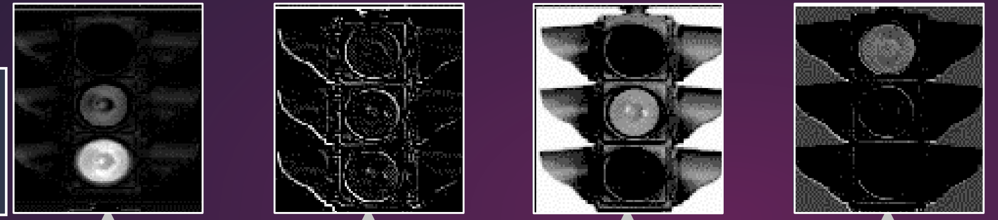
2D Convolution: Flat Silicon View

Reduction per output channel

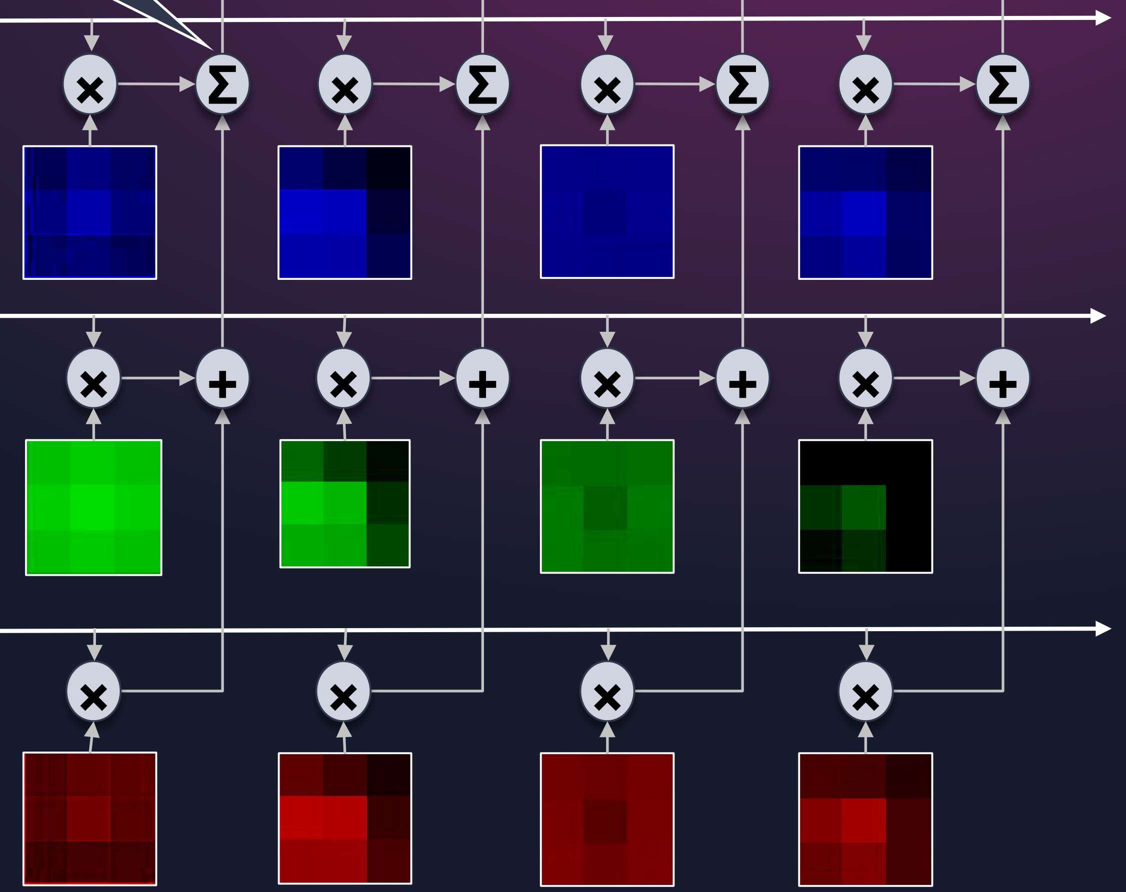
Input Image

Input Channels





Accumulates
$3 \times 3 \times 3 = 27$ products

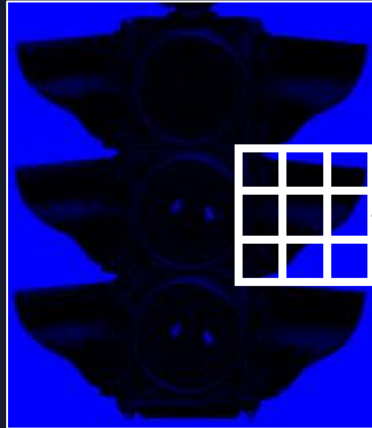


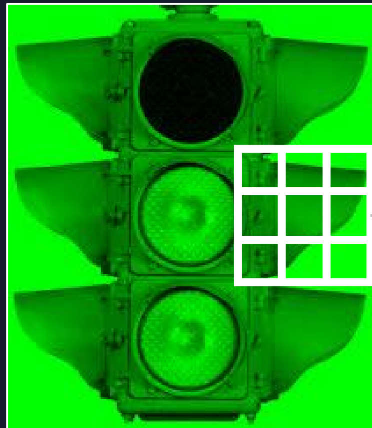
© Copyright 2019 Xilinx

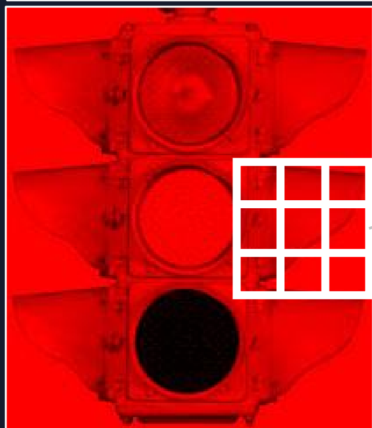
Interpreting Weights (Sort of)

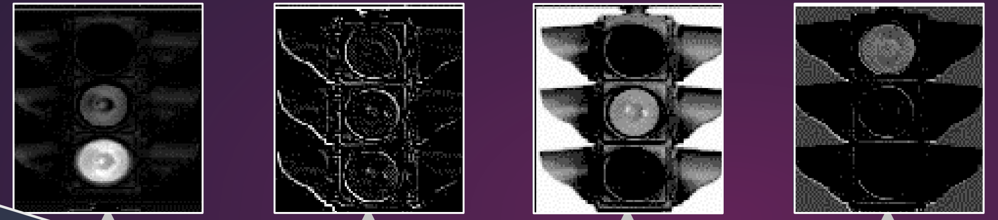
Input Image

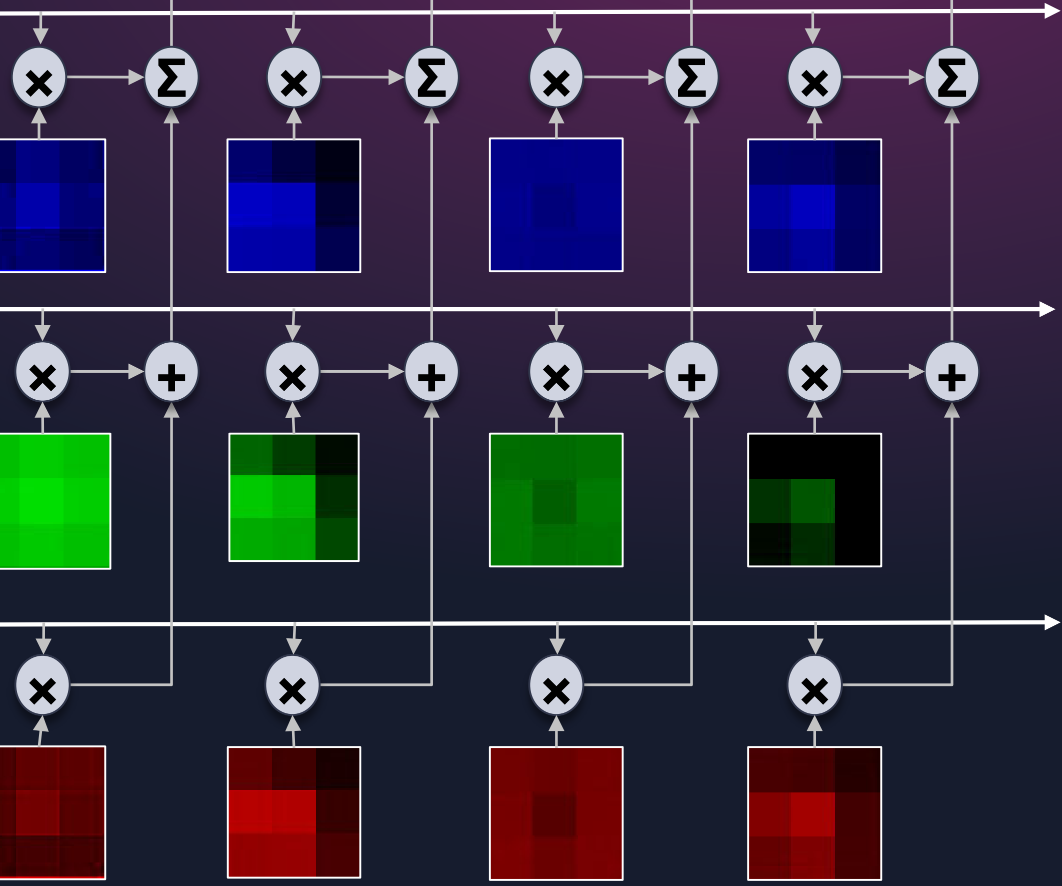
Input Channels











© Copyright 2019 Xilinx

XILINX

Computing Convolution: Tensor Matricization and MxV

© Copyright 2019 Xilinx

Computing Convolution

- For each element in some output channel
  - Input vector: Flattened input patches
  - Weight vector: Flattened filter weights for this output channel
  - Compute dot product between input vector and weight vector
- Input vector broadcast to multiple output channels

→ Hardware matrix-vector multiplier (MxV)

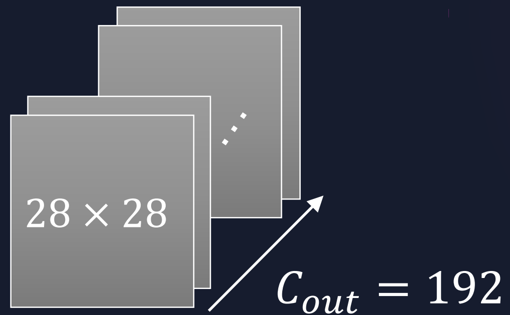
© Copyright 2019 Xilinx

XILINX

Reshaping Tensors to Matrices

3-Mode Output Tensor

$\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



Elements per Output Channel

$N = 28 \times 28 = 784$

Output channels

$M = 192$

Output Matrix

$Y \in \mathbb{R}^{C_{out} \times H_2 W_2}$

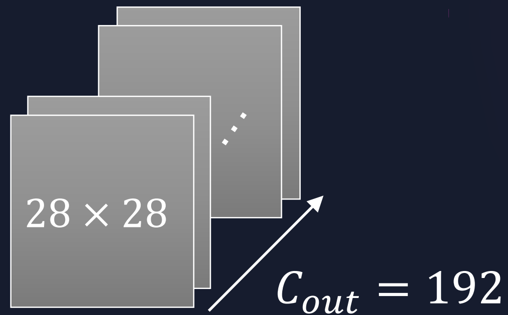
© Copyright 2019 Xilinx

XILINX

Reshaping Tensors to Matrices

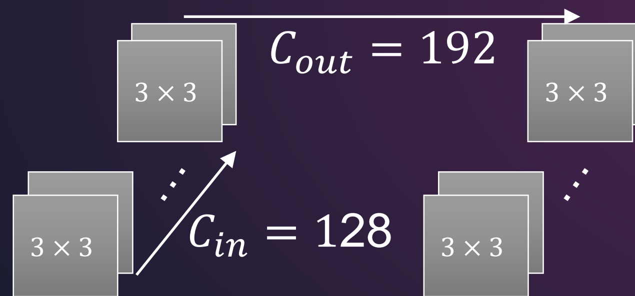
3-Mode Output Tensor

$\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



4-Mode Weight Tensor

$\mathcal{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}}$



Elements per Output Channel

$N = 28 \times 28 = 784$

Output channels

$M = 192$

Output Matrix

$Y \in \mathbb{R}^{C_{out} \times H_2 W_2}$

Input Channels × Weights per Filter

$K = 3 \times 3 \times 128 = 1152$

$M = 192$

Weight Matrix

$W \in \mathbb{R}^{C_{out} \times F_y F_x C_{in}}$

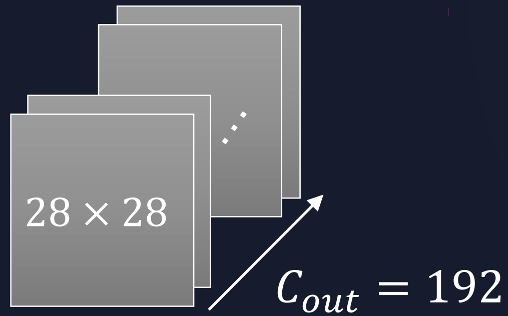
© Copyright 2019 Xilinx

XILINX

Reshaping Tensors to Matrices

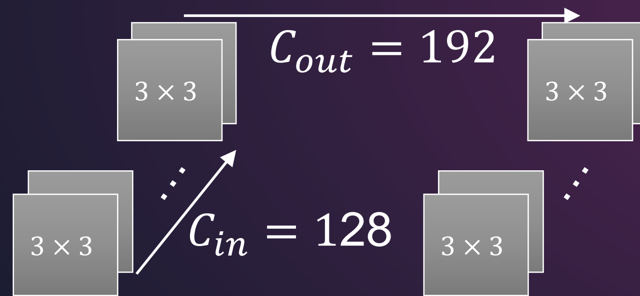
3-Mode Output Tensor

$\mathcal{Y} \in \mathbb{R}^{H_2 \times W_2 \times C_{out}}$



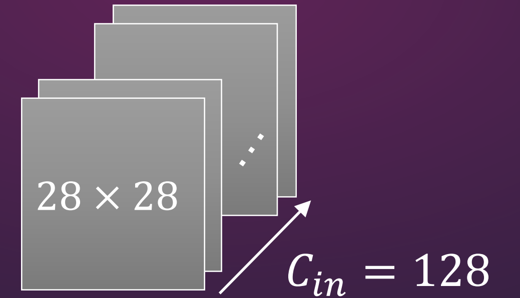
4-Mode Weight Tensor

$\mathcal{W} \in \mathbb{R}^{F_y \times F_x \times C_{in} \times C_{out}}$



3-Mode Input Tensor

$\mathcal{X} \in \mathbb{R}^{H_1 \times W_1 \times C_{in}}$



Elements per Output Channel

$N = 28 \times 28 = 784$

Output channels

$M = 192$

Output Matrix

$Y \in \mathbb{R}^{C_{out} \times H_2 W_2}$

=

Input Channels × Weights per Filter

$K = 3 \times 3 \times 128 = 1152$

$M = 192$

Weight Matrix

$W \in \mathbb{R}^{C_{out} \times F_y F_x C_{in}}$

$N = 28 \times 28 = 784$

$K = 1152$

Input Matrix

$X \in \mathbb{R}^{F_y F_x C_{in} \times H_2 W_2}$

© Copyright 2019 Xilinx

Example Weight Tensor Matricization Using Input Channel Interleaving

Input channel R

1	4	7
2	5	8
3	6	9

Input channel G

1	4	7
2	5	8
3	6	9

Input channel B

1	4	7
2	5	8
3	6	9

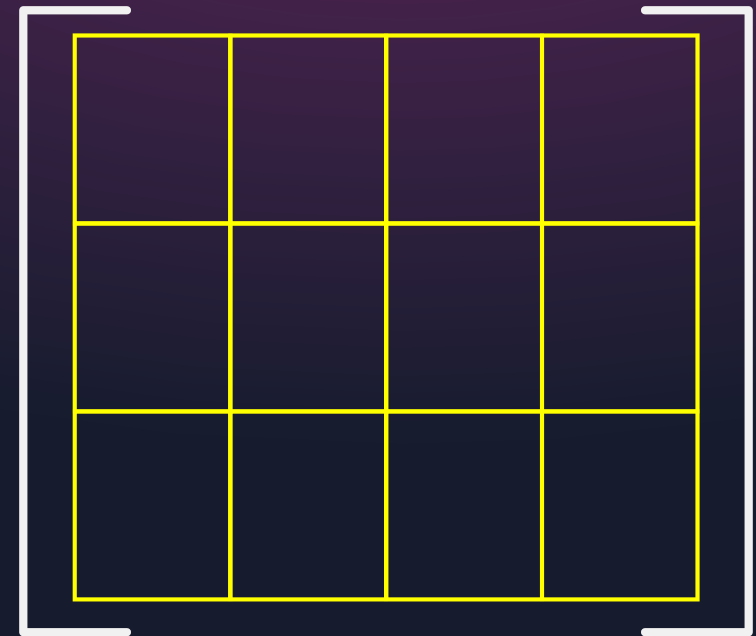
Ordering:
Weight column, weight row, input channel

© Copyright 2019 Xilinx

XILINX

Distributing Compute Over Hardware MxV Array

1. Divide weight matrix into blocks.
2. Copy weight block matrices into MxV array.
3. Divide input matrix into column vectors.
4. Run matrix-vector multiply-accumulate.

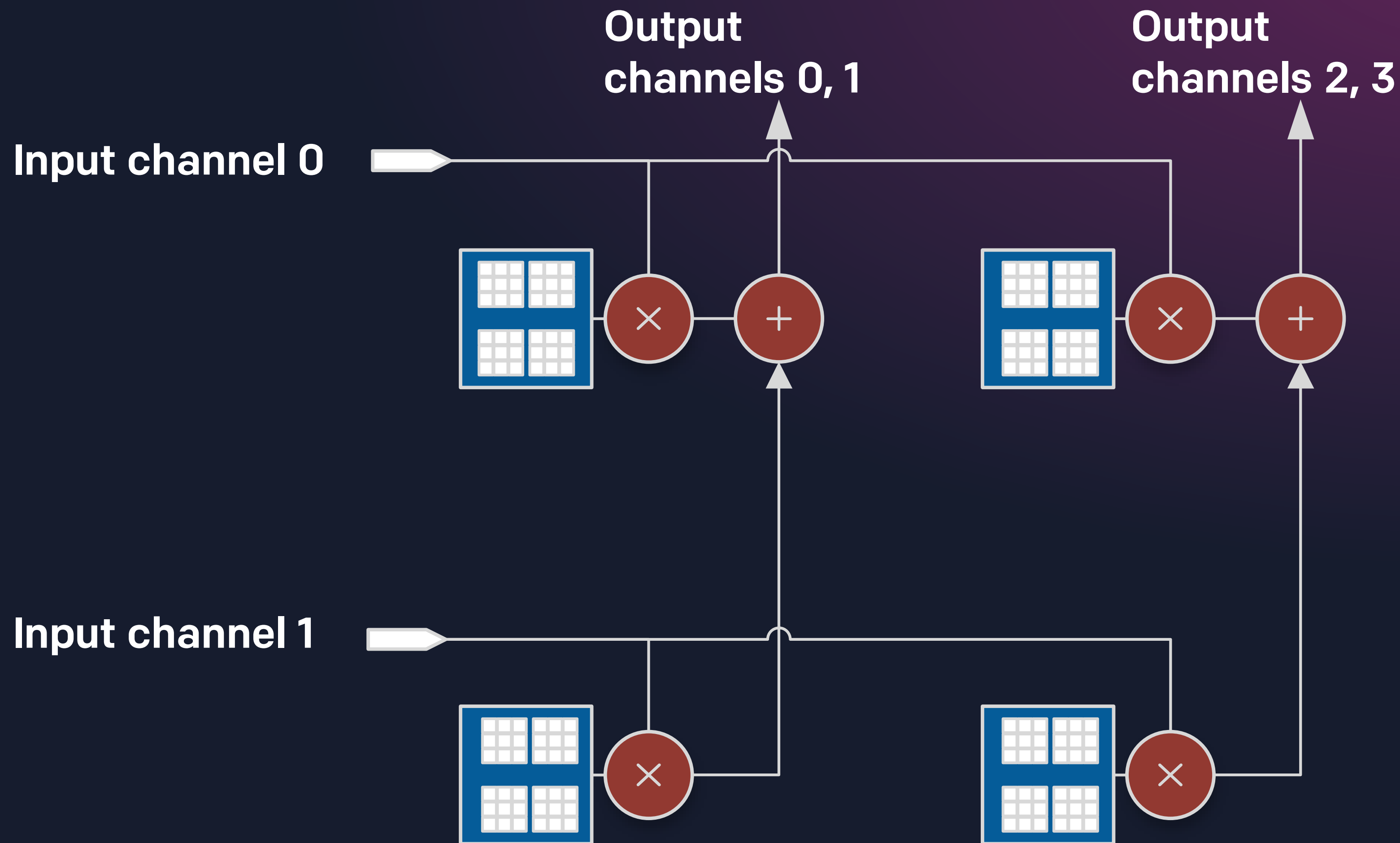


© Copyright 2019 Xilinx

Maximizing Array Compute Efficiency

Assign input channels to MxV input lanes.

Assign output channels to MxV output lanes.

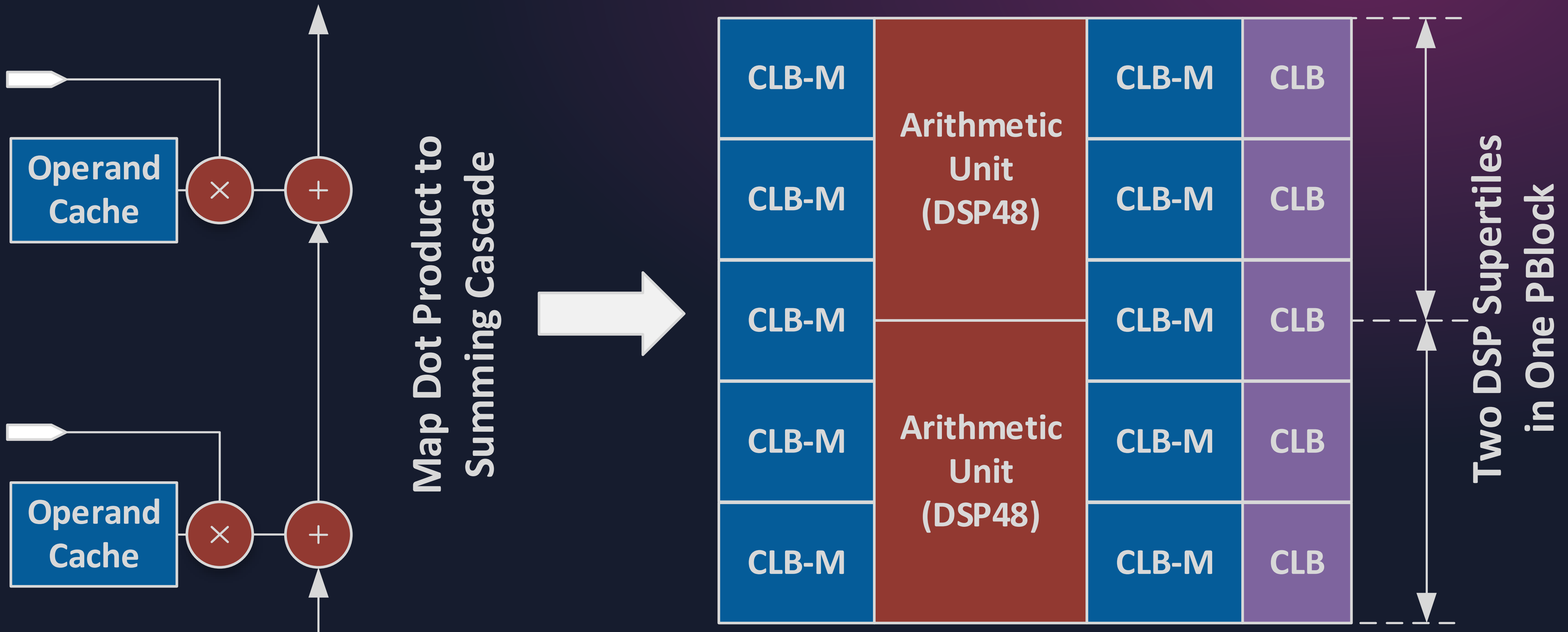


© Copyright 2019 Xilinx

Maximize Clock Rate: Use DSP Supertiles [FPL '17]

DSP supertile column: 4 physically adjacent columns good for timing closure

Use DSP summing cascades instead of summing trees

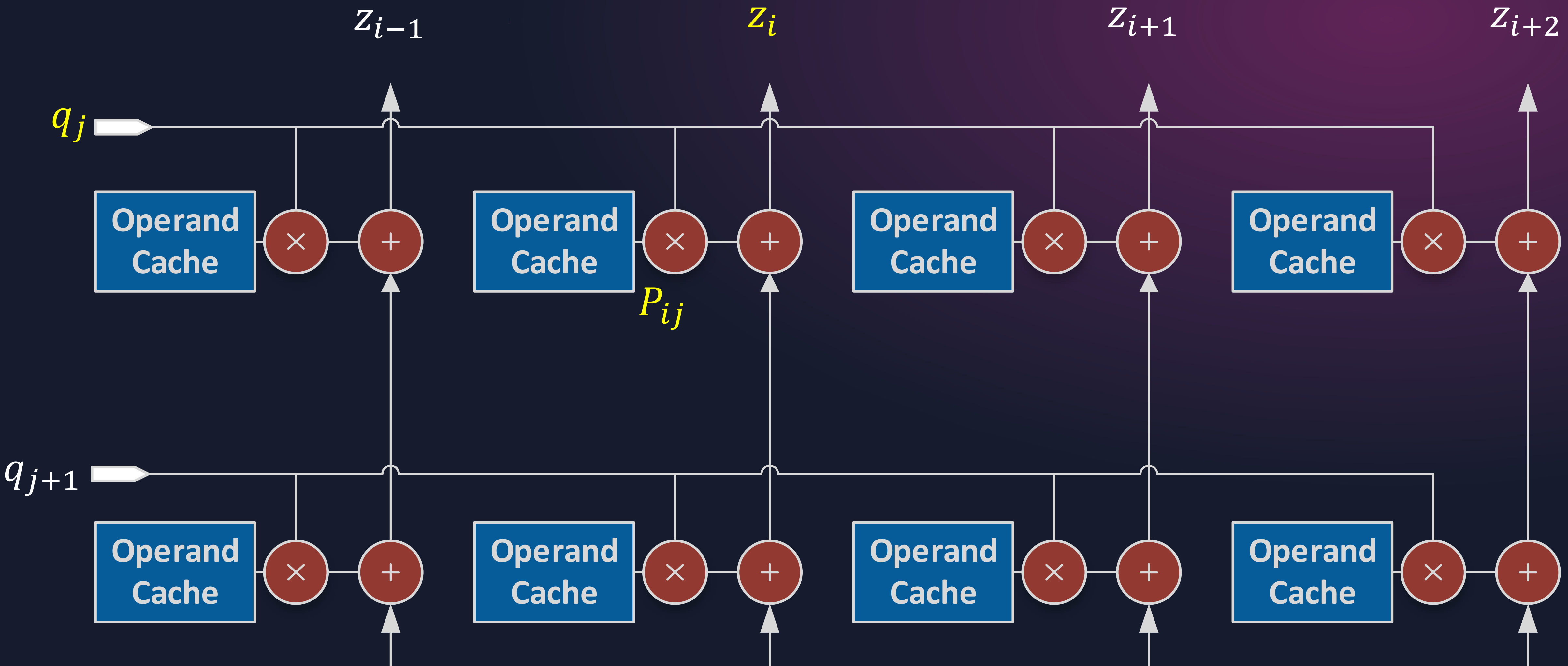


© Copyright 2019 Xilinx

XILINX

Maximize Clock Rate: Use DSP Supertiles

DSP Supertile Array = Matrix-Vector Multiplier: $z \leftarrow Pq + r$



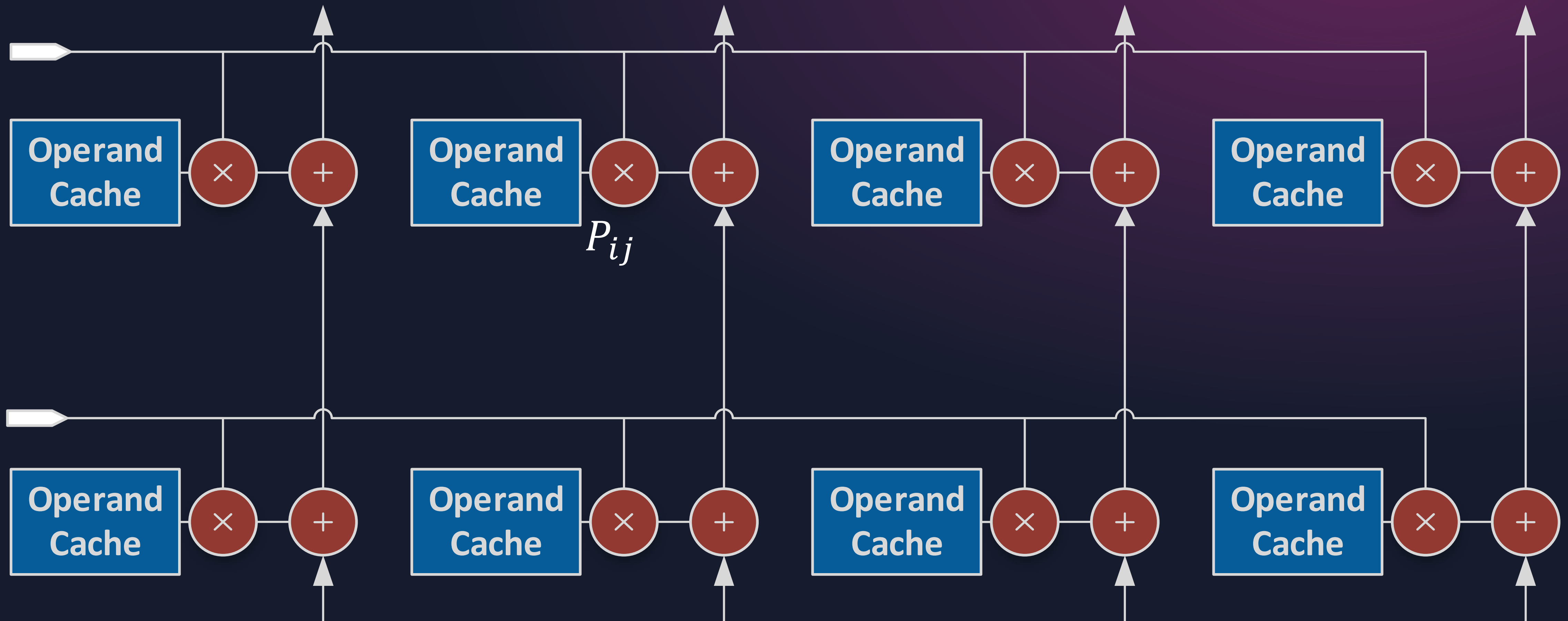
© Copyright 2019 Xilinx

XILINX

Maximize Clock Rate: Use DSP Supertiles

Double-buffered operand cache in distributed RAM stores filter weights

Distributed RAM can run as fast as DSP tiles



© Copyright 2019 Xilinx

Example: Time Interleaving Four Output Channels

3 × 3 Conv2D
Stride 1

Input feature reused for 4 cycles.

Input channel address (green box) moves ¼ as fast as weight address (red box).

© Copyright 2019 Xilinx

Mapping GoogLeNet to a Processor Chain

© Copyright 2019 Xilinx

GoogLeNet

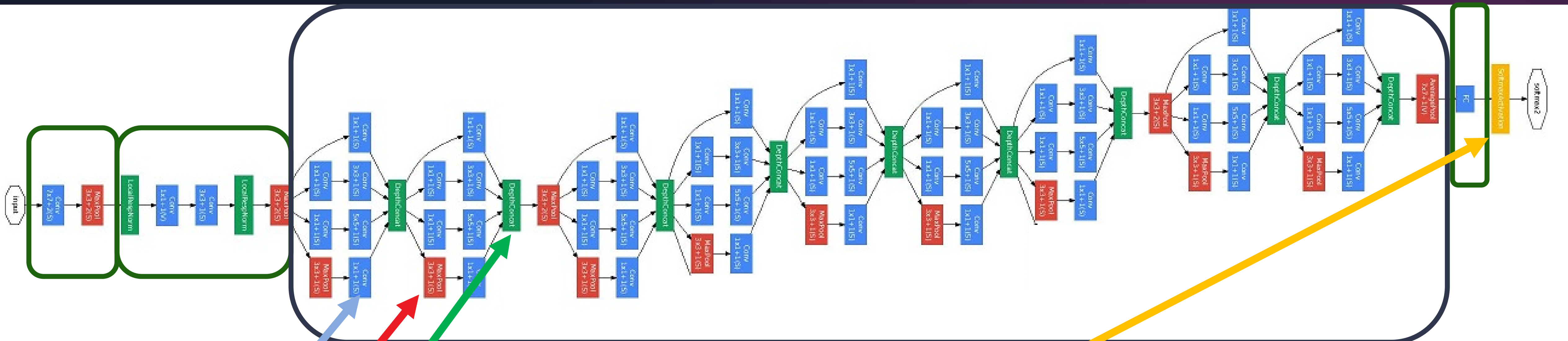


Image in

1000 classes out

Blue: 2D convolution or fully-connected plus activation (ReLU)
Red: Max-pooling layer
Green: Concatenation layer
Yellow: Softmax

Inception v1 graph from https://arxiv.org/pdf/1409.4842.pdf

© Copyright 2019 Xilinx

XILINX

Mapping GoogLeNet to a Four-Processor Chain

P1

P2

P3

P4

Softmax

Input bias

© Copyright 2019 Xilinx

One Chain on Each VU9P Super Logic Region (SLR)

There are 3 SLRs on each VU9P.

Each SLR contains a 4-processor chain.

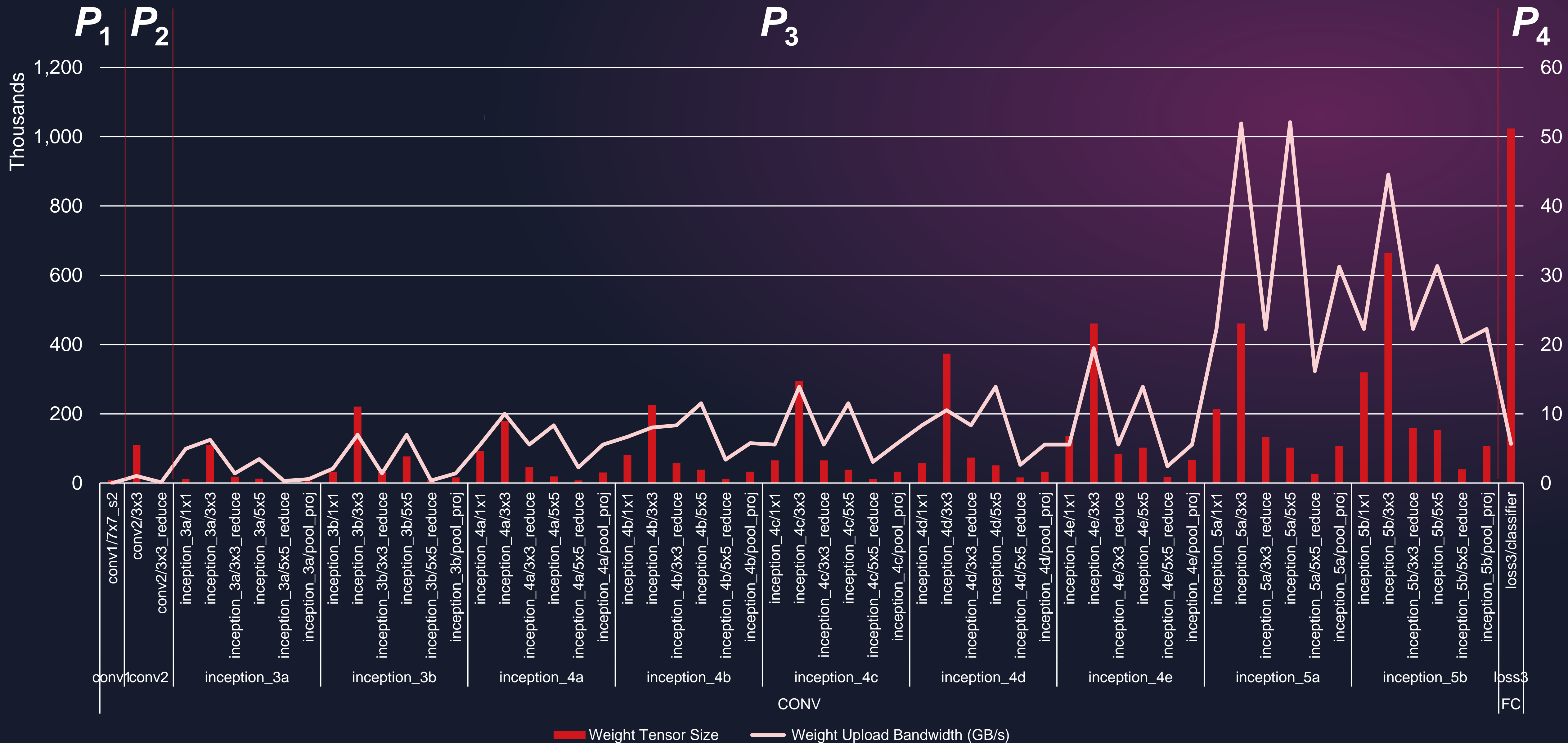
Each chain is independent and has it own set of weights.

Each processor in the chain is event-driven (w/o central controller).

Processor	DSP Supertile Array Input Lanes N_1	DSP Supertile Array Output Lanes N_2	Linear Layers in GoogLeNet	Cycles Per Image
P_1	21	8	1	691792
P_2	32	16	2	686432
P_3	96	16	54	708671
P_4	8	1	1	128000

© Copyright 2019 Xilinx

Fast and Big Memory Wanted

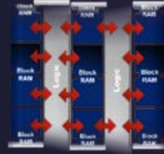


© Copyright 2019 Xilinx

Keep All Tensors On-Die in UltraRAM and BRAM

Bandwidth

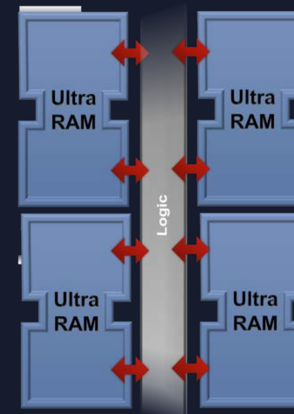
Distributed RAM
(bits to kilobits)



Block RAM
(10s of megabits)

MxV doesn’t have to wait for memory

New in UltraScale+™



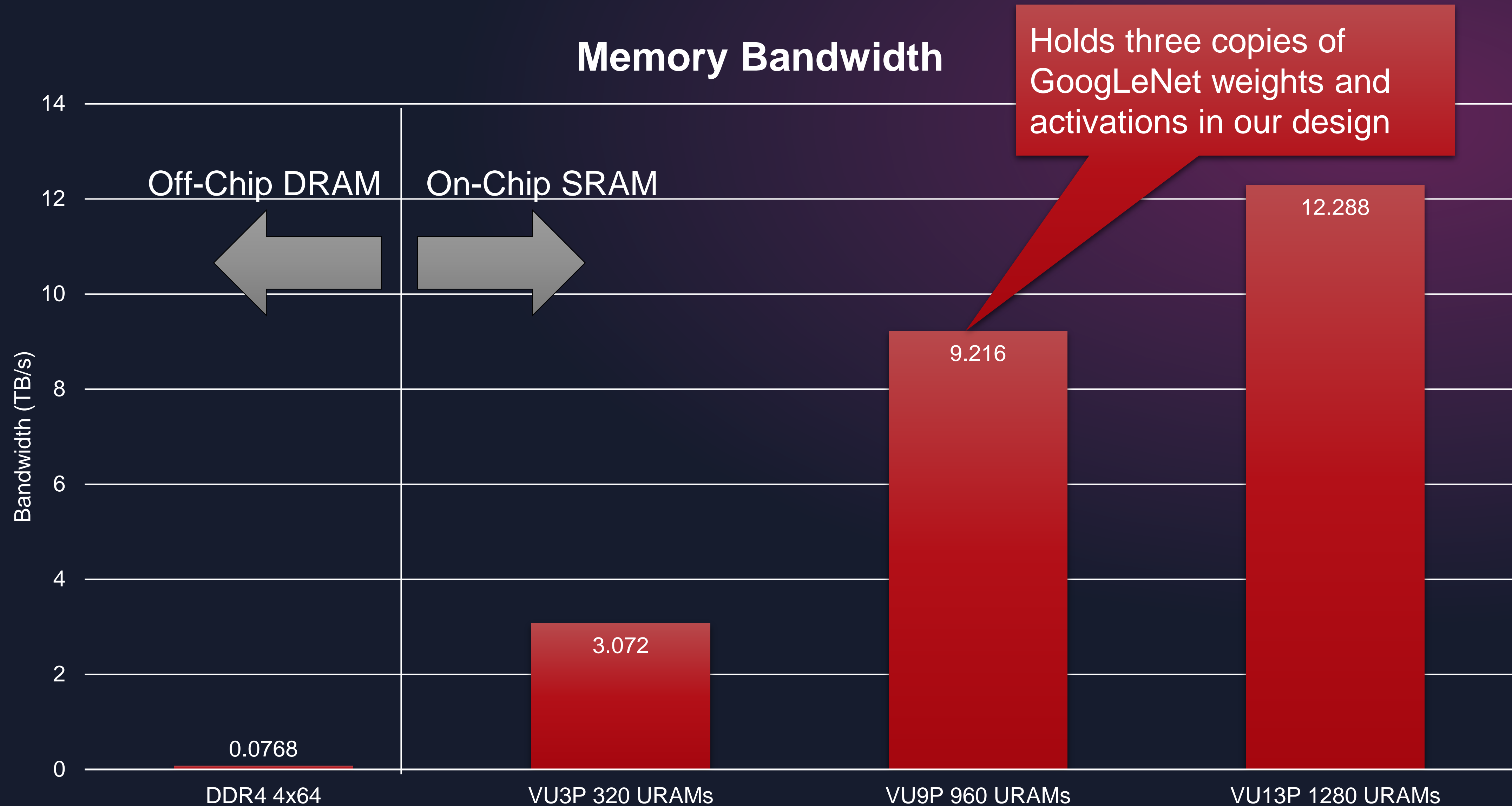
UltraRAM
(100s of megabits)

External DDR DRAM
(10s of gigabits)

Capacity

© Copyright 2019 Xilinx

UltraRAM vs. DDR4 Bandwidth



© Copyright 2019 Xilinx

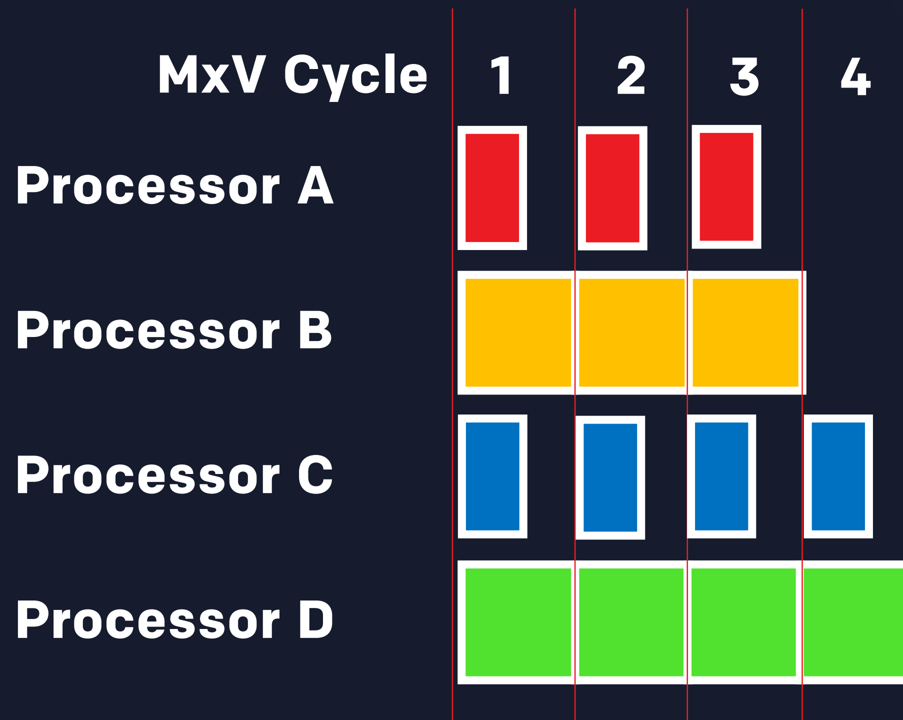
XILINX

Two Components of Compute Efficiency

Balance factor **is a "hurry-up-and-wait" metric.**

Distribution efficiency **measures multiply-add cycle utilization.**

Compute Efficiency **= Balance Factor × Distribution Efficiency**



Balance Factor	Distribution Efficiency
Low	Low
Low	High
High	Low
High	High

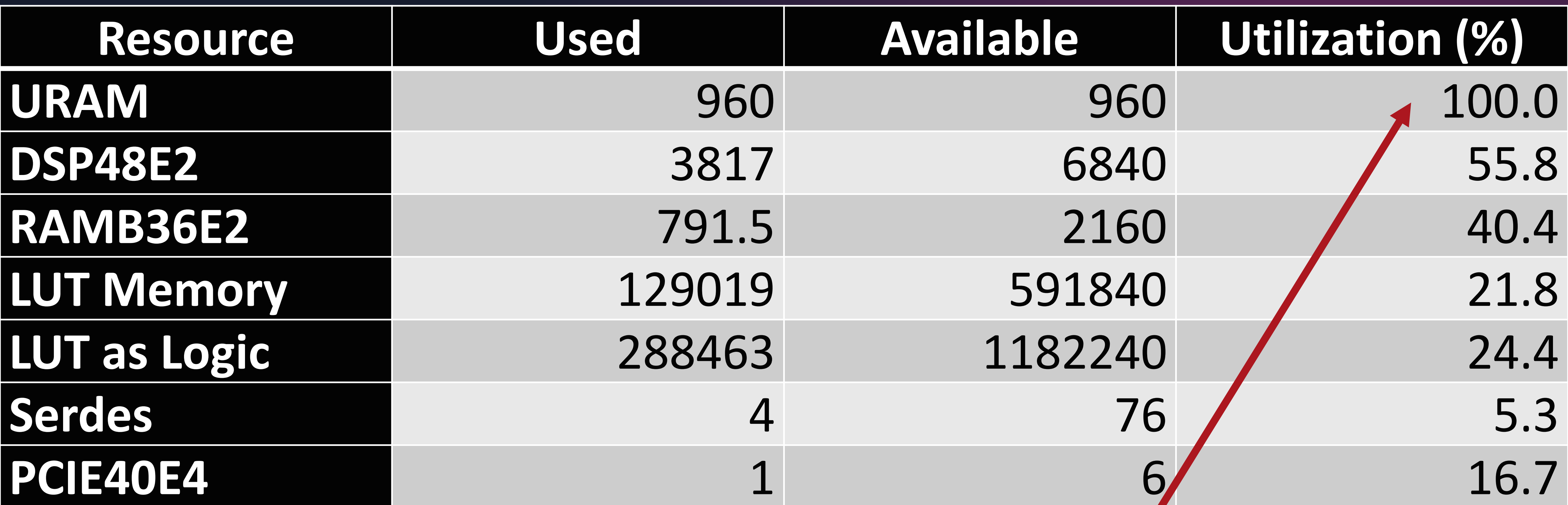
© Copyright 2019 Xilinx

GoogLeNet Efficiency Metrics (ImageNet 1K Dataset)

Processor	MxV Shape	Cycles Per Image	Balance Factor	Distribution Efficiency	Compute Efficiency
P_1	21×8	691792	97.6%	100.0%	**97.6%**
P_2	32×16	686432	96.9%	100.0%	**96.9%**
P_3	96×16	708671	100.0%	95.2%	**95.2%**
P_4	8×1	128000	18.1%	100.0%	**18.1%**
Pipeline		**708671**	**98.8%**	**96.7%**	**95.5%**

© Copyright 2019 Xilinx

Resource Utilization



Resource	Used	Available	Utilization (%)
URAM	960	960	100.0
DSP48E2	3817	6840	55.8
RAMB36E2	791.5	2160	40.4
LUT Memory	129019	591840	21.8
LUT as Logic	288463	1182240	24.4
Serdes	4	76	5.3
PCIE40E4	1	6	16.7

Can close timing despite 100% URAM utilization due to simple SRAM-DSP interconnections

© Copyright 2019 Xilinx

XILINX

Summary

- Make compute fast: Use DSP supertile arrays

- Keep compute busy
  - Store tensors in SRAM (UltraRAM and BRAM)
  - Map convolution channels to MxV ports
  - Configure MxV shapes to maximize pipeline balance factors

- Simplify implementation
  - Use straight routes. (Recall the flat view.)
  - Don't build convoluted memory -compute networks
  - Use temporal locality to reduce tensor memory read bandwidth
  - Run the rest of the design at half the DSP clock rate

© Copyright 2019 Xilinx