HybridDNN: A Framework for High-Performance Hybrid DNN Accelerator Design and Implementation

Hanchen Ye[1], Xiaofan Zhang[1], Zhize Huang[2],
Gengsheng Chen[2], Deming Chen[1]
[1]University of Illinois at Urbana-Champaign,
[2]Fudan University

ILLINOIS
Electrical & Computer Engineering
GRAINGER COLLEGE OF ENGINEERING

FUDAN UNIVERSITY

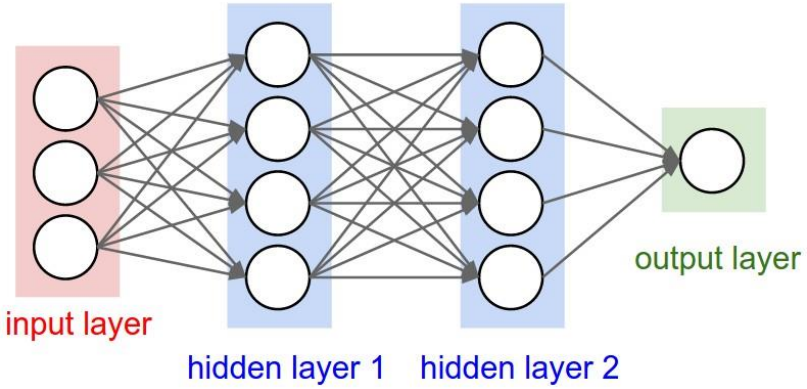
Hanchen Ye

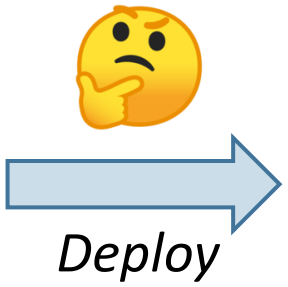
- I’m a PhD student in University of Illinois at Urbana-Champaign (UIUC), advised by Prof. Deming Chen. I obtained my Bachelor and Master degree in Fudan University in 2017 and 2019, respectively. My research interests lie in the area of Hardware Acceleration, High-Level Synthesis (HLS), and Deep Learning.
- Personal website: hanchenye.com/about/

Outline

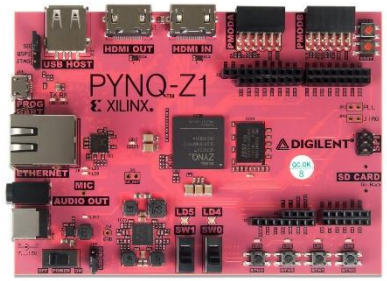
- Motivation
- HybridDNN Framework
- Accelerator Design
- Design Space Exploration
- Experimental Results
- Conclusion

Motivation (1)





Cloud FPGAs



Embedded FPGAs

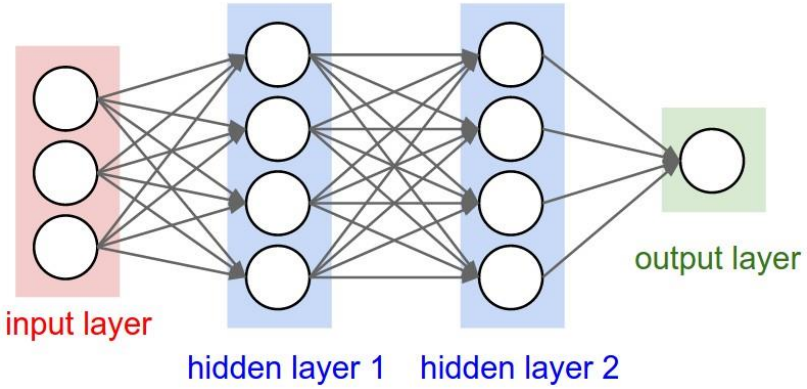
Goals

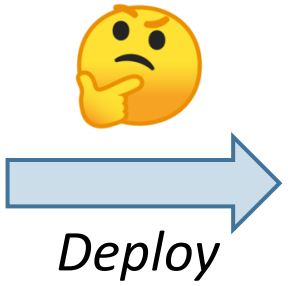
- Easy and fast deployment
- Flexibility regarding different applications / scenarios
- High performance & efficiency

Solution?

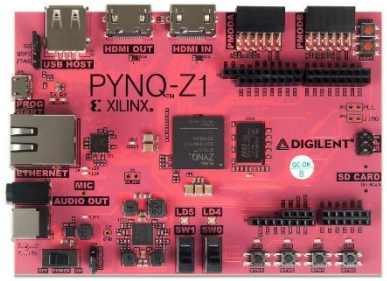
- Complete and automated design flow which is flexible for various DNNs and FPGAs
- Winograd fast algorithm

Motivation (1)





Cloud FPGAs



Embedded FPGAs

Goals

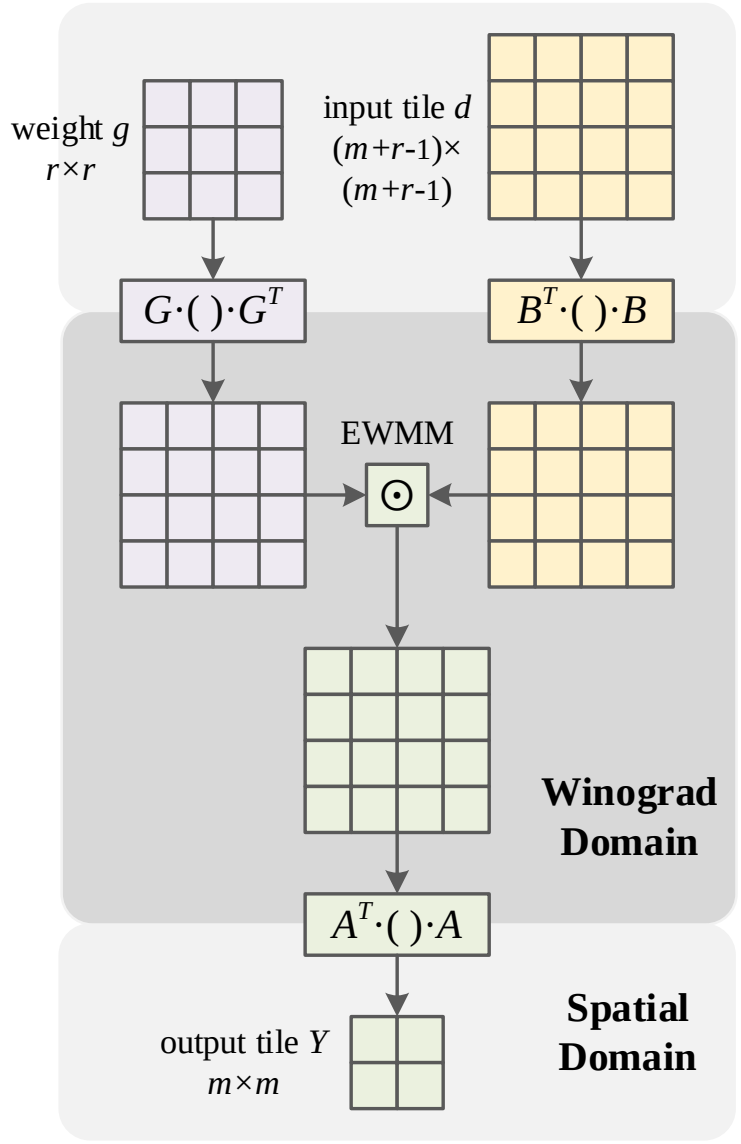
- Easy and fast deployment
- Flexibility regarding different applications / scenarios
- High performance & efficiency

Solution?

- Complete and automated design flow which is flexible for various DNNs and FPGAs
- Winograd fast algorithm

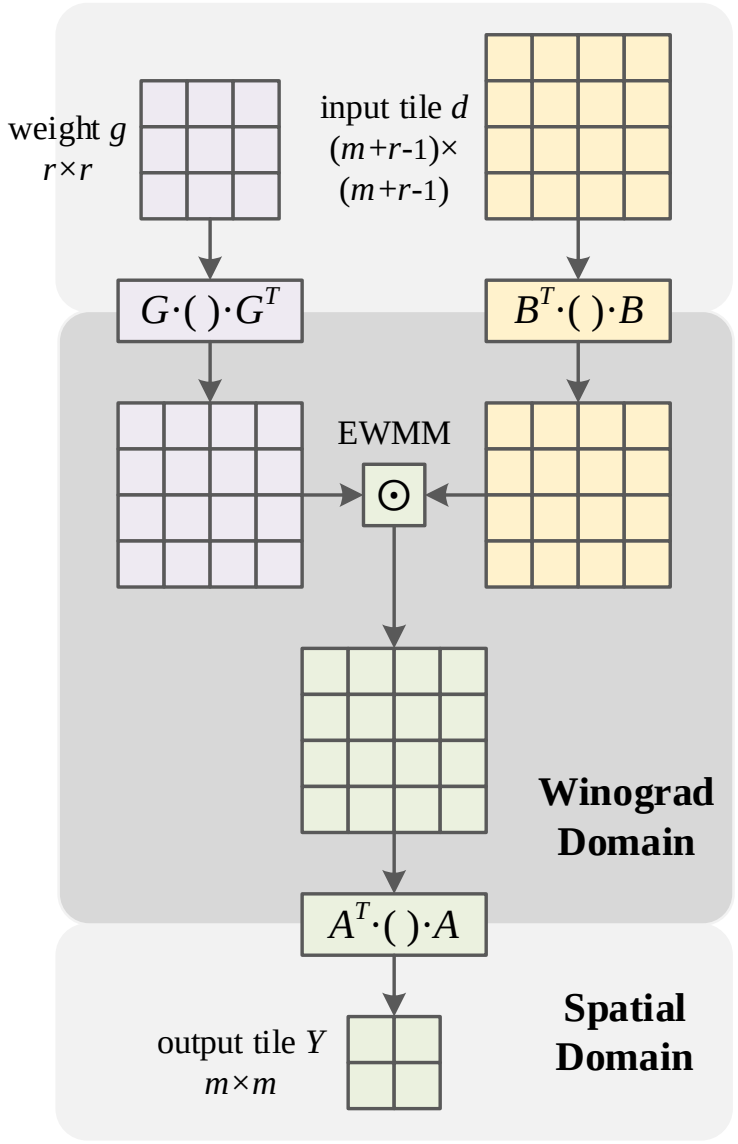
Motivation (2)

- Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T\left[[GgG^T] \odot [B^T dB]\right]A$
  - G, B, A: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)
- Pros:
  - Reduce MAC number of convolution by 2.25× ($m = 2, r = 3$) to 4× ($m = 4, r = 3$)
- Cons:
  - Not friendly to fully-connected (FC) layers, 1×1 convolutional (CONV) layers, and >1 stride size => **Low flexibility** for various DNNs
  - High memory bandwidth demand => **Low flexibility** for various FPGAs



Motivation (2)

- Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T\left[[GgG^T] \odot [B^T dB]\right]A$
  - G, B, A: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)
- Pros:
  - Reduce MAC number of convolution by $2.25\times$ ($m = 2, r = 3$) to $4\times$ ($m = 4, r = 3$)
- Cons:
  - Not friendly to fully-connected (FC) layers, 1×1 convolutional (CONV) layers, and >1 stride size => **Low flexibility** for various DNNs
  - High memory bandwidth demand => **Low flexibility** for various FPGAs



Motivation (3)

- Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T\left[[GgG^T] \odot [B^T dB]\right]A$
  - G, B, A: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)
- Pros:
  - Reduce MAC number of convolution by $2.25\times$ ($m = 2, r = 3$) to $4\times$ ($m = 4, r = 3$)
- Cons:
  - Not friendly to fully-connected (FC) layers, 1×1 convolutional (CONV) layers, and >1 stride size => **Low flexibility** for various DNNs
  - High memory bandwidth demand => **Low flexibility** for various FPGAs

Architecture?

× Homogeneous Winograd

✓**Hybrid Spatial / Winograd**

Motivation (3)

- Winograd Algorithm $F(m \times m, r \times r)$:
  - $Y = A^T\left[[GgG^T] \odot [B^T dB]\right]A$
  - G, B, A: Winograd transformation matrices
  - $\odot$: Element-Wise Matrix Multiply (EWMM)
- Pros:
  - Reduce MAC number of convolution by $2.25\times$ ($m = 2, r = 3$) to $4\times$ ($m = 4, r = 3$)
- Cons:
  - Not friendly to fully-connected (FC) layers, 1×1 convolutional (CONV) layers, and >1 stride size => **Low flexibility** for various DNNs
  - High memory bandwidth demand => **Low flexibility** for various FPGAs

Architecture?

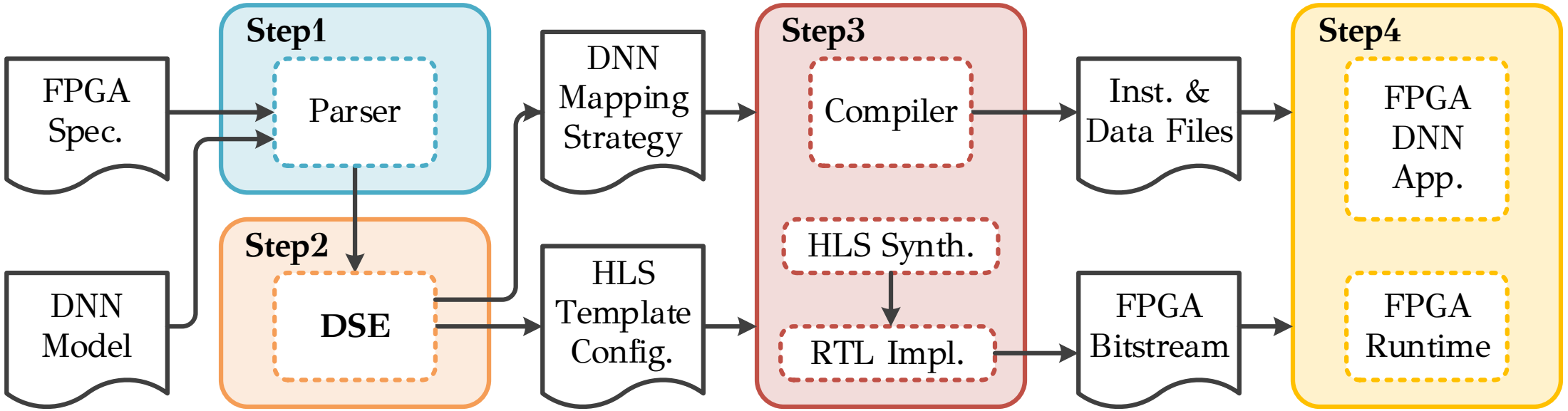
× Homogeneous Winograd

✓**Hybrid Spatial / Winograd**

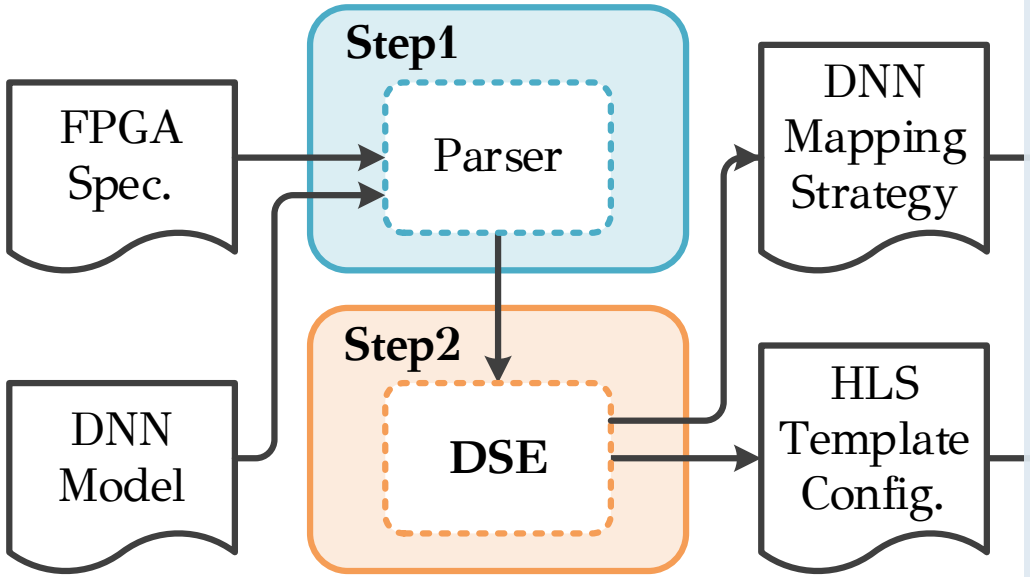
Challenges

- Accelerator design:
  - Computation resource reuse
  - Memory management
  - Inter-layer context switch
  - … …
- Large design space
  - Modeling
  - Exploration

HybridDNN Framework (1)



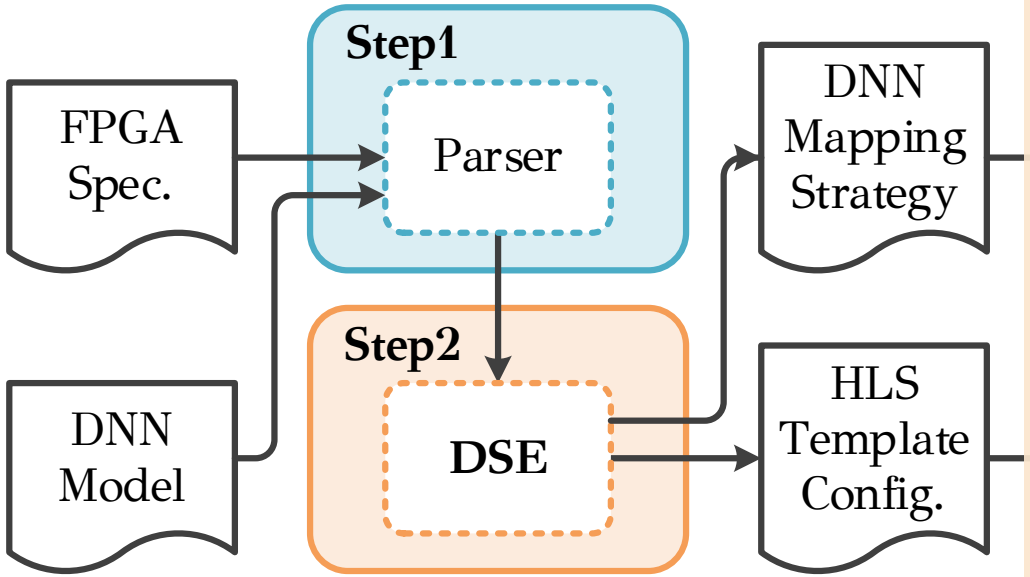
HybridDNN Framework (2)



Input of Step1 & 2

- FPGA Specification:
  - On-chip resources (LUT, DSP, and BRAM)
  - External memory bandwidth
  - Number of dies (for cloud FPGA)
- DNN Model:
  - DNN architecture description
  - Pre-trained weights

HybridDNN Framework (3)



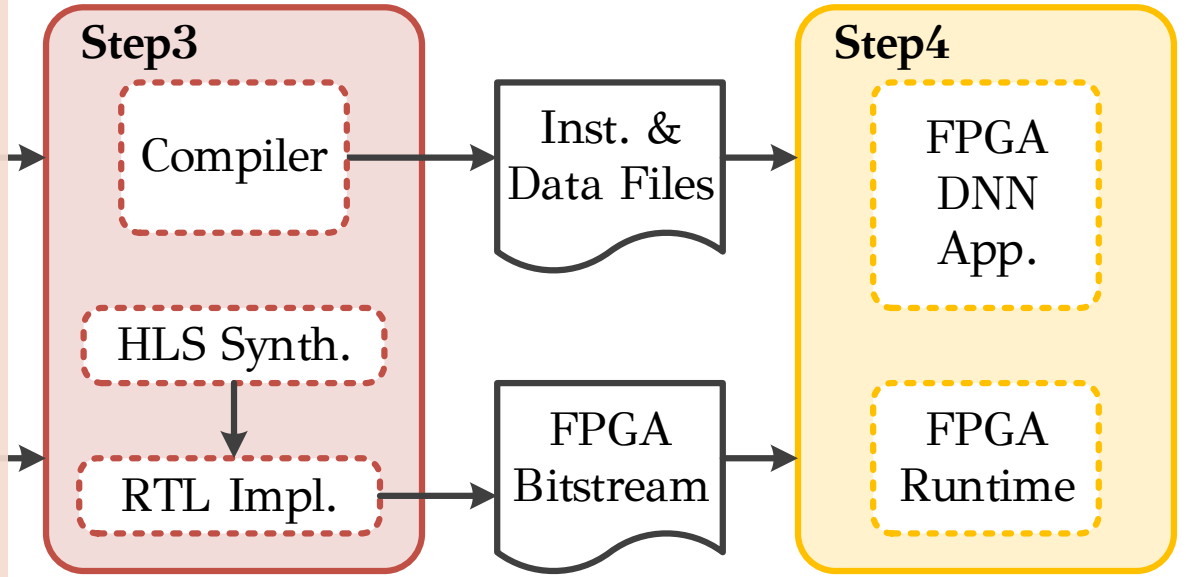
Output of Step1 & 2

- DNN Mapping Strategy:
  - Dataflow, CONV mode of each layer
  - Partition strategy of each layer
- HLS Template Configuration:
  - Parallel factors PI, PO, and PT
  - Number of instances NI

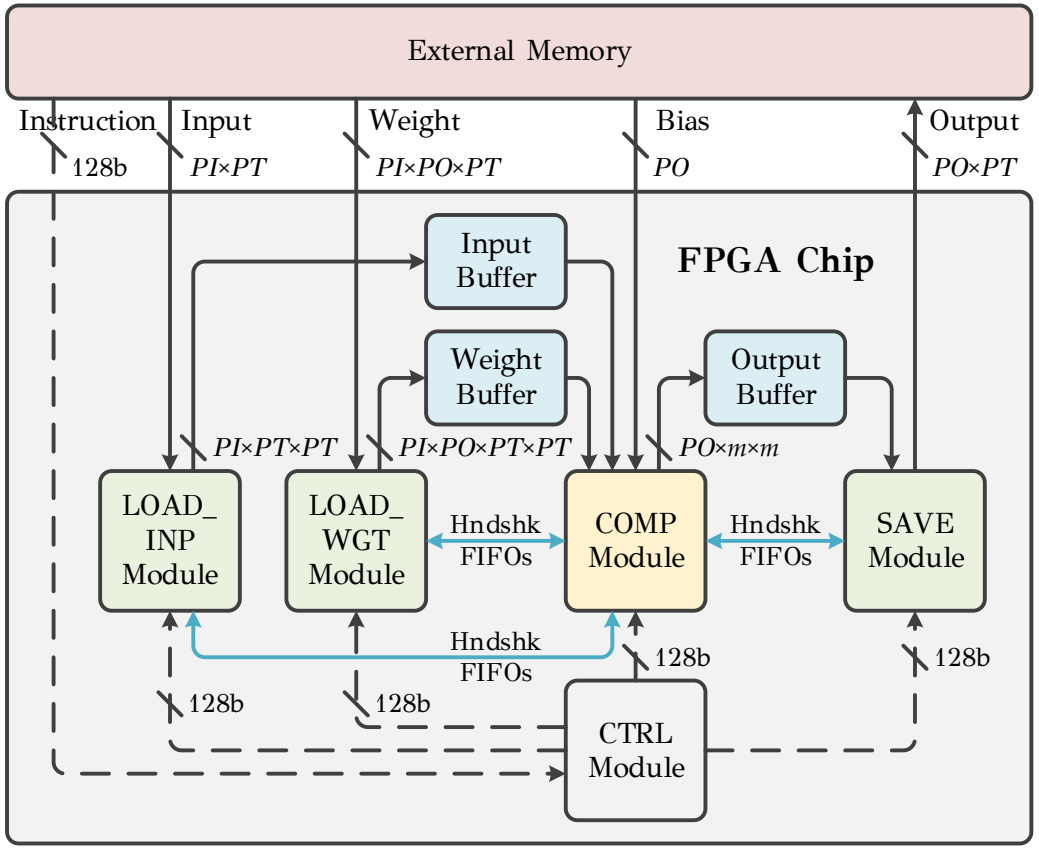
HybridDNN Framework (4)

Output of Step3

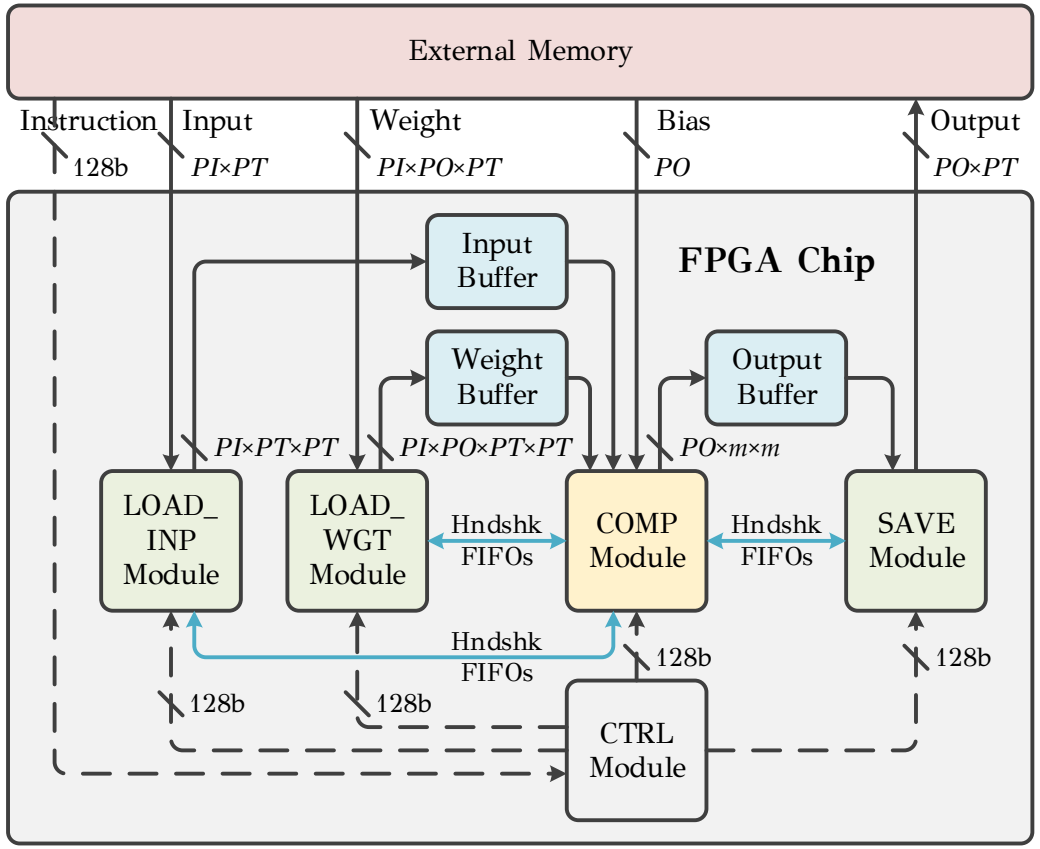
- Instructions & Data Files:
  - Instructions (.bin)
  - Reordered weights data (.bin)
  - Host executable file
- FPGA Bitstream



Accelerator Design (1)



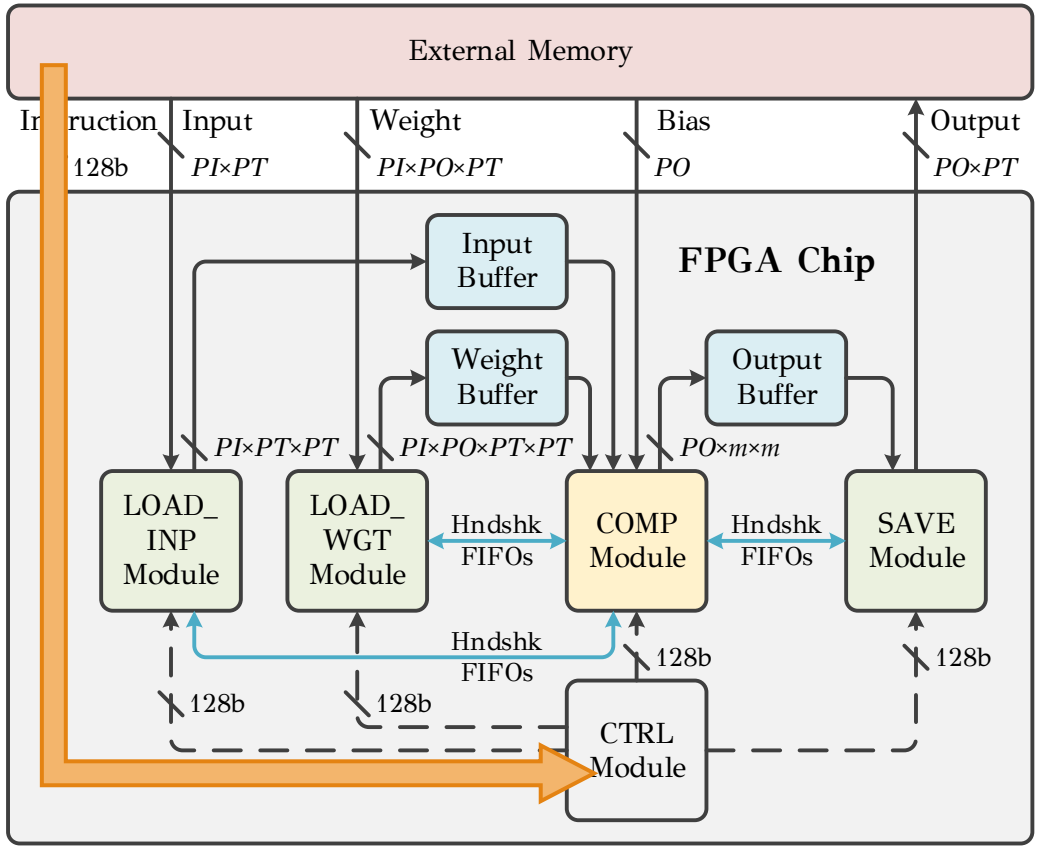
Accelerator Design (1)



Efficient and Flexible

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules

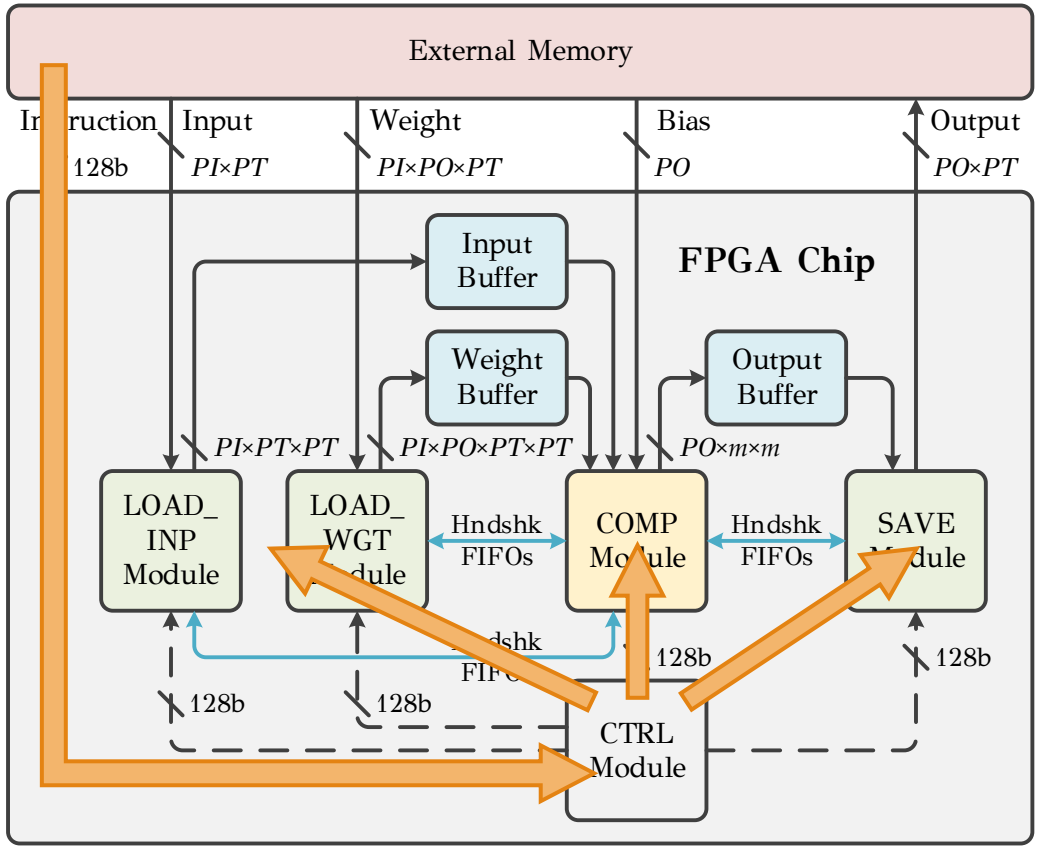
Accelerator Design (1)



Efficient and Flexible

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules

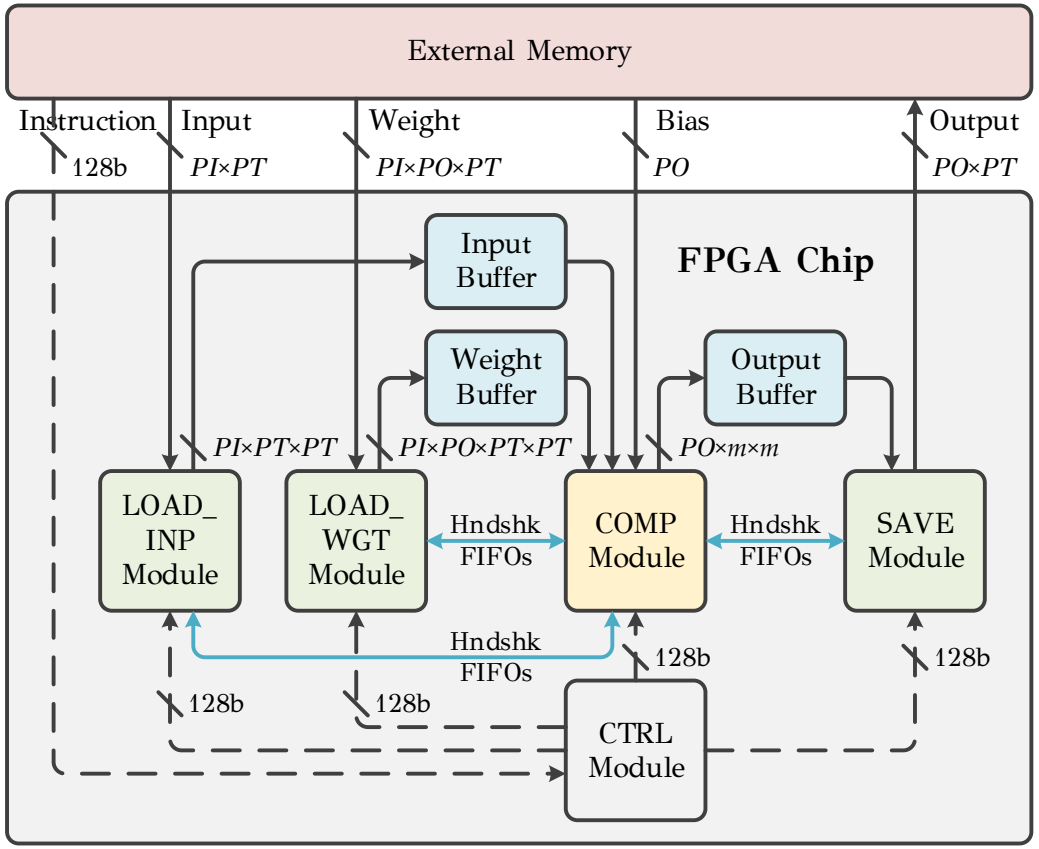
Accelerator Design (1)



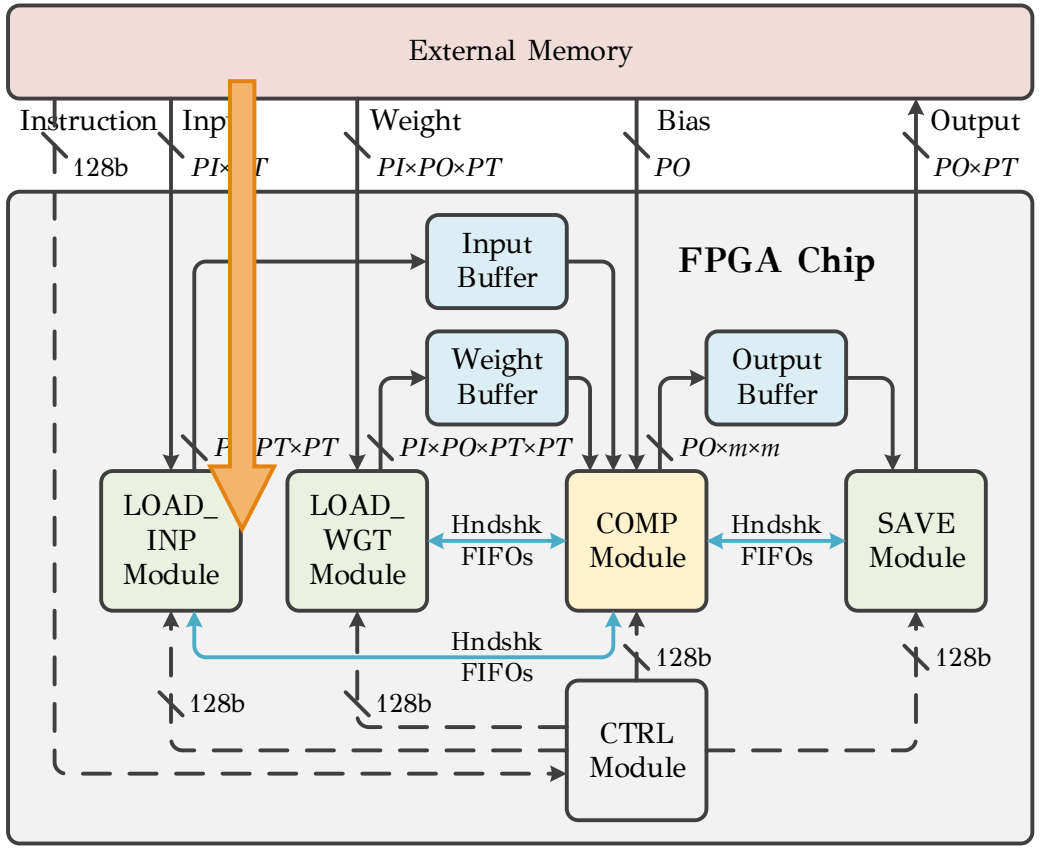
Efficient and Flexible

- Instruction-based accelerator with customized instruction set
- CTRL Module:
  - Load, decode, and distribute instructions to functional modules

Accelerator Design (2)

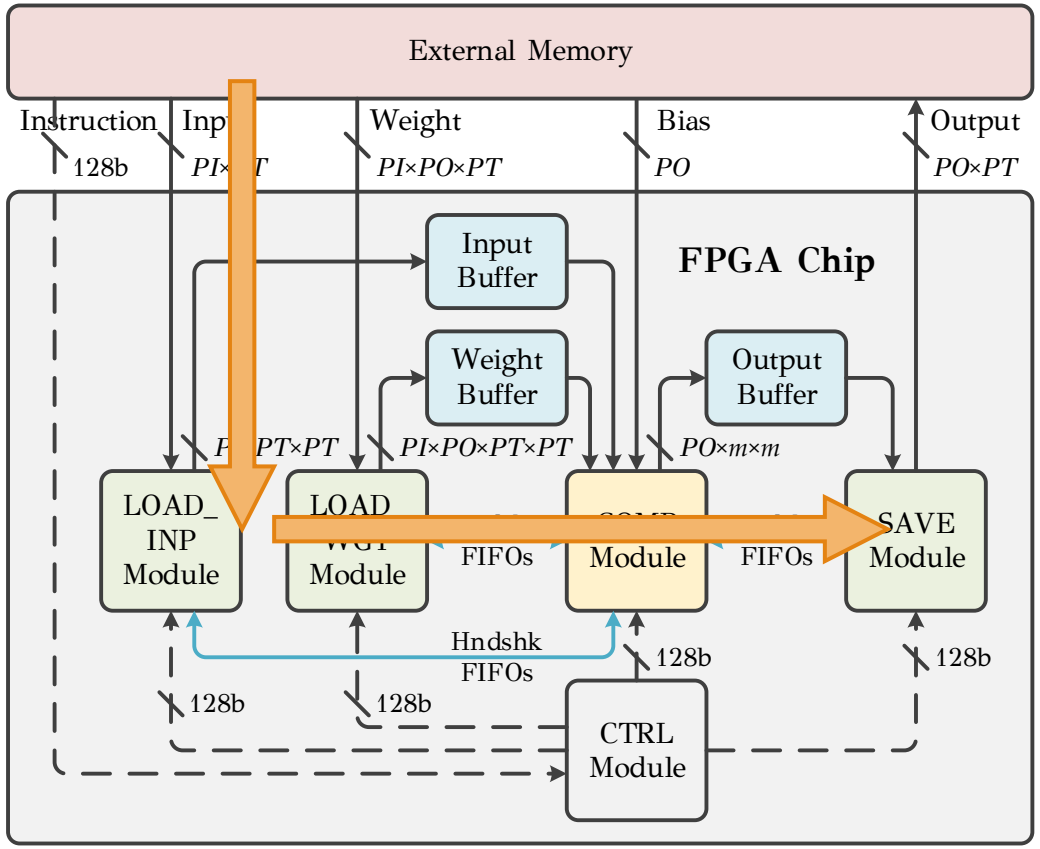


Accelerator Design (2)



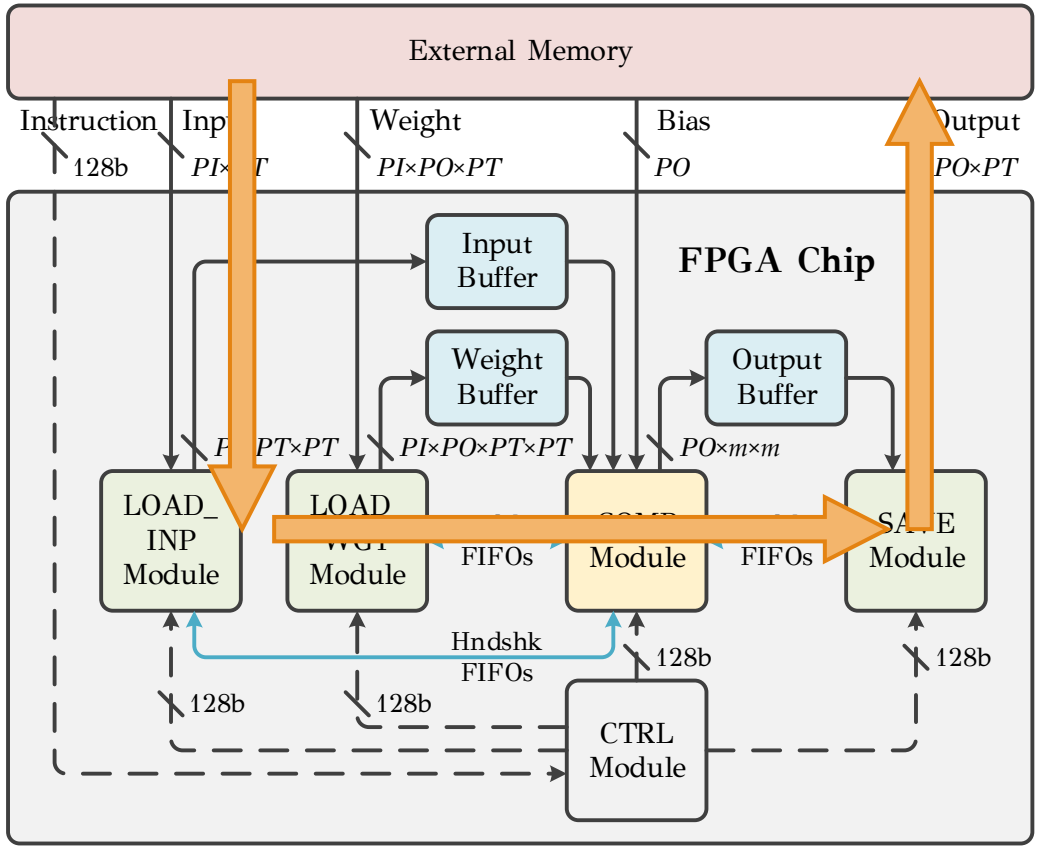
- LOAD_INP & _WGT Module:
  - Load input feature maps and weights from external memory
- COMP Module:
  - Carry out the computation
- SAVE Module:
  - Write back output feature maps

Accelerator Design (2)



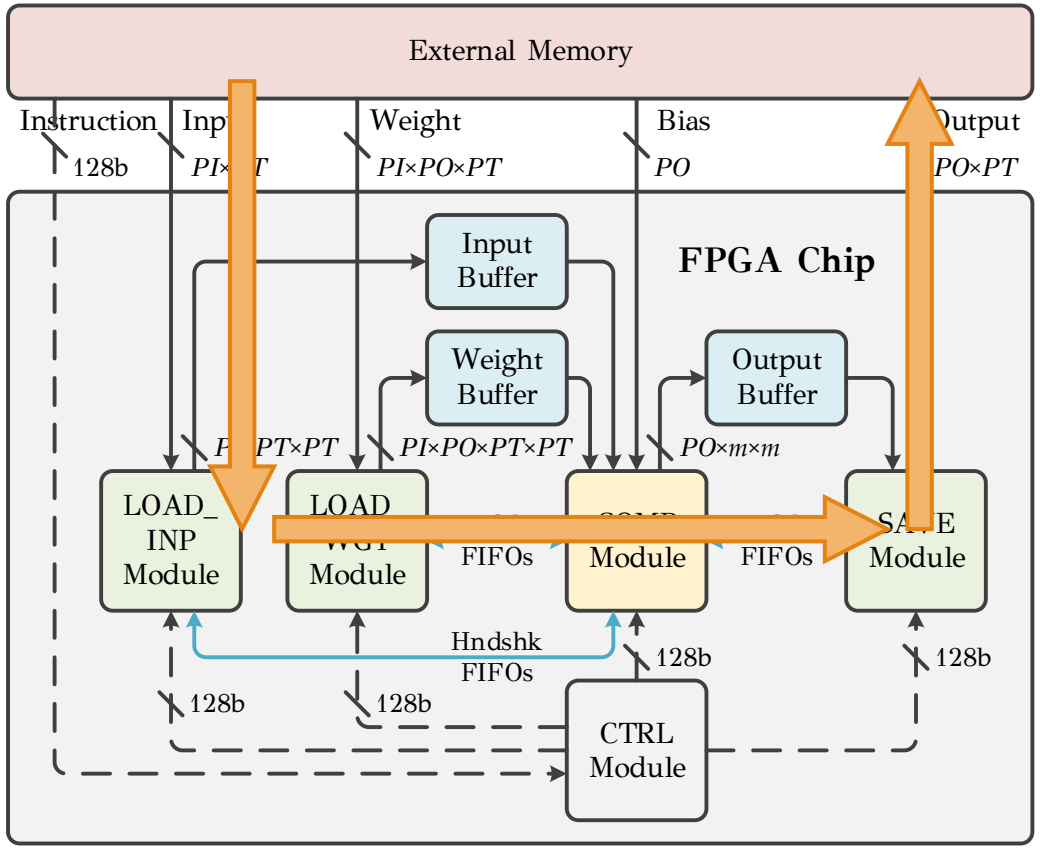
- LOAD_INP & _WGT Module:
  - Load input feature maps and weights from external memory
- COMP Module:
  - Carry out the computation
- SAVE Module:
  - Write back output feature maps

Accelerator Design (2)



- LOAD_INP & _WGT Module:
  - Load input feature maps and weights from external memory
- COMP Module:
  - Carry out the computation
- SAVE Module:
  - Write back output feature maps

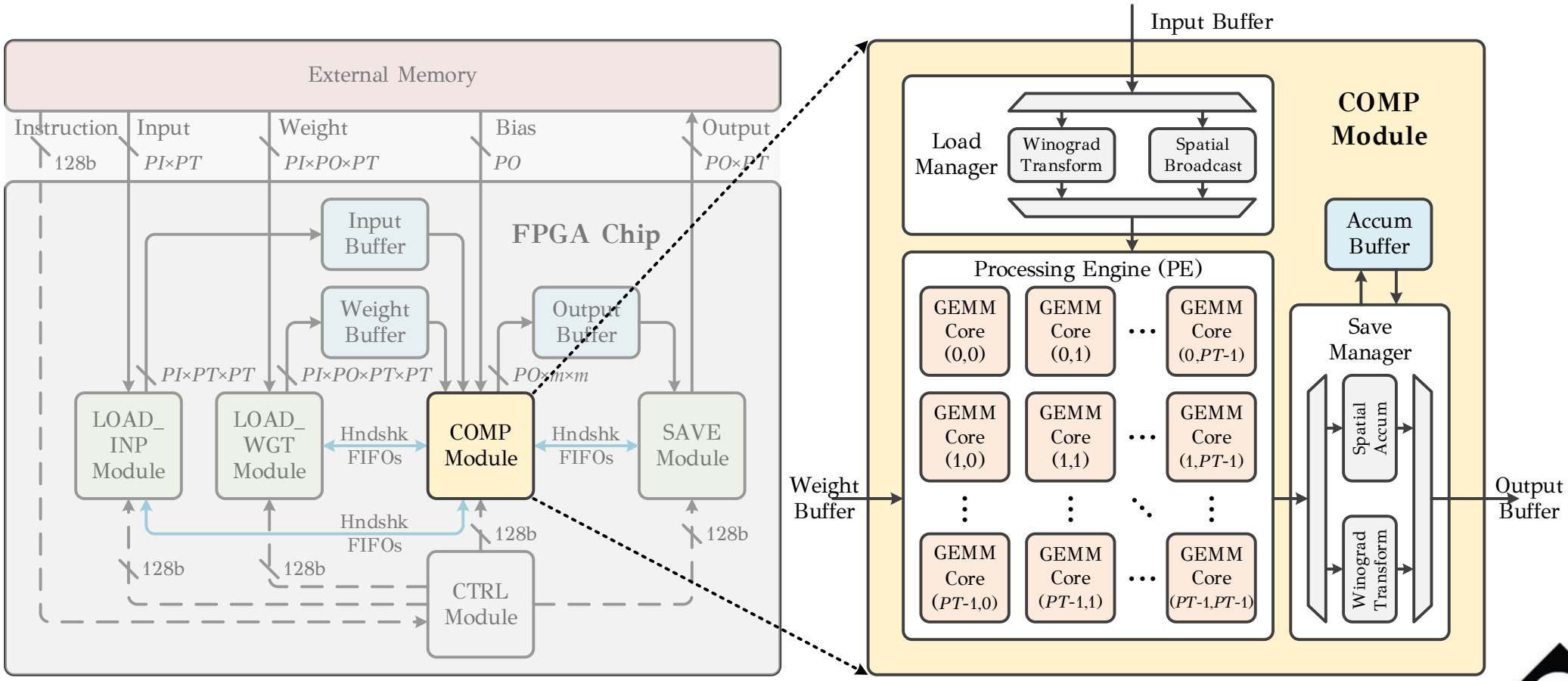
Accelerator Design (2)



- LOAD_INP & _WGT Module:
  - Load input feature maps and weights from external memory
- COMP Module:
  - Carry out the computation
- SAVE Module:
  - Write back output feature maps

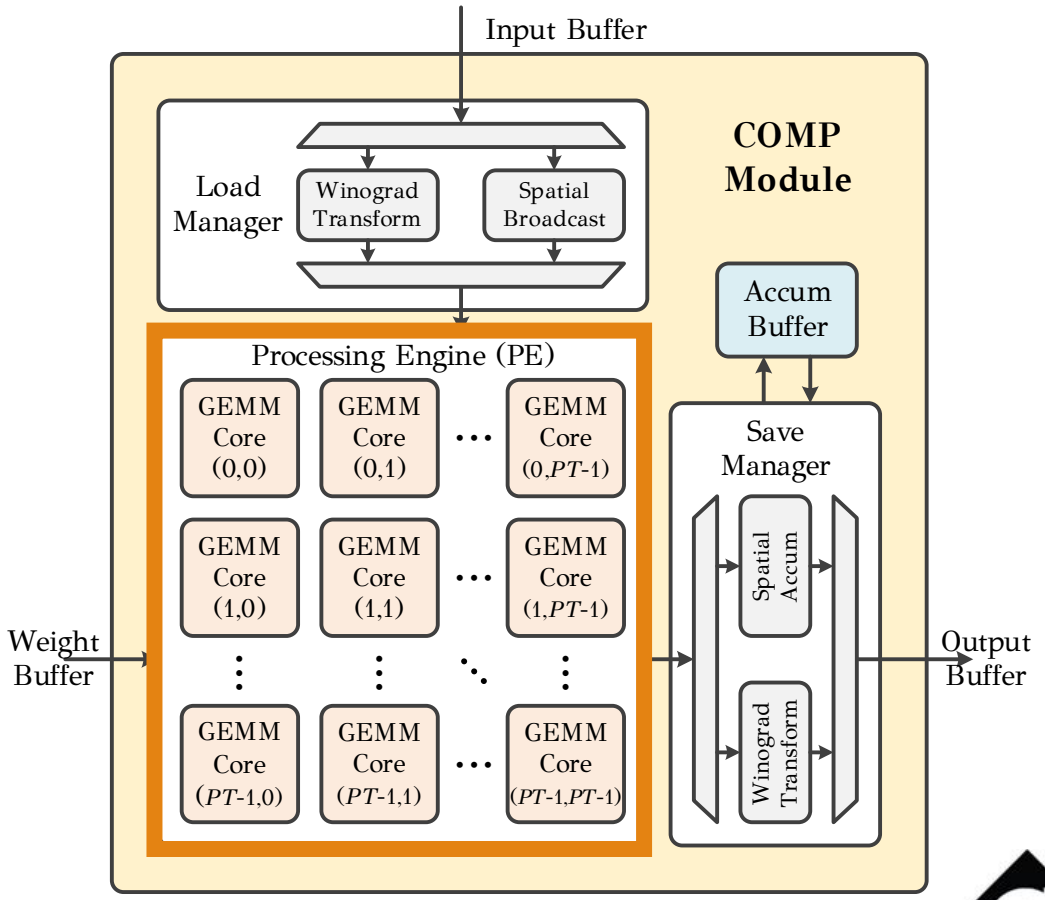
Module-level Pipeline

Accelerator Design (2)



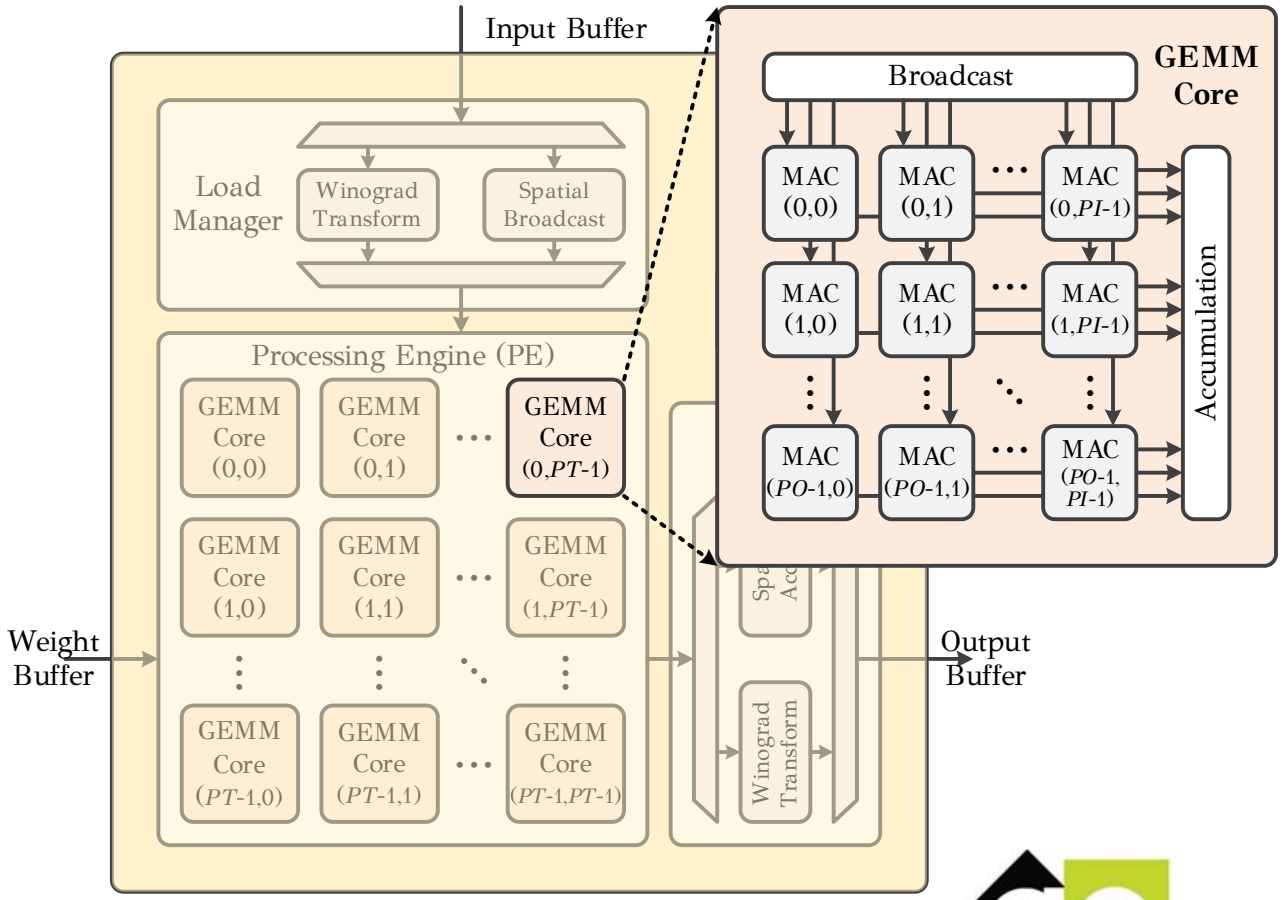
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



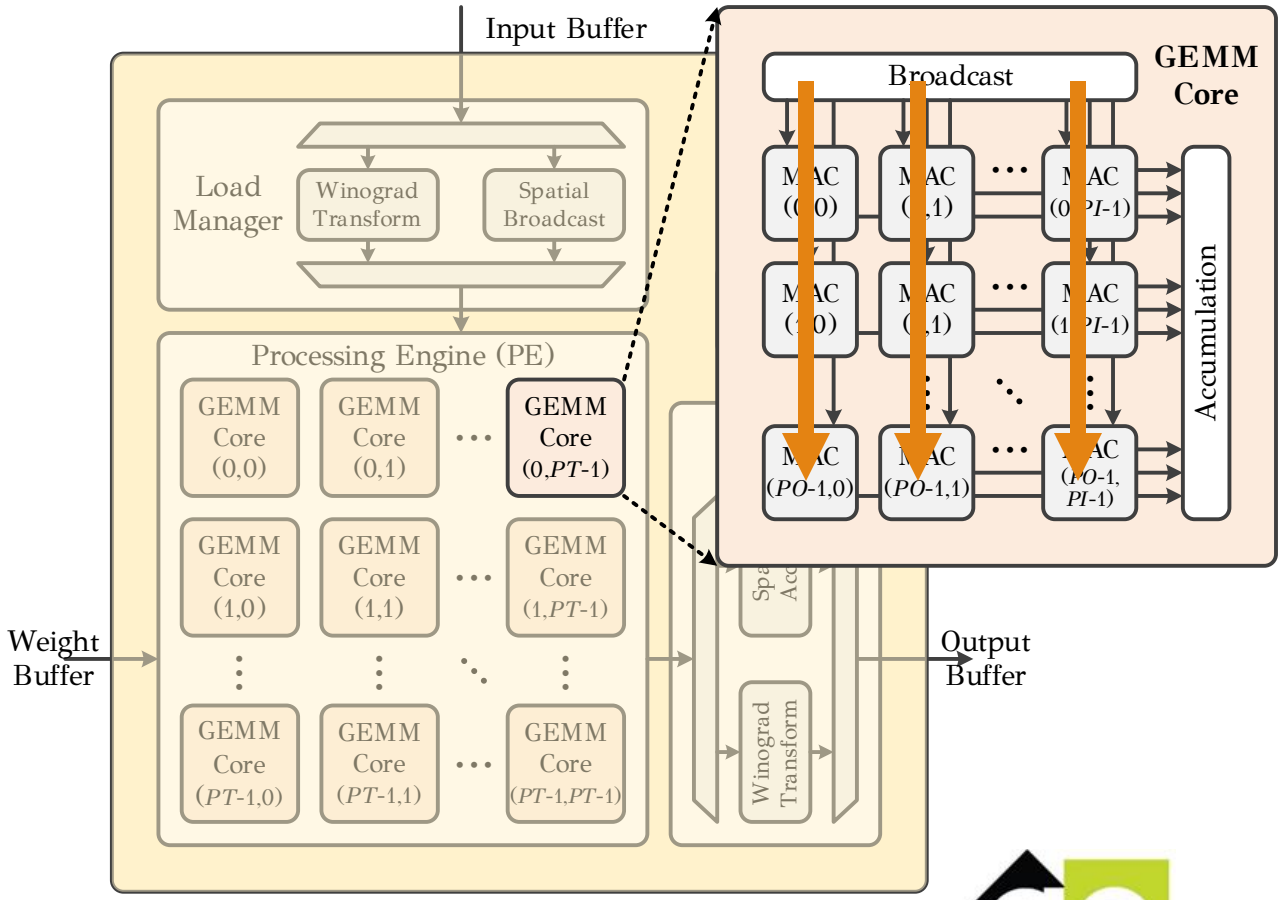
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



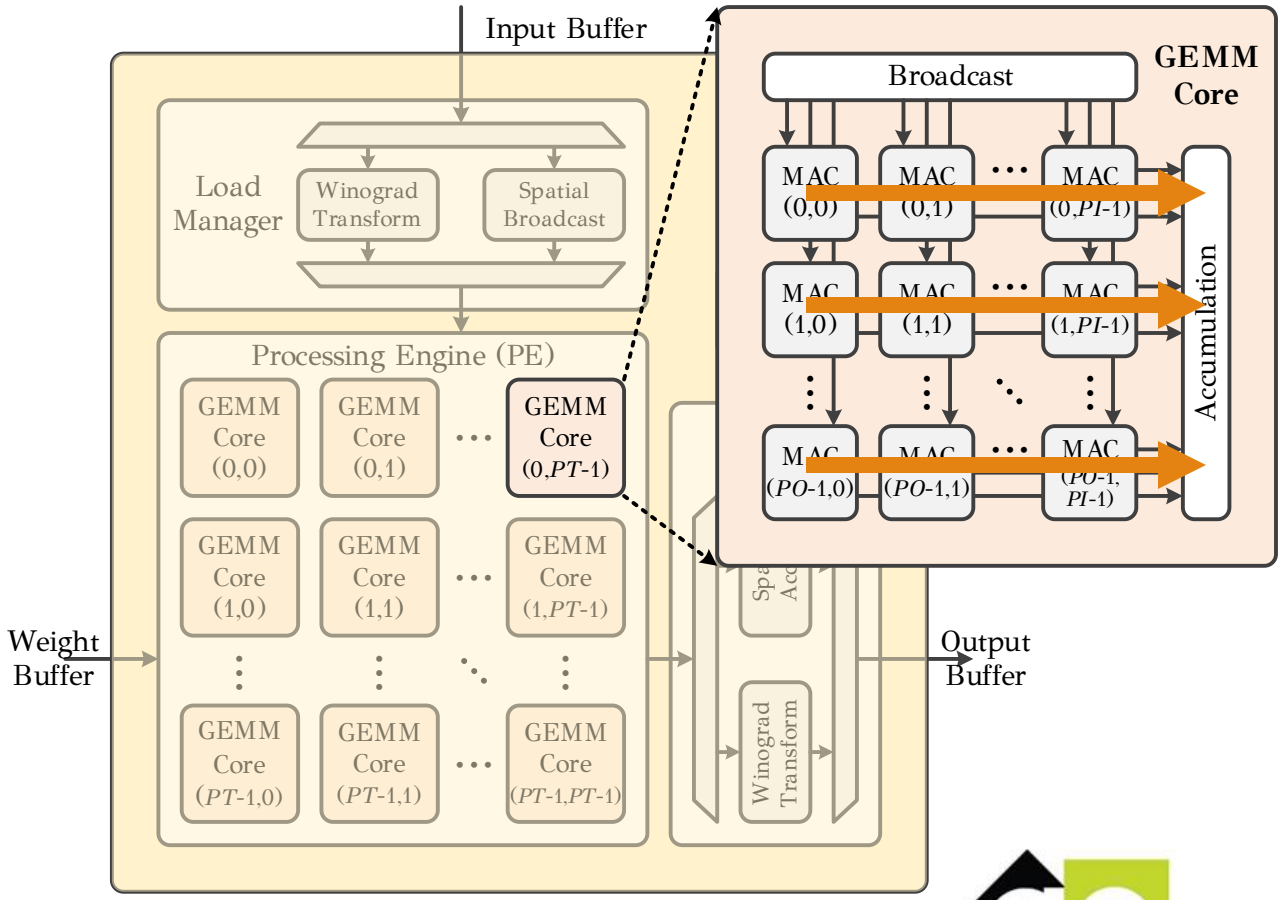
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



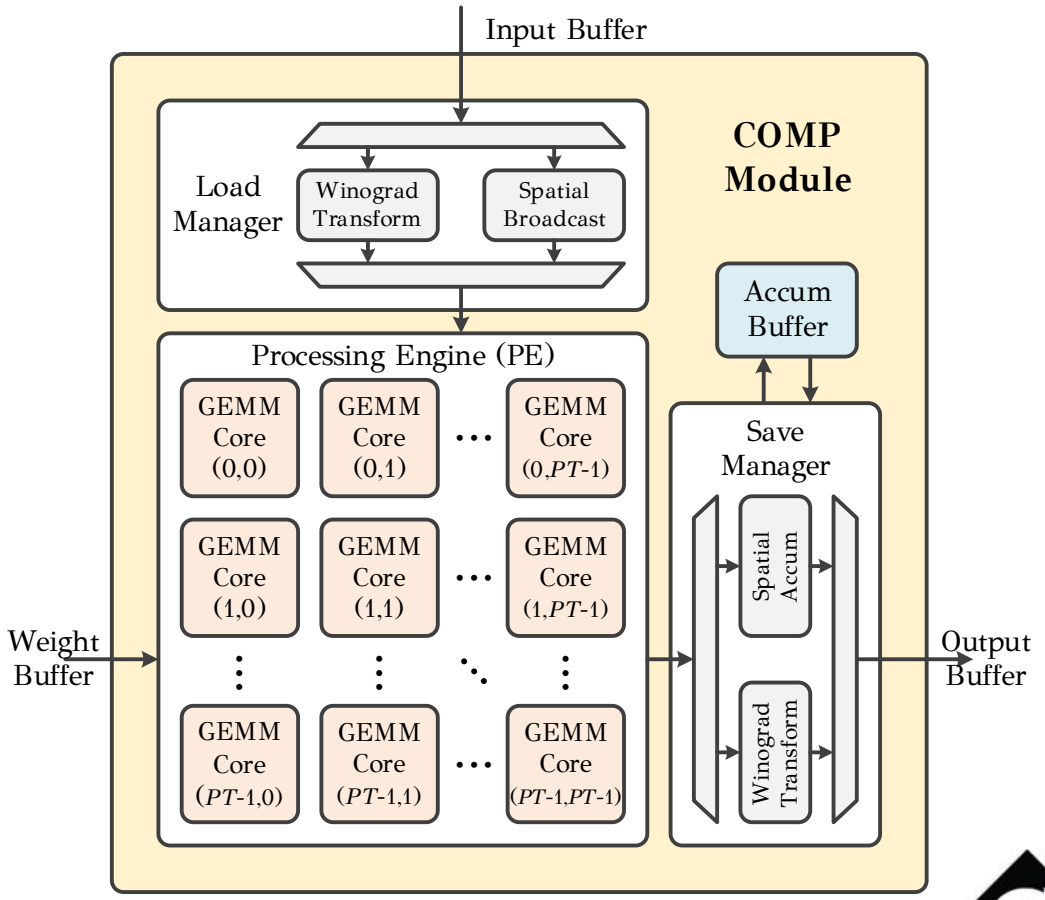
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



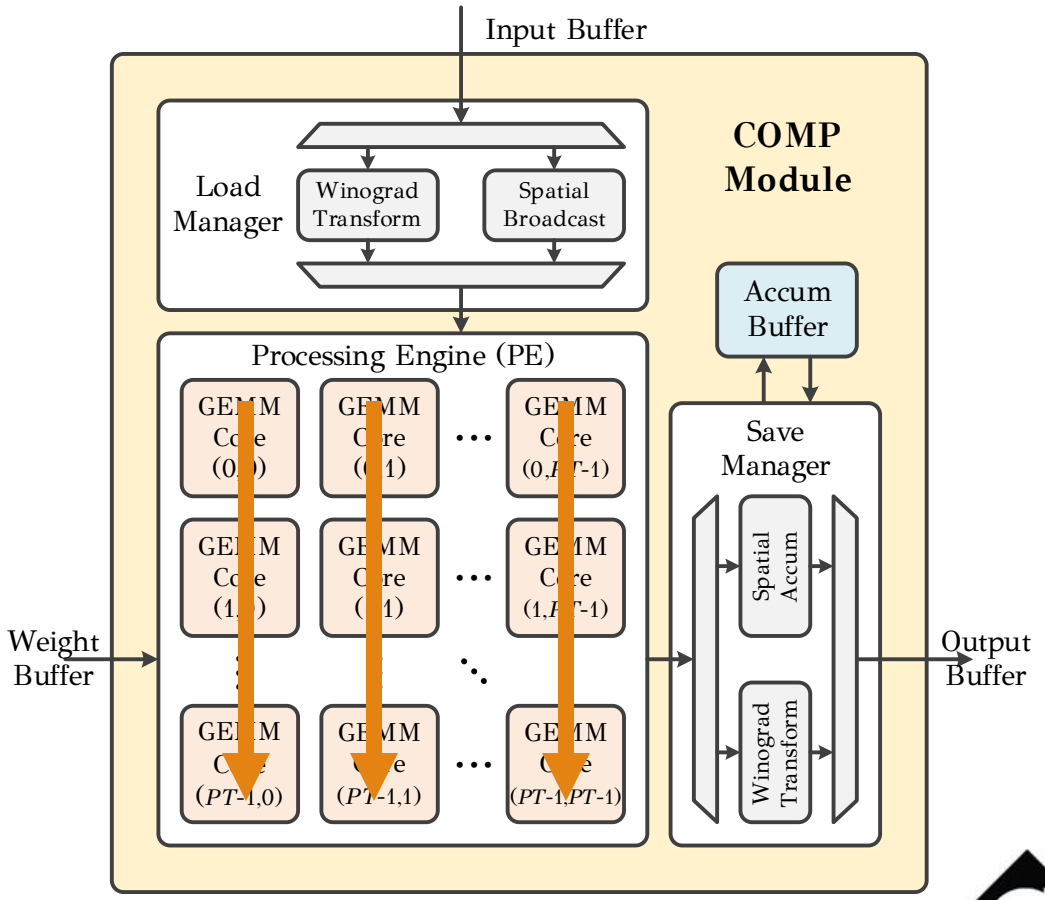
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



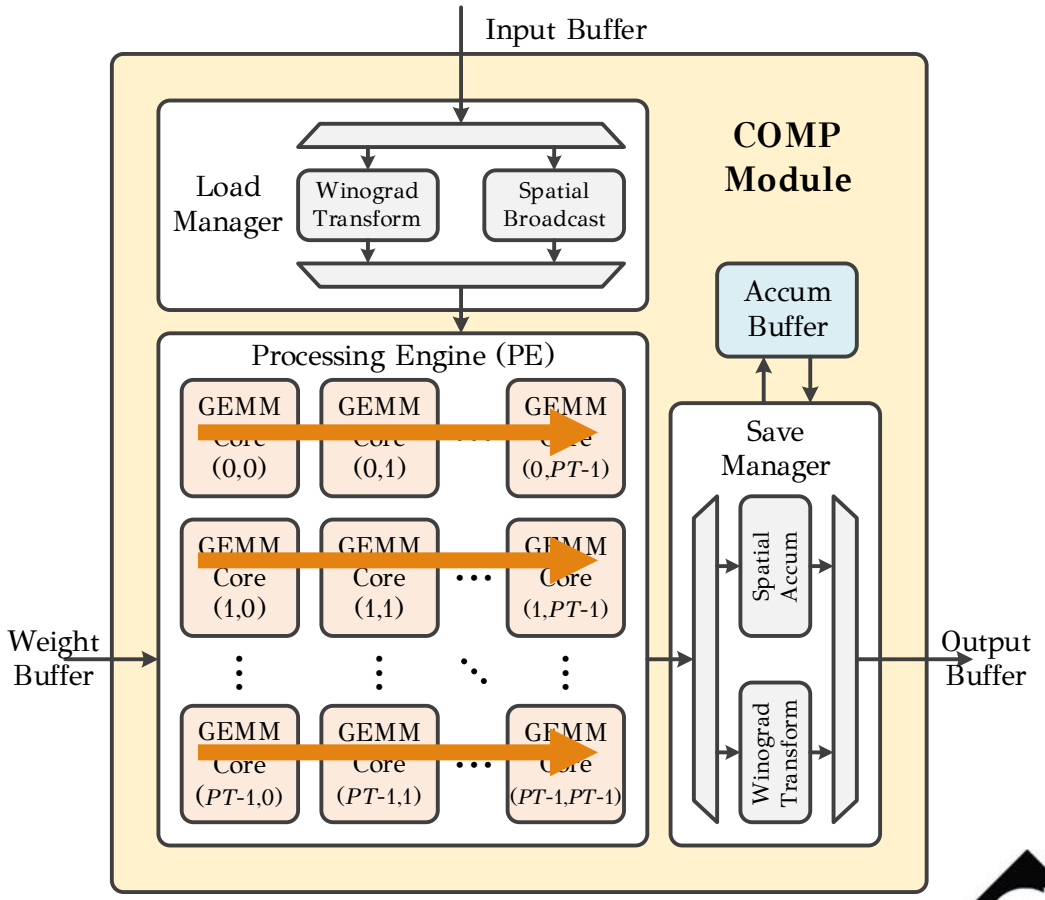
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



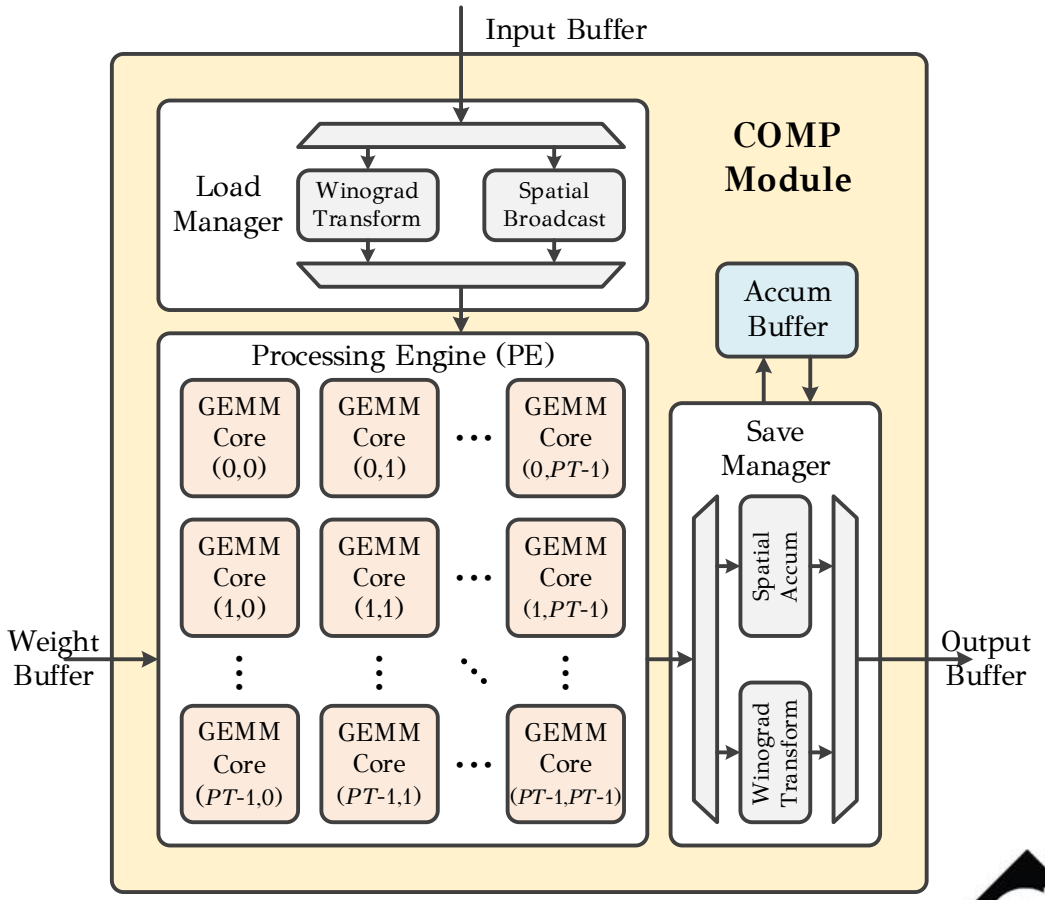
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



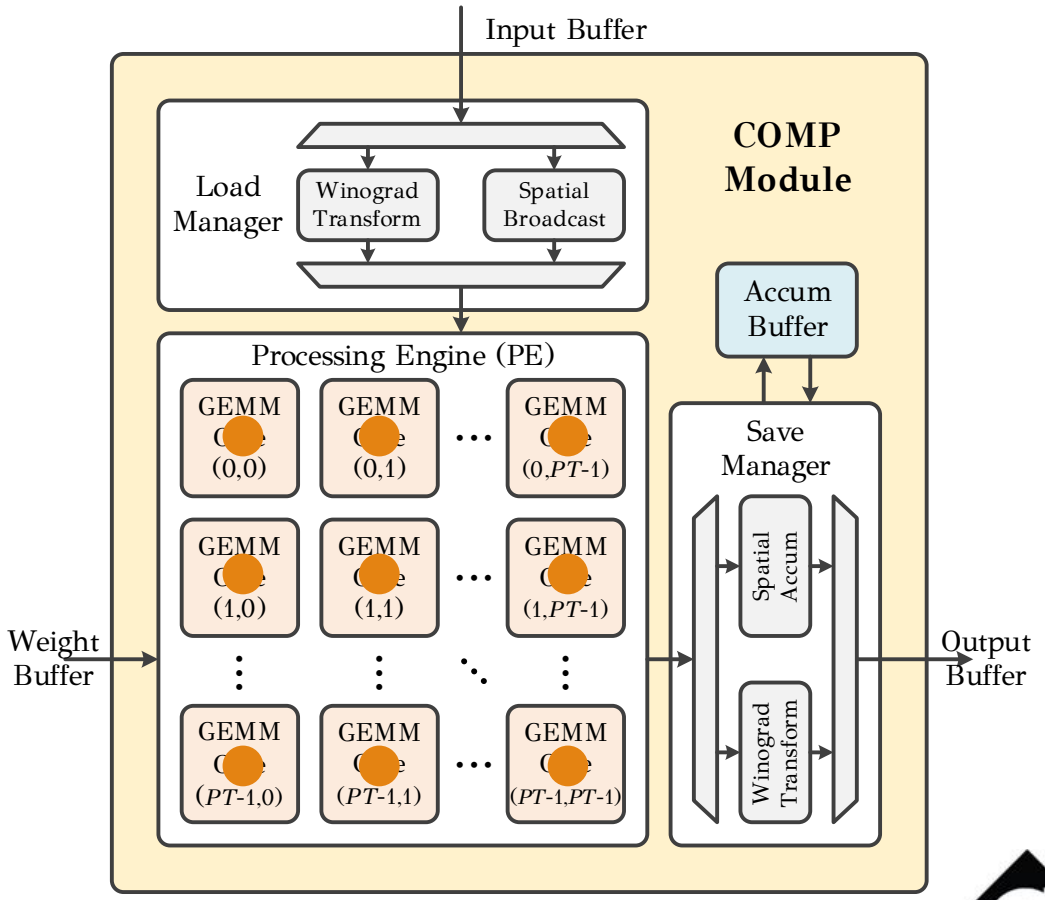
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



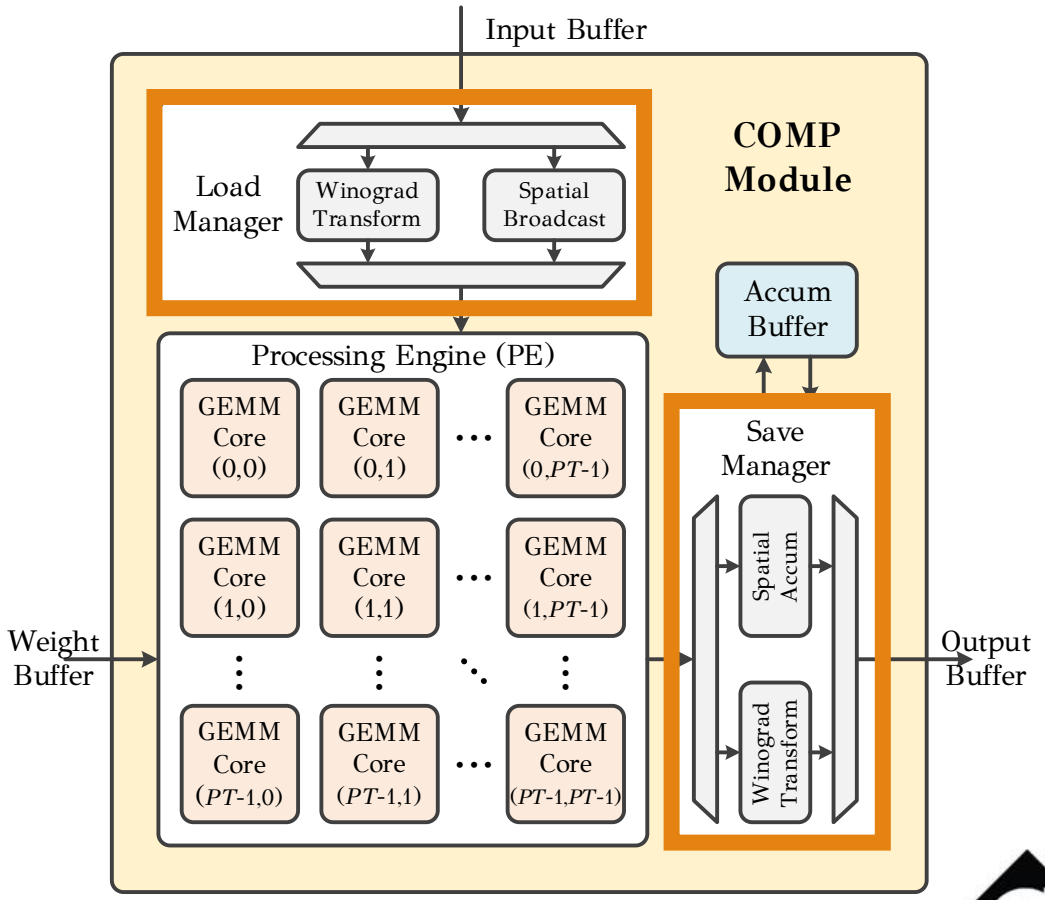
Accelerator Design (3)

- Processing Engine (PE):
  - Reused by Winograd and Spatial CONV
  - $PT \times PT$ GEMM Cores
- GEMM Cores:
  - MAC broadcast-array paralleled along input (PI) and output channels (PO)
- GEMM Cores Organization:
  - Spatial: A large broadcast array
  - Winograd: Compute independently



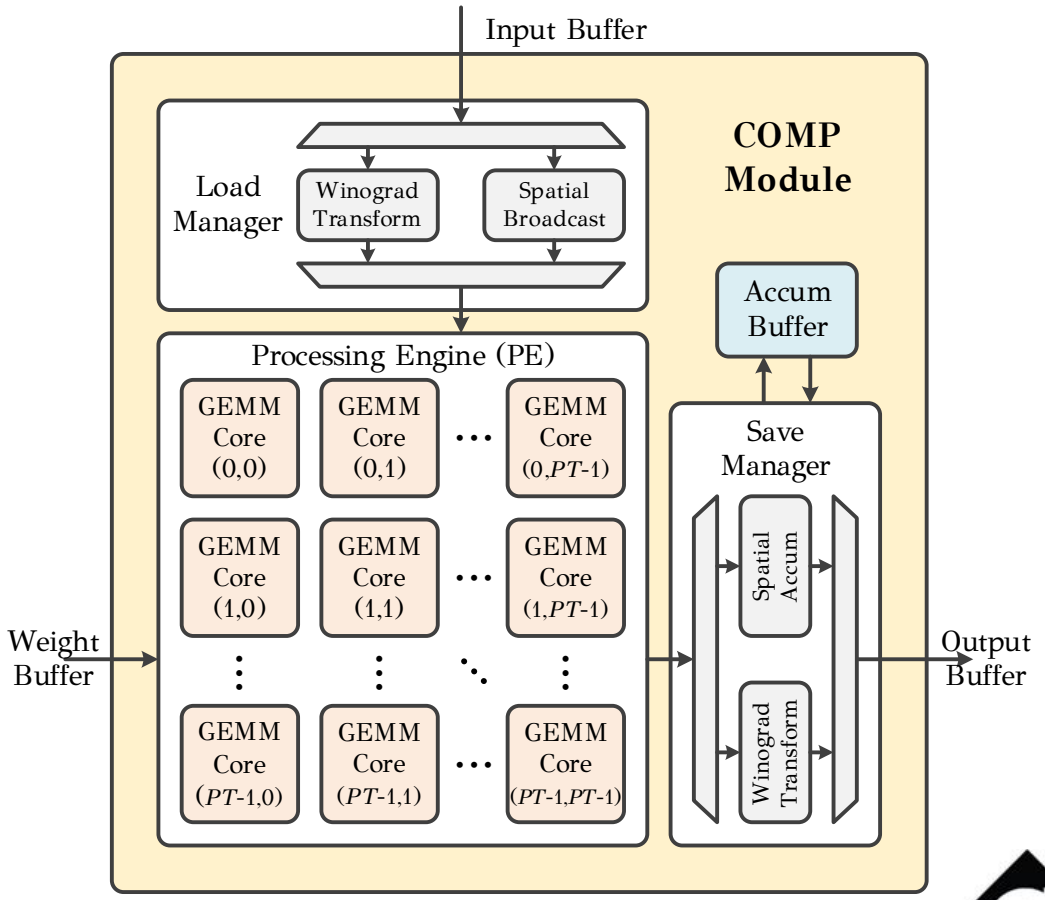
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



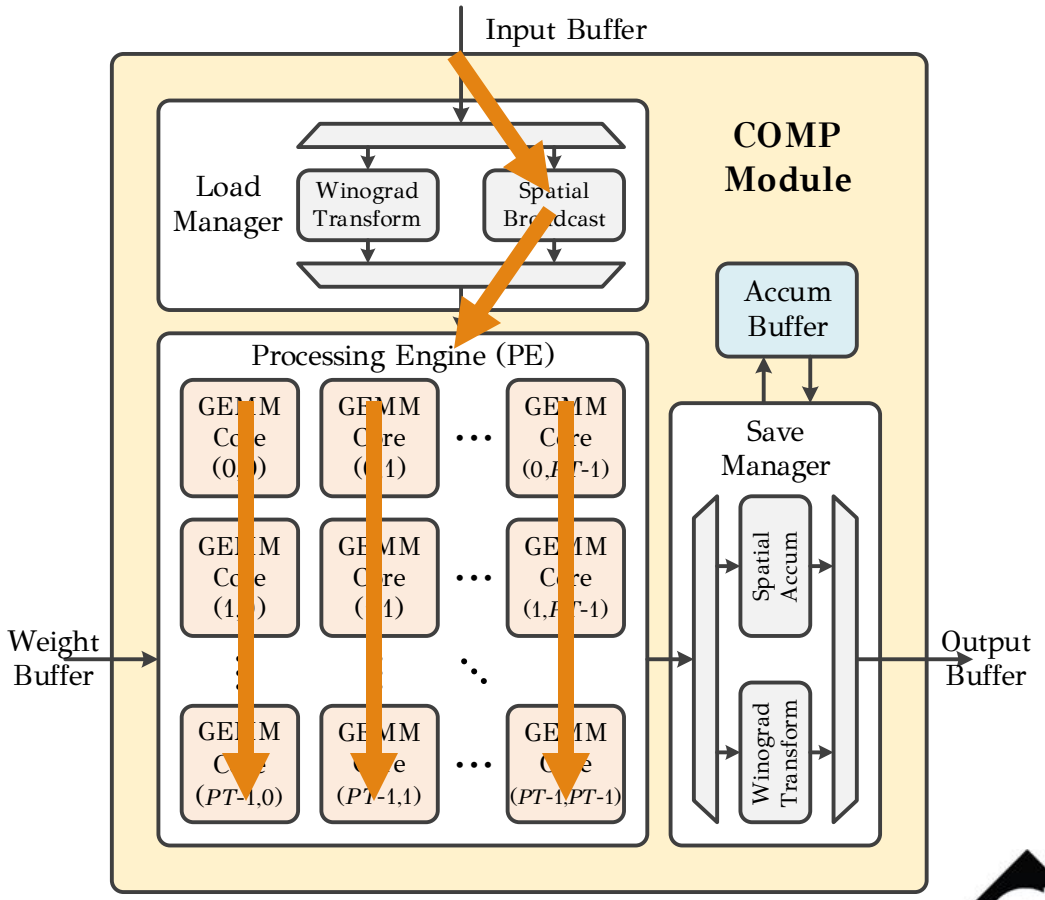
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



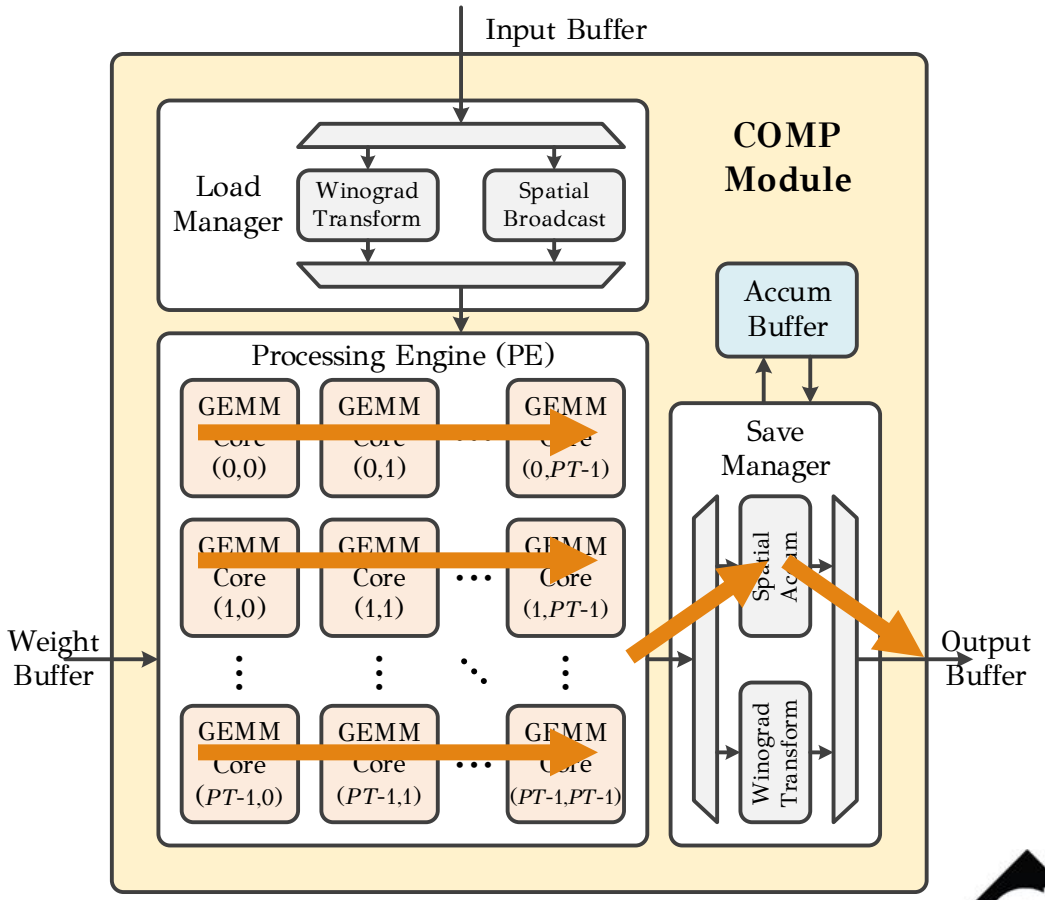
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



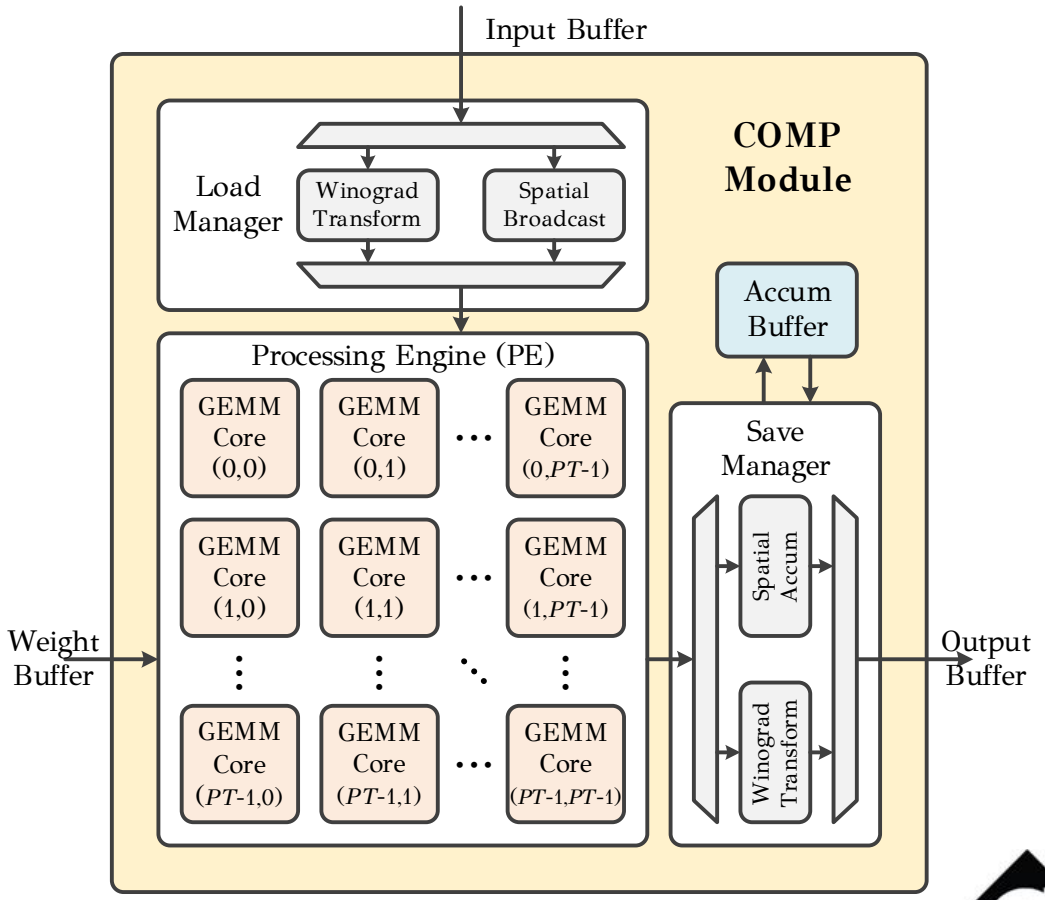
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



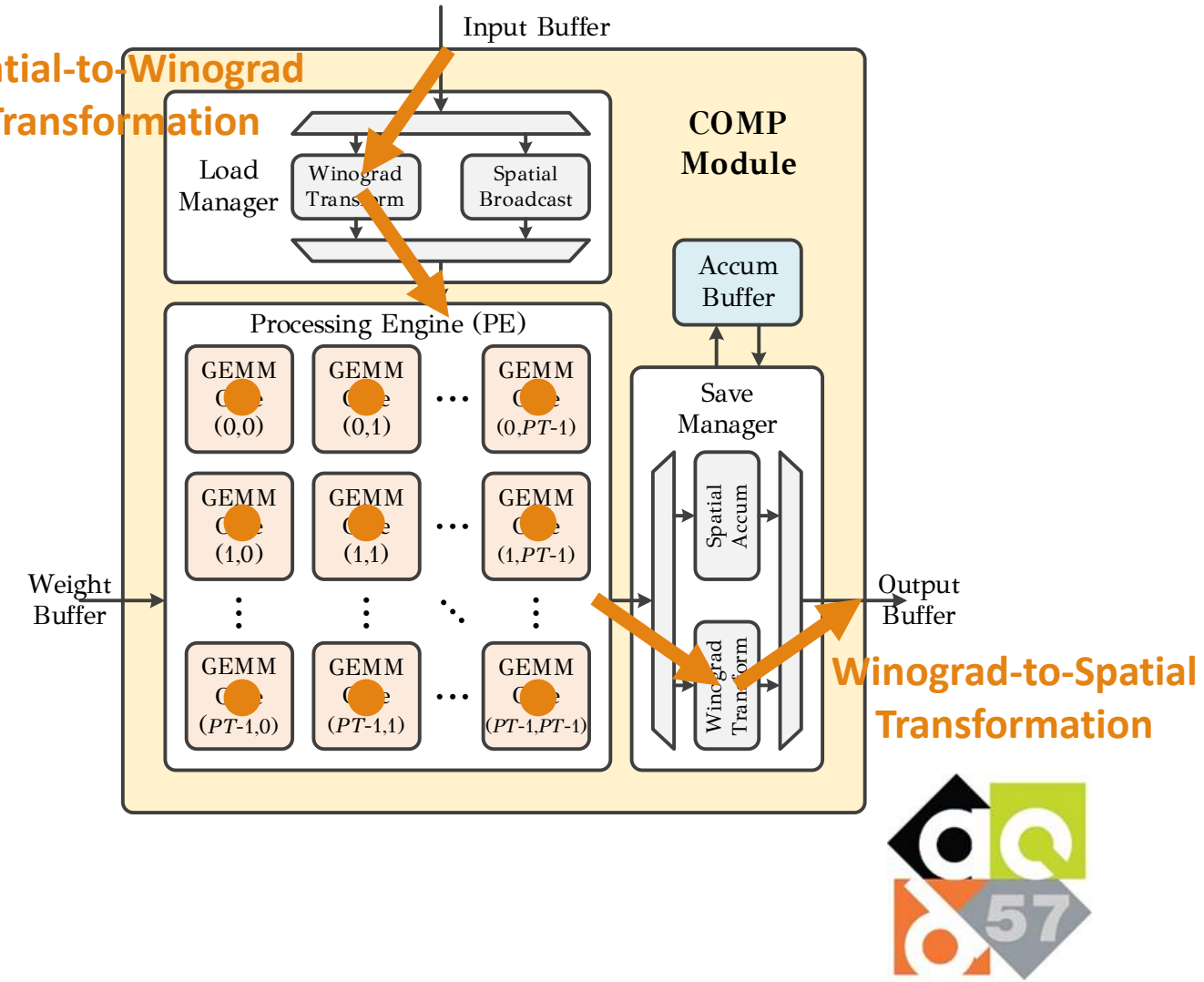
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



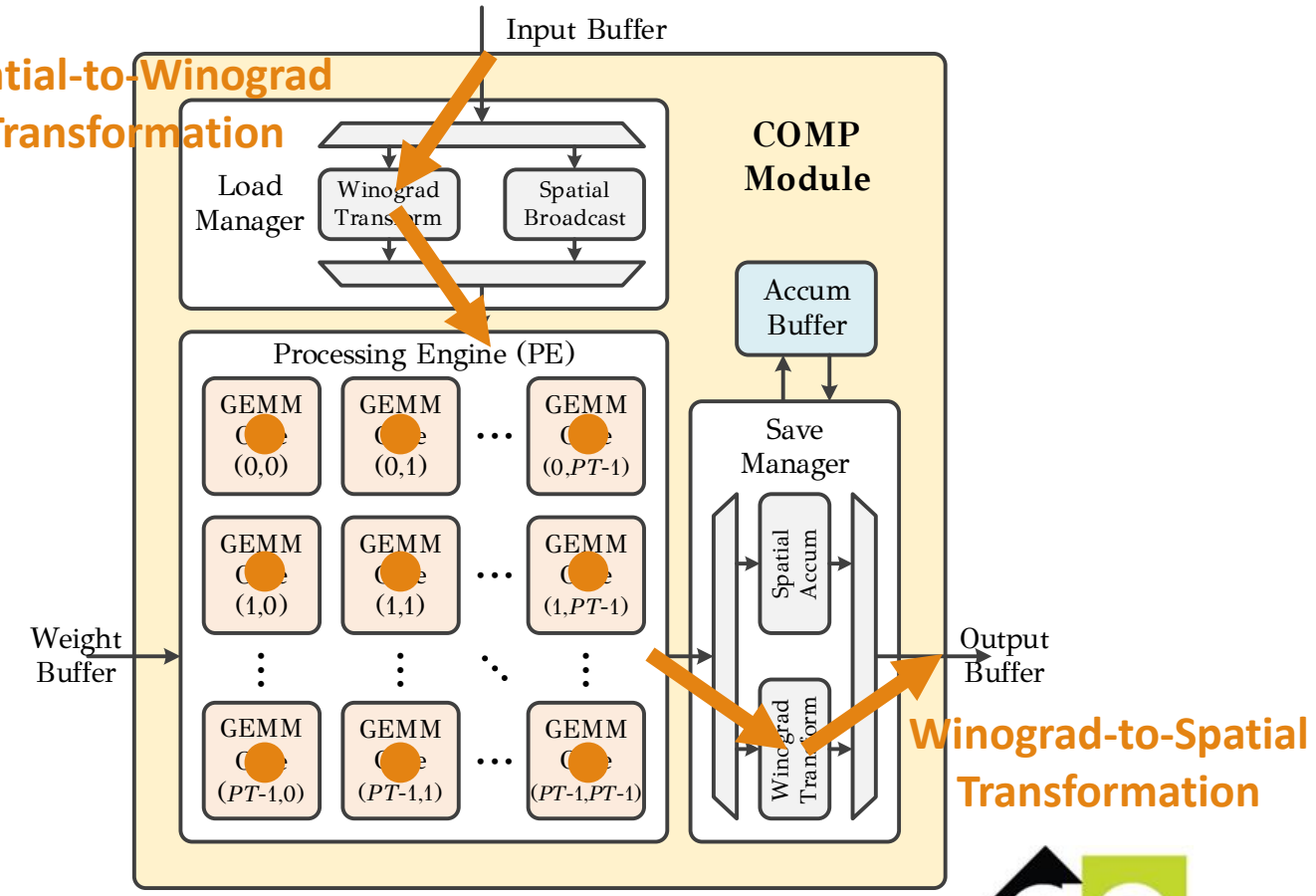
Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



Accelerator Design (4)

- Load & Save Manager:
  - Can switch between Winograd and Spatial CONV mode
- Spatial Mode
- Winograd Mode:
  - LUT-based Winograd transformation between two domains



Resource Overhead:

LUTs for Winograd transformation

Design Space Exploration

HW Parameters	PI, PO, PT, NI
SW Parameters	$\{mode_1, mode_2, \ldots mode_L\}$, $\{dataflow_1, dataflow_2, \ldots dataflow_L\}$
Constraints	$PI \geq PO \geq 1, PT \in \{4, 6\}$, $N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM$, $mode_l \in \{"spat", "wino"\}, dataflow_l \in \{"is", "ws"\}$
Objective	$\sum_{l=1}^{L} T_l$

Presume

- Totally L CONV / FC layers
- is and ws means input and weight stationary
- T_l: latency of the l-th layer

- ***Step0:*** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- ***Step1:*** Search for design candidates with different HW parameters
- ***Step2:*** Search for optimal SW parameters combination for each candidate
- ***Step3:*** Select the design candidate with the lowest latency

Design Space Exploration

HW Parameters	PI, PO, PT, NI
SW Parameters	$\{mode_1, mode_2, \ldots mode_L\}$, $\{dataflow_1, dataflow_2, \ldots dataflow_L\}$
Constraints	$PI \geq PO \geq 1, PT \in \{4, 6\}$, $N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM$, $mode_l \in \{"spat", "wino"\}, dataflow_l \in \{"is", "ws"\}$
Objective	$\sum_{l=1}^{L} T_l$

Presume

- Totally L CONV / FC layers
- is and ws means input and weight stationary
- T_l: latency of the l-th layer

- ***Step0:*** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- ***Step1:*** Search for design candidates with different HW parameters
- ***Step2:*** Search for optimal SW parameters combination for each candidate
- ***Step3:*** Select the design candidate with the lowest latency

Design Space Exploration

HW Parameters	PI, PO, PT, NI
SW Parameters	$\{mode_1, mode_2, \ldots mode_L\}$, $\{dataflow_1, dataflow_2, \ldots dataflow_L\}$
Constraints	$PI \geq PO \geq 1, PT \in \{4, 6\}$, $N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM$, $mode_l \in \{"spat", "wino"\}, dataflow_l \in \{"is", "ws"\}$
Objective	$\sum_{l=1}^{L} T_l$

Presume

- Totally L CONV / FC layers
- is and ws means input and weight stationary
- T_l: latency of the l-th layer

- ***Step0:*** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- ***Step1:*** Search for design candidates with different HW parameters
- ***Step2:*** Search for optimal SW parameters combination for each candidate
- ***Step3:*** Select the design candidate with the lowest latency

Design Space Exploration

HW Parameters	PI, PO, PT, NI
SW Parameters	$\{mode_1, mode_2, \ldots mode_L\}$, $\{dataflow_1, dataflow_2, \ldots dataflow_L\}$
Constraints	$PI \geq PO \geq 1, PT \in \{4, 6\}$, $N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM$, $mode_l \in \{"spat", "wino"\}, dataflow_l \in \{"is", "ws"\}$
Objective	$\sum_{l=1}^{L} T_l$

Presume

- Totally L CONV / FC layers
- is and ws means input and weight stationary
- T_l: latency of the l-th layer

- ***Step0:*** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- ***Step1:*** Search for design candidates with different HW parameters
- ***Step2:*** Search for optimal SW parameters combination for each candidate
- ***Step3:*** Select the design candidate with the lowest latency

Design Space Exploration

HW Parameters	PI, PO, PT, NI
SW Parameters	$\{mode_1, mode_2, \ldots mode_L\}$, $\{dataflow_1, dataflow_2, \ldots dataflow_L\}$
Constraints	$PI \geq PO \geq 1, PT \in \{4, 6\}$, $N_{LUT} < LUT, N_{DSP} < DSP, N_{BRAM} < BRAM$, $mode_l \in \{"spat", "wino"\}, dataflow_l \in \{"is", "ws"\}$
Objective	$\sum_{l=1}^{L} T_l$

Presume

- Totally L CONV / FC layers
- is and ws means input and weight stationary
- T_l: latency of the l-th layer

- ***Step0:*** Latency and FPGA resource (LUT, DSP, and BRAM) modeling
- ***Step1:*** Search for design candidates with different HW parameters
- ***Step2:*** Search for optimal SW parameters combination for each candidate
- ***Step3:*** Select the design candidate with the lowest latency

Experimental Results (1)

	[26]	[4]	[6]	Ours	
Device	Xilinx VU9P	Arria10 GX1150	Xilinx VU9P	Xilinx VU9P	PYNQ Z1
Model	VGG16	VGG16	VGG16	VGG16	VGG16
Precision	16-bit	16-bit	16-bit	12-bit*	12-bit*
Freq.(MHz)	210	385	214	167	100
DSPs	4096	2756	5349	5163	220
CNN Perf.(GOPS)	1510	1790	1828.6	**3375.7**	83.3
Power(W)	NA	37.5	49.3	45.9	2.6
DSP Effi. (GOPS/DSP)	0.37	0.65	0.34	**0.65**	0.38
Energy Effi. (GOPS/W)	NA	47.78	37.1	**73.5**	32.0

*DNN parameters are quantized to 8-bit; input feature maps are set to 12-bit in PE due to the Winograd matrix transformation

- Xilinx VU9P Configuration:
  - $PI = 4, PO = 4, PT = 6, NI = 6$
- PYNQ-Z1 Configuration:
  - $PI = 4, PO = 4, PT = 4, NI = 1$

	LUTs	DSPs	18Kb BRAMs
VU9P	706353 (59.8%)	5163 (75.5%)	3169 (73.4%)
PYNQ-Z1	37034 (69.61%)	220 (100%)	277 (98.93%)

Experimental Results (1)

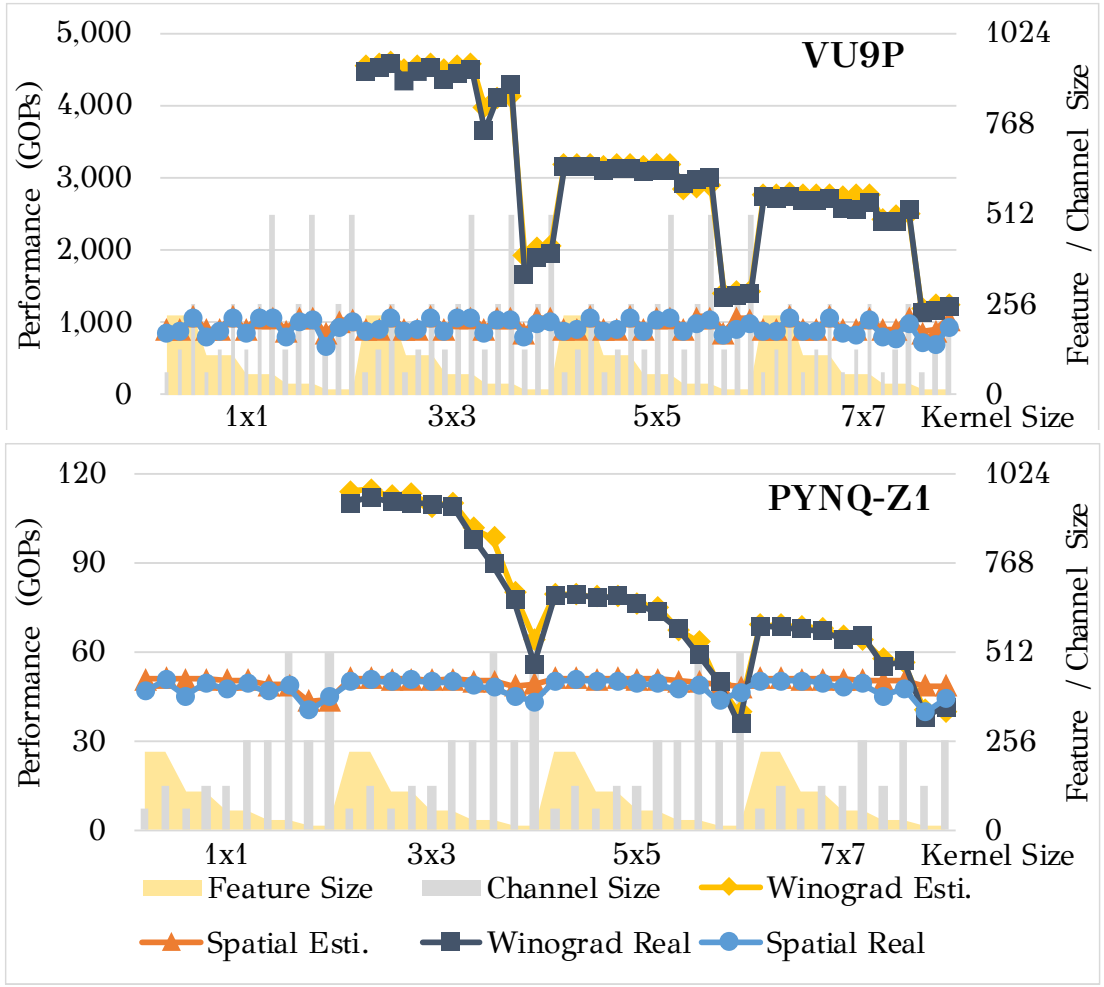
	[26]	[4]	[6]	Ours	
Device	Xilinx VU9P	Arria10 GX1150	Xilinx VU9P	Xilinx VU9P	PYNQ Z1
Model	VGG16	VGG16	VGG16	VGG16	VGG16
Precision	16-bit	16-bit	16-bit	12-bit*	12-bit*
Freq.(MHz)	210	385	214	167	100
DSPs	4096	2756	5349	5163	220
CNN Perf.(GOPS)	1510	1790	1828.6	**3375.7**	83.3
Power(W)	NA	37.5	49.3	45.9	2.6
DSP Effi. (GOPS/DSP)	0.37	0.65	0.34	**0.65**	0.38
Energy Effi. (GOPS/W)	NA	47.78	37.1	**73.5**	32.0

*DNN parameters are quantized to 8-bit; input feature maps are set to 12-bit in PE due to the Winograd matrix transformation

- Xilinx VU9P Configuration:
  - $PI = 4, PO = 4, PT = 6, NI = 6$
- PYNQ-Z1 Configuration:
  - $PI = 4, PO = 4, PT = 4, NI = 1$

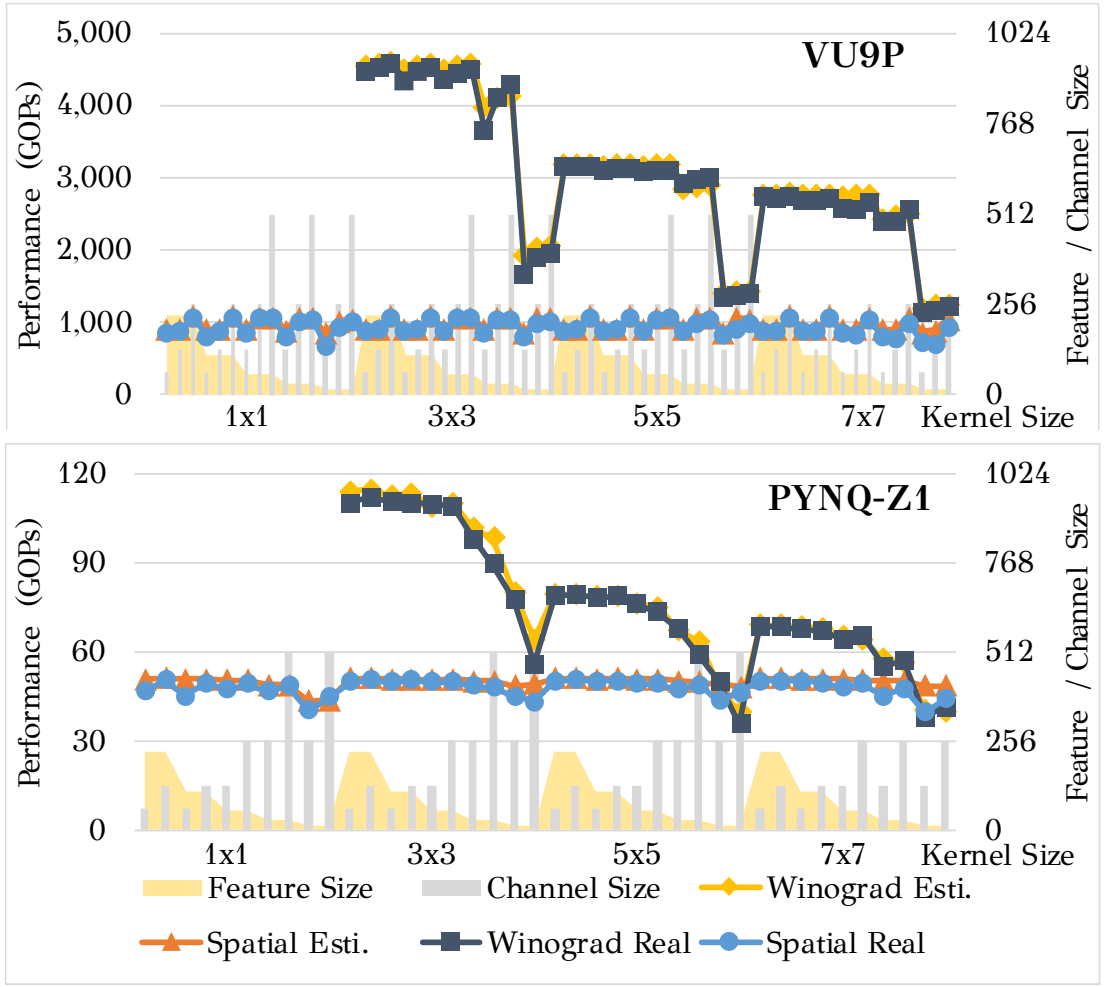
	LUTs	DSPs	18Kb BRAMs
VU9P	706353 (59.8%)	5163 (75.5%)	3169 (73.4%)
PYNQ-Z1	37034 (69.61%)	220 (100%)	277 (98.93%)

Experimental Results (2)



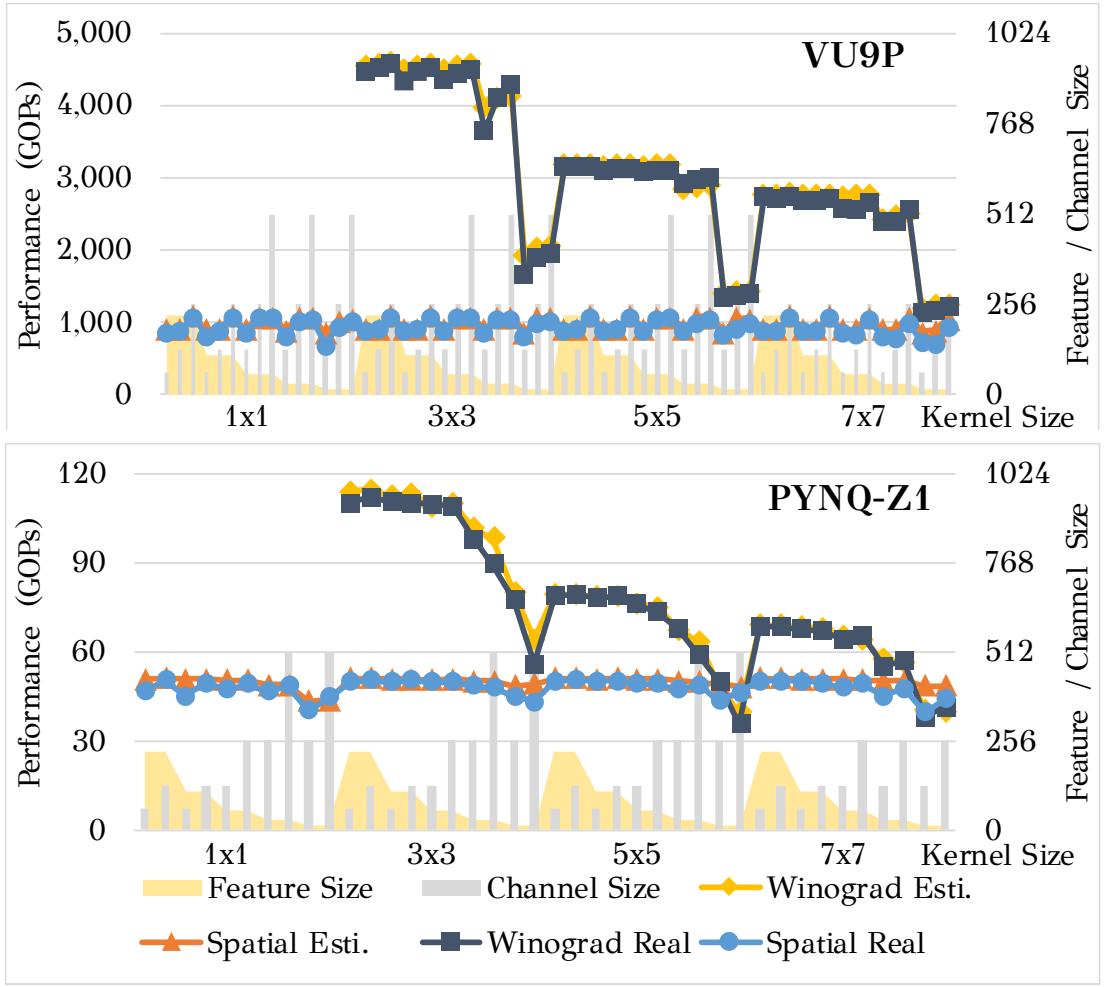
- CONV Layer Test Cases:
  - Kernel Size: 1x1, 3x3, 5x5, 7x7
  - Feature Size: 224, 112, 56, 28, 14
  - Channel Size: 512, 256, 128, 64
- Spatial CONV:
  - Support all kernel sizes
  - Stable and close to peak perf.
- Winograd CONV:
  - Higher peak perf. than Spatial CONV
  - Small feature size => Low weight reuse rate => Bounded by memory BW

Experimental Results (2)



- CONV Layer Test Cases:
  - Kernel Size: 1x1, 3x3, 5x5, 7x7
  - Feature Size: 224, 112, 56, 28, 14
  - Channel Size: 512, 256, 128, 64
- Spatial CONV:
  - Support all kernel sizes
  - Stable and close to peak perf.
- Winograd CONV:
  - Higher peak perf. than Spatial CONV
  - Small feature size => Low weight reuse rate => Bounded by memory BW

Experimental Results (2)



- CONV Layer Test Cases:
  - Kernel Size: 1x1, 3x3, 5x5, 7x7
  - Feature Size: 224, 112, 56, 28, 14
  - Channel Size: 512, 256, 128, 64
- Spatial CONV:
  - Support all kernel sizes
  - Stable and close to peak perf.
- Winograd CONV:
  - Higher peak perf. than Spatial CONV
  - Small feature size =>
    Low weight reuse rate =>
    Bounded by memory BW

Conclusion

- HybridDNN Framework: generate highly optimized accelerators for the latest generation of cloud and embedded FPGAs
- Instruction-based Architecture:
  - Hybrid CONV Processing Engine (Spatial and Winograd CONV)
  - Support multiple dataflow (input and weight stationary)
  - Scalable parallel factors (PI, PO, PT, and NI)
- Design Space Exploration:
  - Performance estimation model (4.27% and 4.03% error for VU9P and PYNQ-Z1)
  - Efficient algorithm for optimizing HW & SW parameters
- On-board Experimental Results:
  - 3375.3 (VU9P) and 83.3 (PYNQ-Z1) GOPS performance on VGG-16

Questions

Thank you!

July 22, 2020