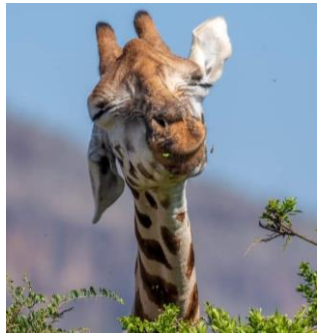
Lab 4
Processing Element

Advisor: Chia-Chi Tsai
TAs: 湯詠涵、洪翊碩、劉子齊

Video Link Click Me (๑˘₃˘๑) ♫

Gmail: course.aislab@gmail.com

Outline



- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

Outline

- **Chapter1 PE Introduction**
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

Reference paper

- Y. -H. Chen, T. Krishna, J. S. Emer and V. Sze, "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks," in IEEE Journal of Solid-State Circuits, vol. 52, no. 1, pp. 127-138, Jan. 2017, doi: 10.1109/JSSC.2016.2616357.

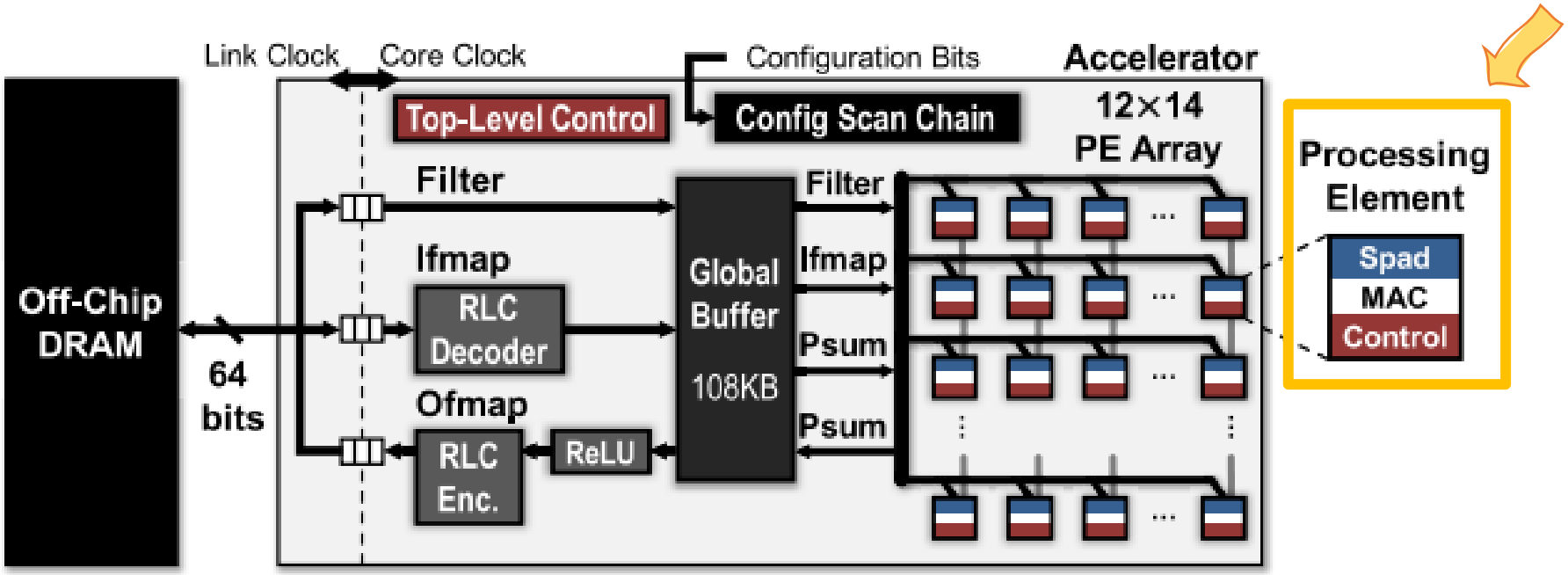


Fig. 2. Eyeriss system architecture.

What is PE ?

- Compose three parts :
  - **MAC(Multiply and Accumulate)** is the smallest computing unit.
  - **Spad(Scratch Pad)** is memory space inside PE.
  - **Control** commands spad, MAC and MUX based on different ifmap size, stride, etc.

FIFO
Store upcoming data beforehand

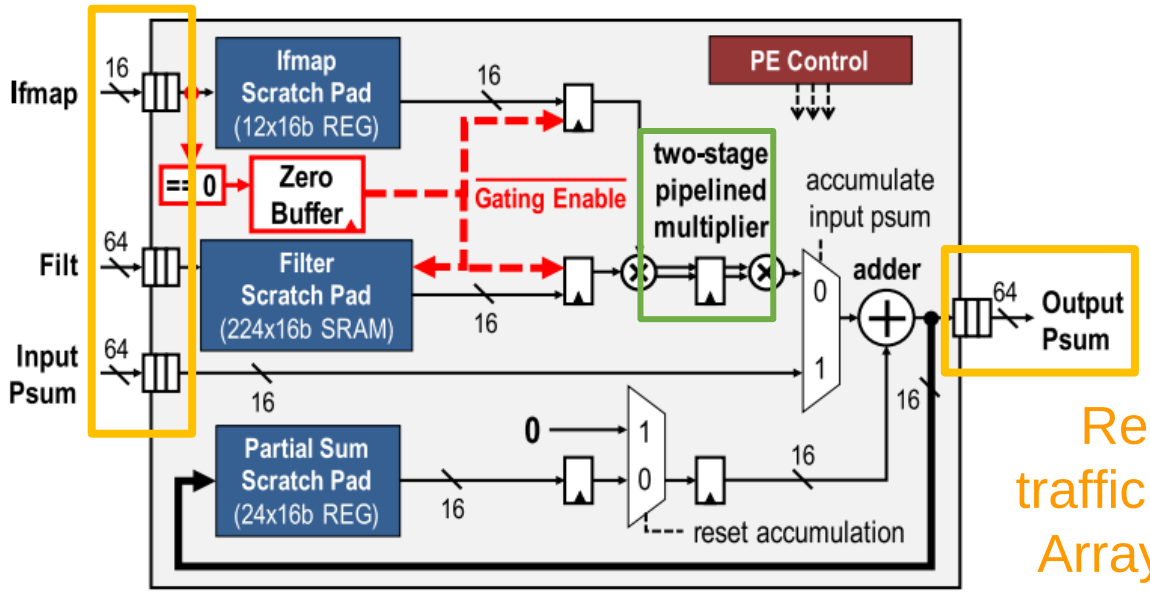


Fig. 12. PE architecture. The datapaths in red show the data gating logic to skip the processing of zero ifmap data.

Reduce data traffic between PE Array and Buffer

Rathore, Hitender Singh et al. "**ASIC Implementation of Two Stage Pipelined Multiplier.**" *International journal of engineering research and technology* 3 (2014): n. pag.

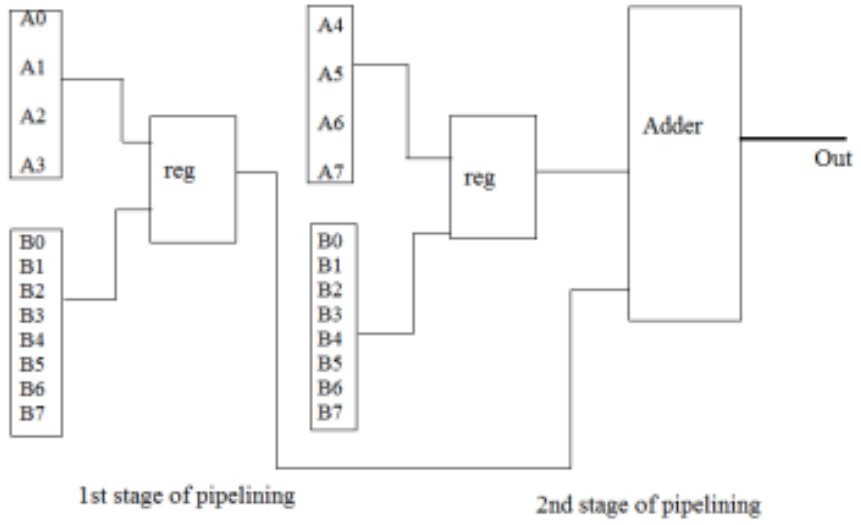


Fig3: Architecture of pipelined multiplier

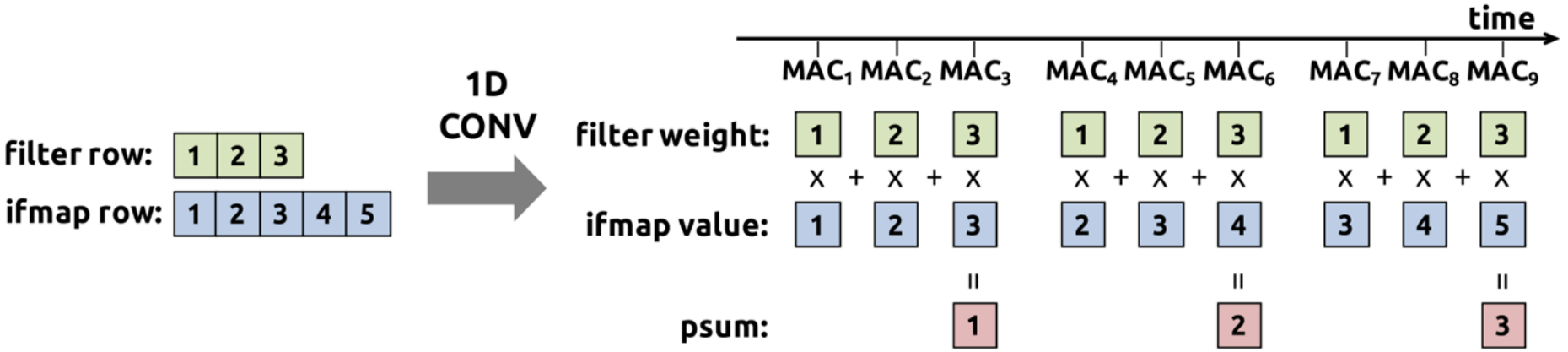
PE Demonstration

1-D Convolution Primitive in a PE

Row Stationary

Each PE receives different row of ifmap and filter

Primitive | One row ifmap * One row filter = One row psum

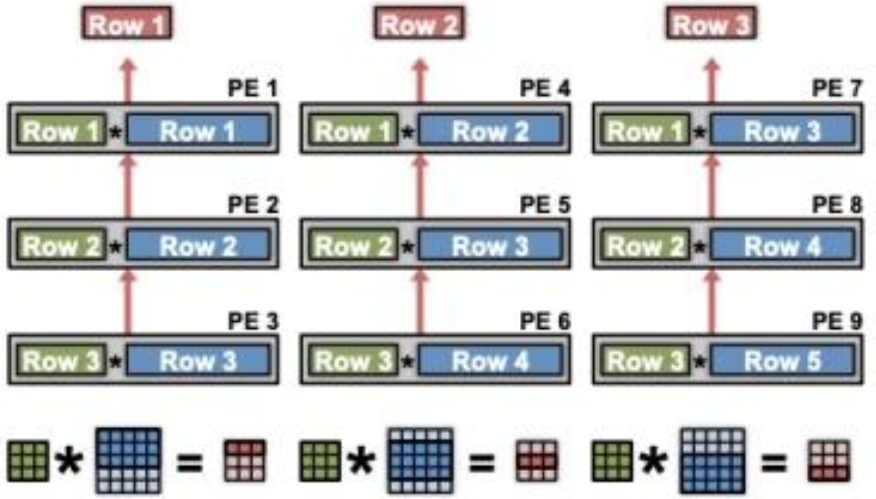


More details in AOC2024spring_lec4(p.208-212)

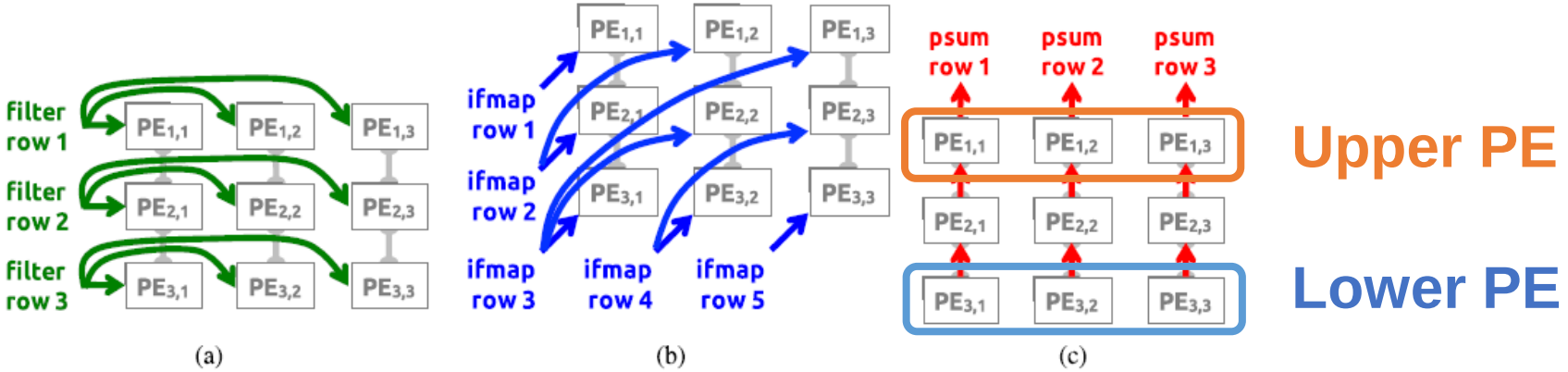
PE Array

- Spatial Architecture **(lec3 p.3-10 / lec4 p.115-124)**
- **Eyeriss V.** SYSTEMMODULES **B. Network-on-Chip**

1) Global Input Network : Buffer **multicast** ifmap, filter, psum to the PE Array
2) Global Output Network : read the psums back to Buffer
3) Local Network : pass the psums between PEs



1 1-channel 5x5 ifmap does convolution with 1 1-channel 3x3 filter.



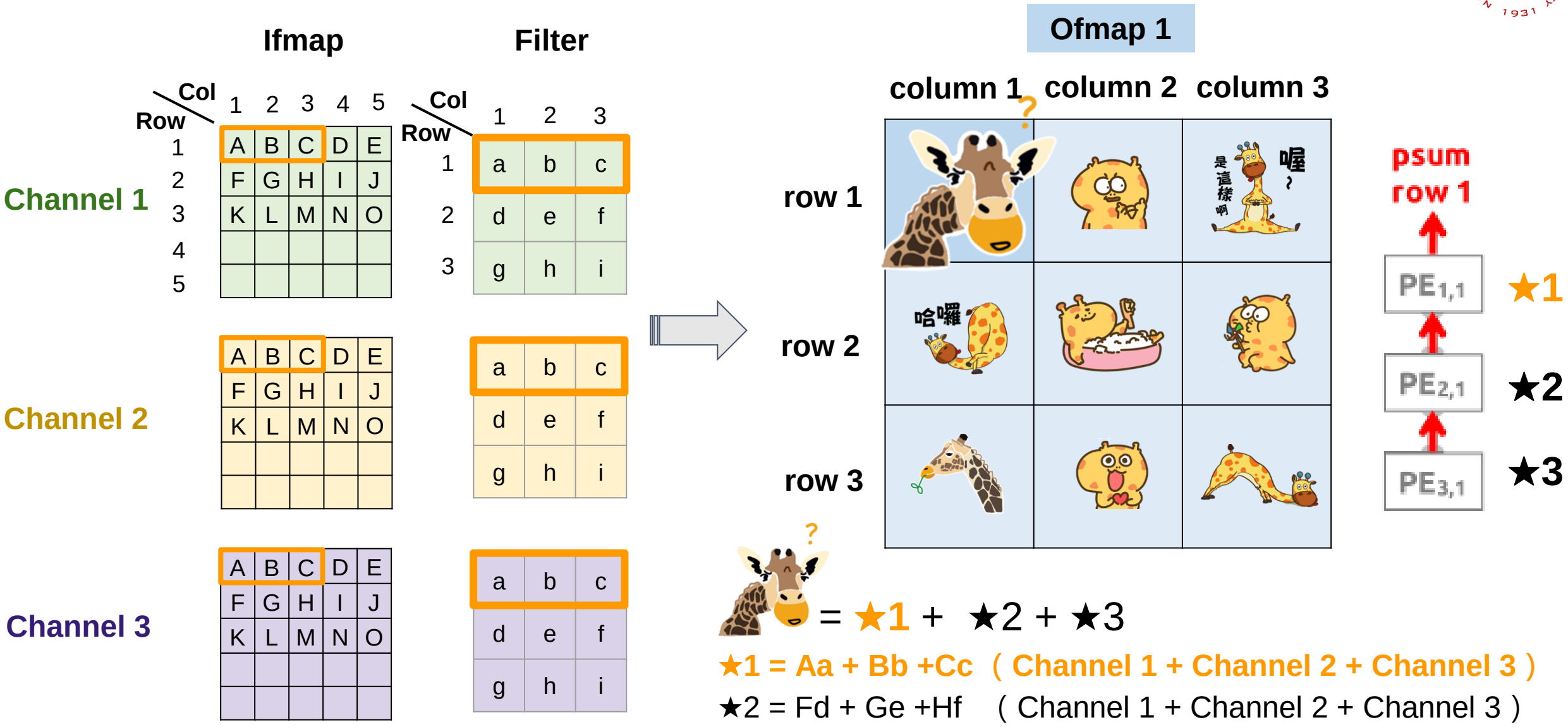
Lower PEs transfer psums to upper PEs.

Fig. 4. Dataflow in a PE set for processing a 2-D convolution. (a) Rows of filter weights are reused across PEs horizontally. (b) Rows of ifmap values are reused across PEs diagonally. (c) Rows of psums are accumulated across PEs vertically. Reuse and accumulation of data within a PE set reduce accesses to the GLB and DRAM, saving data movement energy cost. In this example, the number of filter rows (R), ifmap rows (H), and ofmap rows (E) are 3, 5, and 3, respectively. Therefore, the PE set size is 3×3. Filter and ifmap values from different rows are sent to the PE set in a time-interleaved fashion; all the PEs that reuse the same value receive it at the same cycle. The psums generated from one PE are sent to its neighbor PE immediately.

Outline

- Chapter1 PE Introduction
- **Chapter2 Demonstration Example**
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

PE Demonstration | Example



PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	0	0
ch2	A	0	0
ch3	A	0	0
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	0	0
ch2	0	0	0
ch3	0	0	0
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

A*a

+

A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	0
ch2	A	B	0
ch3	A	B	0
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	0	0
ch2	a	0	0
ch3	0	0	0
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

A*a

+

A*a

A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

B*b

+

A*a

A*a

A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	0
ch2	a	0	0
ch3	a	0	0
ch4	0	0	0

	Ifmap	Filter
Channel 1	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 2	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 3	A B C D E F G H I J K L M N O	a b c d e f g h i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

B*b

+

B*b
A*a
A*a
A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	0
ch2	a	b	0
ch3	a	0	0
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

B*b

+

B*b
B*b
A*a
A*a
A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	0
ch2	a	b	0
ch3	a	b	0
ch4	0	0	0

	Ifmap	Filter
Channel 1	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 2	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 3	A B C D E F G H I J K L M N O	a b c d e f g h i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

C*c

+

B*b
B*b
B*b
A*a
A*a
A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	0
ch3	a	b	0
ch4	0	0	0

	Ifmap	Filter
Channel 1	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 2	A B C D E F G H I J K L M N O	a b c d e f g h i
Channel 3	A B C D E F G H I J K L M N O	a b c d e f g h i

PE Demonstration | Example

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

C*c

+

C*c

B*b

B*b

B*b

A*a

A*a

A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	0
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

Minimum required time for computing : 9 / 12 cycles
9 cycles -> filter / ifmap spad : 3 channels (1st layer)
12 cycles -> filter / ifmap spad : 4 channels (most layers)

Calculate ★1

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

C*c + C*c + C*c + B*b + B*b + B*b + A*a + A*a + A*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Ifmap

Filter

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

a	b	c
d	e	f
g	h	i

PE Demonstration | Example

After finishing ★1, we need to do the following operations in Eyeriss architecture.

🐾 **Update ifmap spad based on stride(U) and get new ifmaps.**

Since stride is 1 in the demonstration example, we shift ifmap spad by 1 column.

🐾 **Keep filter spad for calculating different columns psums along the same row.**

Since we don't have to replace weight by a new kernel or another 4 channels, we retain our filters.

🐾 **Receive psum (★2+★3) from the lower PE, update psum again and send out updated psum.**

We assume that PE2,1 and PE3,1 have finished calculating ★ 2 and ★3, respectively.

Besides, PE2,1 has updated its psum to (★2+★3) and transfered it to PE1,1.

Hint : As for a pipelined PE, it can do a multiplication (B*a) at the same time.

= updated psum = ★1 + ★2 + ★3

★1 = Aa + Bb +Cc (Channel1 + Channel2 + Channel3)

★2 = Dd + Ee +Ff (Channel1 + Channel2 + Channel3)

★3 = Gg + Hh +Ii (Channel1 + Channel2 + Channel3)

PE Demonstration | Example

Calculate updated psum and row1/column2 psum

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

★1

+

(★2+★3)

B*a

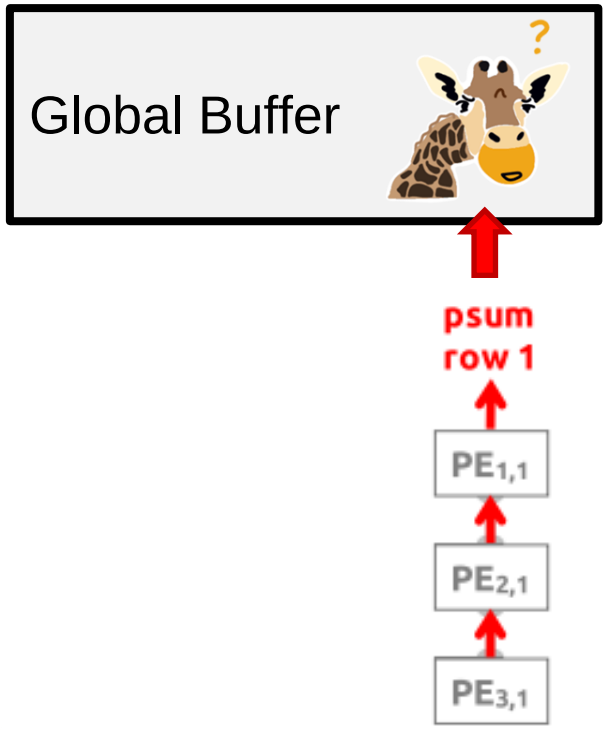
Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	B	C	D
ch2	B	C	D
ch3	B	C	D
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Subsequent to the completion of updated psum, we store it in Global Buffer.



= updated psum

= ★1 + ★2 + ★3

PE Demonstration | Example

Calculate row1 column2 ofmap by repeating the above steps

Ofmap 1

	column 1	column 2	column 3
row 1			
row 2			
row 3			

MAC operation

B*a

+

B*a

Ifmap Spad

	Col x	Col (x+1)	Col (x+2)
ch1	B	C	D
ch2	B	C	D
ch3	B	C	D
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Ifmap

Channel 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

Channel 2

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

Channel 3

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O	

Filter

a	b	c
d	e	f
g	h	i

a	b	c
d	e	f
g	h	i

a	b	c
d	e	f
g	h	i

Outline

- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- **Chapter3 Dataflow apply to VGG16-Cifar10 1st layer**
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

Parameter Setup

In a pass, each input data are read only once from the GLB, and the psums are stored back to the GLB only once when the processing is finished.

-- -- **Eyeriss IV.** ENERGY-EFFICIENT FEATURES **A.** Energy-Efficient Dataflow: Row Stationary

2) Multiple PE Sets in the PE Array **a)** PE array processing passes

- The parameter setting is listed below.

n = 1 ： A PE processes 1 ifmap at a time.

e = 8 ： A PE set is consisted of 3x8 PEs.

p = 1 ： 1 kernel stays in a PE at a time.

q = 4 ： A PE accumulates 4 channels psums consecutively.

TABLE II

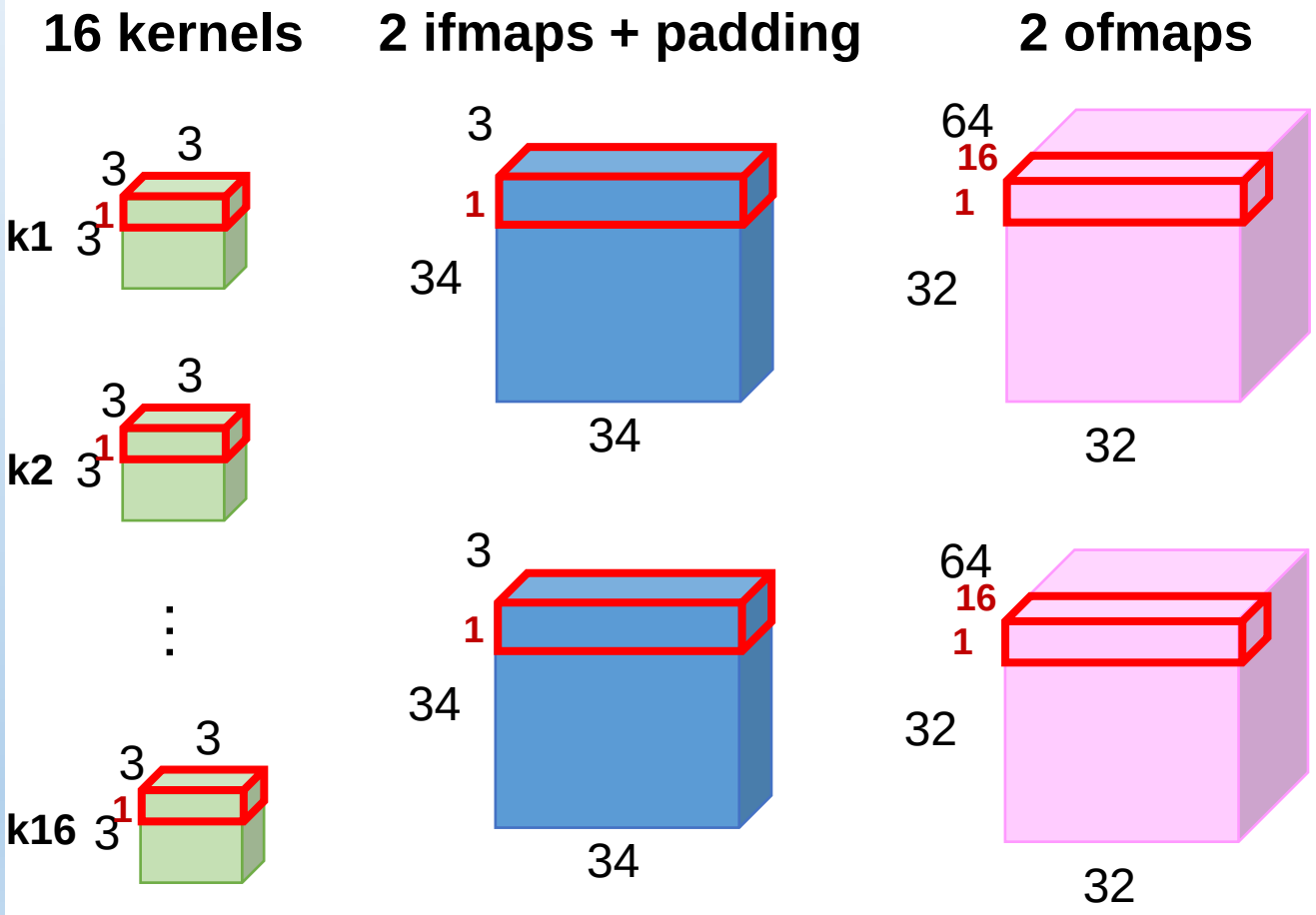
MAPPING PARAMETERS OF THE RS DATAFLOW

Parameter	Description
m	number of ofmap channels stored in the global buffer
n	number of ifmaps used in a processing pass
e	width of the PE set (strip-mined if necessary)
p	number of filters processed by a PE set
q	number of channels processed by a PE set
r	number of PE sets that process different channels in the PE array
t	number of PE sets that process different filters in the PE array

- We will **complete doing convolution with 1 row of 1 kernel and accumulating 3 channels.**
- We set up those parameters by cooperating with model VGG16 and dataset Cifar10 .
- After setting up the parameters, our dataflow and PE design synchronize with them.

Dataflow for VGG16-Cifar10

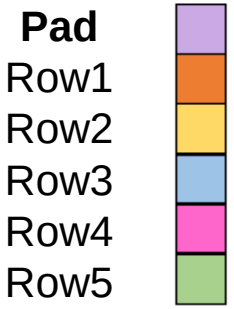
- In the next pages, we demonstrate how VGG16-Cirfar10 applies on the PE1,1.
- Assume a PE set receives 16 kernels that corresponding ofmap channels are consecutive.



Filter distributed on 3x8 PE Array

Row 1	**PE1,1**	**PE1,2**	…	…	…	…	…	**PE1,8**
Row 2	PE2,1	PE2,2	…	…	…	…	…	PE2,8
Row 3	PE3,1	PE3,2	…	…	…	…	…	PE3,8

Ifmap distributed on 3x8 PE Array



Pad	1	2	3	4	5	6	7
Row1	2	3	4	5	6	7	8
Row2	3	4	5	6	7	8	9

$PE_{1,1}$ Demonstration

- Firstly, we accumulate 3 channels psums, wait for PE2,1 send corresponding psum to PE1,1 and transfer updated psum to Global Buffer.

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

A*a + B*b + C*c

+

A*a + B*b + C*c

+

A*a + B*b + C*c

Ifmap Spad

	Col 1	Col 2	Col 3
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Ifmap 1

34

34

Channel 1

A	B	C	D	…	G
H	I	J	K	…	N
O	P	Q	R	…	U
…	…				

Channel 2

A	B	C	D	…	G
H	I	J	K	…	N
O	P	Q	R	…	U
…	…				

Channel 3

A	B	C	D	…	G
H	I	J	K	…	N
O	P	Q	R	…	U
…	…				

Kernel 1

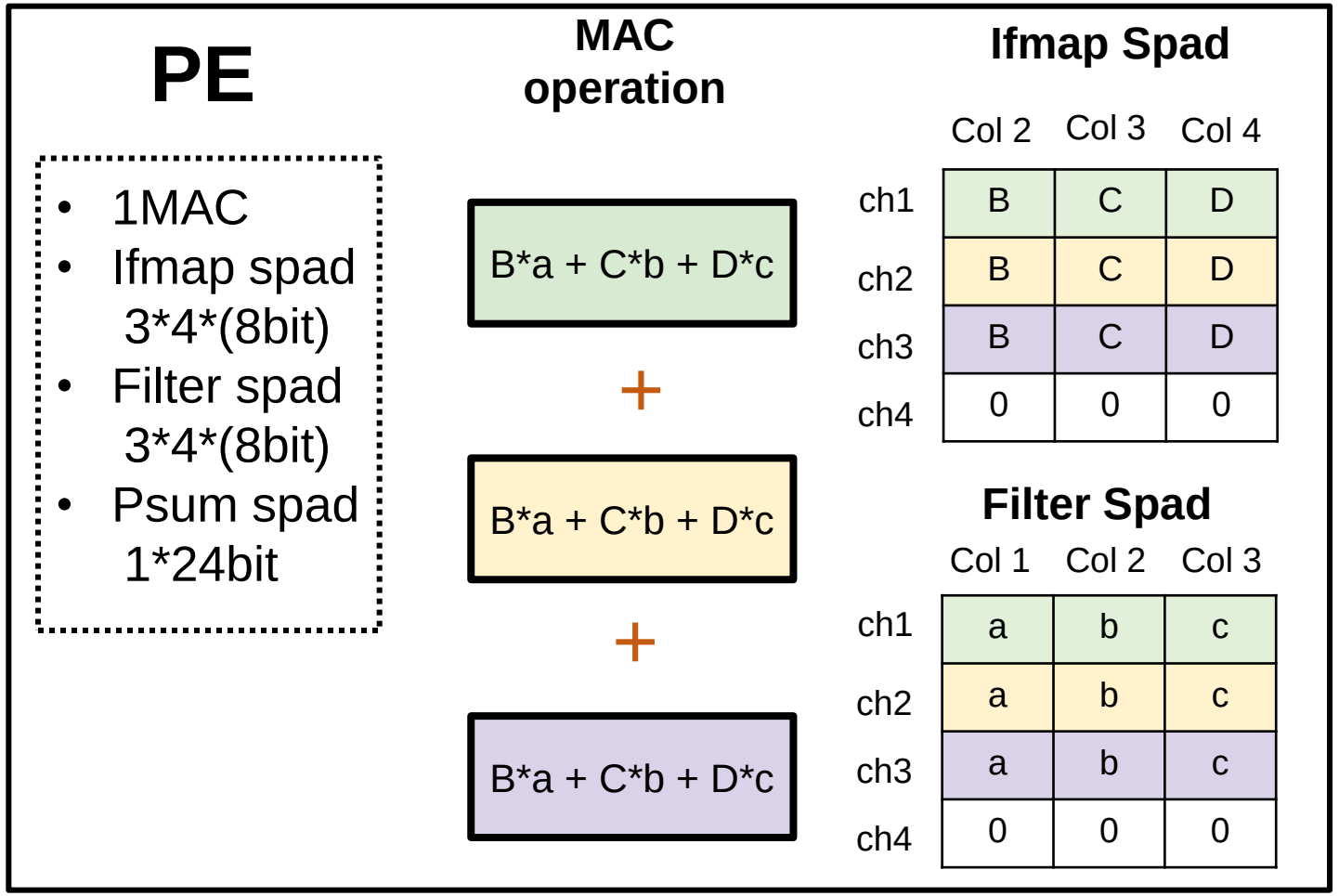
a	b	c
d	e	f
g	h	i

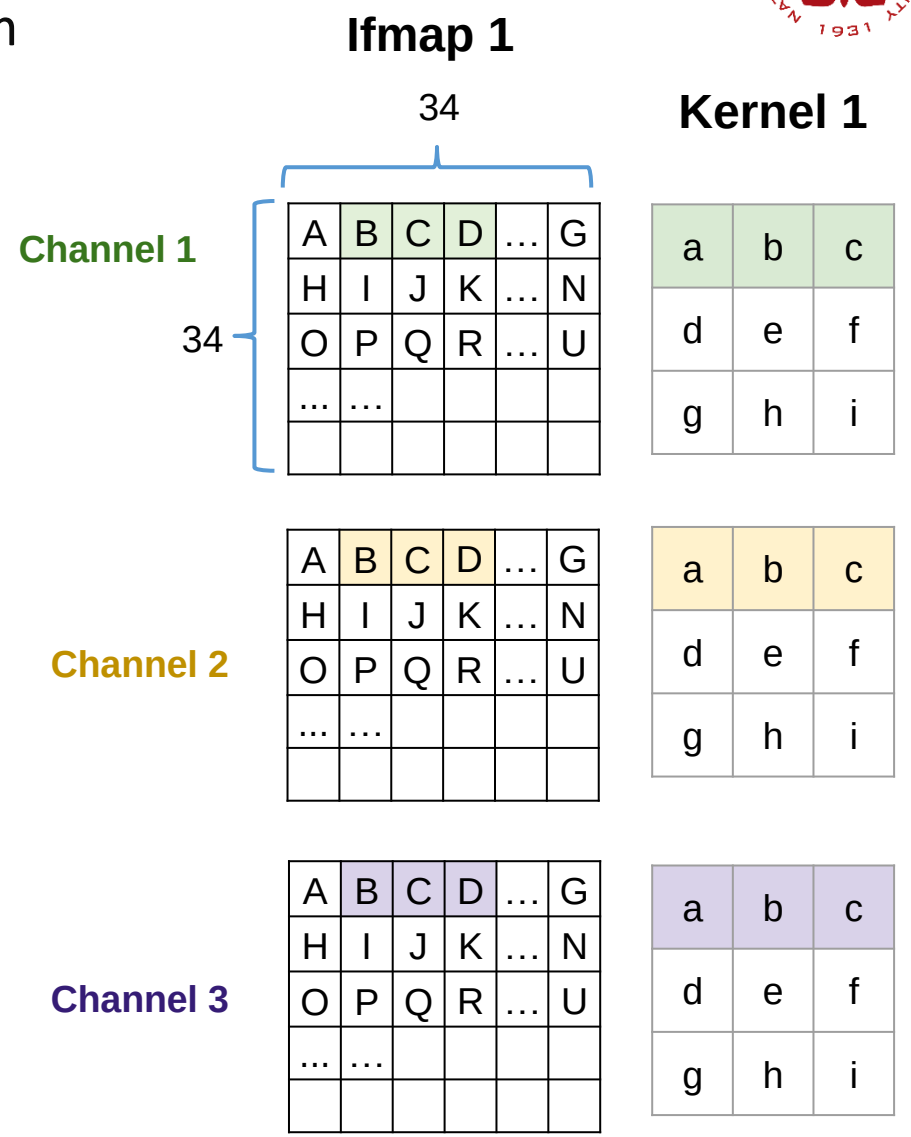
a	b	c
d	e	f
g	h	i

a	b	c
d	e	f
g	h	i

$PE_{1,1}$ Demonstration

- Then, we update the ifmap spad, receive incoming ifmaps from Buffer and compute the updated psum.





$PE_{1,1}$ Demonstration

- We keep shifting our ifmap spad by 1 column until the end of 1st row ifmap0, receives psums from lower PEs and store ofmaps in the Buffer.
- **Here, we have completed a processing pass.**

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

E*a + F*b + G*c

+

E*a + F*b + G*c

+

E*a + F*b + G*c

Ifmap Spad

	Col 32	Col 33	Col 34
ch1	E	F	G
ch2	E	F	G
ch3	E	F	G
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Ifmap 1

34

34

Channel 1

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Channel 2

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Channel 3

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Kernel 1

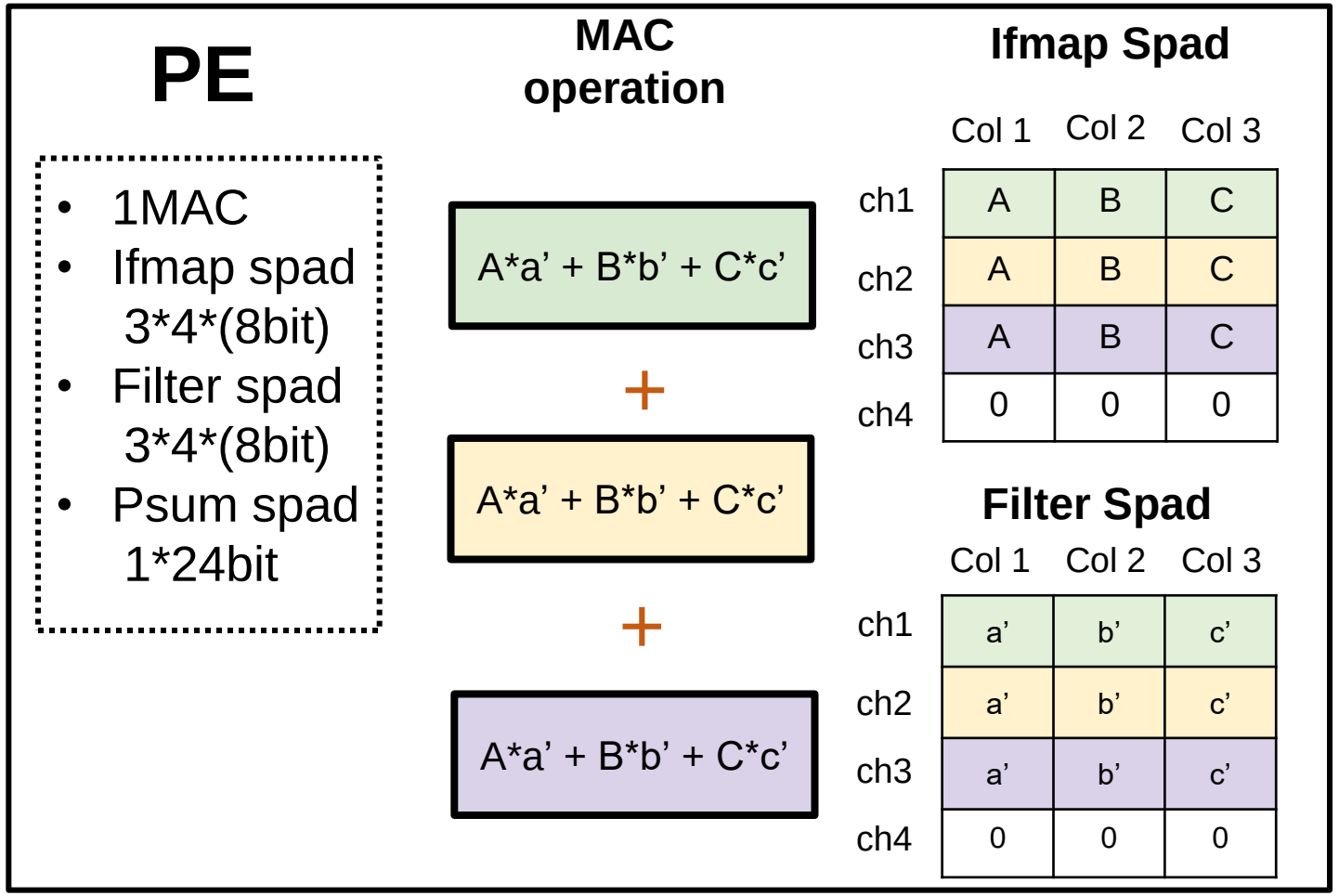
a	b	c
d	e	f
g	h	i

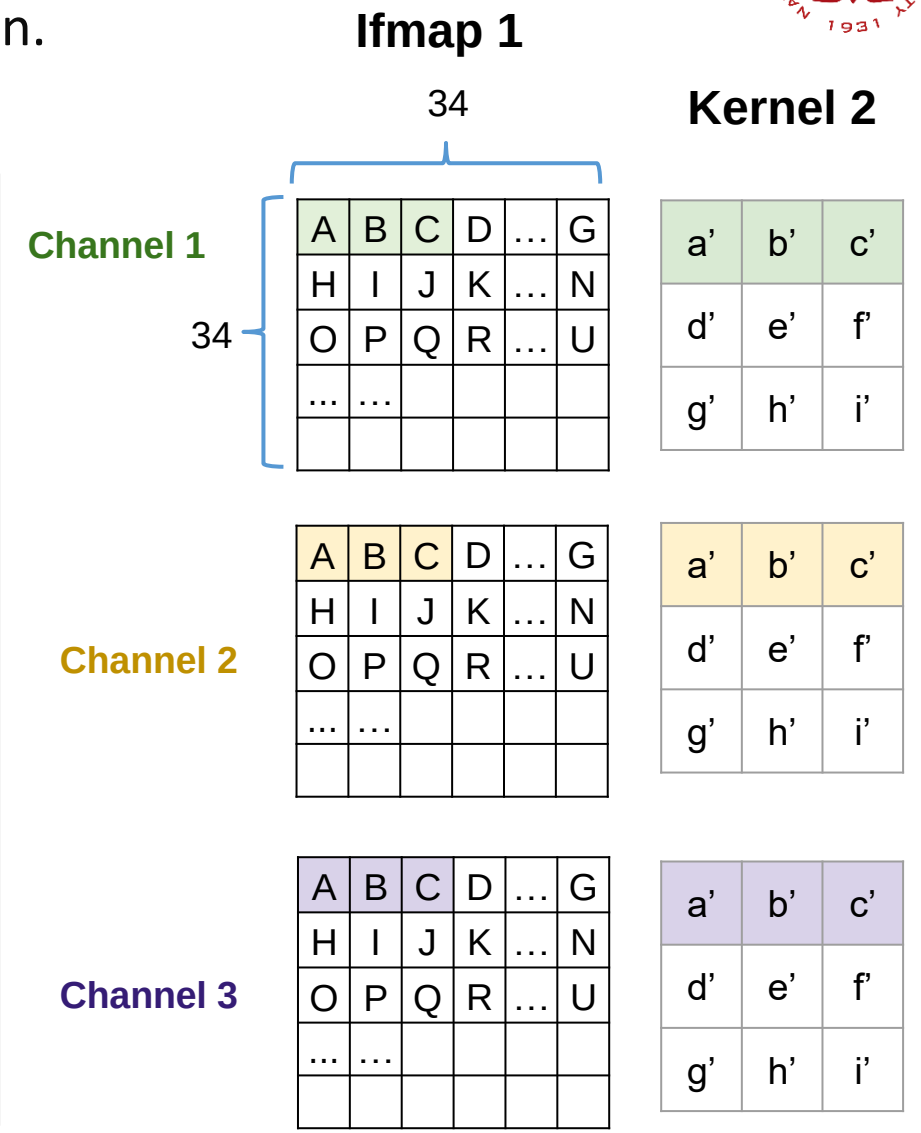
a	b	c
d	e	f
g	h	i

a	b	c
d	e	f
g	h	i

PE1,1 Demonstration

- Secondly, Ifmap Buffer sends 1st row ifmap0 from scratch again.
- PE starts to receive next kernels.





PE1,1 Demonstration

- We keep computing psums until the end of 1st row.

PE

- 1MAC
- Ifmap spad 3*4*(8bit)
- Filter spad 3*4*(8bit)
- Psum spad 1*24bit

MAC operation

E*a’ + F*b’ + G*c’

+

E*a’ + F*b’ + G*c’

+

E*a’ + F*b’ + G*c’

Ifmap Spad

	Col 32	Col 33	Col 34
ch1	E	F	G
ch2	E	F	G
ch3	E	F	G
ch4	0	0	0

Filter Spad

	Col 1	Col 2	Col 3
ch1	a’	b’	c’
ch2	a’	b’	c’
ch3	a’	b’	c’
ch4	0	0	0

Ifmap 1

34

34

Channel 1

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Channel 2

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Channel 3

A	…	..	E	F	G
H	…	…	L	M	N
O	…	…	S	T	U
…	…				

Kernel 2

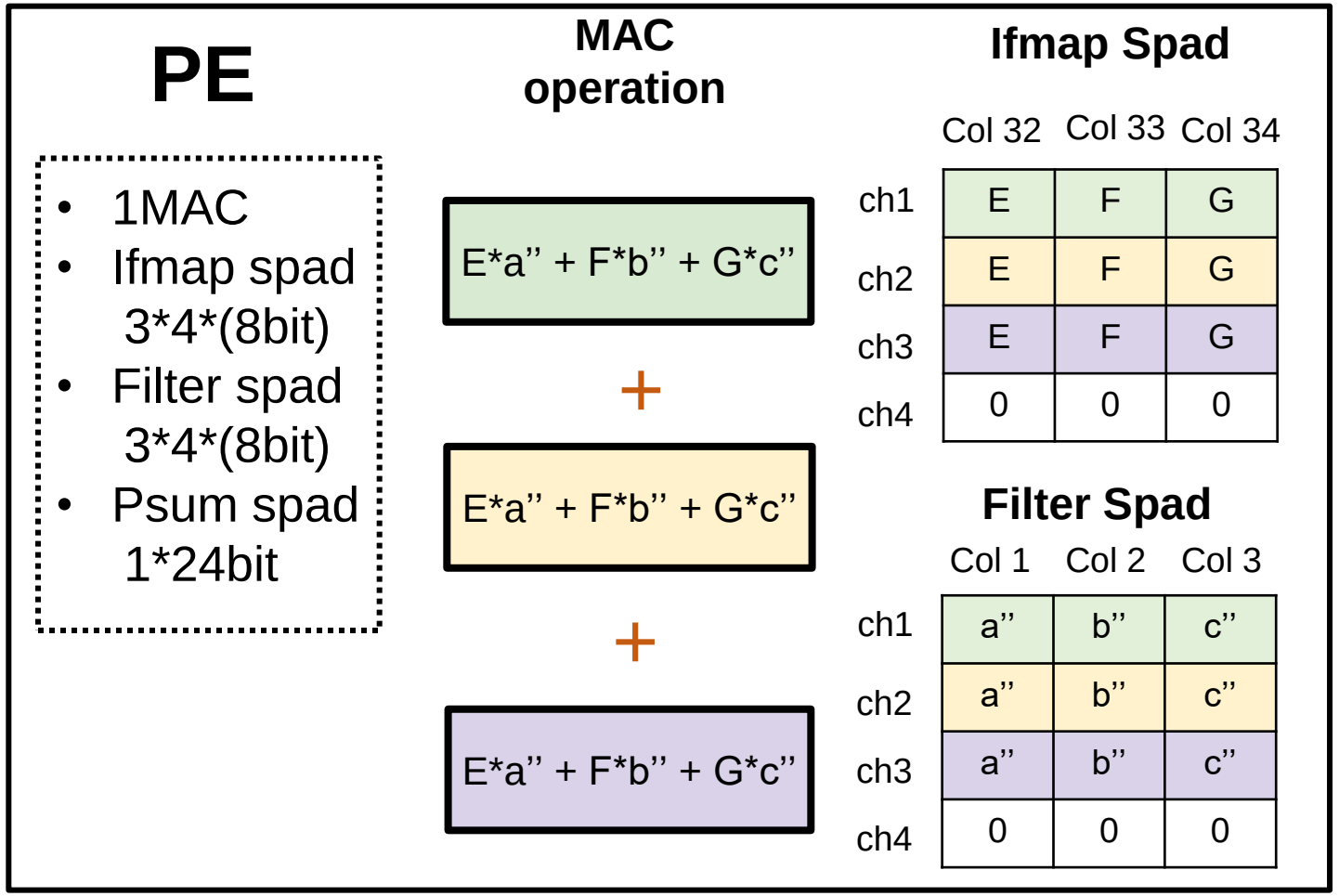
a’	b’	c’
d’	e’	f’
g’	h’	i’

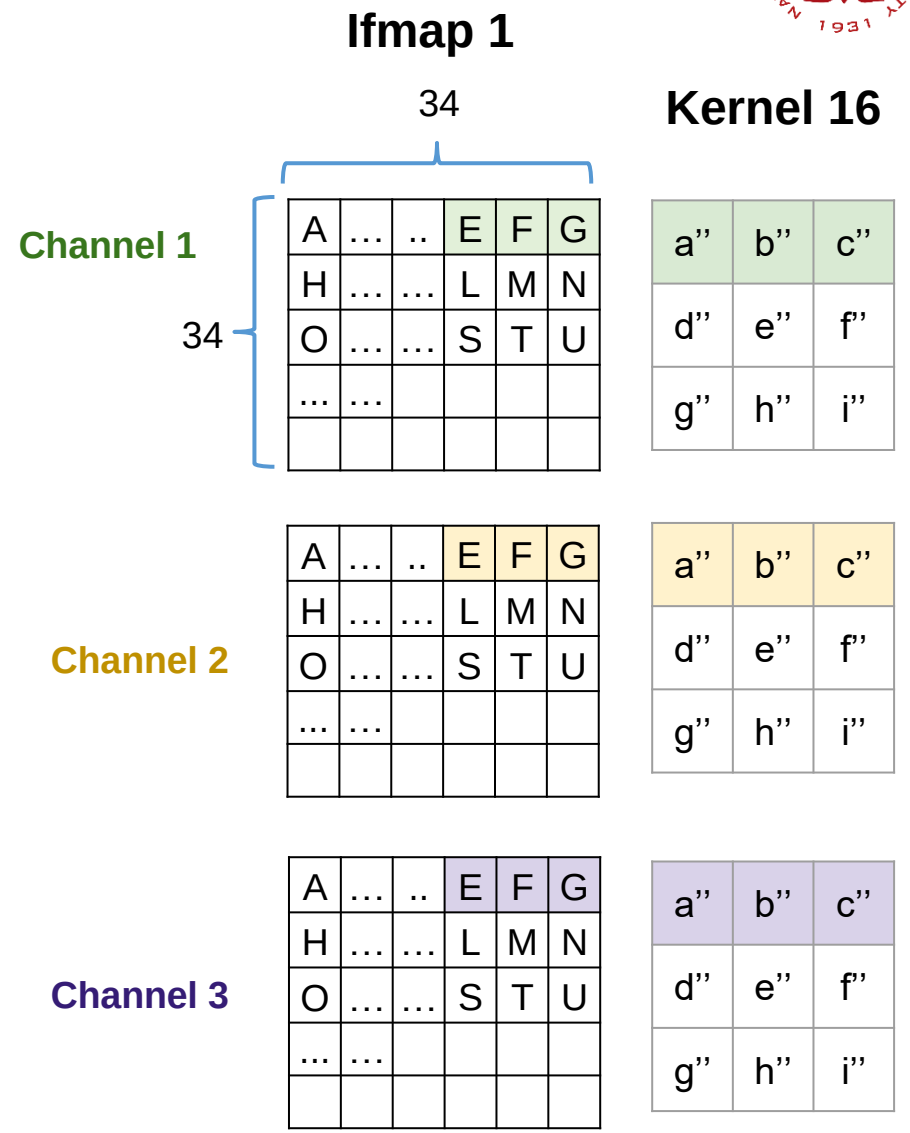
a’	b’	c’
d’	e’	f’
g’	h’	i’

a’	b’	c’
d’	e’	f’
g’	h’	i’

$PE_{1,1}$ Demonstration

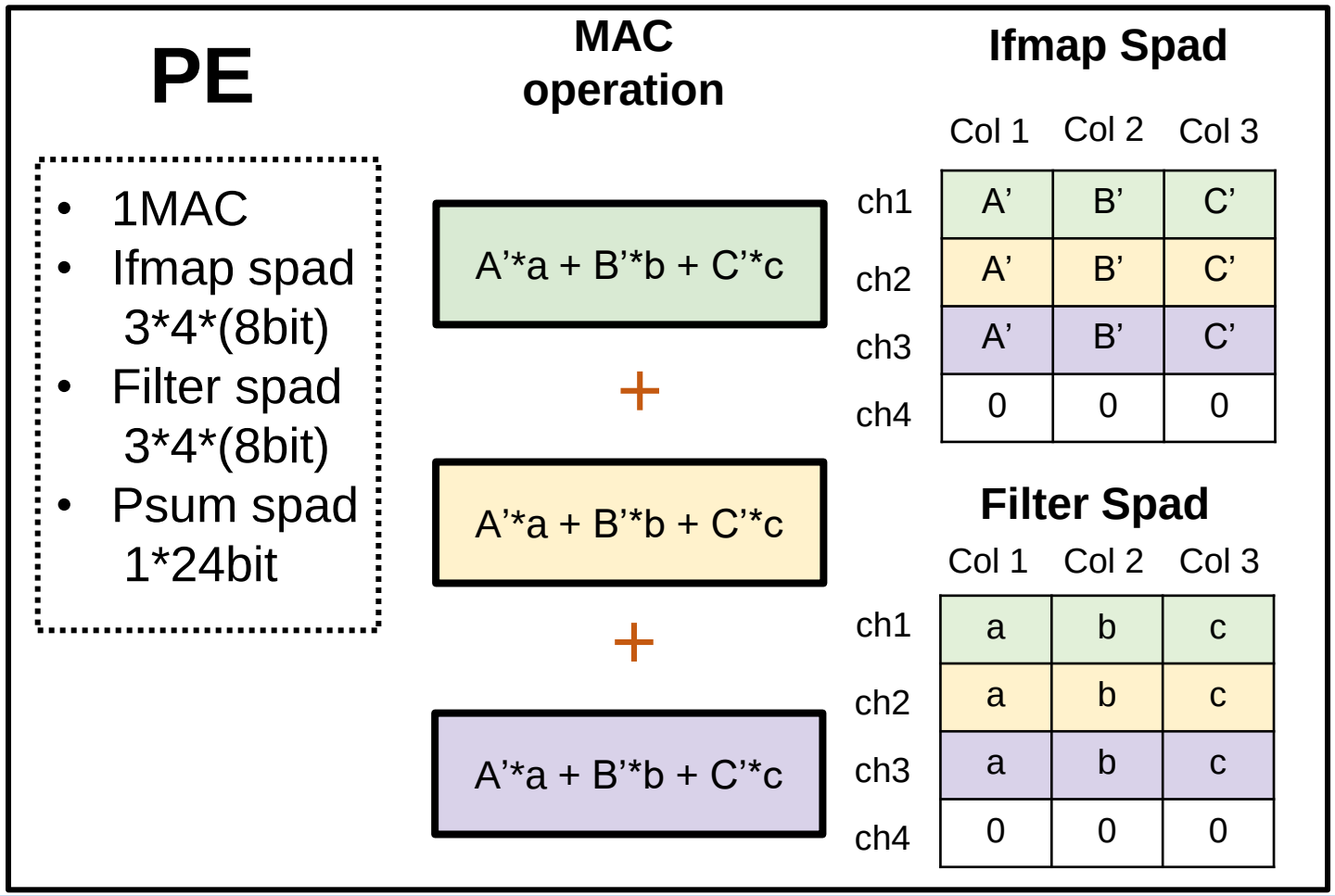
- Ifmap Buffer sends 1st row ifmap0 for 16 times.
- 1st row ifmap0 does convolution with 16 kernels.

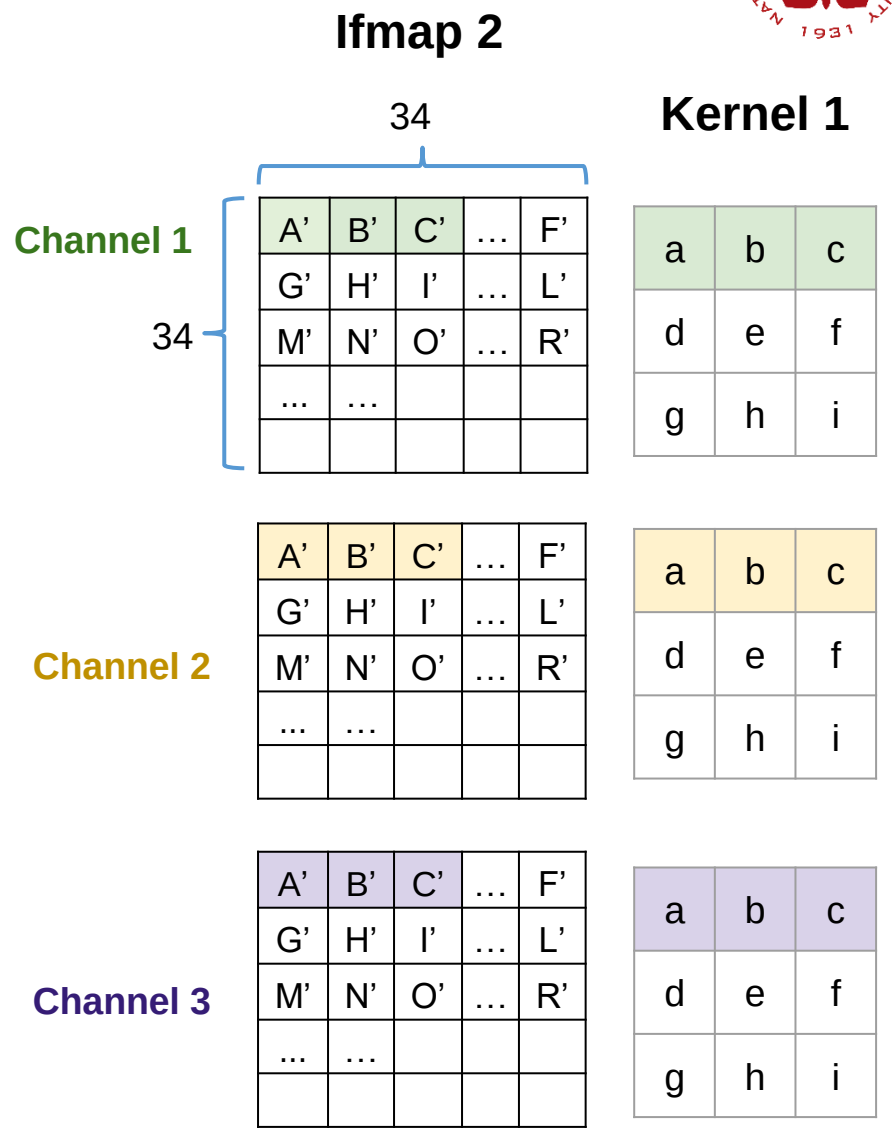




PE1,1 Demonstration

- At the end of processing ifmap0, Ifmap Buffer begins to send 1st row ifmap1 for 16 times. Weight Buffer sends 16 kernels again. Finally, PE1,1 finishes 32 processing passes.





Outline

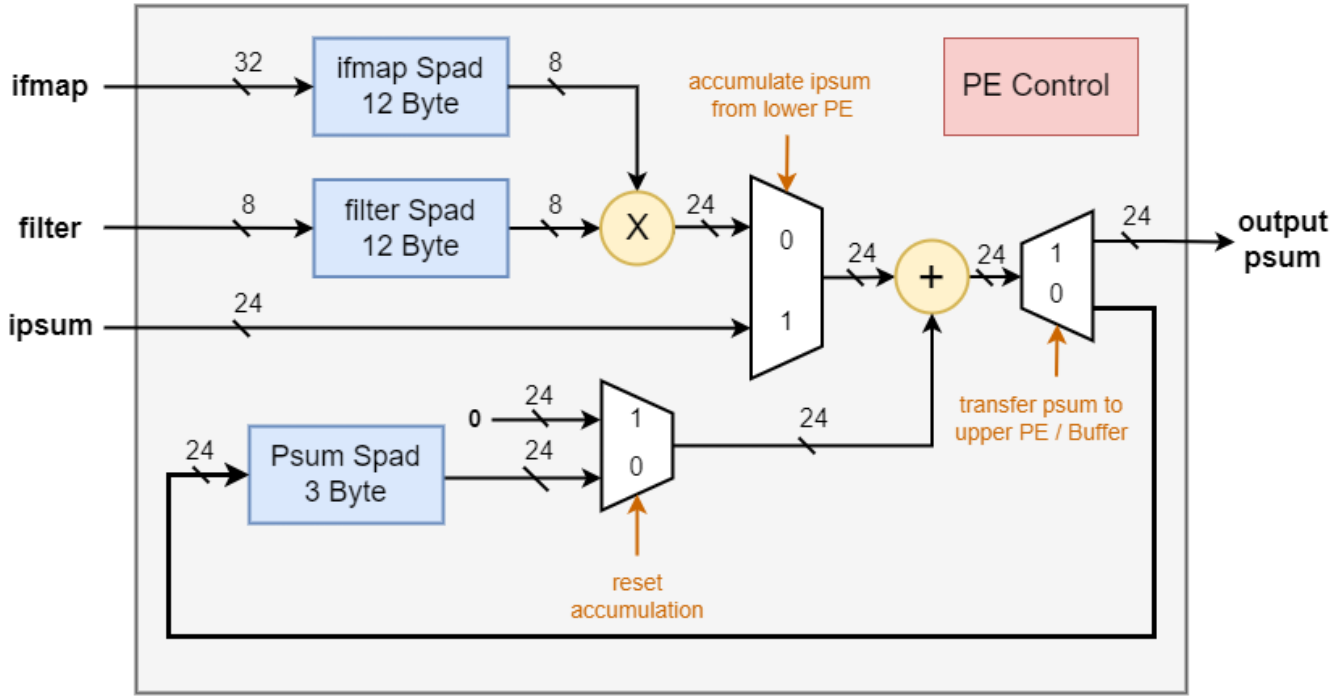
- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- **Chapter4 PE Architecture**
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

PE Architecture

The following figure is a reference for you to design your own architecture.

Your design must under the following restriction :

- 1 MAC
- Maximum 50 Byte Spad in total
- 32 bit ifmap
- 8 bit filter
- 24 bit input psum
- 24 bit output psum
- Add or remove other logic devices is allowed



Parameter Setting

- Eyriss paper parameter setting design principles can be refer to paper[32], Table II and Table III.

[32]Y.-H. Chen, J. Emer, and V. Sze, "**Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks**," in Proc. 43rd Annu. Int. Symp. Comput. Archit. (ISCA), 2016, pp. 367–379.

- **Optimize GLB size, spad size, and number of PEs**
- **Optimize Data Access energy by three forms of data reuse**

(**Eyeriss IV.** ENERGY-EFFICIENT FEATURES **A.** Energy-Efficient Dataflow: Row Stationary)

1) Convolutional Reuse: Each filter weight is reused E×F times in the same ifmap plane, and each ifmap pixel is usually reused R × S times in the same filter plane.

2) Filter Reuse: Each filter weight is reused across the batch of N ifmaps.

3) Ifmap Reuse: Each ifmap pixel is reused across M filters.

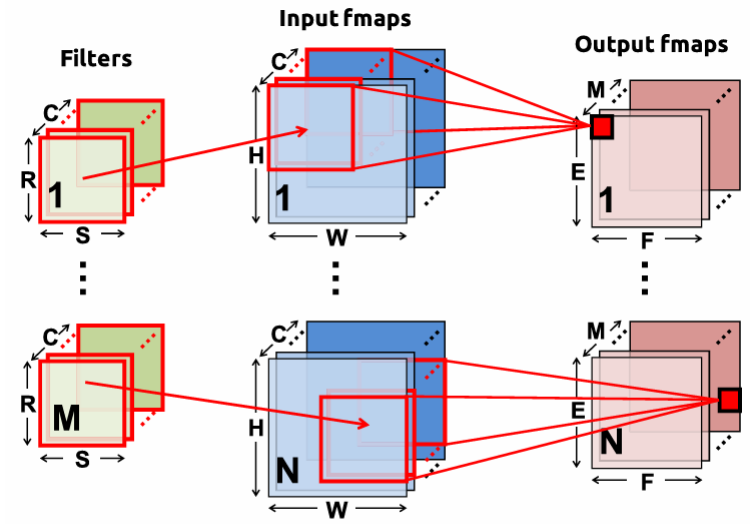


Fig. 1. Computation of a CNN layer.

Parameter Setting & PE Spad Size Design

Eyeriss Chapter IV. ENERGY-EFFICIENT FEATURES

A. Energy-Efficient Dataflow: Row Stationary

What if we want to complete scenario C under n=2, p=16, q=4 ?

- **In a pass, each input data are read only once from the GLB, and the psums are stored back to the GLB only once when the processing is finished.**
- Each PE runs p × q primitives simultaneously from q different channels of p different filters. The required spad capacity for each data type is:
- 1) **p×q×S** for the rows of **filter** weights from q channels of p filters
- 2) **q × S** for q sliding windows of **ifmap** values from q different channels
- 3) **p** for the accumulation of **psum**s in p ofmap channels
- S -- Sliding Window

Parameter	Description
m	number of ofmap channels stored in the global buffer
n	number of ifmaps used in a processing pass
e	width of the PE set (strip-mined if necessary)
p	number of filters processed by a PE set
q	number of channels processed by a PE set
r	number of PE sets that process different channels in the PE array
t	number of PE sets that process different filters in the PE array

Parameter Setting & PE Spad Size Design

PE

- **1MAC**
- **Ifmap spad**
  3col*4ch*(8bit)
- **Filter spad**
  3col*4ch*16kernel*(8bit)
- **Psum spad**
  16ch*(24bit)

Ifmap Spad

	Col 1	Col 2	Col 3
ch1	A	B	C
ch2	A	B	C
ch3	A	B	C
ch4	0	0	0

Psum Spad

ch1	P1
ch2	P2
⋮	⋮
ch16	P16

Filter Spad

Kernel 1

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Kernel 2

	Col 1	Col 2	Col 3
ch1	a’	b’	c’
ch2	a’	b’	c’
ch3	a’	b’	c’
ch4	0	0	0

⋮

Kernel 16

	Col 1	Col 2	Col 3
ch1	a’’	b’’	c’’
ch2	a’’	b’’	c’’
ch3	a’’	b’’	c’’
ch4	0	0	0

IDLE Cycle & MAC Operation

#1 P1 = A*a

#2 P1 = A*a + A*a

#3 P1 = A*a + A*a + A*a

#4 P1 = A*a + A*a + A*a + B*b

… …

#12 P1 = A*a + A*a + A*a + B*b + B*b + B*b + C*c + C*c + C*c

… …

#(12*2) P2 = A*a’ + A*a’ + A*a’ + B*b’ + B*b’ + B*b’ + C*c’ + C*c’ + C*c’

… …

#(12*16 = 192) complete P16

In clock 192, we complete 16 channels of column 1 ofmap.

Parameter Setting & PE Spad Size Design

PE

- **1MAC**
- **Ifmap spad**
  3col*4ch*(8bit)
- **Filter spad**
  3col*4ch*16kernel*(8bit)
- **Psum spad**
  16ch*(24bit)

Ifmap Spad

	Col 2	Col 3	Col 4
ch1	B	C	D
ch2	B	C	D
ch3	B	C	D
ch4	0	0	0

Psum Spad

ch1	P1
ch2	P2
⋮	⋮
ch16	P16

Filter Spad

Kernel 1

	Col 1	Col 2	Col 3
ch1	a	b	c
ch2	a	b	c
ch3	a	b	c
ch4	0	0	0

Kernel 2

	Col 1	Col 2	Col 3
ch1	a’	b’	c’
ch2	a’	b’	c’
ch3	a’	b’	c’
ch4	0	0	0

⋮

Kernel 16

	Col 1	Col 2	Col 3
ch1	a’’	b’’	c’’
ch2	a’’	b’’	c’’
ch3	a’’	b’’	c’’
ch4	0	0	0

IDLE Cycle & MAC Operation

… …

#(12*17) complete P1

… …

#(12*18) complete P2

… …

#(12*32 = 384) complete P16

In clock 384, we complete 16 channels of column 2 ofmap.

You can derive the following steps !

Outline

- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 Architecture
- **Chapter5 3 scenarios of accumulating psum**
- Chapter6 Homework
- Chapter7 Bonus
- Chapter8 Supplementary

3 Scenarios of Accumulating Psum

- There are three ways to finish a psum within a PE.
- The above example demonstrates scenario(C).
- These 3 scenarios have different dataflows.

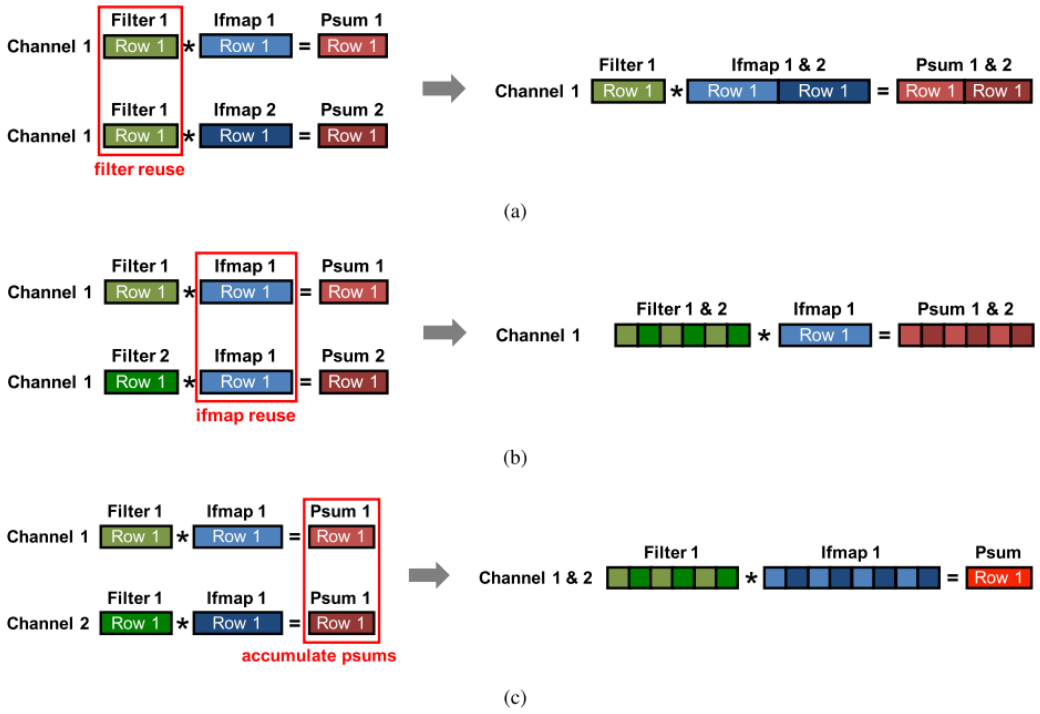


Fig. 6. Handling the dimensions beyond 2-D in each PE by (a) concatenating the ifmap rows, each PE can process multiple 1-D primitives with different ifmaps and reuse the same filter row and (b) time interleaving the filter rows, each PE can process multiple 1-D primitives with different filters and reuse the same ifmap row. (c) By time interleaving the filter and ifmap rows, each PE can process multiple 1-D primitives from different channels and accumulate the psums together.

Scenario A Demonstration

- We will demonstrate how to apply scenario (a) filter reuse to **compute row 1 column 1, 2, 3 ofmap1,2** under **parameters n=2 (batch size), p=1 (1 kernel), q=1(1 channel)**
- To complete doing convolution with 1 row of kernels and accumulating 3 channels.

(a)

Channel 1,2,3

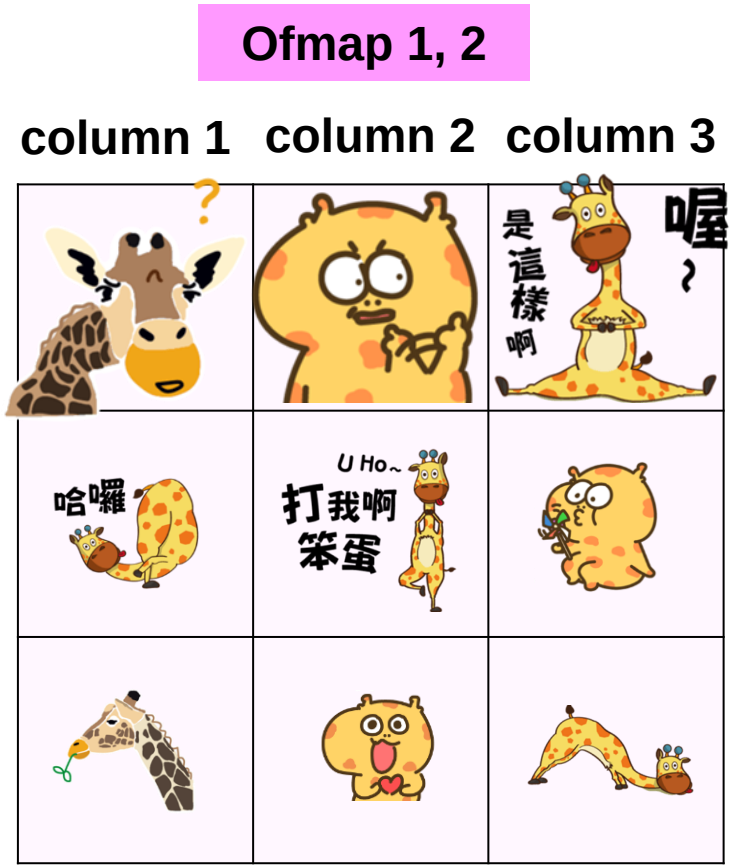
Ifmap 1, 2

Row \ Col	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	K	L	M	N	O
4					
5					

Filter

Row \ Col	1	2	3
1	a	b	c
2	d	e	f
3	g	h	i

Ofmap 1, 2



Scenario A Demonstration

- X = ★Px1 + ★Px2 + ★Px3 + ★Px1 + ★Px2 + ★Px3 + ★Px1 + ★Px2 + ★Px3
- X’ = ★Px1’ + ★Px2’ + ★Px3’ + ★Px1’ + ★Px2’ + ★Px3’ + ★Px1’ + ★Px2’ + ★Px3’
- ★Px1 = A*a + B*b + C*c (channel 1) (channel 2) (channel 3)
- ★Px1’ = A’*a + B’*b + C’*c (channel 1) (channel 2) (channel 3)
- ★Px2 = F*d + G*e + H*f (channel 1) (channel 2) (channel 3)
- ★Px2’ = F’*d + G’*e + H’*f (channel 1) (channel 2) (channel 3)
- ★Px3 = K*g + L*h + M*i (channel 1) (channel 2) (channel 3)
- ★Px3’ = K’*g + L’*h + M’*i (channel 1) (channel 2) (channel 3)

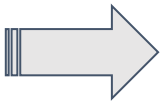
Channel 1,2,3

Ifmap 1, 2

Row \ Col	1	2	3	4	5
1	A	B	C	D	E
2	F	G	H	I	J
3	K	L	M	N	O
4					
5					

Filter

Row \ Col	1	2	3
1	a	b	c
2	d	e	f
3	g	h	i



Ofmap 1

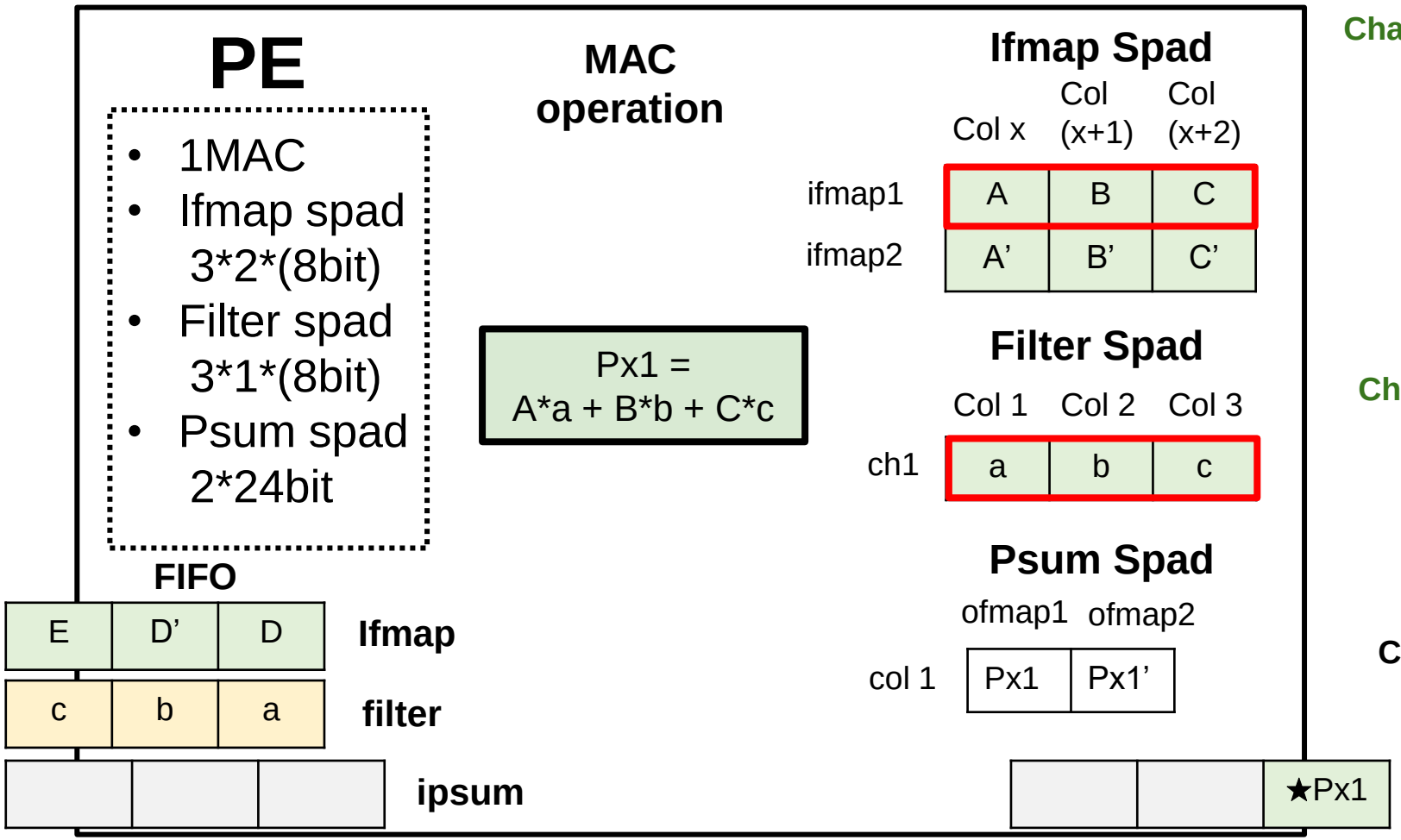
Row \ Col	1	2	3
1	X	Y	Z
2			
3			

Ofmap 2

Row \ Col	1	2	3
1	X’	Y’	Z’
2			
3			

Scenario A Demonstration

- Assume PE1,1 has received 6 ifmaps and 3 weights from FIFO.
- We spend 3 clks computing Px1.



Channel 1

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

A'	B'	C'	D'	E'
F'	G'	H'	I'	J'
K'	L'	M'	N'	O'

Channel 1

Kernel 1

a	b	c
d	e	f
g	h	i

Kernel 2

a'	b'	c'
d'	e'	f'
g'	h'	i'

Channel 1

Ofmap 1

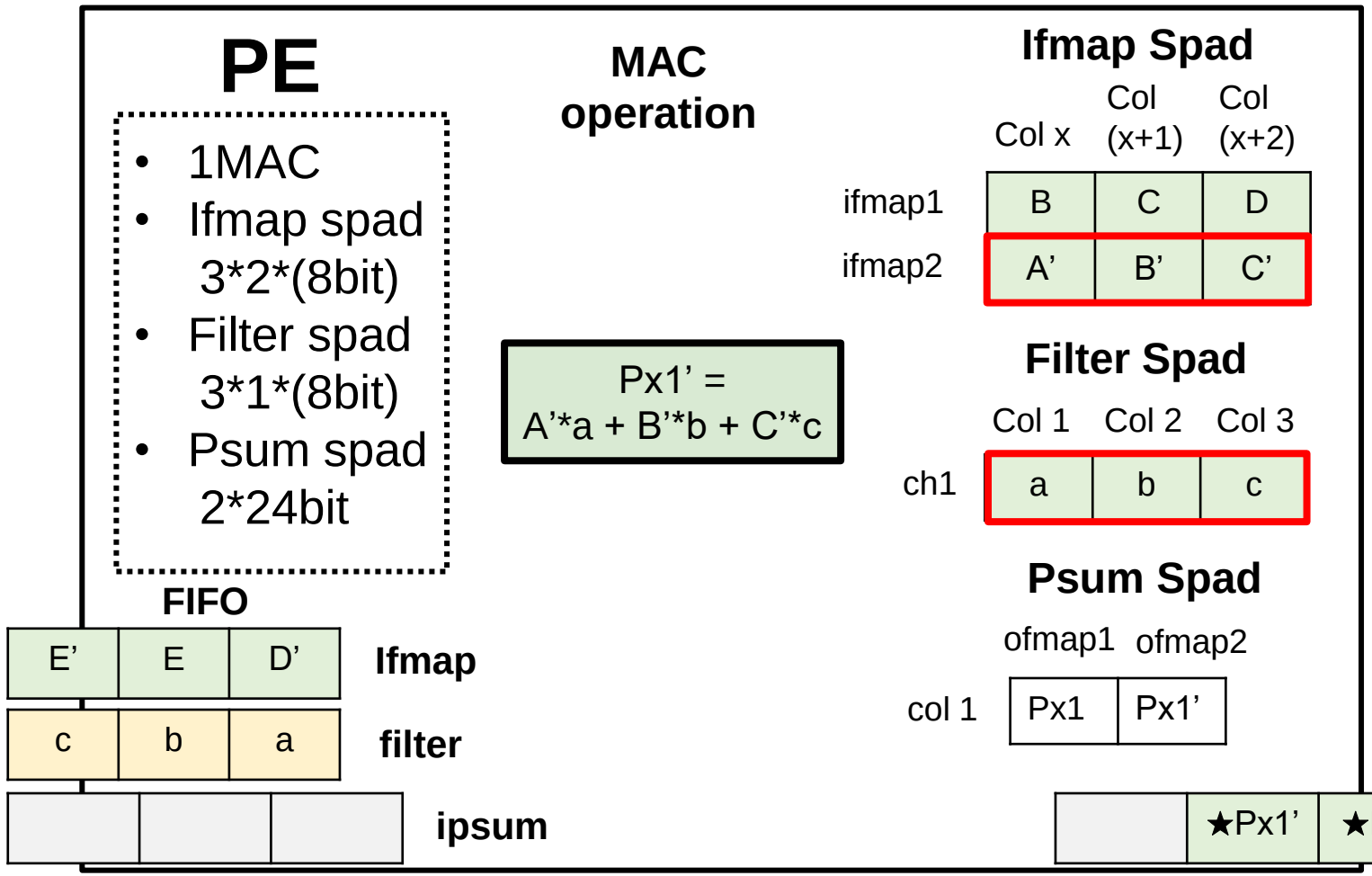
X	Y	Z

Ofmap 2

X'	Y'	Z'

Scenario A Demonstration

- We then computing Px1’ for 3 clks. Simultaneously, we get ifmap D from FIFO and renew our ifmap spad.



Channel 1

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

Channel 1

Kernel 1

a	b	c
d	e	f
g	h	i

Kernel 2

a’	b’	c’
d’	e’	f’
g’	h’	i’

Channel 1

Ofmap 1

X	Y	Z

Ofmap 2

X’	Y’	Z'

Scenario A Demonstration

- After PE1,1 gets the psums (**★Px2 + ★Px3**)、(**★Px2’ + ★Px3’**) from PE2,1 , PE1,1 sends (**★Px1 + ★Px2 + ★Px3**)、(**★Px1’ + ★Px2’ + ★Px3’**) to the Global Buffer.
- In this step, we’ve completed a processing pass.
- **★Px1 = A*a + B*b + C*c** **(channel 1)**
- **★Px1’ = A’*a + B’*b + C’*c** **(channel 1)**
- **★Px2 = F*d + G*e + H*f** **(channel 1)**
- **★Px2’ = F’*d + G’*e + H’*f** **(channel 1)**
- **★Px3 = K*g + L*h + M*i** **(channel 1)**
- **★Px3’ = K’*g + L’*h + M’*i** **(channel 1)**

Global Buffer

(★Px1 + ★Px2 + ★Px3)
(★Px1’ + ★Px2’ + ★Px3’)

psum row 1

★Px1、★Px1’ → $PE_{1,1}$

★Px2、★Px2’ → $PE_{2,1}$

★Px3 、★Px3’ → $PE_{3,1}$

Channel 1

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

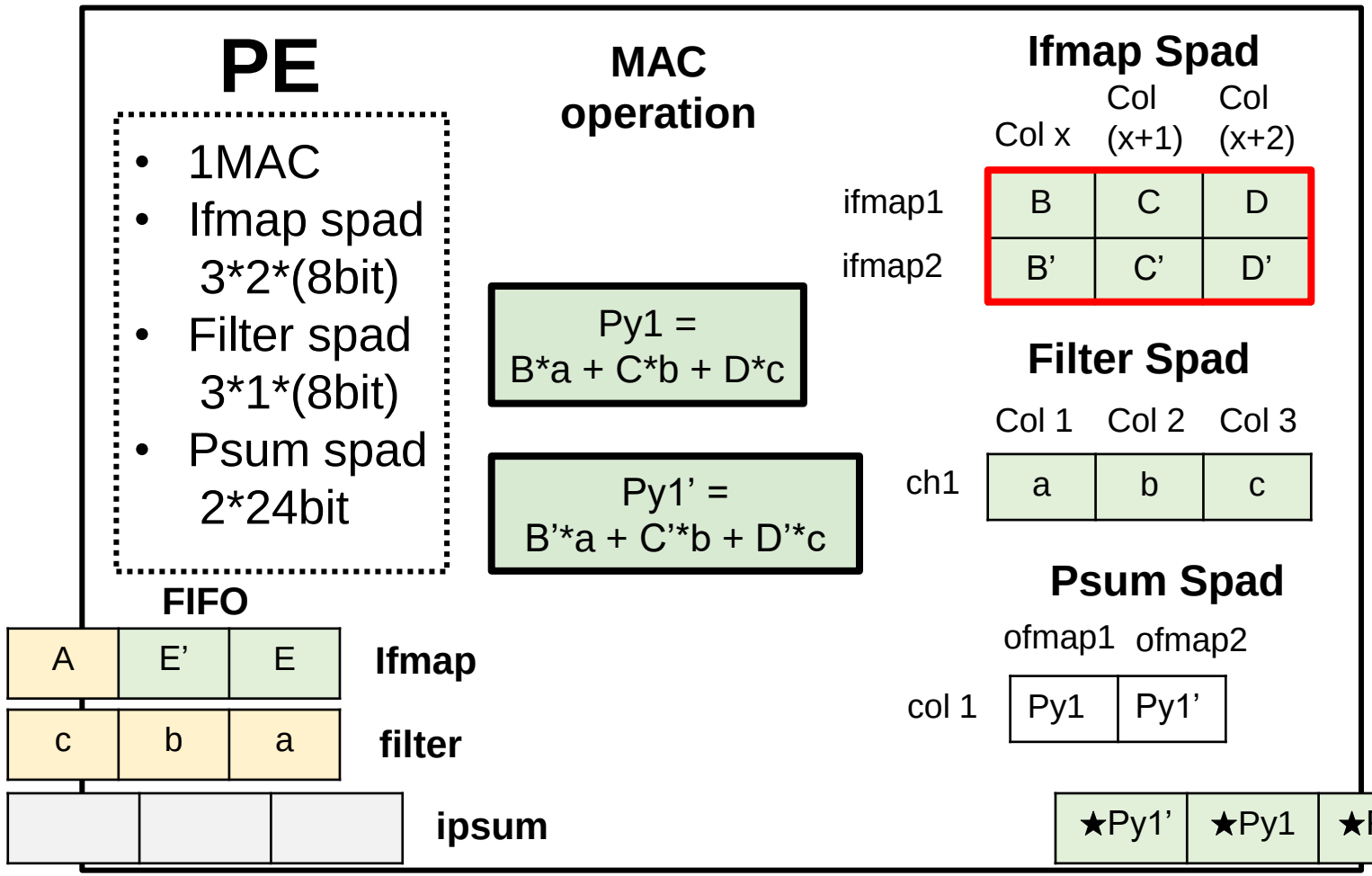
A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

Filter

a	b	c
d	e	f
g	h	i

Scenario A Demonstration

- PE1,1 renews its ifmap spad and receives 2 ifmaps D D’ from Buffer.



Channel 1

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

Channel 1

Kernel 1

a	b	c
d	e	f
g	h	i

Kernel 2

a’	b’	c’
d’	e’	f’
g’	h’	i’

Channel 1

Ofmap 1

X	Y	Z

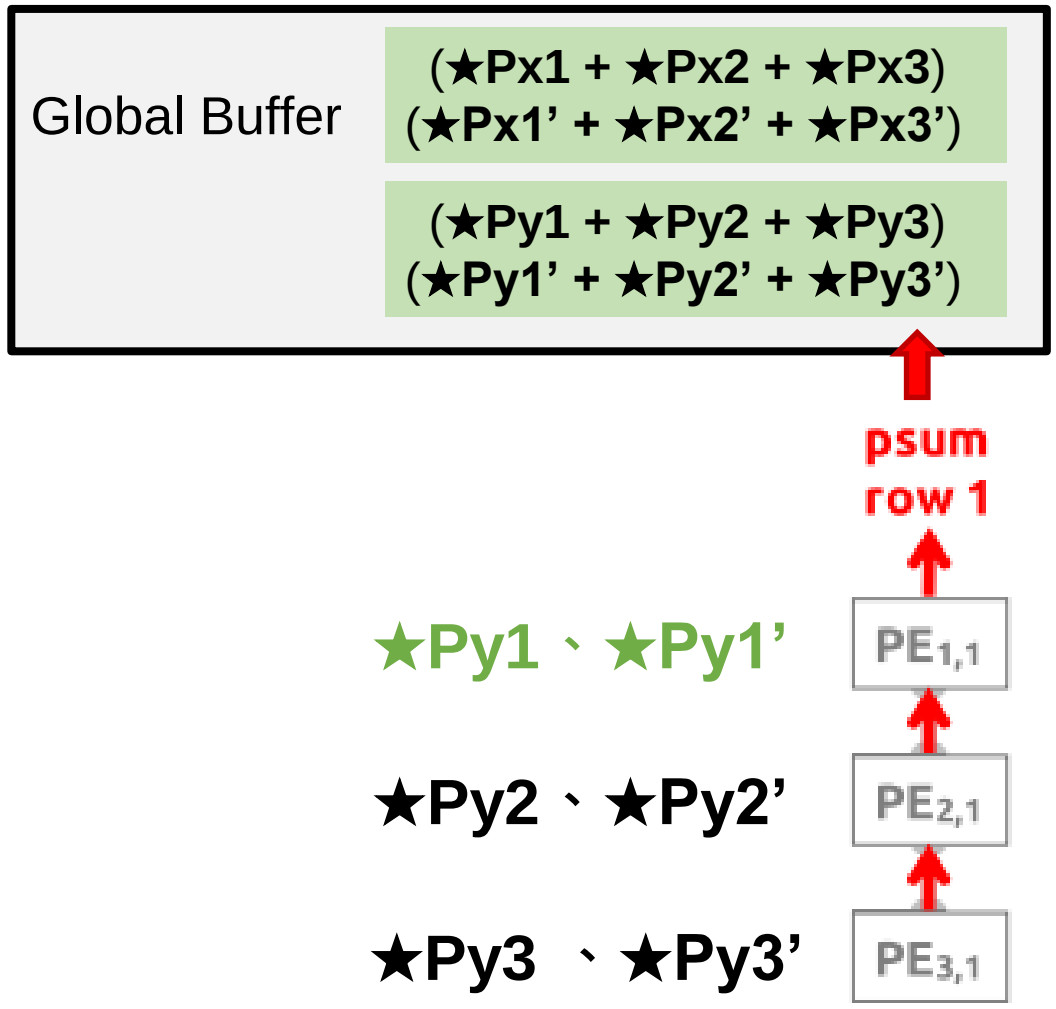
Ofmap 2

X’	Y’	Z’

Scenario A Demonstration

- After PE1,1 gets the psums (**★Py2 + ★Py3**)、(**★Py2’ + ★Py3’**) from PE2,1 , PE1,1 sends (**★Py1 + ★Py2 + ★Py3**)、(**★Py1’ + ★Py2’ + ★Py3’**) to the Global Buffer.

- **★Py1 = B*a + C*b + D*c** (channel 1)
- **★Py1’ = B’*a + C’*b + D’*c** (channel 1)
- **★Py2 = G*d + H*e + I*f** (channel 1)
- **★Py2’ = G’*d + H’*e + I’*f** (channel 1)
- **★Py3 = L*g + M*h + N*i** (channel 1)
- **★Py3’ = L’*g + M’*h + N’*i** (channel 1)



Channel 1

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

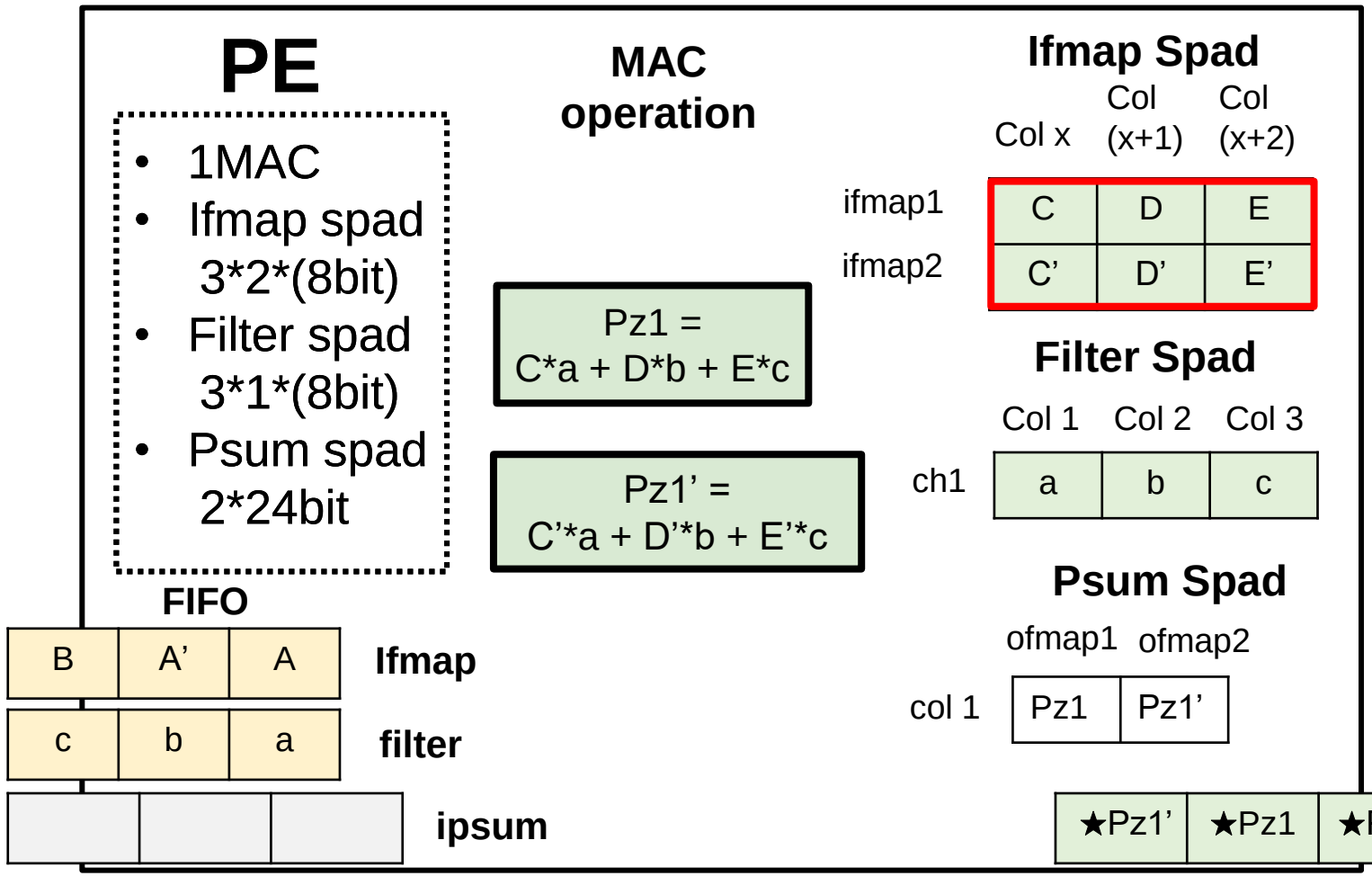
A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

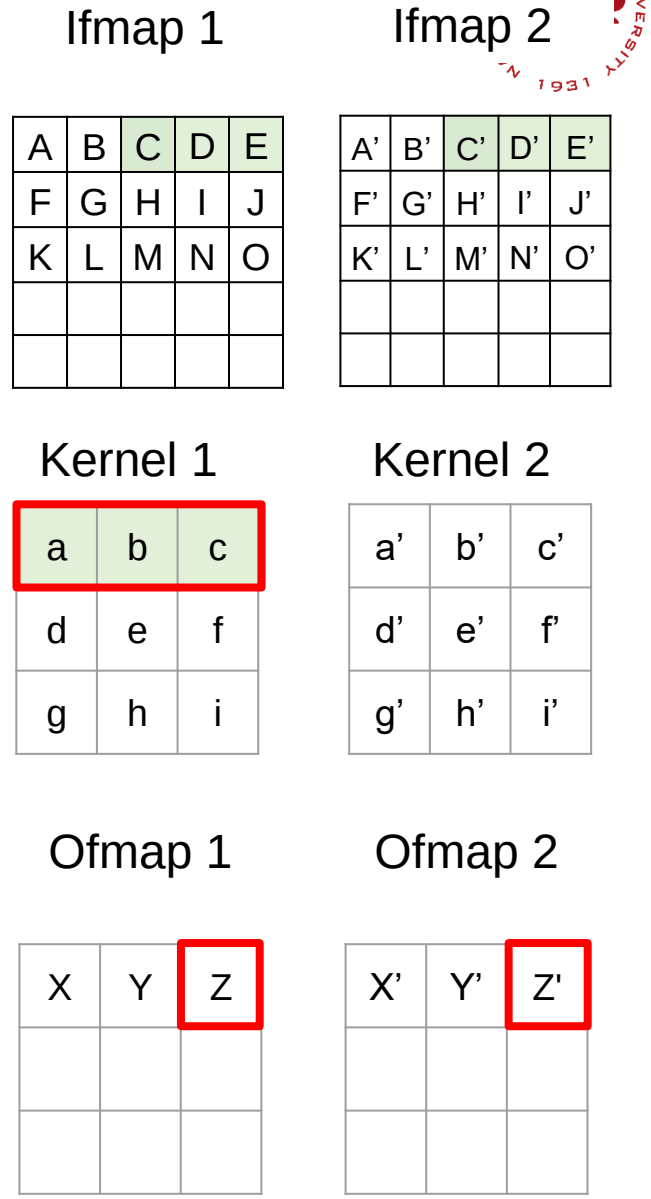
Filter

a	b	c
d	e	f
g	h	i

Scenario A Demonstration

- PE1,1 repeats the above steps and finishes computing next psums.





Scenario A Demonstration

- We repeat the above steps until the last three columns of ifmap1,2 do convolution with the filter.

- **★Pz1 = C*a + D*b + E*c** (channel 1)
- **★Pz1’ = C’*a + D’*b + E’*c** (channel 1)
- **★Pz2 = H*d + I*e + J*f** (channel 1)
- **★Pz2’ = H’*d + I’*e + J’*f** (channel 1)
- **★Pz3 = M*g + N*h + O*i** (channel 1)
- **★Pz3’ = M’*g + N’*h + O’*i** (channel 1)

Channel 1

Ifmap 1

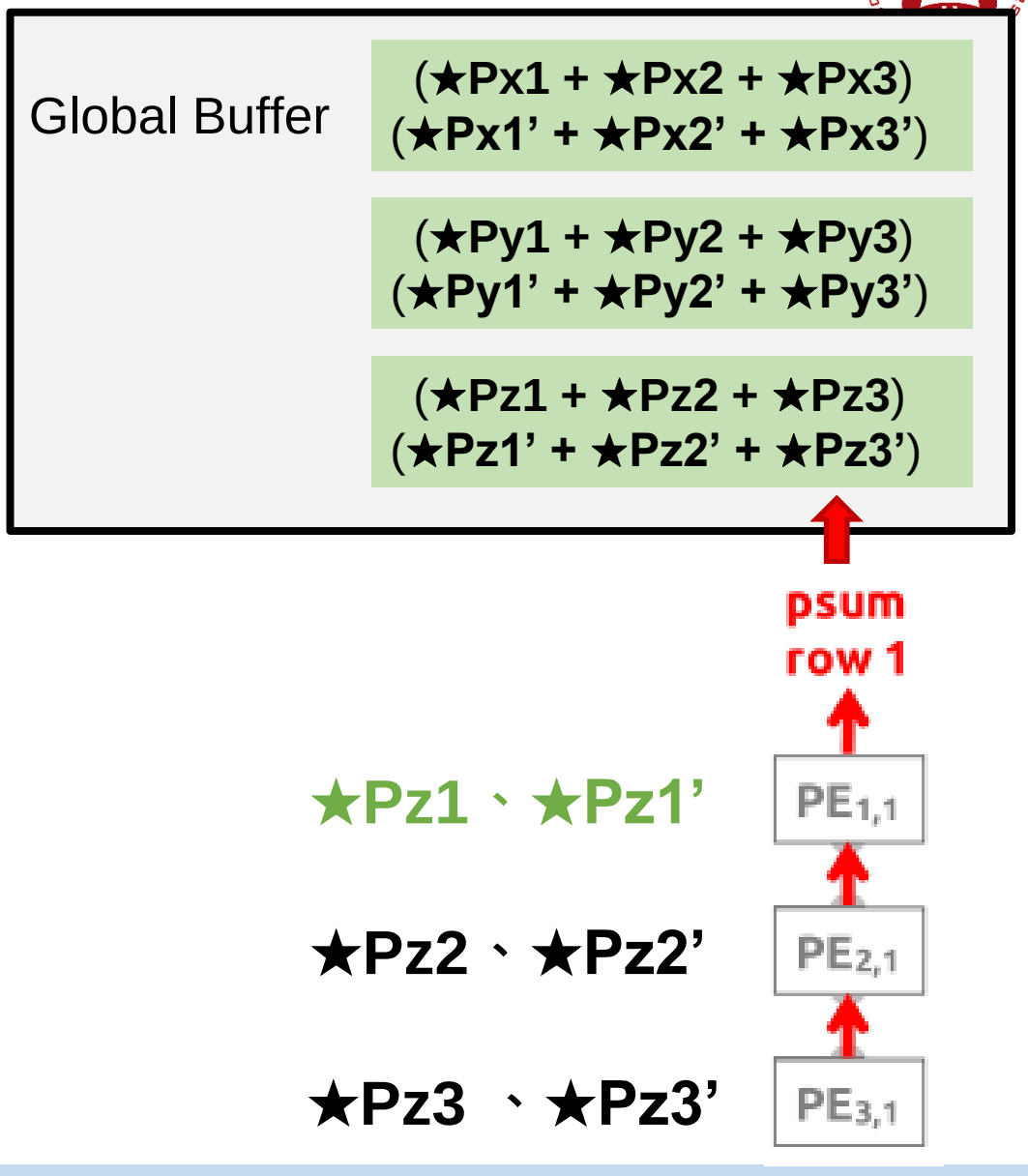
A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

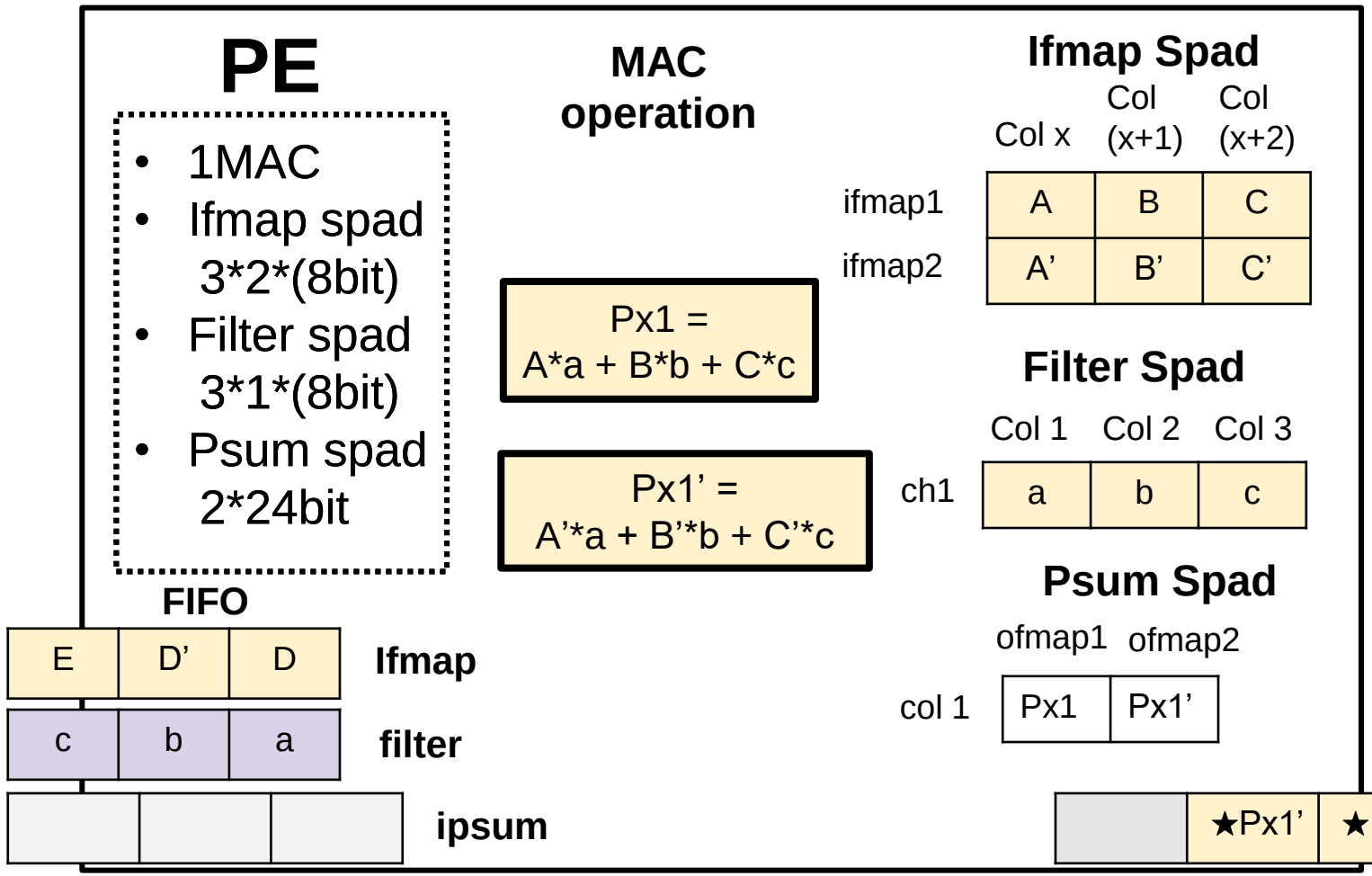
Filter

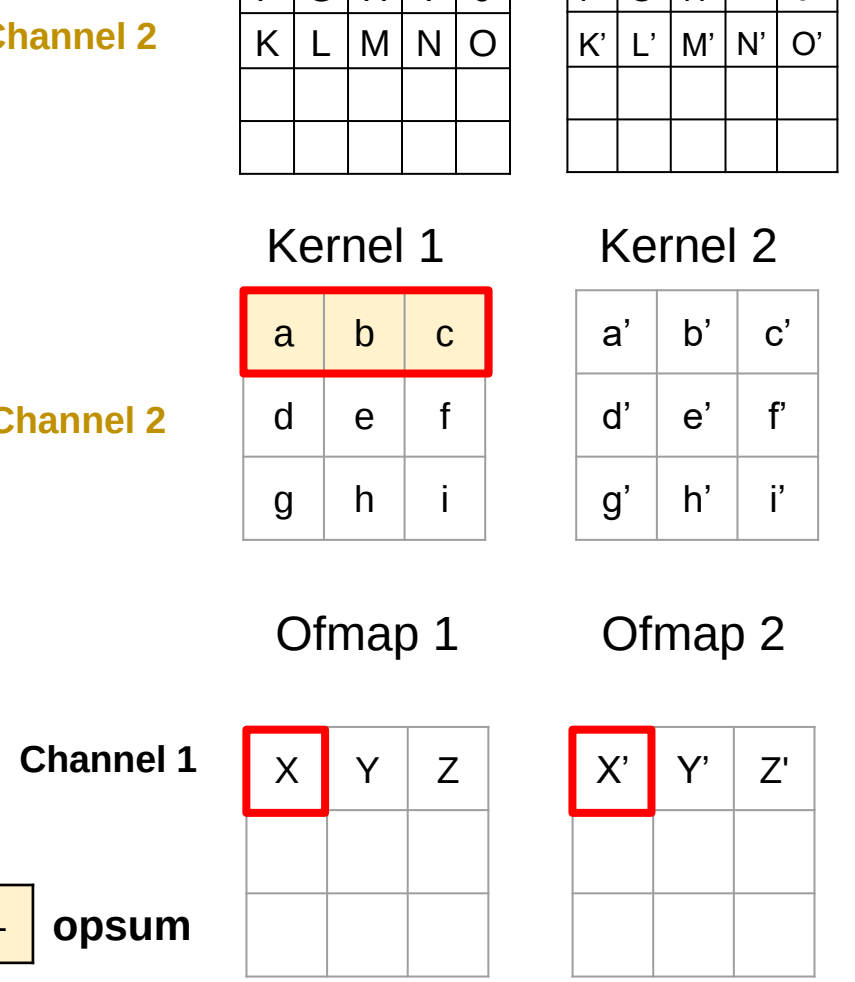
a	b	c
d	e	f
g	h	i



Scenario A Demonstration

- Since channel 1 filters have finished computing with ifmap 1,2 , we then compute channel 2 psums.

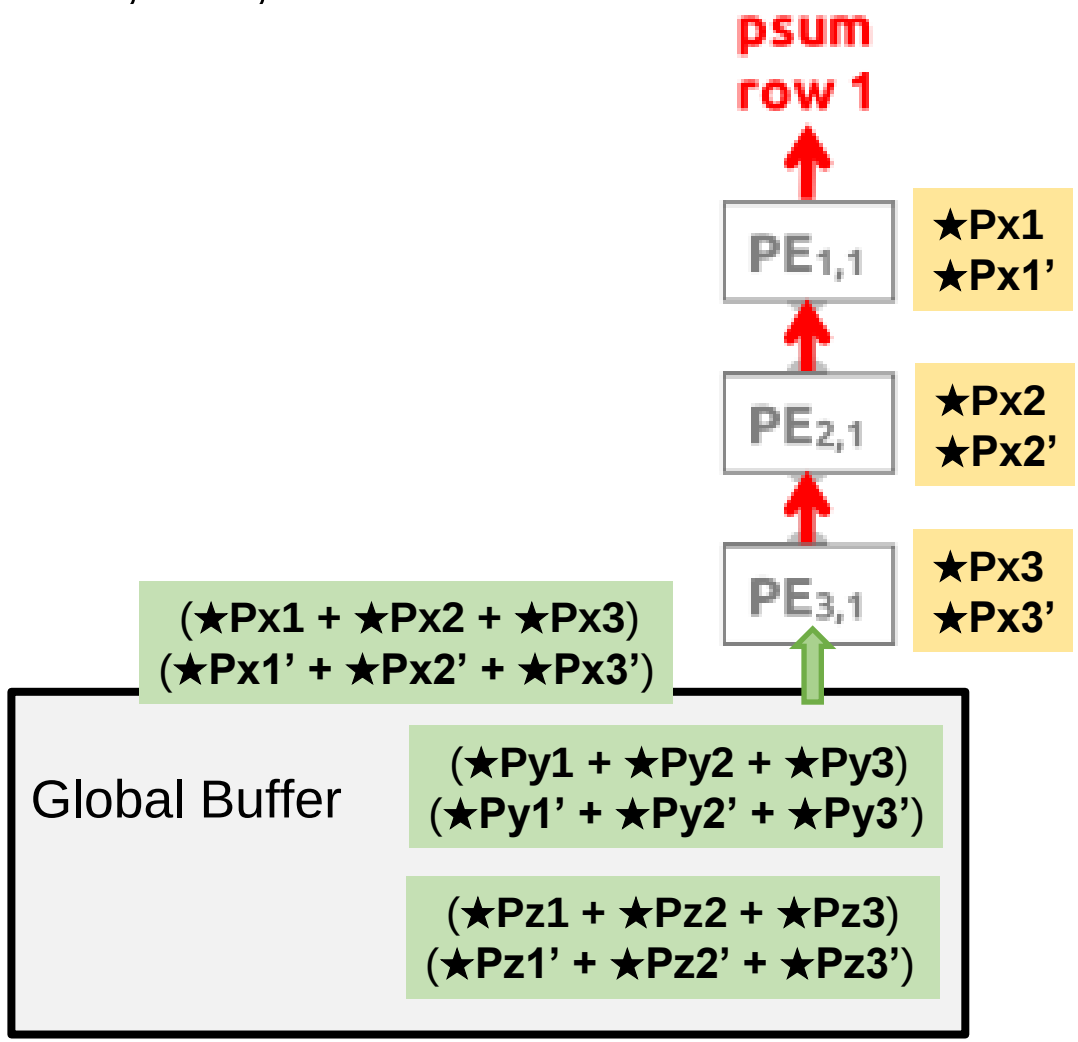




Scenario A Demonstration

- While PE1,1 PE2,1 PE3,1 are computing channel2 psums, PE3,1 receives channel1 psums from Global Buffer.

- **★Px1 = A*a + B*b + C*c (channel 1)**
- **★Px1’ = A’*a + B’*b + C’*c (channel 1)**
- **★Px2 = F*d + G*e + H*f (channel 1)**
- **★Px2’ = F’*d + G’*e + H’*f (channel 1)**
- **★Px3 = K*g + L*h + M*i (channel 1)**
- **★Px3’ = K’*g + L’*h + M’*i (channel 1)**



Channel 2

Ifmap 1

A	B	C	D	E
F	G	H	I	J
K	L	M	N	O

Ifmap 2

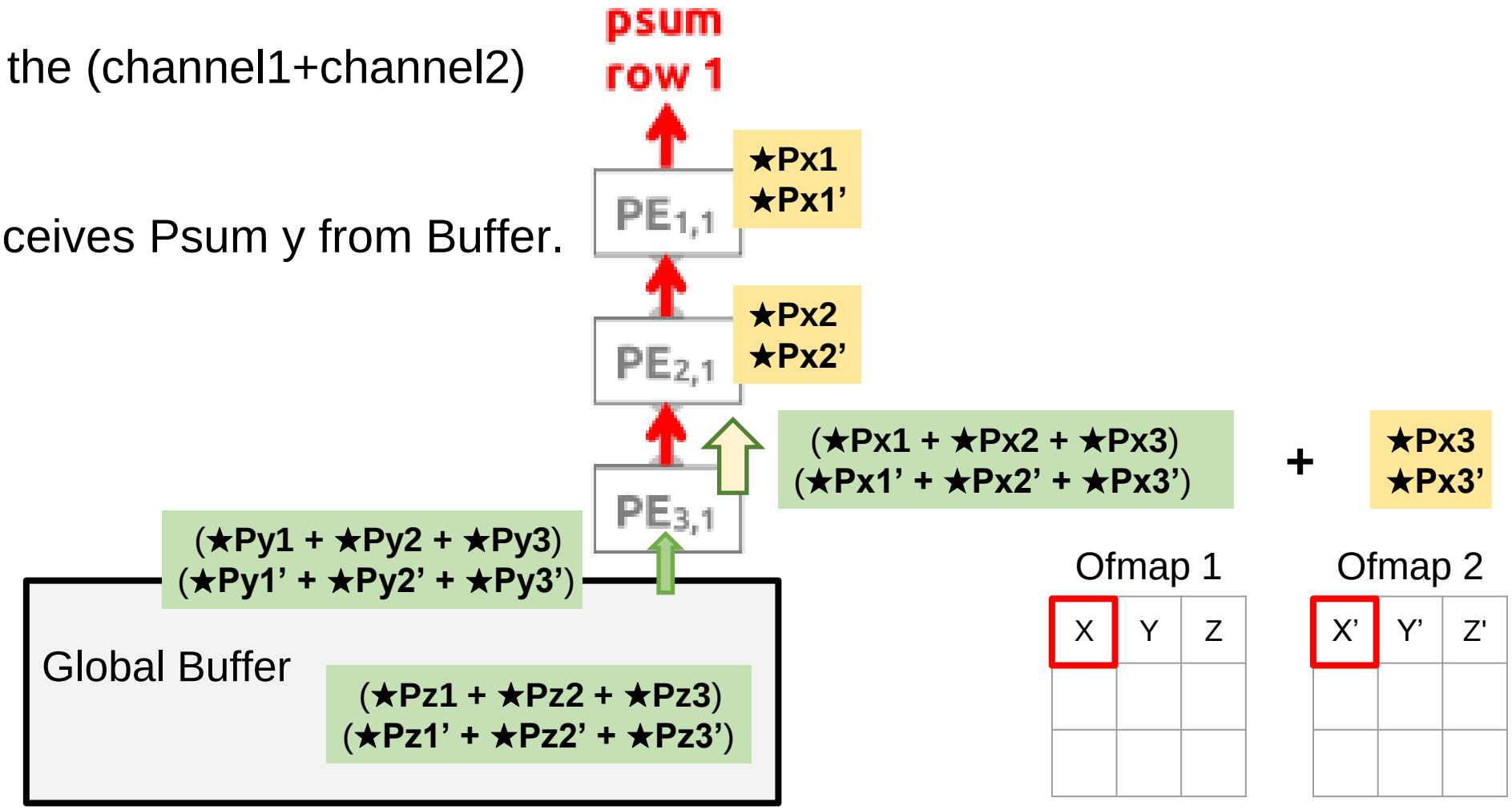
A’	B’	C’	D’	E’
F’	G’	H’	I’	J’
K’	L’	M’	N’	O’

Filter

a	b	c
d	e	f
g	h	i

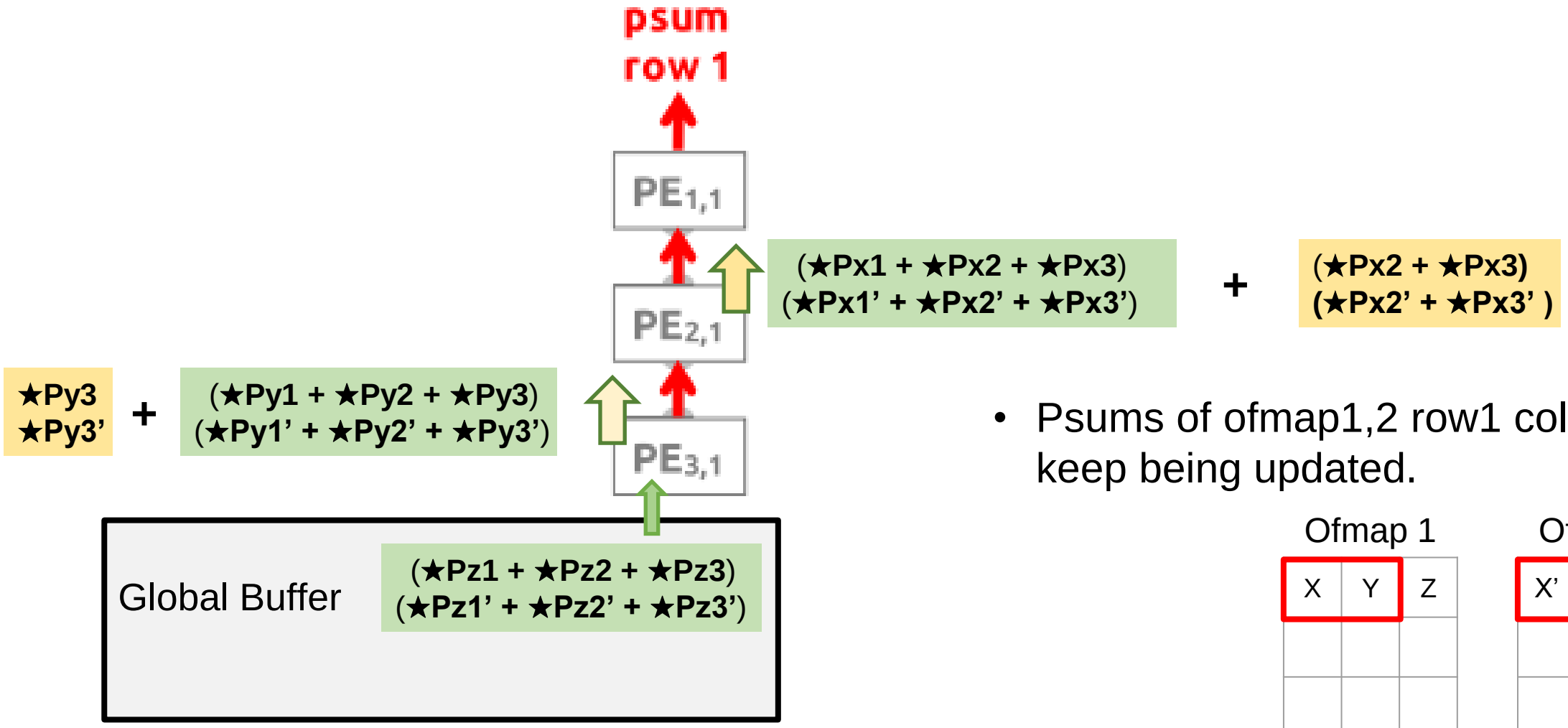
Scenario A Demonstration

- After PE3,1 finishes computing channel2 psums, it adds channel1 psums and channel2 psums together.
- Then, PE3,1 sends the (channel1+channel2) psums to PE2,1.
- Moreover, PE3,1 receives Psum y from Buffer.



Scenario A Demonstration

- PE1,1, PE2,1 and PE3,1 continue to compute channel2 psums and update psums.



- Psums of ofmap1,2 row1 column1,2 keep being updated.

Ofmap 1

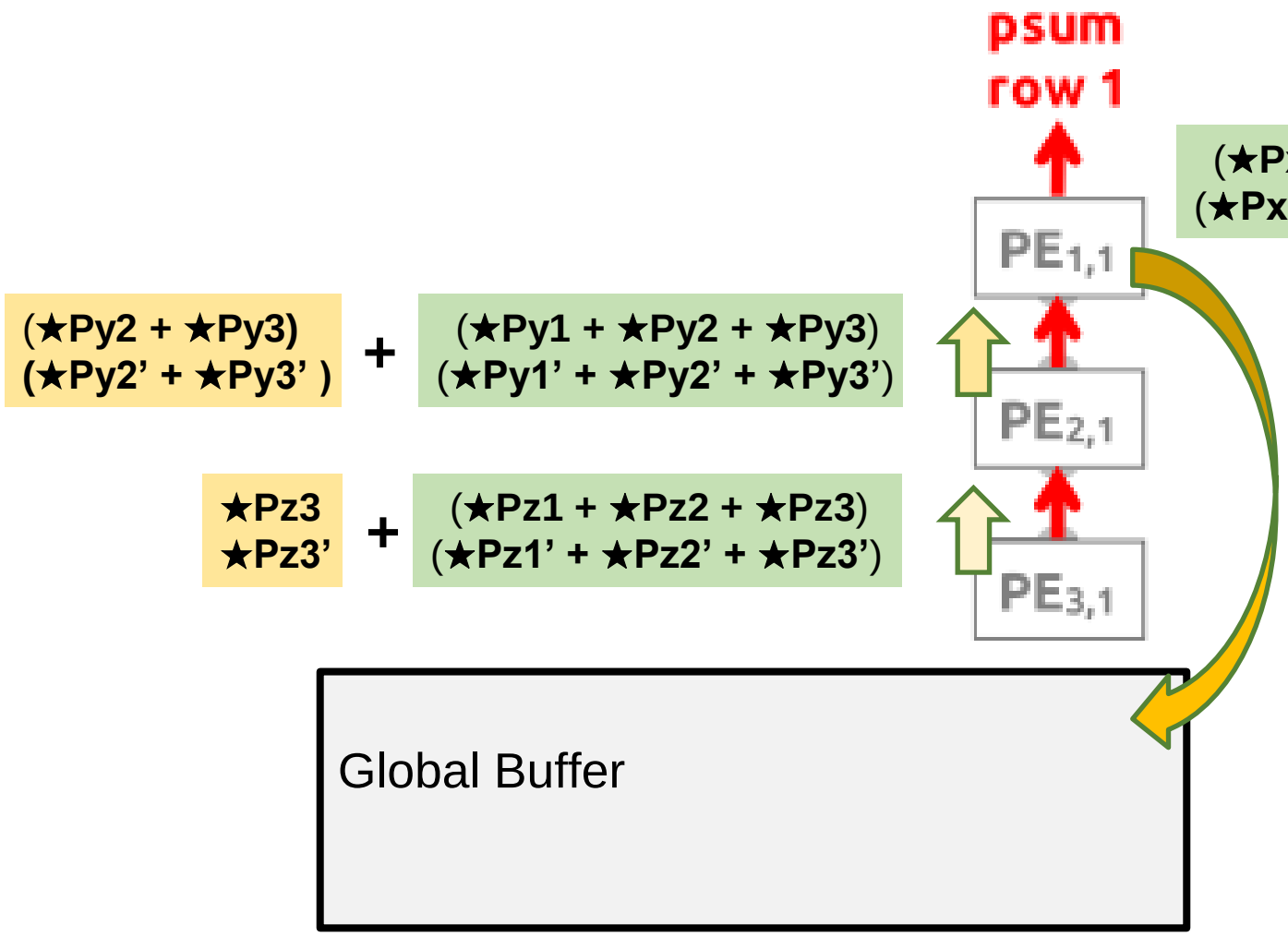
X	Y	Z

Ofmap 2

X'	Y'	Z'

Scenario A Demonstration

- PE1,1, PE2,1 and PE3,1 continue to compute channel2 psums and update psums.



- Psums of ofmap1,2 row1 column1,2,3 keep being updated.

Ofmap 1

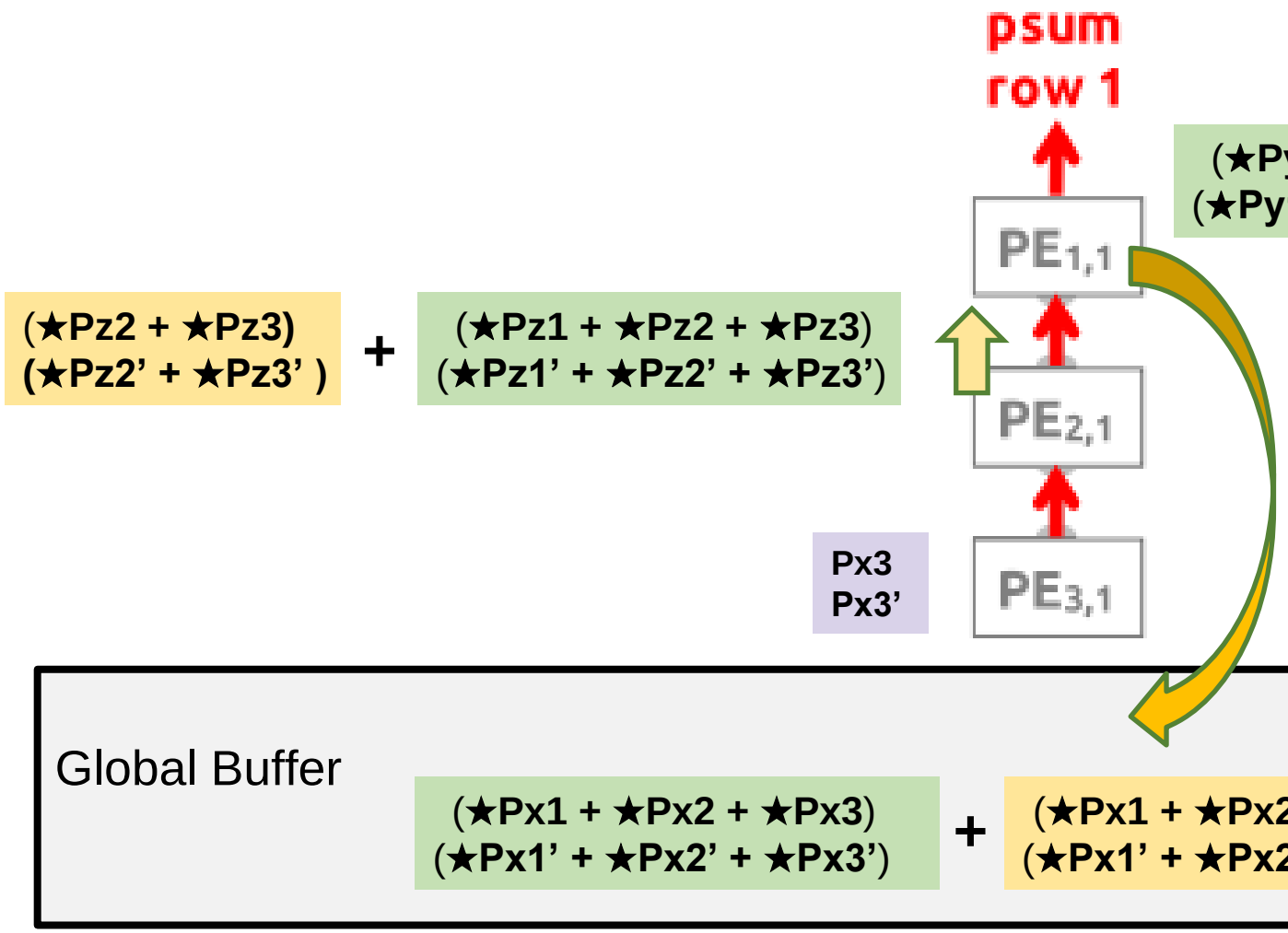
X	Y	Z

Ofmap 2

X'	Y'	Z'

Scenario A Demonstration

- PE1,1 and PE2,1 continue to compute channel2 psums and update psums.
- At the same time, PE3,1 starts to compute channel3 psum.



- Psums of ofmap1,2 row1 column1,2,3 keep being updated.

Ofmap 1

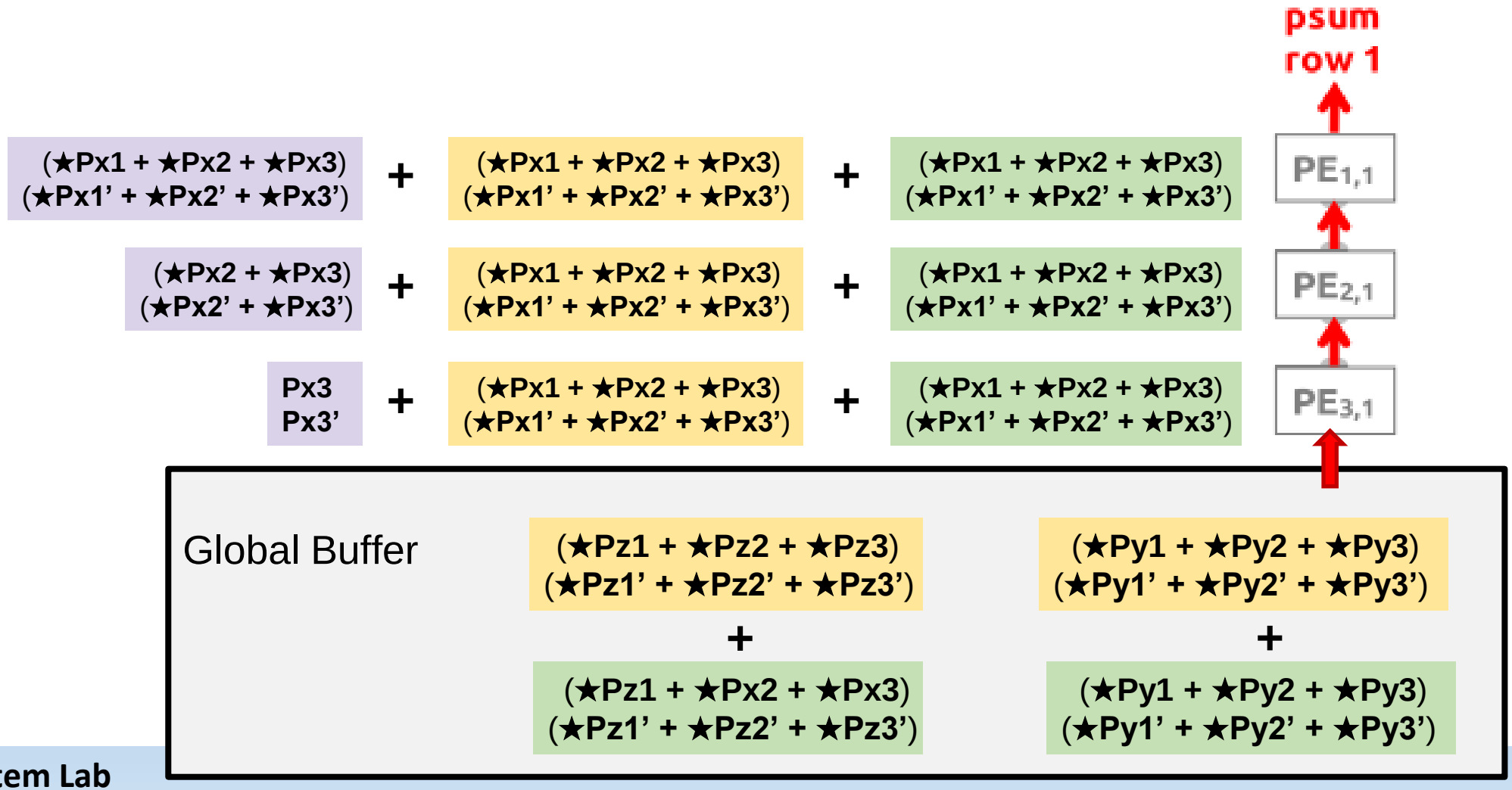
X	Y	Z

Ofmap 2

X'	Y'	Z'

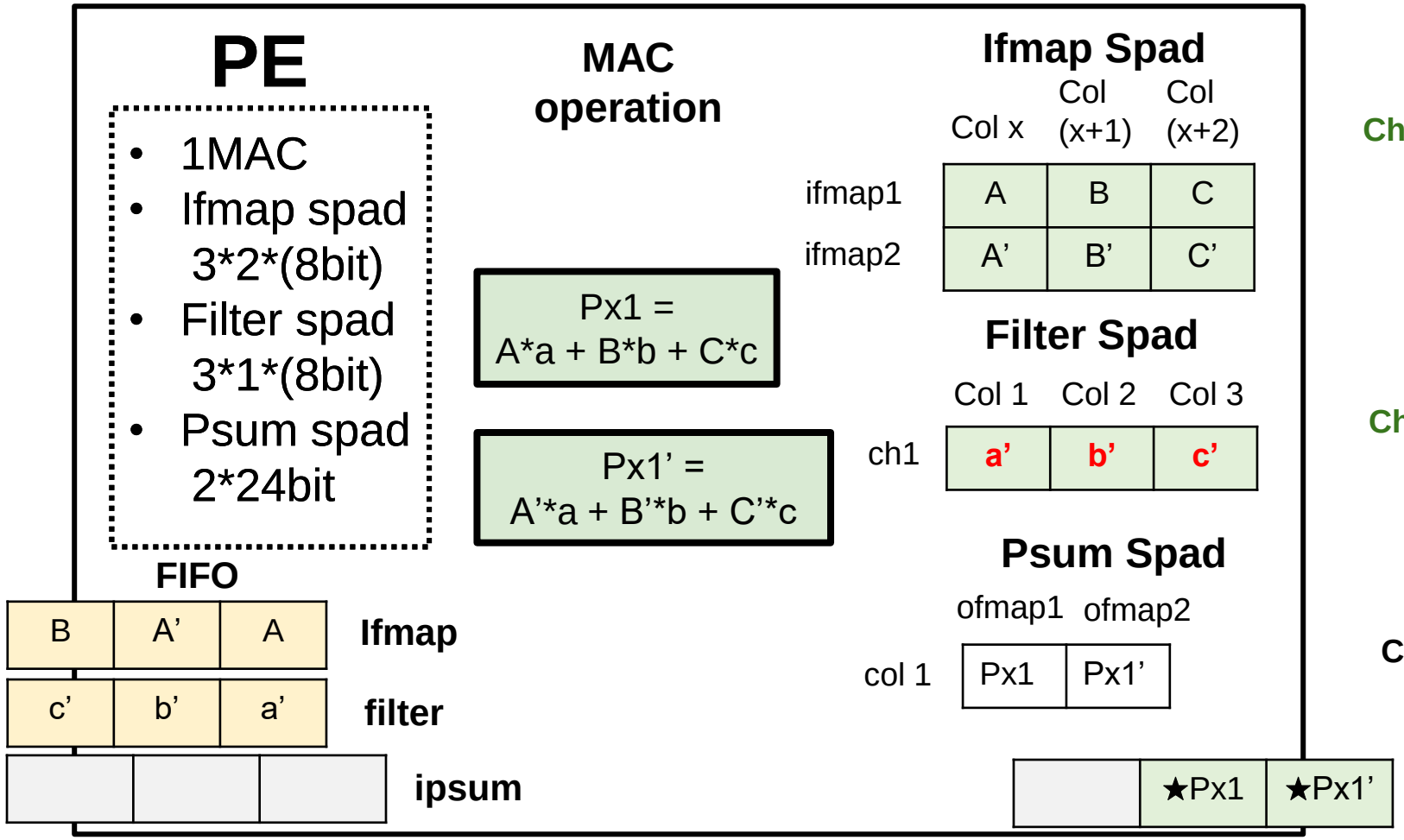
Scenario A Demonstration

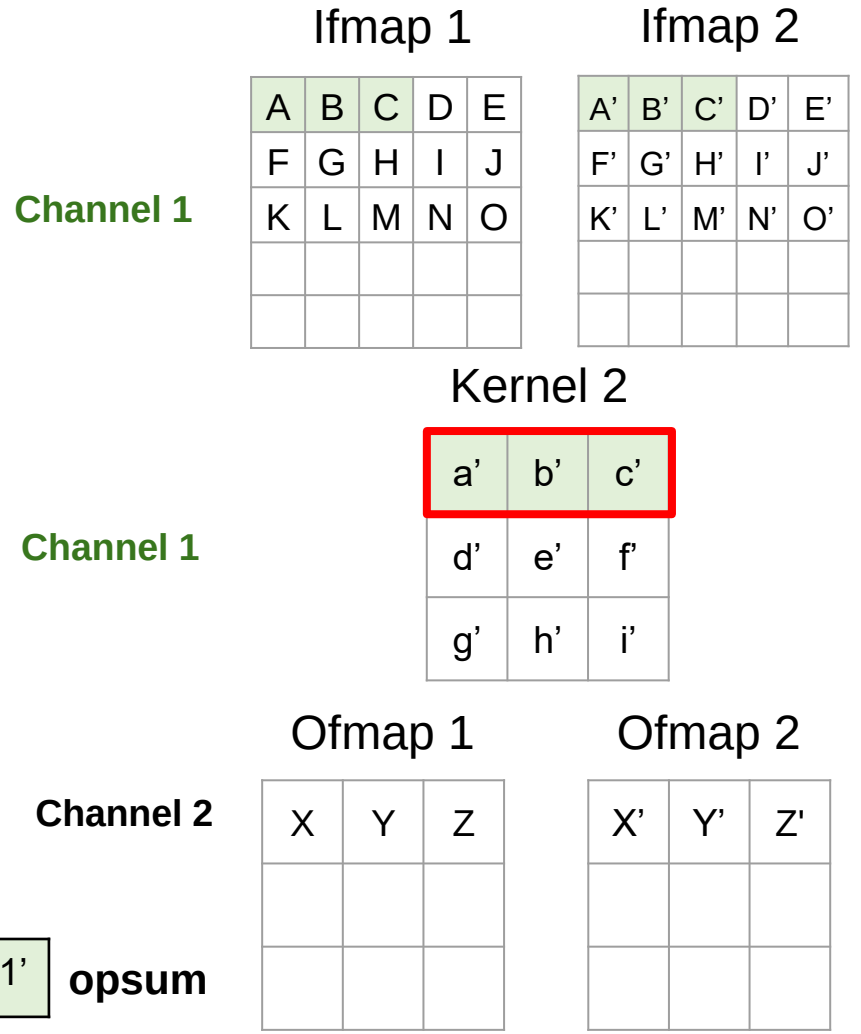
- We repeat the above steps for Pz.
- At last, PE 1,1 adds channel3 psums with (channel2 psums + channel1 psums).



Scenario A Demonstration

- The above steps demonstrates how to complete doing convolution with 1 row of kernels and accumulating 3 channels. If we want to do convolution with 16 kernels, we can **repeat page 41-56 and replace with kernel 2 – 16.**

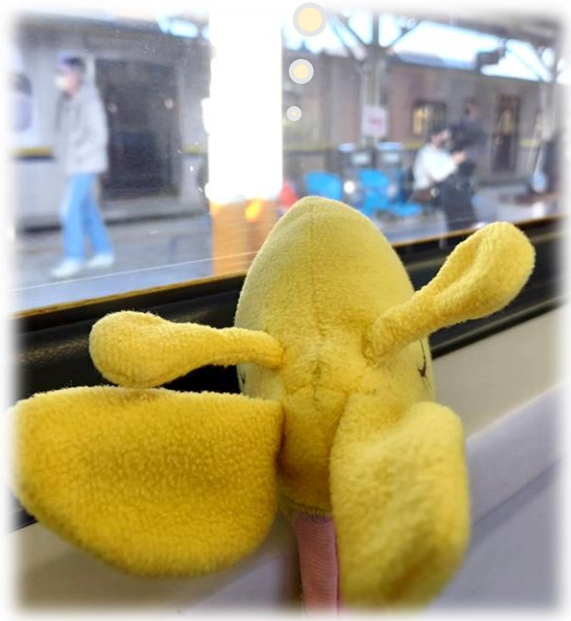




Outline

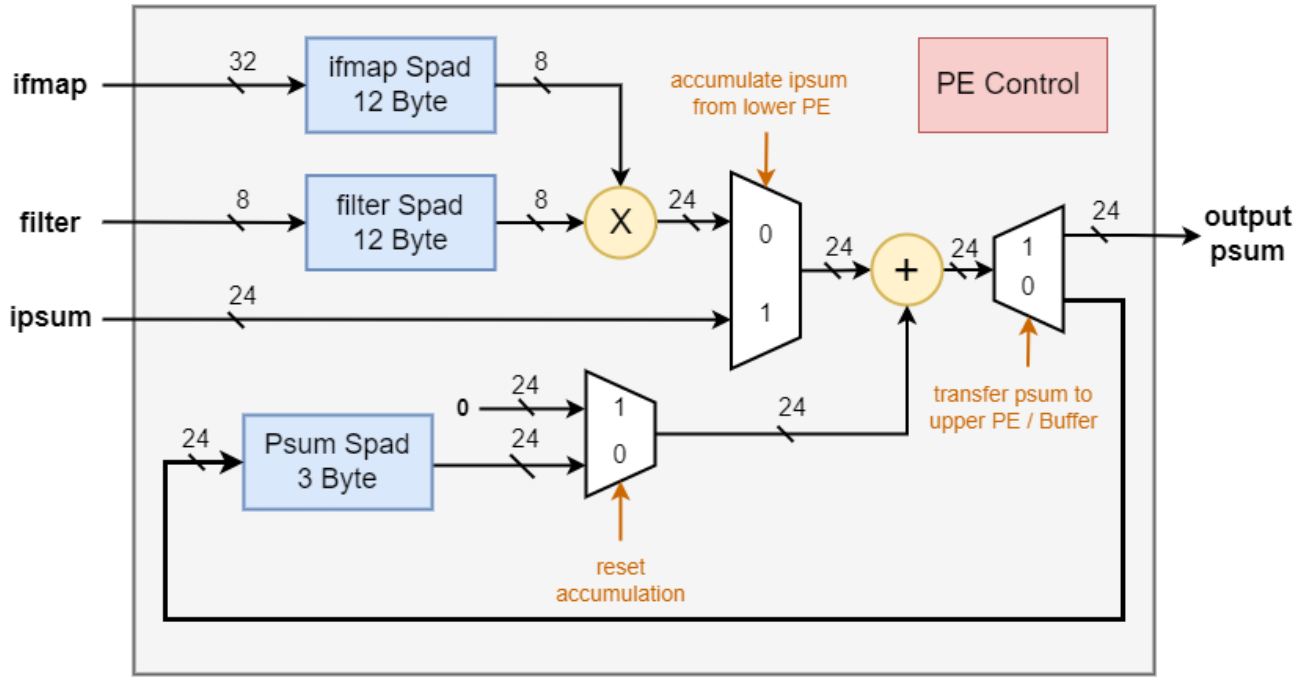
- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- **Chapter6 Homework**
- Chapter7 Bonus
- Chapter8 Supplementary

Giraffe Lulu



Homework Requirement

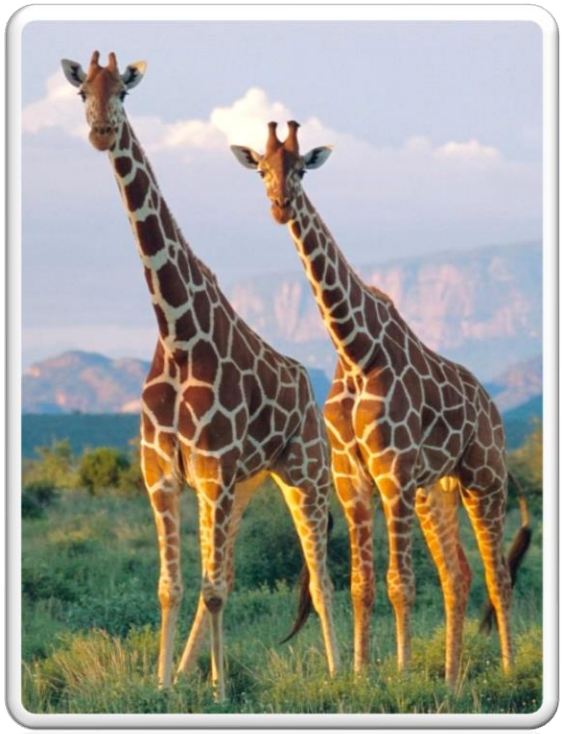
- You need to implement a PE by **Scenario C** with a total spad less than 50 byte and 1 MAC.
- You have two choice to implement your PE
  - Follow the architecture in the right figure
  - Design your own PE structure



PE Design Regulation

- 1MAC
- Total spad space $\leq$ 50 Byte
- 32 ifmap
- 8 filter
- 24 ipsum
- 24 opsum

Testbench



TB

🦄 Simulate layer information given by customize instructions.

🐘 Simulate Global Buffer distributes filter, ifmap and ipsum to PE.

🐏 Simulate psums transfer between upper, lower PEs and Buffer

🐏 Check if opsums computation are correct.

🐏 Check if adder can process overflow.

clk
rst
set_info
layer_info
filter
filter_enable
filter_ready
ifmap
ifmap_enable
ifmap_ready
ipsum
ipsum_enable
ipsum_ready
opsum
opsum_enable
opsum_ready

PE

🦄 Be able to deal with different configurations by Control.

🐘 Be Able to receive filter, ifmap,ipsum and start computing in the correct time.

🐏 Be Able to compute correct opsums and process overflow.

Testbench Signal

Signal	Bit Width	Indication	Note
clk	1	Clock	10ns as a cycle
rst	1	Reset	Turn everything to initial state
set_info	1	Simulate Configuration Scan Chain receives layer information from instructions, and send them to the PE Array	All PEs need to renew their layer information
Ch_size ifmap_column ofmap_column	3 6 6	the number of ifmap filter channels the number of columns in an ifmap row the number of columns in an ofmap row	Given values equal to actual values e.g. 8 bit ifmap ifmap_Quant_size = 4'd8
ifmap_Quant_size filter_Quant_size	4 4	Ifmap bit width Filter bit width	

Testbench Signal

When handshake (enable and ready both are high) happens,
Buffer can transfer the next data.

Signal	Indication	Note
filter filter_enable filter_ready	Weight If weight has been prepared in Global Buffer If PE is ready to receive next weight	1 element at a time
ifmap ifmap_enable ifmap_ready	Input feature map If ifmap has been prepared in Global Buffer If PE is ready to receive next ifmap	[7:0] = ifmap channel x [15:8] = ifmap channel (x+1) [23:16] = ifmap channel (x+2) [31:24] = ifmap channel (x+3)
ipsum ipsum_enable ipsum_ready	Input partial sum from the lower PE or Buffer If ipsum has been prepared in the lower PE or Buffer If the PE is ready to receive next ipsum	Simulate psums transfer among upper, lower PEs and Buffer.
opsum opsum_enable opsum_ready	Output partial sum to the upper PE or Buffer If opsum has been prepared in the upper PE or Buffer If the PE is ready to receive next opsum	

Testbench's Dataflow

- After reset, you need to read the following layer information and store them in your PE.
- Since the model we apply its stride is always 1 and filter column is always 3, we don't have to provide these info to the PE Array.
- TB sends all information in 1 clock.

set_info == 1

Ch_size
ifmap_column
ofmap_column
ifmap_Quant_size
filter_Quant_size

- **After you've stored layer info, you can start to read data of ifmap, filter and ipsum.**

How to let testbench start sending data | Take ifmap for example

Ifmap_ready — When **your PE** is ready to receive data

Ifmap_enable — When **testbench** is ready to send data
To simulate real case, **buffer is not ready to send data all the time**.

When both two signals are high, we call it **"handshake"**.
When handshake, the testbench will send data.

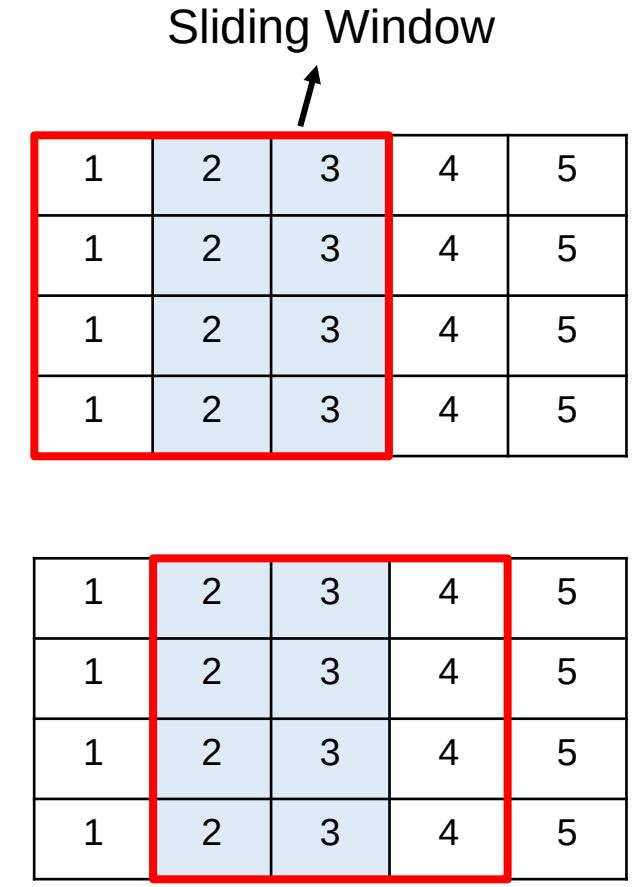
Testbench's Dataflow

Ifmap

Ifmap_enable and Ifmap_ready are high

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

- The numbers in the table represent the order in which the data is given.
- The bandwidth of ifmap is 32bit → give 4 cells at a time
- (different channels in the same column)



Notice

Overlapping values will not be given again.

You must find a way to keep them.

Testbench's Dataflow

Filter

filter_enable and filter_ready are high

- The numbers in the table represent the order in which the data is given.
- The bandwidth of filter is 8bit → give 1 cell at a time
- Each element will only be given once.

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

...

Kernel 16

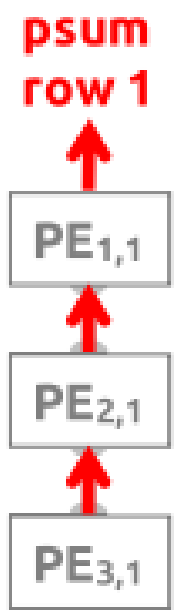
	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Opsum Order **By Scenario C**

Notice

Remember to add ipsum
(the opsum of the lower PE)
Psum stores in Buffer is PE3,1's ipsum
PE3,1's opsum is PE2,1's ipsum
PE2,1's opsum is PE1,1's ipsum



Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Notice

Remember to add ipsum
(the opsum of the lower PE)

Opsum Order **By Scenario C**

Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Notice

Remember to add ipsum
(the opsum of the lower PE)

Opsum Order **By Scenario C**

Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Notice

Remember to add ipsum
(the opsum of the lower PE)

Opsum Order **By Scenario C**

Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Buffer distributes ifmap from scratch again.
Since we have to do convolution with 16 kernels, we need to send out ifmap for 16 times in total.

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Notice

Remember to add ipsum
(the opsum of the lower PE)

Opsum Order **By Scenario C**

Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

Testbench's Dataflow

Notice

Remember to add ipsum
(the opsum of the lower PE)

Opsum Order

By Scenario C

Ifmap

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

Ofmap

	Col 1	Col 2	Col 3
Ch 1	1	2	3
Ch 2	4	5	6
⋮	…	…	…
Ch 16	46	47	48

Last Ofmap !

Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

…

Kernel 16

	Col 1	Col 2	Col 3
Ch 1	181	185	189
Ch 2	182	186	190
Ch 3	183	187	191
Ch 4	184	188	192

TestBench 0 Instruction

- The testbench instruction takes PE1,1 as example.
- In real scenario, each PE has to process different ifmap, filter and psum rows.
- Assume a PE set receives 16 kernels that corresponding ofmap channels are consecutive.
- **Data Distribution Order:**

Ifmap

Distributed for **16 times**

	Col 1	Col 2	Col 3	Col 4	Col 5
Ch 1	1	2	3	4	5
Ch 2	1	2	3	4	5
Ch 3	1	2	3	4	5
Ch 4	1	2	3	4	5

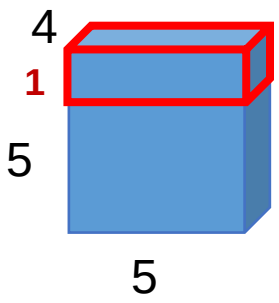
Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

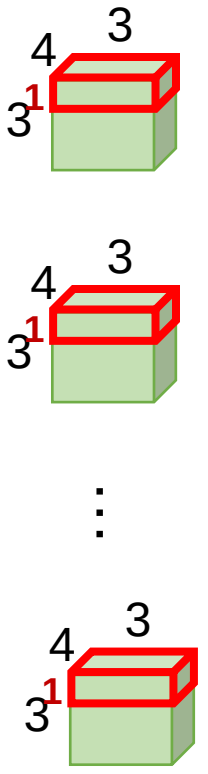
Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

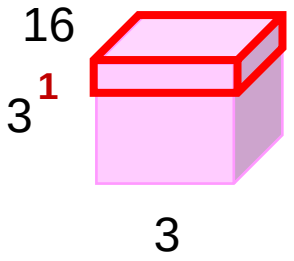
1 ifmap



16 kernels
16 processing pass



1 psum (ofmap)



TestBench 1 Instruction

- Processing passes also can address same set of kernel on different channels.
- Simulate **VGG16-Cifar10** processing passes of **1st layer**
- Ifmap is distributed for 64 times.

Ifmap1

Ifmap1 is distributed for **64 times**.

		Col 1	Col 2	Col 3	Col 4 …
[7:0]	Ch 1	1	2	3	4 …
[15:8]	Ch 2	1	2	3	4 …
[23:16]	Ch 3	1	2	3	4 …
[31:24]	Ch 4	0	0	0	0 …

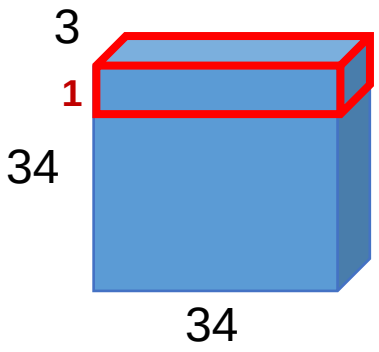
Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	4	7
Ch 2	2	5	8
Ch 3	3	6	9
Ch 4	0	0	0

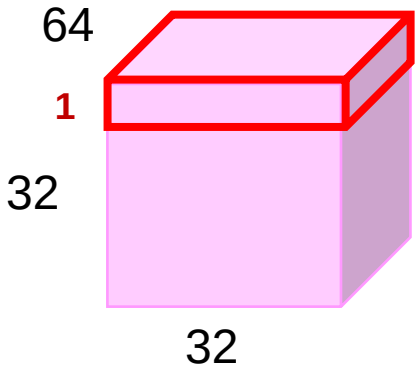
Kernel 2

	Col 1	Col 2	Col 3
Ch 1	10	13	16
Ch 2	11	14	17
Ch 3	12	15	18
Ch 4	0	0	0

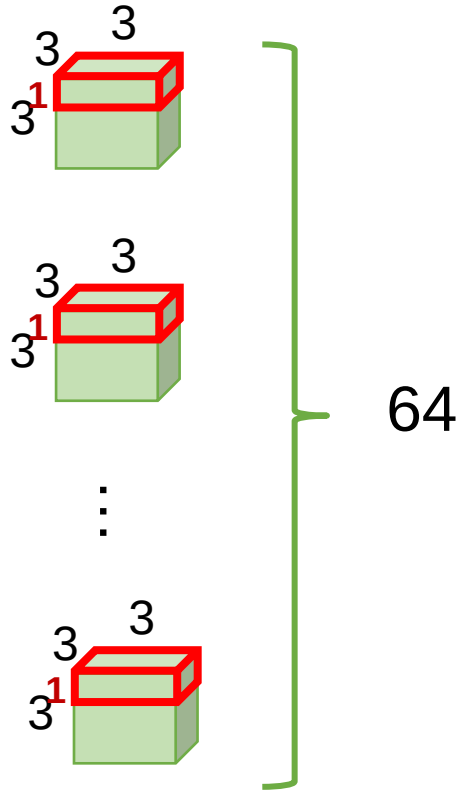
1 ifmap



1 psum (ofmap)



64 kernels
64 processing passes



TestBench 2 Instruction

- Simulate **VGG16-cifar10 2nd layer**. Simulate **zero data** produced by pruned model and ReLU.
- Ifmap1 has to be distributed for 16 times and do convolution with 16 kernels.

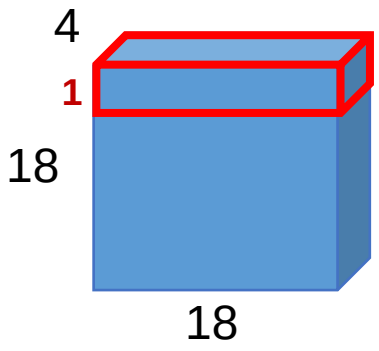
  After which, ifmap2 can start to be distributed.

Ifmap1,2

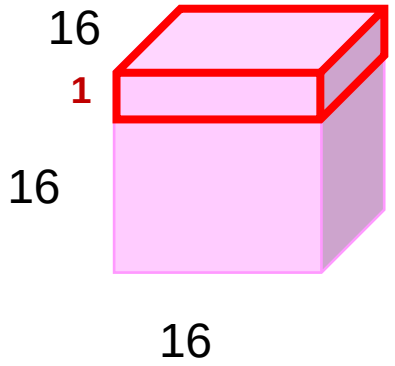
Ifmap1 is distributed for **16 times**, after which ifmap2 does so.

	Col 1	Col 2	Col 3	Col 4	...
Ch 1	1	2	3	4	…
Ch 2	1	2	3	4	…
Ch 3	1	2	3	4	…
Ch 4	1	2	3	4	…

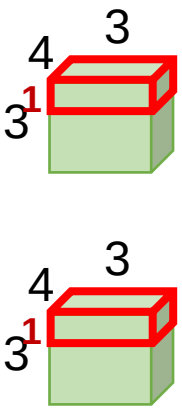
ifmap1



psum (ofmap) 1



16 kernels



- Kernel 1-16 need to be sent for **2 times**.

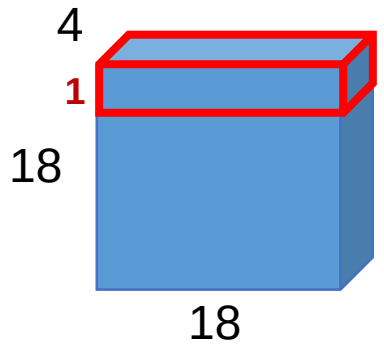
Kernel 1

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

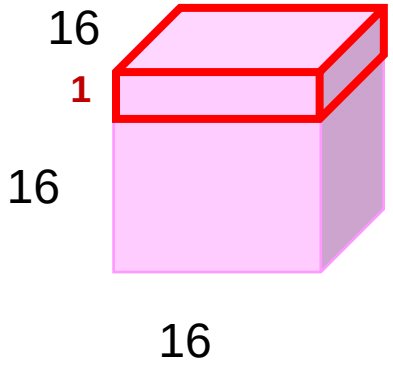
Kernel 2

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

ifmap2



psum (ofmap) 2



Simulation Information

- Take tb0 for example

At time 10ns, Buffer resets and prepares for first set of data.

```
*Ifmap Distribution Info* Time:          10, Ifmap Buffer Coordinate [Batch Size][Column][Channel]
*Ifmap Distribution Info* Time:          10, Enable:0, Ifmap Buffer is preparing for[0][0][3,2,1,0]

*Filter Distribution Info* Time:         10, Filter Buffer Coordinate [Kernel][Column][Channel]
*Filter Distribution Info* Time:         10, Enable:0, Filter Buffer is preparing for[0][0][0]

*Input Psum Distribution Info* Time:         10, Ipsum Buffer Coordinate [Batch Size][Channel][Column]
*Input Psum Distribution Info* Time:         10, Enable:0, Ipsum Buffer is preparing for[0][0][0]
```

From info display on the terminal, we get to know the 1st data PE receive will be

Ifmap : **Batch Size** 0 **Column** 0 **Channel** 3,2,1,0

Filter : **Kernel** 0 **Column** 0 **Channel** 0

Ipsum : **Batch Size** 0 **Channel** 0 **Column** 0

Simulation Information

At time 30ns, Buffer sends our first set of filter and ipsum.

Filter[Kernel][Column][Channel] = Filter[0][0][0] = 70

Ipsum[Batch Size][Channel][Column] = Ipsum[0][0][0] = 5936644

Next data

Filter : **Kernel** 0 **Column** 0 **Channel** 1

Ipsum : **Batch Size** 0 **Channel** 0 **Column** 1

```
*Filter* Time:                  30, filter:   70, next filter Buffer[ 0][ 0][ 1]
*Ipsum* Time:                  30, ipsum:   5936644, next ipsum Buffer[ 0][ 0][ 1]
```

At time 70ns, Buffer sends our first set of ifmap.

Ifmap Coordinate [Batch Size][Column][Channel]

Ifmap[0][0][3] = 16, Ifmap[0][0][2] = -105, Ifmap[0][0][1] = -110, Ifmap[0][0][0] = 22

Next data

Ifmap : **Batch Size** 0 **Column** 1 **Channel** 3,2,1,0

```
*Ifmap* Time:                  70, ifmap:   16 -105 -110   22, next ifmap Buffer[ 0][ 1][3,2,1,0]
```

Simulation Information

- Terminal will display last data.

```
*Filter* Time:                    2420, last filter: -128

*Ipsum* Time:                   750, last ipsum:  -8388608

*Ifmap* Time:                   1060, last ifmap:  127  100  -74   36
```

- Check the answers

Correct answer (in yellow)

Batch Size

```
Time:              120, Ofmap  0 channel  0 column  2 :   -1381046 is correct.
```

Wrong answer

```
 Time:              130, Ofmap  0 channel  1 column  0 is  WRONG.
Correct ans    1685887 / Your ans      12345
```

Unknown answer

```
 Time:              160, Ofmap  0 channel  1 column  2 is WRONG.
Correct ans   -7974830 / Your ans Unknown
```

Simulation Information

- If you haven't sent out all the opsums, terminal will display

```
!Error! You only have only        8 sets of opsum
```

- If you send out all the output psums, terminal will show the **total cycles** in yellow for computing all the opsums at the end of simulation whether the answers are correct or not.

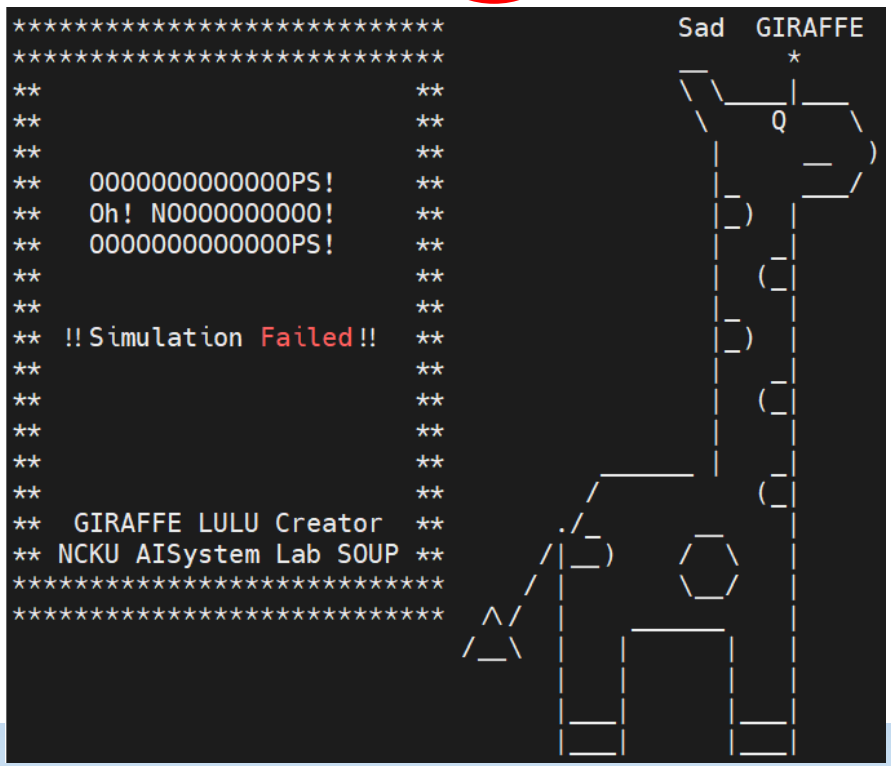
```
          Totally has        44 errors

In                    56 cycles, You have sent out all opsums.
```

Write down in Report Q2

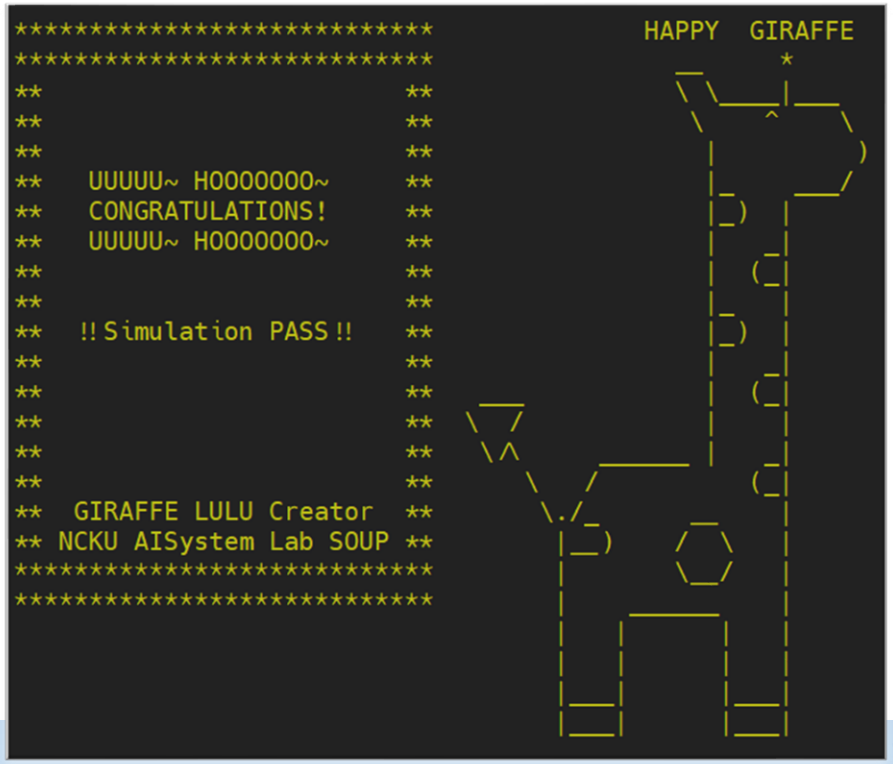
- **Fail**

Sad Giraffe



PASS tb0

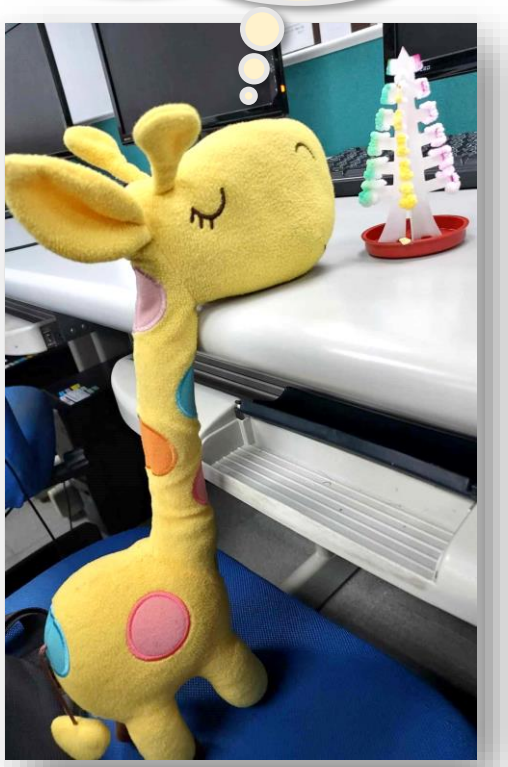
Golden HAPPY Giraffe



Simulation Information

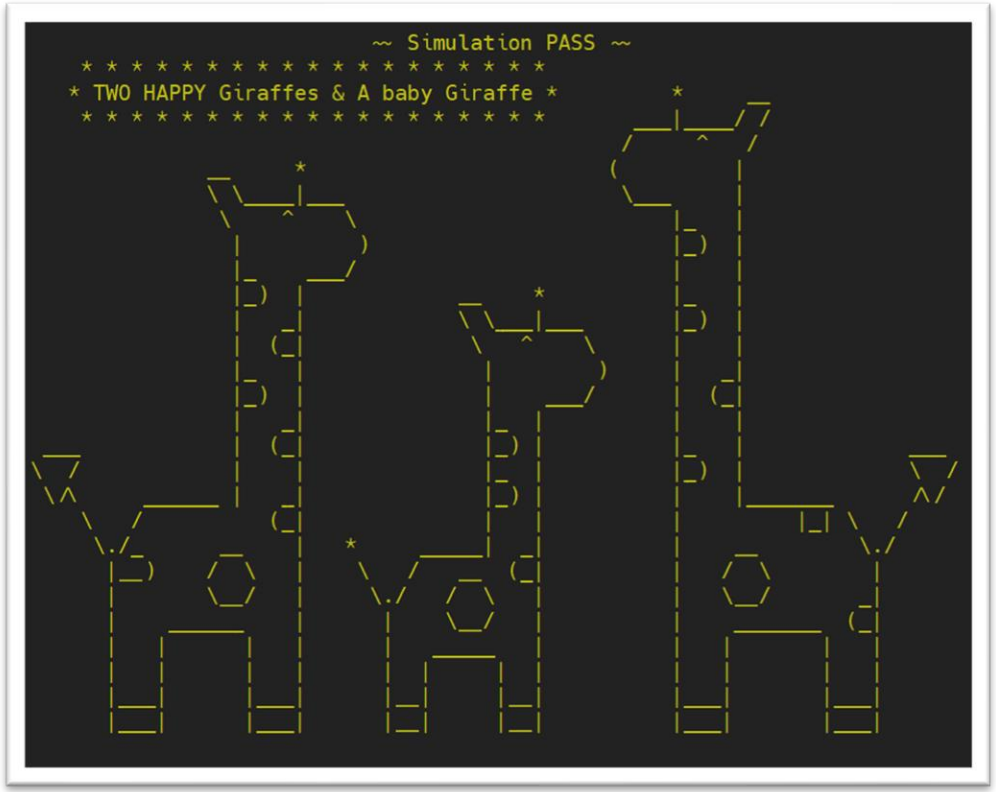
PASS Golden Happy Giraffe

prototype
~ ԼUԼU~ԼUԼU ~



PASS Tb 1

PASS Tb 2、3



Simulation Information

Please use command **make tb0, make tb1, make tb2, make tb3**.

Make Clean before submitting your code.

Students can only adjust src file and tb line 3 4 5.

Adjust line 3 if your file is “PE.v”.

Adjust line 4 to your own file path let tb_data able to read by testbench.

```
Warning-[STASKW_CO] Cannot open file
/home/rita/AOC/lab4_PE2/PE_tb.sv, 136
  The file '/tb0_data/ifmap0.txt' could not be opened. No such file or
  directory.
  Please ensure that the file exists with proper permissions.


Warning-[STASKW_INFDPFE] Invalid argument to $feof
/home/rita/AOC/lab4_PE2/PE_tb.sv, 150
  Expecting a valid file descriptor as argument to $feof, but NULL is passed.
  Please pass a valid file descriptor to $feof.
```

That terminal displays the warnings indicates you don’t adjust your path successfully.

Adjust line 5 to change your total simulation cycles.

PE_tb.sv | PE.sv

```
1  `timescale 1ns/10ps
2  `define CYCLE 10.0
3  `include "PE.sv"  // change to "PE.v" , if you implent by verilog
4  `define PATH "/home/.../lab4_PE/"  //change to your own path
5  `define MAX 100000 // adjust the cycle you need
6  `include "def.svh"
```

Report

Please follow the PE design rules on page 33, 58.

Cycle Ranking	point
Rank 0%-25%	3
Rank 25%-50%	2.4
Rank 50%-75%	1.7
Rank 75%-100%	1

(15% / 15% / 15%) **1. Pass all testbenches. Screenshots of passing 3 tbs.**

(10%) **2. Cycle comparison of each tb.** Each tb contains 3 points. 送一分

(20%) **3. Draw your own PE architecture.**

Write down your total spad storage space and the corresponding functions.

Explain how your PE works with waveform clearly. Mark down the values you mention.

(10%) **4. Demonstrate Scenario B dataflow** by completing 2 3row x 3col x 2ch ofmaps by 2-channel of 2 3x3 filters(kernel) and 2 5x5 ifmaps under parameters n=1, p=2(kernel), q=1(channel).

(b)

In a pass, each input data are read only once from the GLB, and the psums are stored back to the GLB only once when the processing is finished.

Report

(10%) **5. Compare scenario A,B,C from different aspects organized into a table.**

Analyze under **scenario A(n=2, p=1, q=1)**, **scenario B(n=1, p=2, q=1)**, **scenario C(n=1, p=1, q=2)** within a processing pass and across multiple passes to **complete 2 3row x 3col x 2ch ofmaps by 2-channel of 2 3x3 filters(kernel) and 2 5x5 ifmaps**.

(Hint : reuse distances of ifmap, filter and psum(lec4 p.116-126), proper spad size in a PE, numbers of memory read / write of ifmap, filter and psum and energy consumption (lec4 p.146、lec5 p.60-67), pros and cons etc.)

You can find the author's comments in Chapter "Flexibility to Map Multiple Dimensions" of this video.
https://www.youtube.com/watch?v=brhOo-_7NS4&t=2691s

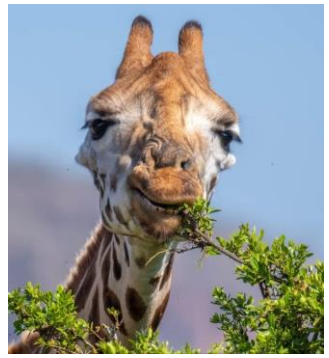
(5%) **6. Share your thoughts on this lab.** Any takeaways or advices?

你想說的話~ U Ho~ 送分題

(Bonus 20%) **7.** (15%)**Modify your PE to a pipelined architecture or with zero-gating / zero skipping function.** You can choose one from 3 functions to implement.

(5%) Theoretically compare with basic PE architecture on page 33, 58 by required cycles, effectual and ineffectual operations(lec4 p.16-19、25), etc.

File Format

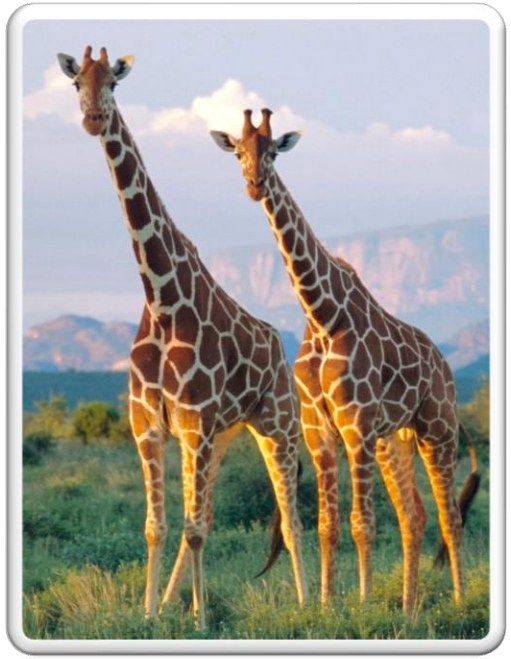


- Lab4_StudentID_Name.zip /.tar
  - StudentID_Name.pdf — Report
  - Lab4_PE — Code
    - 4 tb_data
    - PE_tb.sv
    - src (your design)
    - include
    - Makefile

Outline

- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- **Chapter7 Bonus**
- Chapter8 Supplementary

Testbench



TB

🦄 Simulate layer information given by customize instructions.

🦏 Simulate Global Buffer distributes filter, ifmap and ipsum to PE.

🐌 **Check if your design can process hybrid bandwidth.**

🐑 Simulate psums transfer between upper, lower PEs and Buffer

🐑 Check if opsums computation are correct.

🐑 Check if adder can process overflow.

clk
rst
set_info
layer_info
filter
filter_enable
filter_ready
ifmap
ifmap_enable
ifmap_ready
ipsum
ipsum_enable
ipsum_ready
opsum
opsum_enable
opsum_ready

PE

🦄 Be able to deal with different configurations by Control.

🦏 Be Able to receive filter, ifmap, ifmap and start computing in the correct time.

🐌 **Be Able to apply hybrid multiplier(lab3), adder, spad … etc to process hybrid bandwidth .**

🐑 Be Able to compute correct opsums and process overflow.

Testbench Signal

Signal	Bit	Indication	Note
clk	1	Clock	10ns as a cycle
rst	1	Reset	Turn everything to initial state
set_info	1	Simulate Configuration Scan Chain receives layer information from instructions, and send them to the PE Array	All PEs need to renew their layer information
Ch_size ifmap_column ofmap_column	3 6 6	the number of ifmap filter channels the number of columns in an ifmap row the number of columns in an ofmap row	3’d4 6’d10 6’d8
ifmap_Quant_size filter_Quant_size	4 4	Ifmap bit width Filter bit width	4’d4 4’d4

Given values equal to actual values

Testbench Signal

Signal	Bit	Note	
filter filter_enable filter_ready	8 1 1	filter [3:0] = **kernel** a **col** b filter [7:4] = **kernel** (a+1) **col** b	
ifmap ifmap_enable ifmap_ready	32 1 1	[19:16] = ifmap **col** (a+1) **ch** b [23:20] = ifmap **col** (a+1) **ch** (b+1) [27:24] = ifmap **col** (a+1) **ch** (b+2) [31:28] = ifmap **col** (a+1) **ch** (b+3)	[3:0] = ifmap **col** a **ch** b [7:4] = ifmap **col** a **ch** (b+1) [11:8] = ifmap **col** a **ch** (b+2) [15:12] = ifmap **col** a **ch** (b+3)
ipsum ipsum_enable ipsum_ready	24 1 1	[11:0] = ipsum / opsum ch a col b [23:12] = ipsum / opsum ch (a+1) col b	
opsum opsum_enable opsum_ready	24 1 1		

TestBench 3 Instruction

- **Data Distribution Order:**

		[15:0]	**[31:16]**	[15:0]	**[31:16]**	[15:0]	**[31:16]**	[15:0]	**[31:16]**	
		Col 1	**Col 2**	Col 3	**Col 4**	Col 5	**Col 6**	Col 7	**Col 8**	
Ifmap	Ch 1	1	**1**	2	**2**	3	**3**	4	**4**	…
	Ch 2	1	**1**	2	**2**	3	**3**	4	**4**	…
Distributed for 16 times	Ch 3	1	**1**	2	**2**	3	**3**	4	**4**	…
	Ch 4	1	**1**	2	**2**	3	**3**	4	**4**	…

Ifmap Buffer resets and prepares for 1st set of data

```
*Ifmap Distribution Info* Time:              10, Ifmap Buffer Coordinate [Batch Size][Column][Channel]
*Ifmap Distribution Info* Time:              10, Enable:0, Ifmap Buffer is preparing for[0][1,0][3,2,1,0]
```

1st set of data

ifmap[31:16] = { ifmap[0][1][3], ifmap[0][1][2],
ifmap[0][1][1], ifmap[0][1][0] } = {-1, -1, -4, -2}

ifmap[15:0] = { ifmap[0][0][3], ifmap[0][0][2],
ifmap[0][0][1], ifmap[0][0][0] } = { 6, 2, 3, -3}

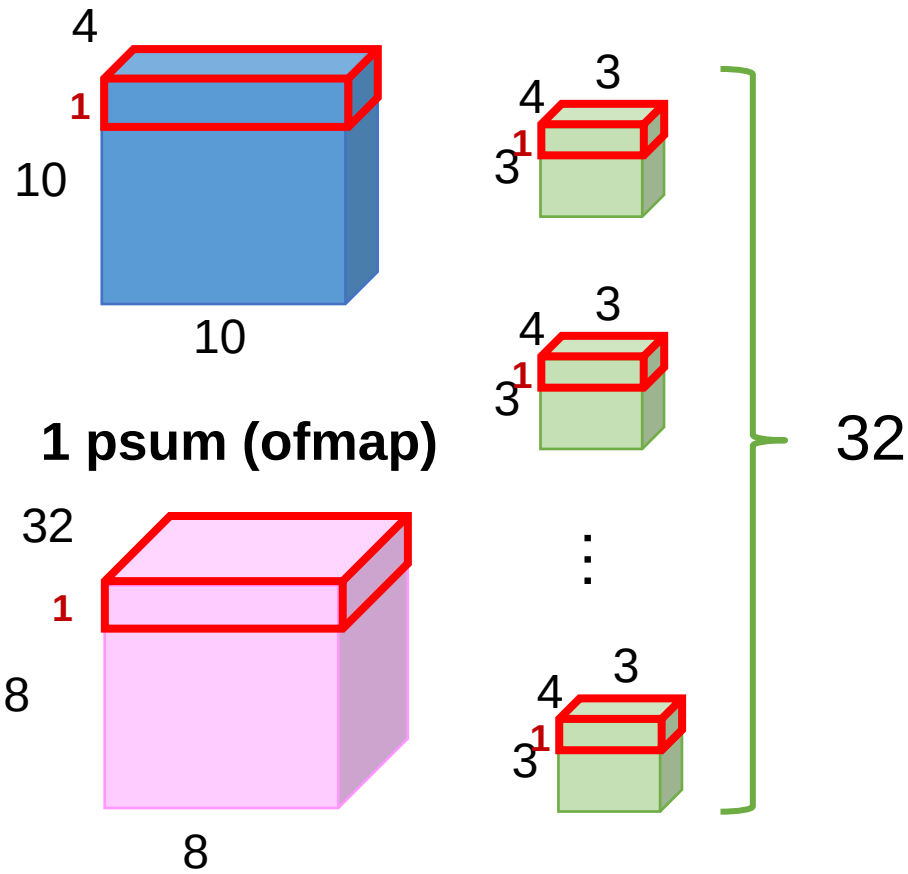
```
*Ifmap* Time:                70, ifmap[31:16]  -1 -1 -4 -2 / [15:0]   6  2  3 -3
*Ifmap* Time:                70, next ifmap Buffer[ 0][ 3, 2][3,2,1,0]
```

Last set of ifmap

```
*Ifmap* Time:              1060, last ifmap[31:16]:  0 -7 -1  2
*Ifmap* Time:              1060, last ifmap[15:0]:  1  5 -4  4
```

1 ifmap

32 kernels
16 processing pass



1 psum (ofmap)

TestBench 3 Instruction

- **Data Distribution Order:**

Kernel 1 [0:3]

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 2 [4:7]

	Col 1	Col 2	Col 3
Ch 1	1	5	9
Ch 2	2	6	10
Ch 3	3	7	11
Ch 4	4	8	12

Kernel 3 [0:3]

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

Kernel 4 [4:7]

	Col 1	Col 2	Col 3
Ch 1	13	17	21
Ch 2	14	18	22
Ch 3	15	19	23
Ch 4	16	20	24

Filter Buffer resets and prepares for 1st set of data

```
*Filter Distribution Info* Time:              10, Filter Buffer Coordinate [Kernel][Column][Channel]
*Filter Distribution Info* Time:              10, Enable:0, Filter Buffer is preparing for[1,0][0][0]
```

1st set of data

filter [7:0] = { filter[1][0][0], filter[0][0][0] } = {7, -7}

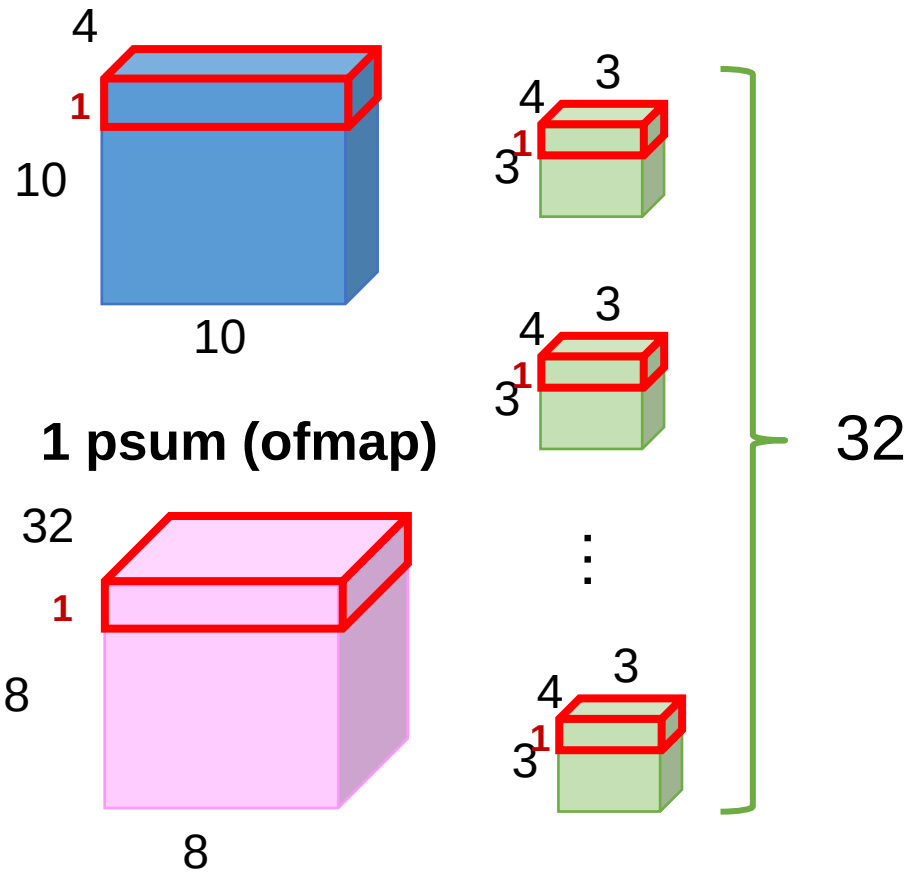
```
*Filter* Time:                30, filter[7:4][3:0]:  7 -7
*Filter* Time:                30, next filter Buffer[ 1, 0][ 0][ 1]
```

Last set of filter

```
*Filter* Time:                2410, last filter: -2 -2.
```

1 ifmap

32 kernels
16 processing pass



TestBench 3 Instruction

- **Ipsum Distribution Order:**

		Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
[11:0]	Ch 1	1	2	3	4	5	6	7	8
[23:12]	**Ch 2**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
[11:0]	Ch 3	9	10	11	12	13	14	15	16
[23:12]	**Ch 4**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
		… …							
[11:0]	Ch 31	121	122	123	124	125	126	127	128
[23:12]	**Ch 32**	**121**	**122**	**123**	**124**	**125**	**126**	**127**	**128**

Ipsum Buffer resets and prepares for 1st set of data

```
*Input Psum Distribution Info* Time:              10, Ipsum Buffer Coordinate [Batch Size][Channel][Column]
*Input Psum Distribution Info* Time:              10, Enable:0, Ipsum Buffer is preparing for[0][1,0][0]
```

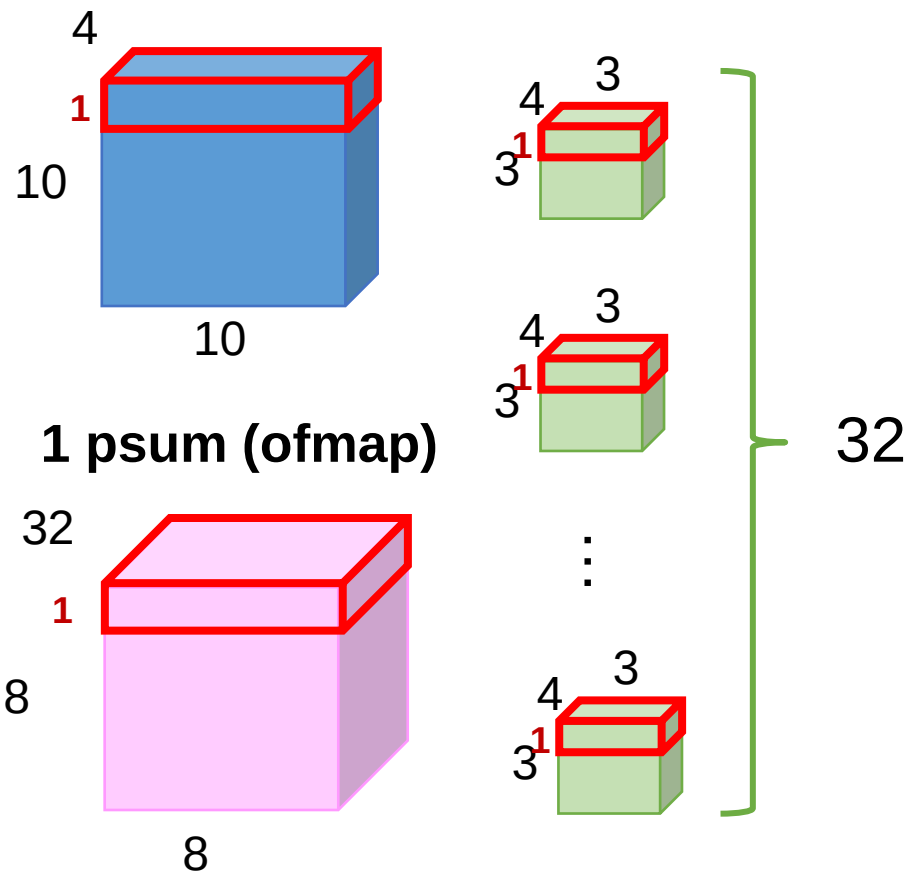
1st set of data

ipsum[23:0] = {ipsum[0][1][0], ipsum[0][0][0] } = { 355, 44}

```
*Ipsum* Time:                30, ipsum:    355     44, next ipsum Buffer[ 0][ 1, 0][ 1]
```

Last set of ipsum

```
*Ipsum* Time:                1890, last ipsum:    -575    -166
```



TestBench 3 Instruction

- **Opsum Order:**

		Col 1	Col 2	Col 3	Col 4	Col 5	Col 6	Col 7	Col 8
[11:0]	Ch 1	1	2	3	4	5	6	7	8
[23:12]	**Ch 2**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
[11:0]	Ch 3	9	10	11	12	13	14	15	16
[23:12]	**Ch 4**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
		… …							
[11:0]	Ch 31	121	122	123	124	125	126	127	128
[23:12]	**Ch 32**	**121**	**122**	**123**	**124**	**125**	**126**	**127**	**128**

Wrong Answer

[23:12] → [11:0]

```
 Time:                  110: Ofmap  0 channel 1, 0 column  1 is  WRONG.
Channel  1 Correct ans   -1512 / Your ans   -1555
Channel  0 Correct ans     883 / Your ans    1859
```

[23:12] → [11:0]

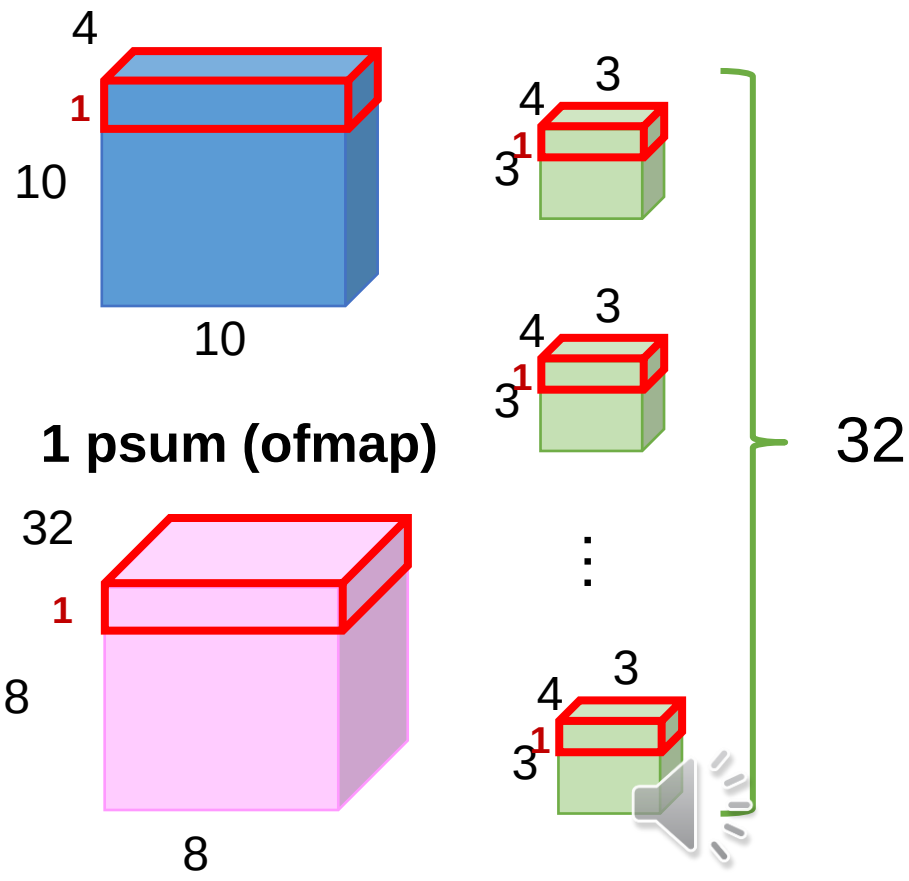
```
 Time:                  130: Ofmap  0 channel 1, 0 column  3 is  WRONG.
Channel  1 Correct ans    -283 / Your ans    -283
Channel  0 Correct ans    -676 / Your ans Unknown
```

Correct Answer (in yellow)

```
Time:                  140: Ofmap  0 channel 1, 0 column  4 :   -1095  -1040 is correct.
```

1 ifmap

32 kernels
16 processing pass



1 psum (ofmap)

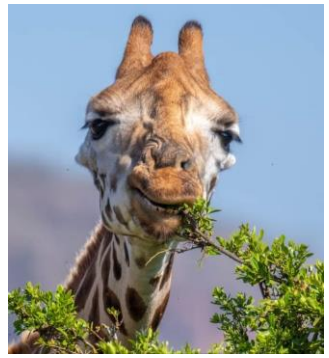
Hybrid PE Implementation

- Apply hybrid multiplier and hybrid adder
- Reuse your spad space be able to put in 4-bit data and related information
- Add extra hybrid functions to your PE Control Path

Report

- Screenshots of passing 4 tbs. Total required cycles of 4 tbs. Total spad space for accommodating 8-bit and 4-bit data info.
- Simply explain how your PE works, your modules supporting to address 4-bit data and draw hybrid architecture . (skip introducing hybrid multiplier)
- Write down how do you reuse hardware resources to accomplish 4tbs.
- 心得
- Deadline : 5/28 23:59:59
- 加Lab總分 2 分

File Format



- Bonus_StudentID_Name.zip /.tar
  - StudentID_Name.pdf — Report
  - StudentID — Code
    - 4 tb_data
    - PE_tb.sv
    - src (your design)
    - include
    - Makefile

Outline

- Chapter1 PE Introduction
- Chapter2 Demonstration Example
- Chapter3 Dataflow apply to VGG16-Cifar10 1st layer
- Chapter4 PE Architecture
- Chapter5 3 scenarios of accumulating psum
- Chapter6 Homework
- Chapter7 Bonus
- **Chapter8 Supplementary**

Related Material

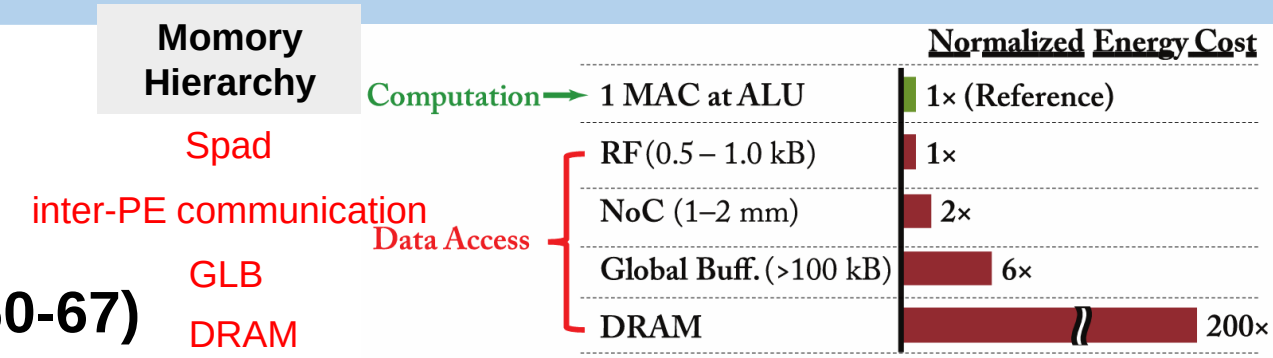
- The PE architecture was drawn by **draw.io**.
- **SRAM and Register**
- https://computationstructures.org/lectures/caches/caches.html
- **Key word for addressing zero data** : sparse tensor format、compressed sparse format、sparse cnn accelerator
- **Zero gating**

Data gating logic is implemented to exploit zeros in the ifmap for saving processing power. An extra 12-b *Zero Buffer* is used to record the position of zeros in the ifmap spad. If a zero ifmap value is detected from the zero buffer, the gating logic will disable the read of the filter spad and prevent the MAC datapath from switching. Compared with the PE design without the data gating logic, it can save the PE power consumption by 45%.

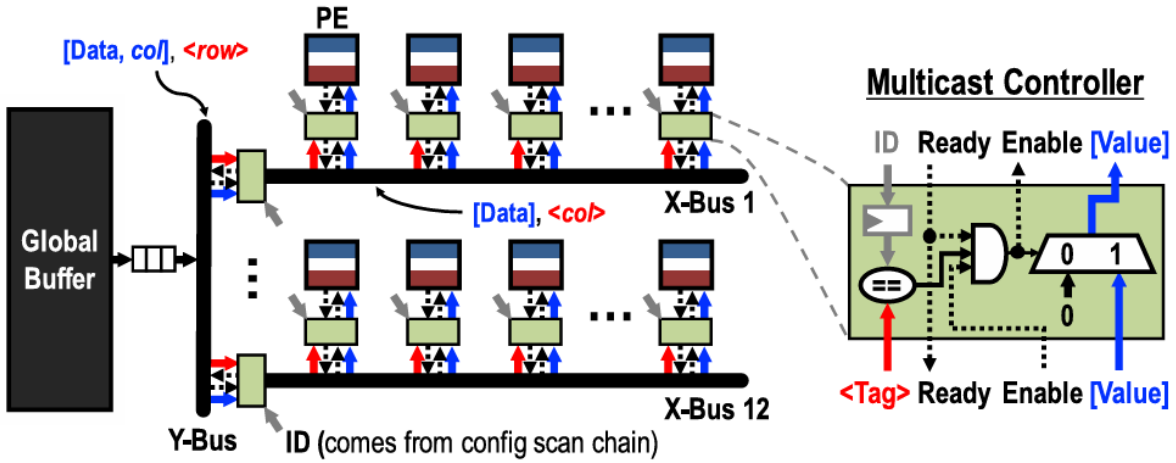
- **Pipelined**

This configuration controls the pattern with which the PE steps through the three spads. The datapath is pipelined into three stages: one stage for spad access, and the remaining two for computation. The computation consists of a 16-b two-stage

Related Material



- **Energy Estimation (lec4 p.146、lec5 p.60-67)**
- Y. -H. Chen, J. Emer and V. Sze, "**Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolutional Neural Networks**," 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), Seoul, Korea (South), 2016, pp. 367-379, doi: 10.1109/ISCA.2016.40.
- **Ready and Enable signals are to simulate Multicast Controller behavior**



Eyeriss has separate GINs for each of the three data types (filter, ifmap, and psum) to provide sufficient bandwidth from the GLB to the PE array. All GINs have 4-b *row* IDs to address the 12 rows. The filter and psum GINs use 4-b *col* IDs to address the 14 columns, while ifmap GIN uses 5 b to support maximum 32 ifmap rows passing in diagonal. The filter and psum GINs have data bus width of 64 b ($4b \times 16$ b), while the ifmap GIN has the data bus width of 16 b.

- **Network-on-Chip Keyword : unicast multicast broadcast (lect4 p.250-267)**

Related Material

- **Set_info signal is to simulate Configuration Scan chain behavior**
- Our testbench required bandwidth is $(3 + 6 + 6 + 4 + 4 = 23)$ bits.
- To reduce bandwidth, we can reduce our total bandwidth and apply **a 4-bit scan chain**.

Signal	Bits	Max Value	Note
Ch_size ifmap_column ofmap_column	3 -> 2 6 6 -> 5	4 34 32	Given values equal to (actual values-1) e.g. 8 bit ifmap ifmap_Quant_size = 3'd7
ifmap_Quant_size filter_Quant_size	4 -> 3 4 -> 3	8 8	
Total required B.W.	**23->19**	**Minimum required cycles to reconfigure PEs reduce from 6 to 5.**	

The accelerator runs the processing of a CNN layer-by-layer. For each layer, it first loads the configuration bits into a 1794 b scan chain serially to reconfigure the entire accelerator, which takes less than 100 μs. These bits configure the accelerator for the processing of filters and fmaps in a certain shape, which includes setting up the PE array computation mappings (Section IV-A) and NoC data delivery patterns (Section V-B). They are generated offline and are statically accessed at runtime. Then, the accelerator loads tiles of the ifmaps and filters from DRAM for processing, and the computed ofmaps are written back to DRAM. Batches of ifmaps for the same layer can be processed sequentially without further reconfigurations of the chip.

- **Accelerator Addressed Multiple Layers**
- M. Alwani, H. Chen, M. Ferdman and P. Milder, "**Fused-layer CNN accelerators**," 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, 2016

Related Material

Giraffe Photos

- Discovery – Giraffes Have Daycares, Lunch Buddies, and Yearslong Relationships
- https://www.discovery.com/nature/giraffes-have-daycares--lunch-buddies--and-yearslong-relationshi
- Animal Planet – chomp chomp chomp
- https://www.facebook.com/AnimalPlanet/posts/pfbid0mHxmU78QAGZAHWHmDt9ZYdoP3spFzpbYa1spe9HBvBgL5Qr2SA6NmtHGGcafnWgYl
- Great Africa – mother-child bond
- https://www.facebook.com/Greatafrica6/photos/a.100241138452369/6805302770900116/?type=3&source=57&paipv=0&eav=AfZelTUTPBWZmkZk4H5TWFnQcZpzrkGcOTp1MZhLrhN4jfsJm-2DujH47os7dCIzXmk&_rdr
- Sleeping Baby Giraffe
- https://www.pinterest.com/jaydee08ca/giraffe-sleeping/

Supplementary – 長頸鹿

長頸鹿

Creator：
Yiten Tsai

煩頸鹿

Creator：黃咕哩
口頭禪：UHO~ 唷齁唷齁
興趣：瑜珈

格格不鹿

Creator：屁股沙
朋友：豬豬

Lulu

興趣 ： 睡覺
Best Friend : Sleepy
Favorite Color : Bright Yellow

Supplementary – Sleepy Confidential Document

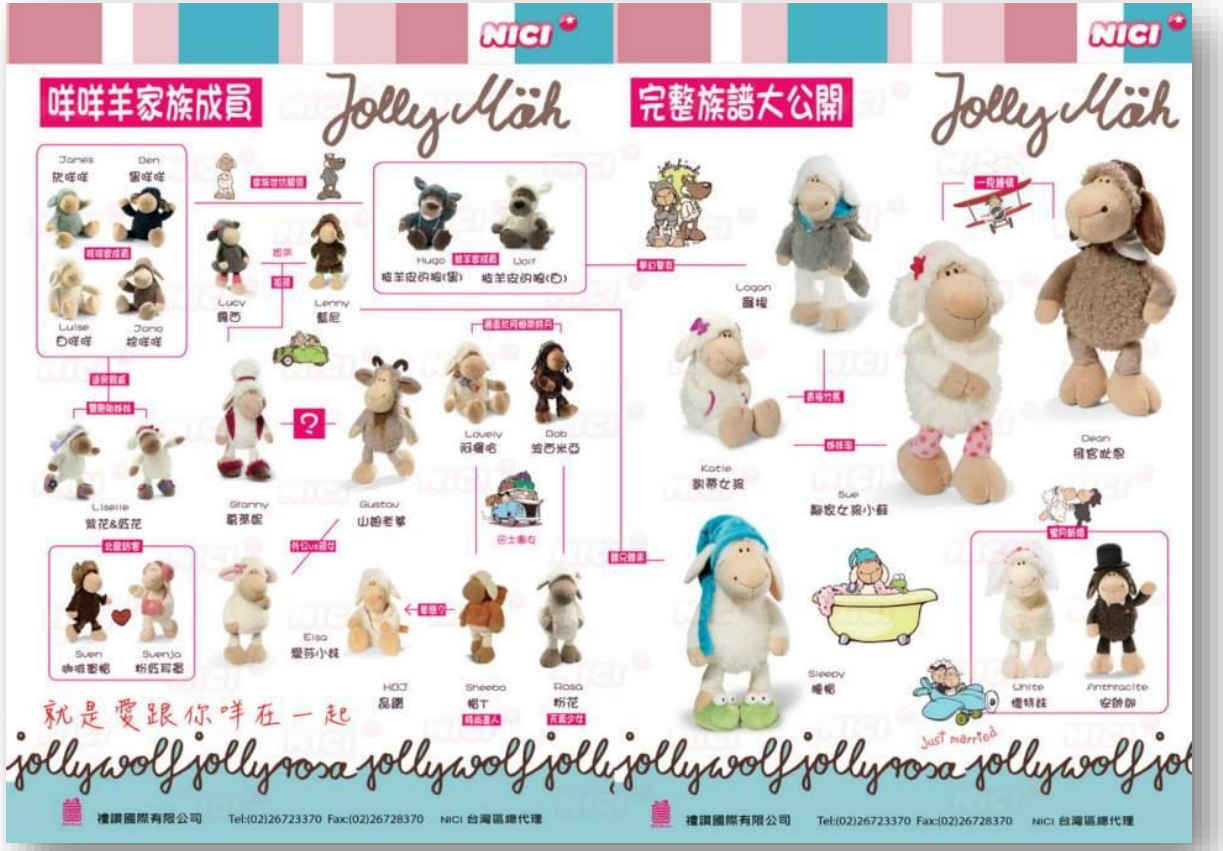
Sleepy Self Introduction

https://www.eslite.com/product/1004175202682303279000?utm_source=APP&utm_medium=share&utm_campaign=sku

Sleepy Family Introduction

https://www.facebook.com/nici.fans/photos/a.595106473874225/595106493874223/

年齡：1.1
血型：Love Peace的O型
星座：居家巨蟹座
個性：好睡
專長：連睡72小時
喜好：參加睡覺馬拉松比賽
專屬名言：睡眠是一種藝術



Thank you for listening ~