Lab 3
Hybrid Multiplier

Video Link Click Me (*´д`)~
Advisor: Chia-Chi Tsai
TAs: 湯詠涵 吳柄葳
Architecture2 supported by CASlab 林柏維學長 invaluably.
Gmail : course.aislab@gmail.com

Outline

- Introduction
- Hardware Architecture
- Signed Number Multiplication
- Homework
- Supplementary

Outline

- **Introduction**
- Hardware Architecture
- Signed Number Multiplication
- Homework
- Supplementary

2D CNN Computation

- Filters with **M** Kernels ⇨ **M** Ofmap Channels
- Ifmap **N** Batch sizes ⇨ **N** Ofmaps
- **C** Filters、Ifmaps Channels ⇨ Accumulate psums of **C** Channels

TABLE I
SHAPE PARAMETERS OF A CNN LAYER

Shape Parameter	Description
N	batch size of 3D fmaps
M	# of 3D filters / # of ofmap channels
C	# of ifmap/filter channels
H/W	ifmap plane height/width
R/S	filter plane height/width
E/F	ofmap plane height/width

In the following labs, we define "kernel" as:

Kernel 1

Kernel M

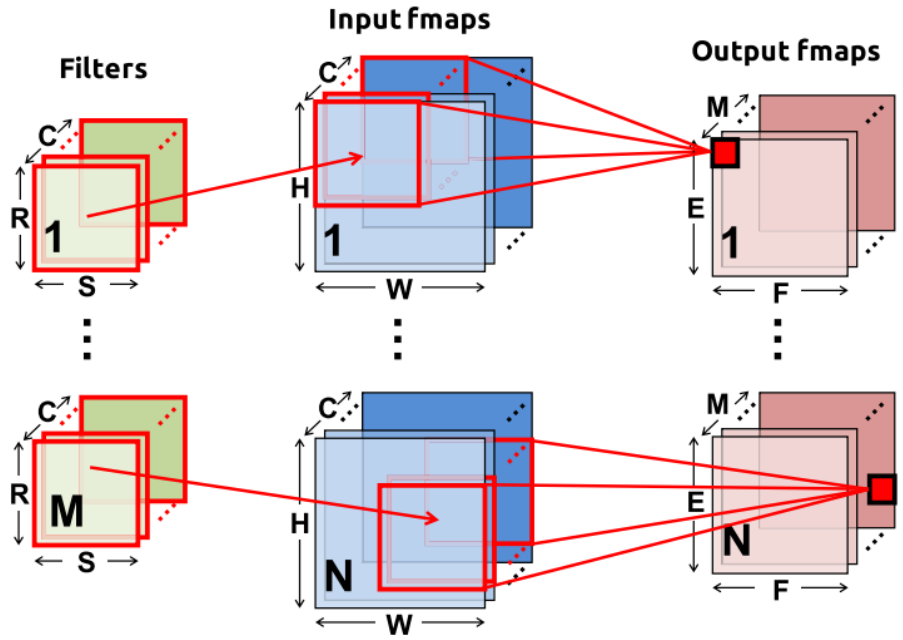


Fig. 1. Computation of a CNN layer.

$$\mathbf{O}[z][u][x][y] = \text{ReLU}\left(\mathbf{B}[u] + \sum_{k=0}^{C-1}\sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\mathbf{I}[z][k][Ux+i][Uy+j] \times \mathbf{W}[u][k][i][j]\right),$$

$$0 \le z < N, \quad 0 \le u < M, \quad 0 \le y < E, \quad 0 \le x < F$$

$$E = (H - R + U)/U, \quad F = (W - S + U)/U \qquad (1)$$

where **O**, **I**, **W**, and **B** are the matrices of the ofmaps, ifmaps, filters, and biases, respectively. U is a given stride size. Fig. 1 shows a visualization of this computation (ignoring biases). After the convolutions, activation functions, such as the rectified linear unit (ReLU) [31], are applied to introduce nonlinearity.

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.

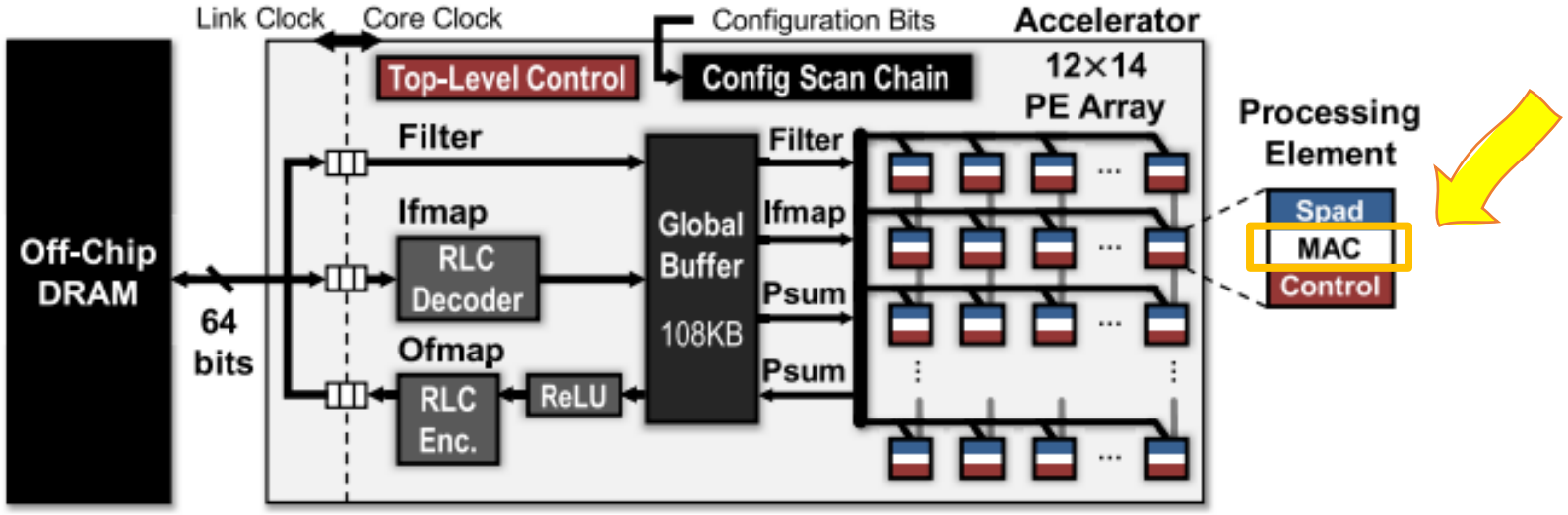
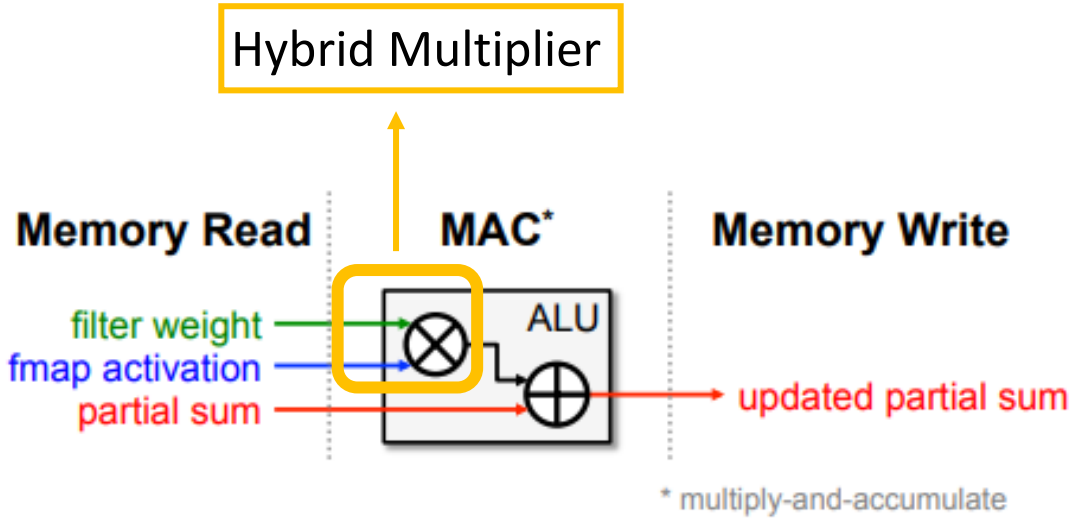


Fig. 2. Eyeriss system architecture.

MAC Introduction

- Full Name : Multiply and Accumulator
- Compute the product of two numbers and add that product to an accumulator.

$$a \leftarrow a + (b \times c)$$



Metrics	LeNet 5	AlexNet
Top-5 error†	n/a	16.4
Top-5 error (single crop)†	n/a	19.8
Input Size	28×28	227×227
# of CONV Layers	2	5
Depth in # of CONV Layers	2	5
Filter Sizes	5	3,5,11
# of Channels	1, 20	3-256
# of Filters	20, 50	96-384
Stride	1	1,4
Weights	2.6k	2.3M
MACs	283k	666M
# of FC Layers	2	3
Filter Sizes	1,4	1,6
# of Channels	50, 500	256-4096
# of Filters	10, 500	1000-4096
Weights	58k	58.6M
MACs	58k	58.6M
Total Weights	60k	61M
Total MACs	341k	724M
Pretrained Model Website	[56]‡	[57, 58]

Maximum 2896M memory accesses required
Without optimizing hardware design

Outline

- Introduction
- **Hardware Architecture**
- Signed Number Multiplication
- Homework
- Supplementary

Mixed-Precision Accelerator

- **Mixed-precision accelerator is the trend**.

- Wang, Kuan et al. "**HAQ: Hardware-Aware Automated Quantization With Mixed Precision**." 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (2018): 8604-8612.

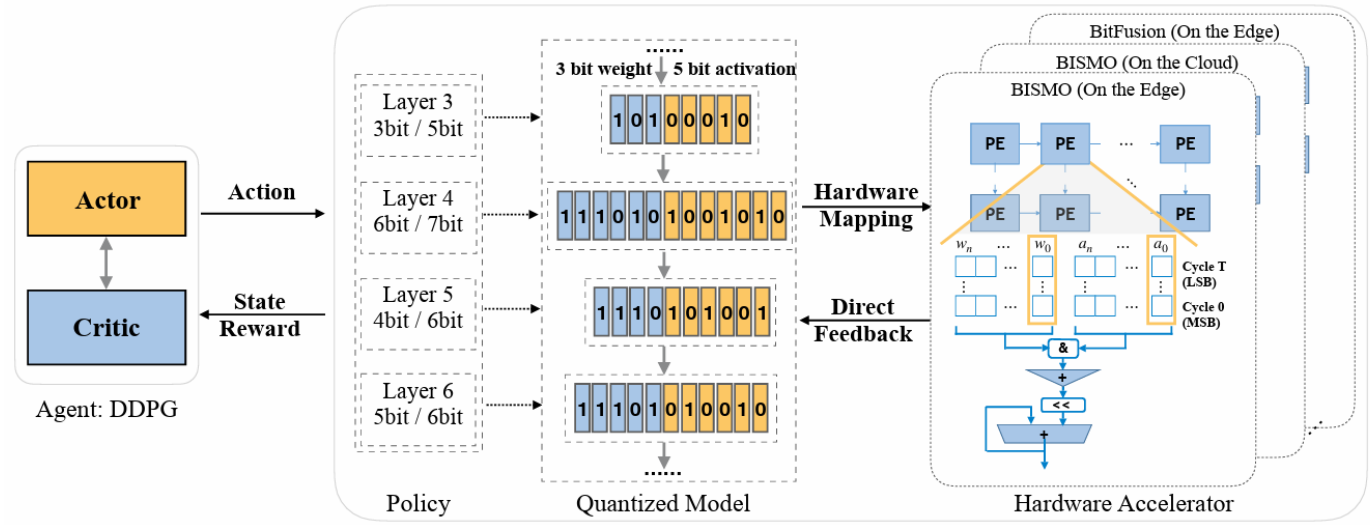


Figure 2: An overview of our **H**ardware-Aware **A**utomated **Q**uantization (HAQ) framework. We leverage the reinforcement learning to automatically search over the huge quantization design space with hardware in the loop. The agent propose an optimal bitwidth allocation policy given the amount of computation resources (*i.e.*, latency, power, and model size). Our RL agent integrates the hardware accelerator into the exploration loop so that it can obtain the direct feedback from the hardware, instead of relying on indirect proxy signals.

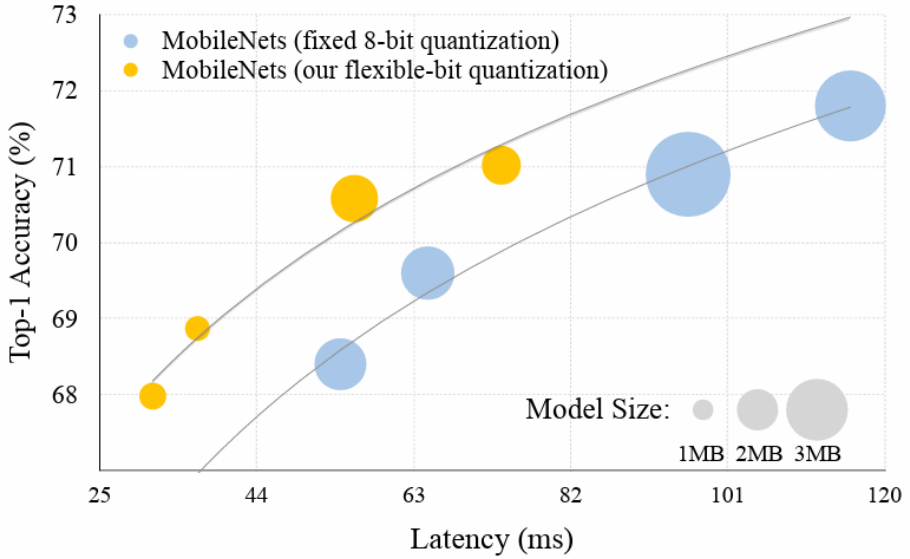
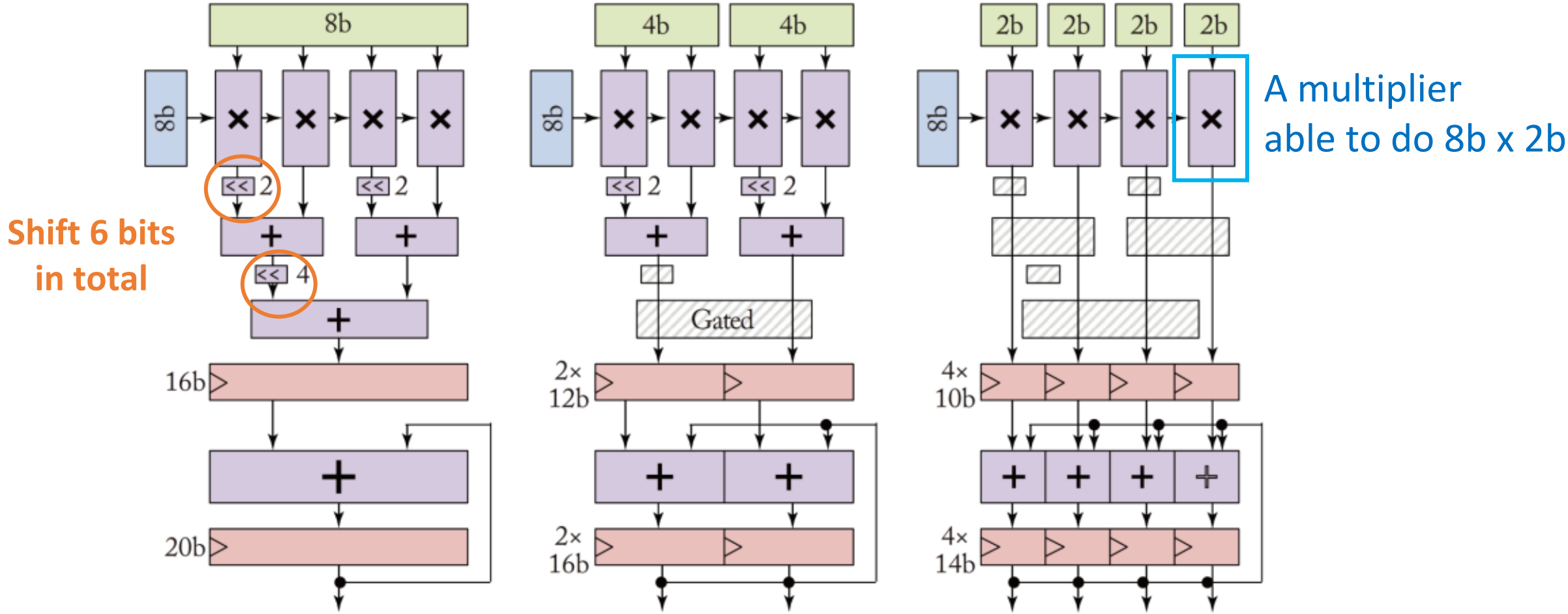


Figure 1: We need **mixed precision** for different layers. We quantize MobileNets [12] to different number of bits (both weights and activations), and it lies on a better pareto curve (yellow) than fixed bit quantization (blue). The reason is that different layers have different redundancy and have different arithmetic intensity (OPs/byte) on the hardware, which advocates for using mixed precision for different layers.

Hybrid Multiplier Architecture 1

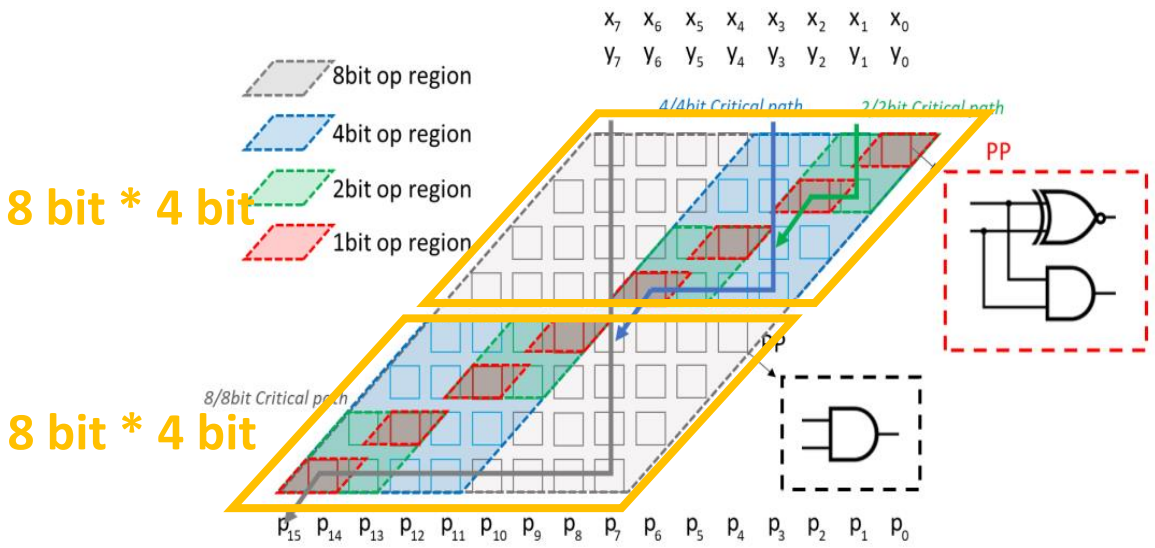
V. Camus, L. Mei, C. Enz and M. Verhelst, "**Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing,**" in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 4, pp. 697-711, Dec. 2019



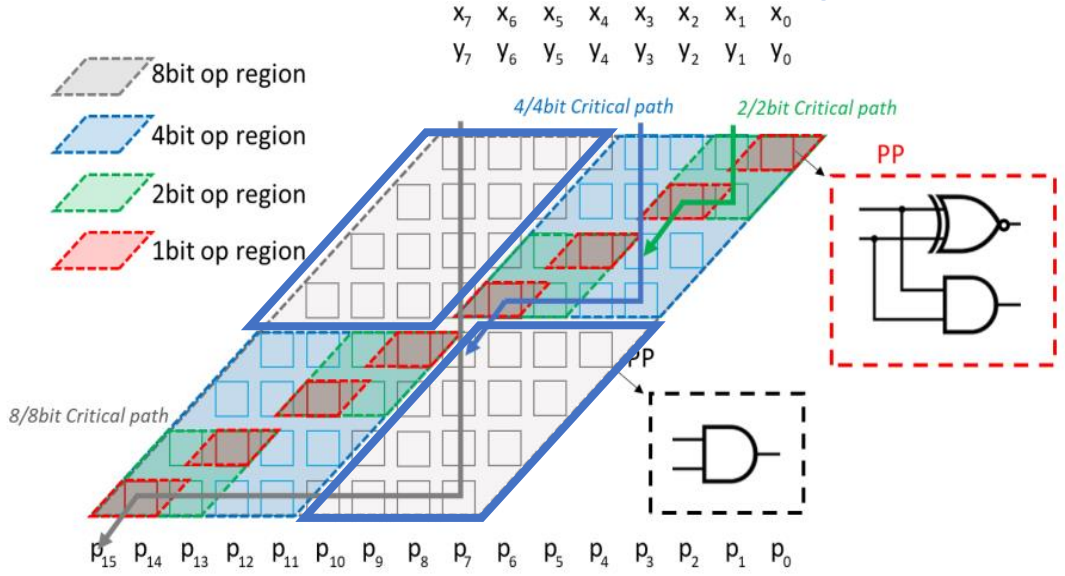
Architecture 2

- Unsigned Number Multiplication
- To implement the architecture, we can first complete a "4-bit & 4-bit numbers multiplier".
- By using 2 sets of a "4-bit numbers multiplier", we can finish "8-bit & 4-bit numbers multiplication".
- In the same way, "8-bit numbers multiplier" can be completed by a patchwork of "4-bit & 4-bit numbers multiplier".

2 sets of 8-bit & 4-bit numbers multiplication



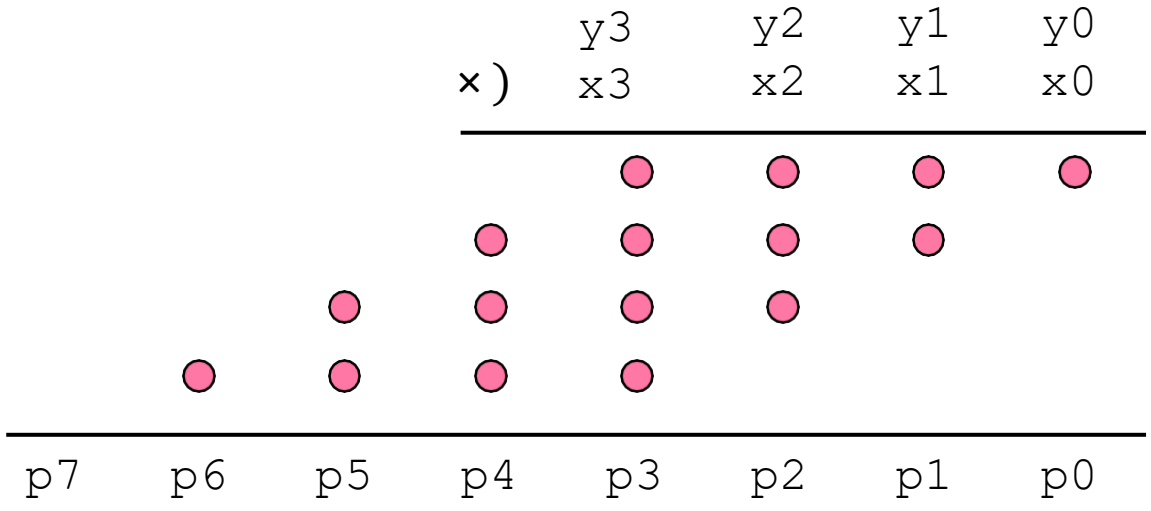
4 sets of 4-bit & 4-bit numbers multiplication

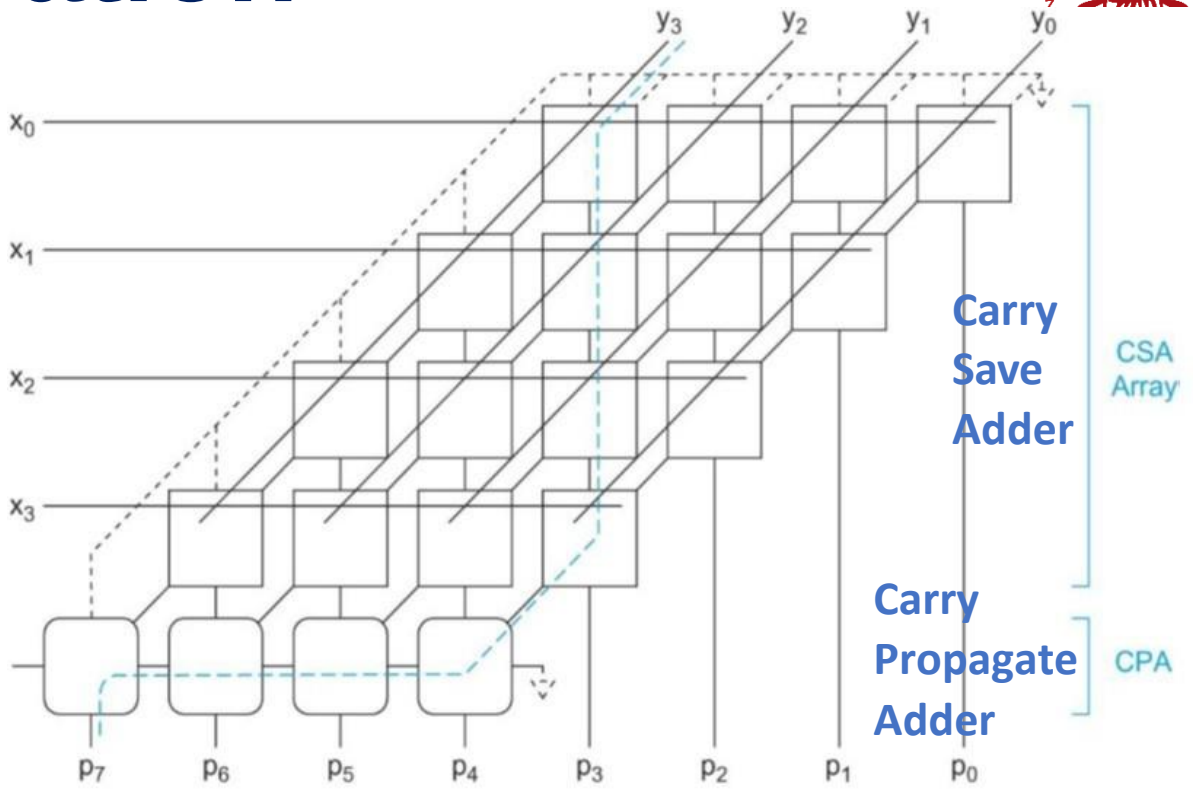


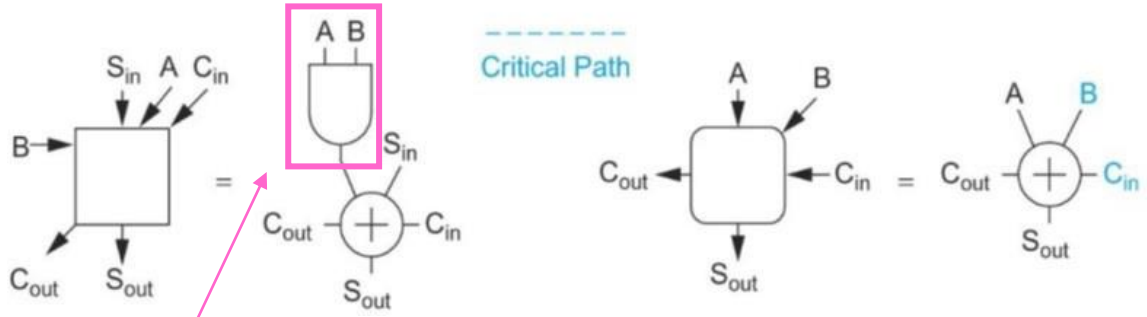
4-bit Numbers Multiplication

Multiply (3 steps)

1. Partial Product Generator (PPG)
   - Do AND operation on each bit of x & y
   - Produce the pink points ●
2. Partial Product Reduction Tree (PPRT)
   - Efficiently add all pink points ●, we use CSA
3. Carry-Propagate Adder (CPA)
   - Last Adder







● Pink point in the left graph

CPA vs CSA

Take 3 numbers addition as example

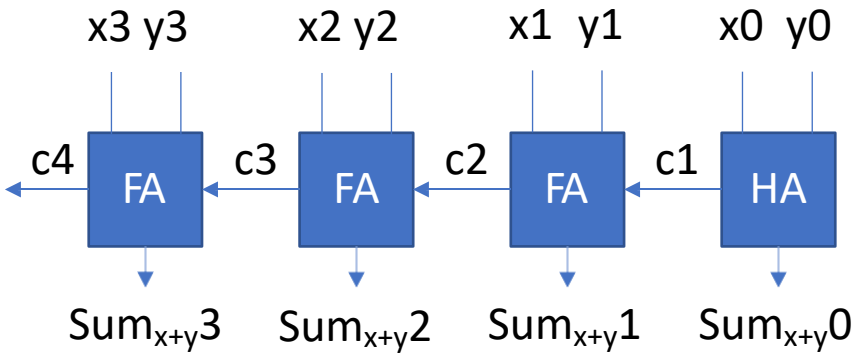
$$\begin{array}{llll} x3 & x2 & x1 & x0 \\ +)\ y3 & y2 & y1 & y0 \\ +)\ z3 & z2 & z1 & z0 \\ \hline C'4 & Sum3 & Sum2 & Sum1 & Sum0 \end{array}$$

M = How many numbers to add = 3
N = How many bits in a number = 4

Carry-Propagate Adder

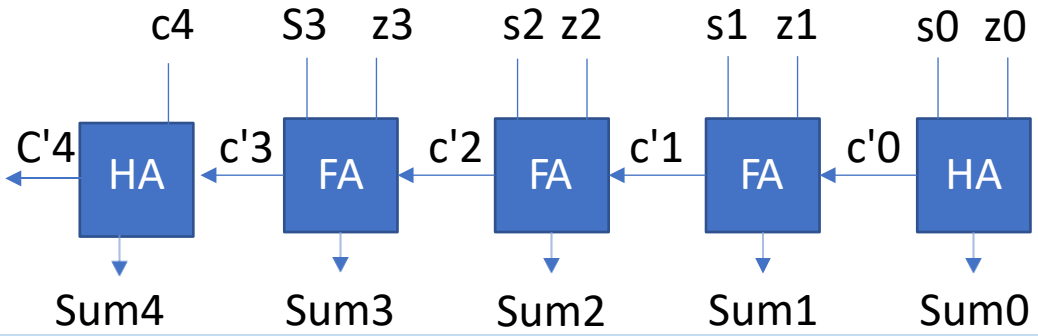
Step 1 : $Sum_{x+y} = x + y$

Time : N Adder



Step 2 : $Sum = Sum_{x+y} + z$

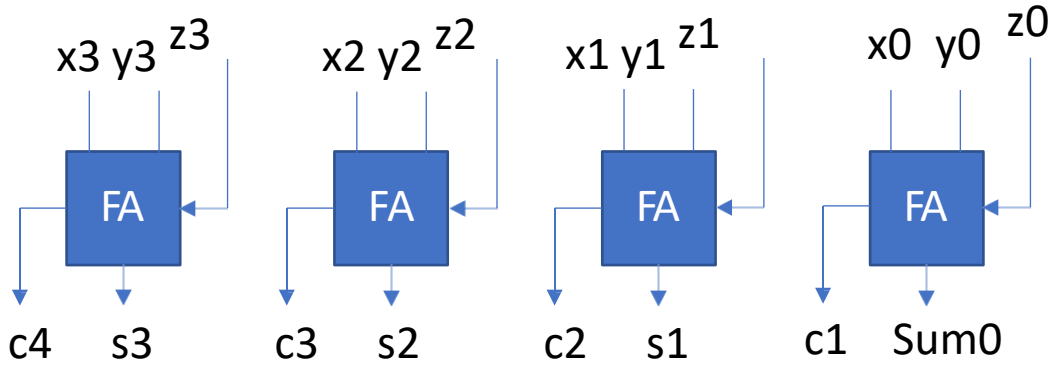
Time : (N+1) Adder



Carry-Save Adder

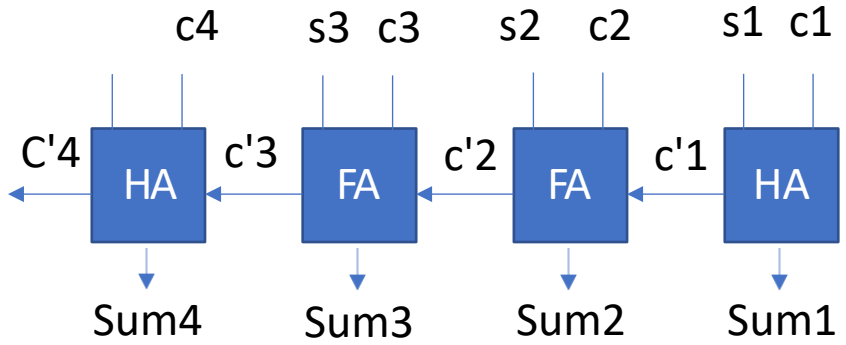
Step 1 : Each bit does FA independently

Time : 1 FAs



Step 2 : Do Carry-Propagate, Sum = c + s

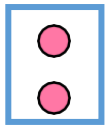
Time : N Adder



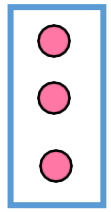
$$\begin{array}{llll} s3 & s2 & s1 & s0 \\ +)c4 & c3 & c2 & c1 & 0 \end{array}$$

Example of 4-bit Unsigned Num. Multiplication

- Your hardware design doesn't have to follow the architecture below.
- For pink points ●, which we can generate the results simply by AND gates, we use CSA to reduce clock delay.
- For the rest of computation, which generates the final answer, we use CPA.

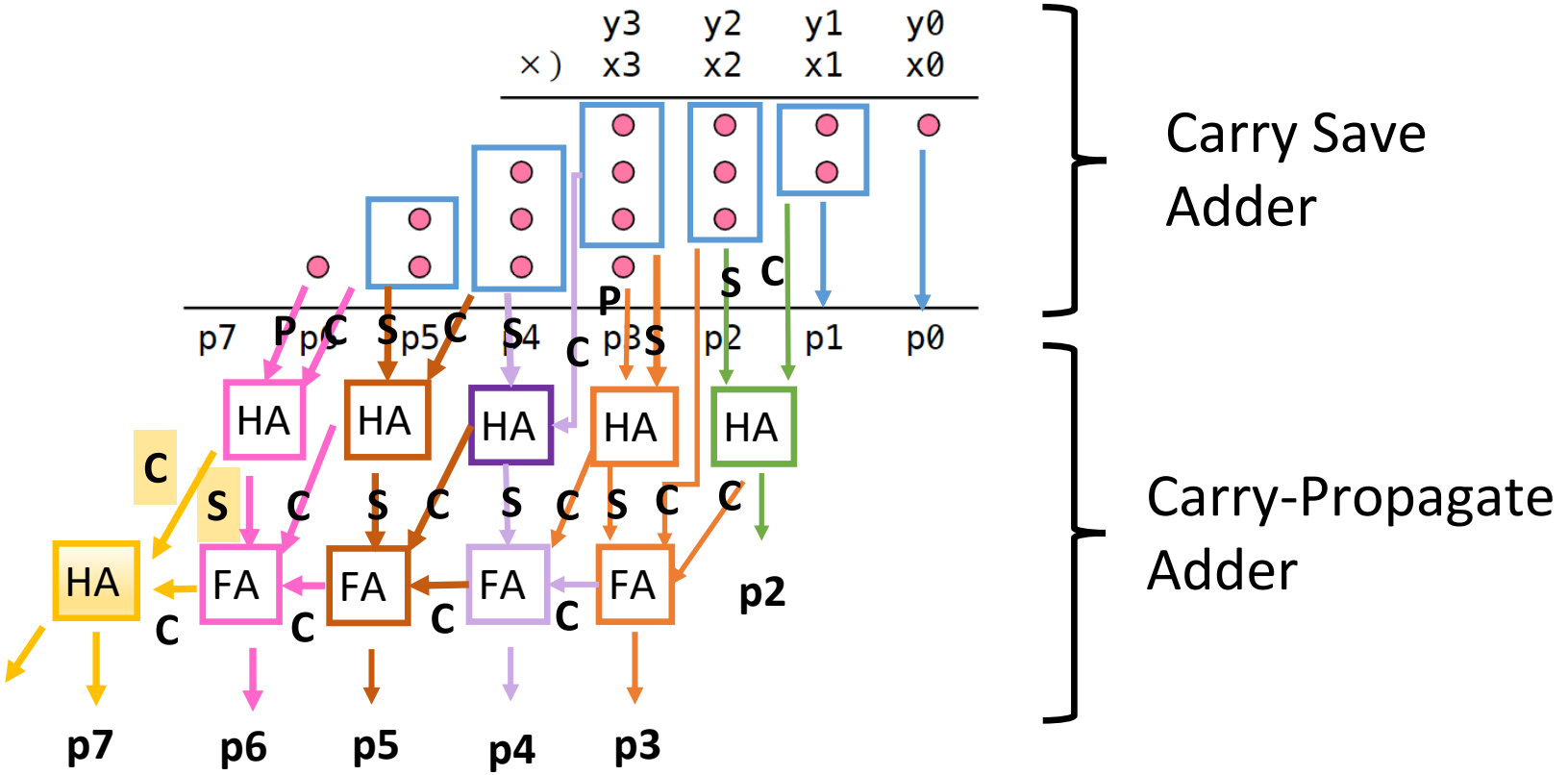


Use Half Adder
add 2 pink points



Use Full Adder
add 3 pink points

P Pink Point
S Sum
C Carry



Yellow C,S can't be 1 at the same time
Cout is no possible to be 1
Yellow HA can be replaced by a OR Gate

Wallace Tree Adder

- Related website : https://www.twblogs.net/a/5ef17ca24b16c91a28496d91
- Another Partial-Product-Reduction-Tree Method Proposed by Chris Wallace in 1964.
- 每3個加數分爲一組，壓縮至2個加數，循環往復
- Perform multiple addition operations in parallel.
- Excel in high-speed applications with large data bit widths.

Number of Partial Products	Number of Levels of Wallace Tree
3	1
4	2
5≤p≤6	3
7≤p≤9	4
10≤p≤13	5
14≤p≤19	6
20≤p≤28	7
29≤p≤42	8
43≤p≤63	9

Architecture Figure

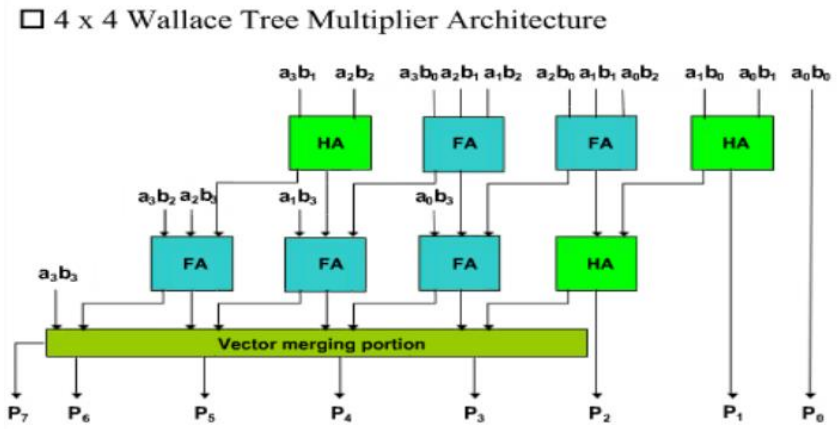
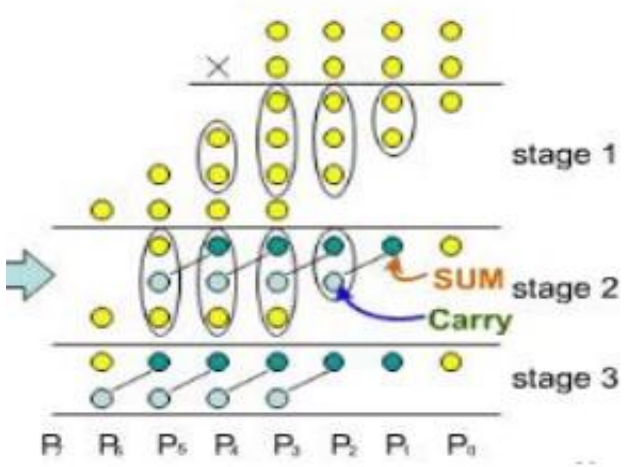


Fig.5. Circuit diagram of 4x4 multiplier architecture

Point Figure



- 圖片來源
- http://www.ijarse.com/images/fullpdf/1523964041_IIMT19.pdf (April 2018)
- Dubey, Ashwani & Sharma, Poonam & Goyal, Ayush. (2018). Efficient Computing in Image Processing and DSPs with ASIP based Multiplier. Recent Patents on Engineering.

Other Architectures

- Key word : mixed-precision、bit-accurate accelerator
- **Following two papers introduce Spatial & Temporal Architecture**

V. Camus, L. Mei, C. Enz and M. Verhelst, "Review and Benchmarking of Precision-Scalable Multiply-Accumulate Unit Architectures for Embedded Neural-Network Processing," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 9, no. 4, pp. 697-711, Dec. 2019, doi: 10.1109/JETCAS.2019.2950386.

Able to do signed and unsigned number multiplication

H. Sharma et al., "**Bit Fusion**: Bit-Level Dynamically Composable Architecture for Accelerating Deep Neural Network," 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA), Los Angeles, CA, USA, 2018, pp. 764-775,doi:10.1109/ISCA.2018.00069.

1. Sx, Sy : Signed Bit.
   X_{2b} = {x1,x0}. Y_{2b} = {y1,y0}.
   operands are signed(-2~1)
   or unsigned(0~3)
2. Extend 2-bit operands by and-gate
   X'_{3b} = {x2,x1,x0}. Y'_{3b} = {y2,y1,y0}
3. Do multiplication
   Product P6b = {p5,p4,p3,p2,p1,p0}

2-bit binary	Unsigned Value Extended binary	Signed Value Extended Binary
00	0(10) 000	0(10) 000
01	1(10) 001	1(10) 001
10	2(10) 010	-2(10) 110
11	3(10) 011	-1(10) 111

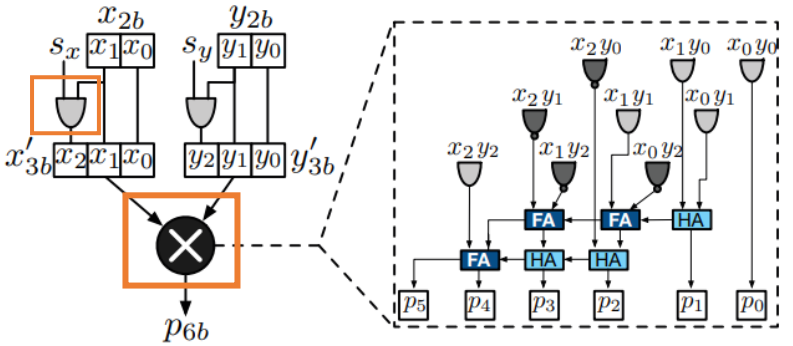


Fig. 5: A single BitBrick. (HA: Half Adder, FA: Full Adder.)

Outline

- Introduction
- Hardware Architecture
- **Signed Number Multiplication Algorithm**
- Homework
- Supplementary

Signed Binary Multiplication -- Robertson Algorithm

Example : multiplicand X = 0100 , (-X) = 1100

multiplier Y = 1010 , $Y = y3\ y2\ y1\ y0$, $Y = -2^3 + 2^1$

Signed Extension

$$
\begin{array}{rll}
 & 0100 & (4)_{10} \\
X) & 1010 & (-6)_{10} \\
\hline
 & 0000000 & \\
 & 000100 & \\
 & 00000 & \\
+) & 1100 & \\
\hline
 & 1101000 & (-24)_{10}
\end{array}
$$

$since\ Y\ is\ negetive\ \ (y_3 = 1)$

Value of 4-bit signed number

$$Y = \begin{cases} \sum_{i=0}^{3} 2^i y_i & ,if\ Y\ is\ positive\ or\ zero\ (y_3 = 0) \\ -2^3 + \sum_{i=0}^{2} 2^i y_i\ , & if\ Y\ is\ negetive\ (y_3 = 1) \end{cases}$$

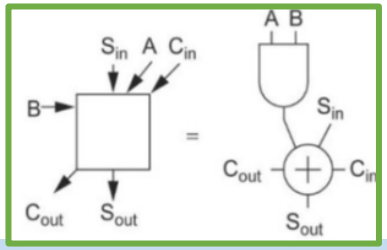
Example : $Y_1 = (1010)_2 = (-6)_{10} = (-2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0)$

$Y_2 = (0110)_2 = (+6)_{10} = (\ 2^3 \times 0 + 2^2 \times 1 + 2^1 \times 1 + 2^0 \times 0)$

Product of 2 signed numbers

$$\text{product} = \begin{cases} (\sum_{i=0}^{3} 2^i y_i) \times X & ,if\ Y\ is\ positive\ or\ zero\ (y_3 = 0) \\ (-2^3 + \sum_{i=0}^{2} 2^i y_i) \times X\ , & if\ Y\ is\ negetive\ (y_3 = 1) \end{cases}$$

Hardware
Add a Mux for A
Selector y_3



Signed Binary Multiplication
-- Booth's Algorithm

- multiplicand X : 0100 / two's complement (-X) : 1100
- Y is a n-bit two's complement signed number

$$Y = -y_{(n-1)}2^{(n-1)} + \sum_{0}^{N=n-2} y_N \, 2^N$$
$$Y = -y_{(n-1)}2^{(n-1)} + y_{(n-2)}2^{(n-2)} + y_{(n-3)}2^{(n-3)} + \ldots\ldots + y_1 2^1 + y_0 2^0$$
$$Y = (-y_{(n-1)} + y_{(n-2)})2^{(n-1)} + (-y_{(n-2)} + y_{(n-3)})2^{(n-2)} + \ldots\ldots + (-y_1 + y_0)2^1 + (-y_0 + y_{-1})2^0$$

Step 1:
y(-1) = 1'b0

Step 2: y[0][-1] =2'b00 (shift)

Step 2: y[1][0] =2'b10(sub)
Get 4*(-2)

Step 2: y[2][1]=2'b01(add)
Get 4*(4)

Step 2: y[3][2] =2'b10(sub)
Get 4*(-8)

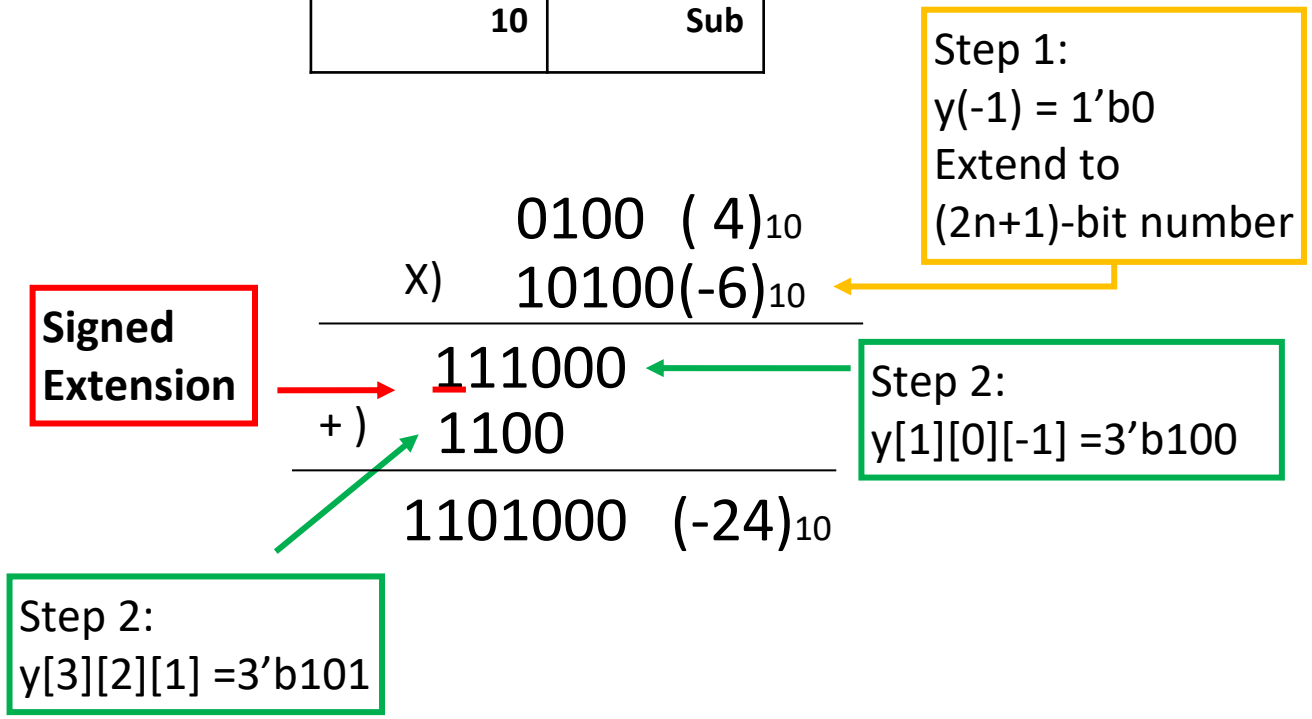
```
      0100   ( 4)10
X)    10100 (-6)10
-------------------
      0000000
      111100
      00100
+ )   1100
-------------------
      1101000   (-24)10
```

Multiplier	Operation
00,11	Shift(Zero)
01	Add X
10	Sub X / Add (-X)

Signed Binary Multiplication -- Modified Booth's Algorithm

- Derivated from Booth's Algorithm

Multiplier	Operation
00,11	Shift(Zero)
01	Add
10	Sub



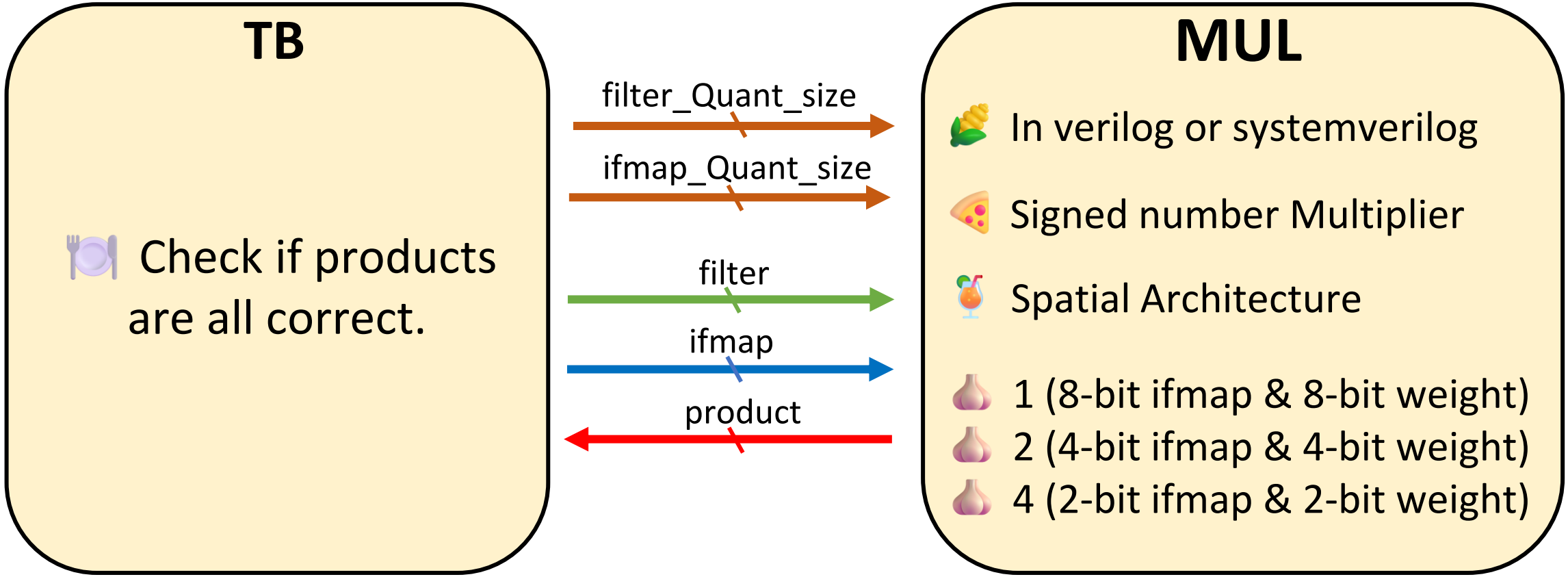
Multiplier	X(multiplicand)	operation
000	Zero	Shift + shift
001	X	Shift + Add
010	2X-X = X	Add + Sub
011	2X	Add + Shift
100	-2X	Sub + Shift
101	-2X+X = -X	Sub + Add
110	-X	Shift + Sub
111	Zero	Shift + Shift

Outline

- Introduction
- Hardware Architecture
- Signed Number Multiplication
- **Homework**
- Supplementary

Testbench

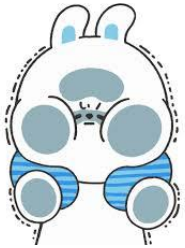
- Design a combinational multiplier by applying Signed Number Multiplication Algorithm and perform hybrid multiplication within a clock cycle (10ns).



TestBench Signal Introduction

Signal	Indication	Example TB0	Example TB1	Example TB2
ifmap	Input feature Map (Signed number)	[7:0] = ifmap	[7:4] = 4'd0000 [3:0] = ifmap	[5:4] = i1 [1:0] = i0
filter	Weight (Signed number)	[7:0] = filter	[7:4] = f1 [3:0] = f0	[5:4] = f1 [1:0] = f0
ifmap_Quant_size	Ifmap bit number	4'd8 = 8bit	4'd4 = 4bit	4'd2 = 2bit
filter_Quant_size	Filter bit number	4'd8 = 8bit	4'd4 = 4bit	4'd2 = 2bit
product	Ifmap * filter	Psums accumulated by 24 bits [23:0] = ifmap * filter	Psums accumulated by 12 bits [23:12] = ifmap * f1 [11:0] = ifmap * f0	Psums accumulated by 6 bits [23:18] = i1 * f1 [17:12] = i0 * f1 [11:6] = i1 * f0 [5:0] = i0 * f0

Simulation Result tb 0



- **Debug Tip :** Set Notation -> Signed 2's Complement
- **Error Message :**

Unknown

```
 Hybrid Multiplier has some errors
Your Result:        127 *        -128 is unknown
```

Error

```
 Hybrid Multiplier has some errors
Correct Ans:        -69 *         109=  -7521
Your result:        -69 *         109=-122865
```

- **Display Pattern :**

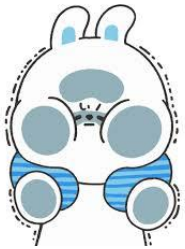
PASS

```
                !!Simulation PASS!!
                  __
               _\/_____________________________
              /      |_| |_|                    \
             |   .-.                             |
             _\   __                           _|__
            (___(_____________________________________)

          Creator NCKU AISystem Lab SOUP
Inspired by https://www.instagram.com/p/CujcWdHysYn/?img_index=1
```

FAIL

```
                !!Simulation Failed!!
                  __
               _\/_____________________________
              /      |_| |_|                    \
             |   .-.                             |
             _\   __                           _|__
            (___(_____________________________________)

          Creator NCKU AISystem Lab SOUP
Inspired by https://www.instagram.com/p/CujcWdHysYn/?img_index=1
```

Simulation Result tb 1 2



- **TB1** Error message :

```
 Hybrid Multiplier has some errors
1. Correct Ans:          -1*          -5=          5
1. Your result:          -1*          -5=         -1
2. Correct Ans:          -1*          -5=          5
2. Your result:          -1*          -5=      -1050
```

1. [23:12] = ifmap * f1

2. [11:0] = ifmap * f0

- **TB2** Error message :

```
 Hybrid Multiplier has some errors
1. Correct Ans:           1*           1=          1
1. Your result:           1*           1=          0
2. Correct Ans:           0*           1=          0
2. Your result:           0*           1=          0
3. Correct Ans:           1*          -2=         -2
3. Your result:           1*          -2=          0
4. Correct Ans:           0*          -2=          0
4. Your result:           0*          -2=          0
```

1. [23:18] = i1 * f1

2. [17:12] = i0 * f1

3. [11:6] = i1 * f0

4. [5:0] = i0 * f0

Report

- (10% / 10% / 15%) Choose 1 from 3 algorithms to implement.

  Pass TB0/TB1/TB2.Screenshot all pass screens.

- (10%) Explain how your multiplier works, its architecture and the algorithm you apply.

  (10%) Introduce each module and write down how do you reuse hardware resources to accomplish 3tbs.

- **Answer following questions in your report. Don't have to implement.**

- (10%) How many (minimum) FLOPs and MACs does VGG16-Cifar10 1st layer require? Write down your calculation process. (without batch normalization and bias)

- (10%) How many (minimum) FLOPs and MACs does a fully connected (512-10) layer require? Write down your calculation process. (without batch normalization and bias)

- (10%) Compare the difference among using Robertson Algorithm, Booth Algorithm and Modified Booth Algorithm with Hybrid Multiplier with a table.

- (10%) (Architecture 2) Design 4-bit & 4-bit multiplier (signed-number multiplication) with full and half adders follow architecture and point figure format in page 14. Introduce your architecture and method. Count and mark the latency/critical path of 4-bit & 4-bit multiplier.

  (you can use Adder as time unit).

- (5%) Share your thoughts on this lab. Any takeaways or advices? Or anything you want to say to the assistants. 認真表達心得一律滿分.

Wallace Tree Adder

- Proposed by Chris Wallace in 1964.
- Another Partial-Product-Reduction-Tree Method.
- Perform multiple addition operations in parallel.
- Excel in high-speed applications with large data bit widths.

Number of Partial Products	Number of Levels of Wallace Tree
3	1
4	2
5≤p≤6	3
7≤p≤9	4
10≤p≤13	5
14≤p≤19	6
20≤p≤28	7
29≤p≤42	8
43≤p≤63	9

Architecture Figure

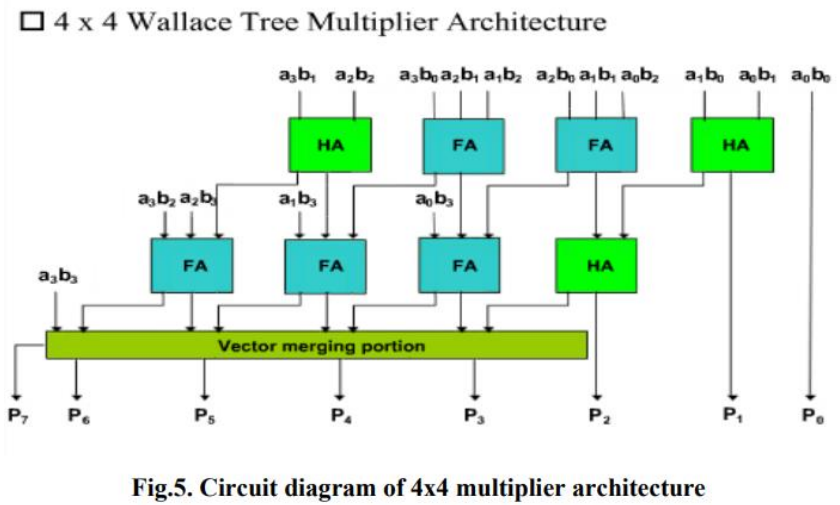
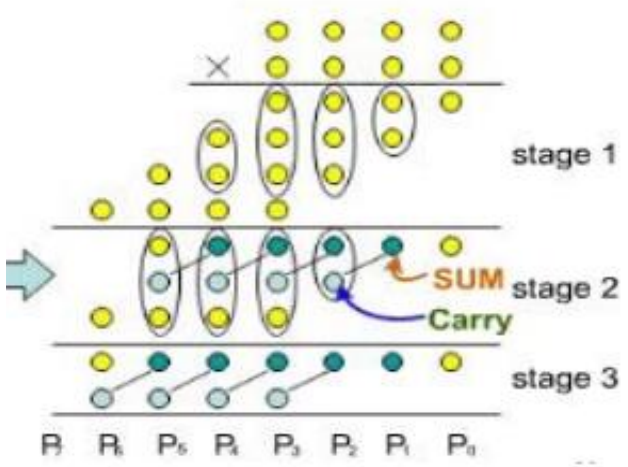


Fig.5. Circuit diagram of 4x4 multiplier architecture

Point Figure



- 圖片來源
- http://www.ijarse.com/images/fullpdf/1523964041_IIMT19.pdf (April 2018)
- Dubey, Ashwani & Sharma, Poonam & Goyal, Ayush. (2018). Efficient Computing in Image Processing and DSPs with ASIP based Multiplier. Recent Patents on Engineering.

File Format

Lab3_StudentID_Name.zip /.tar

- StudentID_Name.pdf — Report
- StudentID — Code
  - 3 TBs
  - src
  - include
  - Makefile

Make Clean

before submitting your code.

Outline

- Introduction
- Hardware Architecture
- Signed Number Multiplication
- Homework
- **Supplementary**

8-bit AI Accelerator

- A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi and J. Kepner, **"AI and ML Accelerator Survey and Trends,"** 2022 IEEE High Performance Extreme Computing Conference (HPEC), Waltham, MA, USA, 2022, pp. 1-10, doi: 10.1109/HPEC55821.2022.9926331.

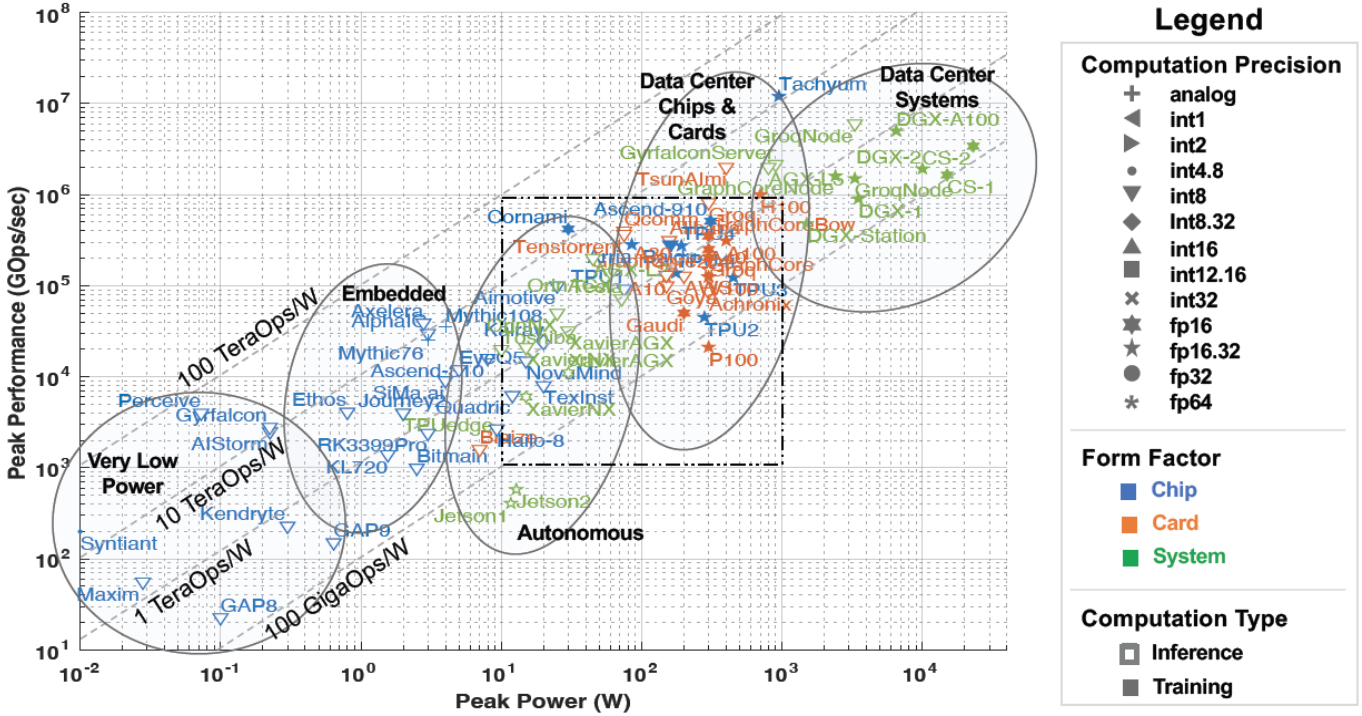


Fig. 2: Peak performance vs. power scatter plot of publicly announced AI accelerators and processors.

There are several observations comments for us to appreciate on Figure 2.

- Int8 continues to be the default numerical precision for embedded, autonomous and data center inference applications. This precision is adequate for most AI/ML applications with a reasonable number of classes. However, some accelerators also use fp16 and/or bf16 for inference. For training, has become integer representations
- Among the very low power chips, what is not captured is the other features beyond the machine learning accelerator on the chip. It is very common in this category and the Embedded category to release system-on-chip (SoC) solutions, which often include low-power CPU cores, audio and video analog-to-digital converters (ADCs),

Psum Bit Width

- V. Sze, Y. -H. Chen, T. -J. Yang and J. S. Emer, "**Efficient Processing of Deep Neural Networks: A Tutorial and Survey,"** in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

- We must add (8bit filter * 8bit ifmap) for $C\ x\ R\ x\ S$ times.
- Maximum / Minimum psum is equal to $\pm\ 2^8\ x\ 2^8\ x\ C\ x\ R\ x\ S$.
- Therefore, using $log_2\ (\ 2^8\ x\ 2^8\ x\ C\ x\ R\ x\ S\ )$ bits to accumulate it is quit enough.
- The parameters in the right figures are the following values :

  N = 8

  R = S = 3

  maximum **C = 512** in VGG16

  M = $log_2\ (\mathbf{C\ x\ R\ x\ S}\ ) = \mathbf{12.17}$

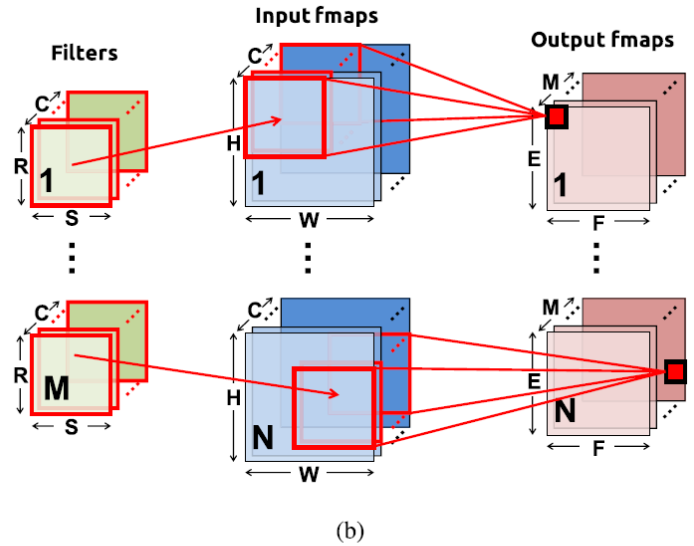


Fig. 9. ***Dimensionality of convolutions. (a) 2-D convolution in traditional image processing. (b) High dimensional convolutions in CNNs.***

- We select 24-bit as our psum bitwidth.
- **The adder needs to be able to detect overflow and clamp to $[-2^{24},\ (\ 2^{24}-1)\ ]$.**

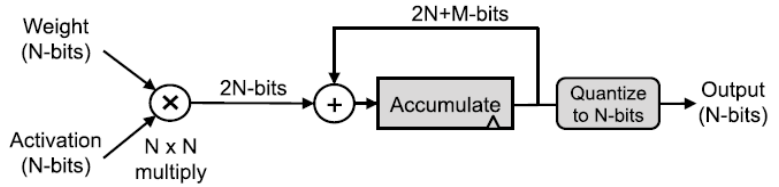


Fig. 39. ***Reducing the precision of MAC.***

Related Material

- **FLOPs** : floating-point operations **FLOPS** : floating point operations per **second**
- **Prerequisite Knowledge**

  Related Keyword : Binary. Two's complement. Combinational. Sequential.
- https://www.youtube.com/@silvinahanonowachman3310/videos
- **Temporal Architecture of Booth Algorithm**
- https://witscad.com/course/computer-architecture/chapter/fixed-point-arithmetic-multiplication
- **Division of Binary Number**
- https://www.wikihow.com/Divide-Binary-Numbers
- **CNN Metrics**
- https://medium.com/ching-i/cnn-parameters-flops-macs-cio-%E8%A8%88%E7%AE%97-9575d61765cc
- Dtto friends：Remi 兔 ， 腸太郎
- https://www.instagram.com/p/CujcWdHysYn/?img_index=1
- https://www.instagram.com/dttofriends/

Thank you for listening ~