WASHINGTON STATE UNIVERSITY

High-Throughput Multiplier Architectures Enabled by Intra-Unit Fast Forwarding

Jihee Seo and Dae Hyun Kim

School of Electrical Engineering and Computer Science
Washington State University

ARITH'19, Kyoto, Japan (June 10 - 12)

Outline

- **Motivation**
- **Related work**
  - Conventional arithmetic operation.
  - On-line arithmetic operation.
- **Our main work**
  - Intra-unit forwarding.
  - High-throughput multiplier architectures (***proposed***).
  - Application of our proposed architectures.
    - NBBE-2, RBBE-4, and CRBBE-4.
- **Simulation results**
- **Conclusion**

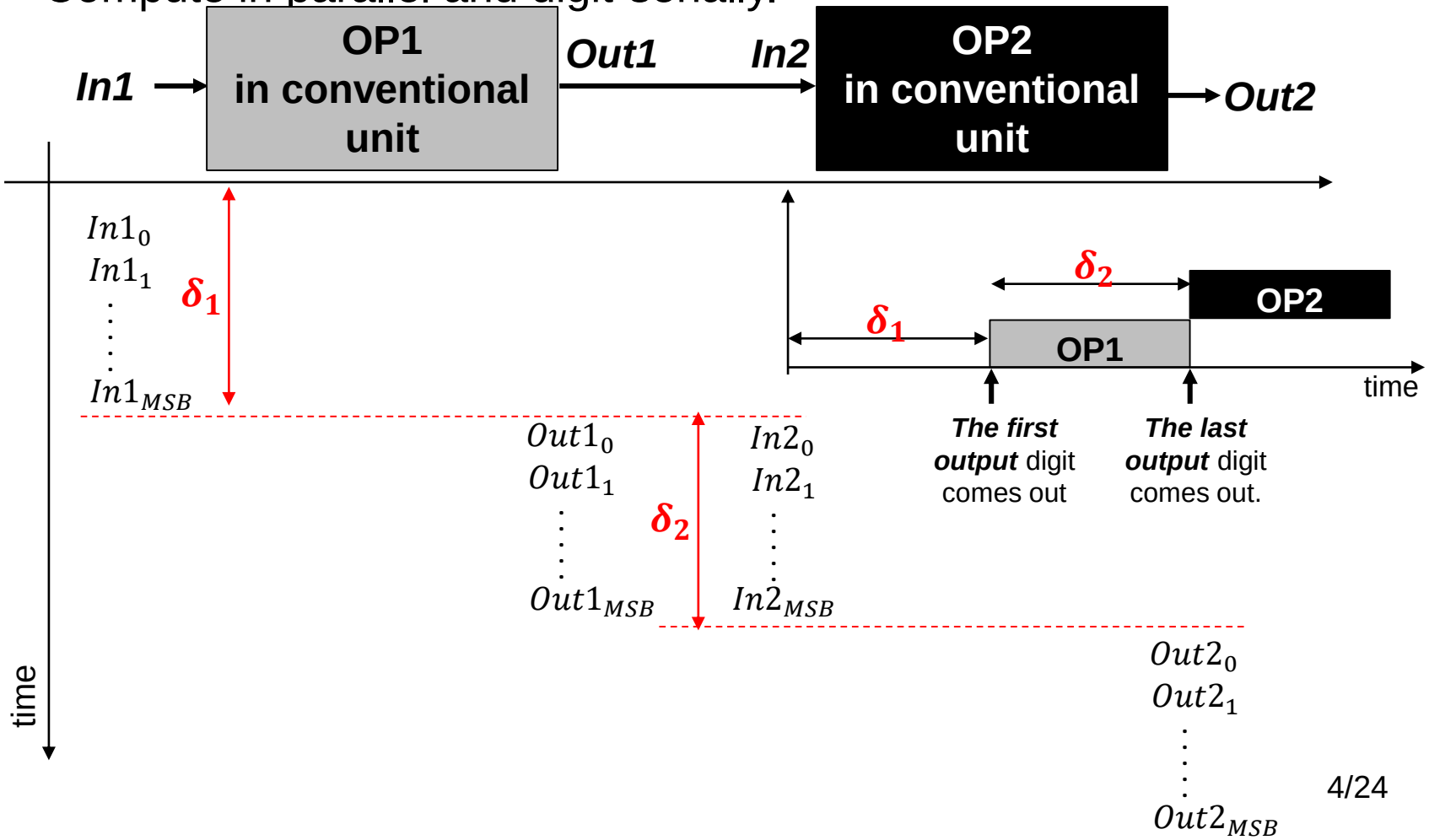
Arithmetic unit for high throughput

- The amount of data to be processed is hugely increased.
  - Compute-intensive application : need to complete computation with shorter execution time.
  - Memory-intensive application : need to process large data loaded from memory in time.
- ➔ The importance of **high-throughput processing unit** goes up.

- **The performance of arithmetic units** has a great impact on the throughput of processing unit.

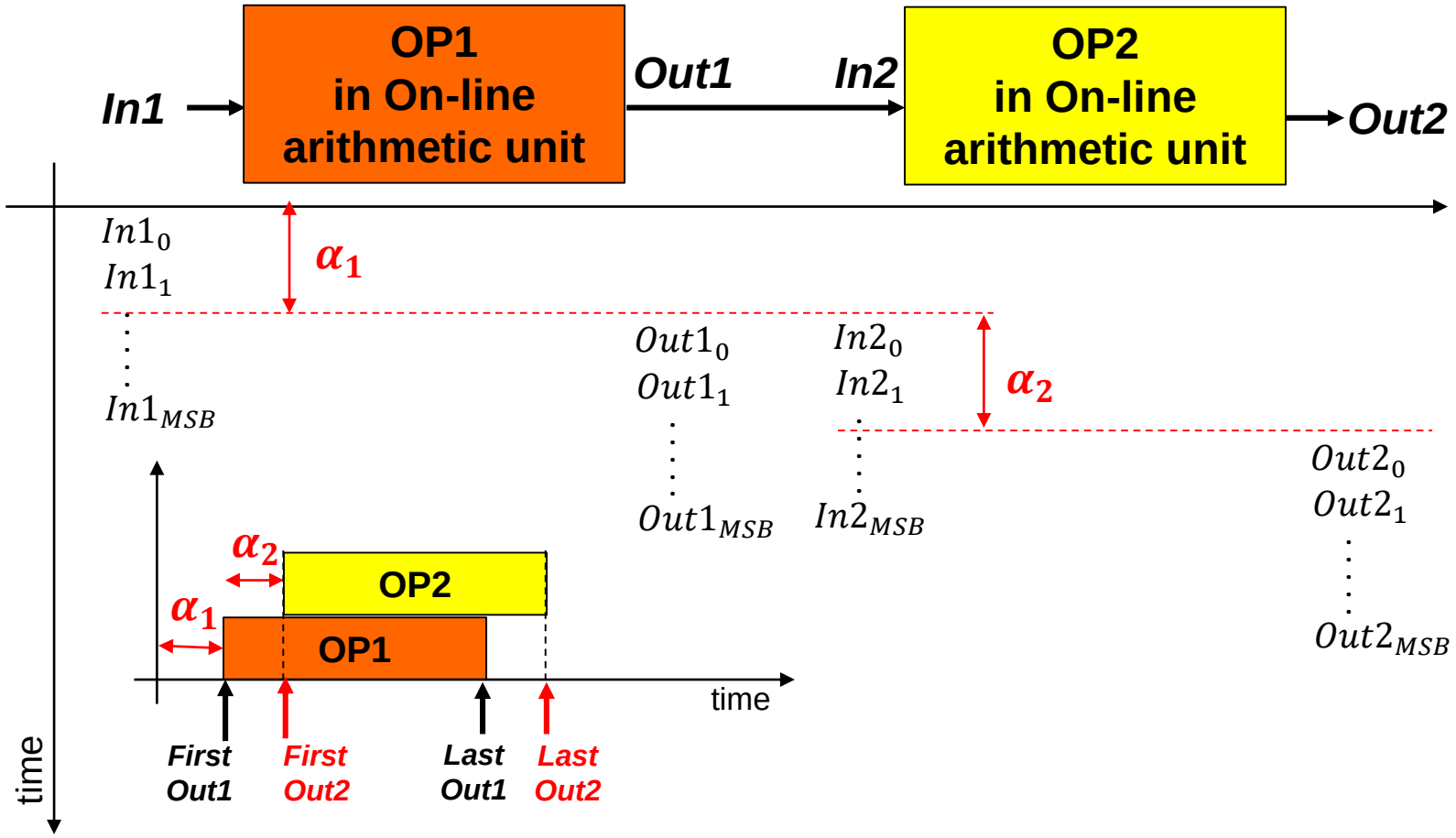
Conventional arithmetic operation

- All digits must be known.
- Compute in parallel and digit-serially.



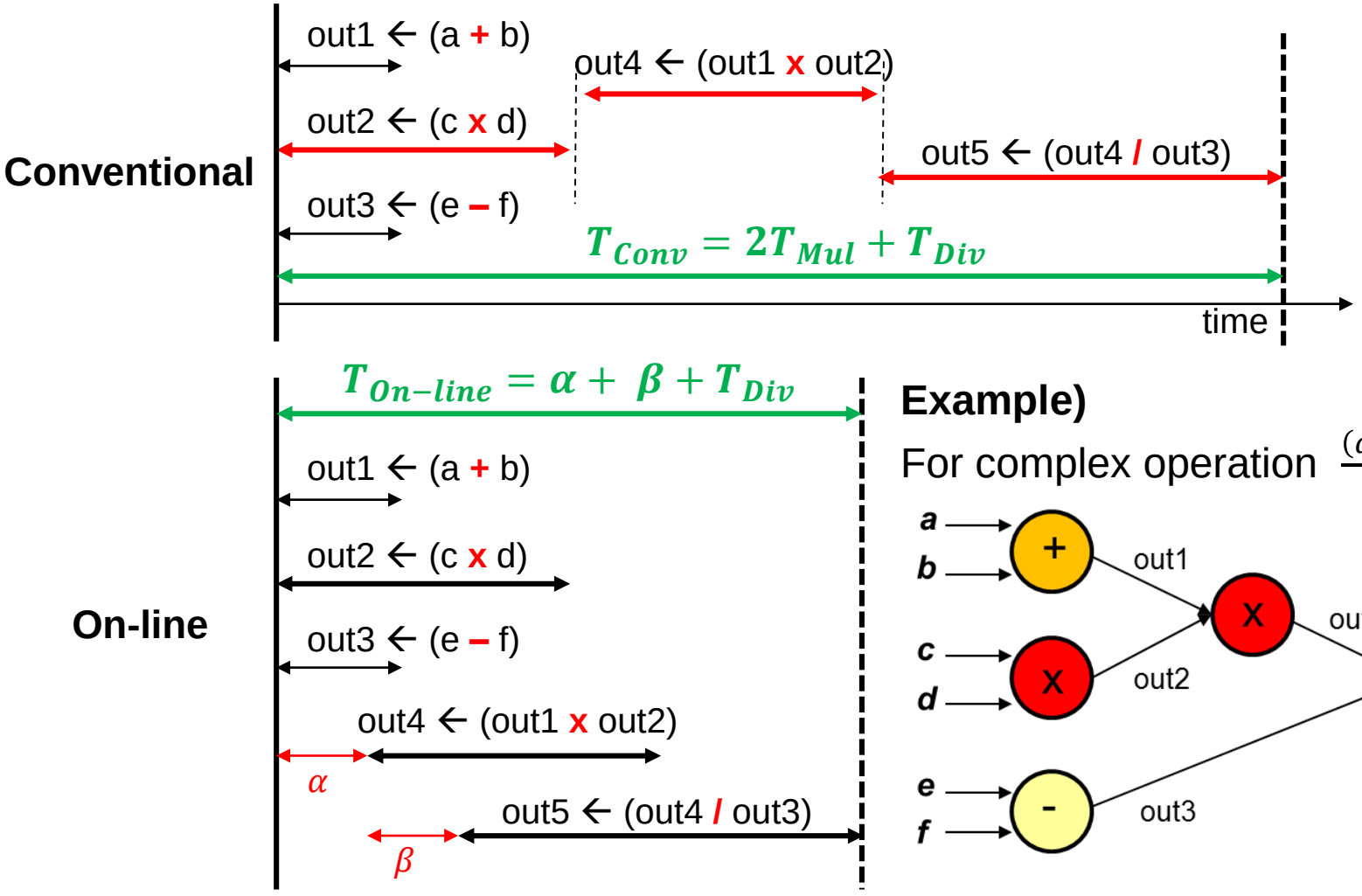
On-line arithmetic operation [1]

- Can process **partial** input.
  - So, it can be executed in **overlapped** manner.



[1] M. D. Ercegovac, "On-line arithmetic : An overview," in Real Time Signal Processing VIII,Proc. SPIE, vol. 495, pp.86-93

Conventional vs On-line arithmetic operation [1]



[1] M. D. Ercegovac, "On-line arithmetic : An overview," in Real Time Signal Processing VIII,Proc. SPIE, vol. 495, pp.86-93

Dependency distance

- Distance between the instruction under data dependency.
- Example1)

i1 : **R1** = A x B
i2 : R2 = C x **R1**

Dependency distance : 1
(= **D1** dependency)

- Example2)

i1 : **R1** = A x B
i2 : R2 = C x D
i3 : R3 = **R1** x **R1**

Dependency distance : 2
(= **D2** dependency)

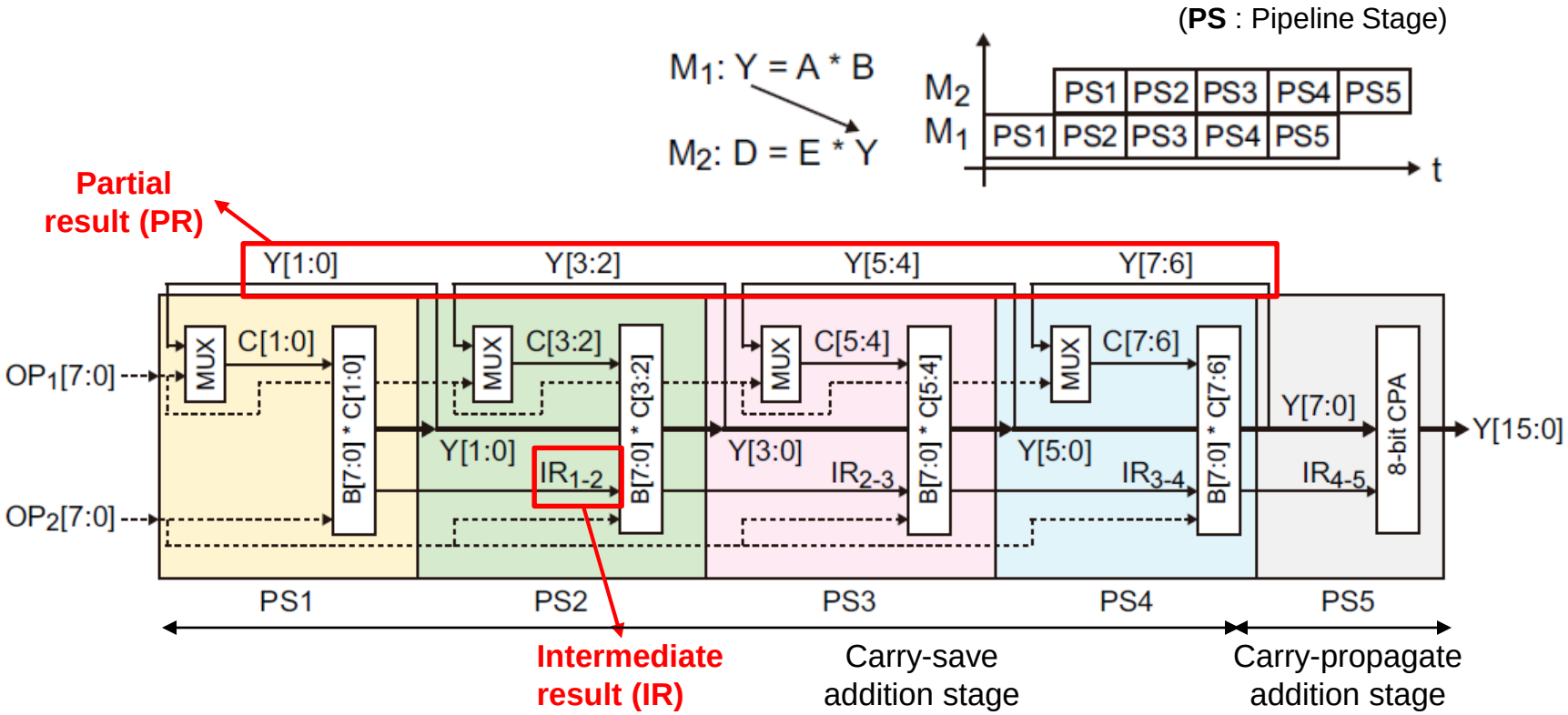
- Example3)

i1 : **R1** = A x B
i2 : **R2** = C x D
i3 : R3 = **R2** x **R1**

Dependency distance : 2
Dependency distance : 1

Intra-unit forwarding

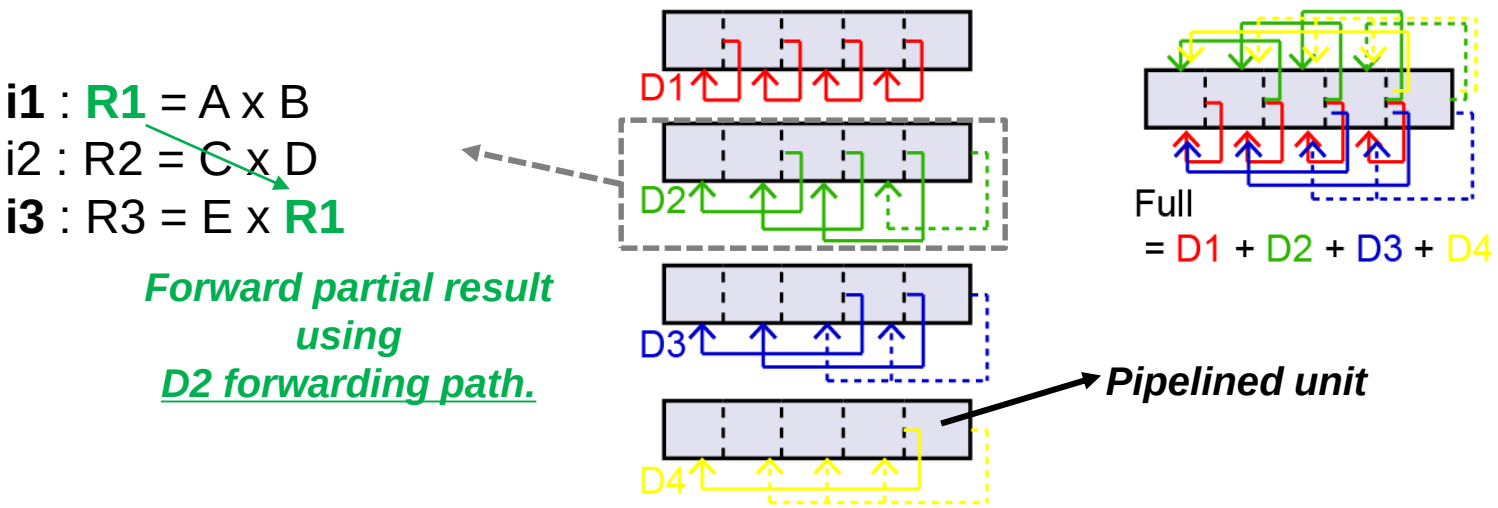
Example) When Dependency distance = 1
- 5-stage 8bit x 8bit multiplication.



Intra-unit forwarding

- Example) 5-stage unit.
  - D1 ~ D4 dependency can be considered.
  - **D1 ~ D4 forwarding path** can be added.

*** Forwarding path type :**

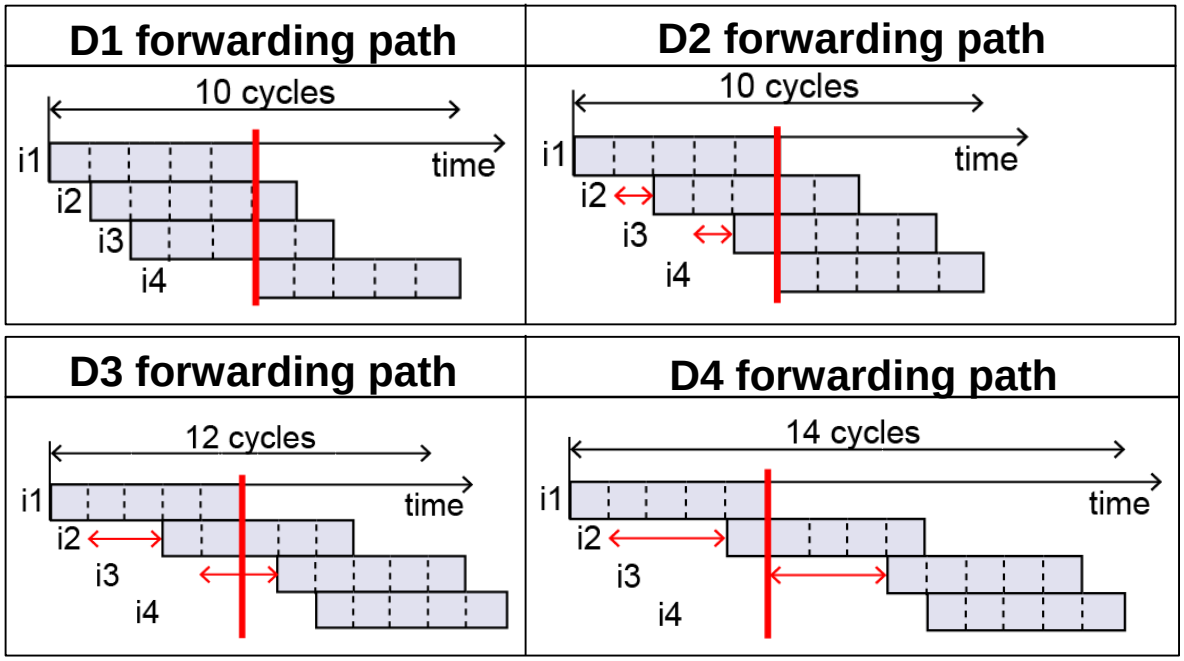


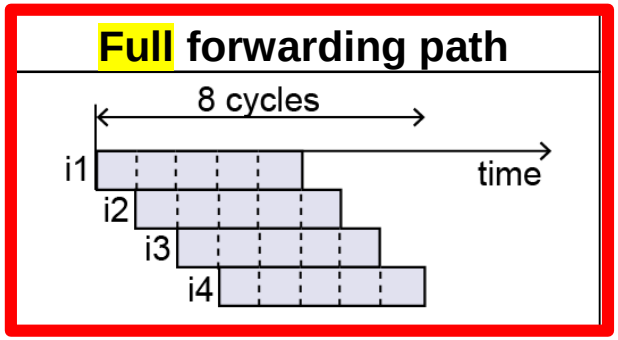
Intra-unit forwarding

- How about this case?

  i1 : R1 = A1 x B1
  i2 : R2 = A2 x R1 (**D1** dependency)
  i3 : R3 = A3 x R2 (**D1** dependency)
  i4 : R4 = A4 x R1 (**D3** dependency)

Suppose,
each stage takes **1 clock cycle**.





Dependency type

- There are three types of dependencies we consider.

For Y = OP1 x OP2		
Dependency	**OP1**	**OP2**
Type 01	Independent	Dependent
Type 10	Dependent	Independent
Type 11	Dependent	Dependent

Example)

Dependency Type :

Type 01

i1 : X = A x B

i2 : Y = C x X

Type 10

i1 : X = A x B

i2 : Y = X x C

Type 11

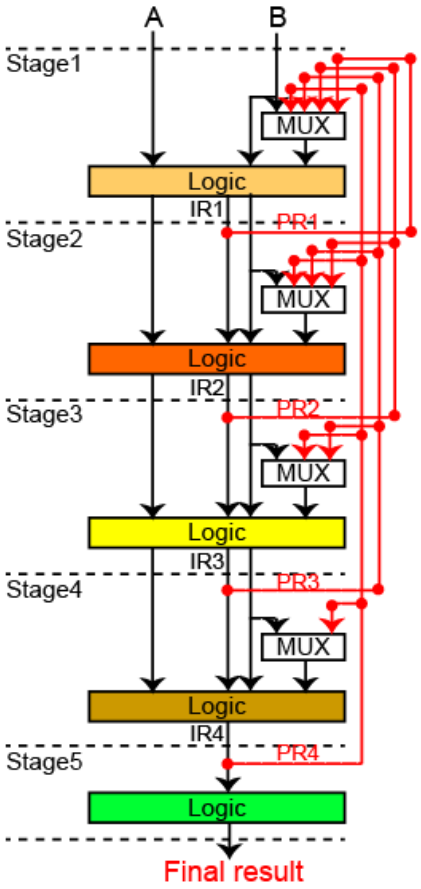
i1 : X = A x B

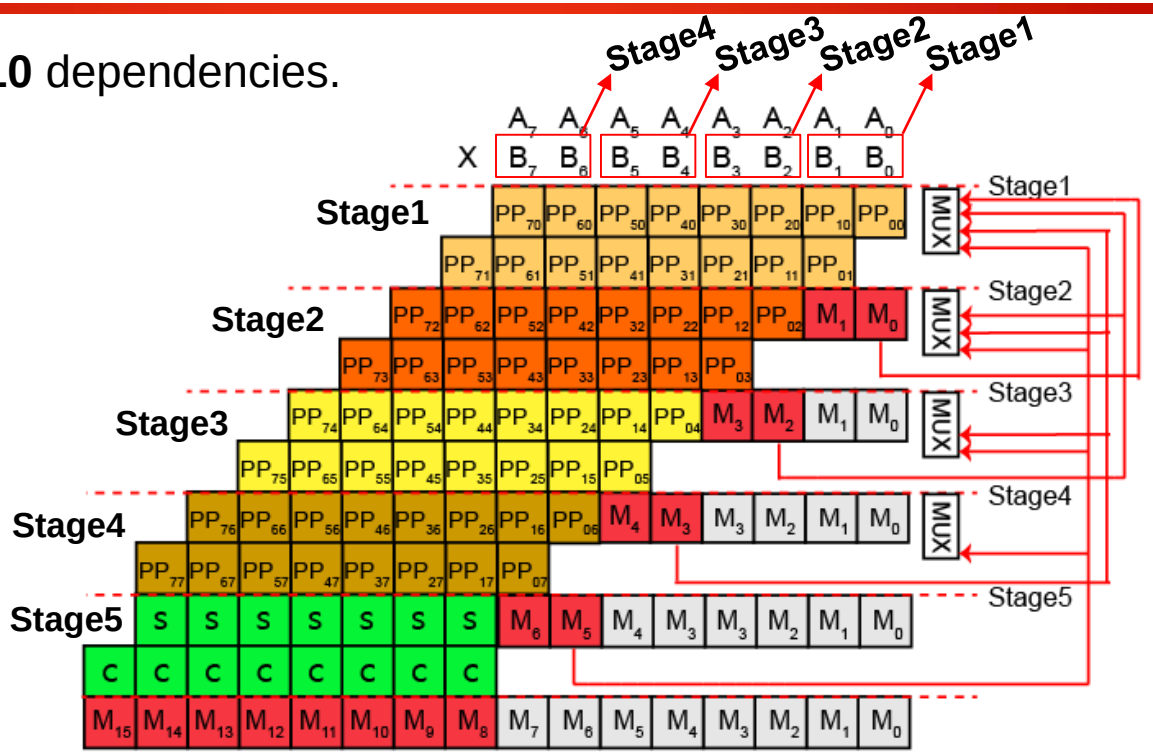
i2 : Y = X x C

i3 : Z = X x Y

High-throughput multiplier architectures _Arch1 (proposed)

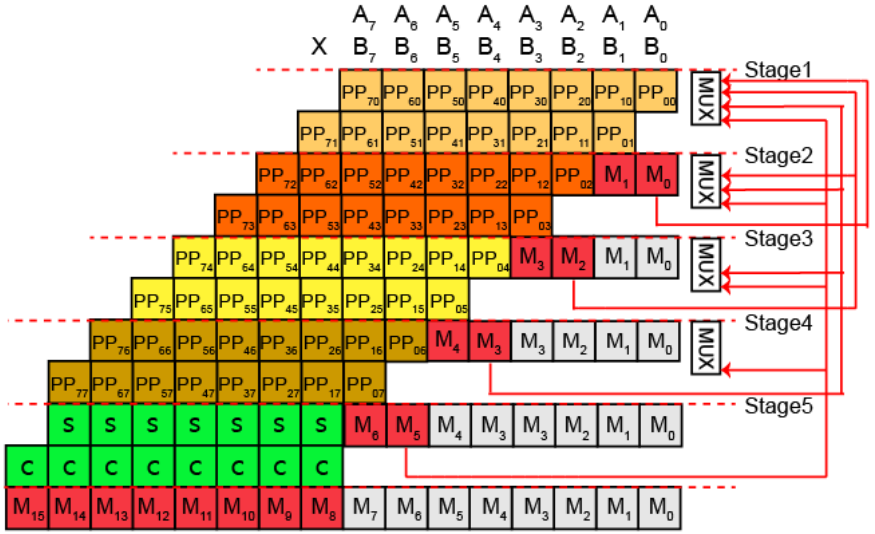
- Resolve **Type 01/10** dependencies.





Arch1 (proposed)_example

- **Example)** **i1** : X = A x B
  i2 : Y = C x X_{low}

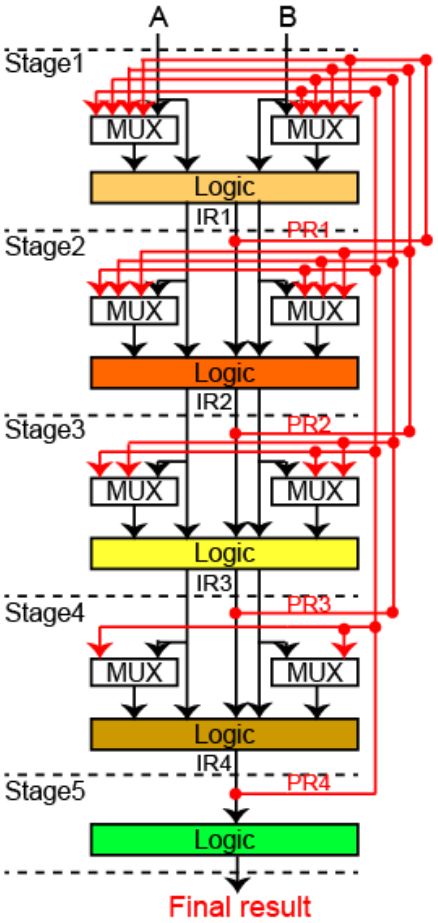


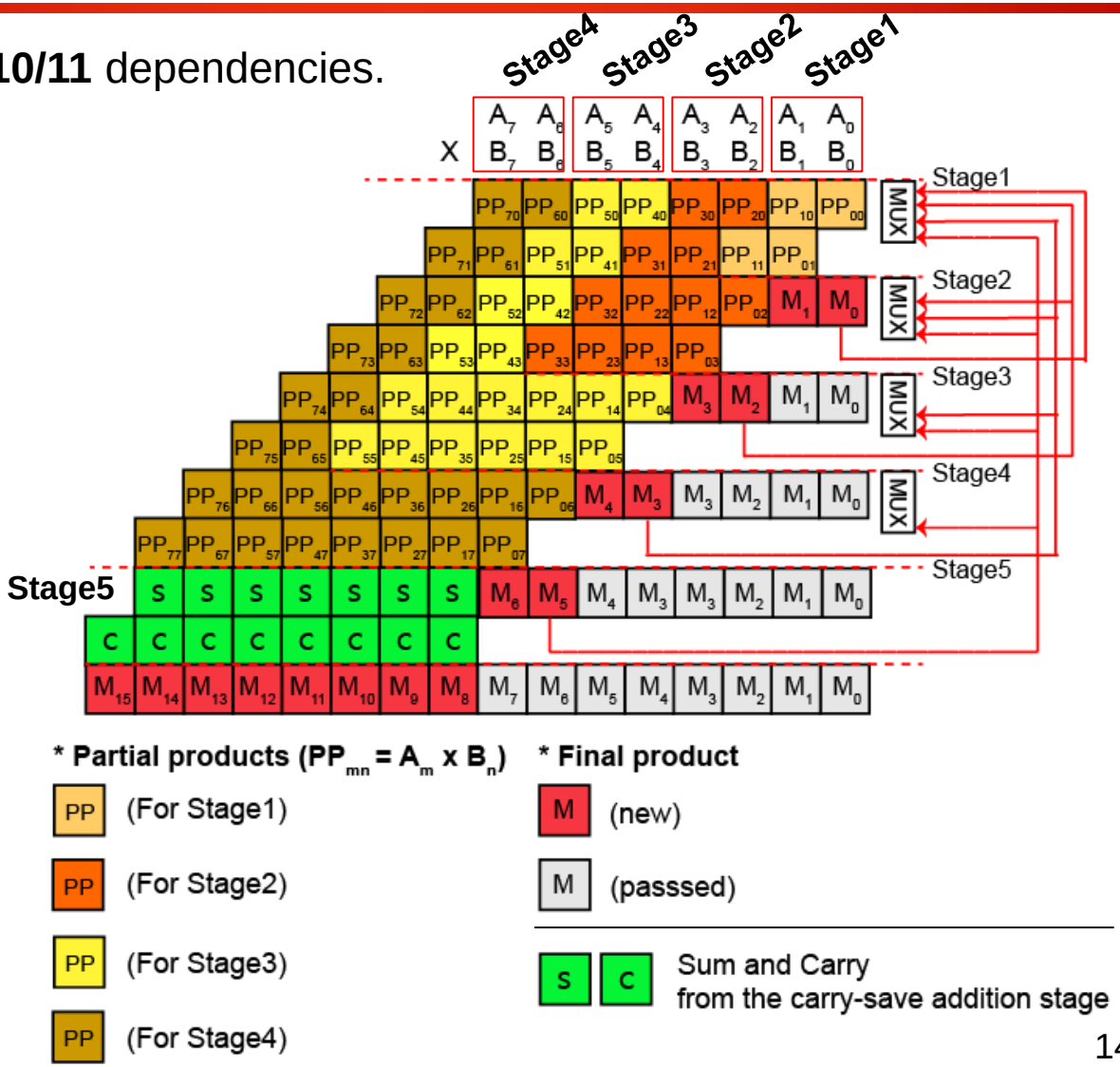
(***Clk*** : Clock cycle, ***ST*** : pipeline stage, ***Gen / Acc PR*** : Generated/Accumulated Partial Result)

	i1				i2			
Clk	**ST**	**Processed**	**Gen PR**	**Acc PR**	**ST**	**Processed**	**Gen PR**	**Acc PR**
1	1	A[7:0] x B[1:0]	**X[1:0]**	X[1:0]	-	-	-	-
2	2	A[7:0] x B[3:2]	**X[3:2]**	X[3:0]	1	C[7:0] x **X[1:0]**	Y[1:0]	Y[1:0]
3	3	A[7:0] x B[5:4]	**X[5:4]**	X[5:0]	2	C[7:0] x **X[3:2]**	Y[3:2]	Y[3:0]
4	4	A[7:0] x B[7:6]	**X[7:6]**	X[7:0]	3	C[7:0] x **X[5:4]**	Y[5:4]	Y[5:0]
5	5	Sum + Carry row	X[15:8]	X[15:0]	4	C[7:0] x **X[7:6]**	Y[7:6]	Y[7:0]
6					5	Sum + Carry row	Y[15:8]	Y[15:0]

High-throughput multiplier architectures _Arch2 (proposed)

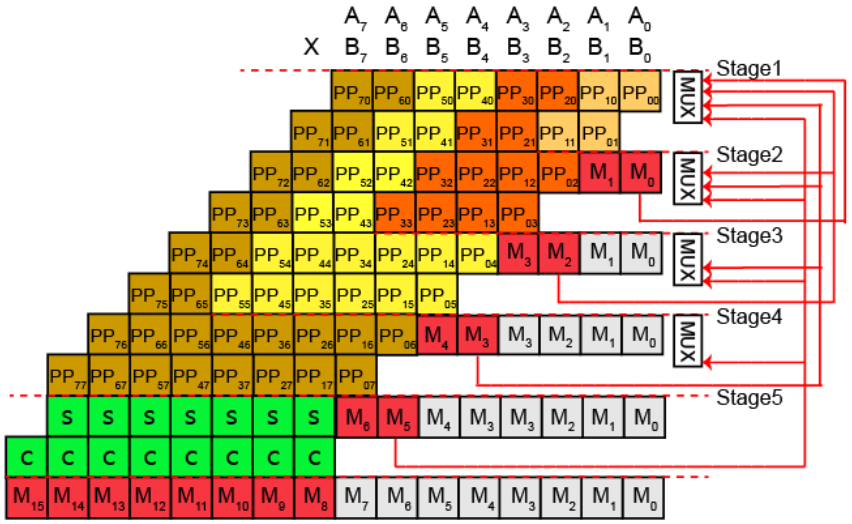
- Resolve **Type 01/10/11** dependencies.





Arch2 (proposed)_example

- **Example)** **i1** : X = A x B
  i2 : Y = C x D
  i3 : Z = X_{low} x Y_{low}

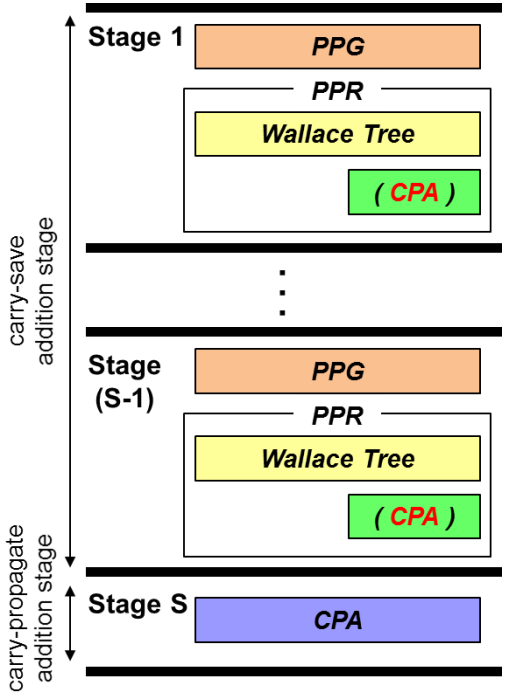


(***Clk*** : Clock cycle, ***ST*** : pipeline stage,
Gen / Acc PR : Generated/Accumulated Partial Result)

	i1				i2				i3			
Clk	**ST**	**Processed**	**Gen PR**	**Acc PR**	**ST**	**Processed**	**Gen PR**	**Acc PR**	**ST**	**Processed**	**Gen PR**	**Acc PR**
1	1	**A[1:0] x B[1:0]**	**X[1:0]**	X[1:0]	-	-	-	-	-	-	-	-
2	2	**A[3:2]** x B[1:0] **B[3:2]** x A[1:0]	**X[3:2]**	X[3:0]	1	**C[1:0] x D[1:0]**	**Y[1:0]**	Y[1:0]	-	-	-	-
3	3	**A[5:4]** x B[3:0] **B[5:4]** x A[3:0]	**X[5:4]**	X[5:0]	2	**C[3:2]** x D[1:0] **D[3:2]** x C[1:0]	**Y[3:2]**	Y[3:0]	1	**X[1:0]** x **Y[1:0]**	Z[1:0]	Z[1:0]
4	4	**A[7:6]** x B[5:0] **B[7:6]** x A[5:0]	**X[7:6]**	X[7:0]	3	**C[5:4]** x D[3:0] **D[5:4]** x C[3:0]	**Y[5:4]**	Y[5:0]	2	**X[3:2]** x Y[1:0] **Y[3:2]** x X[1:0]	Z[3:2]	Z[3:0]
5	5	Sum + Carry row	X[15:8]	X[15:0]	4	**C[7:6]** x D[5:0] **D[7:6]** x C[5:0]	**Y[7:6]**	Y[7:0]	3	**X[5:4]** x Y[3:0] **Y[5:4]** x X[3:0]	Z[5:4]	Z[5:0]
6					5	Sum + Carry row	Y[15:8]	Y[15:0]	4	**X[7:6]** x Y[5:0] **Y[7:6]** x X[5:0]	Z[7:6]	Z[7:0]
7									5	Sum + Carry row	Z[15:8]	Z[15:0]

Hardware implementation

For **S**-stage **N**-bit x **N**-bit multiplication



Stage type	Step		NBBE-2	RBBE-4	CRBBE-4
			Normal Binary Based	Redundant Binary Based	
carry save addition stage : (1 ~ (S-1))	PPG		- Sign extension technique [1]		
			- Radix-4 Booth encoding [2,3]	- Radix-16 Booth encoding 1 [4,5]	- Radix-16 Booth encoding 2 [6]
	PPR	Wallace Tree	- by FA / HA	- by Carry-free adder1 [4,5]	-by Carry-free adder2 [6]
	PPR	(CPA) (Arch1/2)	KSA (Kogge-Stone Adder) [7]		
carry propagate addition stage : S	CPA		KSA (Kogge-Stone Adder) [7]		

PPG : Partial Product Generation,
PPR : Partial Product Reduction
CPA : Carry-Propagate addition

NBBE-2 : Radix-4 Normal Binary based Booth encoded multiplier
RBBE-4 : Radix-16 Redundant Binary based Booth encoded multiplier
CRBBE-4 : Radix-16 Covalent Radix-16 based Booth encoded multiplier

[1] D. P. Agrawal and T. R. N. Rao, "On Multiple Operand Addition of signed Binary Numbers," in IEEE Trans. on Computers, vol. c27, no. 11, Nov. 1978, pp. 1068 – 1070.
[2] A. D. Booth, "A Signed Binary Multiplication Technique" in The Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 3, Jan. 1951, pp. 236 – 240.
[3] X. Cui, W. Liu, X. Chen, Earl E. Swartzlander Jr., and F. Lombardi, "A Modified Partial Product Generator for Redundant Binary Multipliers," in IEEE Trans. on Computers, vol. 65, no. 4, Apr. 2016, pp 1165 – 1171.
[4] H. Makino, Y. Nakase, H. Suzuki, H. Morinaka, H. Shinohara et al., "An 8.8-ns 54x54-Bit Multiplier with High Speed Redundant Binary Architecture," in IEEE Journal of Solid State Circuits, vol. 31, no. 6, 1996, pp 773-783.
[5] N. Besli and R. G. Deshmukh, "A Novel redundant Binary Signed-Digit(RBSD) Booth's Encoding," in Proc. IEEE SoutheastConf, Apr. 2002, pp 426 – 431.
[6] Y. He and C.-H. Chang, "a New Redundant Binary Booth Encoding for Fast 2^n-Bit Multiplier Design," in IEEE Trans. on Circuits and Systems, vol. 56, no. 6, 2009, pp. 1192 – 1201.
[7] P. M. Kogge and H. S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," in IEEE Trans. on Computers, vol. C-22, no. 8, Aug. 1973, pp. 786 – 793.

Simulation setting

- **2 / 3 / 5** stages **32 / 64** bit signed integer multiplier architectures.

- Implementation:
  - VHDL

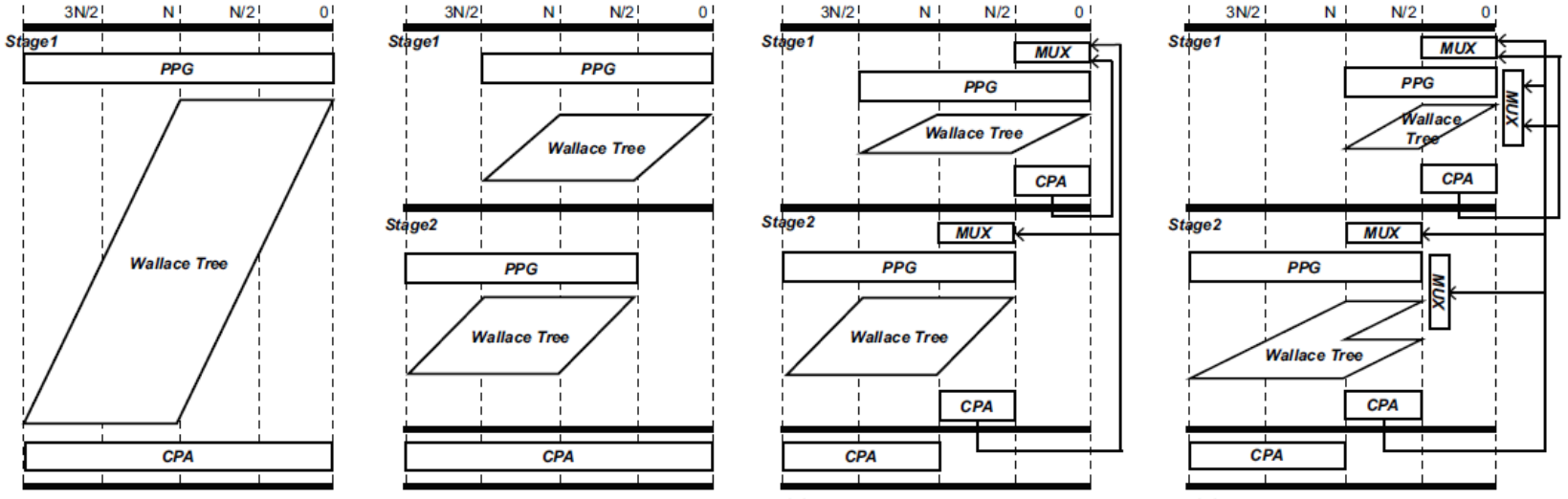
- Synthesis:
  - Synopsys Design Compiler
  - Nangate 45nm Open Cell Library

- Execution time simulation:
  - C/C++

- Metrics:
  - Clock period
  - Area
  - Power consumption
  - Execution time

Simulation setting

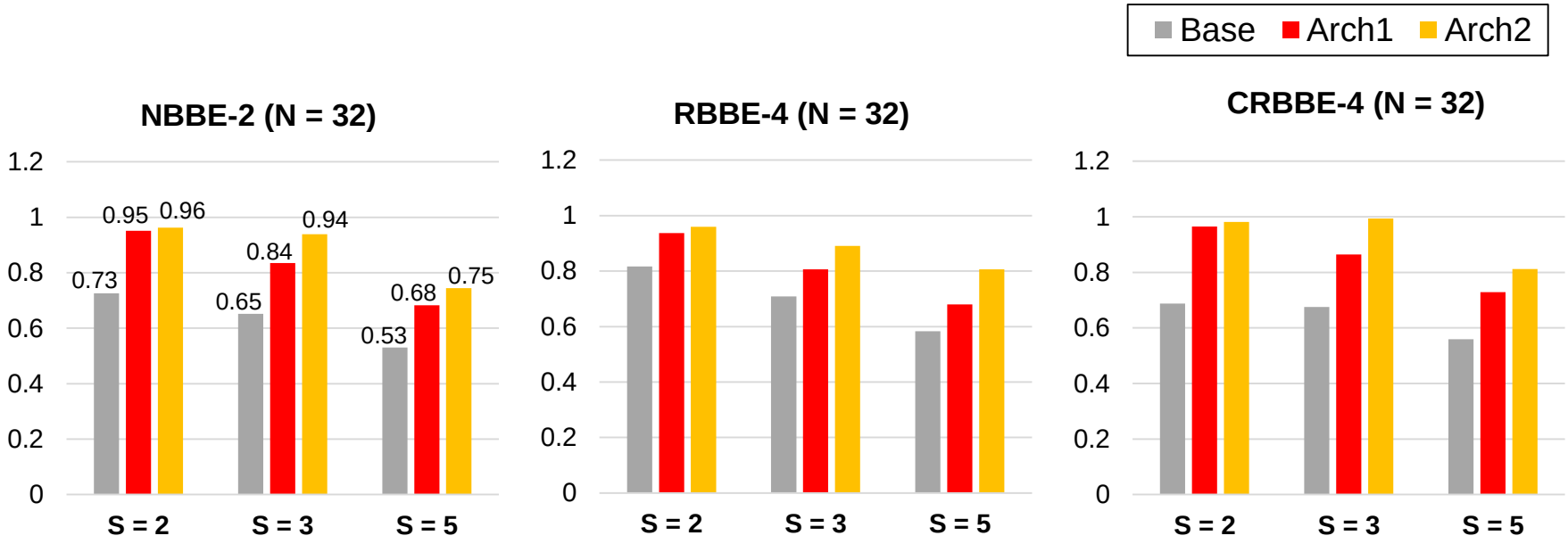
- Compare four architectures for each multiplier (NBBE-2/ RBBE-4/ CRBBE-4).
  - ***N-P*** : Non-Pipelined multiplier architecture.
  - ***Base*** : Pipelined architecture without intra-unit forwarding paths.
  - ***Arch1*** : Pipelined architecture with intra-unit forwarding paths.
    Type 01/10 dependencies can be resolved.
  - ***Arch2*** : Pipelined architecture with intra-unit forwarding paths.
    Type 01/10/**11** dependencies can be resolved.



N-P ***Base*** ***Arch1 (Proposed)*** ***Arch2 (Proposed)***

Clock period

- Base, Arch1, and Arch2 are **scaled to N-P.**



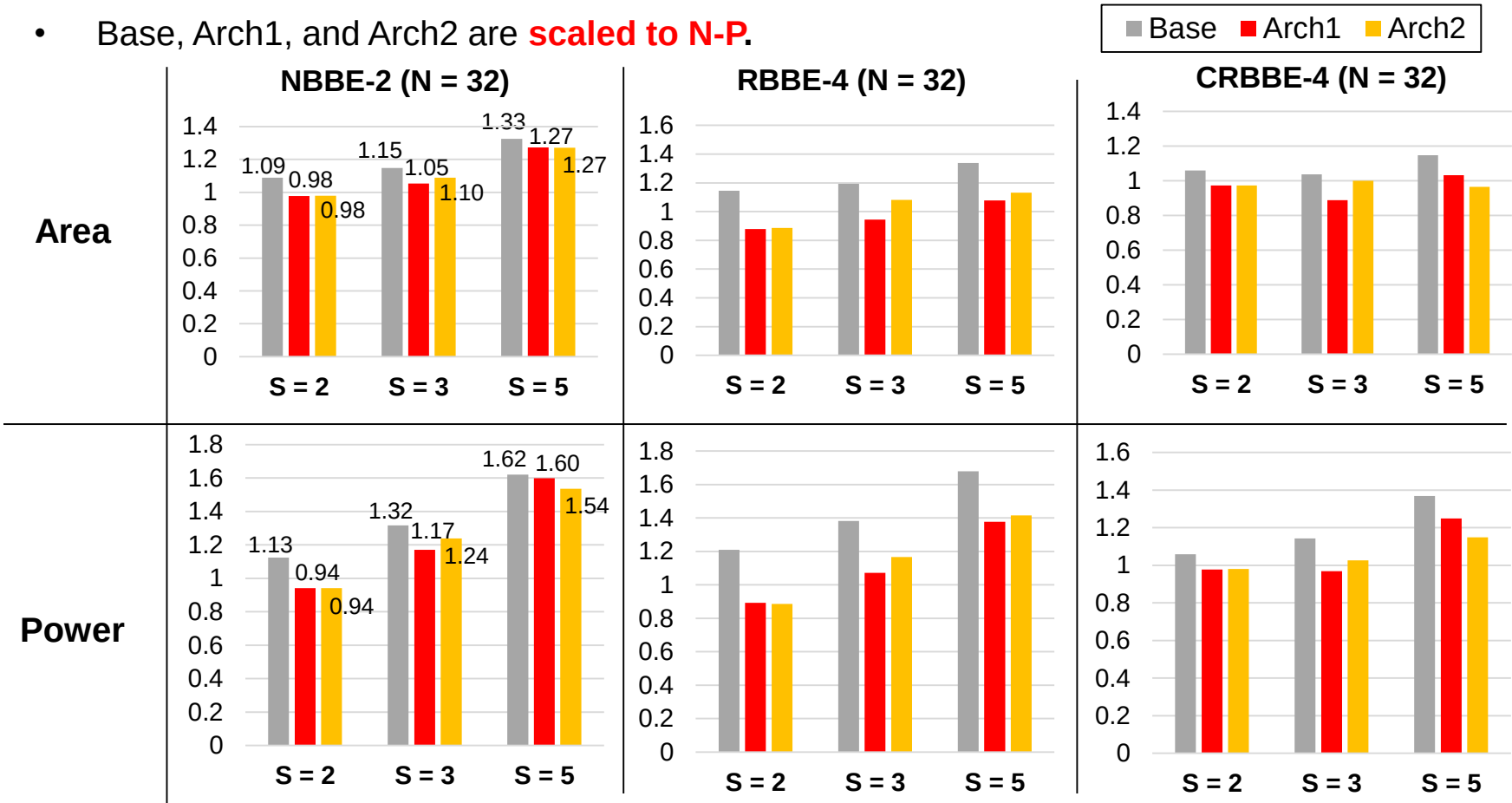
***Comparison metrics**

- **N : # operand bits**
- **S : total #stages**
- **C.S : Carry-save addition stage**
- **C.P : Carry-propagate addition stage**

	#MAX(partial product rows) in C.S	CPA in C.S	MUX
Base	$\frac{N}{S-1}$	X	X
Arch1	$\frac{N}{S-1}$	O	O
Arch2	$\frac{2N}{S-1}$	O	O

Area / Power

- Base, Arch1, and Arch2 are **scaled to N-P.**



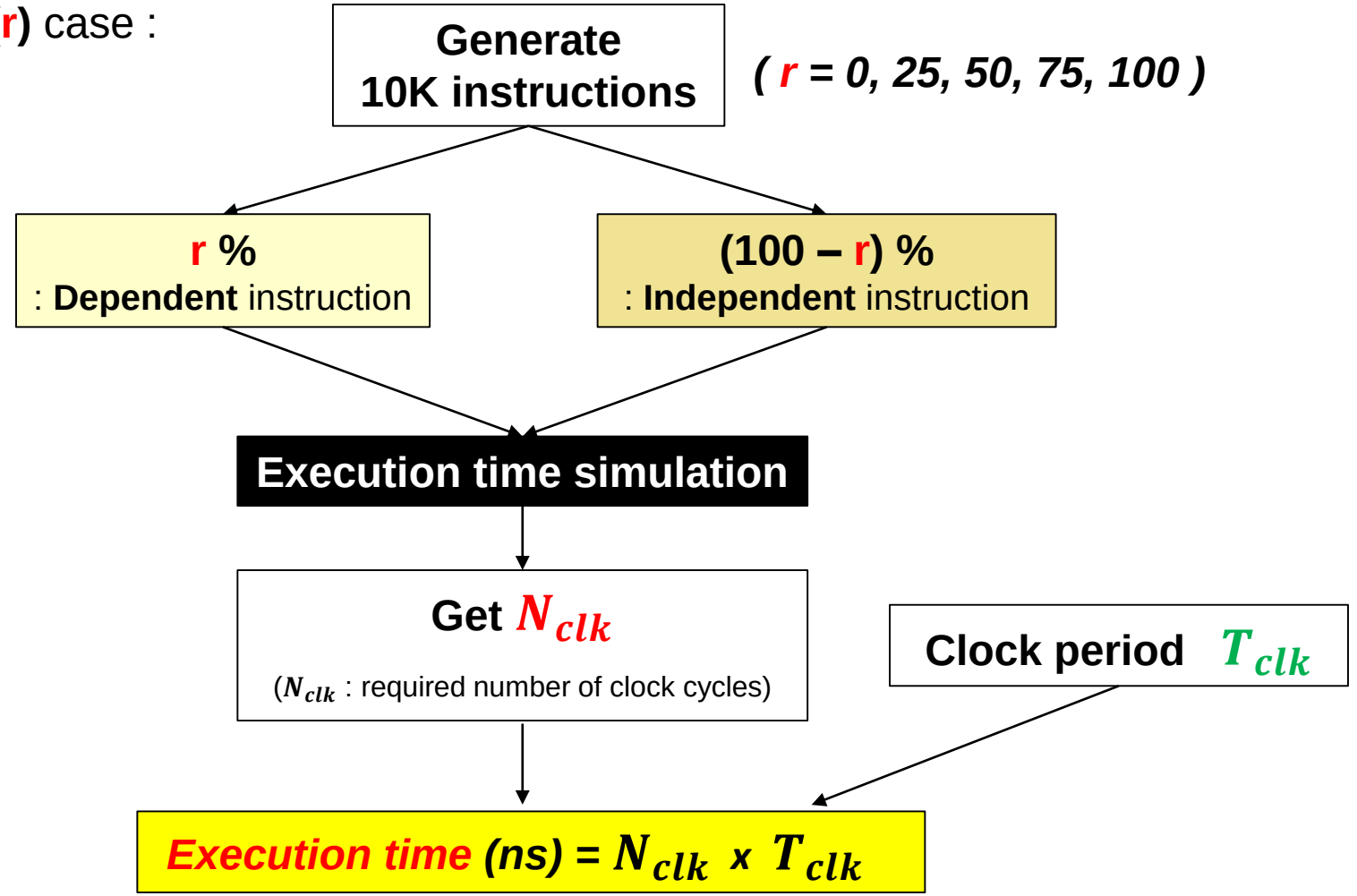
***Comparison metrics**

- **N : # operand bits**
- **S : total #stages**
- **C.S : Carry-save addition stage**
- **C.P : Carry-propagate addition stage**

	#FF	CPA in C.S	CPA in C.P	MUX
Base	↑	X	wide	X
Arch1	↓	O	narrow	O
Arch2	↓	O	narrow	O

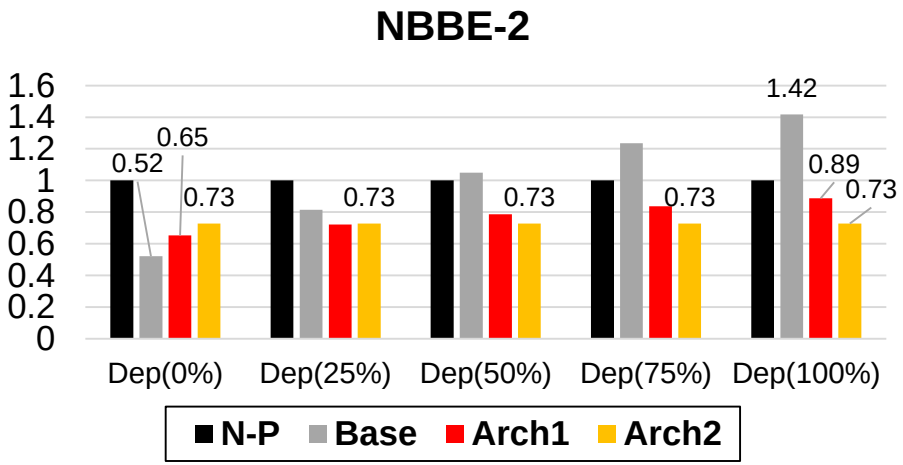
Execution time

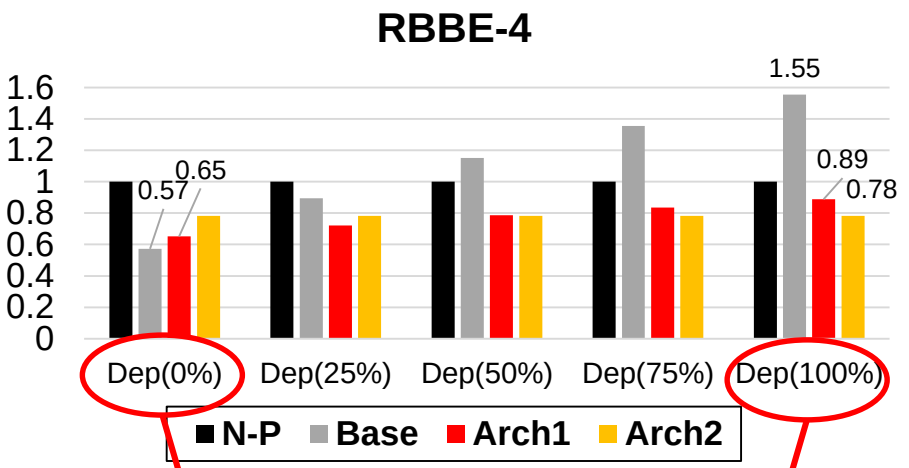
For **Dep(r)** case :

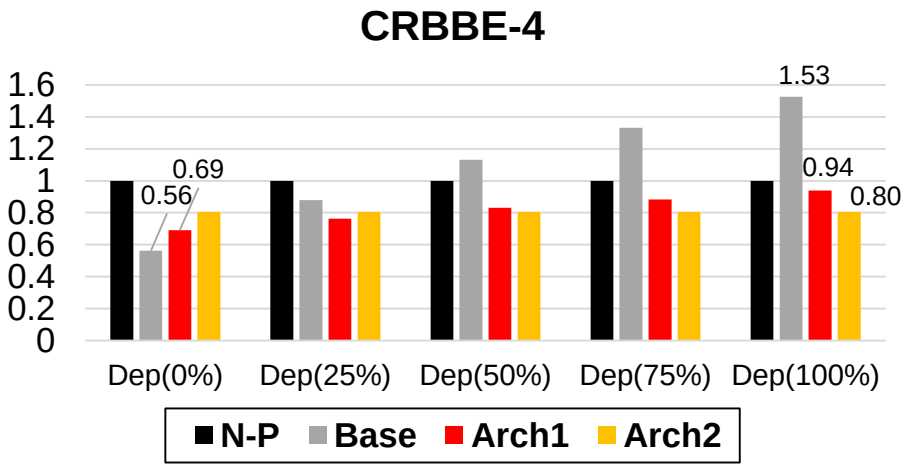


Execution time

- Measured for 5stage 64bit multiplier architectures (**scaled to N-P**).







All instructions have no dependency

All instructions have dependency

Conclusion

- The performance of arithmetic units has a great impact on the throughput of processing unit.
- Since there is certain-operation dominated situation and multiplication is heavily used operation, we focus on improving throughput in integer multiplication.

- **Our main work** is :
- 1. propose high-throughput multiplier architectures(**Arch1 & 2**) by
- inserting fast-forwarding path to intra-unit pipelined architecture.
- 2. show details of hardware implementation.
- 3. We also apply proposed architectures to existing multipliers
- (NBBE-2, RBBE-4, CRBBE-4).

- The simulation results show that, compared to N-P, Arch1 and Arch2 achieve **6~35%** and **20~27%** execution time reduction with small area and power overhead.

Thank you!

&

Question? ☺