

Flexible PIM  
Die Photo  
2024

# Compute-In-Memory Design For AI

Bonan Yan  
Peking University  
June 16, 2025



北京大学 人工智能  
研究院  
INSTITUTE FOR ARTIFICIAL INTELLIGENCE, PEKING UNIVERSITY

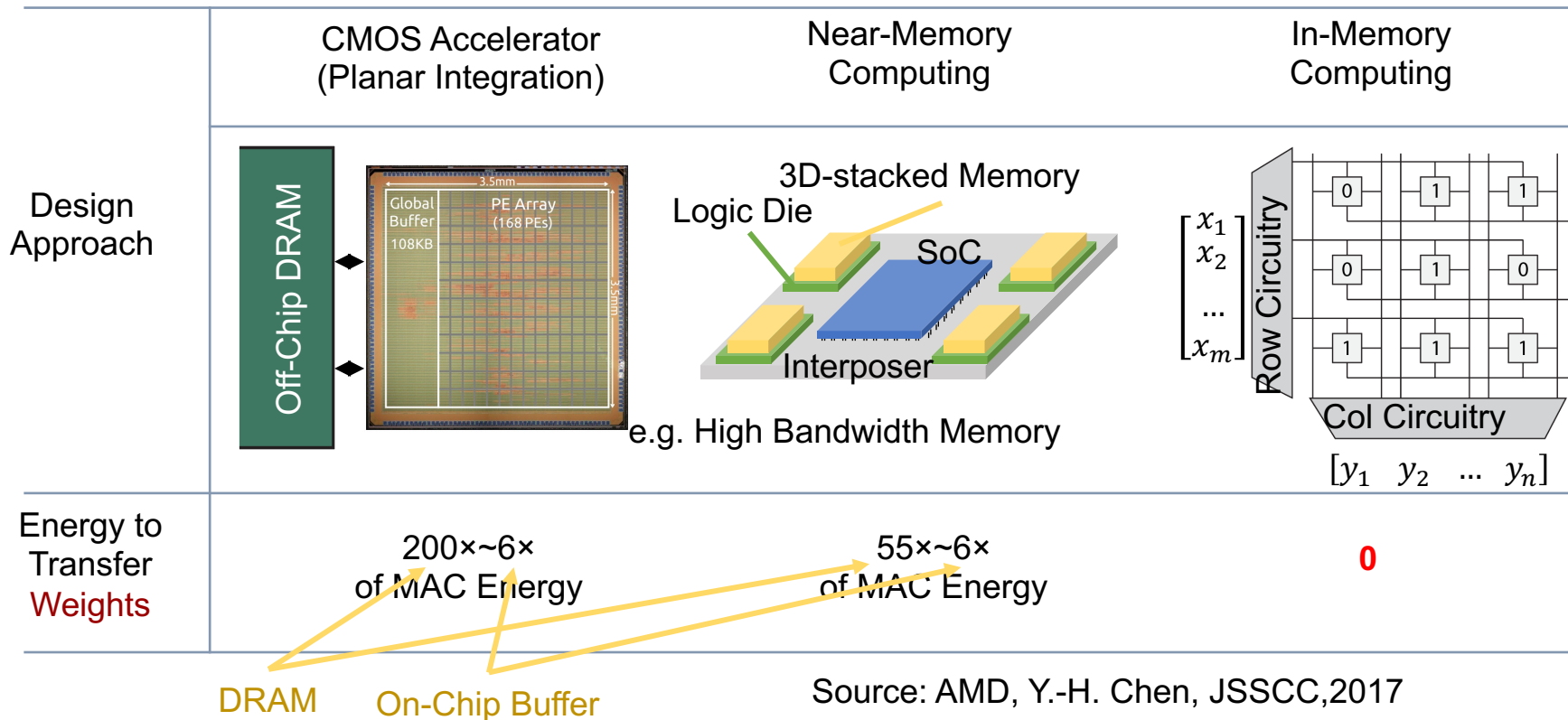
AGI-CHIP



# Outline

- Introduction of The Focused Paper
  - A 1.041Mb/mm<sup>2</sup> 27.38TOPS/W Signed-INT8 Dynamic-Logic-Based ADC-Less SRAM Compute-In-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications [ISSCC'2022]
- Our Subsequent Related Works
  - SRAM CIM [TCAS-I'2024, HPCA'2025]
  - RRAM CIM [Nature Electronics'2024]
  - Near-Memory Accelerator for Reinforcement Learning [DATE'2025]

# Motivation



Source: AMD, Y.-H. Chen, JSSCC, 2017

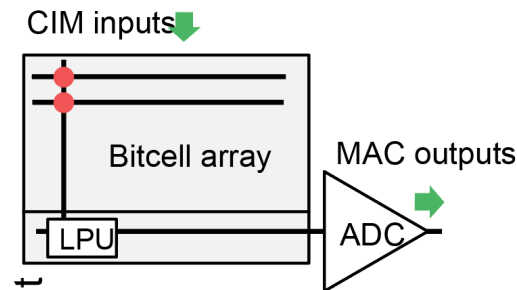
# Challenges

## Challenge 1: Large Compute Circuit Area

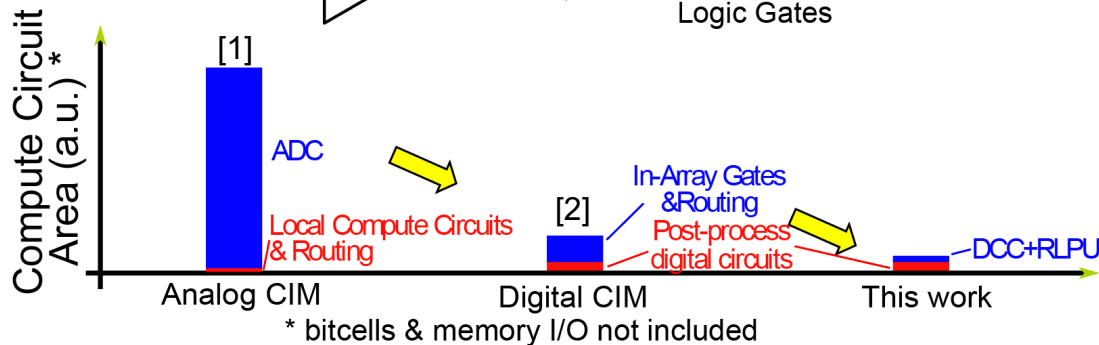
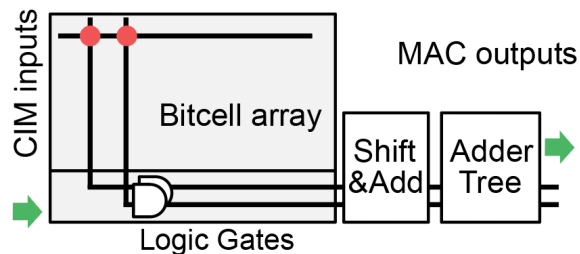
## Challenge 2: Limited CIM Function

## Challenge 3: CIM to DNN Deploy

LPU: Local Process Units



[1] J. -W. Su, et al., ISSCC'2021  
[2] Y. -D. Chi, et al., ISSCC'2021



**Need Smaller Compute Circuits & CIM Architecture**

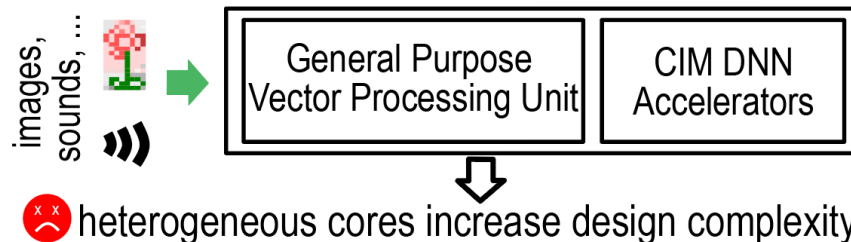
# Challenges

**Challenge 1: Large  
Compute Circuit Area**

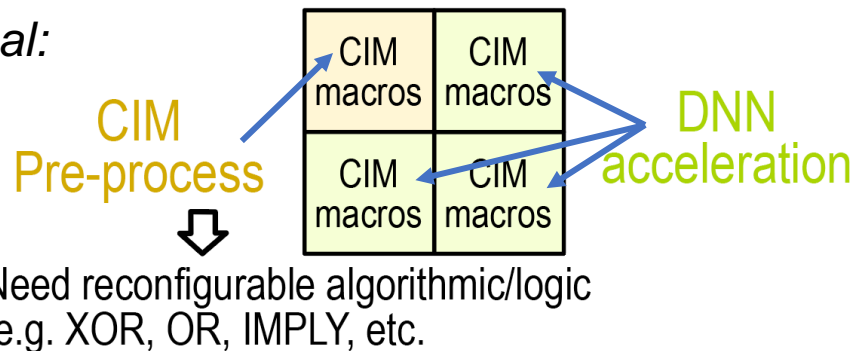
**Challenge 2: Limited  
CIM Function**

**Challenge 3: CIM to  
DNN Deploy**

*Conventional:*



*Ideal:*



# Challenges

## Challenge 1: Large Compute Circuit Area

## Challenge 2: Limited CIM Function

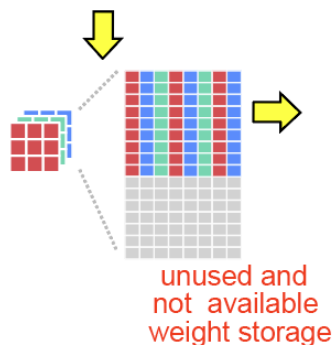
## Challenge 3: CIM to DNN Deploy

DNN kernels in commonly used architectures:

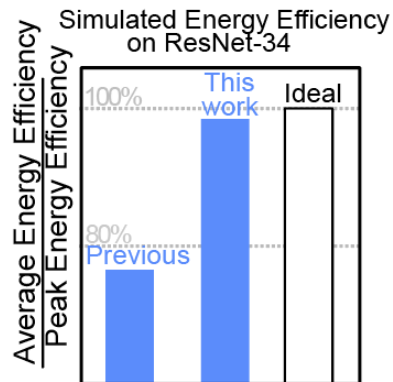
VGG-19	ResNet-34
3×3 conv, 64 ×2	conv2_x: 3×3 conv, 64 ×6
3×3 conv, 128 ×2	conv3_x: 3×3 conv, 128 ×8
3×3 conv, 256 ×4	conv4_x: 3×3 conv, 256 ×12
3×3 conv, 512 ×4	conv5_x: 3×3 conv, 512 ×6
fc 4096 ×2	fc 1000 ×1
fc 1000 ×1	

3×3 conv is ubiquitous

Direct mapping has low utilization



🔴 low weight spatial utilization rate cause low average energy efficiency

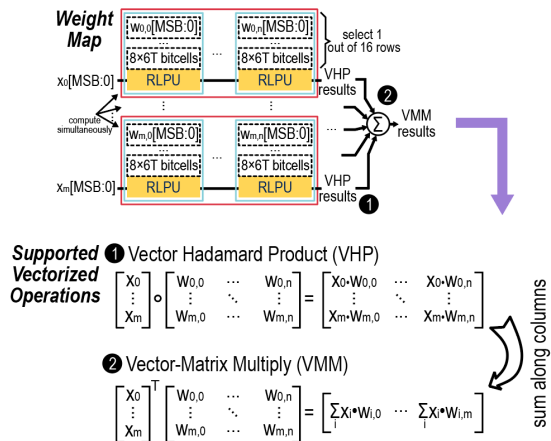


# Key Contributions

## Challenge 1: Large compute circuit area



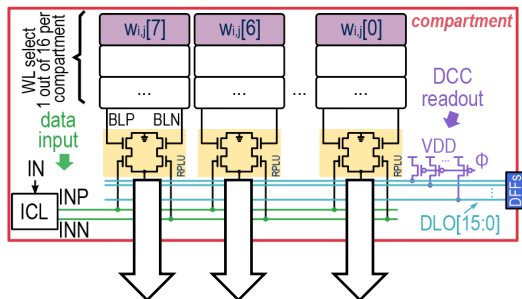
### ADC-Less Vector-Hadamard-Product (VHP) Architecture



## Challenge 2: Limited CIM function



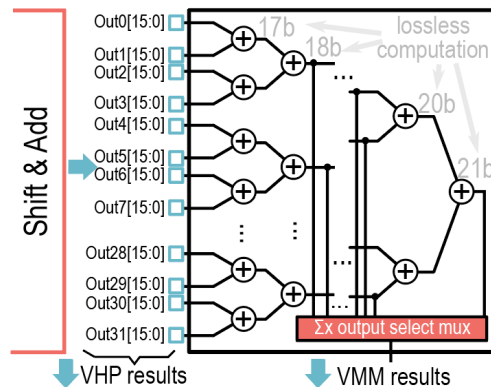
### Reconfigurable Local Process Units (RLPU) + Dynamic Logic Compute Circuits (DCC)



## Challenge 3: CIM to DNN deploy

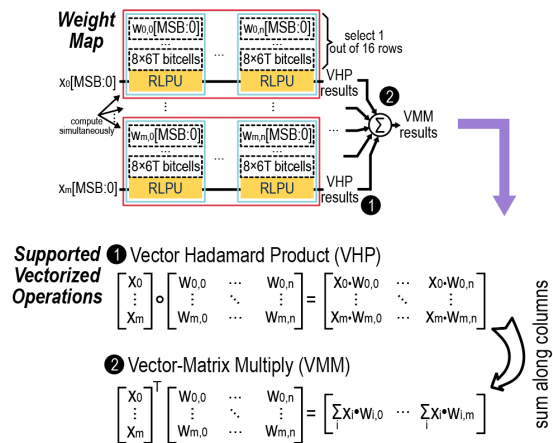


### Flexible VHP/VMM Support With Post Sum Circuits

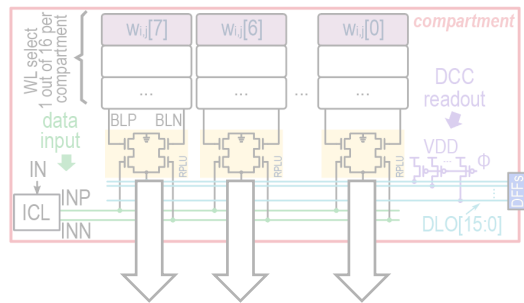


# Key Contributions

## ADC-Less Vector-Hadamard-Product (VHP) Architecture

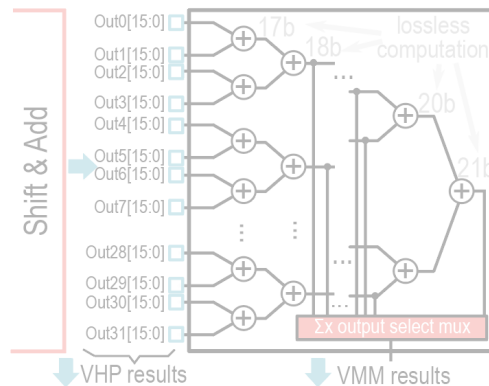


## Reconfigurable Local Process Units (RLPU) + Dynamic Logic Compute Circuits (DCC)



Reconfigurable to  
AND OR XOR

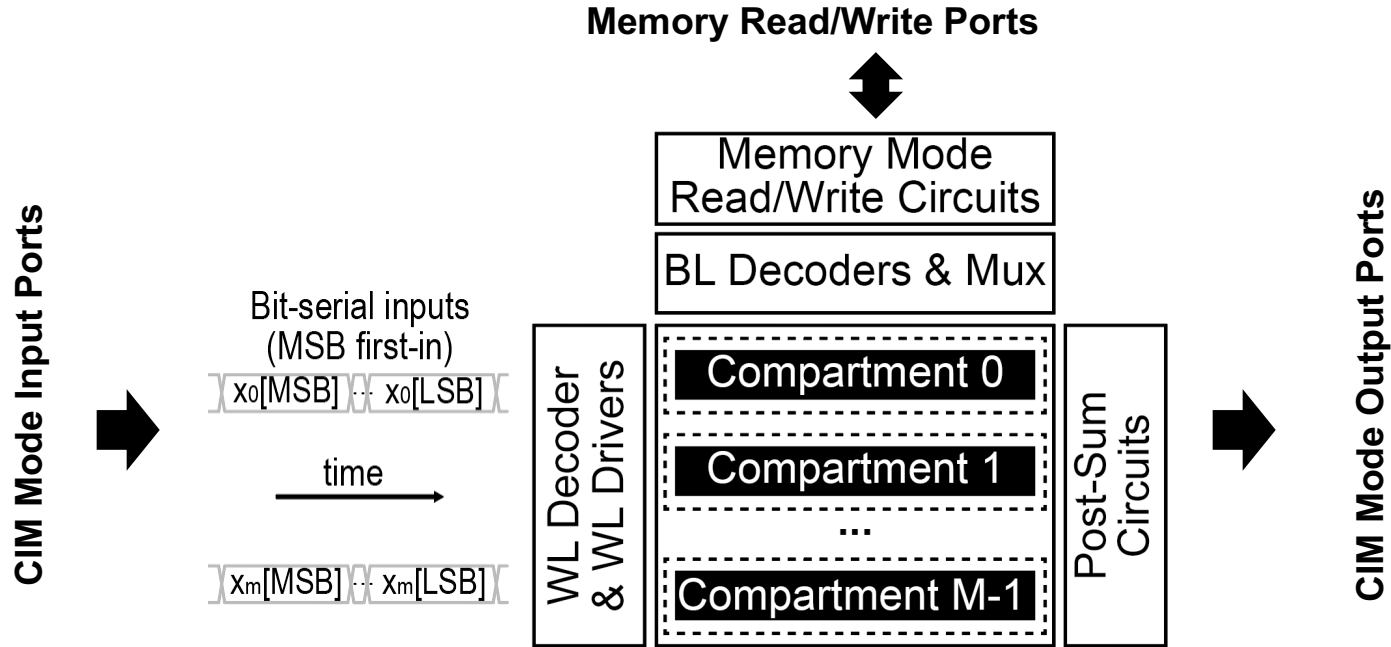
## Flexible VHP/VMM Support With Post Sum Circuits





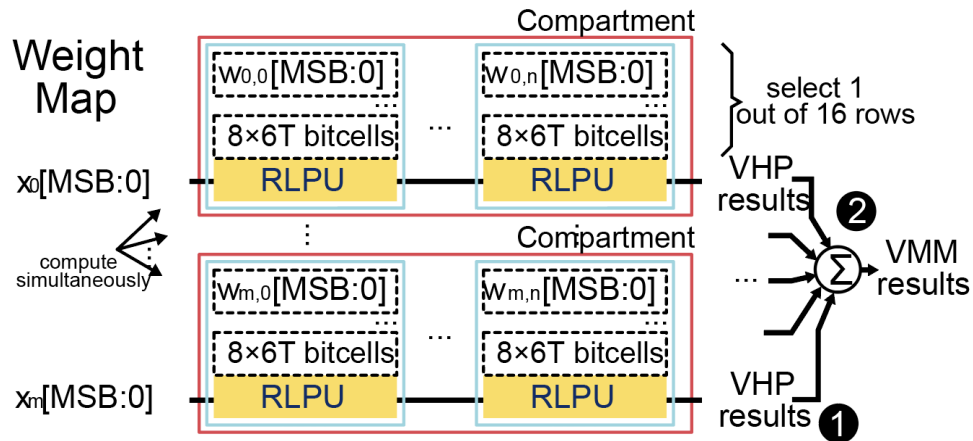
# Compartment-Based Macro Organization

- Separate ports to support Vector Hadamard Product (VHP) operations:



# Macro VHP Architecture

**Compartment-based vector element selection**



**Effective vector-matrix multiplication (VMM)**

① Vector Hadamard Product (VHP)

$$\begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix} \circ \begin{bmatrix} w_{0,0} & \dots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{m,0} & \dots & w_{m,n} \end{bmatrix} = \begin{bmatrix} x_0 \cdot w_{0,0} & \dots & x_0 \cdot w_{0,n} \\ \vdots & \ddots & \vdots \\ x_m \cdot w_{m,0} & \dots & x_m \cdot w_{m,n} \end{bmatrix}$$

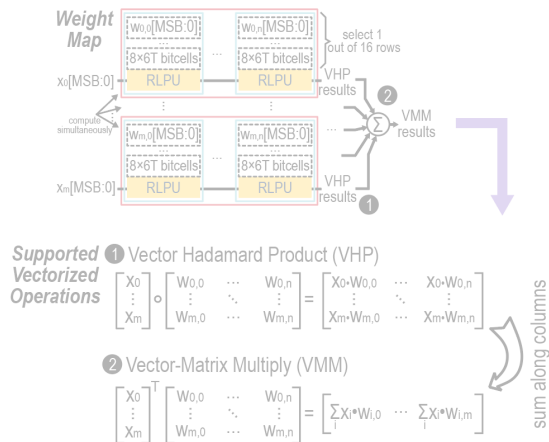
② Vector-Matrix Multiply (VMM)

$$\begin{bmatrix} x_0 \\ \vdots \\ x_m \end{bmatrix}^T \begin{bmatrix} w_{0,0} & \dots & w_{0,n} \\ \vdots & \ddots & \vdots \\ w_{m,0} & \dots & w_{m,n} \end{bmatrix} = \begin{bmatrix} \sum_i x_i \cdot w_{i,0} & \dots & \sum_i x_i \cdot w_{i,n} \end{bmatrix}$$

sum along columns

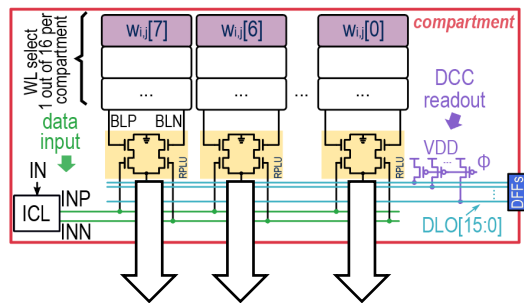
# Key Contributions

## ADC-Less Vector-Hadamard-Product (VHP) Architecture



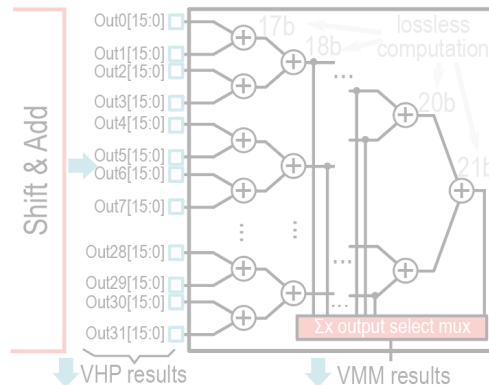
## Inside a compartment

## Reconfigurable Local Process Units (RLPU) + Dynamic Logic Compute Circuits (DCC)



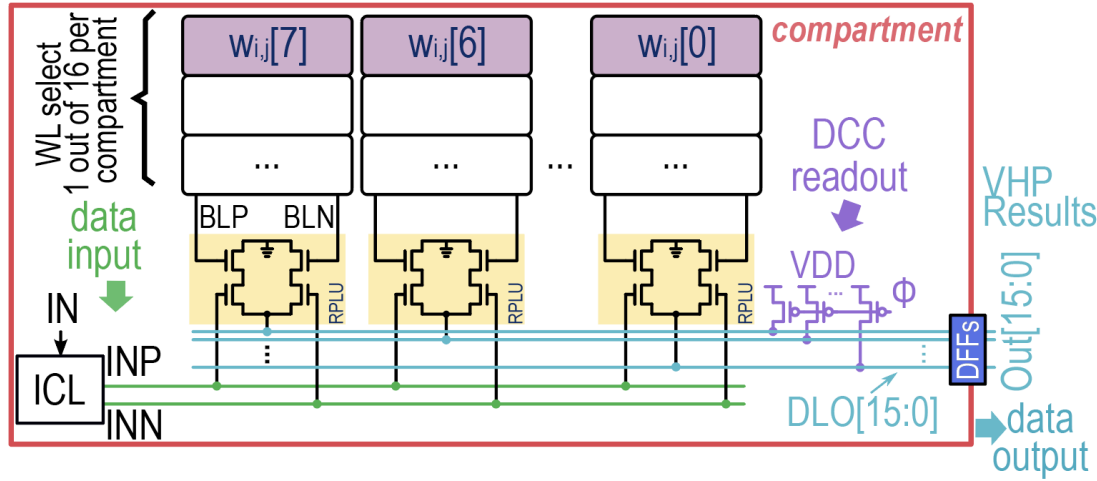
**Reconfigurable to  
AND OR XOR**

## Flexible VHP/VMM Support With Post Sum Circuits



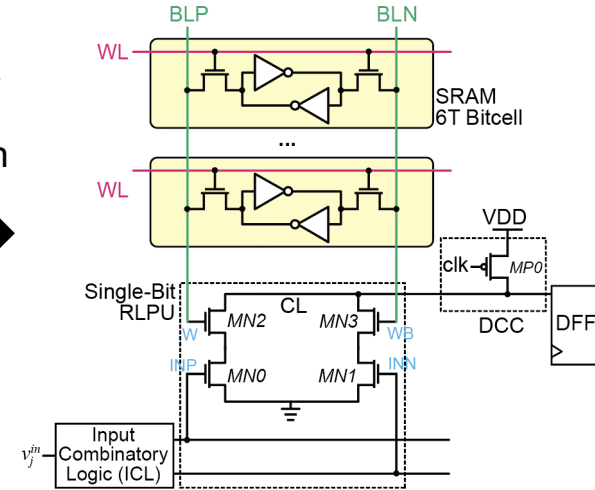
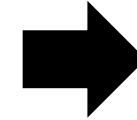
# Principle of RLPU+DCC

## Inside a compartment



RLPU: Reconfigurable Local Process Units  
DCC: Dynamic logic Compute Circuits

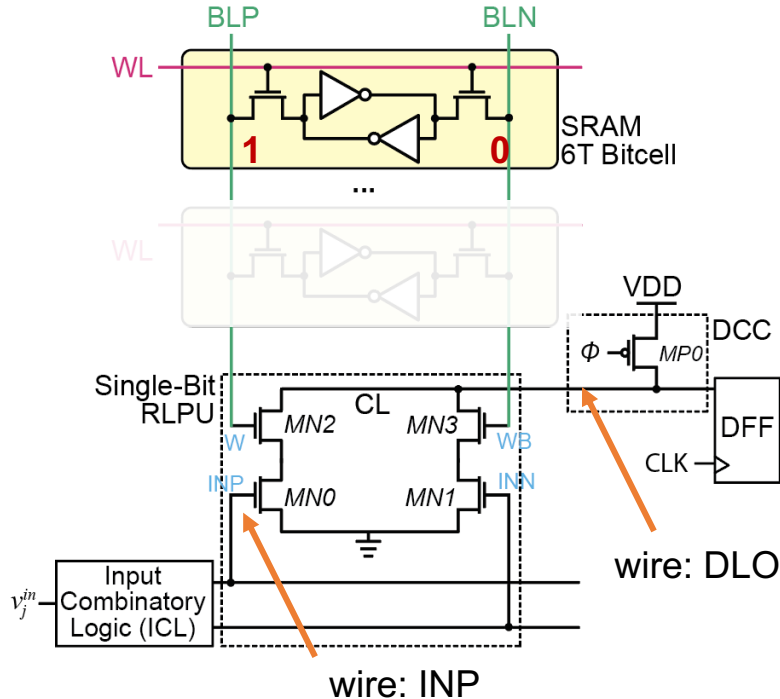
Select  
one  
column



Reconfigure through ICL  
Available bitwise logic options:  
AND, OR, XOR

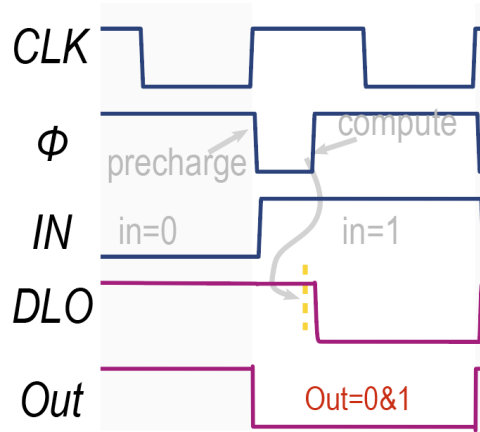
# Principle of RLPU+DCC

- Take RLPU configured to AND as an example



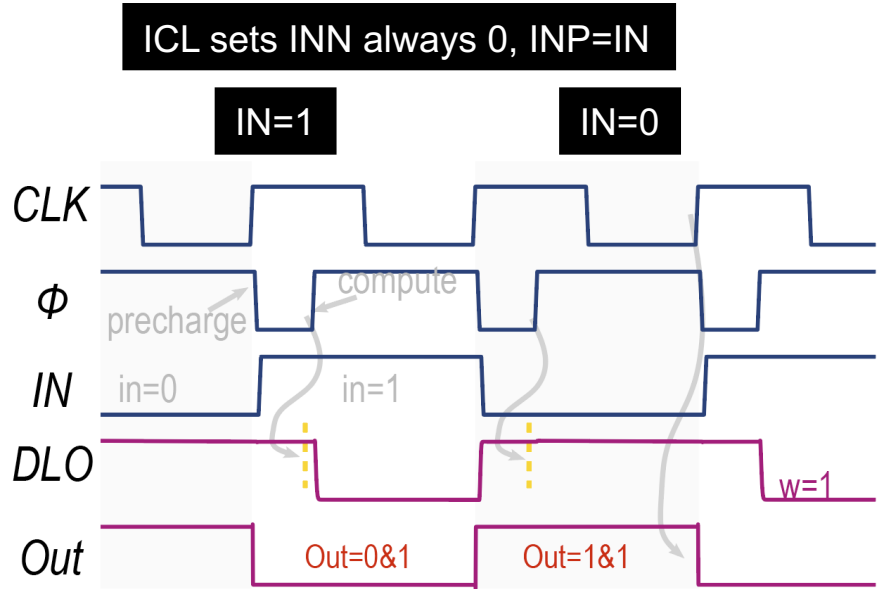
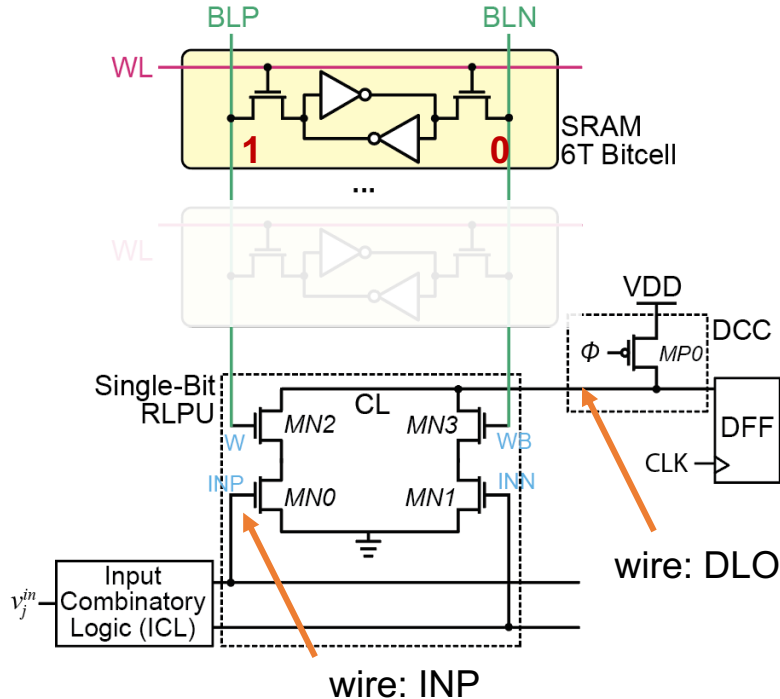
ICL sets INN always 0, INP=IN

IN=1



# Principle of RLPU+DCC

- Take RLPU configured to AND as an example



# RLPU Reconfigurability

- All possible bitwise configurations

$$\text{Dynamic Logic Output} = \overline{\text{INP} \cdot w + \text{INN} \cdot \bar{w}}$$

RPLU Logic	Config via ICL	IN	INP	INN	w	$\bar{w}$	Dynamic Logic Output DLO[k]	RPLU Logic Result Out[k]
AND (multiply)	INP=IN INN=0	0	0	0	0	1	1	0
		1	1	0	0	1	1	0
		0	0	0	1	0	1	0
		1	1	0	1	0	0	1
OR	INP=0 INN=IN	0	0	1	0	1	0	0
		1	0	0	0	1	1	1
		0	0	1	1	0	1	1
		1	0	0	1	0	1	1
XOR	INP=IN INN=IN	0	0	1	0	1	0	0
		1	1	0	0	1	1	1
		0	0	1	1	0	1	1
		1	1	0	1	0	0	0

Applications



**Element multiplication**  
-Vector matrix  
multiplications  
-Hadamard product



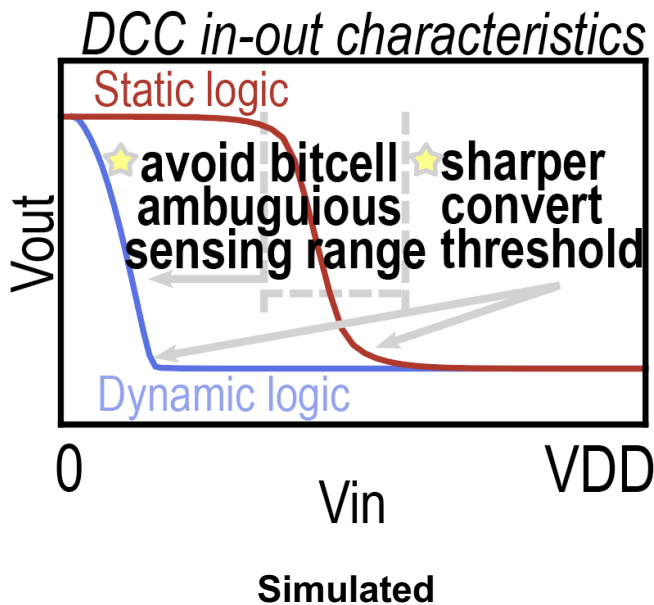
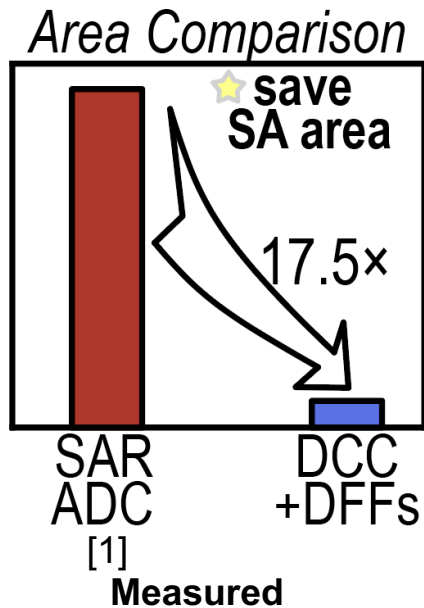
**CIM data mask**



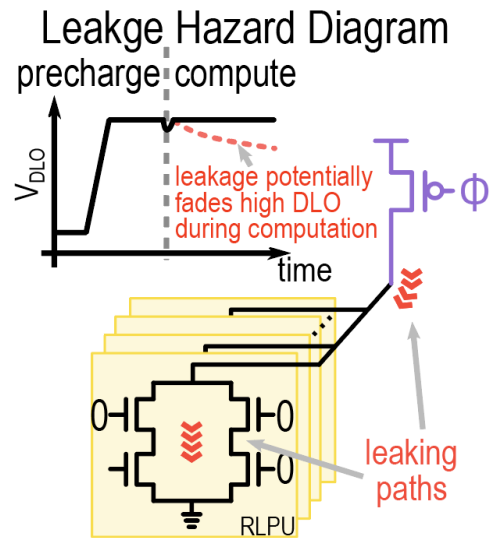
**Hamming distance  
Computation**

# DCC Characteristics

## Strengths

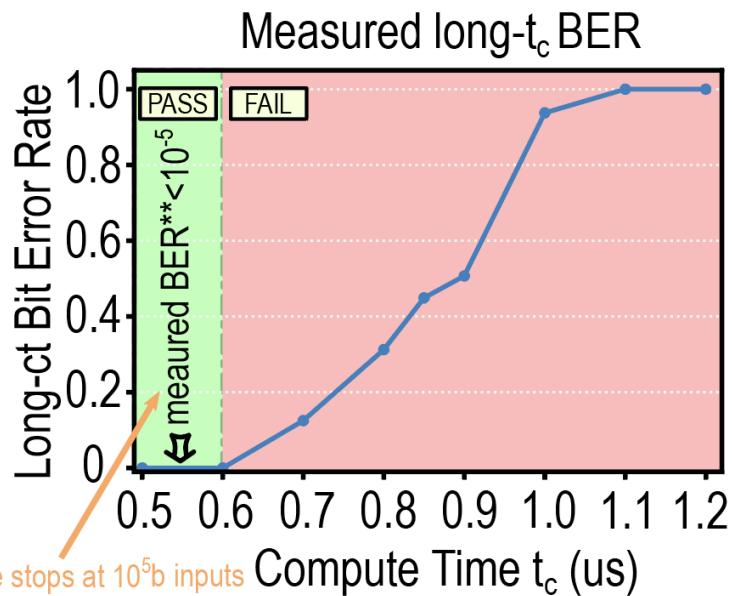


## Possible Weakness??





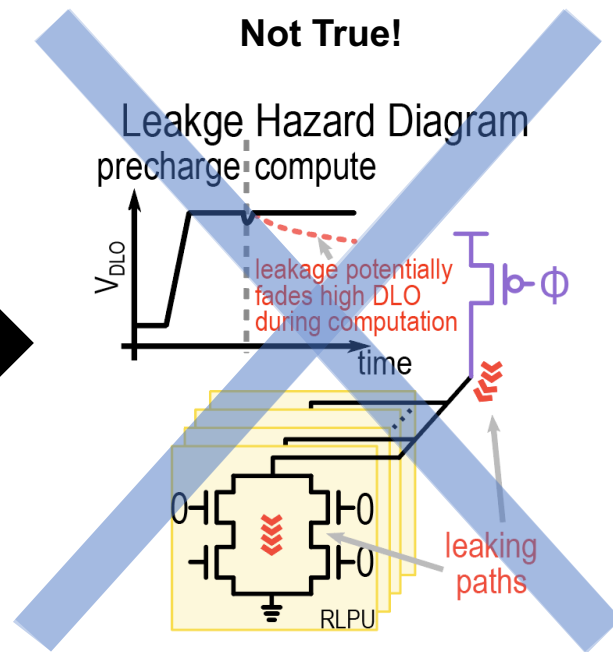
# DCC Measurement



Common compute time is <10ns

Test @ 25°C

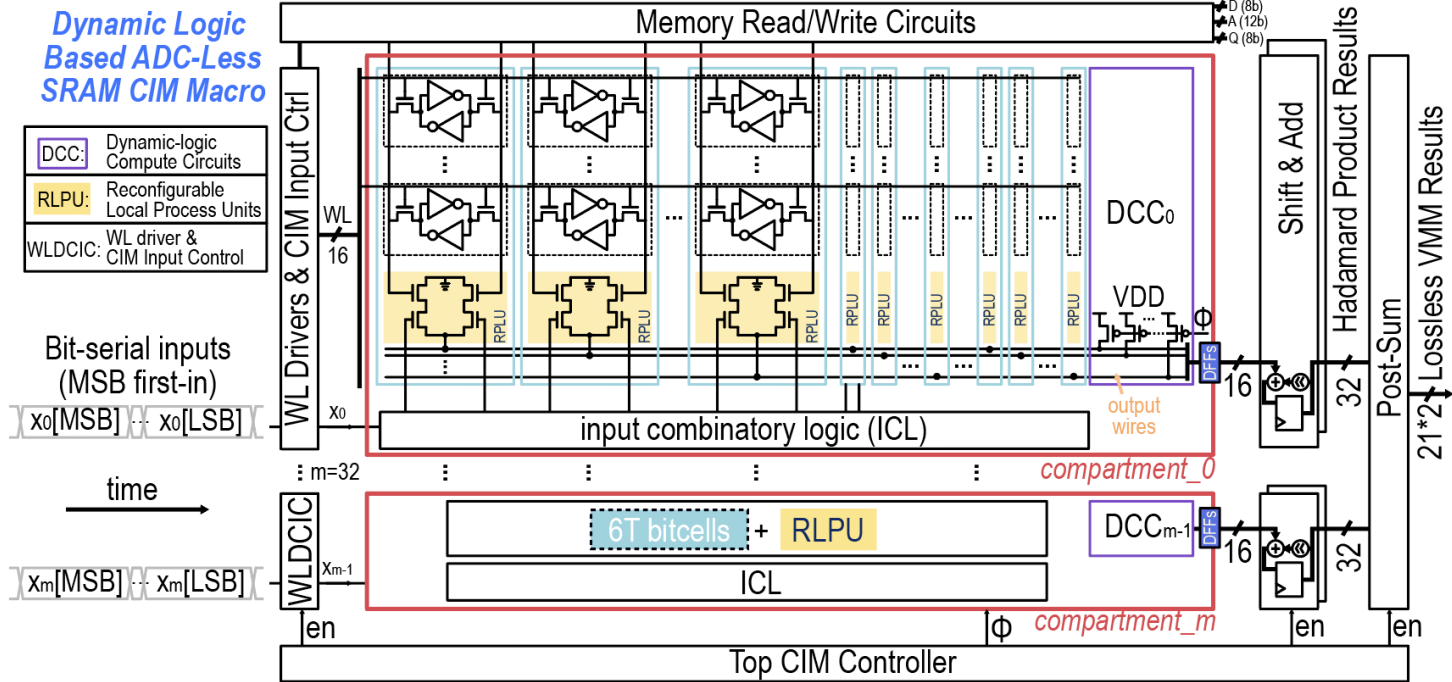
Dismiss  
this  
concern



# Entire Architecture

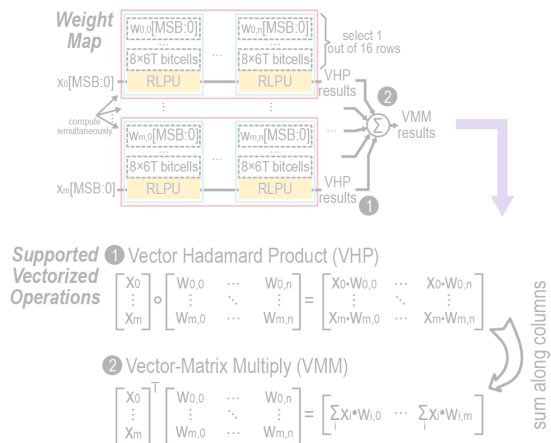
Novelty:

- Compartment-based organization
- VHP architecture
- DCC+RLPU circuits

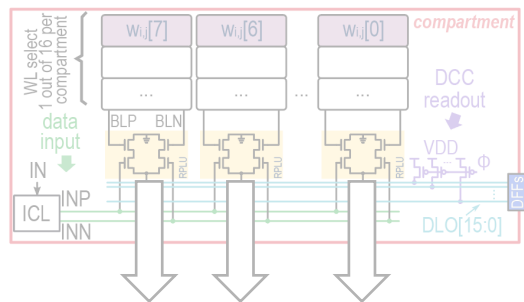


# Key Contributions

## ADC-Less Vector-Hadamard-Product (VHP) Architecture



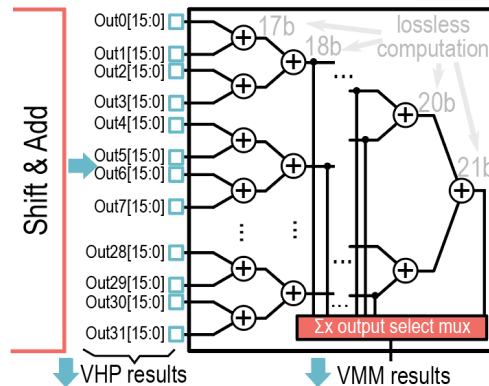
## Reconfigurable Local Process Units (RPU) + Dynamic Logic Compute Circuits (DCC)



Reconfigurable to  
AND OR XOR

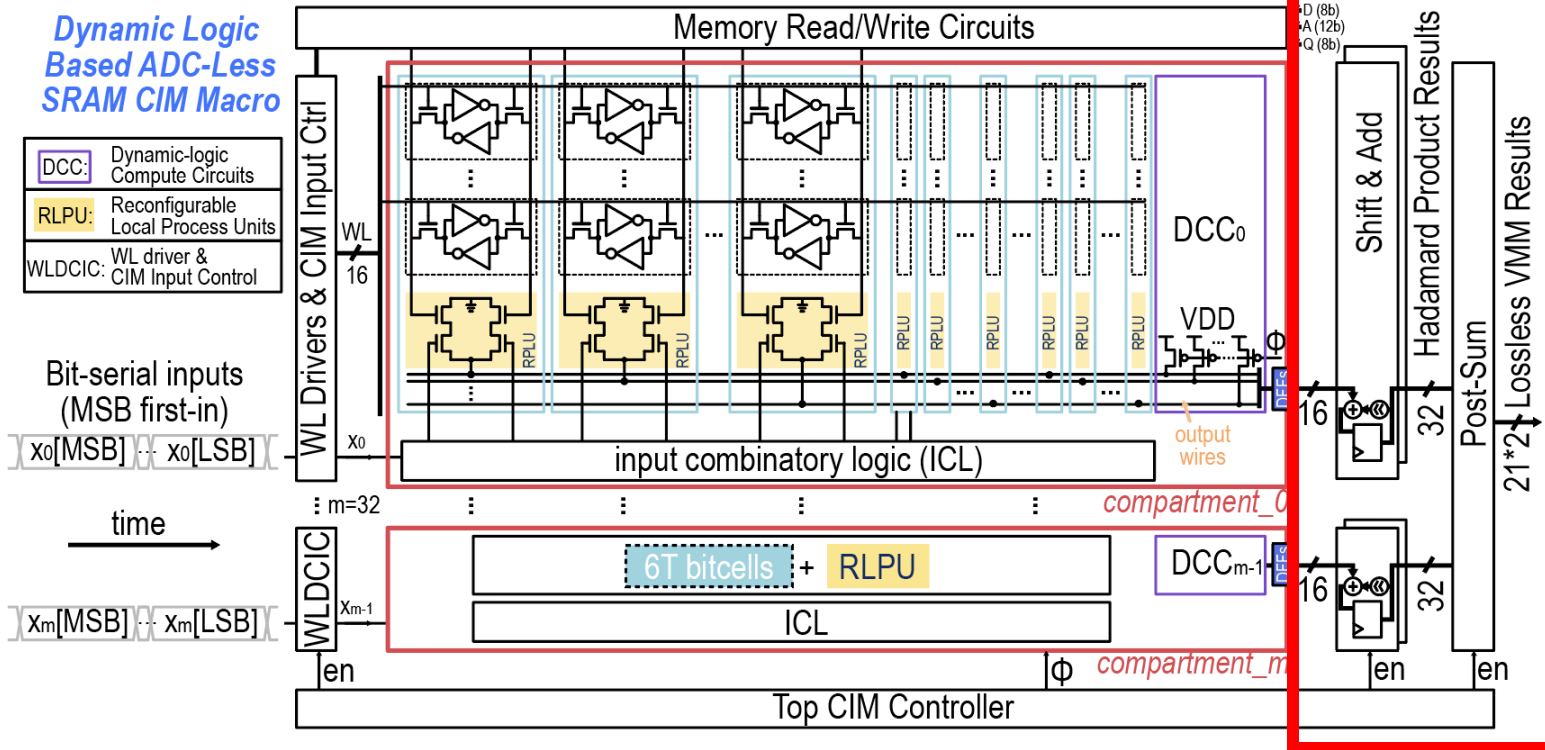
## After a compartment

## Flexible VHP/VMM Support With Post Sum Circuits

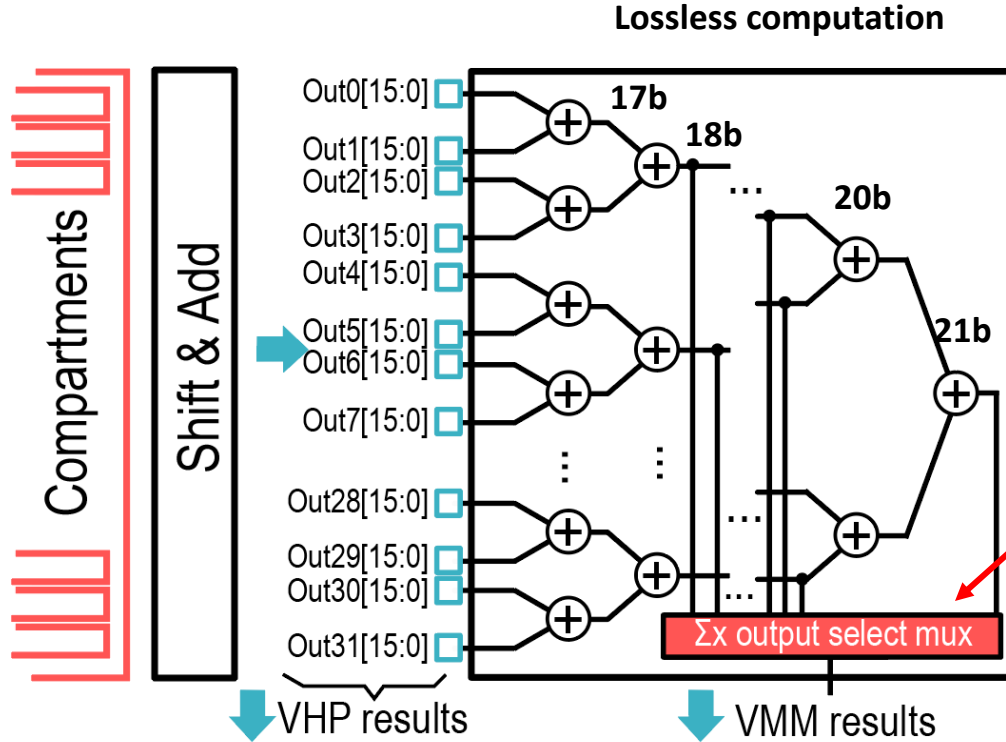


# Introduction of Post-Sum Circuits

After a compartment



# Post-Sum Circuits

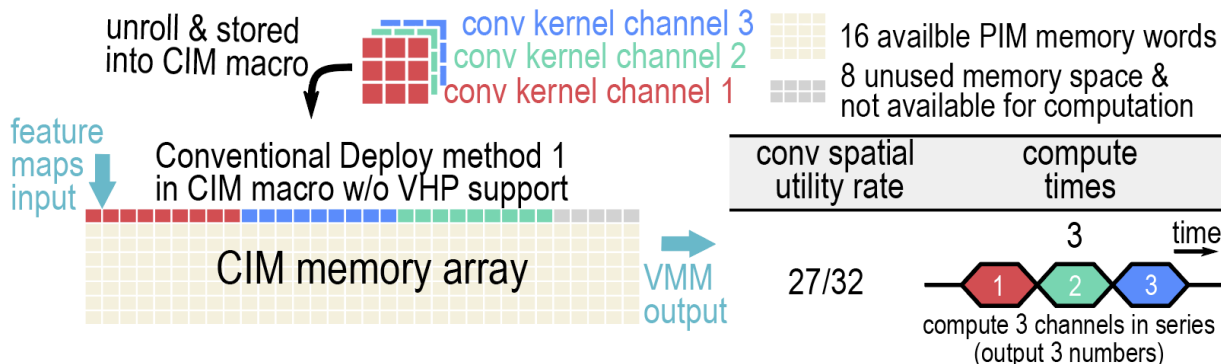


- Shift & Add: Combine Bit-Serial Inputs
- Adder Tree: Convert Hadamard Products Into VMM Results

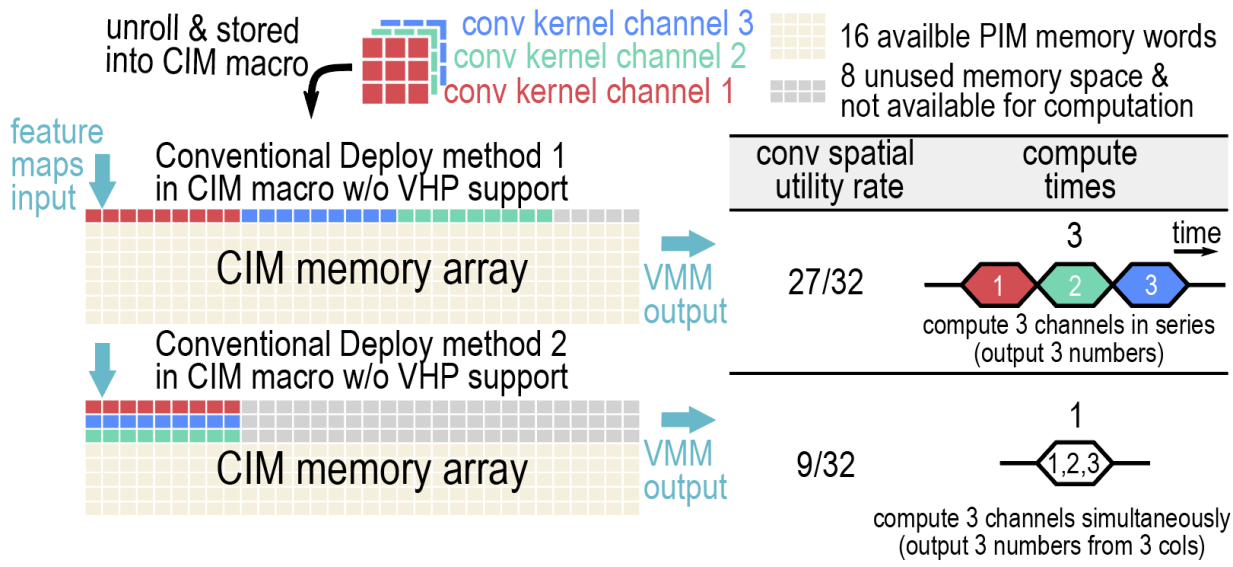
We Have A Little Trick Here:  
Arrange A  $\Sigma 9$  Here

$\Sigma k$ : sum of the  $k$  numbers

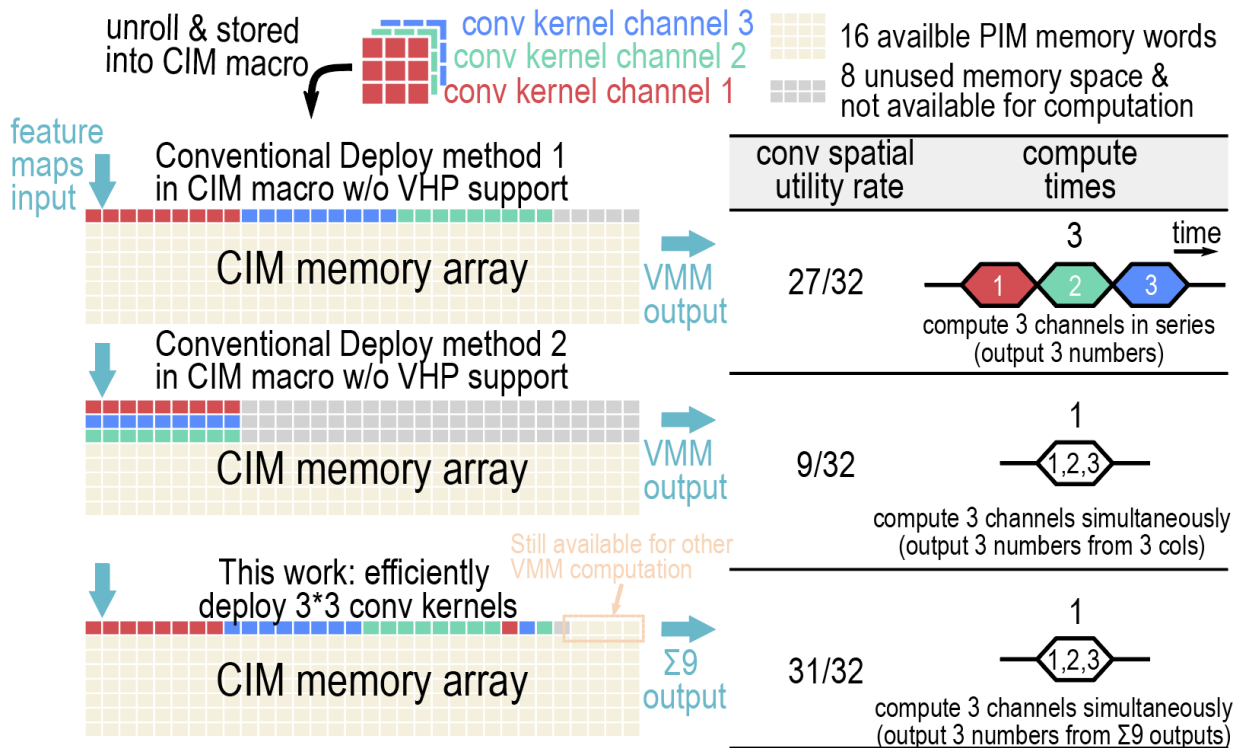
# Why $\Sigma k$ ?



# Why $\Sigma k$ ?



# Why $\Sigma k$ ?

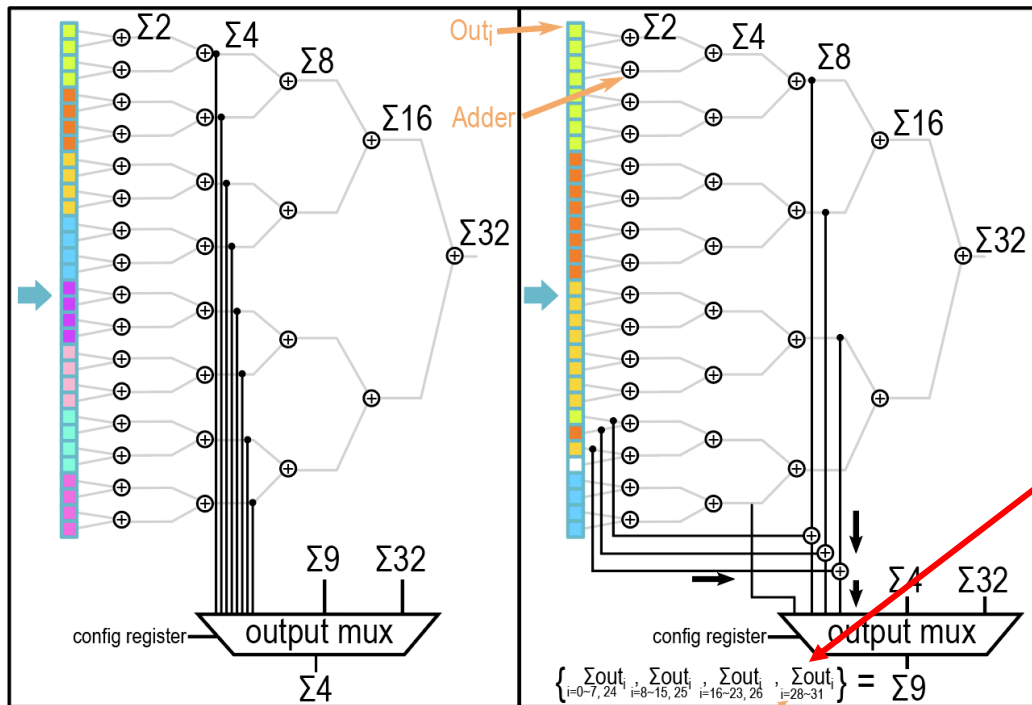




# How $\Sigma k$ ?

Detailed circuit diagram of  $\Sigma 4$  &  $\Sigma 9$ :

■ w/o  $\Sigma 9$  ■ w/  $\Sigma 9$  (This work)



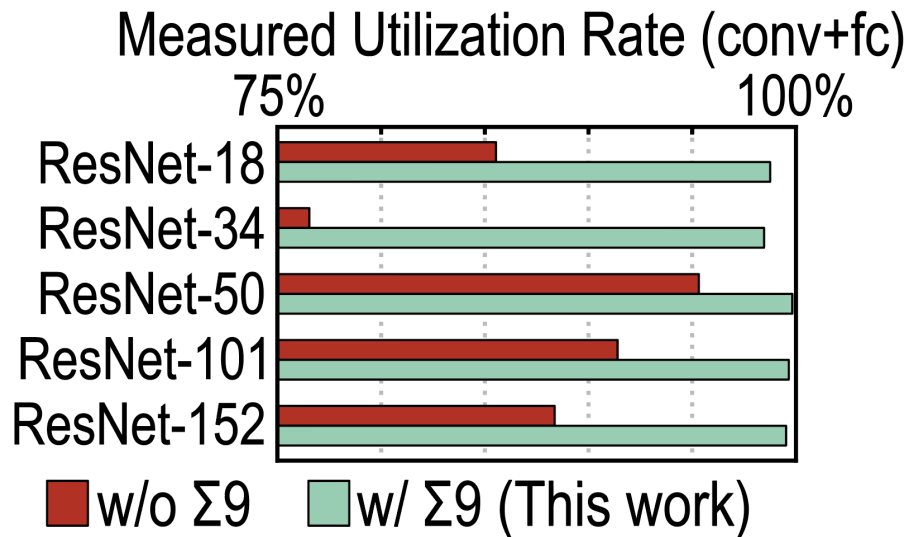
- $\Sigma 4$ : for 2\*2 Conv Kernel
- $\Sigma 9$ : for 3\*3 Conv Kernel

Three groups of  $\Sigma 9$   
and One  $\Sigma 4$  fits in a 32-element output  
vector

Why "32": our design has 32  
compartments

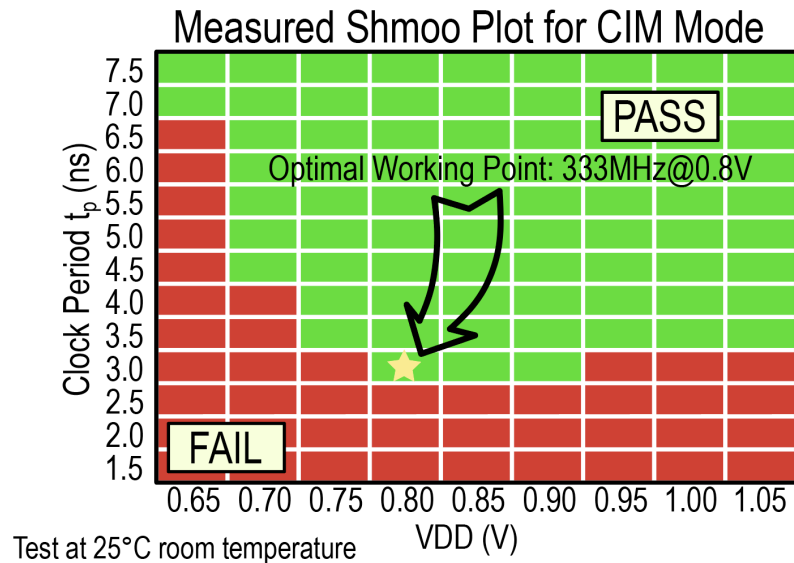
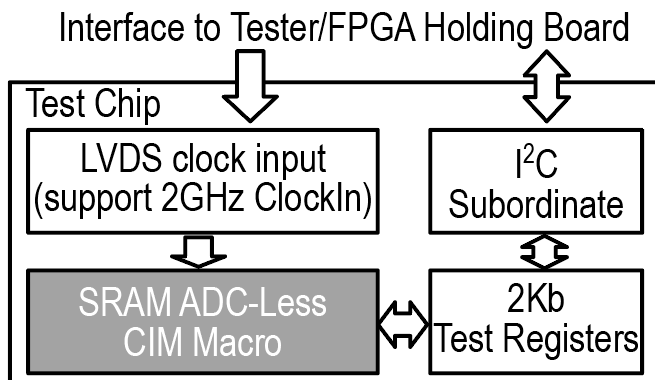
additional  $\Sigma 4$  result to improve utilization rate

# $\Sigma k$ Benefits



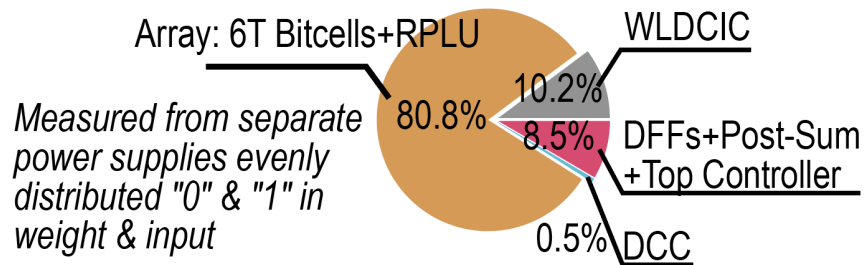
- Increase Utilization Rate
  - (Regular Shaped) CIM Macro is Naturally Good At Dense (Fully-Connected) Layer
  - $\Sigma 9$  is Added Specially for Ubiquitous 3\*3 Conv Kernel

# Measurement



# Measurement

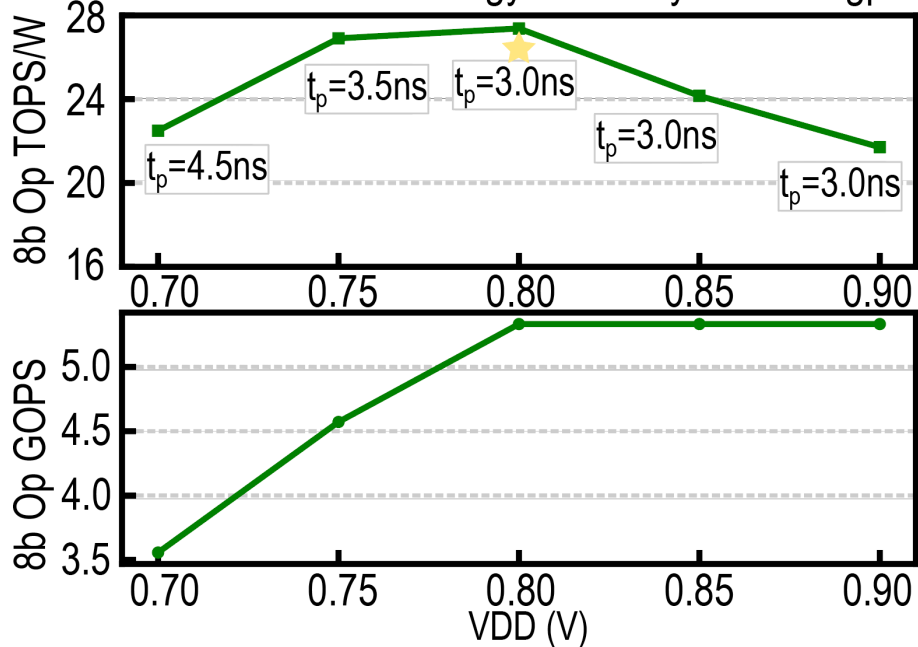
## Measured Power Breakdown



## Testing Configuration:

- Average values w/ precision (input, weight)=(8b,8b)
- Benchmarked by conv & fc layers of quantized ResNet-34 & MobileNet, data reload not included

## Measured CIM Mode Energy Efficiency & Throughput



# Comparison

## On Form Factor

CIM Macro Type	<i>Analog CIM</i>				<i>Digital CIM</i>		
Work	ISSCC'18 [1]	ISSCC'20 [2]	JSSC'21 [3]	ISSCC'21 [4]	ESSCIRC'19 [5]	ISSCC'21[6]	This Work
Technology	65nm	28nm	7nm	28nm	65nm	22nm	<b>28nm</b>
Array Size	4Kb	64Kb	4Kb	384Kb	16Kb	64Kb	<b>32Kb</b>
Cell Type	S6T	6T	8T	6T	6T	6T	<b>6T</b>
Macro Area	N/A	0.362mm <sup>2</sup>	0.0032mm <sup>2</sup>	1.4mm <sup>2†</sup>	0.2272mm <sup>2</sup>	0.202mm <sup>2</sup>	<b>0.030mm<sup>2</sup></b>
CIM Weight Density	N/A	177Kb/mm <sup>2</sup> @28nm	1250Kb/mm <sup>2</sup> @7nm	234Kb/mm <sup>2</sup> @28nm	71Kb/mm <sup>2</sup> @65nm	317Kb/mm <sup>2</sup> @22nm	<b>1067Kb/mm<sup>2</sup> @28nm</b>
CIM Weight Density (normalized to 28nm)	N/A	177Kb/mm <sup>2</sup>	78Kb/mm <sup>2</sup>	234Kb/mm <sup>2</sup>	383Kb/mm <sup>2</sup>	196Kb/mm <sup>2</sup>	<b>1067Kb/mm<sup>2</sup></b>
Power Supply	1V, 0.8V	0.7V-0.9V	1V, 0.8V	0.7-0.9V	0.6V-0.8V	0.72V	<b>0.8V</b>

† Estimated from [4]

# Comparison

# On Function Diversity

CIM Macro Type	<i>Analog CIM</i>				<i>Digital CIM</i>		
Work	ISSCC'18 [1]	ISSCC'20 [2]	JSSC'21 [3]	ISSCC'21 [4]	ESSCIRC'19 [5]	ISSCC'21[6]	<b>This Work</b>
Technology	65nm	28nm	7nm	28nm	65nm	22nm	<b>28nm</b>
Compute Circuits	CMI-VSA	LMAR-SAR-ADC	Flash ADC	Ph-ADC	CMOS Static Logic	CMOS Static Logic	<b>Dynamic Logic Compute Circuit</b>
Support Bitwise Operation	AND	AND	AND	AND	AND	AND	<b>AND, XOR, OR</b>
Fundamental Vector Operation	VMM	VMM	VMM	VMM	VMM	VMM	<b>Hadamard Product, VMM</b>

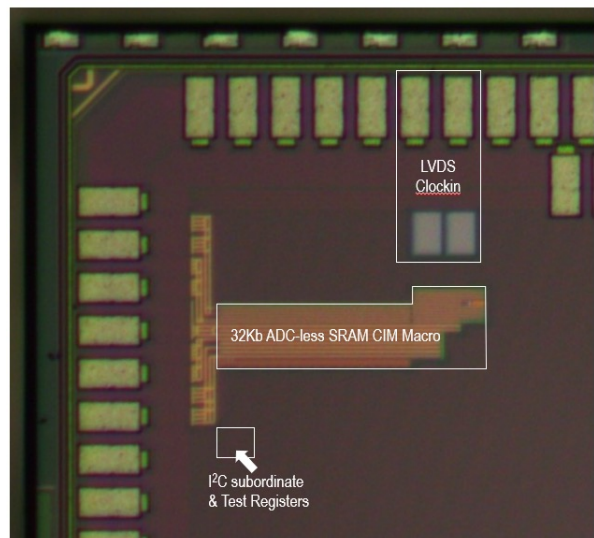
# Comparison

## On Computation & Efficiency

CIM Macro Type	Analog CIM				Digital CIM		
Work	ISSCC'18 [1]	ISSCC'20 [2]	JSSC'21 [3]	ISSCC'21 [4]	ESSCIRC'19 [5]	ISSCC'21[6]	This Work
Input Bits	1	4b/8b	4b	4b/8b	1b-16b	1b-8b	1b-8b
Weight Bits	1	4b/8b	4b	4b/8b	4b/8b/12b/16b	4b/8b/12b/16b	1b/4b/8b
Output Bits	1	12b (4b/4b) 20b (8b/8b)	4b	12b (4b/4b) 20b (8b/8b)	8b-23b	16b (4b/4b) 24b (8b/8b)	VMM: 21b VHP: 8b (8b/8b)
Cycle Time	2.3ns	4.1ns (4b/4b) 8.4ns (8b/8b)	4.5ns (1.0V) 5.5ns (0.8V)	4ns (4b/4b) 7.2ns (8b/8b)	N/A	10ns (4b/4b) 18ns (8b/8b)	3ns/input bit
Energy Efficiency** (TOPS/W)	55.8 (1b/1b)	58.1(4b/4b) 14.1(8b/8b)	351 (1b/1b)	77.3(4b/4b) 18.9(8b/8b)	117.3 (1b/1b)	24.7 (8b/8b)	27.38 (8b/8b)
CIM Weight Density (normalized to 28nm)	N/A	177Kb/mm <sup>2</sup>	78Kb/mm <sup>2</sup>	234Kb/mm <sup>2</sup>	383Kb/mm <sup>2</sup>	196Kb/mm <sup>2</sup>	1067Kb/mm <sup>2</sup>
SWaP (FoM)*** (TOPS/W·Mb/mm <sup>2</sup> )	N/A	161	26.7	283	8.13	303	1826

\*\* Average energy efficiency, no sparsity employed \*\*\*. The figure of merit (FoM) SWaP (space, wattage and performance)=(Weight Density\*Performance)/Power=Weight Density\*Energy Efficiency

# Summary For This Digital SRAM CIM Design



- VHP Organization Architecture
  - Increase Flexibility
- RLPU+DCC Design
  - Simplified ADC-Less CIM Operation
- $\sum k$  Technique for CIM Macro Post-Sum Circuits
  - Increase Conv Kernel Utilization Rate

Chip Demo Video:

<https://www.bilibili.com/video/BV15b4y1H7gP>

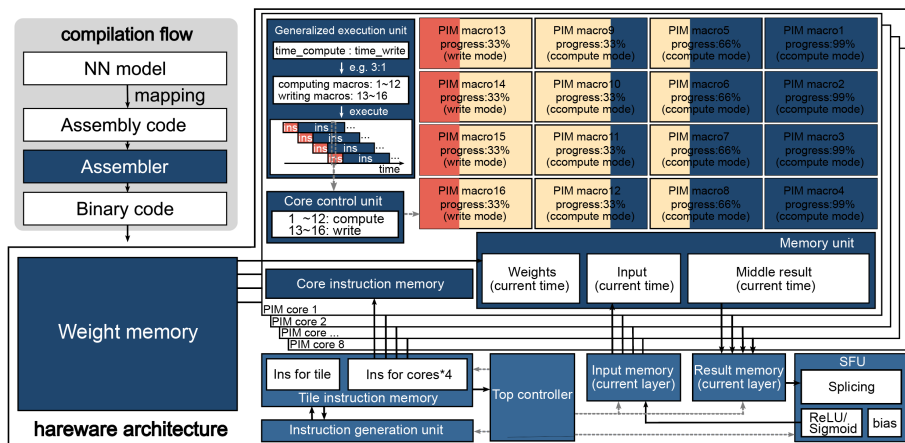


# Subsequent Works: PIM SoC Open-Source Project

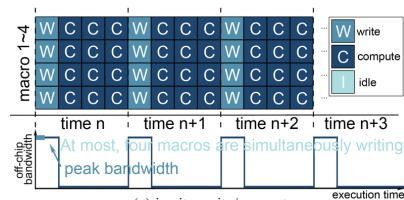
PIM-SoC Design: (graduate course project @ PKU)

- PIM Macros (Verilog Behaviors)
- Programmable Interface

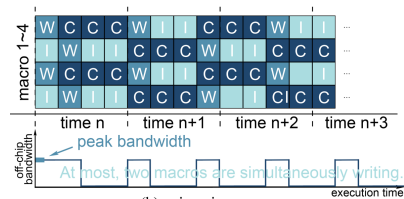
- Simulate Complex Dataflows w/ instructions



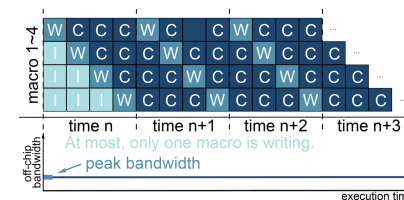
Available at: <https://github.com/rw999creator/gpp-pim>  
<https://arxiv.org/abs/2411.13054>



(a) in situ write/compute

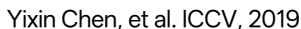


(b) naive ping-pong



(c) generalized ping-pong

## Emerging Embodied AI "Think&Estimate" Applications



**Probabilistic Program**

```

with pm.Model() as model:
    variable[0] = prior.distribution[0]
    variable[1] = prior.distribution[1]
    variable[n-1] = prior.distribution[n-1]
    obs.data = Likelihood("obs.data", observed=data)
with model:
    data = pm.sample(step = pm.MetropolisDiag())
  
```

**Overall PROCA Architecture**


The architecture consists of a **Top-level Scheduler** and **Current Sample Register** at the top. Below them are multiple **PROCA core n-1** (random variable[1]) and **PROCA core n** (random variable[2]). Each core contains a **Counter**, **Sample Generation Module**, **URG Module**, **GPC**, and **Acceptance Ratio Calculation Module**.

**PROCA core**

The core is divided into three main functional blocks:

- Sample Generation Module:** Includes a **Sampling Scheduler**, **Calculate (BF16)**, **Sample (BF16)**, **INT8-BF16 Converter**, and **Sample (INT8)**.
- Probabilistic PIM Macro:** A grid of **Sampling Block** and **Sampling Block** units.
- Acceptance Ratio Calculation Module:** Includes a **Calculation Scheduler**, **BF16 Adder**, **BF16 Comparator**, **URG Module**, **GPC 0**, **GPC 1**, **Decoder**, and **ALU**.

The **PROCA core** also includes a **Sample** input, **Posterior Value**, **Posterior Value**, **Observed Data Memory**, and **Observed Data Memory**.

- 

# Algorithm

# Framework

## AI Chips

## Probabilistic AI

Bayesian Regression, VAE, Hidden Markov Models, Normalizing Flows, Deep Markov Model, Causal Effect VAE, Gaussian Mixture Model, Gibbs Sampling, Metropolis-Hasting, Probabilistic Circuits, ...

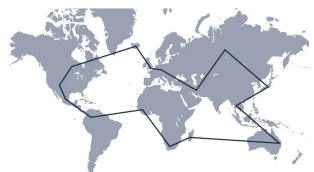


## Lack Chip Infrastructure!

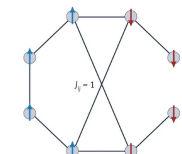
**Yihan Fu, et al., TCAS-I'2025**  
**Yihan Fu, et al., HPCA'2025**

# RRAM CIM Based Universal Ising Machine

- NP-Hard Combinatorial Optimizations Problems



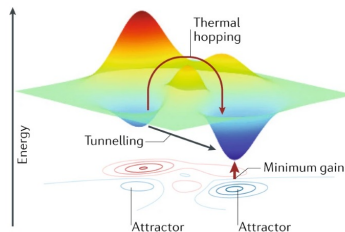
- Deploy to Ising Graphs



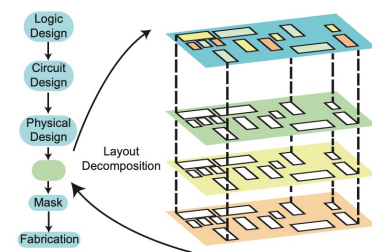
$$H_p = \sum_{i,j=1}^N J_{ij} \sigma_i \sigma_j + \sum_{i=1}^N h_i \sigma_i$$



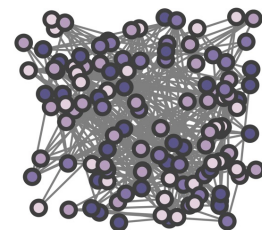
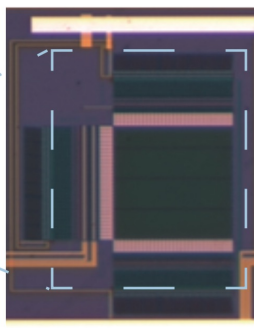
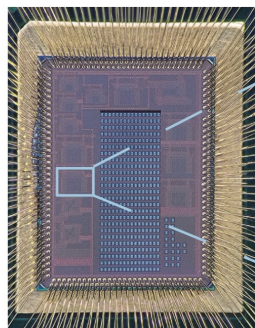
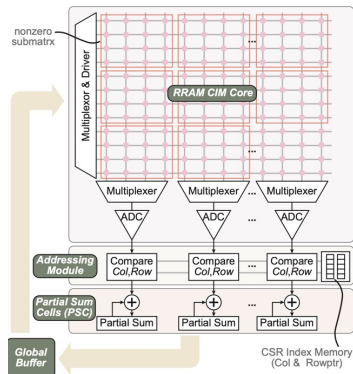
- Simulated Annealing Methods to Find Optimal Solutions



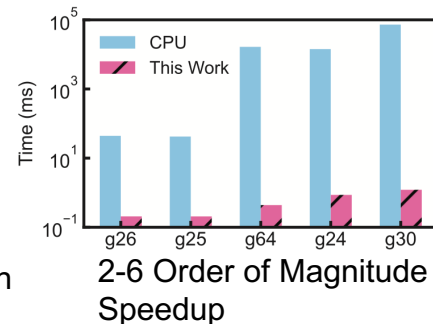
- Graph Coloring in EDA (layout decomposition)



- Develop RRAM-PIM Chip for Universal Ising Machine



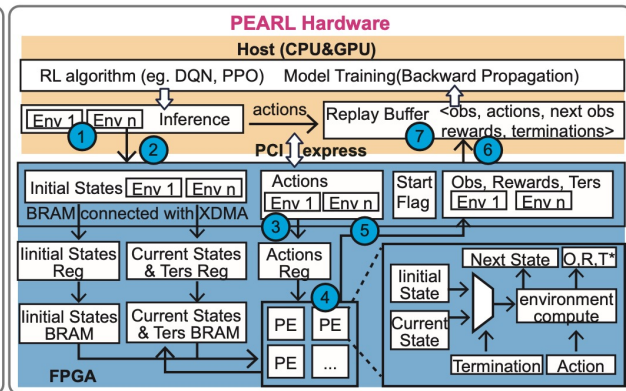
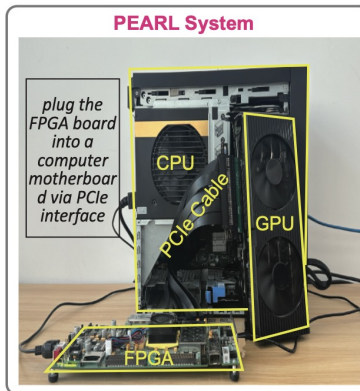
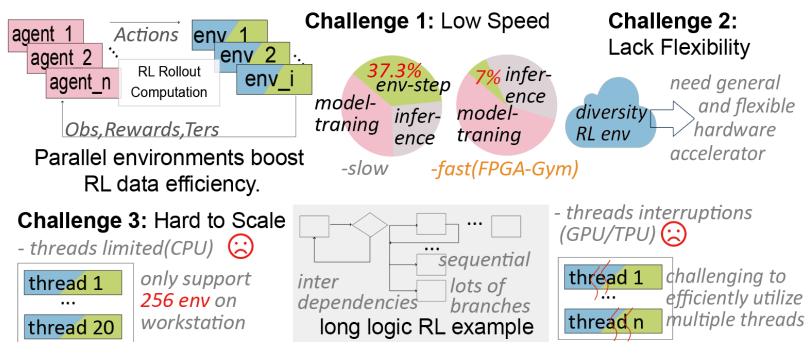
2000-vertex graph



2-6 Order of Magnitude Speedup

# Near-Memory Computing Accelerator for RL

- PEARL: FPGA-based Reinforcement Learning Environment Accelerator
- [https://github.com/Selinaee/FPGA\\_Gym](https://github.com/Selinaee/FPGA_Gym)



Challenges: GPU can accelerator policy computing, but **NOT** environment update

Introduce near-memory pipelining to accelerate RL rollout,

# More Ways to Know Us: [bonany.cc](http://bonany.cc)

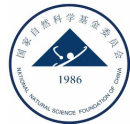
1. [HPCA'25] Fu, Yihan, et al. "PROCA: Programmable Probabilistic Processing Unit Architecture with Accept/Reject Prediction & Multicore Pipelining for Causal Inference".
2. [DATE'25] Li, Jiayi, et al. "PEARL: FPGA-Based Reinforcement Learning Acceleration with Pipelined Parallel Environments".
3. [Nature Communications'25] Yue, Wenshuo, et al., "Physical Unclonable In-Memory Computing for Simultaneous Protecting Private Data and Deep Learning Models".
4. [Nature Electronics'24] Yue, Wenshuo, et al., "A Scalable Universal Ising Machine Based on Interaction-Centric Storage and Compute-In-Memory." 7, 904–913 (2024).
5. [IEEE TCAS-I'24] Fu, Yihan, etc. "Probabilistic Compute-in-Memory Design for Efficient Markov Chain Monte Carlo Sampling." vol. 71, no. 2, pp. 703-716, 2024.
6. [ISSCC'22] Yan, Bonan\*, et al., 2022. "A 1.041-Mb/mm<sup>2</sup> 27.38-TOPS/W Signed-INT8 Dynamic-Logic-Based ADC-Less SRAM Compute-in-Memory Macro in 28nm with Reconfigurable Bitwise Operation for AI and Embedded Applications."

① SRAM-PIM Chip Function Demo

② FPGA-Gym for RL Framework



## Acknowledgement



北京通用人工智能研究院  
Beijing Institute for General Artificial Intelligence



中國計算機學會  
CHINA COMPUTER FEDERATION



小米公益基金会  
XIAOMI FOUNDATION