TRIFP-DCIM: A Toggle-Rate-Immune Floating-point Digital Compute-in-Memory Design with Adaptive-Asymmetric Compute-Tree

Xing Wang, Tianhui Jiao, Shaochen Li, Yuchen Ma,

Zhican Zhang, Zhichao Liu, Xi Chen and Xin Si*

ASP-DAC 2025

January 23, 2025

Challenges and proposed solutions

Current FP-DCIM suffers from:

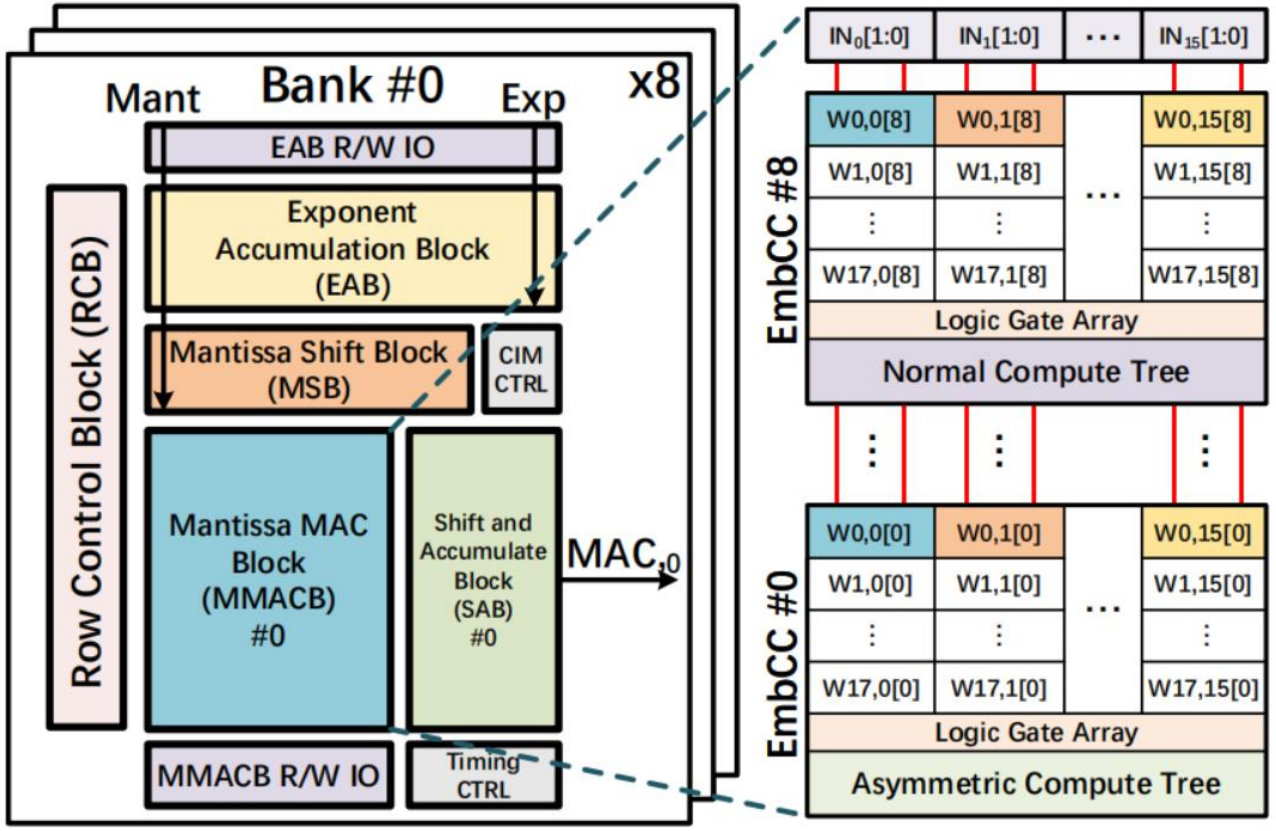
- ☹ Large area overhead of mantissa alignment
- ☹ Large-scale adder tree limits the energy/area efficiency
- ☹ Less sparsity utilization

Solutions in this work

TRIFP-DCIM proposes:

- 😀 Serial Shift Scheme for BF16 MAC
- 😀 Adaptive Asymmetric Compute-tree(AACT) Circuit
- 😀 TrifectaOne method of increasing data sparsity

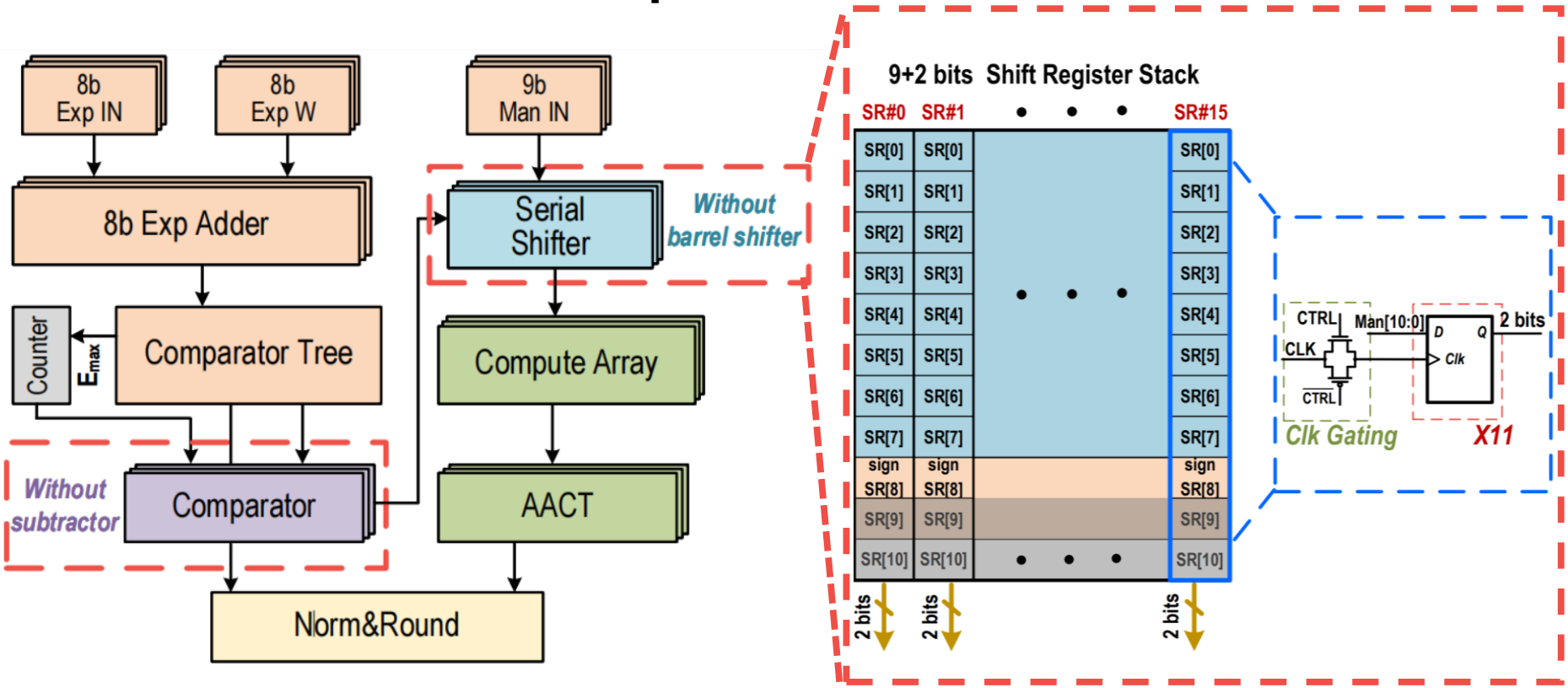
TRIFP-CIM Overall Architecture



Mantissa needs to be processed and expanded to 9 bits, adding a hidden "1" and a sign bit.

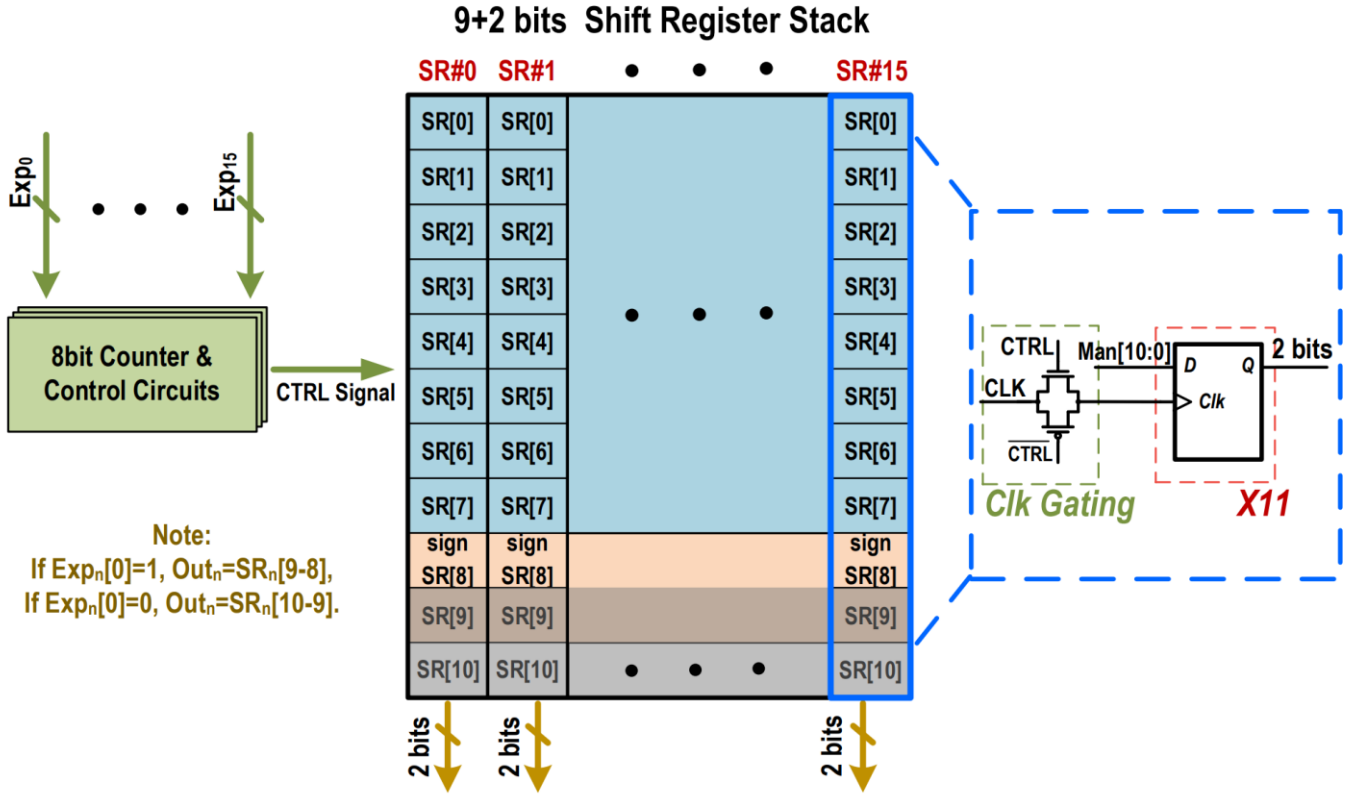
- Mantissa MAC Block (MMACB)
- Shift and Accumulate Block (SAB)
- Exponent Accumulation Block (EAB)
- Mantissa Shift Block (MSB)

BF16 Computation Dataflow



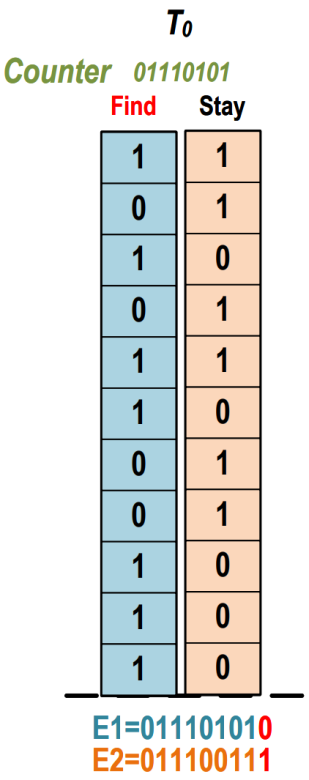
1) Compute 9-bit exponent sums from 8-bit exponents.
2) Find maximum exponent using a comparator tree.
3) Serial shift mantissa based on control signals.
4) Perform multiply-accumulate to get PMACV.
5) Integrate PMACV and exponent into BF16 format.

Serial Shift Module



- Clock Gating Units & Shift Register Stack (SRS)
- A SR is made up of 11 serially connected registers
- Highest 2 bits - sign bit extension, lower 9 bits - mantissa
- Deliver 2b of the mantissa to subsequent computation circuit each cycle

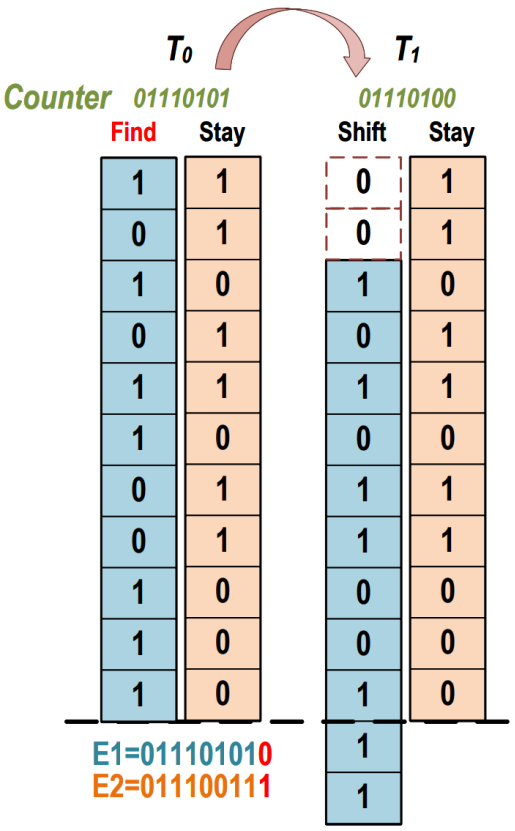
Serial Shift Scheme : An Example



1) Given *E1=011101010* and *E2=011100111*.
2) The counter counts down from *01110101*.
3) The counter value matches *E1* at *T0*, then first SR starts shifting.

When the counter value aligns with E1, the first SR enters the "Find" state at T0.

Serial Shift Scheme : An Example

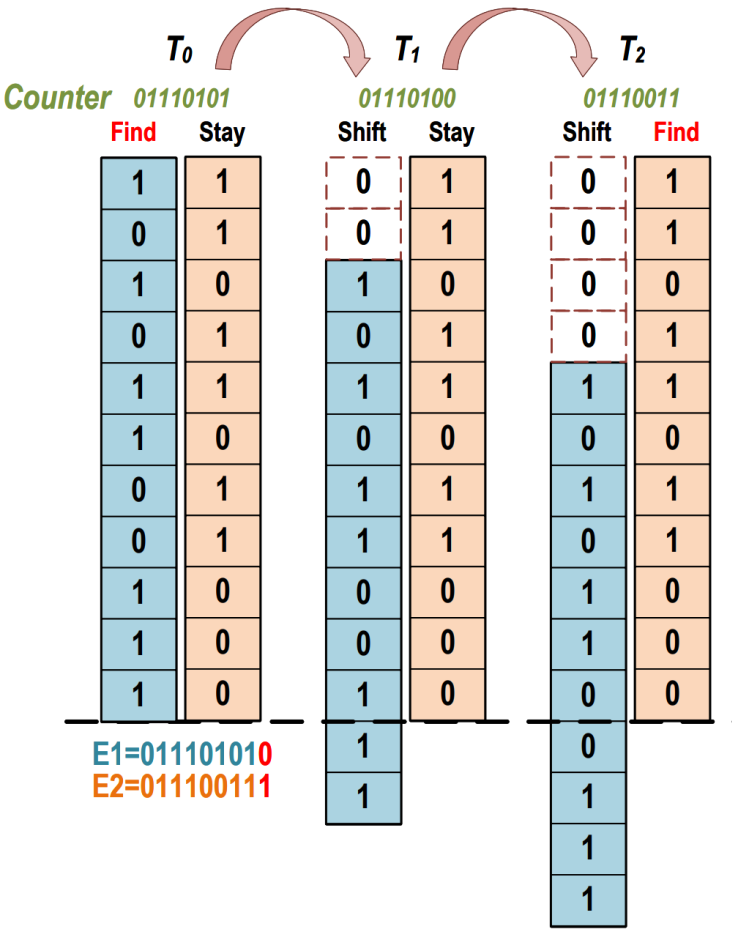


E1[0]=0
OUT1=SR[10 : 9]

1) Given *E1=011101010* and *E2=011100111*.
2) The counter counts down from *01110101*.
3) The counter value matches *E1* at *T0*, then first SR starts shifting.
4) The counter value continues to decrease by one at *T1* and *T2*.

Consequently, the first SR starts shifting at T1, while the second SR remains in the "Stay" state.

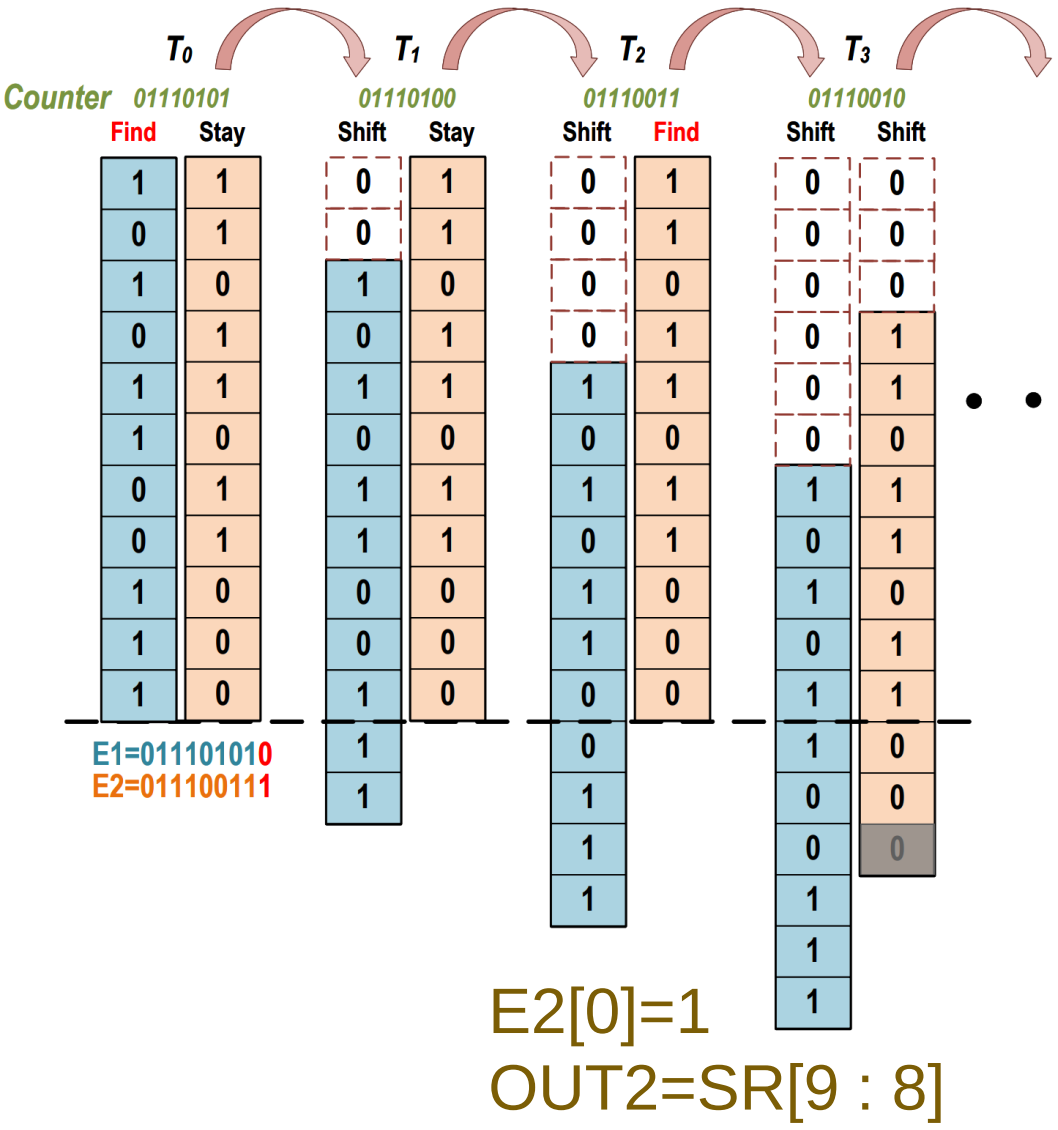
Serial Shift Scheme : An Example



1) Given *E1=011101010* and *E2=011100111*.
2) The counter counts down from *01110101*.
3) The counter value matches *E1* at *T0*, then first SR starts shifting.
4) The counter value continues to decrease by one at *T1* and *T2*.

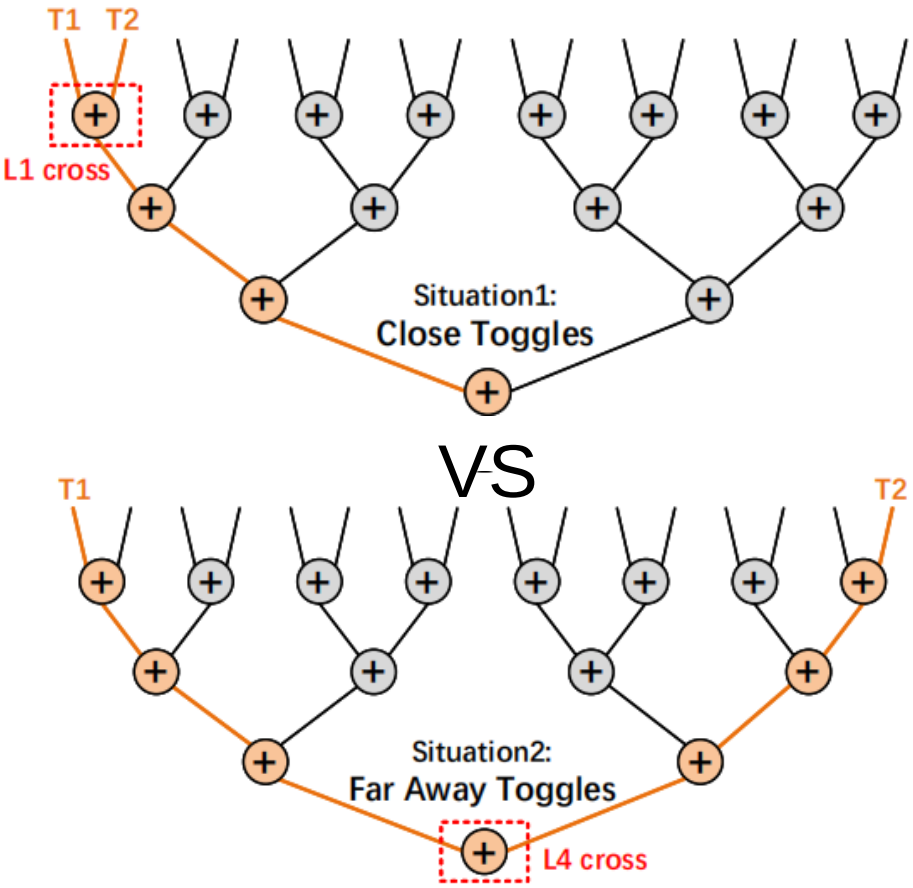
The second SR transitions to the "Find" state at T2 when the counter value equals E2, and the first SR remains in the “Shift" state.

Serial Shift Scheme : An Example



1) Given *E1=011101010* and *E2=011100111*.
2) The counter counts down from *01110101*.
3) The counter value matches *E1* at *T0*, then first SR starts shifting.
4) The counter value continues to decrease by one at *T1* and *T2*.
5) The second SR starts shifting at *T3*.
6) Continue shifting until the specified cycle number is reached.

Toggle Path Research



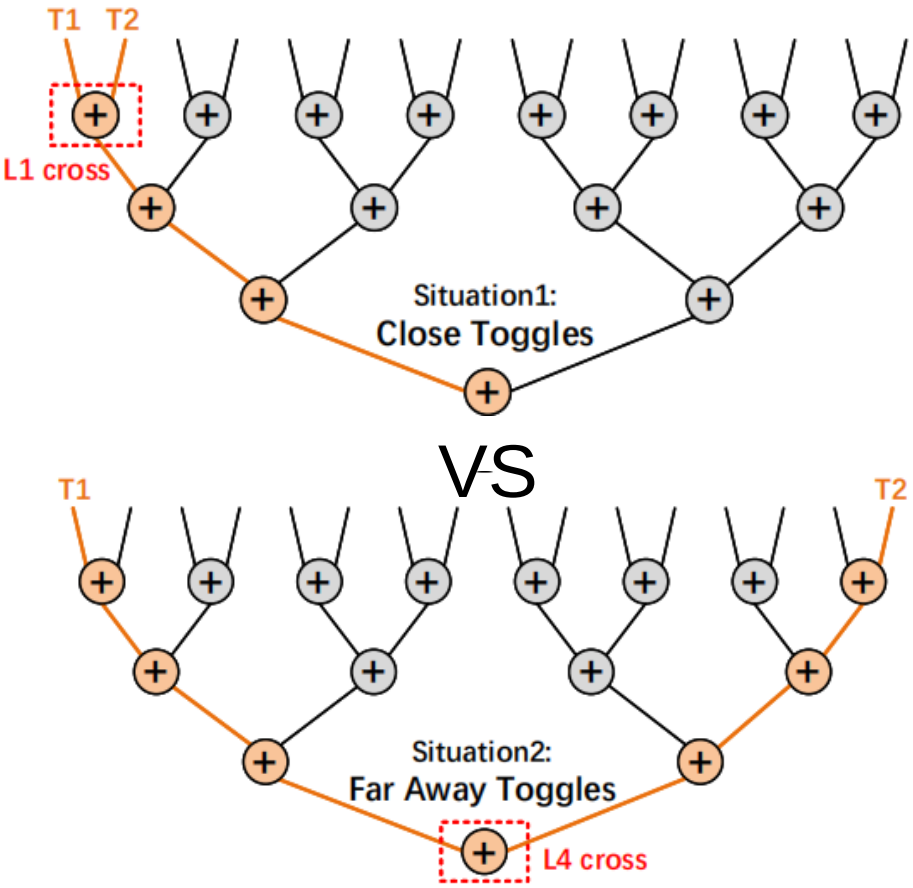
Difference:
Position of two toggling inputs

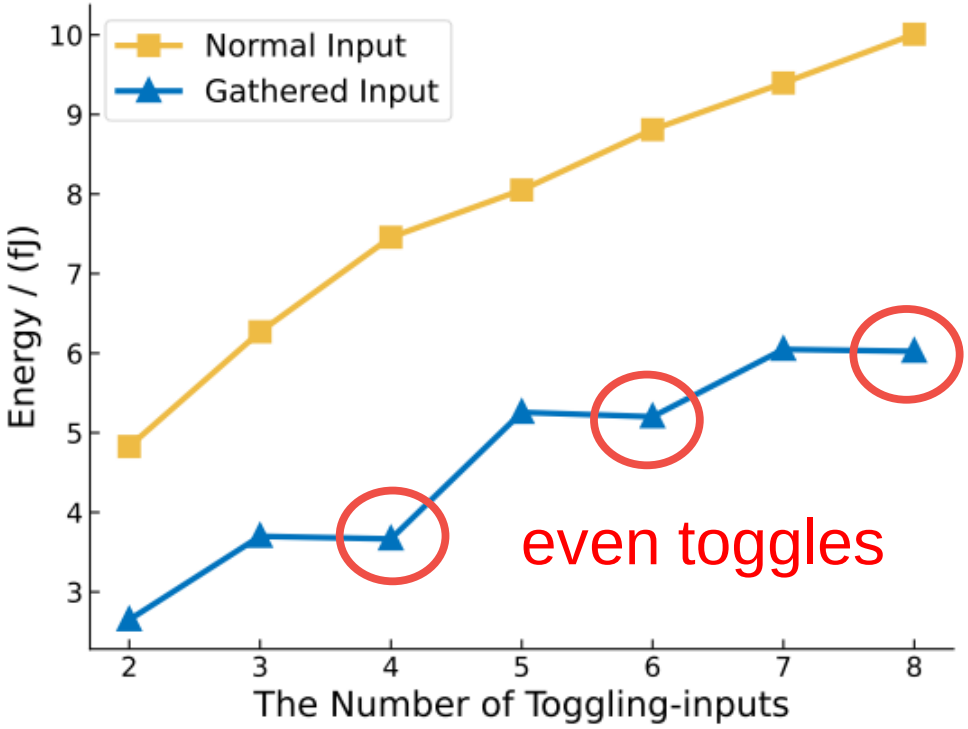
Situation 1: Close Toggles
share a data path
cross at the first adder level

Situation 2: Far Away Toggles
propagate along respective paths
cross at the fourth adder level

The power consumption of situation1 is about 50% less than that of situation2 under certain input patterns.

Toggle Path Research

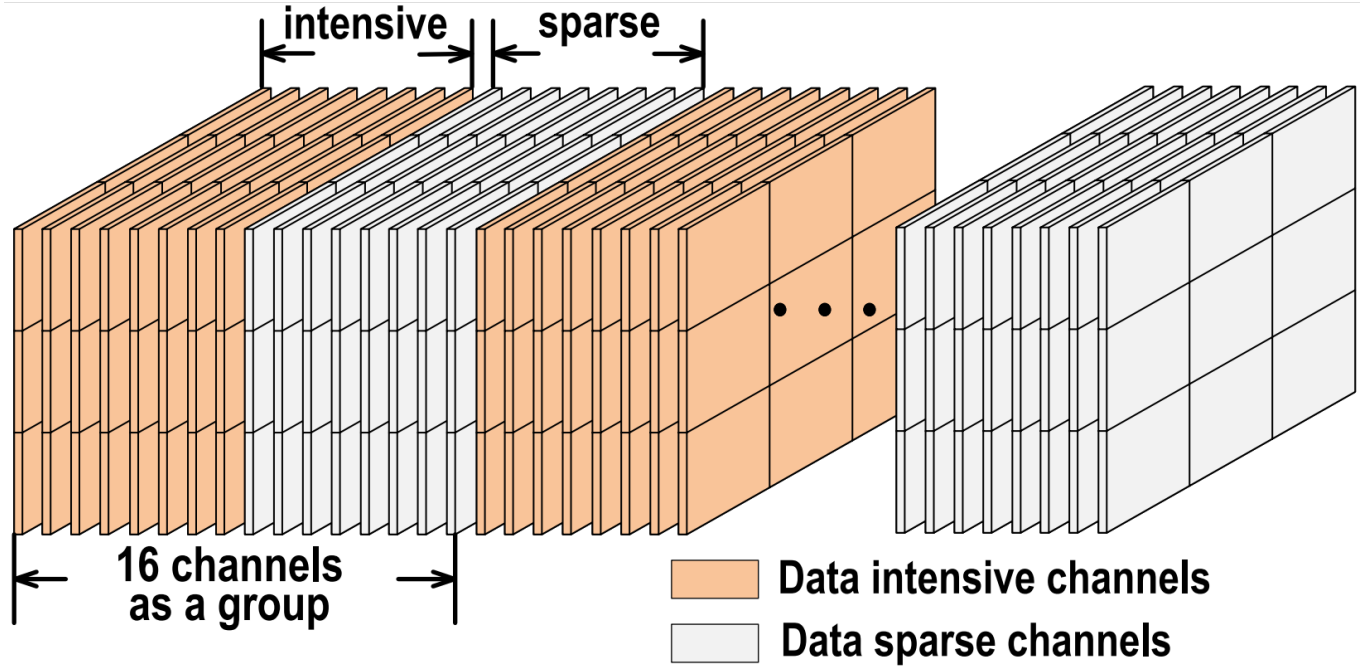




More energy reduction possibility of toggle cancellation

Conclusion: gathering toggling inputs to one side can lead to reduction of power consumption.

TrifectaOne Method



- Concentrate the toggles on one side of the compute-tree.
- Ensure 4-2 compressors on the other side will not affect the inference accuracy.

Weight Asymmetric Sparsification

Algorithm 1 Multi-channel weight asymmetric sparsification.

Input : Four-dimensional tensor W_0
Output: Four-dimensional tensor W_a

```
// Split channel
if input_channel < 8 then
    W_a = W_0;
end
if input_channel > 8 then
    split W_0 → chunks[N-1:0], N = Quotient of
      input_channel/n_size;
end
for i = 0 to (N-1) do
    if i%2 = 1 then     // all data in chunks[i]
        S_c = Sign(chunks[i]), A_c = Abs(chunks[i]);
        // Mantissa 3b Approximation
        exp0 = ⌊log2(A_c)⌋, temp = 2^exp0,
          R_c = A_c − temp;
        exp1 = ⌊log2(R_c)⌋, temp = temp + 2^exp1,
          R_c = A_c − temp;
        exp2 = ⌊log2(R_c)⌋, temp = temp + 2^exp2;
        chunk[i] = temp × S_c;
    end
end
Merge chunks on dim = 1, W_a equal to merged result;
return W_a
```

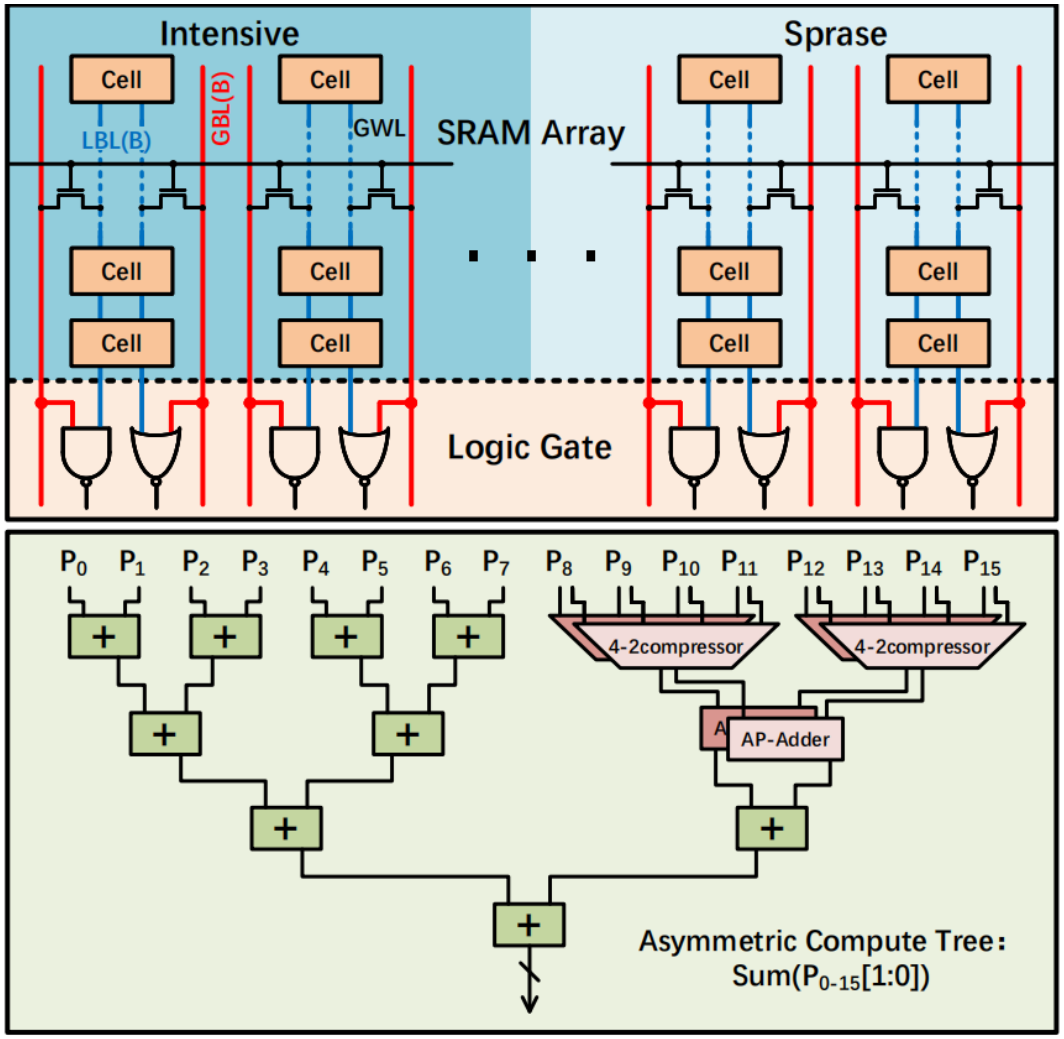
Step1:

Group 8 input channels as a chunk, alternately set as sparse chunks and dense chunks.

Step2:

Increase the sparsity of data in the sparse chunk by retaining only the first 3 ones of the mantissa.

Adaptive-asymmetric Compute-Tree



16 2-bit numbers input

Toggle-intensive side:

Full-precision CT

Toggle-sparse side:

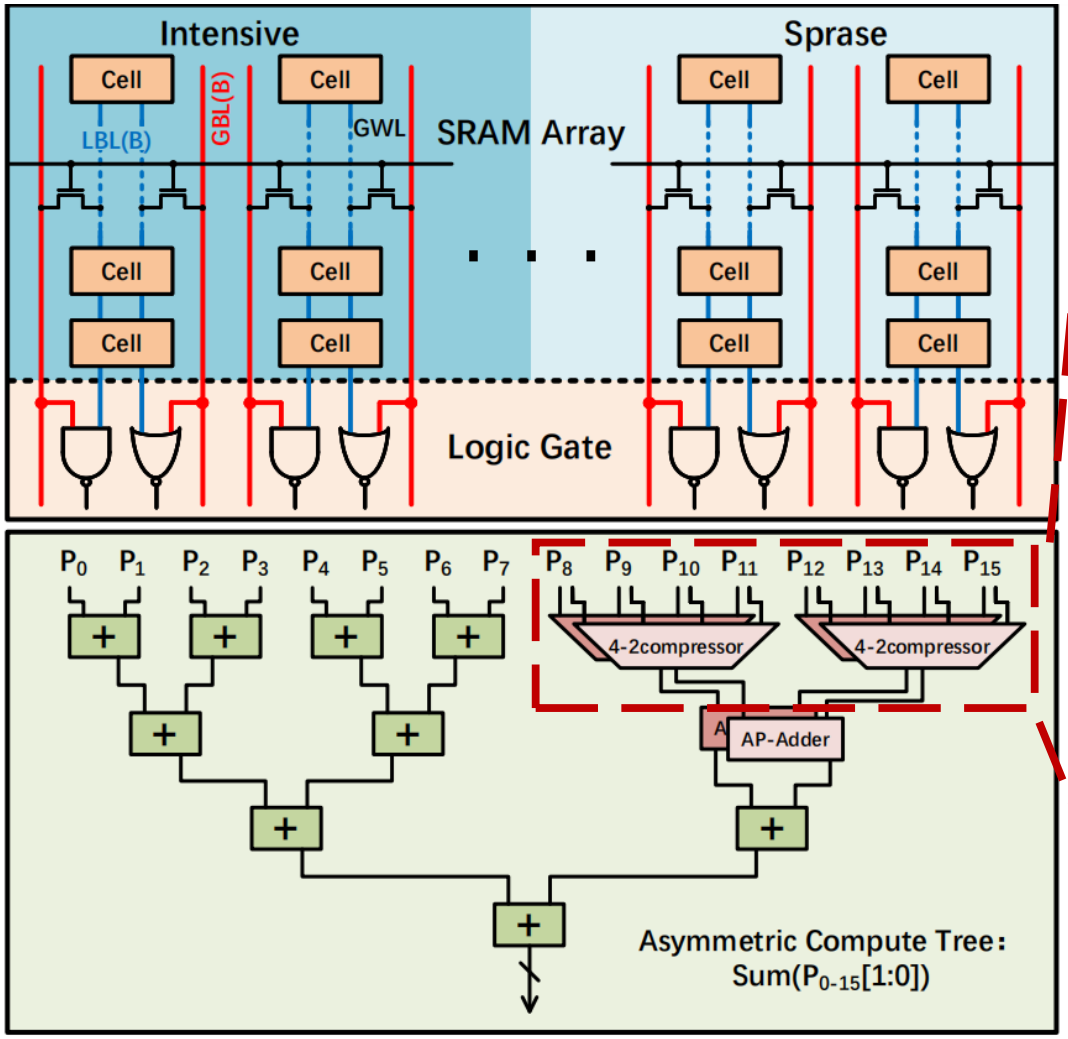
Approximate CT

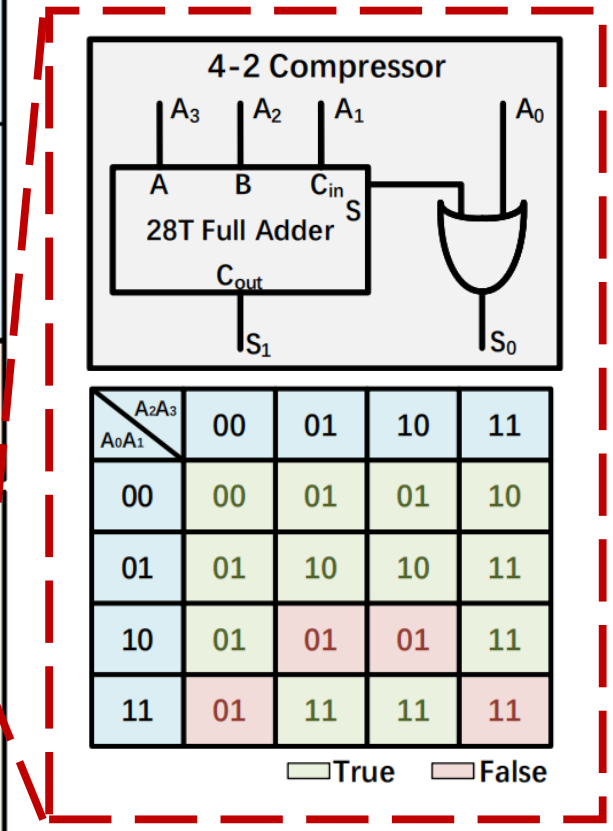
- 4-2 compressors
- approximate 2bit-adders

Asymmetric weight sparsity

Adaptive Asymmetric Compute-Tree Circuit

Adaptive-asymmetric Compute-Tree





A0A1 \ A2A3	00	01	10	11
00	00	01	01	10
01	01	10	10	11
10	01	01	01	11
11	01	11	11	11

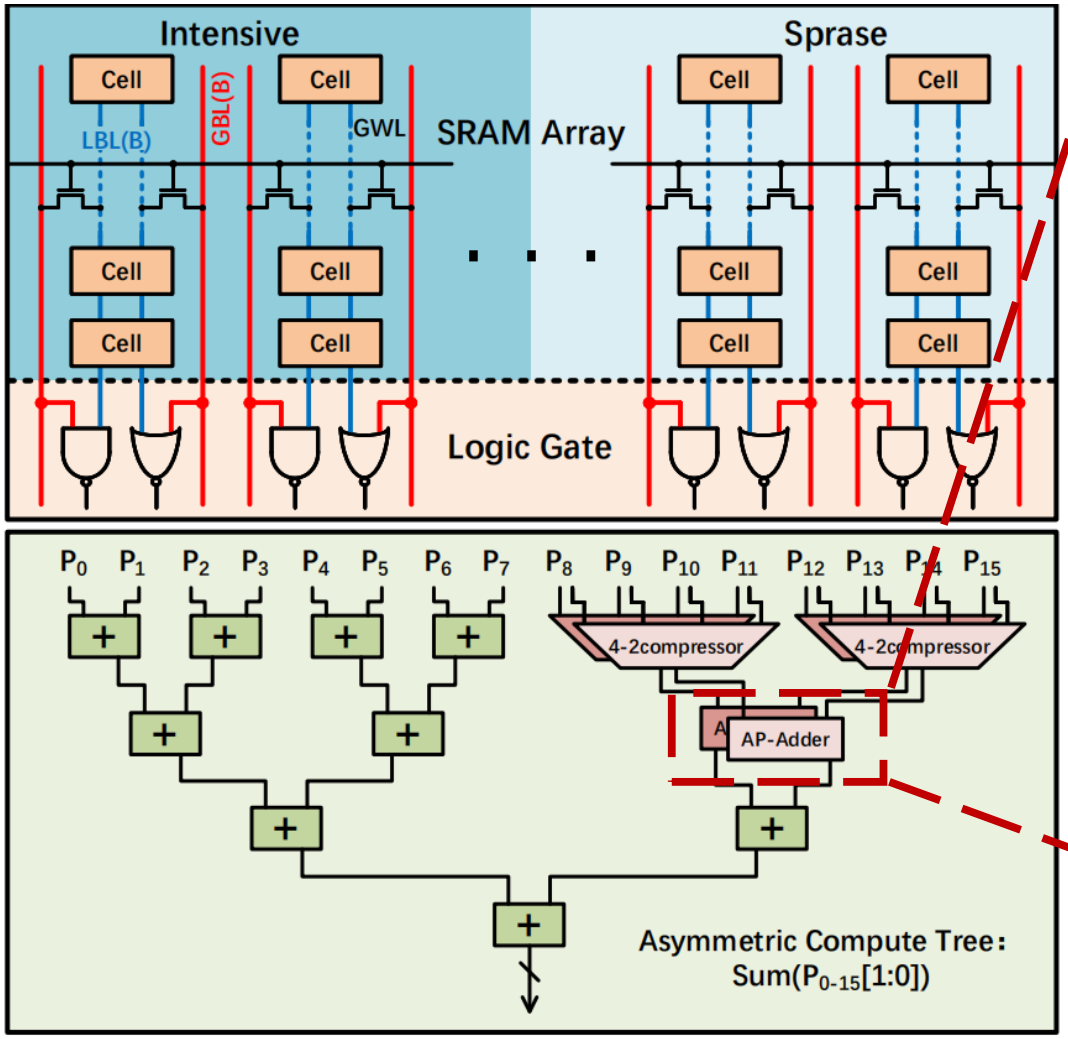
incorrect (A0 = 1)

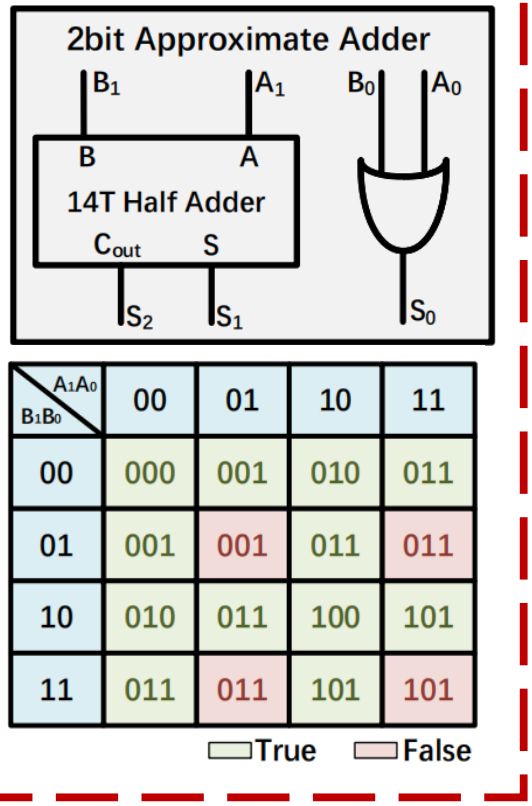
correct (A0 = 0)

4-2 compressor and Truth-table

TrifectaOne

Adaptive-asymmetric Compute-Tree





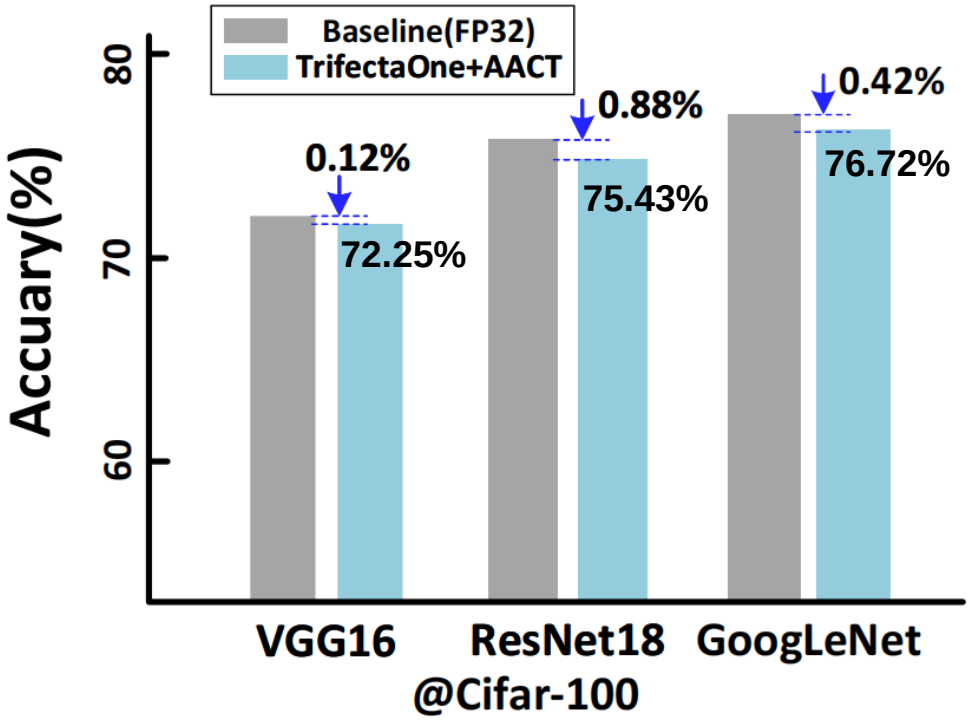
A1A0 / B1B0	00	01	10	11
00	000	001	010	011
01	001	001	011	011
10	010	011	100	101
11	011	011	101	101

True False

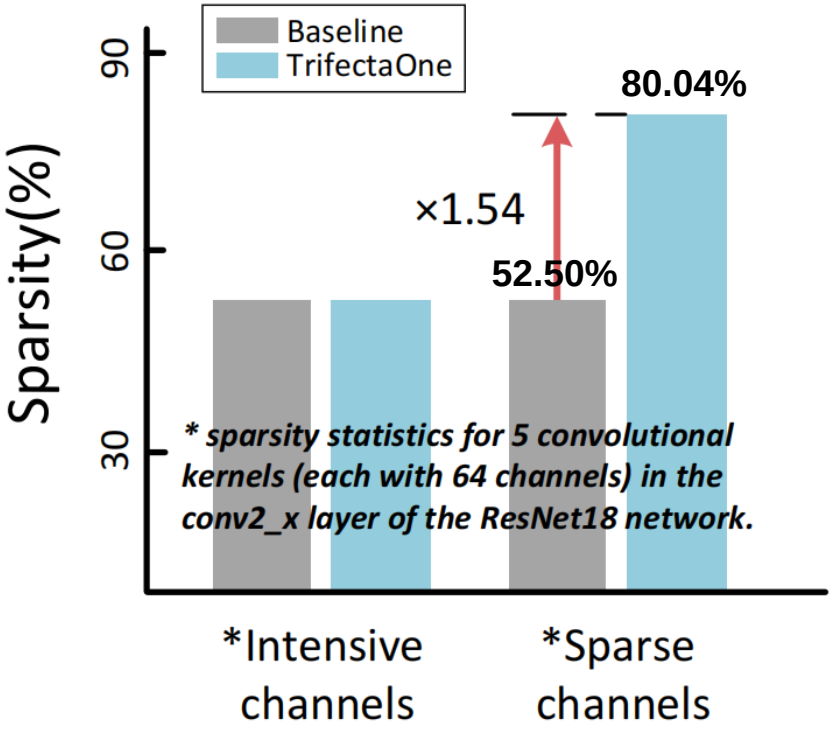
2bit approximate adder and Truth-table

- Save 24T
- Only 1 bit error

Test result of TRIFP-DCIM

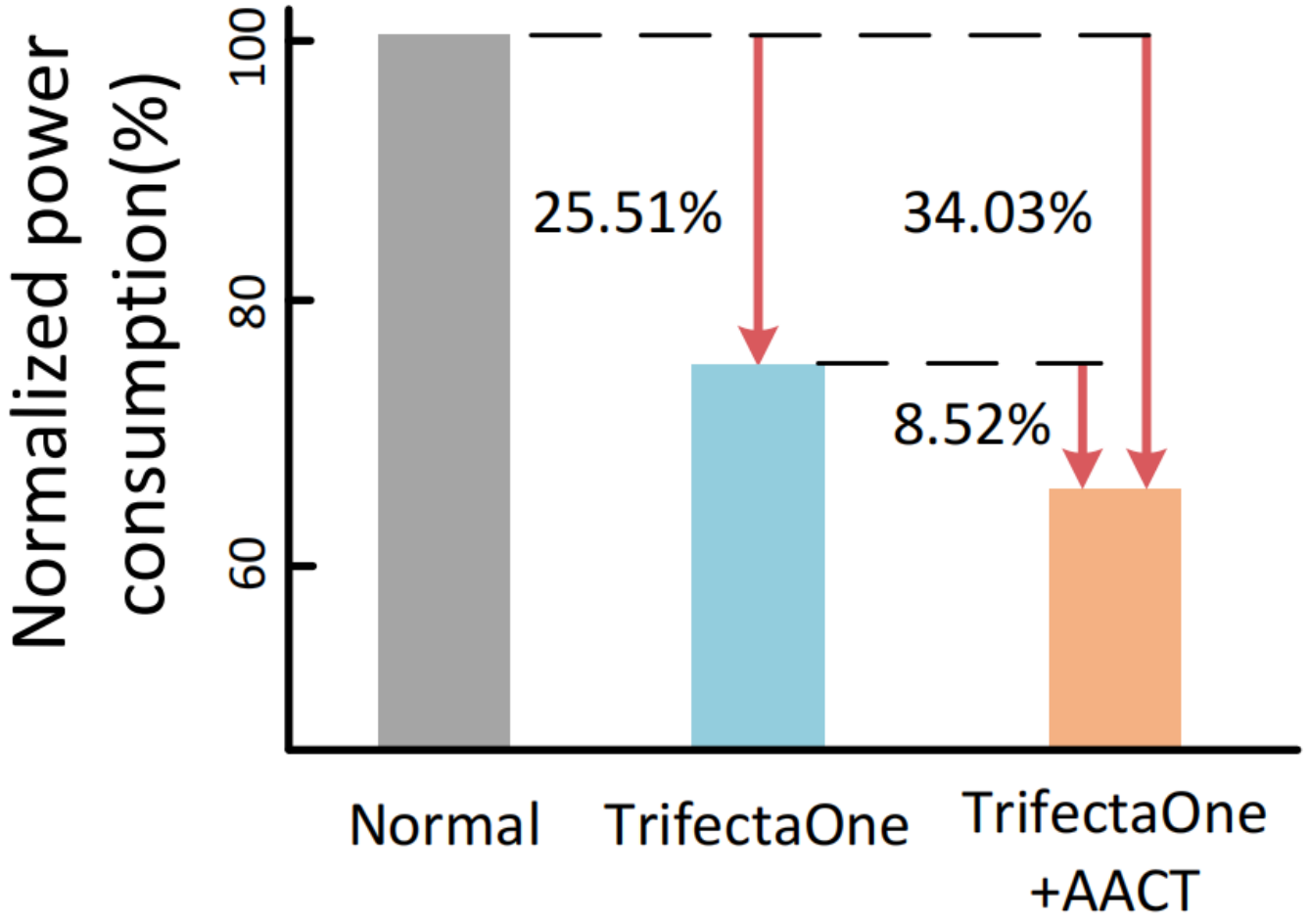


Accuracy test under different networks



Sparsity differences test

Test result of TRIFP-DCIM



Power consumption test of the TrifectaOne alone with AACT

Comparison With Previous Works

	JSSCC'21[1]		DAC'21[2]	ISSCC'23[3]	ISSCC'23[4]		**This Work**
Array size	512×128 (64Kb)		256×64 (16Kb)	832Kb	2×512×128 (128Kb)		**144×512+648×128 (153Kb)**
Technology	28nm		28nm	22nm	28nm		**28nm**
Cell Structure	6T+LCC		8T+Local Peri	6T+DAH-MCA	DBcell+FCU		**6T+AACT**
MAC Operation	Analog		Analog	Hybrid	Digital		**Digital**
Supply Voltage	0.7-0.9		0.5-1.1	0.6-0.8	0.6-0.9		**0.6-0.8**
Precision(bit)	4	8	[1]2/8	BF16	8	BF16	**BF16**
Energy Efficiency (TOPS/W)	47.8	11.5	8.78-36.10	16.22-17.59 TFLOPS/W	19.5-44	14.04-31.6 TFLOPS/W	**[2]14.51-[3]36.83 TFLOPS/W**

[1] The input precision is 2bit, and the weight precision is 8 bit.
[2] Simulation results with FP worst case under 0.8V.
[3] Simulation results with actual ResNet18 model @CIFAR100 under 0.6V.

Combined with the **TrifectaOne method** and **AACT design**, the proposed FP-CIM is simulated under 28nm technology process.

Compared with other works, this work achieves a high energy efficiency of **14.51-36.83 TFLOPS/W** in the digital domain.

Conclusion

TRIFP-DCIM:

28nm digital-domain CIM with **BF16 precision**;

A kind of **Serial Shift Scheme**; → Reduced subtractors and barrel shifters

Adaptive-asymmetric Compute-Tree Circuit;
TrifectaOne algorithm for multiple networks; → Reduced 34.03% power consumption

Achieve **14.51-36.83TFLOPS/W**@BF16.

Reference

[1] Xin Si et al. A local computing cell and 6t sram-based computing-in memory macro with 8-b mac operation for edge ai chips. IEEE Journal of Solid-State Circuits, 56(9):2817–2831, 2021.

[2] Kyeongho Lee et al. A charge-sharing based 8t sram in-memory computing for edge dnn acceleration. In 2021 58th ACM/IEEE Design Automation Conference (DAC), pages 739–744, 2021.

[3] Ping-Chun Wu et al. A 22nm 832kb hybrid-domain floating-point sram in-memory-compute macro with 16.2-70.2tflops/w for high-accuracy ai-edge devices. In 2023 IEEE International Solid-State Circuits Conference (ISSCC), pages 126–128, 2023.

[4] An Guo et al. A 28nm 64-kb 31.6-tflops/w digital-domain floating point-computing-unit and double-bit 6t-sram computing-in-memory macro for floating-point cnns. In 2023 IEEE International Solid-State Circuits Conference (ISSCC), pages 128–130, 2023.

THANK YOU