CIMAT:
A COMPUTE-IN-MEMORY ARCHITECTURE FOR ON-CHIP TRAINING BASED ON TRANSPOSE SRAM ARRAYS

Author : Hongwu Jiang , Xiaochen Peng , Shanshi Huang , and Shimeng Yu , Senior Member

Report Member :
109598061 廖昱翔
109598085 吳承岳

Table of Contents

- Introduction
- Background : DNN Training & Compute-in-Memory
- Hardware Support : 7T, 8T transpose SRAM bit cell design & Periphery Circuit
- CIMAT architecture : FF, BP, Weight Gradient, Weight Update
- Pipeline Design : 7T & 8T SRAM Bit-Cell-Based Pipeline Design

Introduction

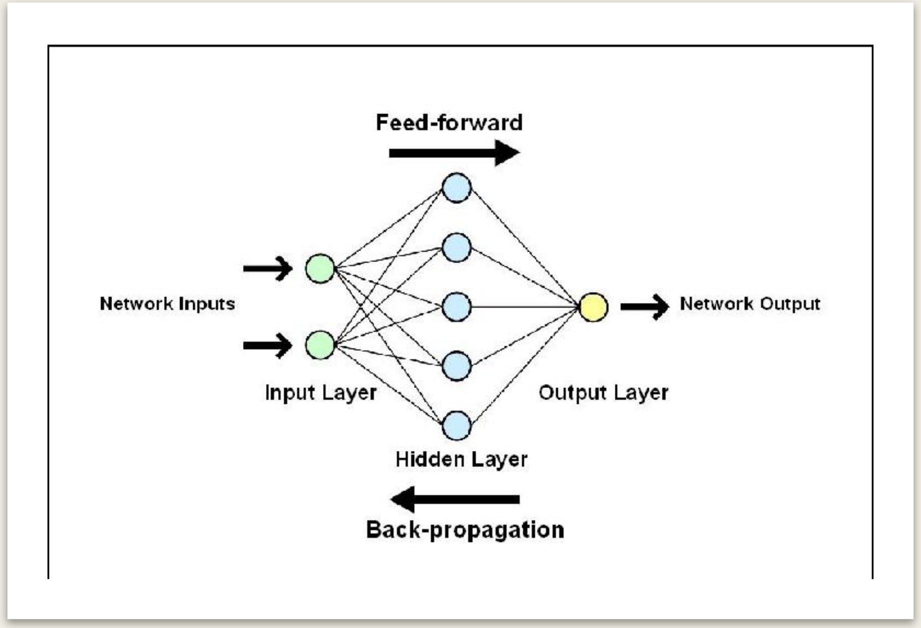
- To increase depth and size so that the accuracy is raised of DNNs,
  need to large computational resource and memory storage for high-precision MAC operation.

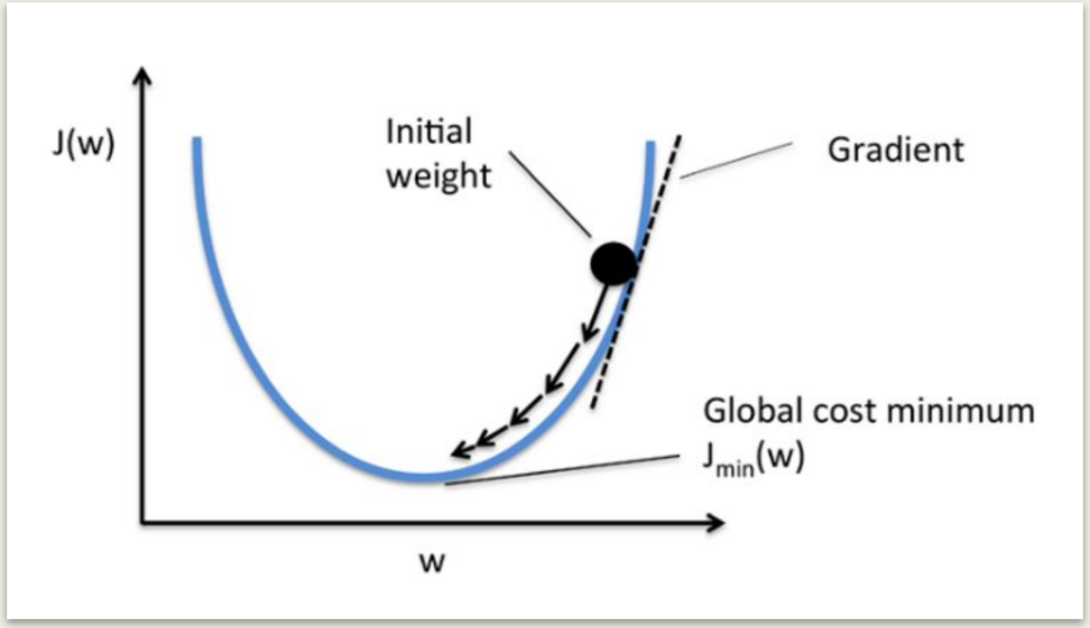
- GPU is popular hardware for DNN training at the cloud,
  but many efforts in design ASIC accelerators for inference or training on–chip.
  We use CMOS ASIC accelerators such as TPU before.

- Parallel computation with optimized data is realized across multiple processing elements (PE),
  but weight & intermediate data need inefficient on–chip or off-chip memory access,
  frequent back and forth data movement is exacerbated for DNN-training,
  CIM can alleviate these drawback, and speed-up DNN hardware.

Background : DNN Training

- The training process of CNN could be divided into four steps:
  - Feed-Forward (FF)
  - Backpropagation (BP)
  - Weight Gradient Calculation
  - Weight update
- The data will divided into two part : training data and inference data.





Background : Compute-in-Memory

- Why use the CIM architecture?
  - Use MAC in both DNN inference and training that access memory and compute memory.
  - Use CIM can solve the problem of weight movement.
  - Challenge: CIM Proposed at present just support DNN inference.

- Why don’t use parallel access?
  - Due to the increased parallelism and reduced data movement,
    CIM is expected to significantly improve the throughput.
  - Because of the limited precision of ADCs and their variations, inference accuracy will degrade.
  - Regular memory can parallel analog read-out with the low-throughput in transpose situation.

- Why use SRAM for CIM?
  - SRAM has been considered as a mature candidate for CIM.
  - Modify the SRAM bit-cell and periphery circuit can enable the parallel access.
    Example: 6T cells into 8T to support bitwise XNOR

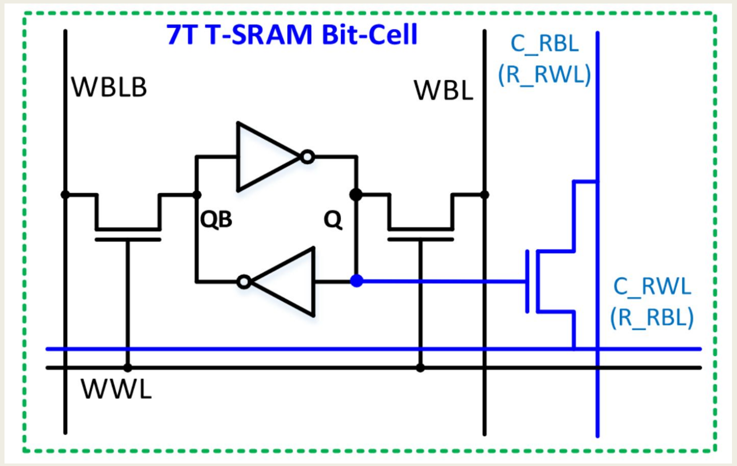
  - CIM base on SRAM can multi-bit inference

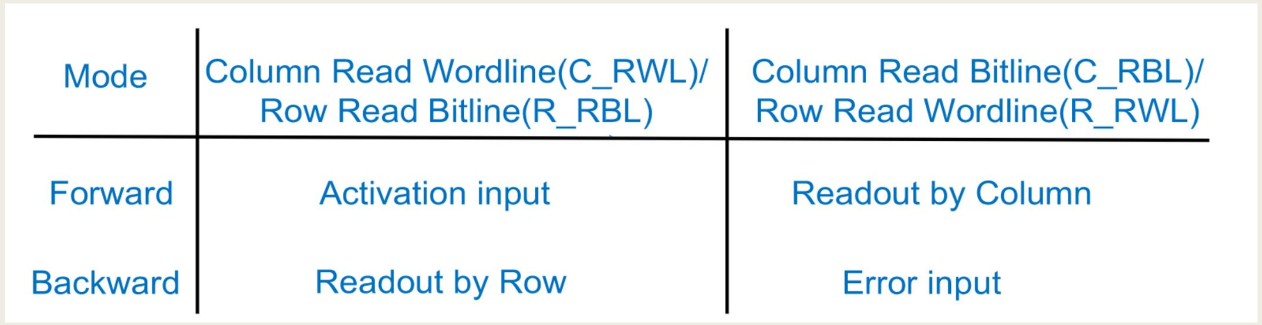
- Propose transpose CIM base on SRAM for multi-bit precision DNN.

Hardware Support : 7T transpose SRAM bit cell design

- Advantage : 1. small cell area overhead
  2. bi-directional overhead
  3. read-disturb-free-access

- black line in the picture: The regular 6T cell is used for the data storage and row-by-row write
- blue line in the picture : on the additional transistor (in blue color) for bi-directional read access.





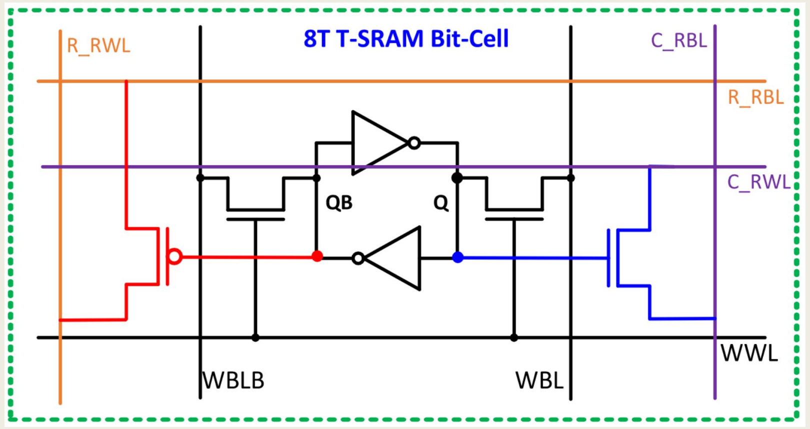
Mode	Column Read Wordline(C_RWL)/ Row Read Bitline(R_RBL)	Column Read Bitline(C_RBL)/ Row Read Wordline(R_RWL)
Forward	Activation input	Readout by Column
Backward	Readout by Row	Error input

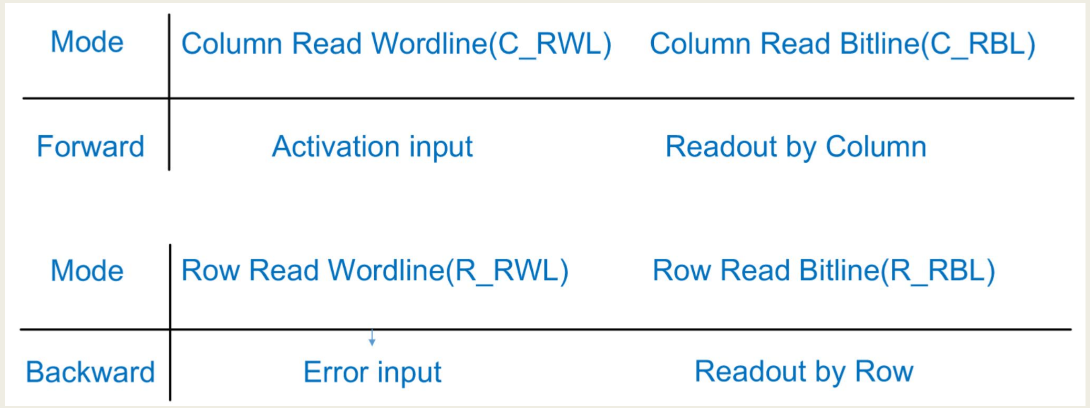
Hardware Support : 8T transpose SRAM bit cell design

- Different with 7T : 1. 7T can bi-direction read access to support both FF and BP,
  but can not access in same cell.
  8T overcomes this problem.
  FF and BP have separate wordline, bit-line, strong point for read access.

  2. add PMOS transistor(to implement bi-directional)

- red line in the picture : add PMOS transistor





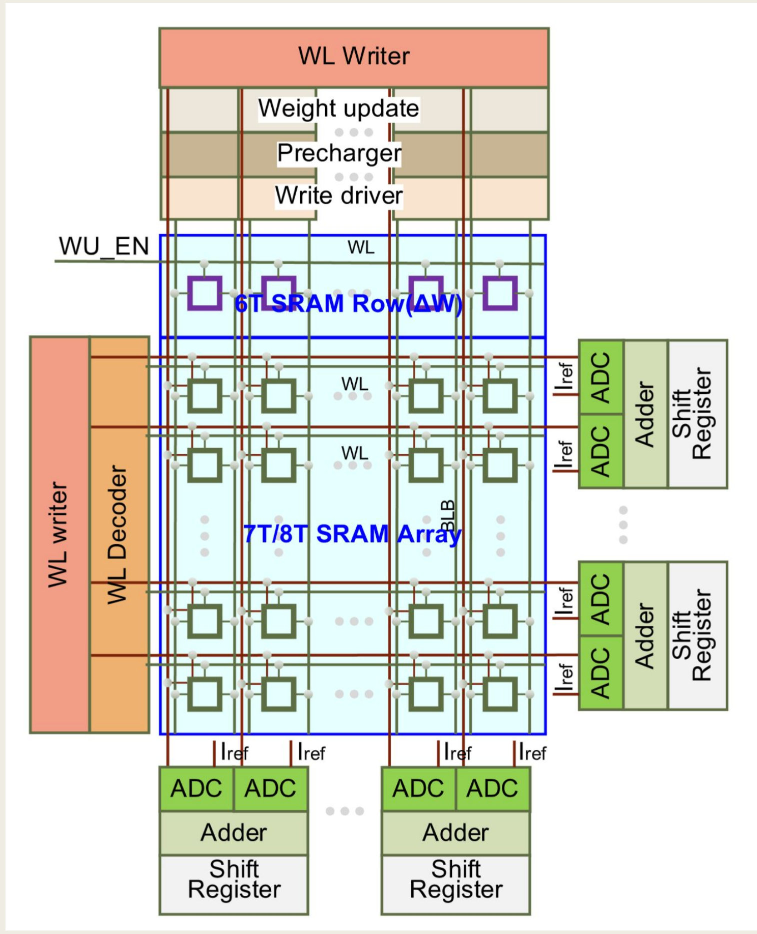
Mode	Column Read Wordline(C_RWL)	Column Read Bitline(C_RBL)
Forward	Activation input	Readout by Column

Mode	Row Read Wordline(R_RWL)	Row Read Bitline(R_RBL)
Backward	Error input	Readout by Row

Hardware Support : Periphery Circuit

- Included : 1. word line(WL) writer[with row and column access].
  2. WL decoders [weight write and per-charge circuit].
  3. flash-ADC [to quantize the partial sum].
  4. Shift and Adder [accumulates digitalized partial sum].
  5. 6T-SRAM Row is use for weight update.

- flash-ADC, Shift, and adder support di-directional access.



CIMAT architecture : Feed-Forward (FF)

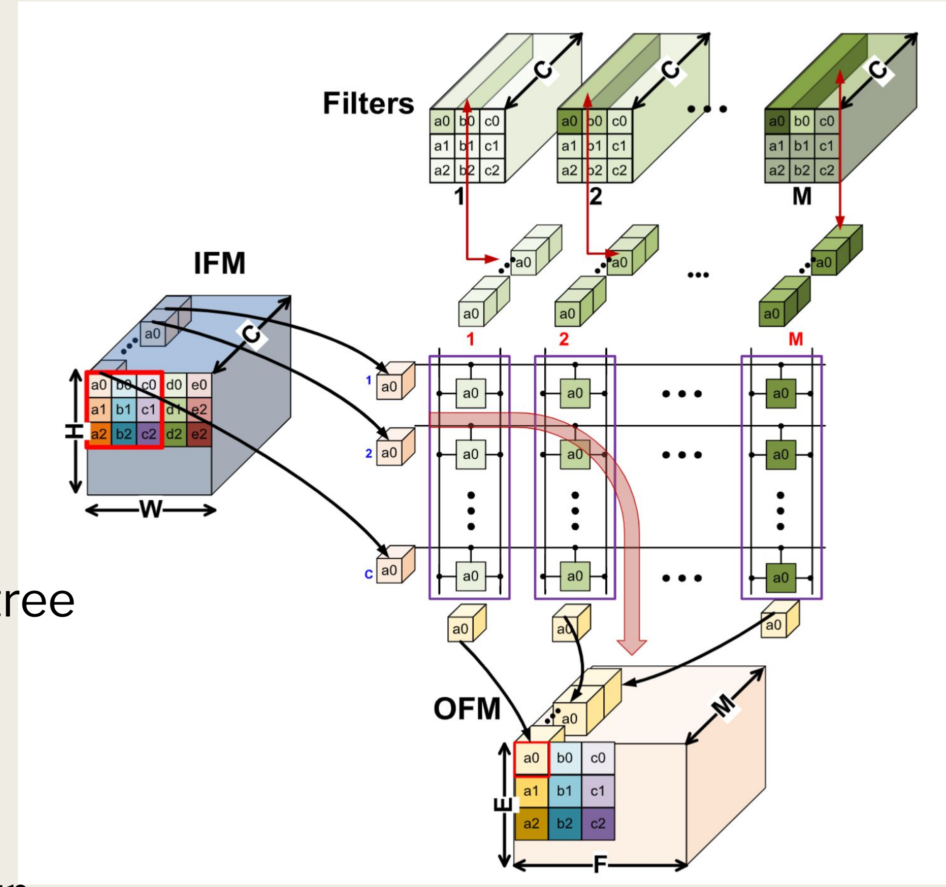
- C is the number of filter channel / IFM(input feature map)
- M is the number of filters / OFM(out feature map) channels
- H/W is the IFM plane height/width
- E/F is the OFM plane height/width.

 Step1.
Put each element in a filter to the same column in a subarray

 Step2.
Use different PEs for different locations of the elements

 Step3.
Sum up the partial sums from different PEs through an adder tree

- For FF,
  the product of input sliding window with the same filter across
  all the channels is summed up to generate one output,
  which means all dot products in same column of the PE is summed up.



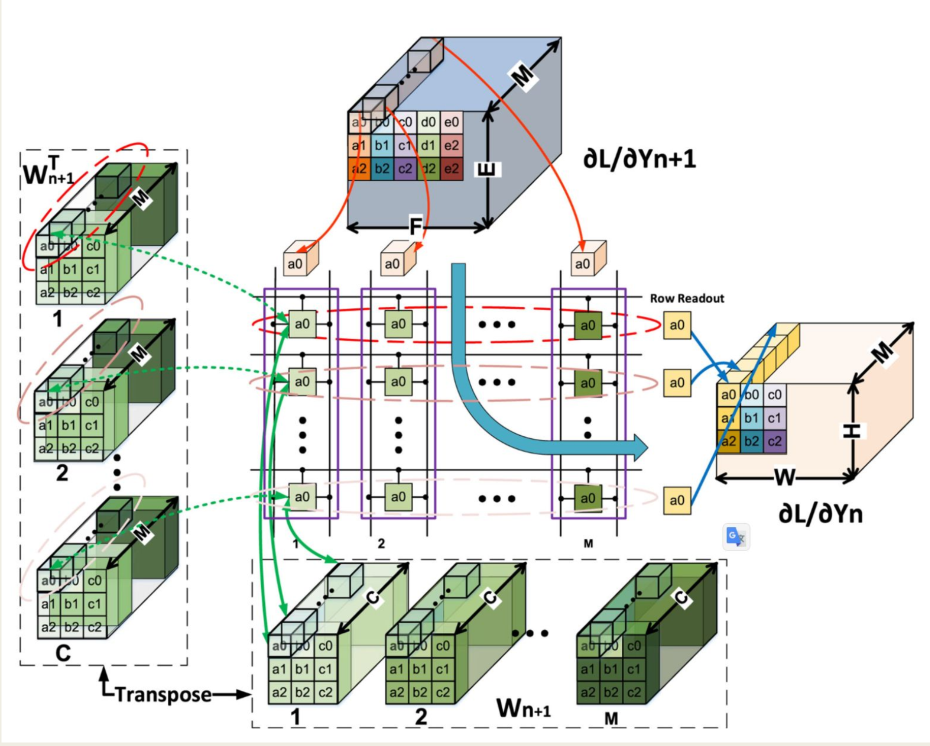
CIMAT architecture : Backpropagation (BP)

$$\frac{\partial L}{\partial Y_n} = \frac{\partial L}{\partial Y_{n+1}} \cdot \frac{\partial Y_{n+1}}{\partial Y_n} = \frac{\partial L}{\partial Y_{n+1}} \cdot W_{n+1}^T$$

- process the transposed version of weight matrix in the BP.
- W_{n+1} is the weights in FF ; W_{n+1}^T is the transposed weights for BP, they are mapped to the same memory array.

- In BP,
  the input vector ($\frac{\partial L}{\partial Y_{n+1}}$) is applied to the column in parallel;
  the output vector ($\frac{\partial L}{\partial Y_n}$) is obtained from the row in parallel.

- FF and error calculation can be performed within the same PE. No additional memory access is needed in error calculation, means an improvement in throughput and energy efficiency.



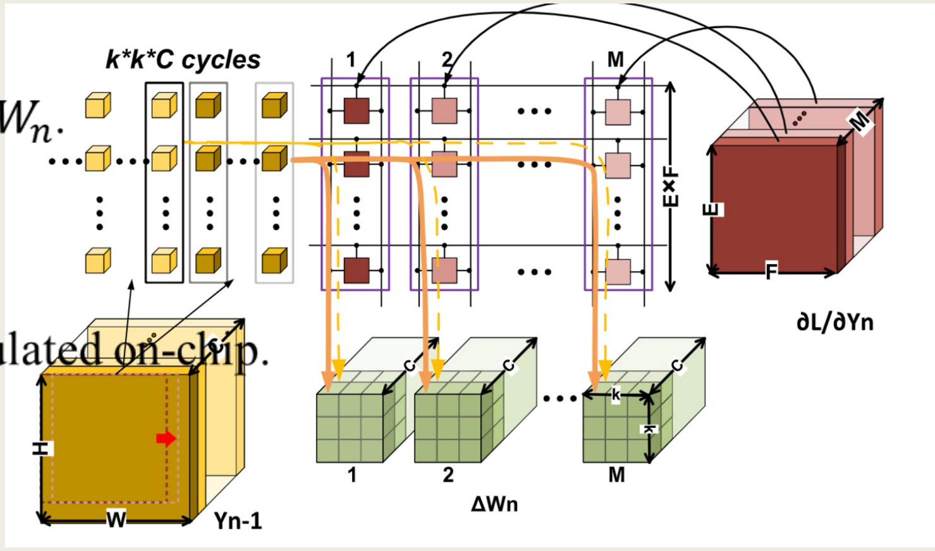
CIMAT architecture : Weight Gradient

- Each plain of $\frac{\partial \mathrm{L}}{\partial \mathrm{Y}_n}$ is stretched into one long column;
  Y_{n-1} are also unrolled to a group of columns;
  The activation columns, k*k , are fed into the array cycle by cycle
  Performing bitwise multiplication and accumulation.

$$\Delta W_n = \frac{\partial L}{\partial W_n} = \frac{\partial L}{\partial Y_n} \cdot \frac{\partial Y_n}{\partial W_n} = \frac{\partial L}{\partial Y_n} \cdot Y_{n-1}$$

- Partial sums from same column with multiple cycles→
  all the weight gradients of one filter→
  partial sums from different columns form the entire gradient matrix ΔW_n.

- In the normal batch training mode,
  $\Delta\, W_n$ of each image is sent to off-chip DRAM for storing
  at the end of each batch, weight gradients are loaded back and accumulated on-chip.



CIMAT architecture : Weight Update

- The 6T SRAM row is disabled during the FF and BP.
- When weight gradients are ready, they are fed into shift register to realize multiplication with learning rate.

$$W_n^t = W_n^{t-1} - LR \times \frac{\partial L}{\partial W_n}.$$

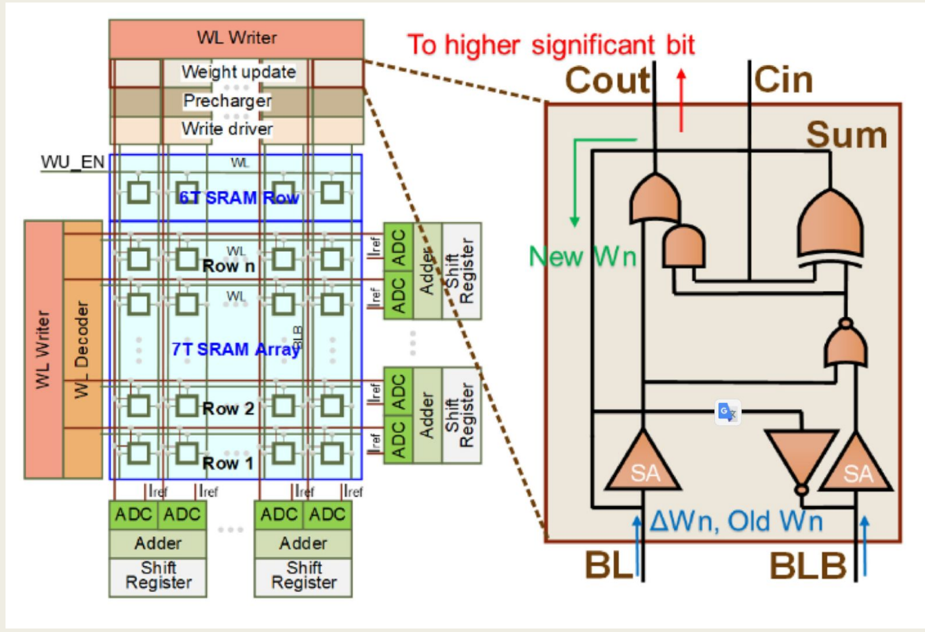
☐ Step1.
One row of the ΔW_n matrix is written into the 6T row of the sub-array.

☐ Step2.
WLs of 6T row and the to-be-updated weight row are activated to send stored information to the weight update module.

☐ Step3.
The weight update module an adder of the sum of ΔW_n and W_n.

☐ Step4.
Obtain the new W_n,
written back to the weight row and
$C_{out}$of the adder is forwarded to the higher significant bit

- The entire process of weight update is completed when all the multi-bit weights, from LSB to MSB, are updated.



7T SRAM Bit-Cell-Based Pipeline Design

1. The latency of the 1st to 5th layers to process an entire image is almost the same.
2. latency of 6th to 17th convolution layers is only half of the previous layers.

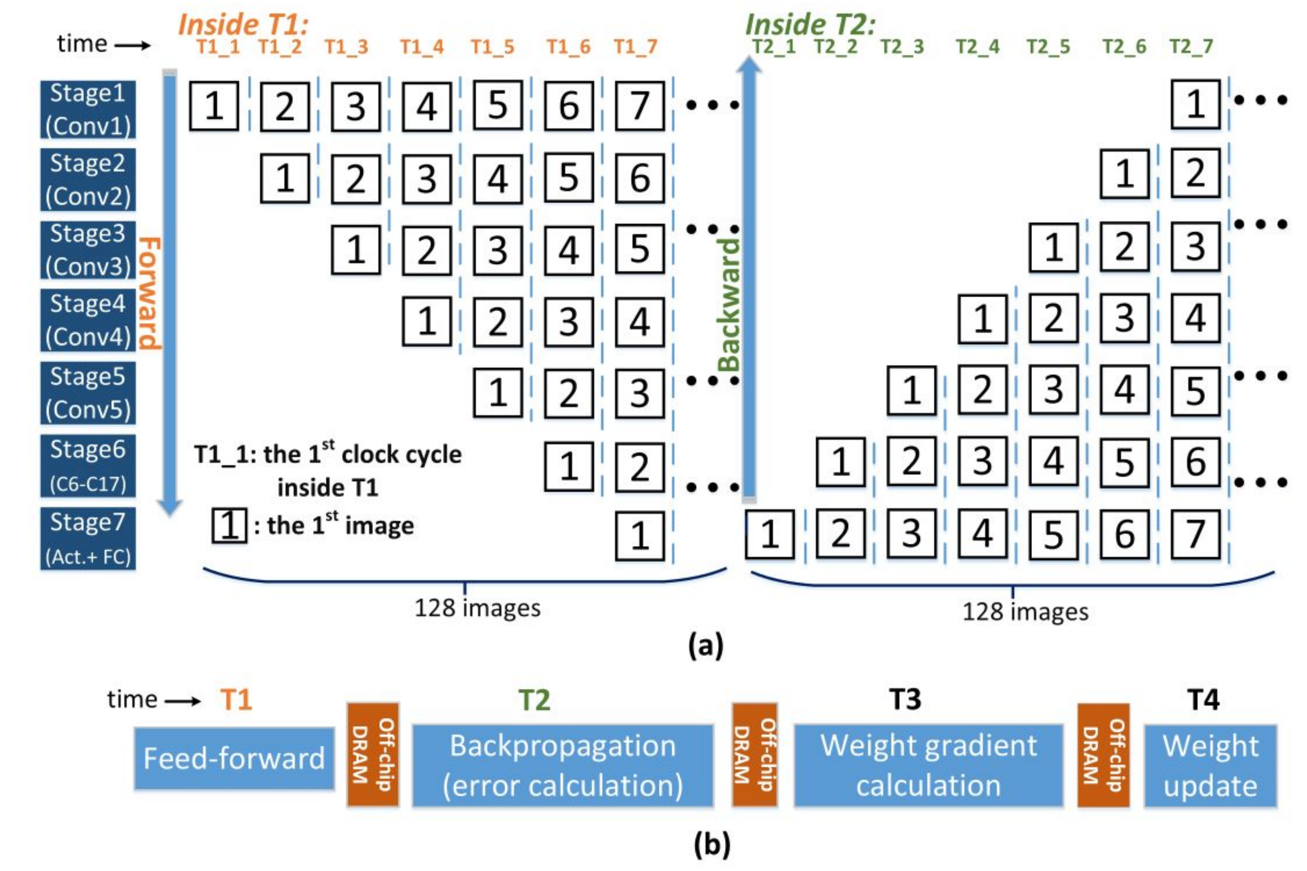
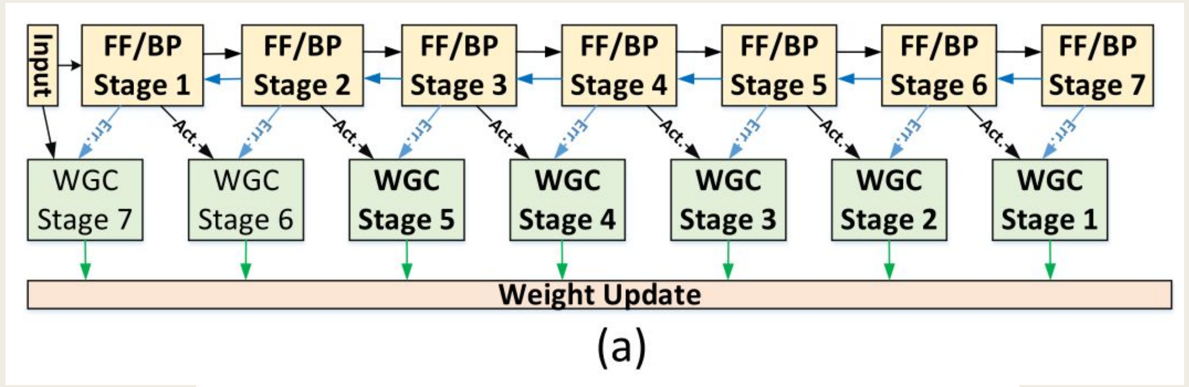
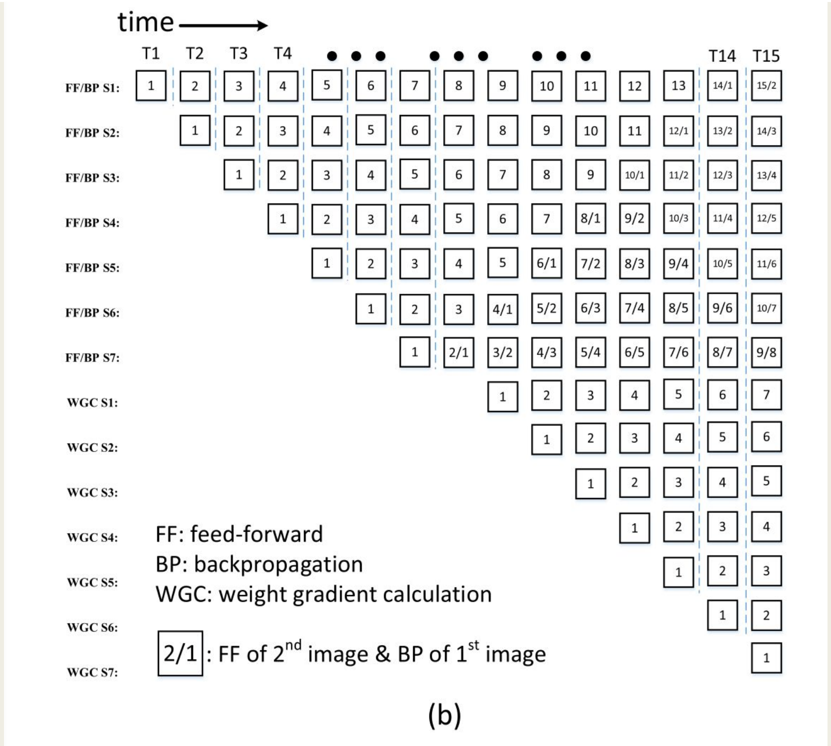


Fig. 9. (a) Intra-pipeline design inside FF and BP. (b) Training process in timeline of 7T-based architecture.

8T SRAM Bit-Cell-Based Pipeline Design




time →

	T1	T2	T3	T4	• • •			• • •			• • •			T14	T15
FF/BP S1:	1	2	3	4	5	6	7	8	9	10	11	12	13	14/1	15/2
FF/BP S2:		1	2	3	4	5	6	7	8	9	10	11	12/1	13/2	14/3
FF/BP S3:			1	2	3	4	5	6	7	8	9	10/1	11/2	12/3	13/4
FF/BP S4:				1	2	3	4	5	6	7	8/1	9/2	10/3	11/4	12/5
FF/BP S5:					1	2	3	4	5	6/1	7/2	8/3	9/4	10/5	11/6
FF/BP S6:						1	2	3	4/1	5/2	6/3	7/4	8/5	9/6	10/7
FF/BP S7:							1	2/1	3/2	4/3	5/4	6/5	7/6	8/7	9/8
WGC S1:									1	2	3	4	5	6	7
WGC S2:										1	2	3	4	5	6
WGC S3:											1	2	3	4	5
WGC S4:												1	2	3	4
WGC S5:													1	2	3
WGC S6:														1	2
WGC S7:															1

FF: feed-forward
BP: backpropagation
WGC: weight gradient calculation

$\boxed{2/1}$: FF of 2nd image & BP of 1st image

(b)

Fig. 10. (a) Pipeline structure of 8T SRAM-based CIM training. (b) The state of each stage as a function of time. 15 super clock cycles are illustrated.