FP-BNN: Binarized neural network on FPGA

2018-11-10

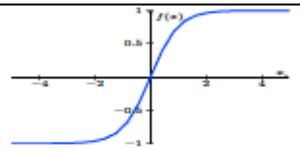
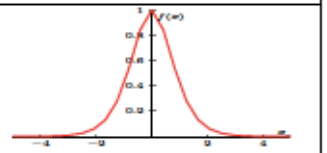
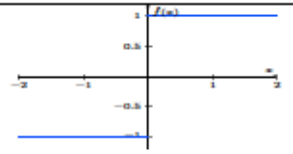
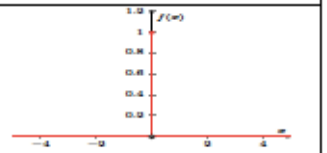
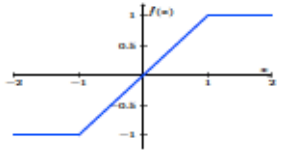
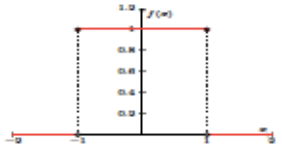
Overview

- 2017

- 通过优化度量将给定的BNN部署到FPGA上

- A datapath design with multipliers replaced by XNOR, popcount and shifting operations for BNNs, and a compression tree generation method for more efficient popcount.
- An optimized data managing pattern with parameter quantization and on-chip storage strategy.

Hardware logic design

Operation	Function plots	Derivative plots
$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$		
$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$		
$HTanh(x) = \begin{cases} +1 & x > 1 \\ x & -1 \leq x \leq 1 \\ -1 & x < -1 \end{cases}$		

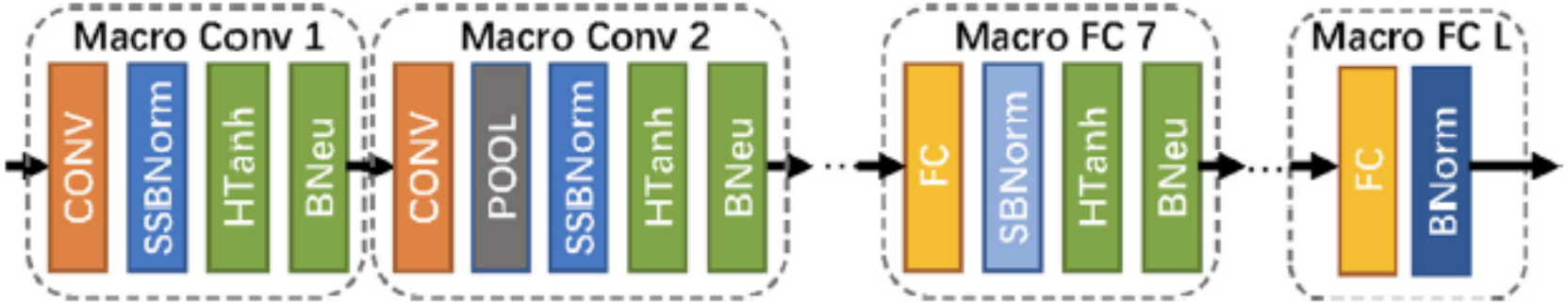


Fig. 3. A normal structure of a BNN model.

Overall architecture

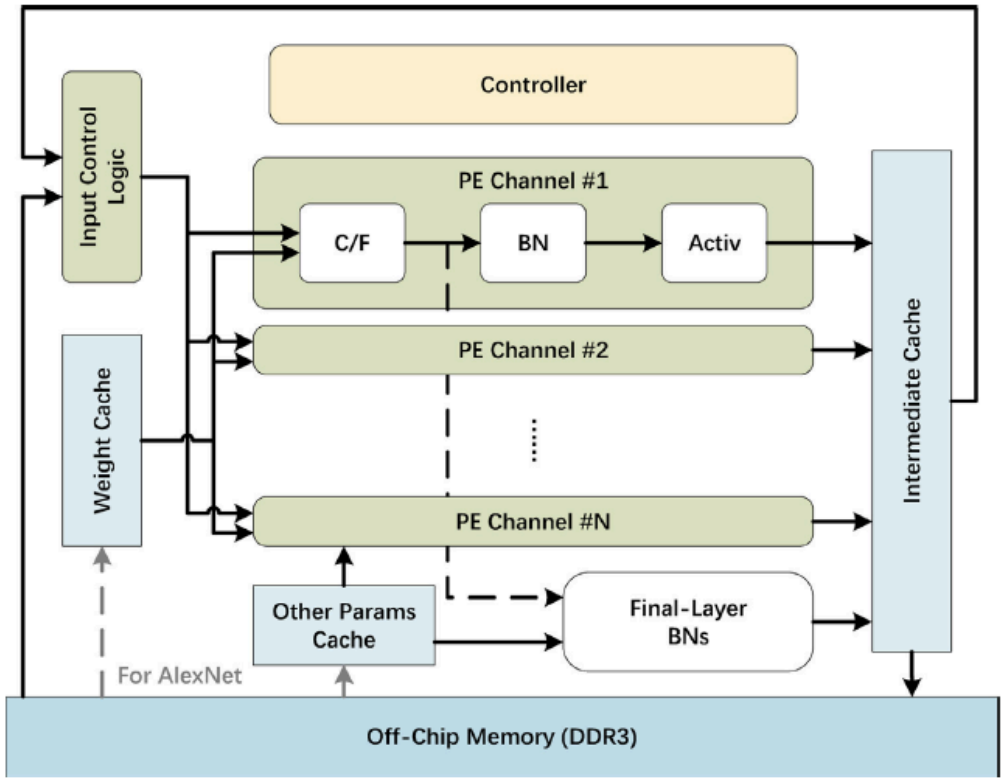


Fig. 4. The overall system architecture design.

We have altogether *N PE* channels to process in parallel the data from the input cache.

CONV/FC (*C/F*) layer includes processing elements (PEs) that are shared by the CONV and FC since they both mainly consist of MAC computations.

Shift-Based Normalization (SBN) layer adopts shift operations to replace multiplications as mentioned in Section 2.3 .

Overall architecture

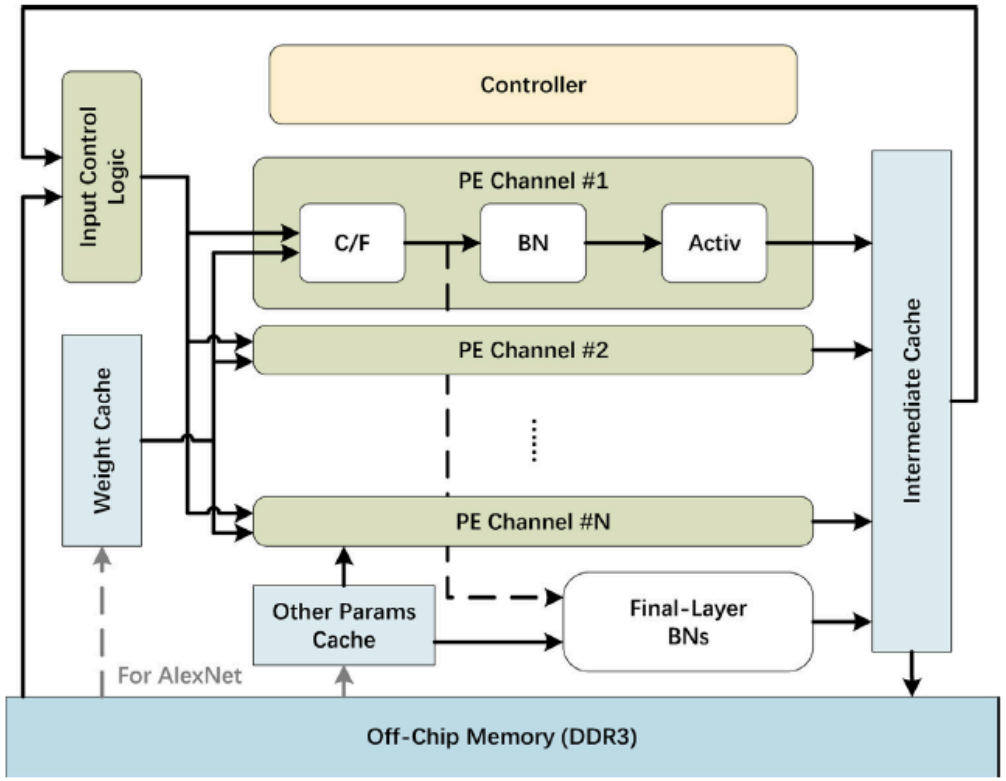


Fig. 4. The overall system architecture design.

Activation layer merges the HTanh and BNeu layers together to produce an output vector containing either 0 or 1.

Parameters for each layer are **fetched from on-chip** BRAMs or registers to meet bandwidth requirements, and control signals select them for each iteration.

The output for each iteration will be transferred to the intermediate result cache.

For each next layer, the interconnection will be reconfigured by the controller according to the type (CONV or FC) of the layer.

C/F PE

XNOR-based Binary MAC

Hardware implementations usually take 2 bits to represent +1 and −1. **If we use only one bit, we should take 0 and 1 as the basic values.** This can be achieved through *affine transformation* .

$$\mathbf{A}_{\langle 0,1\rangle} = \frac{\mathbf{A}_{\langle -1,1\rangle} + \mathbf{A}_{\langle 1\rangle}}{2}$$

Table 5
Truth table of affine transformed inputs and result.

Original multiplication			Affine transformed		
$a_{\langle -1,1\rangle}$	$b_{\langle -1,1\rangle}$	$a \cdot b_{\langle -1,1\rangle}$	$a_{\langle 0,1\rangle}$	$b_{\langle 0,1\rangle}$	$a \cdot b_{\langle 0,1\rangle}$
1	1	1	1	1	1
1	−1	−1	1	0	0
−1	1	−1	0	1	0
−1	−1	1	0	0	1

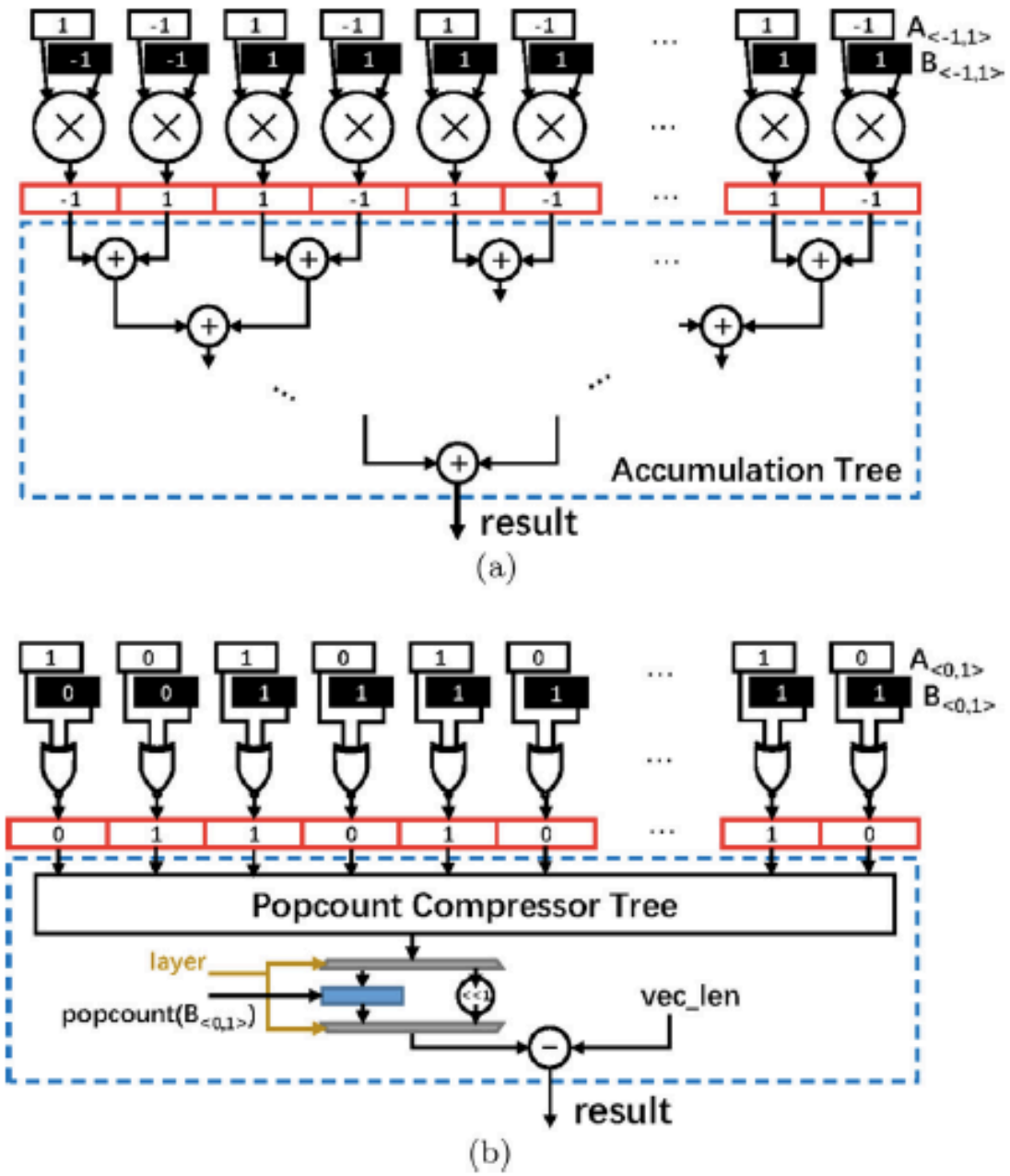


Fig. 5. Conversion from (a) ⟨−1, 1⟩-based MAC to (b) ⟨0, 1⟩-based XNOR and popcount operations.

XNOR-based Binary MAC

of vector $\mathbf{A}_{\langle -1,1\rangle}$ and $\mathbf{B}_{\langle -1,1\rangle}$ of length *vec_len*, then we will have

$$
\begin{aligned}
result &= \mathbf{A}_{\langle -1,1\rangle} \cdot \mathbf{B}_{\langle -1,1\rangle} = \sum_{i=1}^{vec\_len} a_{i\langle -1,1\rangle} \cdot b_{i\langle -1,1\rangle} \\
&= \sum_{i=1}^{vec\_len} \left[(a_{i\langle -1,1\rangle} \cdot b_{i\langle -1,1\rangle}) - (-a_{i\langle -1,1\rangle} \cdot b_{i\langle -1,1\rangle}) \right] \\
&= \sum_{i=1}^{vec\_len} \left[XNOR(a_{i\langle 0,1\rangle}, b_{i\langle 0,1\rangle}) - XOR(a_{i\langle 0,1\rangle}, b_{i\langle 0,1\rangle}) \right] \\
&= 2\,popcount(\mathbf{R}_{\langle 0,1\rangle}) - vec\_len \qquad (16)
\end{aligned}
$$

in which

$$
\mathbf{R}_{\langle 0,1\rangle} = \{XNOR(a_{i\langle 0,1\rangle}, b_{i\langle 0,1\rangle}), i = 1\ to\ vec\_len\} \qquad (17)
$$

If one of the inputs is already $\langle 0, 1\rangle$ based, for example, the first layer, then we get the result with:

$$
\begin{aligned}
result &= \mathbf{A}_{\langle 0,1\rangle} \cdot \mathbf{B}_{\langle -1,1\rangle} = \frac{\mathbf{A}_{\langle -1,1\rangle} + \mathbf{A}_{\langle 1\rangle}}{2} \cdot \mathbf{B}_{\langle -1,1\rangle} \\
&= \frac{2\,popcount(\mathbf{R}_{\langle 0,1\rangle}) - vec\_len}{2} + \frac{\sum b_{i\langle -1,1\rangle}}{2} \\
&= popcount(\mathbf{R}_{\langle 0,1\rangle}) - vec\_len + \sum b_{i\langle 0,1\rangle} \qquad (18)
\end{aligned}
$$

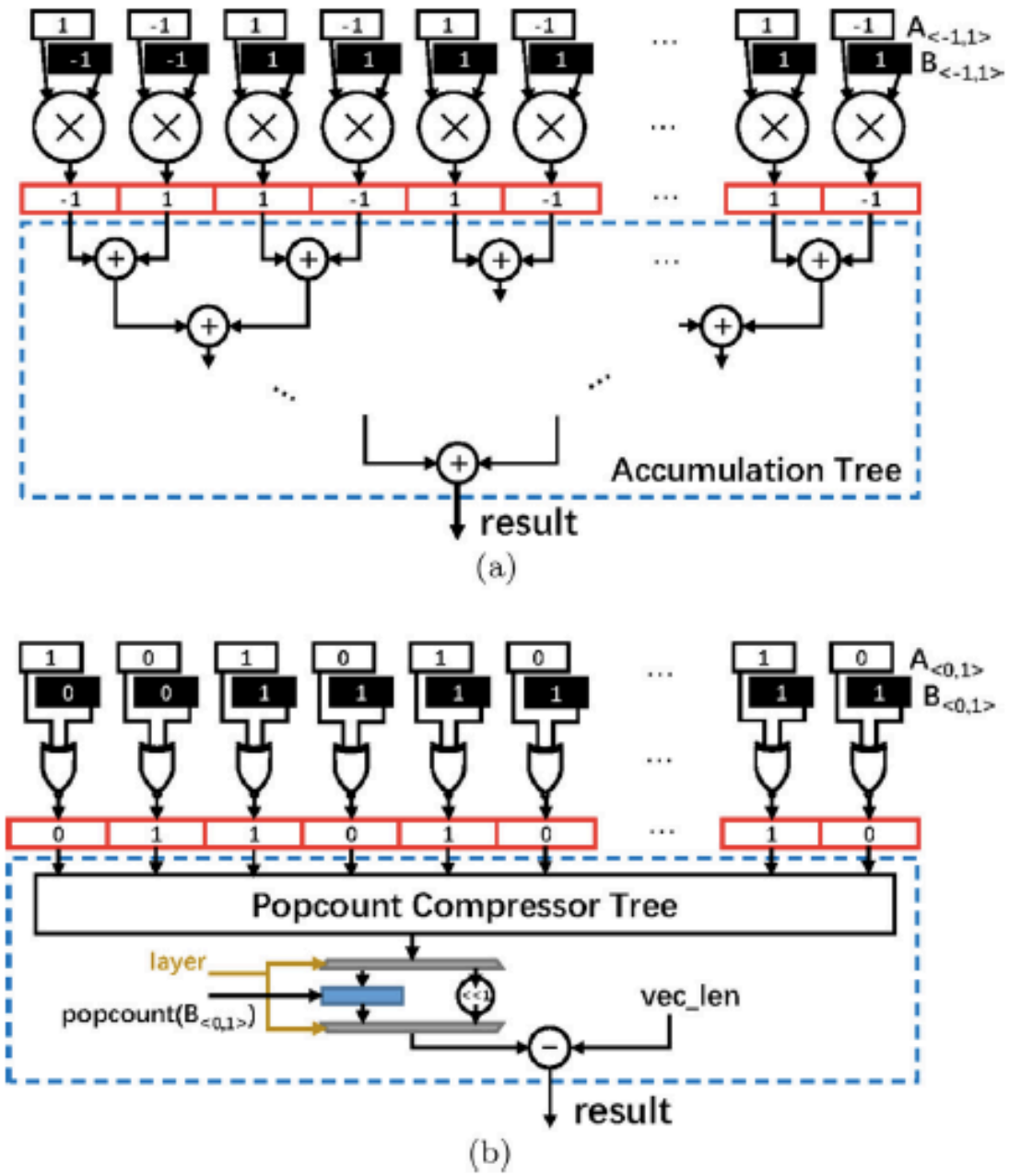


Fig. 5. Conversion from (a) $\langle -1, 1\rangle$-based MAC to (b) $\langle 0, 1\rangle$-based XNOR and popcount operations.

Popcount Compressor (PC) tree

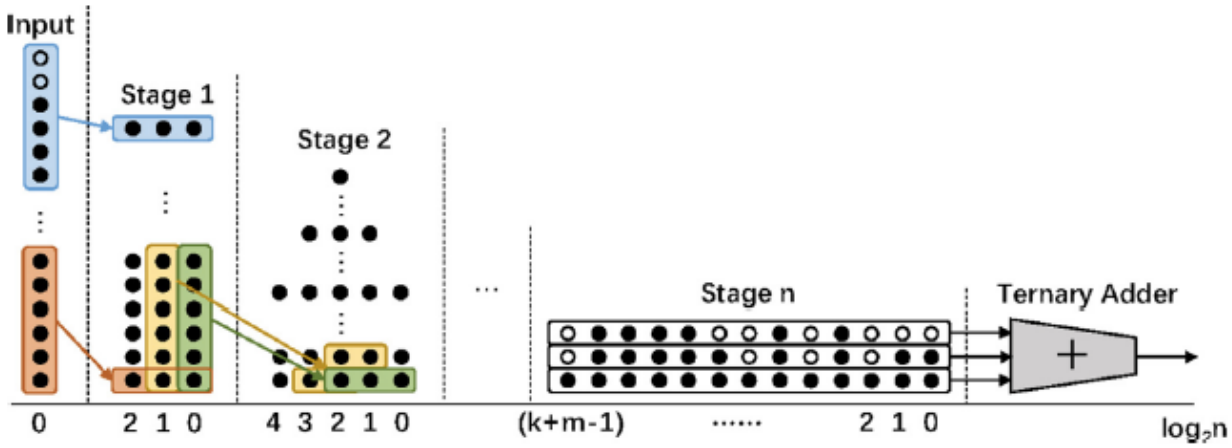


Fig. 6. The popcount compressor tree based on 6:3 compressors and one ternary adder.

Algorithm 2 Popcount compressor tree generation algorithm.

Require: Input vector: **i** of height N
Ensure: Updated: Column vector height $h(i, j)$, i stands for the weight of 2^i and j for the compression stage; Heap of stage j: $\mathcal{H}(j) = \{h(k, j)\}, k = 0, 1, \ldots, log_2(N)$.
$h(0, 0) = N, i = 0, j = 0;$
while $max(\mathcal{H}(j)) > 3$ **do**
  $\mathcal{H}(j+1) = zeros(1, log_2(N));$
  for $k = 1$ to $log_2(N)$ **do**
    if $h(k, j) > 3$ **then**
      $n_{compressor}(k, j) = \lceil h(k, j)/6 \rceil;$
      $h(k, j+1) = h(k, j+1) + n_{compressor}(k, j);$
      $h(k+1, j+1) = h(k+1, j+1) + n_{compressor}(k, j);$
      $h(k+2, j+1) = h(k+2, j+1) + n_{compressor}(k, j);$
    else
      $h(k, j+1) = h(k, j+1) + h(k, j);$
    end if
  end for
  $j = j + 1;$
end while

Table 6
Comparison between accumulation adder tree and popcount compressor tree.

BW_{in} (bits)	BW_{our} (bits)	LUTs		
		Acc.	Pop.	Saved (%)
9 (3^2)	4	9	10	−11.1
16	5	21	19	9.52
64	7	98	79	19.39
256	9	398	291	26.88
1024	11	1596	1106	30.70
1152 (128×3^2)	11	1796	1228	31.63
1200 (48×5^2)	11	1864	1282	31.22
8192	14	12768	8362	34.51

PE reuse

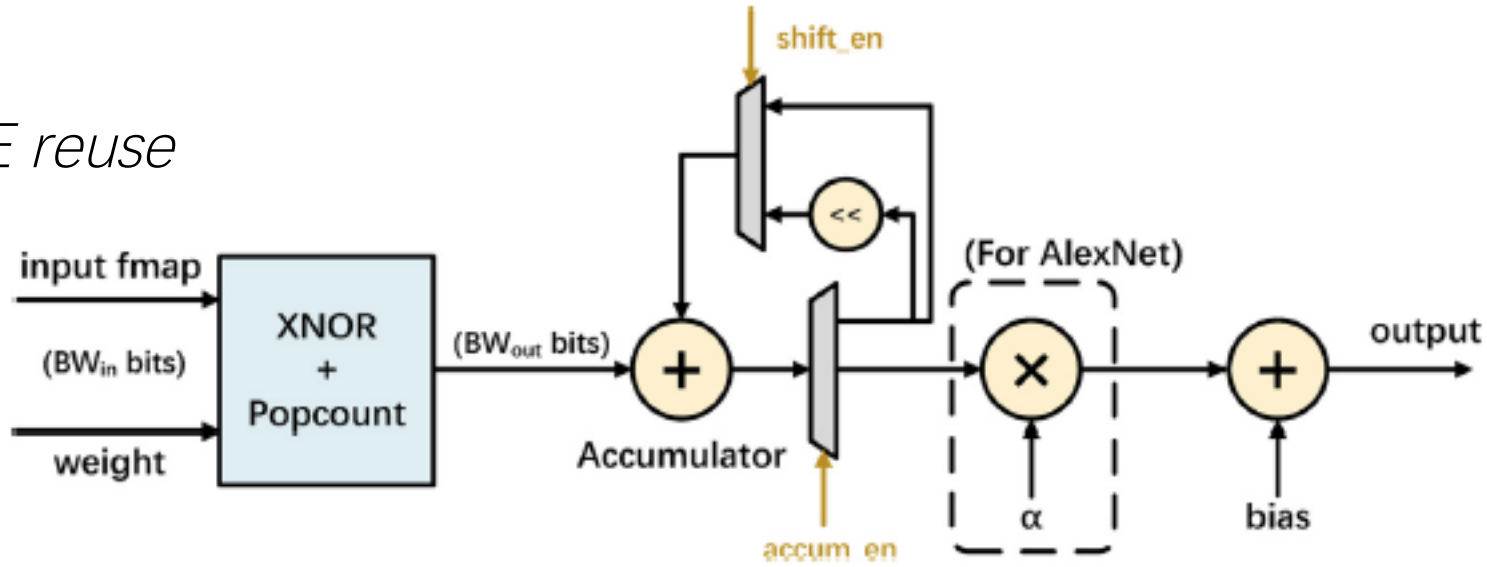


Fig. 7. The C/F layer PE module.

ers' weights are not binarized. To deal with this, if a vector **x** of n fixed-point inputs with m-bit precision:

$$\mathbf{x} = (\overline{x_{n-1}^{m-1} x_{n-1}^{m-2} \ldots x_{n-1}^{0}}, \overline{x_{n-2}^{m-1} x_{n-2}^{m-2} \ldots x_{n-2}^{0}}, \ldots, \overline{x_{0}^{m-1} x_{0}^{m-2} \ldots x_{0}^{0}}) \tag{19}$$

and a vector **w** of n p-bit weights:

$$\mathbf{w} = (\overline{w_{n-1}^{p-1} w_{n-1}^{p-2} \ldots w_{n-1}^{0}}, \overline{w_{n-2}^{p-1} w_{n-2}^{p-2} \ldots w_{n-2}^{0}}, \ldots, \overline{w_{0}^{p-1} w_{0}^{p-2} \ldots w_{0}^{0}}) \tag{20}$$

then the output vector **s** could be calculated by

$$\mathbf{s} = \mathbf{x} \cdot \mathbf{w} = \sum_{i=1}^{p} 2^{i-1} \sum_{j=1}^{m} 2^{j-1} \sum_{k=1}^{n} (x_{k-1}^{j-1} \cdot w_{k-1}^{i-1}) \tag{21}$$

BN PE

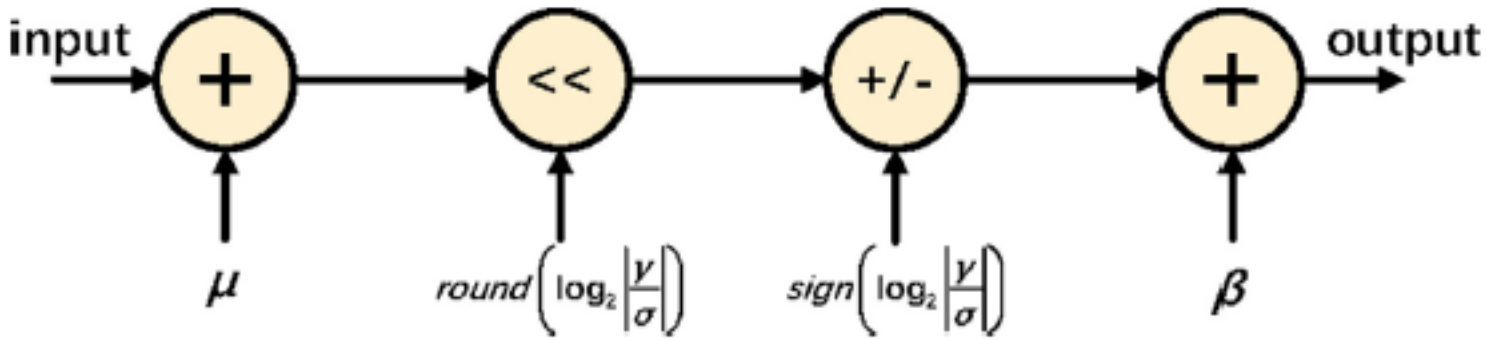


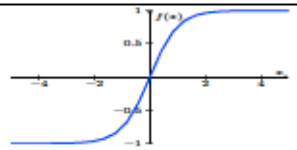
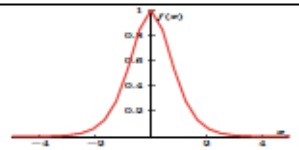
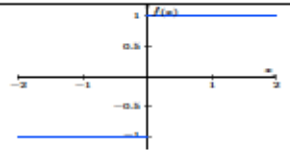
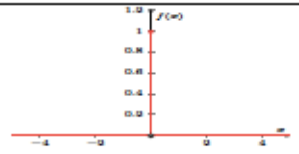
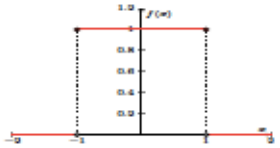
Fig. 8. The SBN layer PE module (MNIST and Cifar-10).

$$y = \frac{x-\mu}{\sigma} \cdot \gamma + \beta, \tag{22}$$

$$y = sal[(x-\mu), \phi] \cdot sign\left|\frac{\gamma}{\sigma}\right| + \beta, \tag{23}$$

where $\phi = round(\log_2\left|\frac{\gamma}{\sigma}\right|)$ is the left-shift value of both σ and γ.

Activation PE

Operation	Function plots	Derivative plots
$Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$		
$sign(x) = \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$		
$HTanh(x) = \begin{cases} +1 & x > 1 \\ x & -1 \leq x \leq 1 \\ -1 & x < -1 \end{cases}$		

Pooling

- A C-P-B-A macro- layer structure is taken. However, for the inference process, a C-B- A-P structure can get an identical result and the pooling is applied to values of 0 and 1 only. **This can be directly implemented with *OR* operations**.

Task tiling and scheduling (T&S)

Table 7
Tiling strategy for different models.

Model	N_{PE}	PE_{size}	L_{in} of layer								
			1	2	3	4	5	6	7	8	9
MNIST		1024	784		1024						
Cifar-10	64	1152	405			1152				1024	
AlexNet		1200	1089	1200		1152			1024		

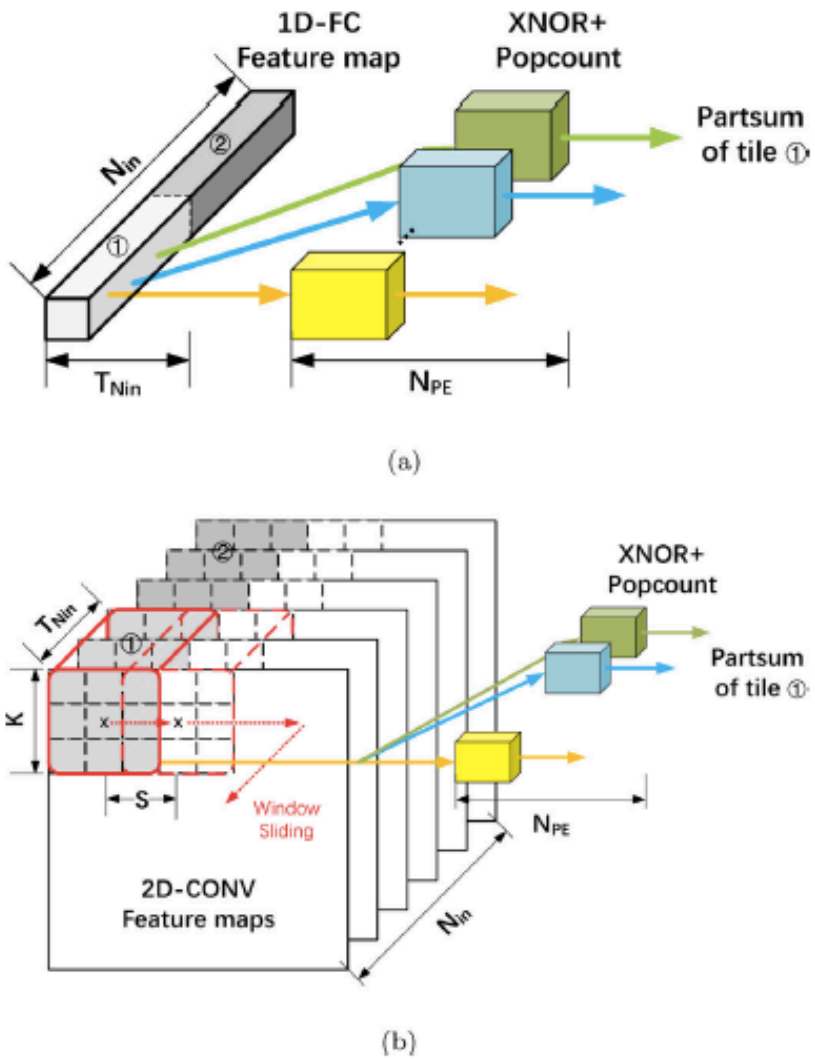


Fig. 9. Task scheduling for C/F layer: (a)FC; (b)CONV.

Memory system design

6.1. Quantization over other parameters

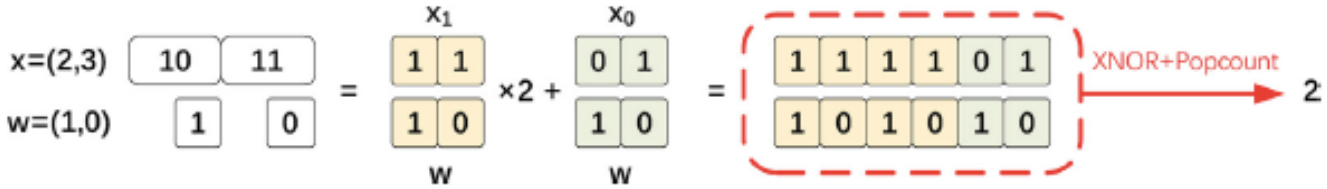


Fig. 10. Example of tiling for multiple bit case (2-bit input and 1-bit weight).

6.2. Memories for parameters

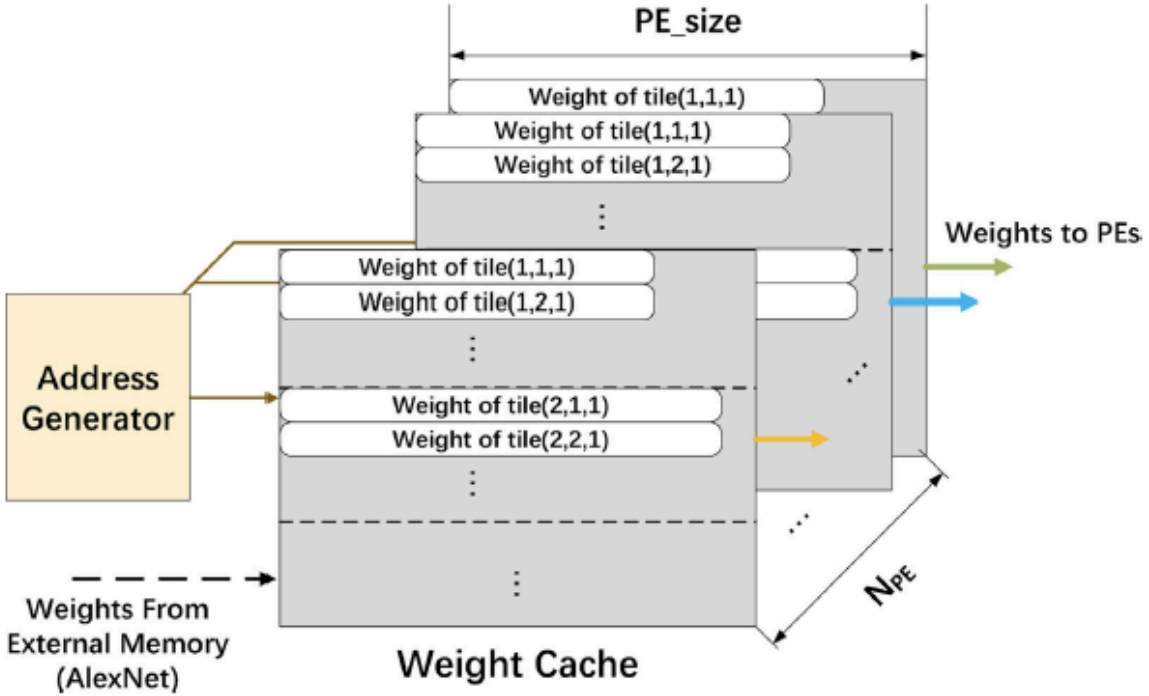


Fig. 12. Memory storage management pattern for weight cache.