Hardware Accelerator Modules for Deep Neural Networks

Project Guide: Dr. Dhanaraj K J (Assistant Professor - Dept of ECE)

Group Members (G10): Harinandan R Nair (B180718EC), Hemanth S (B180550EC), Emmanuel Thomas Devasia (B180711EC), Ellickal Allen Appukuttan (B180702EC)

***Abstract*—In recent years, AI/ML has been increasingly becoming a part of our daily lives and in the technology around us. With this increasing prevalence , the demand for fast and efficient calculations of increasingly complex models has gained immense importance. One of the biggest technical limitation faced today is the Von Neumann bottleneck[1]. We aim to explore the possibility of introducing in-memory computing modules into the architecture, to try and reduce the impact of this limitation. The possibility of using specialized circuits to improve the efficiency of matrix multiplication is also explored, which will aid in further increasing the performance of the neural network. For this we are using Static Random Access Memory (SRAM) based computation techniques. The neural network parameters like weight, inputs and operation like convolution and dot product are also binarzied to further the performance.**

***Index Terms*—In-Memory computing, Matrix multiplication, Neural networks, Static Random Access Memory, Von Neumann bottleneck**

I. Introduction

Deep learning is gaining popularity in academia and industry. The latest deep neural networks still require an enormous amount of computation and memory for training and inference. The computer systems that power today's AI algorithms are built on the Von Neumann architecture, This is very slow and expensive when it comes to moving large amounts of data between memory. In-memory computing makes a viable candidate for overcoming these, with in-memory computing, We perform certain computational tasks in place withing the memory itself, This provides the computation with very high area and energy efficiency. Analog Matrix Multiplication Circuits have been able increase computation efficiency by orders of magnitude for multiply-accumulate (MAC) operation in neural networks. In-memory computing can be realised using SRAM array. When using SRAM based IMC for MAC operations, the weights data are generally stored in the regular SRAM cell array and at the same time, The inputs are applied to wordlines or bitlines within the SRAM cell array or given to the computation circuits near the regular SRAM cell.

II. Concepts

A. Deep Neural Networks

Deep learning, also known as hierarchical learning or deep structured learning, is a type of machine learning that uses a layered algorithmic architecture to analyze data. With the advances in algorithmic and architectural design, significant improvement in computational hardware (e.g. GPU and TPU), and availability of enormous amounts of labeled data, deep learning has shown success across various domains. Deep learning models, have a cascaded structure consisting of multiple layers, with each successive layer using the output from the previous one to inform its results. This process of training the weights across all the layers is an extremely taxing computation which is one of the major limitations faced by deep neural networks.

B. In-Memory Computing

Conventional computers perform computation by repeatedly transferring data between the memory and processing units. This will result in an inherent bottleneck in performance commonly referred to as the von Neumann Bottleneck. Calculating data in the memory unit is faster than the conventional approach uses far less energy. We have started our work of exploring in-memory computing using XNOR-SRAM [2], [3], that computes ternary-XNOR-and-accumulate (XAC) operations in binary/ternary deep neural networks without row-by-row data access. We have looked into two different SRAM cells, both compatible with in-memory-computing and suitable for neural network acceleration.

C. Matrix Multiplication

The computational workhorse of Neural Networks are matrix multiplication, and accelerating the execution of Neural Networks inference by supporting faster matrix multiplication is an incredibly fertile area of research. Upon preliminary evaluation of the LeNET Architecture, we observe that almost 50% of the run-time was spent on matrix multiplication. The usage of analog circuits can speed up matrix multiplication by orders of magnitude, however this is constrained by hardware area requirements and cost. An arbitrarily large matrix cannot be computed at once as scaling hardware as per size of matrix is not a scalable approach. To resolve the same, matrices can be partitioned into sub-parts and resolved individually, to finally recreate the resultant matrix using the computed results of these subparts. This would reduce the amount of hardware required at the cost of time of computation, thus offering a trade-off between performance and hardware cost. The smaller computation can then be computed in one go by making use of

In-Memory Computing Technologies to improve performance.

III. XNOR-SRAM

To overcome the memory access limitations and arithmetic complexity, we can binarize neuron activations and weights to +1 or -1 such that the multiplication of activation and weight becomes an XNOR operation and the accumulation of the XNOR operations becomes the bit count of those XNOR outcomes. XNOR-SRAM macro consists of a 256 × 64 SRAM array, a flash analog-to-digital converter (ADC) at the periphery and a row decoder [Figure 1].

The XNOR-SRAM can operate in two modes: XNOR mode and memory mode. In the XNOR mode, it performs in-memory multiple-accumulate computation with all rows activated concurrently. In memory, mode it acts like a normal SRAM that performs a row-by-row read or writes. XNOR-SRAM is implemented using 12 transistors [Figure 2], Out of these transistors M1-M6 form the regular 6T SRAM cell. Transistors M7-M10 form the pull-up and pull-down circuits for the XNOR mode operations. Transistor M11 and M12 act as a power gate for the pull-up and pull-down circuit which can save power by deactivating a column when it is not required for the computation operations. The binary weight is stored in the 6T SRAM cell and the input signal is translated to the 4 RWL lines by the XNOR mode RWL driver. During the second part of a clock cycle, the power gates are activated for the selected columns and transistor M7-M10 perform XNOR operation between the binary weights and RWLs. In each column, parallel pull-up and pull-down paths form the voltage divider and an analog voltage settles on RBL line.

We have implemented the XNOR-SRAM bitcell using LT-Spice tool, with this bitcell we have designed a 16 × 16 array macro and verified the operation for different inputs and weights combinations. V_{RBL} is obtained as a monotonic function of XAC value [Figure 3]. For obtaining the bitcount we can digitize the analog V_{RBL} with an ADC. For this we are using a flash ADC, which is designed using the values obtained from XAC Value v/s V_{RBL} plot.

The convolution layers of the deep learning network are mapped to the XNOR-SRAM arrays of our hardware. For this mapping, the kernels of separate input channels are stored in different rows, The weights in each kernel are stored in various XNOR-SRAM arrays, and the kernels for different output channels are stored in different columns. We digitally accumulate the XAC results and generate the final sum results for each neuron. In this procedure, the stationary weights in the XNOR-SRAM arrays reuse the input activations.

IV. Alternative SRAM based In-memory computing

A. Algorithm-dependant XNOR-SRAM unit

The 8T XNOR-SRAM unit [Figure 6] consists of two word-lines (WLs) and two sets of pass-gates (PGs). In the first set, Left pass-gate(PGL) is coupled to Qb through BLB to WLB and the right pass-gate is coupled to Q through BL to WL. In the bottom set, bottom-PGL (PGLB) is coupled to Qb through WL to BL and bottom-PGR (PGRB) is coupled to Q through WL to BLB. For write operation, WL is enabled and WLB is disenabled to work like a typical 6T SRAM.[4]

XNOR function is evaluated by charging or discharging BL and BLB by charging current (iC) or discharging current (iD) corresponding to the input and weight patterns. The output polarity of binary activation function is determined by the differential comparison result of total currents from BL and BLB. The number of 1's associated with BL and BLB is compared, since '1' denotes discharge and iC is much smaller than iD. Table III shows the coding schemes of inputs and weights in the XNOR-SRAM cell and corresponding current flowing in BL or BLB.

B. Low-power In-memory-computing SRAM unit

In this 8T SRAM, for each bit cell, two extra transistors are used for vector multiplication in the generic 6T SRAM cell [Figure 7]. The extra two transistors are directly connected to RWL and can be turned on or off by controlling weight values (Q and Qb). They can be assigned a binary input (0 or 1) using RWL for BNN multiplication. The weights can be stored in Q and Qb in advance through a write operation (–1 is Q = H, Qb = L). [5]

The write and read unit of the SRAM cell are completely isolated. Weight values for BNN can be stored in Q and Qb with the write unit and BNN multiplication can be performed in the read unit. For BNN calculation, 0 or 1 is provided as input while applying a positive pulse through RWL. The input 0 is the basic low state, and input 1 can be expressed by applying a positive short pulse. The weight value can be stored as –1 (Q = L, Qb = H) or +1 (Q = H, Qb = L) through the write unit of SRAM cell.

When RWL is set to L, the read unit transistor which is connected to RWL is turned off, the output of RBL remains the same (precharged state). When RWL is H, the read unit transistor which is connected to RWL is turned on and the RBL is charged or discharged.

V. Comparison of sram based IMC

We have calculated the average power consumption of the three observed SRAM cells for all operations using LTspice tool and observed that, XNOR-SRAM [Figure 2] consumes 19.8 μW power. Alternative methods, 8T XNOR-SRAM [Figure 6] and 8T SRAM [Figure 7] consumes 55.4 μW and 31.1 μW of power respectively.

For the designed 16 × 16 array of XNOR-SRAM, We have measured the power consumption and observed that the power consumption depends on the XAC value shown in [Figure 4]. It can also observed that more power is consumed when the XAC value is near to zero.

VI. Binarizing the neural network

In classical neural networks, the input to the convolutional layers are real valued elements. This input needs to be binarized [6] so our network can perform the necessary XNOR and bit count operations on them. The weights are binarized by taking the signum of the weight values, the corresponding scaling factor alpha is the average of absolute values in the weight matrix.

A L-layer CNN architecture is represented as a triplet $\langle I, W, * \rangle$. I is a set of tensors, where each element $I = I_l : l \in [1, L]$ is the input tensor for the l^{th} layer of CNN. W is a set of tensors where each element $\mathbf{W} = W_{lk(k=1,\ldots,K^l)}$ is the k^{th} weight filter in the l^{th} layer of the CNN. K^l is the number of weight filters in the l^{th} layer of the CNN. $\mathbf{I} \in \mathbb{R}^{c \times w_{in} \times h_{in}}$, where (c, w_{in}, h_{in}) represents channels, width and height respectively. $\mathbf{W} \in \mathbb{R}^{c \times w \times h}$, where w $\leq w_{in}$, h $\leq h_{in}$.

$$B = sign(W)$$
$$\alpha = \frac{1}{n}||W||_l$$

Here, W vectors in $\mathbb{R}^n$, and $B \in \{+1, -1\}^n$, where $n = c \times w \times h$. The dot product is In binary networks are approximated as

$$X^T W \approx \beta H^T \alpha B$$
$$H = sign(X)$$
$$\beta = \frac{1}{n}||X||_l$$

Binary convolution is approximated as

$$I * W \approx (sign(I) \circledast sign(W)) \odot K\alpha$$

where $\circledast$ represents convolution operation using XNOR and bitcount operations and $\odot$ represents element wise product. This shown in Figure 5 Also,

$$K = A * k,\ where\, \forall i, j; k_{ij} = \frac{1}{w \times h}$$

$$A = \frac{\sum [I_{:,:,i}]}{c}$$

Here c is the number of channels

VII. 4x4 Microkernel using IMC

Existing schemes, to vectorize matrix multiplication in order to take full advantage of IMC Hardware, either have high memory requirements or have latency issues due to bottlenecks in the architecture. [7] proposes an algorithm that neither requires data duplication nor requires any sort of horizontal alignment. The matrix can be stored in a single vector and after computation the resultant matrix can be reconstructed from the vector using the algorithm detailed below. The same has been illustrated in Fig 8

1) Transorm input matrix A into $A_j(0)$ such as

$$A_j(0)[x][y] = A[(x + y)\%N][x]$$
$$x, y \in [0, N]$$

2) On each iteration $i > 0$, we rotate the rows of $A_j(i-1)$ by 1 byte to compute a partial result of C, which gets accumulated
3) At the end matrix C can be obtained as

$$C[x][y] = C_j[x][(x + y)\%N]$$

However this has the limitation of data-line width as an 8-bit 4×4 matrix already requires a 128-bit line, and thus this approach would not scale. We intend to use the unit described by [7] as a building block, making use of aforementioned XNOR-SRAM as the IMC unit, and design hardware units capable of multiplying larger matrices which would ultimately impact the performance of all kinds of Neural Networks. Further the microkernel itself will be directly useful in image processing, and graphical computations

VIII. Conclusion

Using in-memory computational units, computations can be made more efficient by saving the read-write delay seen in traditional systems. Different XNOR-SRAM architectures were explored, and an efficient 4×4 microkernel was studied which makes use of IMC to multiply 8-bit 4×4 matrices in a much more efficient manner. By using this as a building block for larger matrices could lead to significant energy efficiency and performance improvements to Neural Networks.

References

[1] J. von Neumann, "First draft of a report on the edvac," *IEEE Annals of the History of Computing*, vol. 15, no. 4, pp. 27–75, 1993.

[2] S. Yin, Z. Jiang, J.-S. Seo, and M. Seok, "Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 6, pp. 1733–1743, 2020.

[3] S. Yin, Z. Jiang, M. Kim, T. Gupta, M. Seok, and J.-S. Seo, "Vesti: Energy-efficient in-memory computing accelerator for deep neural networks," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 48–61, 2020.

[4] H. Jiang, R. Liu, and S. Yu, "8t xnor-sram based parallel compute-in-memory for deep neural network accelerator," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2020, pp. 257–260.

[5] Y. Kim, S. Li, N. Yadav, and K. K. Choi, "A novel ultra-low power 8t sram-based compute-in-memory design for binary neural networks," *Electronics*, vol. 10, no. 17, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/17/2181

[6] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," pp. 525–542, 2016.

[7] K. MAMBU, H.-P. Charles, and M. Kooli, "Efficient 8-bit matrix multiplication on Computational SRAM architecture," in *DATE 2020 ; Computing-in-Memory (CiM) Workshop*, Grenoble, France, Mar. 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02934838

APPENDIX

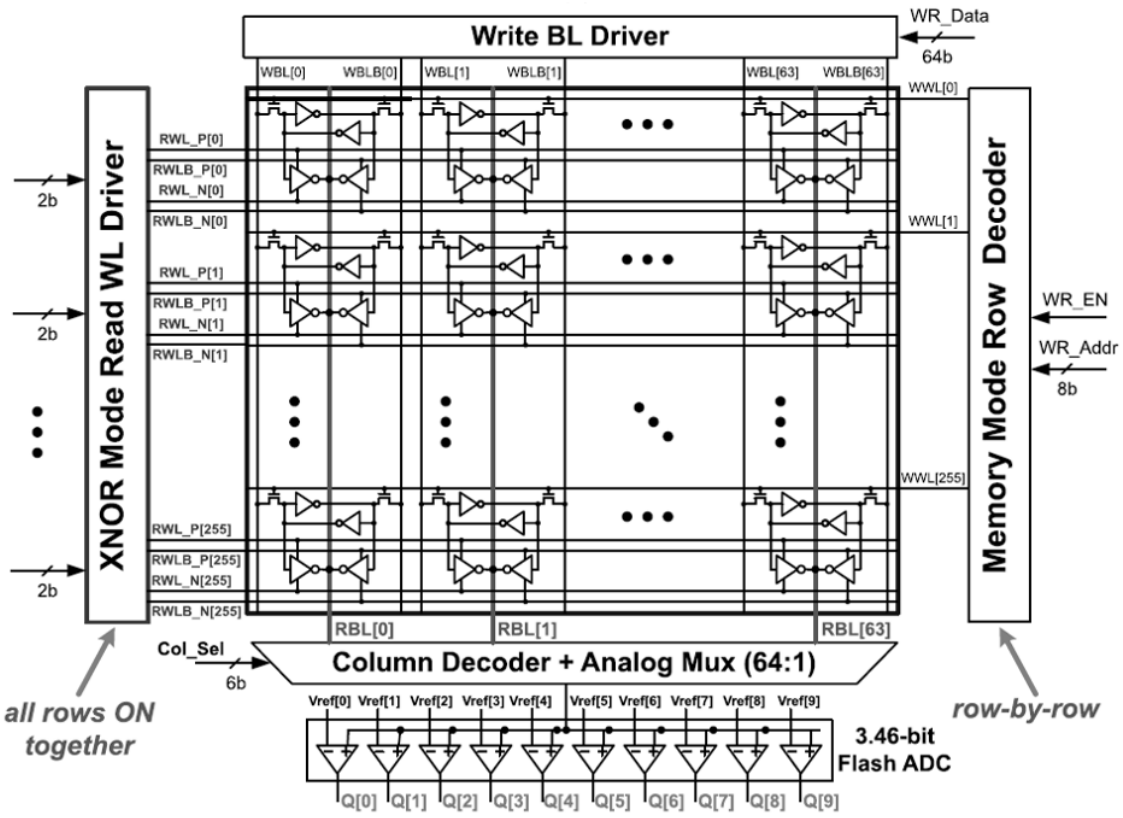

Figure 1. SRAM array architecture [2]

Table I
RWL DRIVER LOGIC

Input / RWL	+1	-1	0 odd row	0 even row
RWL_P	VDD	0	0	VDD
RWLB_P	0	VDD	0	VDD
RWL_N	VDD	0	VDD	0
RWLB_N	0	VDD	VDD	0

Table II
XNOR VALUE MAPPING

Input / XNOR	Strong pull-up	Weak pull-up	Strong pull-down	Weak pull-down
+1	1	1	0	0
-1	0	0	1	1
0 (odd row)	1	0	1	0
0 (even row)	0	1	0	1

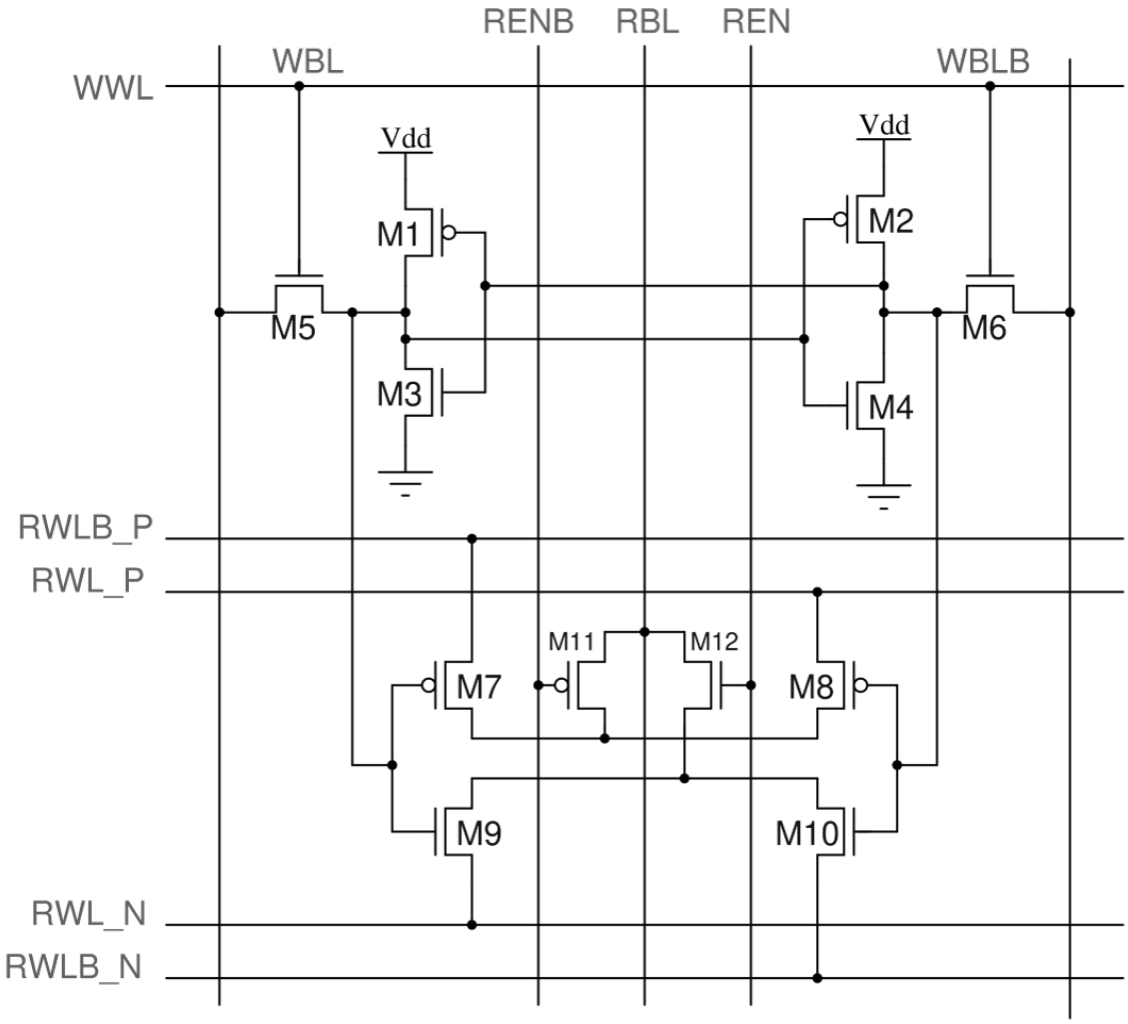

Figure 2. XNOR-SRAM bit cell

Table III
CODING SCHEME OF INPUTS AND WEIGHTS IN 8T XNOR SRAM

Weight			**Input**			**XNOR (Product)**		
Value	WL	WLB	Value	Q	Qb	Value	BL	BLB
+1	1	0	+1	1	0	+1	iC	iD
+1	1	0	-1	0	1	-1	iD	iC
-1	0	1	+1	1	0	-1	iD	iC
-1	0	1	-1	0	1	+1	iC	iD

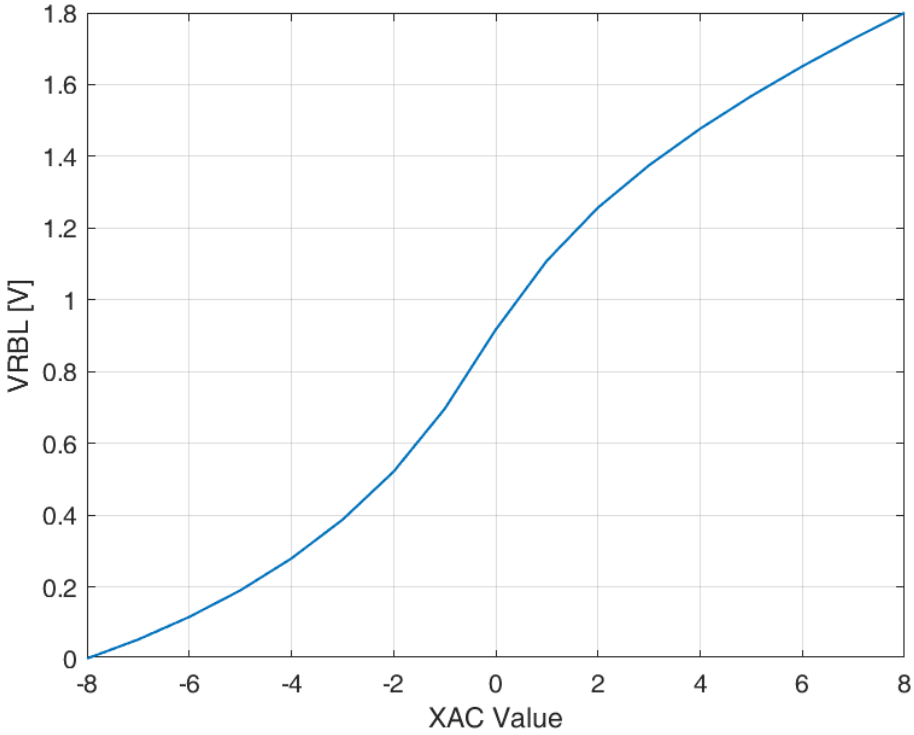

Figure 3. V_{RBL} over logical result of 16-input XAC operations

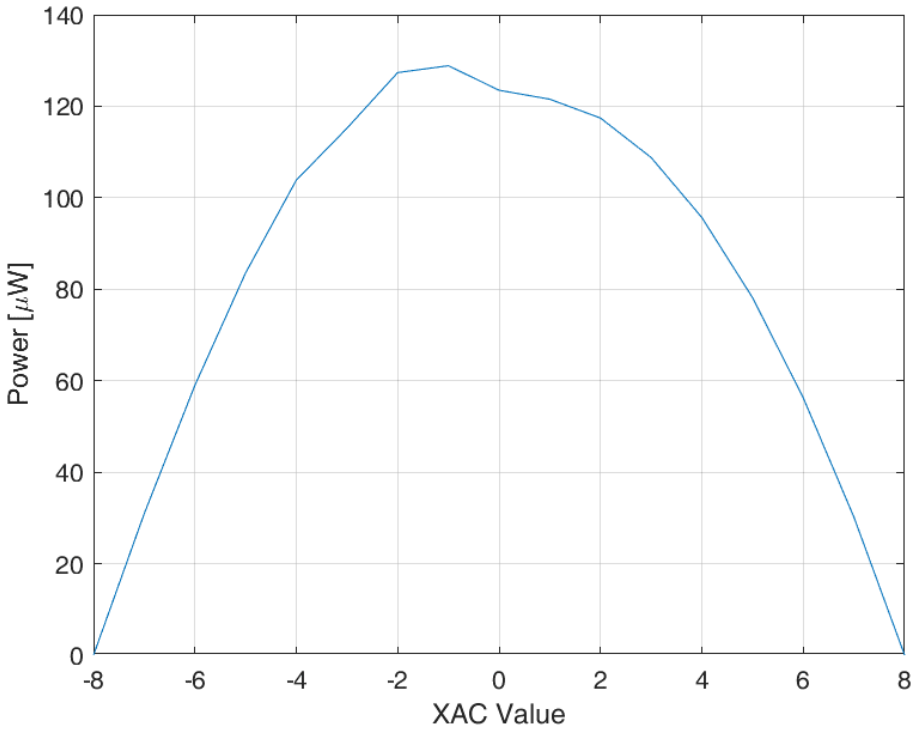

Figure 4. Power consumption over logical result of 16-input XAC operations

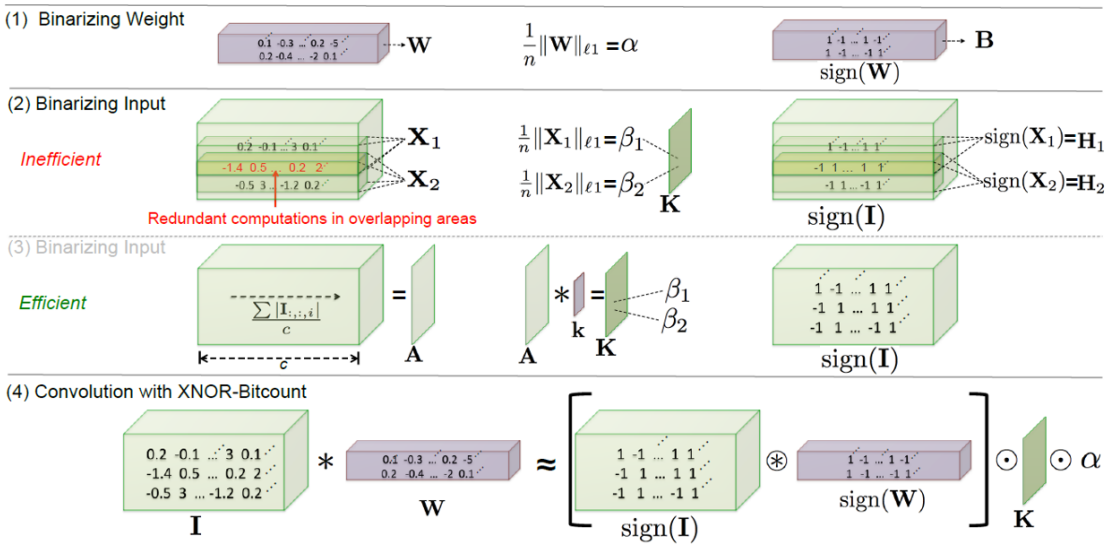

Figure 5. Approximating convolution using binary operations [6]

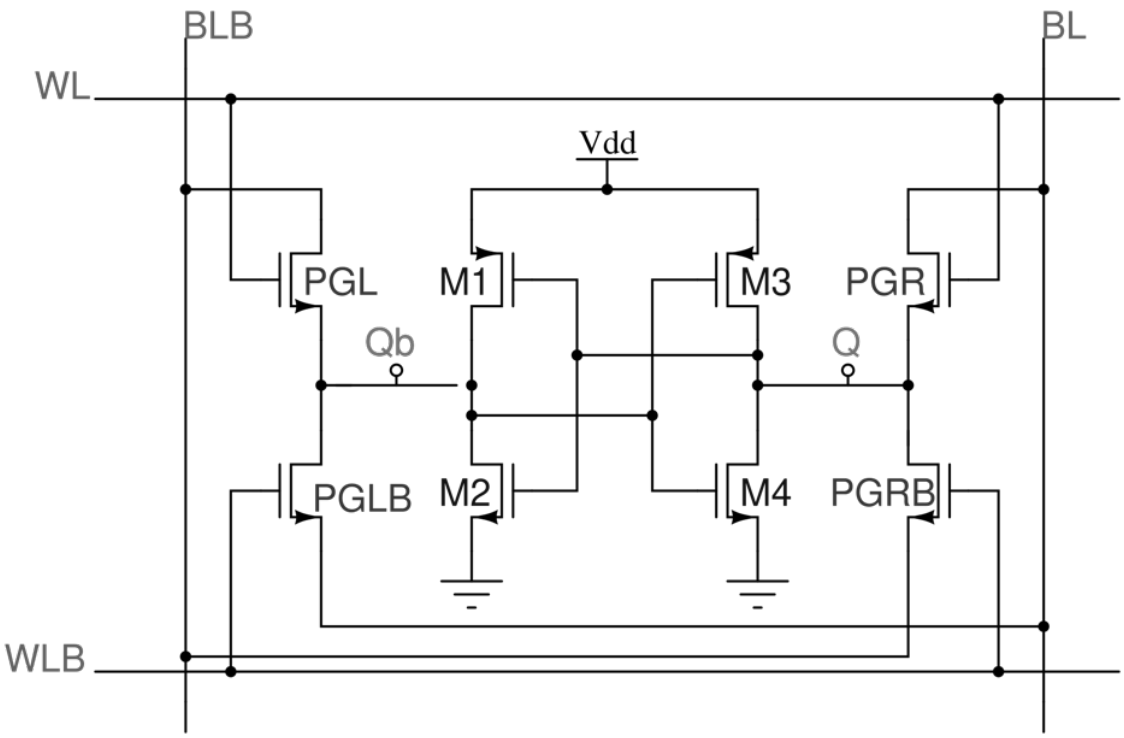

Figure 6. Algorithm dependent 8T XNOR-SRAM bitcell

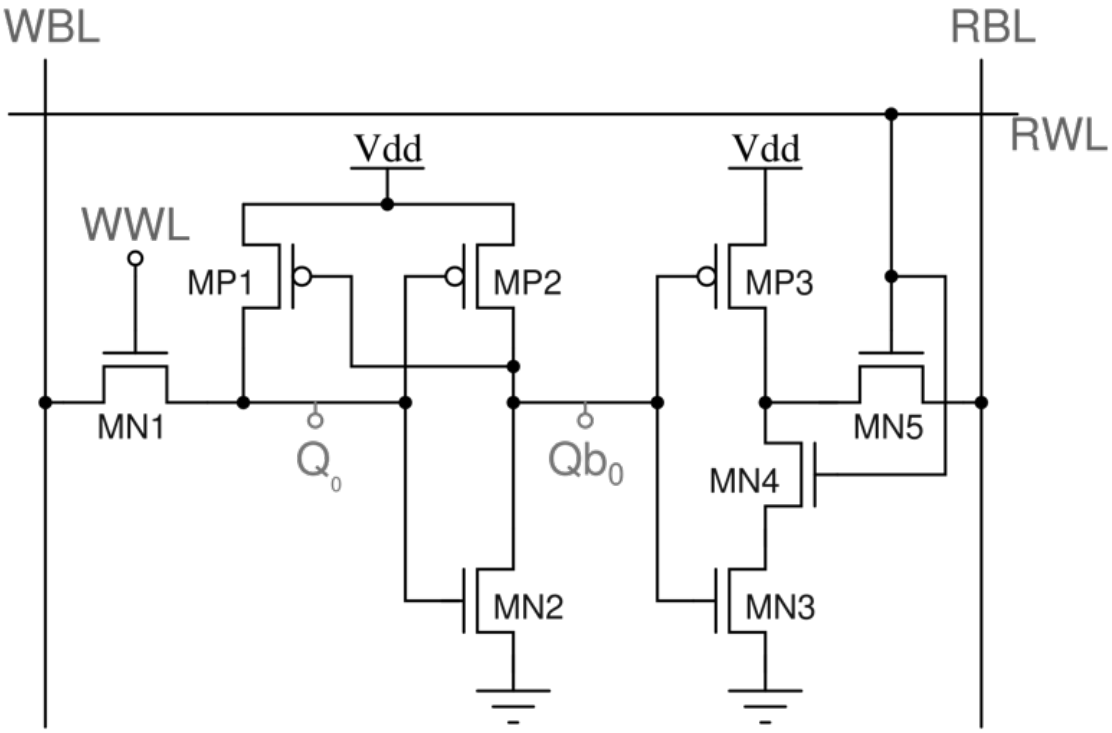

Figure 7. Low power 8T XNOR-SRAM bitcell

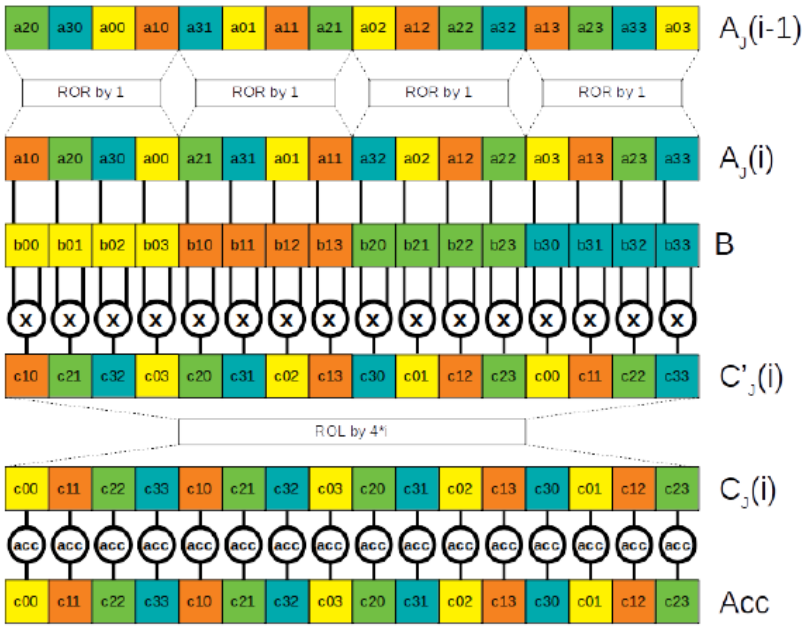

Figure 8. Proposed vectorization scheme for 4×4 microkernel in [7]