Journal of Engineering and Technology for Industrial Applications

ITEGAM-JETIA

Manaus, v.12 n.57, p. 574-580. January/February, 2026.

DOI: https://doi.org/10.5935/jetia. v12i57.2829

ISSN ONLINE: 2447-0228

RESEARCH ARTICLE OPEN ACCESS

AREA AND POWER EFFICIENT SPARSE AWARE ARCHITECTURE FOR POINTWISE CONVOLUTION UNIT USING VARIABLE DENSITY BOUND BLOCK

Golajapu Jyothsna[1], Narayanam Balaji[2]

[1, 2]Department of Electronics and Communications Engineering, Jawaharlal Nehru Technological University, Kakinada, Andhra Pradesh, India

[1]https://orcid.org/0009-0007-3515-907X, [2]https://orcid.org/0009-0003-5885-6688

Email: golajapujoe@gmail.com, narayanambalaji@jntucek.ac.in

ARTICLE INFO

Article History

Received: October 17, 2025

Revised: December 20, 2025

Accepted: January 15, 2026

Published: February 28, 2026

Keywords:

Pointwise Convolution Unit, Variable Density Bound Block, Activation stationary, Sparsity, MobileNet-v1.

ABSTRACT

MobileNet introduced depthwise separable convolution, significantly reducing computations in comparison to standard convolution, making them fit for edge computing. Parameter-efficient models like MobileNet-v1 can be pruned to further reduce the number of parameters. In most of the pruning methods, pointwise convolutional parameters are pruned as they are large in number and depthwise convolutional parameters are kept dense. The efficacy of MobileNet pruning for edge deployment depends critically on a sparsity-aware execution environment. This paper focuses on implementing sparse aware pointwise convolution unit that leverage weight sparsity using Variable Density Bound Block (VDBB) technique. The activation stationary method is also incorporated, as it saves the power to fetch activations continuously. The proposed architecture is implemented on the FPGA platform using Vivado 2024.1. Implementing the sparse-aware architecture on the xc7z045ffv900-1 FPGA resulted in a 358mW reduction in power, a 50% reduction in DSPs, and a 24.06% reduction in LUTs compared to the sparse-unaware architecture. An engine from the proposed architecture was synthesized in ASIC using the TSMC 90 nm standard cell library, resulting in a 30.87% reduction in power and a 32.89% reduction in area when compared to a sparse unaware engine.

I. INTRODUCTION

Convolution Neural Networks (CNNs) are crucial in modern signal processing, such as classification, denoising, and pattern recognition. CNN can learn relevant features directly from the data, reducing the need for manual feature engineering. The popular architectures among CNN are Dense CNN [1], AlexNet [2], VGG [3], and GoogLeNet [4]. These architectures have millions of parameters and billions of computations. Storing them at a single-precision floating-point demands Megabits of storage space. Performing millions of computations demands enormous power. More storage space and high-power consumption are not desirable for edge devices. The dense model parameters can be reduced by using methods such as quantization, pruning, and so forth. Quantization contributes to reduction in memory footprint as it reduces the precision of weights and activations.

Pruning decreases the quantity of parameters; pruning can be structured or unstructured contributing to block sparsity and tensor sparsity respectively. Also, there are lightweight models like MobileNet-v1 [5], MobileNet-v2 [6], ShuffleNet [7], and Squeeze Net [8] with a smaller number of parameters and computations but with a considerable amount of accuracy drop. MobileNet, ShuffleNet, and EfficientNet-Lite use Depthwise separable convolution, as the number of computations got reduced by $((1/N) + (1/D_k^2))$ compared to standard convolution, where N is number of kernels and Dk is the dimension of kernel. The question arises: can lightweight models like MobileNet be pruned? as it is less redundant. But certain pruning techniques for MobileNet-v1 yielded remarkable results. Automated Gradual Magnitude Pruning (GMP) algorithm [9] prunes the lowest magnitude weights, demonstrated that pruned dense MobileNet with 50% sparsity is better in accuracy than 0.75 MobileNet with a lesser number of parameters.

Soft Threshold Weight Reparameterization (STR) [10] achieved good accuracy for CNN in case of unstructured sparsity. STR achieved an accuracy of 68.35% using 101M FLOPS (FLoating point Operations Per Second) at 75.28% sparsity, where GMP achieved an accuracy of 67.7% using 163M FLOPS at 74.11% sparsity for MobileNet-v1. Deep Q-Learning for Pruning Deep Neural Networks (QLP) [11] method based on deep reinforcement learning achieved a good amount of accuracy 71.06% at 50% sparsity with an accuracy drop of 0.89% compared to the baseline MobileNet-v1. These results demonstrate that MobileNet-v1 can be pruned. It is interesting to note that the first layer and the depthwise convolutional layers are kept dense in GMP, QLP methods. STR made depthwise convolutional layers less sparse.

So, only Pointwise convolutional layers are pruned in these methods as it has 74.59% of parameters [5]. All these methods contribute to unstructured sparsity. Unstructured sparsity is advantageous while deploying the model on edge devices only if the designed hardware is sparse aware. So, this work focuses on implementing sparse-aware architecture for pointwise convolutions using the Variable Density Bound Block technique, which works for all sparsity levels without losing non-zero parameters. The activations are kept stationary until the dot product with all the kernels are computed, which reduces the power consumption to bring activations from the buffer to the processing units continuously.

II. POINTWISE CONVOLUTION

Pointwise convolution is also known as 1x1 convolutions. Usually, it is paired with depthwise convolution in MobileNet-v1, as depthwise convolution lacks channel mixing to learn complex hierarchical features [5]. Pointwise convolution is utilized in ResNet bottlenecks to expand and compress channels, in ShuffleNet for channel mixing, and in attention blocks for channel recalibration. So, pointwise convolution is used for channel mixing and for dimension adjustments. Pointwise convolution is quite like standard convolution but the only difference is its kernel spatial dimensions.

In pointwise convolution, the kernel dimension is 1x1xC where C is the number of channels. Pointwise convolution is nothing but the dot product of the kernel and the part of the activation layer as shown in Figure.1. Assuming an activation layer of dimension HxWxC and N number of kernels of dimension 1x1xC, then the pointwise convolution output dimension is HxWxN. N decreases the channel dimension if it is less than C and increases if it is greater than C. Mathematically, let activation layer be A $\in \mathbb{R}^{HxWxC}$ and kernel be W1 $\in \mathbb{R}^{1x1xC}$ then pointwise convolution output at a particular location (h, w) is

$$P(h,w) = \sum_{c'=1}^{C} A(h,w,c') \,.\, W_1(1,1,c') \quad (1)$$

Where

H= Number of rows of activation layer

W= Number of columns of activation layer

C= Number of channels

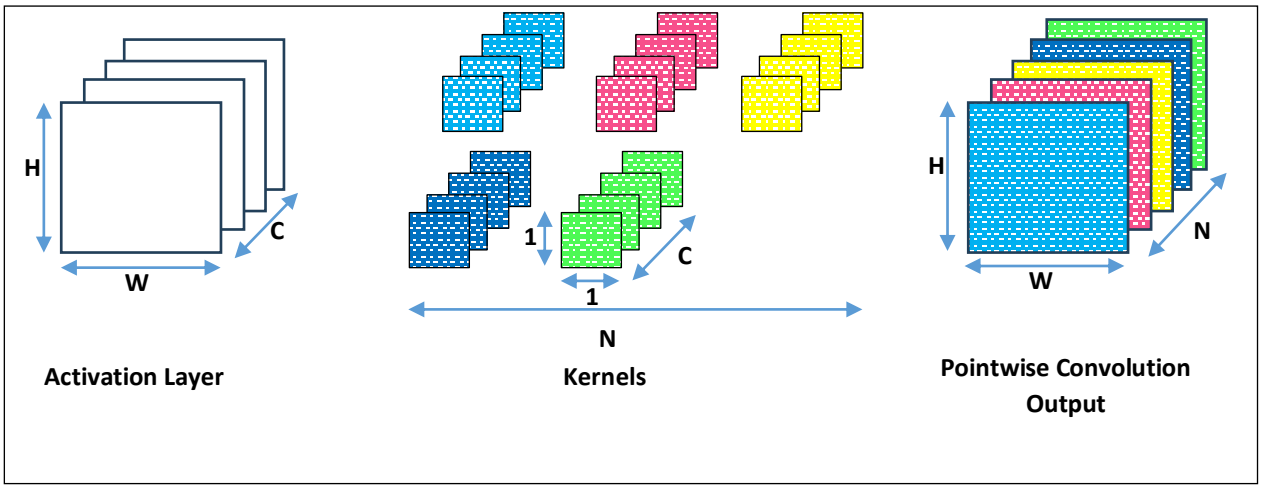


Figure 1: Pointwise Convolution.
Source: Authors, (2026).

III. PROPOSED ARCHITECTURE

Figure.2. shows the proposed architecture for sparse aware pointwise convolution unit. The proposed architecture supports parallelism by segregating feature maps into tiles [12]. The proposed architecture was designed for MobileNet-v1, but it can also be used for different models where pointwise convolutions are needed, just by redesigning buffers. MobileNet-v1 feature maps have channel sizes that are multiples of 32, with the exception of input image channels, as shown in Figure.3. So, each data word in the activation buffer contains 32 activations, which are taken in channel-wise direction to enable 32 channel unrolling. According to the position of non-zero weights, a vector of 1's and 0's was stored as a mask in the mask buffer as shown in Figure.4. As the weights are sparse, the weight buffer conserves memory by storing only non-zero weights. Entire activations along channel-wise at a particular location are fetched into the activation register from the activation buffer. For instance, if there are 64 channels, then two data words are fetched into the activation register. These activations in the register are kept stationary until all kernels are convolved with these activations.

The activation selector uses Variable Density Bound Block (VDBB) to choose activations corresponding to non-zero weights from the activation register using mask value. Because of VDBB, each engine has sixteen multipliers for 32-channel unrolling. The concept of VDBB is discussed in detail in section 4. Each engine takes one clock cycle if sparsity is greater than 50% and two clock cycles if sparsity is less than 50% for 32 channels. Partial results are accumulated until all channels at a specific spatial location within the feature map are processed.

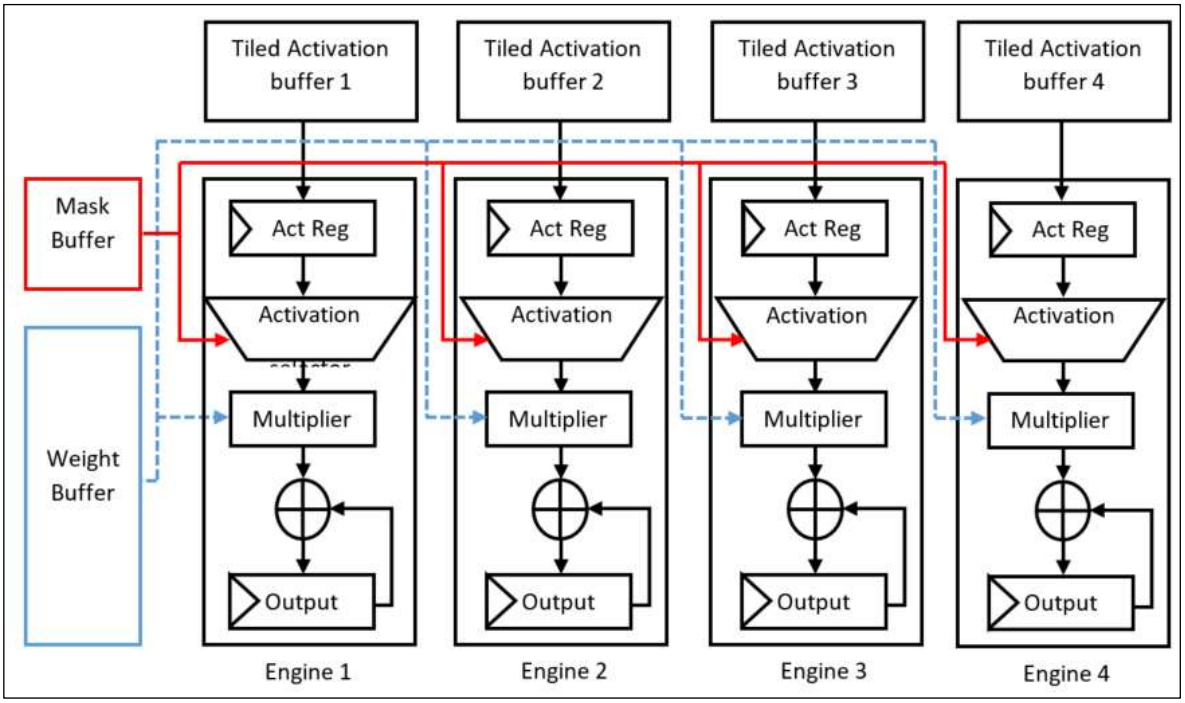


Figure 2: Proposed Architecture of Sparse-Aware Pointwise Convolution Unit.
Source: Authors, (2026).

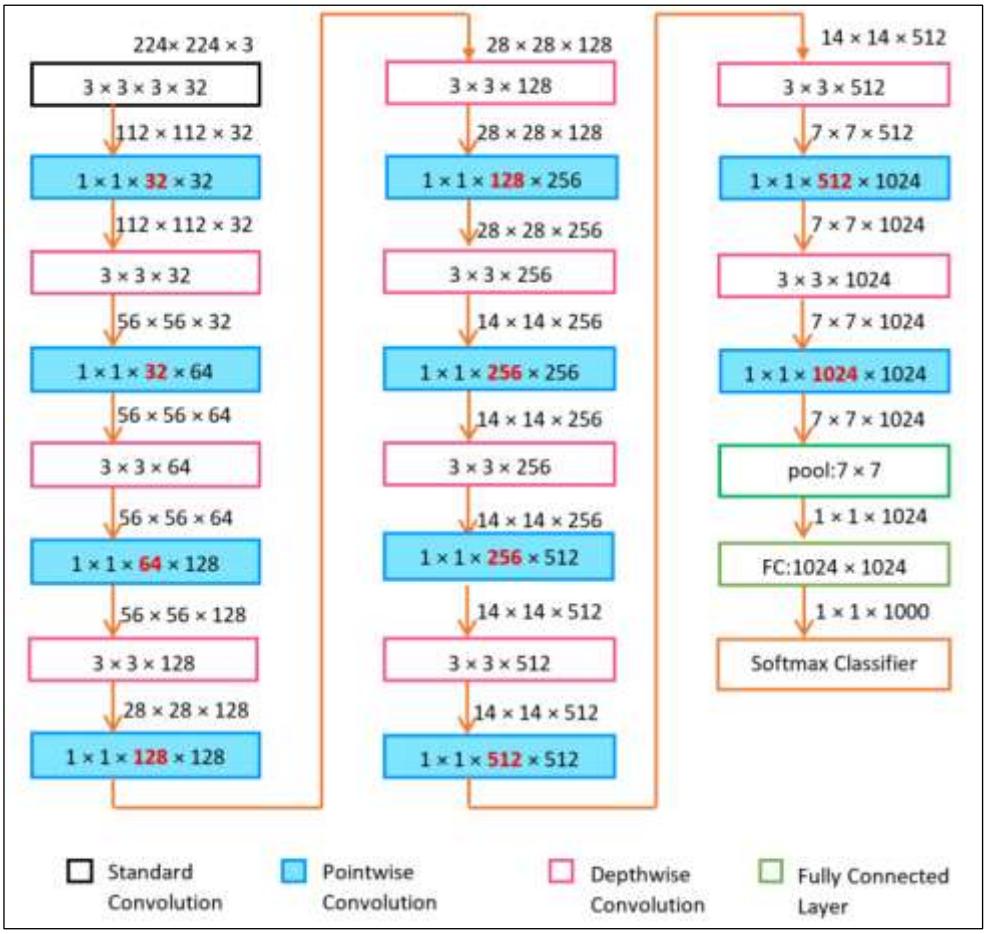


Figure 3: MobileNet-v1 Layer Information.
Source: Authors, (2026).

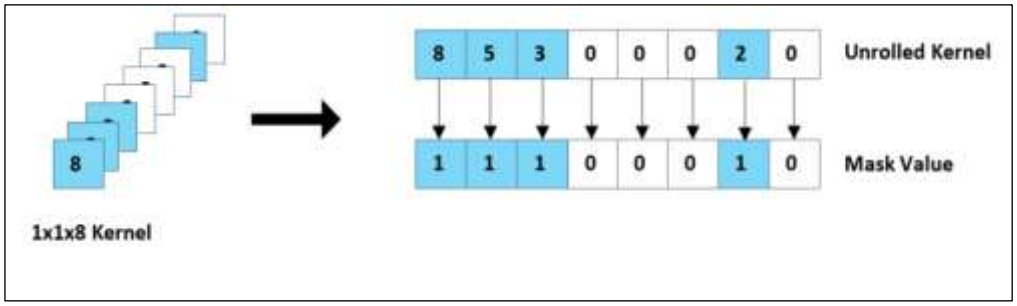


Figure 4: Mask generation.
Source: Authors, (2026).

IV.ACTIVATION SELECTOR

Sparsity is nothing but the existence of zeros in the weight tensor or in activation tensor. Pruning causes sparsity in weights, while ReLU causes sparsity in activations. Based on the distribution of non-zero elements in sparse tensors, sparsity can be Random, block sparse, density bound block as shown in Figure.5. If non-zero elements are distributed randomly, as illustrated in Figure. 5(a), this configuration is referred to as random sparse. Usually sparsity in Convolutional Neural Networks (CNNs) is random. This randomness creates an overhead in hardware while inferencing the model, as indexes of non-zero elements need to be stored specifically and the load is also unpredictable. In the context of block sparsity, where non-zero elements are organized in blocks, as illustrated in Figure. 5(b), fewer indexes are required compared to a random arrangement. However, imposing block sparsity on CNNs can lead to a decline in accuracy. In Density Bound Block (DBB) [13], the number of non-zero elements in a block are predefined, instead of forcing all the elements in a block to zero or fully dense as shown in Figure.5(c), which saves from accuracy drop.

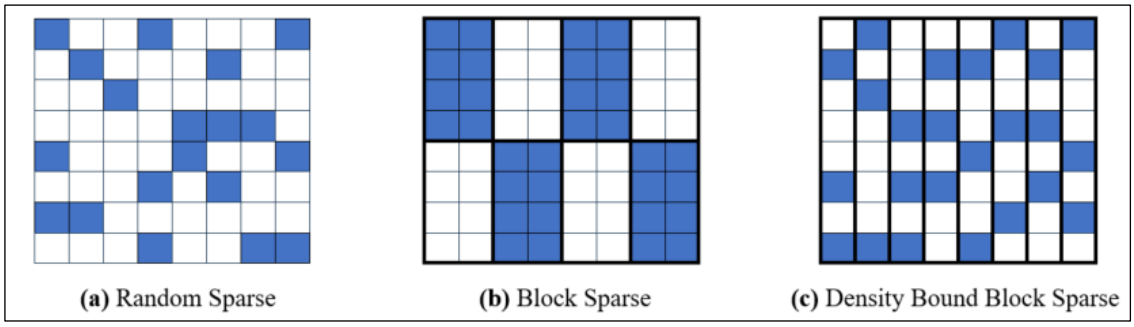


Figure 5: Types of Sparsity, filled boxes represents non-zero elements and empty boxes represent zeros.
Source: Authors, (2026).

DBB enables the use of Multiply and Accumulate (MAC) units in proportion to the level of sparsity. For instance, with 50% sparsity, only 50% of the usual MACs are needed, which significantly lowers hardware costs. But the problem is, sparsity is fixed at design time leading to loss of non-zero elements when sparsity is less than predefined sparsity. To overcome this issue, Variable Density Bound Block Technique (VDBB) [14] had been designed, which works at all sparsity levels. VDBB uses the same number of MACs as DBB. As the weights are randomly sparse, and the engine in the architecture processes 32 weights at once, this results in varying sparsity levels. To handle these varying sparsity levels, the proposed architecture uses VDBB as activation selector. Thus, no non-zero element is discarded during computation. The design for VDBB is taken from [15]. As shown in Figure. 6, sparsity less than 50% takes two clock cycles to compute, and sparsity greater than 50% takes one clock cycle to compute.

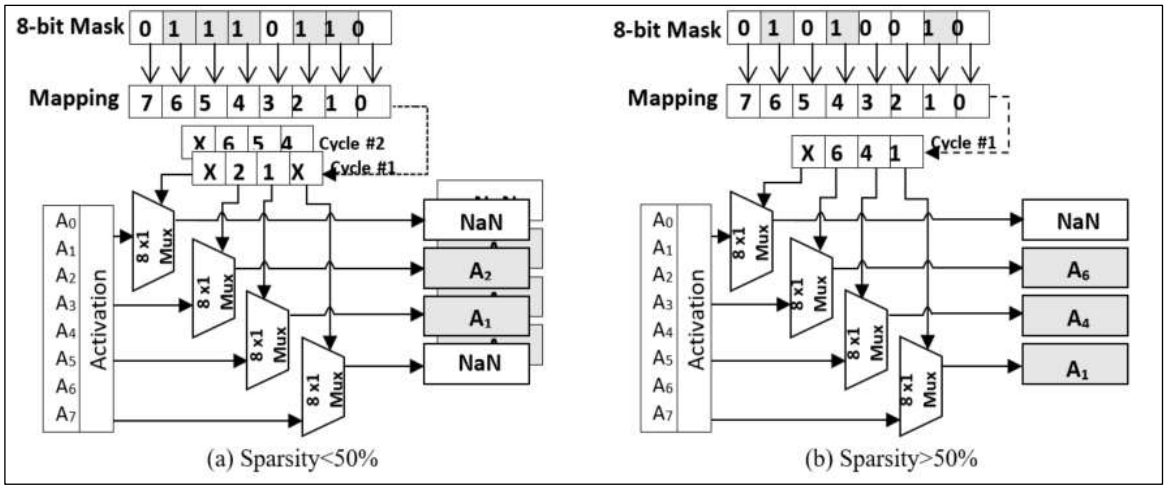
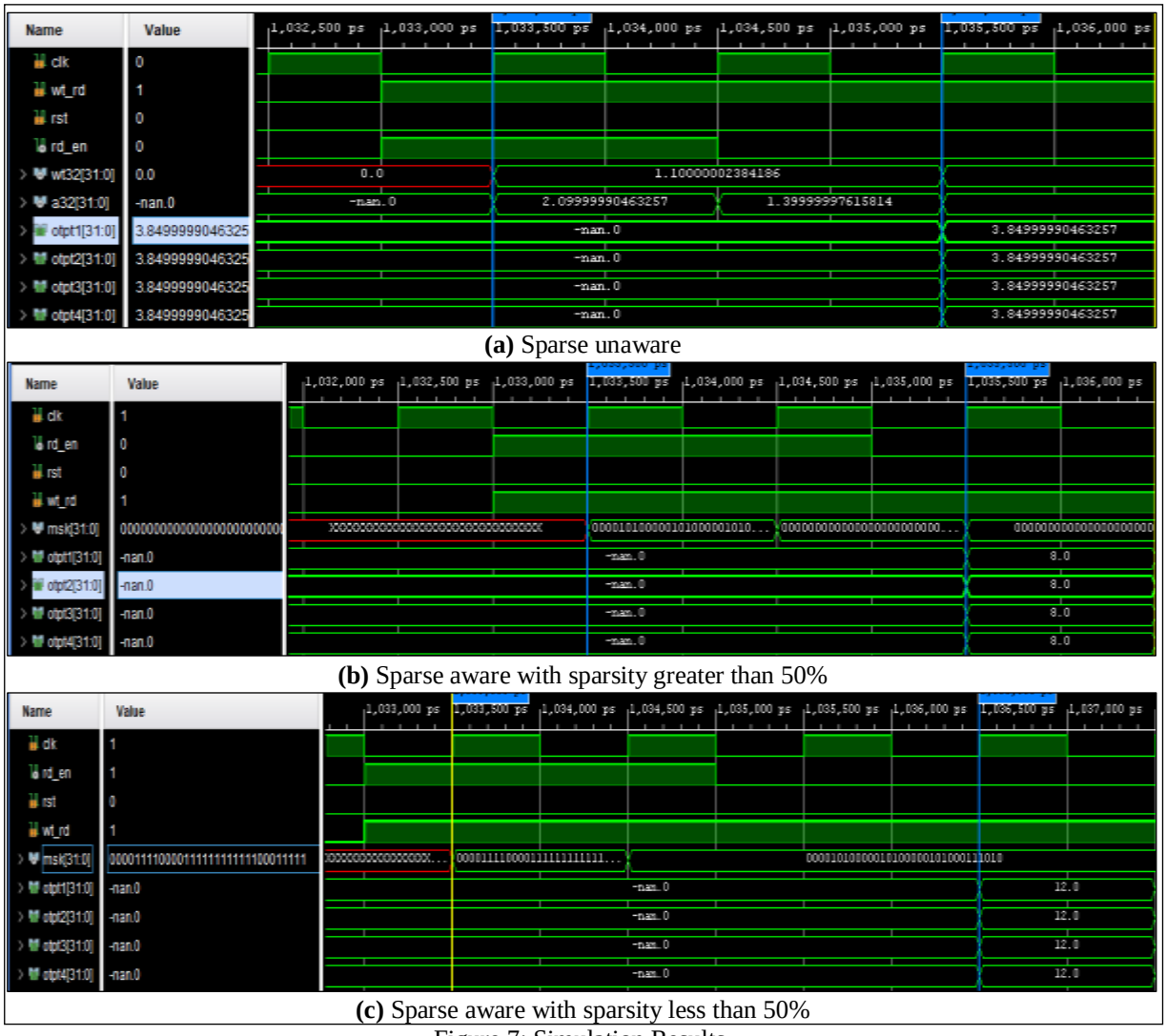


Figure 6: Activation Selector based on VDBB.
Source: Authors, (2026).

The mask value is used to determine the index values to choose activations corresponding to non-zero weights. Indexes are given to multiplexers, which select the corresponding activations. The indexes are organized in such a way that if sparsity is greater than 50%, then the indexes are accumulated in one clock cycle as shown in Figure. 6(a); if sparsity is less than 50%, then the indexes are organized in two clock cycles as shown in Figure.6(b).

V. RESULTS AND DISCUSSIONS

The proposed architecture had been implemented using Xilinx Vivado 2024.1 targeting xc7z045ffv900-1. Verilog HDL is used to design the architecture. Figure.7 shows the simulation traces for sparse-unaware and sparse-aware architectures. For the depth of 64, it is observed from Figure.7(a) and Figure.7(b) that in both sparse unaware and sparse aware (both data words exhibit >50% sparsity) simulation, two clock cycles are needed to get the final output, as two data words need to be processed. Figure. 7(c) presents a mixed-sparsity case in which one data word has sparsity below 50% (requiring two cycles) while the other exceeds 50% (requiring one cycle), resulting in a total of three clock cycles. These results indicate that the sparsity-aware implementation incurs an extra clock cycle whenever a processed data word's sparsity falls below the 50% threshold. Activation-stationary behavior is evident from the rd_en signal, as shown in Figure 7. The signal asserts only when activation words need to be fetched from the activation buffer according to the configured depth, thereby eliminating continuous buffer reads and reducing power consumption.



(a) Sparse unaware

(b) Sparse aware with sparsity greater than 50%

(c) Sparse aware with sparsity less than 50%

Figure 7: Simulation Results.
Source: Authors, (2026).

From Table.1, it is observed that the proposed architecture resulted in 50% reduction in DSPs, 24.06% reduction in LUTs, 8.96% reduction in BRAMs and 10.96% rise in Flip-flops compared to sparse unaware architecture. The results clearly demonstrate that the proposed architecture achieved good resource utilization, enabling efficient deployment on edge devices. It is also observed that the power was reduced by 358mW compared to sparse unaware architecture. ASIC synthesis of the sparse-aware and sparse-unaware engines was carried out in Cadence Genus and Innovus, targeting a 90 nm standard cell library. The core area of sparse aware architecture is 0.55980mm^2, whereas sparse unaware architecture occupies 0.83421mm^2. Although the number of MACs was reduced by 50% compared to the sparse-unaware engine, the inclusion of activation selector units resulted in an overall area reduction of 32.89%. The proposed engine consumes 0.91mW, whereas the sparse-unaware engine consumes 1.32mW as shown in Table 2. Figure.8 shows the layout of the sparse aware engine implemented using Innovus.

Table 1: Performance comparison of sparse unaware and sparse aware architectures implemented using FPGA xc7z100ffg900-1.

	Sparse Unaware architecture	Sparse aware architecture
# LUT	64180	48733
# LUTRAM	2736	2736
# FF	4467	5017
#BRAM	72.50	66
# DSP	256	128
Precision	FP32	FP32
Operating Frequency	100MHz	100MHz
Power(W)	2.231	1.873

Source: Authors, (2026).

Table 2: Performance comparison of ASIC implemented sparse unaware and sparse aware engine.

	Sparse Unaware architecture	Sparse aware architecture
Area(mm^2)	0.83421	0.55980
Power(mW)	1.32787	0.91793

Source: Authors, (2026).

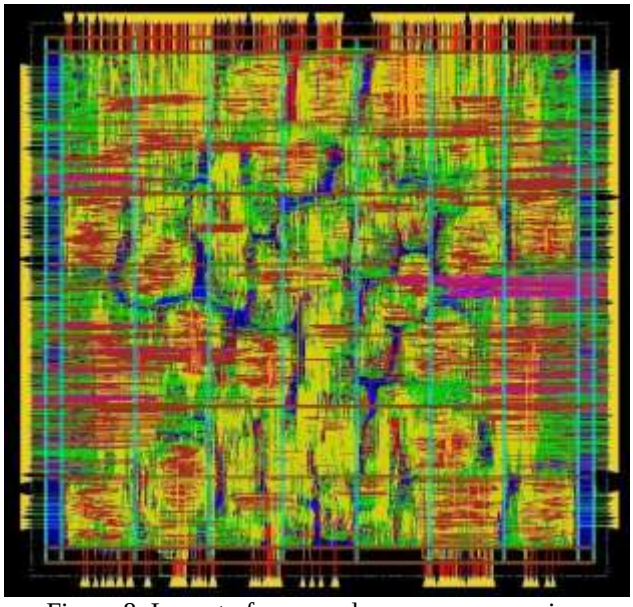


Figure 8: Layout of proposed sparse aware engine.
Source: Authors, (2026).

VI. CONCLUSION

Implementing Neural networks on edge devices demands efficient models and efficient hardware, as neural networks are computationally intensive. There are lightweight models like MobileNet, EfficientNet, SqueezeNet, and so on, with greatly reduced parameters. Even these lightweight models are further pruned using various techniques without much accuracy loss to fit in edge devices. These techniques prune only point wise layers, as they are huge in number. Pruning causes sparsity, this is advantageous only when the hardware is sparse aware. So, the sparse-aware pointwise convolution unit architecture was designed and implemented on FPGA, and an engine of this architecture was implemented in ASIC. In FPGA implementation, sparse aware architecture resulted in 50% reduction of DSPs and 358mW reduction in power compared to sparse unaware. Implementation of the engine in ASIC using 90nm resulted in 32.89% reduction in area and 30.87% reduction in power. These results emphasize the crucial role of sparse-aware design while deploying neural networks on edge devices.

VII.AUTHOR'S CONTRIBUTION

Conceptualization: Golajapu Jyothsna, Narayanam Balaji.
Methodology: Golajapu Jyothsna, Narayanam Balaji.
Investigation: Golajapu Jyothsna, Narayanam Balaji.
Discussion of results: Golajapu Jyothsna, Narayanam Balaji.
Writing – Original Draft: Golajapu Jyothsna, Narayanam Balaji.
Writing – Review and Editing: Golajapu Jyothsna, Narayanam Balaji.
Resources: Golajapu Jyothsna, Narayanam Balaji.
Supervision: Golajapu Jyothsna, Narayanam Balaji.
Approval of the final text: Golajapu Jyothsna, Narayanam Balaji

VIII. REFERENCES

[1] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely Connected Convolutional Networks,” Jan. 2018, [Online]. Available: http://arxiv.org/abs/1608.06993

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun ACM, vol. 60, no. 6, pp. 84–90, Jun. 2017, doi: 10.1145/3065386.

[3] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” Apr. 2015, [Online]. Available: http://arxiv.org/abs/1409.1556

[4] C. Szegedy et al., “Going Deeper with Convolutions,” Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.4842

[5] A. G. Howard et al., “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” Apr. 2017, [Online]. Available: http://arxiv.org/abs/1704.04861

[6] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE Computer Society, Dec. 2018, pp. 4510–4520. doi: 10.1109/CVPR.2018.00474.

[7] X. Zhang, X. Zhou, M. Lin, and J. Sun, “ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices,” Dec. 2017, [Online]. Available: http://arxiv.org/abs/1707.01083

[8] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, “SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size,” Nov. 2016, [Online]. Available: http://arxiv.org/abs/1602.07360

[9] M. Zhu and S. Gupta, “To prune, or not to prune: exploring the efficacy of pruning for model compression,” Nov. 2017, [Online]. Available: http://arxiv.org/abs/1710.01878

[10] A. Kusupati et al., “Soft Threshold Weight Reparameterization for Learnable Sparsity,” Jun. 2020, [Online]. Available: http://arxiv.org/abs/2002.03231

[11] E. Camci, M. Gupta, M. Wu, and J. Lin, “QLP: Deep Q-Learning for Pruning Deep Neural Networks,” IEEE Transactions on Circuits and Systems for Video Technology, vol. 32, no. 10, pp. 6488–6501, Oct. 2022, doi: 10.1109/TCSVT.2022.3167951.

[12] H. Hong, D. Choi, N. Kim, and H. Kim, “Mobile-X: Dedicated FPGA Implementation of the MobileNet Accelerator Optimizing Depthwise Separable Convolution,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 71, no. 11, pp. 4668–4672, 2024, doi: 10.1109/TCSII.2024.3440884.

[13] Z.-G. Liu, P. N. Whatmough, and M. Mattina, “Systolic Tensor Array: An Efficient Structured-Sparse GEMM Accelerator for Mobile CNN Inference,” May 2020, [Online]. Available: http://arxiv.org/abs/2005.08098

[14] Z.-G. Liu, P. N. Whatmough, and M. Mattina, “Sparse Systolic Tensor Array for Efficient CNN Hardware Acceleration,” Sep. 2020, [Online]. Available: http://arxiv.org/abs/2009.02381

[15] Q. Cheng et al., “A Low-Power Sparse Convolutional Neural Network Accelerator With Pre-Encoding Radix-4 Booth Multiplier,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 70, no. 6, pp. 2246–2250, Jun. 2023, doi: 10.1109/TCSII.2022.3231361.