STM JOURNALS

Journal of
Microcontroller Engineering
and Applications

ISSN: 2455-197X
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/JoMEA

https://journals.stmjournals.com/jomea

Review JoMEA

MAC Unit Implementation on FPGA

Dnyaneshwari L. Shinde[1], Alisha B. Mulani[1], Diksha A. Waghmare[1], Swapnil A. Takale[2,*]

Abstract

Multiply–accumulate (MAC) computations account for a large part of machine learning accelerator operations. The pipelined structure is usually adopted to improve performance by reducing the length of critical paths. An increase in the number of flip-flops due to pipelining, however, generally results in significant area and power increase. Using this method, we create and build a feedforward-cutset-free (FCF) MAC architecture that maximizes data propagation and removes superfluous pipeline registers. By reorganizing the computation flow to eliminate unnecessary dependencies, the suggested MAC unit permits continuous forward data movement without compromising correctness. In order to achieve high operating speed with low-latency, pipeline and parallel processing techniques are combined to guarantee that intermediate results propagate smoothly across stages. The suggested architecture, in contrast to conventional designs, minimizes register insertion and prevents over-pipelining, resulting in a more area- and power-efficient implementation. A large number of flip-flops are often required to meet the FCF rule. Based on the observation that this rule can be relaxed in machine learning applications, we propose a pipelining method that eliminates some of the flip-flops selectively. The simulation shows that the proposed MAC unit achieved a 50% area reduction compared with the conventional pipelined MAC. In this work, we propose and implement a FCF MAC unit that eliminates redundant dependencies and optimizes data flow for high-speed and low-latency operation. The architecture leverages pipeline and parallel processing techniques to ensure continuous data propagation without stalling, thereby maximizing throughput.

Keywords: Feedforward-cutset-free (FCF), digital signal processing (DSP), feedforward, Field-programmable gate array (FPGA), implementation, high-speed computing, low-latency, multiply–accumulate (MAC) unit, parallel processing, power efficiency

INTRODUCTION

Recently, deep neural networks (DNNs) have emerged as powerful tools for various applications, including image classification and speech recognition. Because an enormous number of vector–matrix multiplication computations are required in a typical DNN application, a variety of dedicated hardware for machine learning has been proposed to accelerate the computations. In a machine learning accelerator, a large number of multiply–accumulate (MAC) units are included for parallel computations, and timing-critical paths of the system are often found in the unit. In any processor or machine learning application, the MAC operation is the fundamental building block. MAC consists of two operations: multiple and accumulator. In contemporary machine learning accelerators, parallelism is attained through the integration of numerous concurrent multiply–accumulate (MAC) units [1–5]. The basic arithmetic operations required for neural network computations were

***Author for Correspondence**
Swapnil A. Takale
E mail: swapnil.takale@sknscoe.ac.in

[1]Student, Department of Electronics and Telecommunication Engineering, SKN Sinhgad College of Engineering, Pandharpur, Maharashtra, India
[2]Assistant Professor, Department of Electronics and Telecommunication Engineering, SKN Sinhgad College of Engineering, Pandharpur, Maharashtra, India

Received Date: July 16, 2025
Accepted Date: December 08, 2025
Published Date: March 19, 2026

Citation: Dnyaneshwari L. Shinde, Alisha B. Mulani, Diksha A. Waghmare, Swapnil A. Takale. MAC Unit Implementation on FPGA. Journal of Microcontroller Engineering and Applications. 2026; 13(1): 29–37p.

performed using these MAC units. Therefore, the effectiveness and speed of MAC units have a significant impact on the overall performance of the accelerator. Multiply–accumulate (MAC) units are the main computational components of machine learning accelerators. Neural network layers are typically processed in parallel by instantiating a large number of MAC units. Each MAC unit generates partial sums that move toward the final result by performing accumulation and multiplication. The design of MAC units has a significant influence on the accelerator's overall performance, area, and power efficiency owing to their widespread use. The MAC unit serves as the timing-critical channel in many systems, and any inefficiencies in its architecture directly restrict the maximum operating frequency [6–10].

The design of the MAC unit is a major problem in high-performance hardware architectures, because it frequently contains timing-critical pathways in the system. MAC operation is an essential component of both machine learning accelerators and general-purpose processors. The two fundamental operations that constitute a MAC unit are multiplication and accumulation. MAC operations, which regularly accumulate the product of input data values and their accompanying weights, are used to compute partial sums during neural network activities. However, because of the latency caused by multipliers, traditional MAC systems frequently have lengthy critical path delays. The maximum clock frequency that can be achieved and the total system throughput may be reduced by this lengthy delay. The lengthy critical path connected to the multiplication process is one of the main obstacles in the MAC design. This delay limits the possible clock frequency and frequently determines the total latency of the MAC unit. The critical route can be shortened by pipelining techniques; however, this usually results in the addition of more registers, which increases the size and power consumption. For an effective MAC design, it is therefore crucial to optimize the accumulation stage and carefully balance the trade-offs between performance, area, and power [11–16].

In this study, we implemented an accumulator operation on an FPGA. We implemented the accumulator operation in conventional and improved versions and compared the results. Meanwhile, an MAC operation is performed in the machine learning algorithm to compute a partial sum, which is the accumulation of the input multiplied by the weight. However, such a structure still comprises a long critical path delay that is approximately equal to the critical path delay of a multiplier. In any processor or machine learning application, the MAC operation is the fundamental building block. MAC consists of two operations: multiple and accumulator [17–22].

LITERATURE SURVEY

The feedforward-cutset-free (FCF) pipelined multiply–accumulate (MAC) unit is an advanced architecture designed to enhance the efficiency of machine learning accelerators. Traditional pipelined MAC units often require numerous flip-flops to maintain data integrity, leading to increased area and power consumption. The FCF approach addresses this issue by selectively reducing the number of flip-flops without compromising the accuracy of the final output. This selective reduction is based on the observation that in many machine learning applications, only the final accumulated result is utilized, making intermediate values less critical. In this study, we propose a FCF pipelined MAC architecture that selectively removes superfluous pipeline registers, greatly increasing area and energy efficiency. The discovery that only the final aggregated output is used for further processing in many machine learning applications means that intermediate computing outputs do not need to be explicitly recorded or accessible. This observation serves as the basis for the proposed approach [23, 24]. The FCF design permits continuous feedforward data propagation while preserving functional correctness and high throughput by loosening the stringent feedforward-cutset requirement. The FCF architecture achieved significant improvements by relaxing the FCF rule. Studies have demonstrated that this design can lead to a 20% reduction in both energy consumption and area compared with conventional pipelined MAC units. Further optimization involves incorporating advanced components, such as EXOR full adders and 4:2 compressors, in the column addition stage. These modifications contribute to additional reductions in the area and power usage. For instance, research has shown that implementing these components can lead to further enhancements in efficiency, making the FCF pipelined MAC unit a

promising solution for high-performance machine learning accelerators [12–16]. By carefully lowering the number of flip-flops utilized in the design without compromising the accuracy of the final aggregated output, the FCF pipelining technique overcame these constraints. The fundamental finding that many machine learning workloads do not require intermediate computation results to be accurately registered at every pipeline level serves as the foundation for this solution. Rather, only the final collected value was utilized for further processing. By taking advantage of this feature, the FCF architecture streamlines the data flow via the MAC unit by relaxing the stringent feedforward-cutset requirement and eliminating unnecessary pipeline registers [17–22]. The suggested MAC design includes enhanced arithmetic components in the multiplier's column addition stage, such as 4:2 compressors and EXOR-based full adders, to further increase efficiency. These elements contribute to further reductions in area and power consumption by lowering the logic depth, minimizing the switching activity, and shortening the critical path. Selective pipelining and arithmetic optimization work together to create a simplified MAC design that achieves good performance with less hardware overhead. The FCF MAC design can be further improved by optimizing the internal arithmetic components. In particular, the complexity of partial product accumulation is greatly decreased by using 4:2 compressors and EXOR-based full adders in the multiplier column addition stage. These sophisticated arithmetic components reduce the switching activity, shorten the critical path, and decrease the logic depth, which results in further area and power consumption savings. According to previous studies, adding these elements to the FCF architecture increases the overall efficiency and makes the design even more appealing for high-performance applications [1–10].

In conclusion, a major development in MAC unit design for machine learning accelerators is the FCF pipelined MAC unit. The FCF architecture significantly increases the area, power, and energy efficiency without sacrificing computational accuracy by easing the conventional pipelining limits and utilizing enhanced arithmetic structures. For next-generation machine learning hardware, where high throughput, low power consumption, and effective resource usage are crucial, these features make it a viable and workable alternative. The FCF pipelined multiple-accumulate unit represents a significant advancement in MAC unit design, offering notable improvements in area and power efficiency. These enhancements make it particularly suitable for applications in machine learning accelerators, where performance and resource optimization are paramount.

PROPOSED METHODOLOGY

Working

FPGA Design Flow

The FPGA design flow is the process of designing digital circuits for FPGA devices, as shown in Figure 1. This involves the following key steps:

1. *Specification:* Define system requirements and constraints.
2. *High-level design:* Outline the system architecture and algorithms.
3. *RTL design:* Write hardware description code in VHDL/Verilog.
4. *Simulation:* Verify functionality using testbenches.
5. *Synthesis:* Convert HDL code into a gate-level netlist.
6. *Implementation:* Place and route the design on the FPGA.
7. *Bitstream generation:* Create a bitstream file to configure the FPGA.
8. *Hardware verification:* Test the design on actual hardware.
9. *Deployment:* Finalize for use and perform maintenance if needed.

This flow ensures the design meets performance, timing, and functional requirements.

HARDWARE DESCRIPTION

Synthesis and Simulation

Simulation

Simulation in FPGA design is the process of verifying the functionality of HDL (VHDL/Verilog) code before hardware implementation, as shown in Figure 2. It involves:

1. *Creating a testbench:* A separate code that applies inputs to the design and checks the outputs from Figure 3.
2. *Running the simulation:* Using tools such as Model Sim or Vivado Simulator to simulate the design behavior.
3. *Analyzing results:* Check if the design produces correct outputs and meets functional requirements.
4. *Debugging:* If issues are found, the code is modified and re-simulated until the design works as expected.

Simulation helps catch errors early, saving time and resources before hardware deployment, as shown in Figure 4.

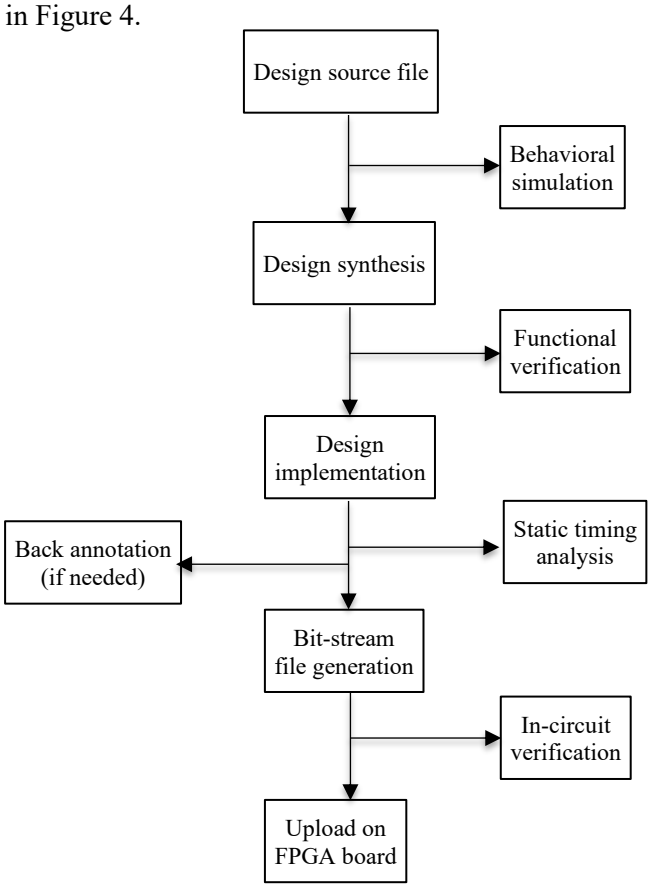

Figure 1. Block diagram.

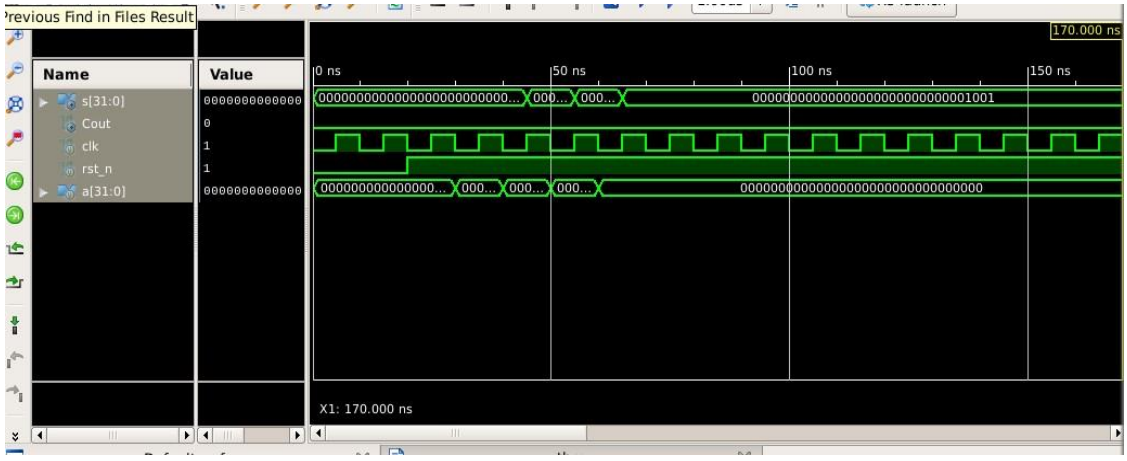

Figure 2. Simulation result.

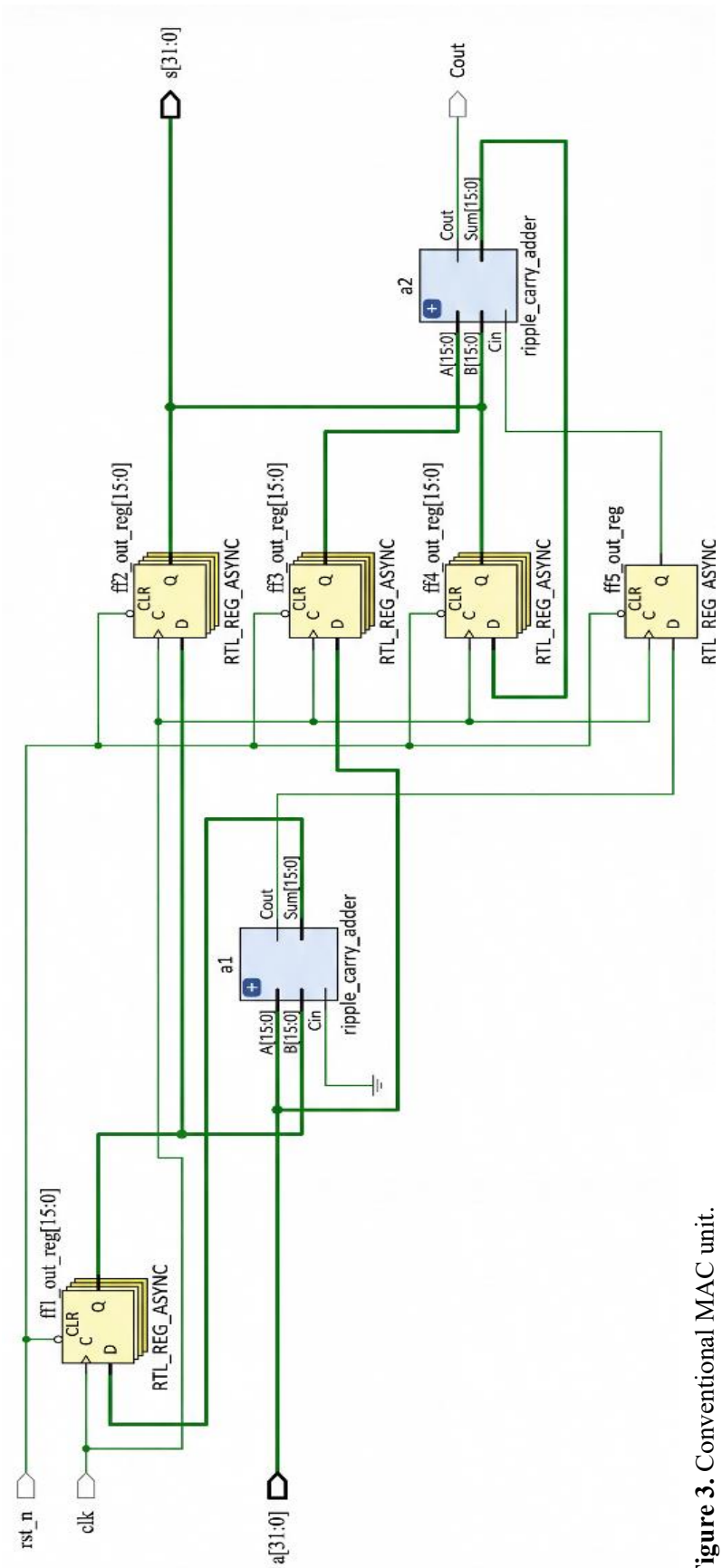

Figure 3. Conventional MAC unit.

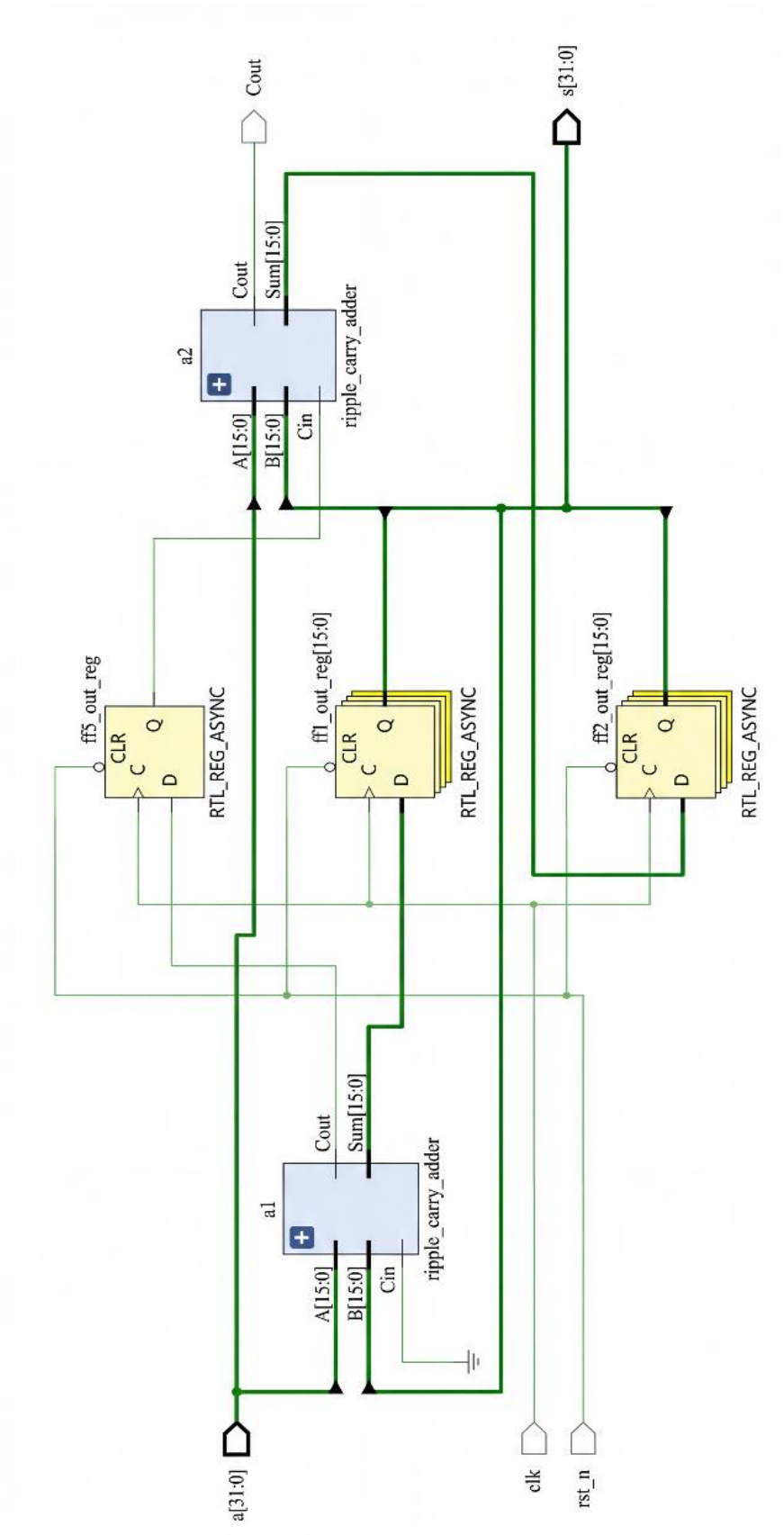

Figure 4. Proposed accumulator unit.

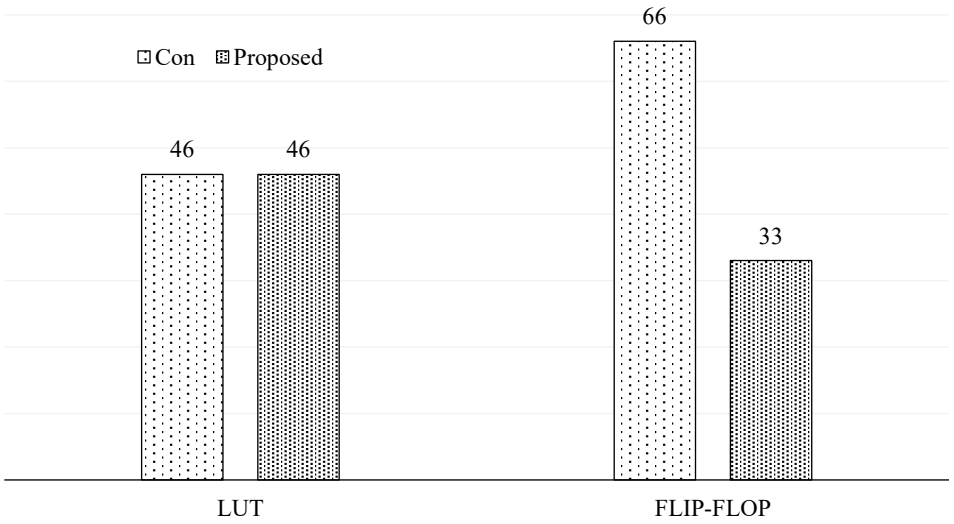

Figure 5. Area comparison of conventional and proposed MAC designs.

Table 1. Parameters of the basic and proposed accumulator.

Parameters	Basic accumulator	Proposed accumulator
LUT	46	46
Flip flop	66	33
Clock cycle	4	4

Synthesis

Synthesis in FPGA design is the process of converting HDL code (VHDL/Verilog) into a gate-level representation that the FPGA can understand. It involves:

1. *Translation:* The HDL code was translated into logic gates, flip-flops, and other digital components from Figure 5.
2. *Optimization:* The synthesis tool optimizes the design for speed, area, and power.
3. *Netlist generation:* A netlist (a list of components and their connections) was created to represent the hardware structure of the design.

Tools such as Xilinx integrated software environment (ISE) handle synthesis and prepare the design for the next step of implementation.

CONCLUSIONS

To assess the performance and area efficiency, an accumulator was constructed in the current work utilizing two distinct architectural techniques. The findings show that the overall efficiency of the MAC unit can be significantly impacted by the careful design of the accumulation stage. There is still room for improvement in the multiplication stage, even if the current implementation concentrates on improving the accumulator. By efficiently scheduling compression operations, the Dadda multiplier minimizes logic complexity and critical path latency, while reducing the number of reduction steps required during partial product accumulation. It is particularly appealing for high-speed and space-constrained hardware implementations because of this feature. It is anticipated that adding a Dadda multiplier to the MAC unit will preserve or even improve the computing performance while significantly reducing the total silicon area and power consumption. Furthermore, the decreased switching activity of the Dadda multiplier may help enhance energy efficiency, which is a crucial prerequisite for contemporary machine learning accelerators. A Dadda multiplier, which is renowned for its lower partial product tree height and enhanced area efficiency, could replace the traditional multiplier in subsequent studies. It is anticipated that adding a Dadda multiplier will further lower the

MAC unit's overall space and power consumption, improving the suitability of the design for high-performance machine learning accelerators. Integrating an advanced multiplier design, such as the Dadda multiplier, with the optimized accumulator architecture created in this study offers a promising direction for future research. High-performance, low-power machine learning applications would benefit more from such an improved MAC unit, especially in large-scale accelerators and FPGA-based systems, where resource efficiency and energy efficiency are crucial.

REFERENCES

1. Krizhevsky A, Sutskever I, Hinton GE. ImageNet classification with deep convolutional neural networks. Commun ACM. 2017;60(6):84–90. doi:10.1145/3065386.
2. Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition [Preprint]. arXiv. 2014;arXiv:1409.1556. doi:10.48550/arXiv.1409.1556.
3. Moons B, Uytterhoeven R, Dehaene W, Verhelst M. 14.5 Envision: a 0.26-to-10TOPS/W subword-parallel dynamic-voltage-accuracy-frequency-scalable convolutional neural network processor in 28nm FDSOI. 2017 IEEE International Solid-State Circuits Conference (ISSCC), San Francisco, CA, USA. 2017. p. 246–247. doi:10.1109/ISSCC.2017.7870353.
4. Chorowski JK, Bahdanau D, Serdyuk D, Cho K, Bengio Y. Attention-based models for speech recognition. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, editors. Advances in Neural Information Processing Systems. Vol. 28. Red Hook (NY): Curran Associates, Inc.; 2015.
5. Parashar A, Rhu M, Mukkara A, Puglielli A, Venkatesan R, Khailany B, et al. SCNN: an accelerator for compressed-sparse convolutional neural networks. ACM SIGARCH Comput Archit News. 2017;45(2):27–40. doi:10.1145/3140659.3080254.
6. Stelling PF, Oklobdzija VG. Implementing multiply-accumulate operation in multiplication time. Proceedings 13th IEEE Sympsoium on Computer Arithmetic, Asilomar, CA, USA. 1997. p. 99–106. doi:10.1109/ARITH.1997.614884.
7. Dadda L. Some schemes for parallel multipliers. Alta Frequenza. 1965;34:349–356.
8. Rastegari M, Ordonez V, Redmon J, Farhadi A. XNOR-Net: ImageNet classification using binary convolutional neural networks [Preprint]. arXiv. 2016;arXiv:1603.05279. doi:10.48550/arXiv.1603.05279.
9. Gao M, Pu J, Yang X, Horowitz M, Kozyrakis C. TETRIS: scalable and efficient neural network acceleration with 3D memory. In: Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17); 2017 Apr 8-12; Xi'an, China. New York (NY): Association for Computing Machinery; 2017. p. 751-764. doi:10.1145/3037697.3037702.
10. Macsorley OL. High-speed arithmetic in binary computers. Proc IRE. 1961;49:67–91. doi:10.1109/JRPROC.1961.287779.
11. Kamble A, Mulani AO. Home automation using Google Assistant. Purakala. 2023;32(1): 1071–1077.
12. Mulani AO, Bang AV, Birajadar GB, Deshmukh AB, Jadhav HM, Liyakat KK. IoT based air, water, and soil monitoring system for pomegranate farming. Ann Agri Bio Res. 2024;29(2):71–86.
13. Kulkarni TM, Mulani AO. Face mask detection on real time images and videos using deep learning. Int J Electr Mach Anal Des. 2024;2(1):22-30.
14. Thigale SP, Jadhav HM, Mulani AO, Birajadar GB, Nagrale M, Sardey MP. Internet of things and robotics in transforming healthcare services. Afr J Biol Sci (S Afr). 2024;6(6):1567–1575.
15. Pol DR, Mulani AO, Bhalerao M. Cloud based memory efficient biometric attendance system using face recognition. Stoch Model Appl. 2021;25(2):403–416.
16. Shelke D, Pawar P, Tate P, Shinde P, Mali AS. Trends & technologies in obstacle avoidance systems based on Arduino. Int J Adv Res Sci Commun Technol. 2025;5(4):440–446.
17. Kulkarni TM, Mulani AO. Deep learning based face-mask detection: an approach to reduce pandemic spreads in human healthcare. Afr J Biol Sci. 2024;6:783–795.
18. Mulani AO, Mane PB. DWT Based Robust Invisible Watermarking. Chisinau (Moldova): Scholars' Press; 2016.

19. Jadhav VS, Salunkhe SS, Salunkhe G, Yawle PR, Pol RS, Mulani AO, et al. IoT based health monitoring system for human. Afr J Biomed Res. 2024;27(3):2835-2841. doi:10.53555/AJBR.v27i3.1606.
20. Nagtilak AG, Ulegaddi SN, Mane M, Mulani AO. Automatic solar powered pesticide sprayer for farming. Int J Microwave Eng Technol. 2023;9(2):7-18.
21. Dandage AS, Rupnar VR, Pise TA, Mulani AO. Real-time language translation application using Tkinter. Int J Digit Commun Analog Signals. 2025;11(1):26–32.
22. Deo SS. Challenges in the implementation of legal aid services. J Const Law Jurisprud. 2025;8(2):33–40.
23. Deo SS. Legal aid in India: framework and policies. J Const Law Jurisprud. 2025;8(2):21–26.
24. Deo SS. Legal aid in criminal justice. J Hum Rights Law Pract. 2025;8(2):30–36.

