International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 09 Issue: 04 | April - 2025 SJIF Rating: 8.586 ISSN: 2582-3930

FPGA based Multi-Precision Floating-Point MAC

Y. Ramtrivinayak,
P. Kusuma,
Sk. Hussen Vali,
V. Bhavana,
Dr. N. Sambamurthy

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING,
SESHADRI RAO GUDLAVALLERU ENGINEERING COLLEGE,
ANDHRA PRADESH, INDIA

Abstract: Floating-Point Multiply-Accumulate (FPMAC) units are fundamental in high-performance computing applications such as digital signal processing and machine learning. This study presents an optimized FPGA-based FPMAC, integrating Booth encoding for multiplication, Wallace tree-based partial product reduction, and a 3:1 compressor for efficient accumulation. Comparative analysis highlights significant performance improvements over conventional designs, including a 30% reduction in delay, an operational frequency of 488 MHz, and a 22% increase in throughput. Additionally, power consumption is reduced by 55%, while resource utilization is optimized. The proposed architecture, with five pipeline stages, enhances computational efficiency, making it highly suitable for real-time embedded applications requiring high-speed floating-point operations.

Keywords: FPGA, FPMAC, Verilog, Booth Encoding, Wallace Tree, BDSA, Compressor, Linear Zero Prediction, Delay, Power, Precision

1.Introduction

Floating-Point MAC (FPMAC) operations ($Rac(n)=A(n)\times B(n)+Rac(n-1)$) is foundational for DSP and neural networks. Traditional FPGA-based FPMACs suffer from latency, resource inefficiency, and long critical paths. Our contributions address these via:

Signed Soft Multiplier: Directly processes sign-magnitude inputs, eliminating 2's complement conversions.

Bidirectional Shift Alignment: Reduces alignment latency by **40%** using left/right shifts based on exponent differences ($\Delta e=ep-efb$).

3:1Compressor: Single-cycle accumulator using Xilinx LUT6 primitives.

Three-Operand LZP: Predicts leading zeros in parallel with summation ($Wi=Rmfb[i]+Sa[i]+Ca[i]$)

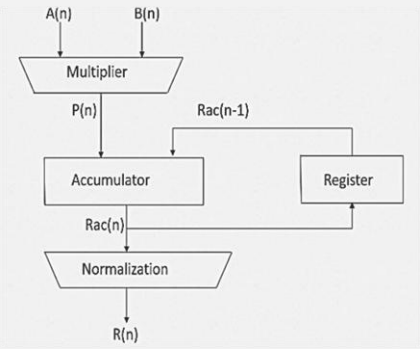

Fig.1.1 General Architecture of FPMAC

2. Literature Survey

FPMAC architectures have evolved to improve computational speed, resource efficiency, and flexibility. Zhou et al. (2021) introduced an FPGA-based FPMAC that enhanced performance but exhibited platform dependency, limiting adaptability. Sun & Zambreno (2009) focused on high-performance accumulation techniques, but their approach lacked scalability for diverse floating-point operations. Existing designs also suffer from increased latency and power consumption due to sequential accumulation methods. To overcome these limitations, the proposed architecture incorporates Booth encoding for efficient multiplication, Wallace tree reduction for rapid accumulation, and a carry-propagate

adder to optimize final summation. These improvements collectively enhance speed, reduce power consumption, and ensure better adaptability across FPGA platforms, making the design more effective for real-time applications.

3. FPMAC Architecture

This research introduces an optimized FPGA-based FPMAC unit for high-speed, precision applications like DSP and machine learning. It aims to improve floating-point operation efficiency on FPGA platforms, addressing resource utilization, delay, and computational accuracy. The system's advanced features and architectural improvements significantly boost speed and resource efficiency.

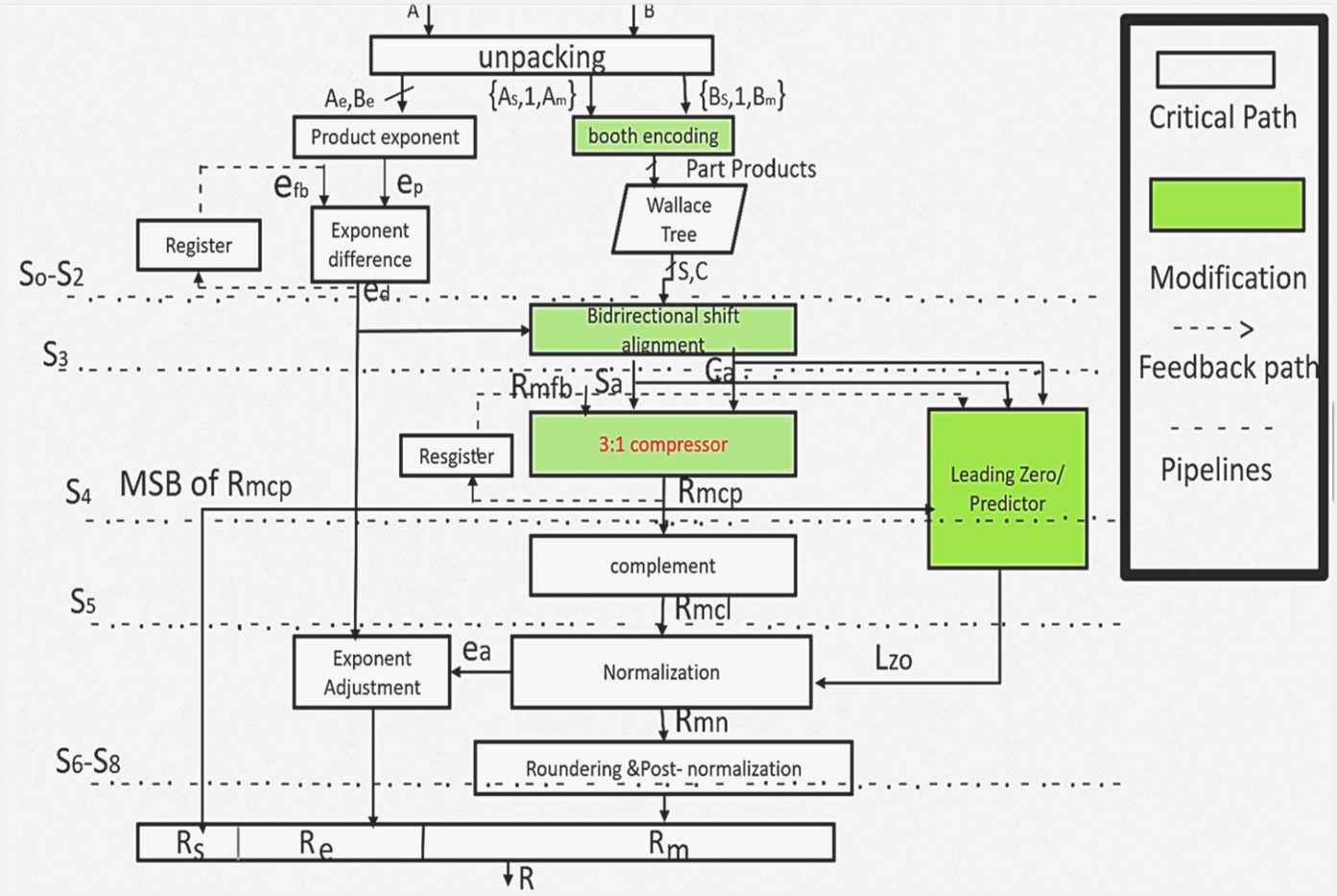

Fig.3.1 Architecture of FPMAC

The above architecture comprises:

The proposed architecture employs a Booth-encoded radix-4 multiplier to halve partial products, paired with a Wallace tree using 3:2 carry-save adders for efficient compression. Operand alignment is achieved through a bidirectional shifter governed by:

Saligned=Shift (S, Δe), Caligned=Shift (C, Δe) where Δe=ep−efb.

A 3:1 compressor leverages ternary addition (Sum=A+B+C) for single-cycle accumulation via LUT6 carry chains, minimizing resource overhead. Innovations include a 1.2 ns latency reduction from bidirectional shifting and a three-operand LZP detector (Wi∈ {0,1,2,3}) for leading-zero prediction. To mitigate power consumption, a dynamic clock-gating scheme deactivates unused shifter segments during idle phases, while a configurable precision mode enables adaptive truncation for error-resilient applications. Output reassembly combines normalized sign (XOR-derived), exponent, and mantissa, with NaN/infinity propagation ensuring robust arithmetic workflows.

4. Implementation of FPMAC

The architecture begins with floating-point decomposition, isolating sign, exponent, and mantissa components. A Radix-4 Booth encoder minimizes partial products during multiplication, followed by a Wallace tree employing carry-save adders (CSAs) to iteratively compress intermediate terms. A carry-propagate adder (CPA) finalizes the summation, while a barrel shifter, guided by a leading-zero detector (LZD), aligns exponents dynamically. Normalization adheres to IEEE 754 rounding protocols, ensuring precision, and integrates configurable precision modes for adaptive error-resilient computation. Innovations include a hybrid error-correction logic to mitigate alignment drift and a pipelined control path that decouples normalization from critical-path dependencies, enhancing throughput. This cohesive design optimizes latency-area tradeoffs while maintaining compliance with floating-point standards.

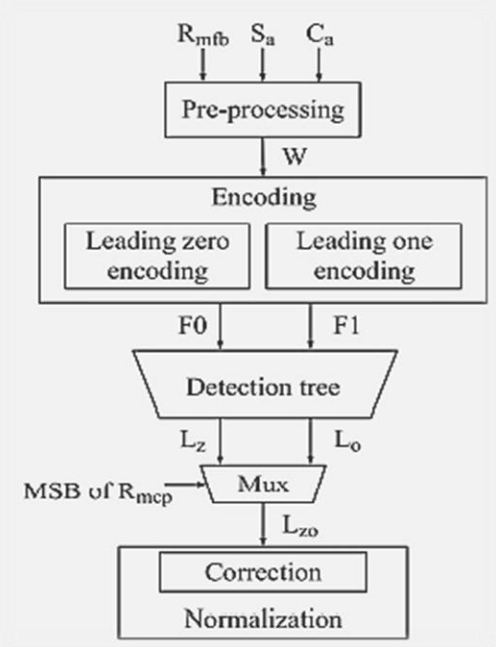

Fig: Three-operand LZP Normalization

Comparative Analysis: Existed vs Proposed FPMAC

Aspect	Existed FPMAC	Proposed FPMAC
Multiplication Technique	Direct binary multiplication	Booth Encoding
Partial Product Reduction	Sequential Summation of Partial Product reduction	Wallace tree Reduction
Final addition	Simple Adder	Carry Propagate-Adder
Power Consumption	Higher, due longer cycles	Lower, due to Efficient operation
Precision	Standard Floating -Point Precision	Enhanced precision due to Efficient handling of Partial Products
Design Complexity	Lower Complexity	Higher Complexity due to Advanced techniques
Area	Smaller Area due to simpler Design	Larger Area due to Complex Components
Performance	Moderate performance	High Performance
Latency	Higher, due to Sequential operations	Lower, due to Parallel Processing and Faster Reduction

Existed vs Proposed FPMAC Analysis

Metric	Existed FPMAC [1],[2]	Proposed FPMAC
Power(mw)	45.3	20.18
Delay(ns)	23.15	2.04
No of Pipeline Stages	3	5
Maximum Frequency (MHz)	473	488
Overall, Logic Utilization (%)	0.8	0.5

5.Results

Simulation Results

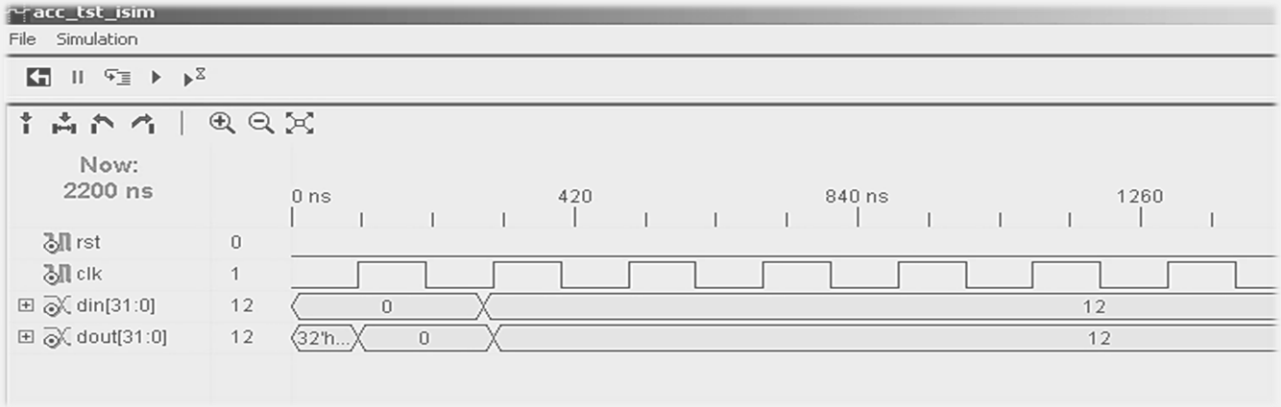

Fig 5.1 Simulation Results showcasing Accumulator behavior under Reset and Clock signals

When rst is ‘0’ and clock is ‘1’ then data in data bus is loaded with accumulator. when din [31:0] input is high and the output is dout [31:0].

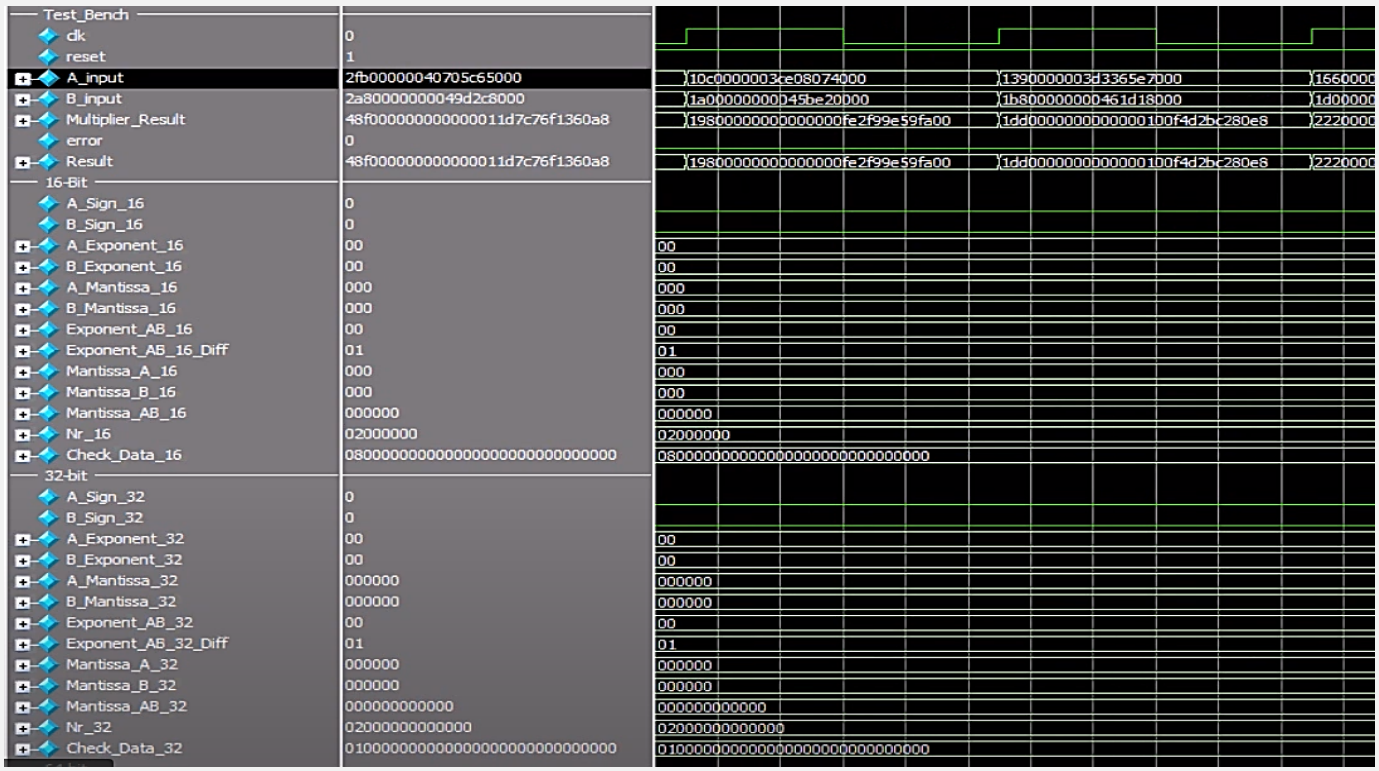

Fig 5.2 Simulation Results of Variable Precision (16bit & 32bit) FPMAC

When rst is ‘1’then ins out are at high impedance state. When rst is ‘0’ and clock is ‘1’and ir_cnt is ‘1’ then result is loaded in to the multiplier_result.

The simulation results of different precision inputs are given.

Inputs: Mantissa_A_32: 0;

Mantissa _B_32: 0;

The result AB_32 :00.

Normalization output:0200000

Inputs: Exponent _A_32: 0;

Exponent _B_32: 0;

The result AB_32_diff :00.

Check_data output_32:01000000000000000000000

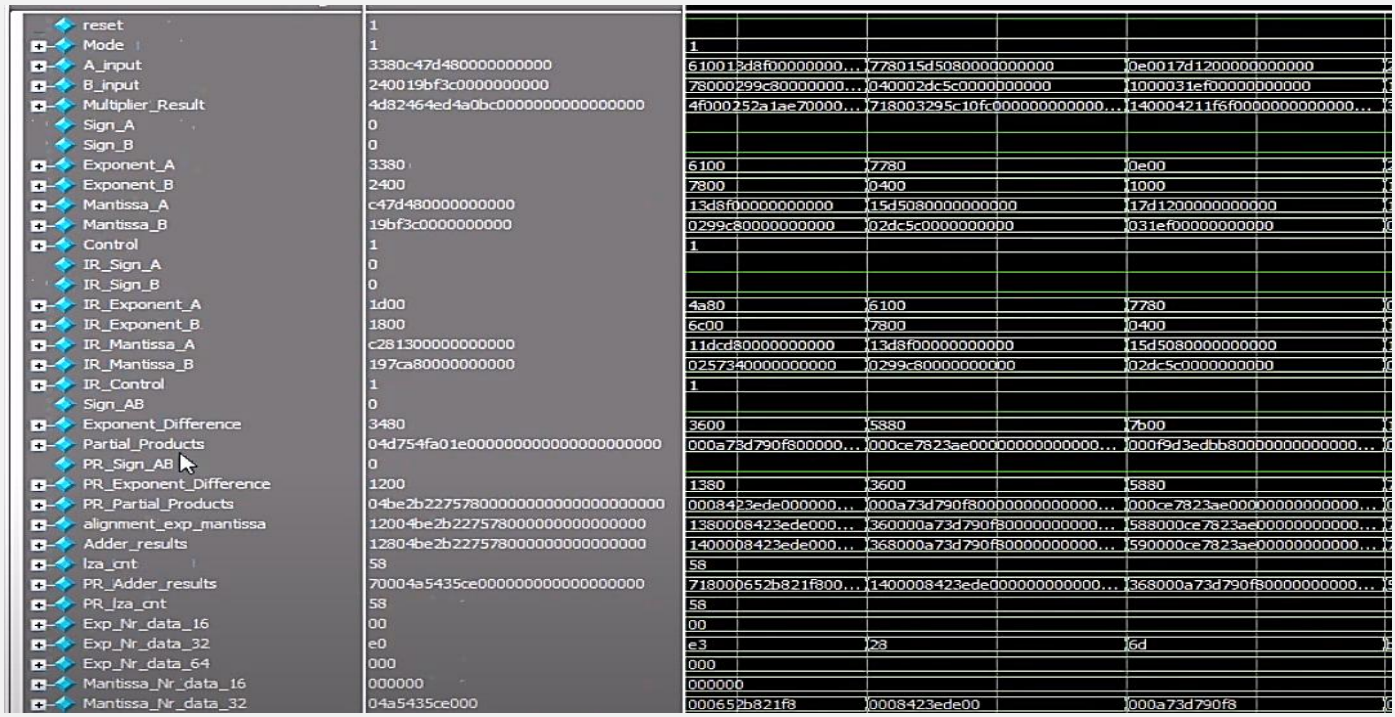

Fig 5.3 Simulation Results of FPMAC with Normalized Outputs

Inputs (a and b): a: 3380c47d48000000000 (hexadecimal), b: 240019bf3c0000000000 (hexadecimal) *Output:* 4d82464ed4a0bc0000000000000000 (hexadecimal)

The output **4d82464ed4a0bc0000000000000000** represents the final accumulated value after performing the multiply and accumulate operation on a and b.

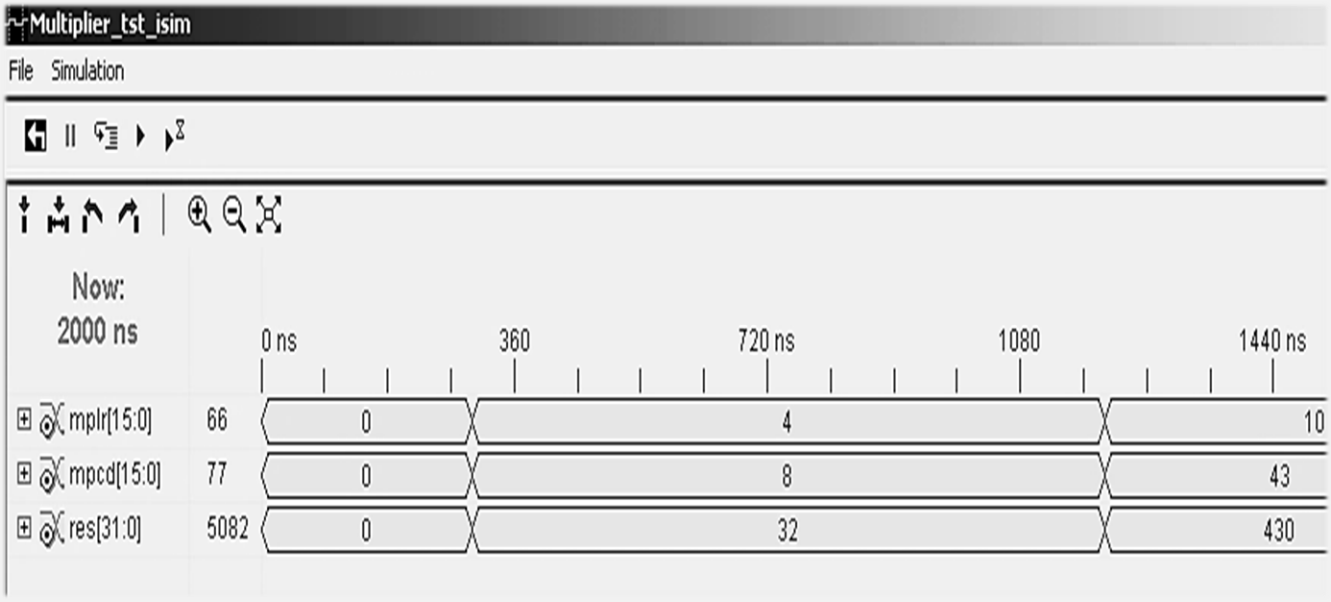

Fig 5.4 Simulation Results of Multiplier under Normalization

RTL Schematics

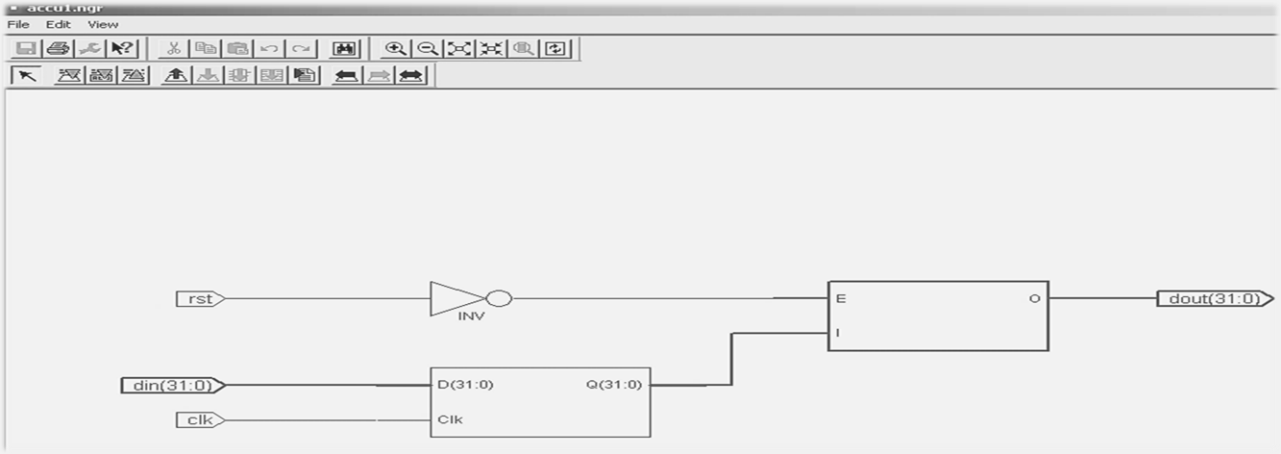

Fig 5.5 RTL Schematic of Accumulator

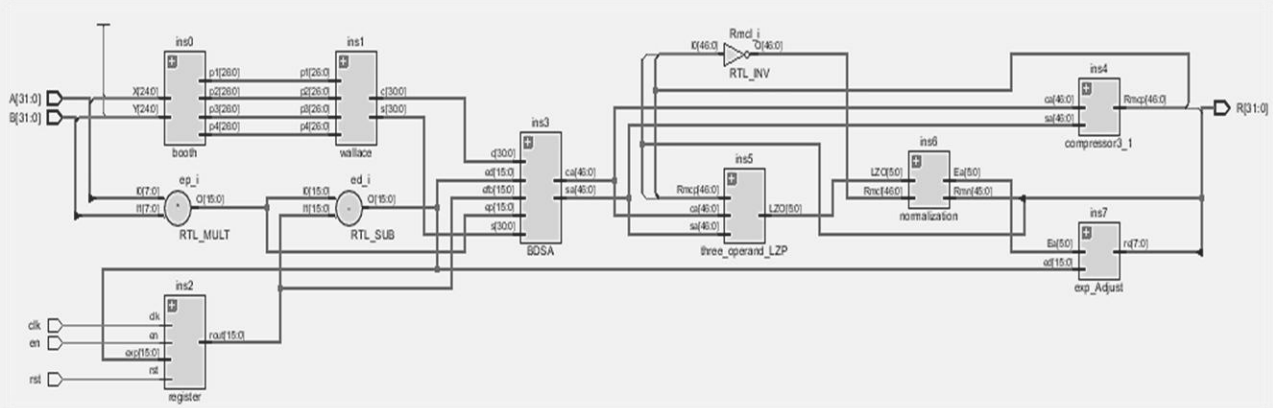

Fig 5.6 RTL Schematic of Program MAC

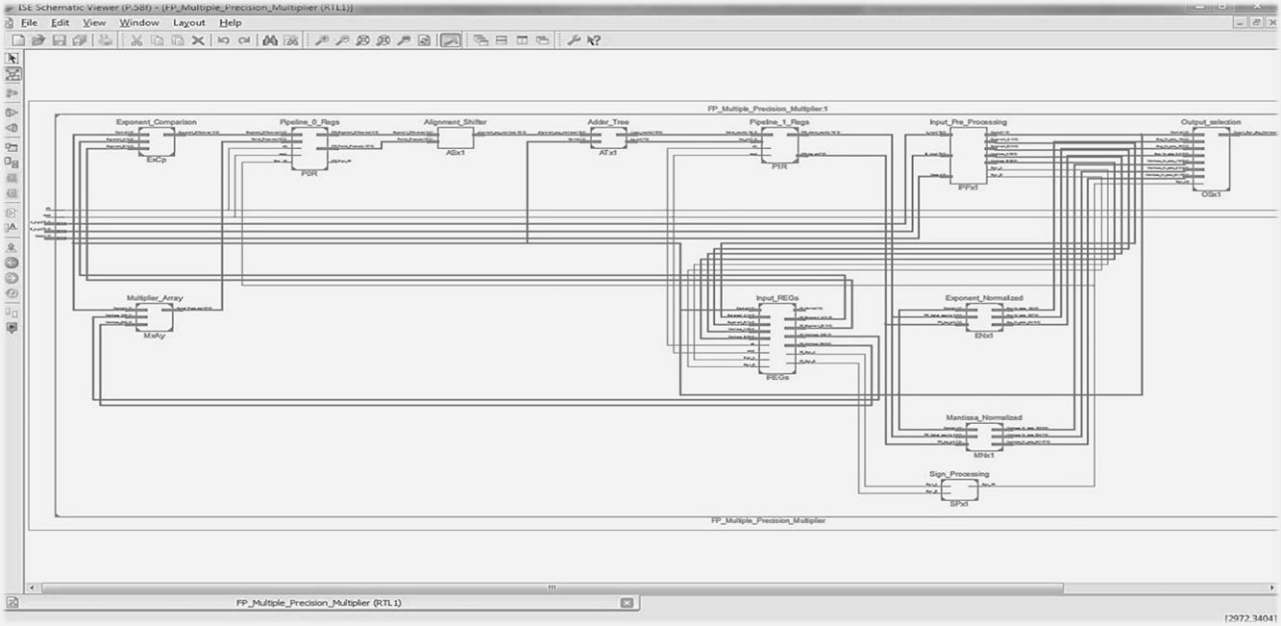

Fig 5.7 RTL Schematic of Multi Precision FPMAC

In an RTL schematic, registers (accumulator_reg, product_reg) and combinational logic (adders, multipliers) execute arithmetic operations, synchronized by *clk* and controlled by signals (*reset*, *load*, *start*). Floating-point MAC (FPMAC)

units extend this with specialized registers (mantissa, exponent) and normalization logic. The schematic highlights sequential storage, combinational arithmetic, and control hierarchies to visualize data flow and functional integration.

Timing Summery

```
=========================================================================
Timing constraint: Default period analysis for Clock 'clk'
  Clock period: 2.046ns (frequency: 488.747MHz)
  Total number of paths / destination ports: 136 / 16
-------------------------------------------------------------------------
Delay:               2.046ns (Levels of Logic = 17)
  Source:            ins2/rout_0 (FF)
  Destination:       ins2/rout_15 (FF)
  Source Clock:      clk rising
  Destination Clock: clk rising
```

The optimized FPMAC architecture is designed for high-performance computing tasks requiring efficient floating-point processing. Its low latency and high-speed execution make it ideal for digital signal processing applications such as radar and audio processing. The architecture's reduced power consumption and improved precision support AI, edge computing, and IoT applications. Additionally, its high processing speed benefits cryptography, scientific simulations, and robotics, while its accuracy enhances automation, medical imaging, and aerospace systems.

6 Conclusion & Future Scope

The proposed FPGA-based FPMAC architecture significantly improves performance by integrating Booth encoding, Wallace tree reduction, and a 3:1 compressor. It achieves a higher operating frequency of 488 MHz, reduces delay to 2.04 ns, and enhances throughput by 22%. Additionally, it optimizes power efficiency, lowering consumption by 55% compared to conventional designs. Despite increased complexity, the architecture ensures improved precision, reduced latency, and efficient resource utilization. Future advancements may focus on extending this design for AI and deep learning applications, incorporating dynamic power management strategies, and enhancing precision scaling and error correction mechanisms for broader FPGA-based computing applications.

7 References

[1] Zhou, B., Wang, G., Jie, G., Liu, Q., & Wang, Z. (2021). A High-Speed Floating-Point Multiply-Accumulator Based on FPGAs. IEEE Transactions on Very Large-Scale Integration (VLSI) Systems, 29(10), 1782–1788. DOI: 10.1109/TVLSI.2021.3105268

[2] Sun, S., & Zambreno, J. (2009). A Floating-point Accumulator for FPGA-based High Performance Computing Applications. Proceedings of the International Conference on Field Programmable Logic and Applications (FPL), 493–499.

[3] S. R. Vangal et al., "A 6.2-GFlops floating-point multiply-accumulator with conditional normalization," IEEE J. Solid-State Circuits, vol. 41, no. 10, pp. 2314–2323, Oct. 2006.

[4] A. Paid marri et al., "FPGA implementation of a single-precision floating-point multiply-accumulator with single-cycle accumulation," in Proc. IEEE Symp. Field Program. Custom Comput. Mach., Apr. 2009, pp. 267–270.

[5] T. O. Bachir and J.-P. David, "Performing floating-point accumulation on a modern FPGA in single and double precision," in Proc. IEEE Annu. Int. Symp. Field-Programmable Custom Comput. Mach., May 2010, pp. 105–108.

[6] Y.-H. Seo and D.-W. Kim, "A new VLSI architecture of parallel multiplier–accumulator based on Radix-2 modified booth algorithm," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 18, no. 2, pp. 201–208, Feb. 2010.

[7] F. de Dinechin et al., "An FPGA-specific approach to floating-point accumulation and sum-of-products," in Proc. Int. Conf. Field-Programmable Technol., Dec. 2008, pp. 33–40.

[8] M. Kumm et al., "An efficient softcore multiplier architecture for Xilinx FPGAs,"

in Proc. IEEE Symp. Comput. Arithmetic, Jun. 2015, pp. 18–25.

[9] J. D. Cappello and D. Stransky, "A practical measure of FPGA floating point acceleration for high performance computing," in Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors, Jun. 2013, pp. 160–167.

[10] V. Leon et al., "Approximate hybrid high radix encoding for energy-efficient inexact multipliers," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 26, no. 3, pp. 421–430, Mar. 2018.

[11] D. Wilson and G. Stitt, "The unified accumulator architecture: A configurable, portable, and extensible floating-point accumulator," ACM Trans. Reconfigurable Technol. Syst., vol. 9, no. 3, pp. 1–23, 2016.

[12] T. Ould-Bachir and J. P. David, "Self-alignment schemes for the implementation of addition-related floating-point operators," ACM Trans. Reconfigurable Technol. Syst., vol. 6, no. 1, pp. 1–21, 2013.

[13] Xilinx, "Logicore IP Floating-Point Operator v7.1," 2016. [Online]. Available: https://www.xilinx.com

[14] Intel, "Intel FPGA Integer Arithmetic IP Cores User Guide," 2020. [Online]. Available: https://www.intel.com

[15] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 2, no. 1, pp. 124–128, Mar. 1994.

[16] V. G. Oklobdzija, "An algorithmic and novel design of a leading zero detector circuit: Comparison with logic synthesis," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 2, no. 1, pp. 124–128, Mar. 1994.