Published by :
https://www.ijert.org/
An International Peer-Reviewed Journal

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

FPGA Implementation of Single Precision Floating Point Multiplier using Booth Recoding Algorithm

Prince Mishra
Electronics and Instrumentation Engineering
Odisha University of Technology and Research,
Bhubaneswar, India

***Abstract*—Through this paper, we focus on implementing a Single Precision Floating Multiplier using IEEE 754 Standards. By reducing partial product generation and addition units, this implementation offers benefits such as faster results, reduced power consumption and reduction in the utilization of hardware resources. Moreover, the implementation deals with multiplication of both signed and unsigned numbers. The paper presents a comparative analysis with a 32-bit multiplier performance in terms of power consumption and FPGA hardware resource utilization. The proposed 32-bit multiplier is designed using Verilog HDL and implemented through Xilinx Vivado 2025.1 software for Xilinx Virtex-7 FPGA.**

Keywords Used- FPGA; Booth Recording; Single Precision; signed multiplier; Verilog HDL.

I. INTRODUCTION

Floating-point multipliers serve as critical computational kernels in high-performance digital signal processing (DSP) architectures, specifically within Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters.

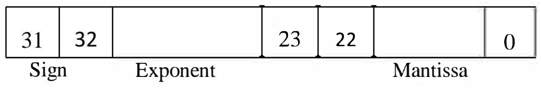


Figure 1.Single precision floating point representation

The multiplication involves concurrent XOR-based sign processing, biased exponent addition (EA+EB-127), and normalized mantissa multiplication. To address the hardware complexity of partial product summation in high-order filters, the proposed architecture employs the Booth Recoding algorithm. This optimizes the 24-bit mantissa multiplication by reducing partial product rows, significantly enhancing throughput and lowering power consumption.

The paper is organized as follows. Section II presents the floating point multiplier algorithm. Section III presents the design of booth recoding multiplier. Section IV describes details of the proposed architecture and its implementation. Section V and VI presents partial product generation and addition respectively. Section VII contains the proposed architecture for the multiplier. Results are displayed in section VIII. Finally the conclusion in section IX marks the end of the paper.

II. FLOATING POINT MULTIPLIER ALGORITHM

In accordance with the IEEE 754 standard, a 32-bit single-precision floating-point datum is partitioned into three distinct functional components: a 1-bit sign (S), an 8-bit biased exponent (E), and a 23-bit fractional mantissa (M). The multiplication of two such operands involves concurrent, independent operations across these fields to derive the product. The detailed description is defined as below.

1. Calculation of the sign bit; i.e. SA XOR SB.
2. Exponent is calculated by adding the exponent EA and EB After that, bias subtraction by 127 to get the final exponent, i.e. EA+EB-127.
3. An implicit leading bit is appended to each 23-bit mantissa to form 24-bit operands. These are processed through a Booth Recoding multiplier to generate a 48-bit intermediate product
4. Normalizing the result, to get the required 23bit mantissa, ensuring the output adheres to the standard format.
5. Combine the calculated sign, exponent and mantissa components to get the desired multiplication result.

III. DESIGN OF BOOTH RECODING MULTIPLIER

A. *24x24 bit multiplier*

The proposed Booth recoding multiplier architecture accepts two 24-bit inputs, serving as the multiplier and multiplicand. Two control signals are included to specify whether the multiplier and multiplicand are treated as signed or unsigned integers.

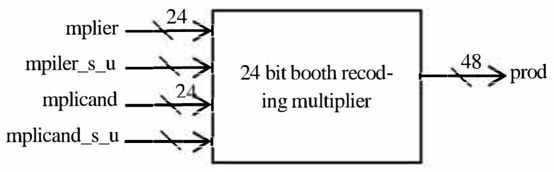


Figure 2.24- bit booth recording multiplier top module

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

Table 1. Signal Description of top module

Signal Name	Width	Source	Description
mplier	24	input	Top module multiplier input
mplier_s_u	1	input	1= multiplier is signed, 0 =multiplier is un-signed
mplicand	24	input	Top module multipli-cand input
mplicand_s_u	1	input	1 = multiplicand is signed, 0 =multiplicand is unsigned
prod	48	output	Output from the multi-plier block

B. *Mathematical Representation of Signed Number*

2's complement representation of A

$$= -2^{23}a_{23} + 2^{22}a_{22} + (2-1)2^{21}a_{21} + 2^{20}a_{20} + 2^{19}a_{19} + (2-1)2^{18}a_{18} + 2^{17}a_{17} + 2^{16}a_{16} + (2-1)2^{15}a_{15} + (2-1)2^{14} + \dots\dots\dots\dots\dots. +2^{2}a_{2} + (2-1)2^{1}a_{1} + 2^{0}a_{0} + 2^{0}a_{-1}$$

Where $a_{-1} \equiv 0$

$$\begin{aligned} = -2^{23}a_{23} + 2^{22}a_{23} + 2^{22}a_{22} + \\ -2^{21}a_{21} + 2^{20}a_{20} + 2^{20}a_{19} + \\ -2^{19}a_{19} + 2^{18}a_{18} + 2^{18}a_{17} + \\ -2^{16}a_{16} + 2^{15}a_{15} + 2^{15}a_{14} + \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ -2^{3}a_{3} + 2^{2}a_{2} + 2^{2}a_{1} + \\ -2^{1}a_{1} + 2^{0}a_{0} + 2^{0}a_{-1} \end{aligned}$$

Where $a_{-1} \equiv 0$

$$\begin{aligned} = 2^{22}(-2a_{23} + a_{22} + a_{21}) \\ 2^{20}(-2a_{21} + a_{20} + a_{19}) \\ 2^{18}(-2a_{19} + a_{18} + a_{17}) \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ 2^{2}(-2a_{3} + a_{2} + a_{1}) \\ 2^{0}(-2a_{1} + a_{0} + a_{-1}) \end{aligned}$$

Where $a_{-1} \equiv 0$

$$=\sum_{i=1}^{11} 2^{2i}(-2a_{2i+1} + a_{2i} + a_{2i-1})$$

$$=\sum_{i=1}^{11} 2^{2i} f_{2i} \cdots \dots\dots\dots\dots\dots\dots\dots (1)$$

Where $f_{2i} = -2a_{2i+1} + a_{2i} + a_{2i-1}$

C. *Mathematical representation of unsigned numbers*

$$\begin{aligned} = -2^{24}a_{24} + 2^{23}a_{23} + 2^{23}a_{22} \\ -2^{22}a_{22} + 2^{21}a_{21} + 2^{21}a_{20} \\ -2^{20}a_{20} + 2^{19}a_{19} + 2^{19}a_{18} \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ -2^{2}a_{2} + 2^{1}a_{1} + 2^{1}a_{0} \\ -2^{0}a_{0} + 2^{-1}a_{-1} + 2^{-1}a_{-2} \end{aligned}$$

Where $a_{-1} \equiv a_{-2} \equiv 0$

$$\begin{aligned} = 2^{23}(-2a_{24} + a_{23} + a_{22}) \\ + 2^{21}(-2a_{22} + a_{21} + a_{20}) \\ \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \\ + 2^{1}(-2a_{2} + a_{1} + a_{0}) \\ + 2^{-1}(-2a_{0} + a_{-1} + a_{-2}) \end{aligned}$$

Where $a_{-1} \equiv a_{-2} \equiv 0$

$$= \sum_{i=0}^{12} 2^{2i-1}(-2a_{2i} + a_{2i-1} + a_{2i-2})$$

$$= \sum_{i=0}^{12} 2^{2i-1} f_{2i} \dots\dots\dots\dots\dots\dots (2)$$

Where $f_{2i-1} = -2a_{2i} + a_{2i-1} + a_{2i-2}$

By comparing Equations (1) and (2), it becomes evident that integer A can be processed prior to the recoding of both basic unsigned and signed integers.

Table 2.Booth recoding truth table

a_{2i+1}	a_{2i}	a_{2i-1}	f_{2i}	$\bar{F}$ $(+/-)$	F^{1} $(\times 1/0)$	F^{2} $(\times 2/0)$
0	0	0	0	0	0	0
0	0	1	1	0	1	0
0	1	0	1	0	1	0
0	1	1	2	0	1	1
1	0	0	-2	1	1	1
1	0	1	-1	1	1	0
1	1	0	-1	1	1	0
1	1	1	0	0	0	0

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 03 , March - 2026

IV. SUB-MODULES OF IMPLEMENTED BOOTH RECODING MULTIPLIER

A. 25th bit extension unit

The proposed architecture employs a unified signed multiplier core to facilitate both signed and unsigned arithmetic operations. To preserve the full dynamic range during unsigned multiplicatio-n and prevent magnitude truncation, the 24-bit operands undergo a bit-width expansion at the most significant bit (MSB) position. For unsigned operands, this 25th bit is initialized to zero. In case of a signed operation, the 25th bit replicates the 24th bit (MSB) to follow up two's compliment integrity. This preprocessing stage ensures that the subsequent booth recording logic can process both number formats using a singular, hardware- efficient internal data path.

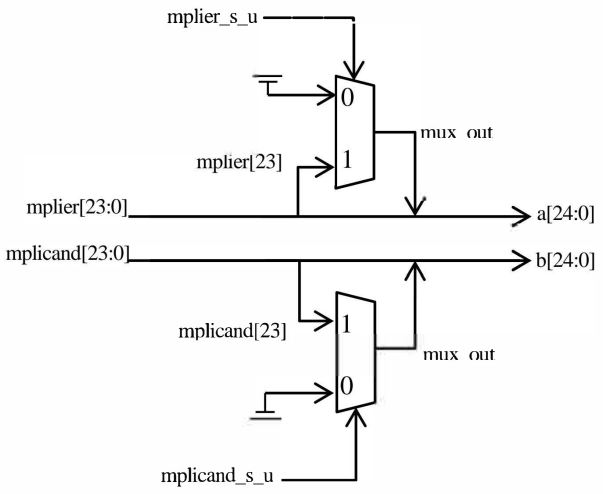


Figure 3.25th bit extension unit block diagram

B. Preprocessing/F block unit formation

The preprocessing unit conditions the operand for the Booth recoding algorithm to effectively minimize the partial product count. In this 24-bit architecture, the 25-bit operand is first initialized by appending a logic "0" at the least significant bit (LSB) position, establishing the essential reference bit ($a_{-1} = 0$) for the initial Booth recoding cycle. This modified 26-bit sequence is then partitioned into overlapping three-bit groupings, designated as "F-blocks", where the most significant bit (MSB) of each block serves as the LSB for the subsequent group. To achieve architectural bit-width alignment for the final three-bit grouping, an additional bit is appended at the MSB position, extending the sequence to 27 bits. This final MSB is a direct replication of the 25th bit, a technique that ensures sign integrity and preserve the numerical value for both signed and unsigned formats during the partial product generation phase.

Extra bit added for block formation	Extended 25 bit number(result of 25bit extension unit)	0 added in the LSB
a[24]	a[24:0]	0

Table 3. Extended 26 bit extension unit

The preprocessing stage concludes with the operand expanded to a total width of 27 bits. This transformation is achieved through combinational rewiring to form overlapping F blocks without further implementation of hardware.

Table 4. F block formation

F0	$\{a_1, a_0, 0\}$
F2	$\{a_3, a_2, a_1\}$
F4	$\{a_5, a_4, a_3\}$
F6	$\{a_7, a_6, a_5\}$
F8	$\{a_9, a_8, a_7\}$
F10	$\{a_{11}, a_{10}, a_9\}$
F12	$\{a_{13}, a_{12}, a_{11}\}$
F14	$\{a_{15}, a_{14}, a_{13}\}$
F16	$\{a_{17}, a_{16}, a_{15}\}$
F18	$\{a_{19}, a_{18}, a_{17}\}$
F20	$\{a_{21}, a_{20}, a_{19}\}$
F22	$\{a_{23}, a_{22}, a_{21}\}$
F24	$\{a_{24}, a_{24}, a_{23}\}$

V. Partial Product Generation

The partial product generation unit utilizes the 13 designated "F" blocks to derive the corresponding control signals- $\bar{F}, F^1$ and F^2 based on the truth table logic defined in Table 2. These signals drive the hardware realization required for bit manipulation to produce the necessary partial product rows. The hardware im-plementation for $\bar{F}, F^1$ and F^2 is listed below.

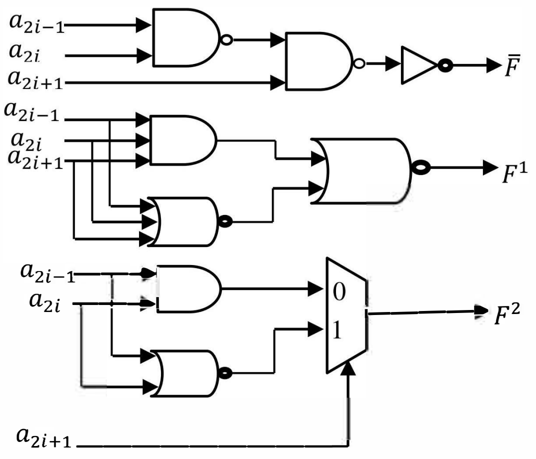


Figure 4. Hardware realization of $\bar{F}, F^1$ and F^2

Note:

1 . $\bar{F}$ is high when f_{2i} is -ve, otherwise low.

2. F^1 is high for $f_{2i} \neq 0$, otherwise low.

3. F^2 is high for $f_{2i} = \pm 2$,otherwise low.

The Booth recoding hardware processes each F-block concurrently to generate the corresponding control signals $\bar{F}, F^1$ and F^2 .For instance, processing the F_0 block yields the specific signals $\overline{F_0}, {F_0}^1, {F_0}^2$while subsequent blocks such as F_2 through F_{24} generate their respective control triplets via parallel hardware blocks. By leveraging this recoding algorithm, the architecture effectively minimizes the total count of partial products, requiring only 13 rows for the final summation stage. To generate the ROW#0 partial product, the F_0 signal triplet is applied to the 25-bit preprocessed multiplicand "b". This functional mapping is repeated for ROW#2 using the F_2control signals, and the process continues across all remaining rows to complete the partial product generation phase.

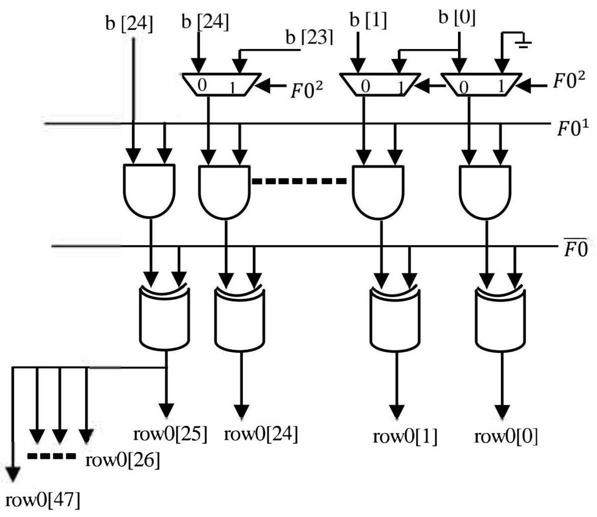


Figure 5. ROW#0 calculation using $\overline{F0}, F0^1$ and $F0^2$

Similar structures can be used for calculation of other ROW's

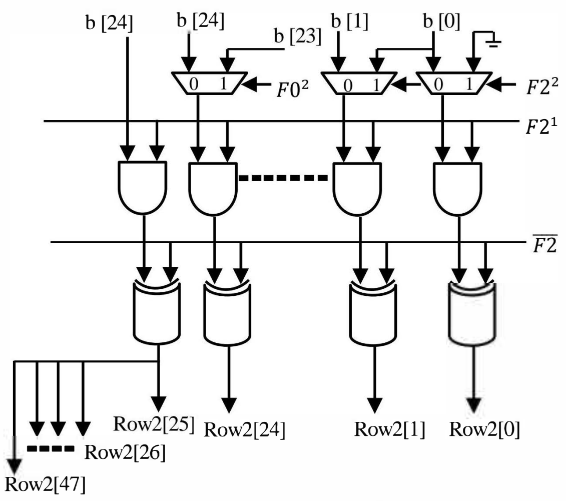


Figure 6. ROW#2 calculation using $\overline{F2}, F2^1$ and F^2

VI. PARTIAL PRODUCT ADDITION UNIT

The partial product summation unit performs the addition of all relevant rows, specifically ROW#0 through ROW#24, by utilizing a network of full adders and half adders. To maximize computational throughput, carries are propagated diagonally to the left and downward throughout the array. However, upon reaching the terminal ROW#24, the absence of a succeeding row necessitates a shift to horizontal carry propagation. This horizontal transition is architecturally feasible because the lower augend inputs remain unoccupied, allowing the final stage to complete the summation without a downward path.

In instances where operand *b* is negative, the system must calculate its 2's complement to ensure mathematical accuracy. While the structures in Figures 4 and 5 utilize XOR operations for bitwise inversion, the required increment of 1 at the LSB position must still be integrated. To address this, a dedicated ROW#-1 is introduced into the architecture, which facilitates the addition of the "1" bit at the aligned LSB position for each relevant row. This modification ensures the hardware correctly implements signed multiplication without disrupting the primary adder tree.

ROW#-1= $\{24'b0,\overline{F24},\ 0\ ,\overline{F22},\ 0\ ,\overline{F20},\ 0\ ,\overline{F18},\ 0\ ,\overline{F16}, 0\ ,\overline{F14}, 0\ ,\overline{F12}, 0\ ,\overline{F10}, 0\ ,\overline{F8}, 0\ ,\overline{F6}, 0\ ,\overline{F4}, 0\ ,\overline{F2}, 0\ ,\overline{F0}\}$

A. *Addition Stage*

The initial addition stage integrates ROW#-1, ROW#0, and ROW#2, with the latter being left-shifted by two positions to align with its binary weight. In the subsequent stage, the architecture sums the intermediate result from the first stage with ROW#4, which is left-shifted by four positions, alongside the carry bits generated during the first stage, shifted left by one position. This iterative process continues systematically, where each successive stage accumulates the previous sum and carry results with the next even-indexed row at its respective bit alignment.

To maintain precision, each partial product is shifted to its appropriate power-of-two significance within the array. This Carry Save Adder (CSA) topology defers final carry propagation to the last stage, simultaneously processing the previous sum, previous carry, and new partial product. By minimizing critical path delay compared to ripple-carry methods, this reduction tree ensures the high-speed performance necessary for real-time 32-bit floating-point multiplication.

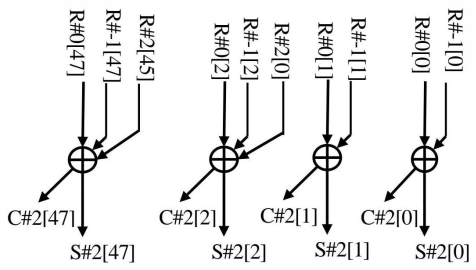


Figure 7.First stage addition block

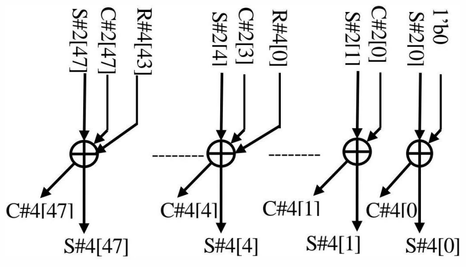


Figure 8. Second stage addition block

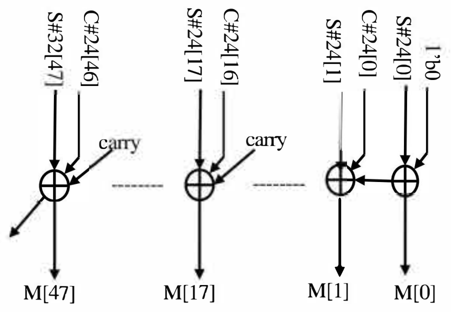


Figure 9. Final stage addition block

VII. Proposed Architecture Of Floating Point Multiplier

The proposed architecture for Single Precision Floating Point Multiplier using a 24bit multiplier using booth recoding algorithm is given in Fig.10

From the calculation perspective whole floating-point multiplication is divided into four sections.

A. Sign section
B. Exponent section
C. Mantissa section
D. Normalization section

A. Sign Section

In the sign section, the final result's sign bit is determined through a logical XOR operation applied to the sign bits of both input operands. This logic ensures that if the signs differ, the result is negative, whereas identical signs yield a positive result. The specific logic gates and outcomes for this process are detailed in the truth table provided in Table-5.

Table 5. Sign bit operation

E_A	E_B	Sign
0	0	0
0	1	1
1	0	1
1	1	0

B. Exponent section

In this section, the two operands are integrated via an 8-bit ripple carry adder, as illustrated in Fig. 11. Following the addition, the result must be normalized by subtracting a bias of 127 to achieve the final exponent. This subtraction is efficiently executed using the 2's complement method, ensuring the hardware maintains consistent logic for both addition and subtraction processes.

C. Mantissa section

The Mantissa computation represents the core performance bottleneck of the floating-point multiplier, necessitating a high-speed 24x24 bit binary multiplier to process the operands. This design utilizes the Booth recoding procedure to streamline the multiplication of the 23-bit mantissas (plus the implicit leading bit). By appending a sign bit to accommodate signed-number logic, the mantissas are converted into a Booth-encoded format that identifies repeating bit patterns. This encoding effectively minimizes the volume of partial products, condensing multiple operations into larger, collective groups to accelerate the hardware execution. Finally, these partial products are accumulated carefully maintaining their relative bit positions and signs to produce the definitive product of the mantissa multiplication.

D. Normalization section

The exponent and mantissa are normalized in the Normalization section. Normalization is completed based on the 47th bit, which is the outcome of the 24x24 bit binary multiplier. The mantissa is normalized to 23 bits by taking the 46th to 24th bit position number and increasing the exponent by decimal value one when the 47th bit of the 24X24 bit binary multiplier is binary one. The mantissa is normalized to 23 bits by taking the 45th to 23rd bit position number and there is no increase in the exponent when the 47th bit of the 24X24 bit binary multiplier is binary zero.

VIII. Results

The design was successfully synthesized for the Xilinx Virtex-7 FPGA (Device: xc7v585tffg1157-1) using the XST tool within the Xilinx 14.4 environment. Functional verification via a comprehensive test bench confirmed that the Booth recoding architecture effectively streamlines hardware resource utilization by minimizing partial product generation. This reduction in complexity, combined with the efficient handling of both signed and unsigned 32-bit operands according to IEEE 754 standards, makes the implementation a reliable solution for high-speed, power-efficient digital signal processing applications.

Table 6. FPGA hardware utilization

Parameter		**Utilization**
Bonded IOB		96
Slice	SLICEL	122
	SLICEM	110
LUT as Logic	using o6 output only	628
	using o5 and 06	157

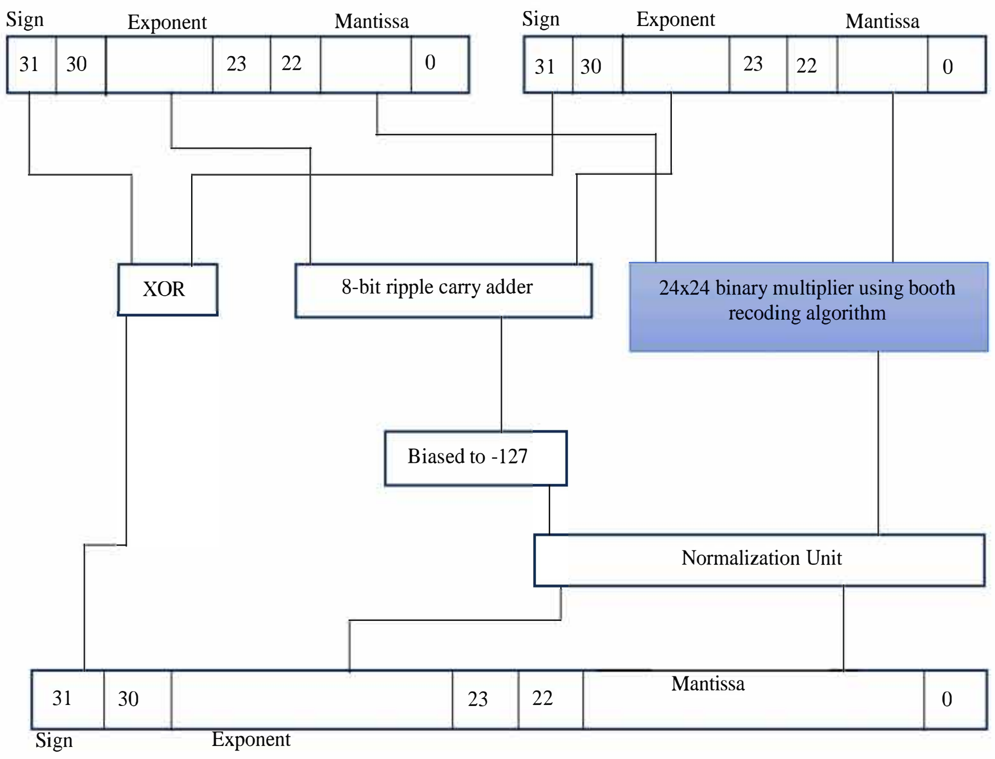


Figure 10. Proposed architecture of a single precision floating point multiplier

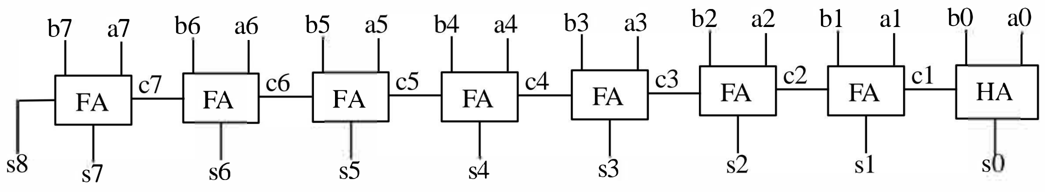
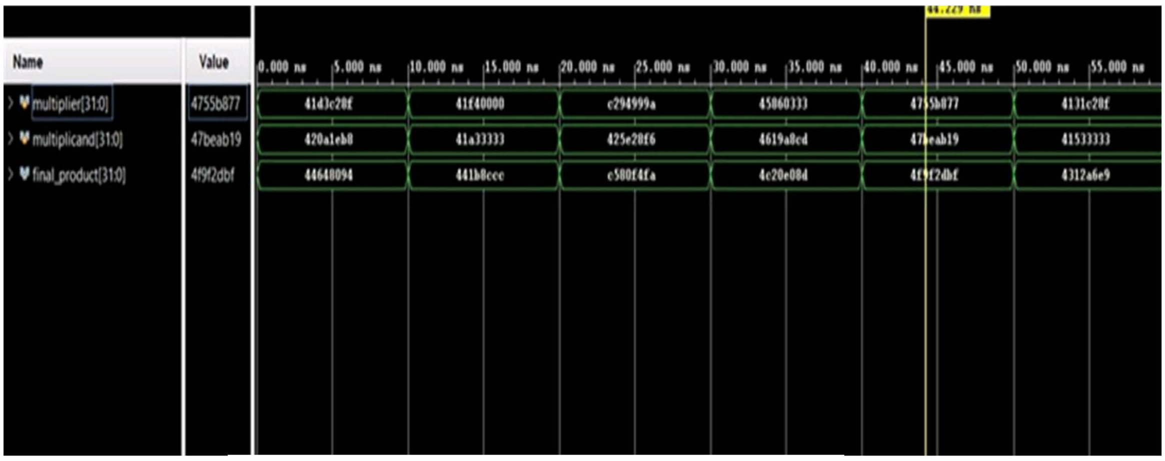


Figure 11. Eight bit ripple carry adder



```
multiplier[31:0]     4755b877  41d3c28f  41f40000  c294999a  45860333  4755b877  4131c28f
multiplicand[31:0]   47beab19  420a1eb8  41a33333  425e28f6  4619a8cd  47beab19  41533333
final_product[31:0]  4f9f2dbf  44648094  441b8ccc  c580f4fa  4c20e08d  4f9f2dbf  4312a6e9
```

Figure 12. Simulation waveform

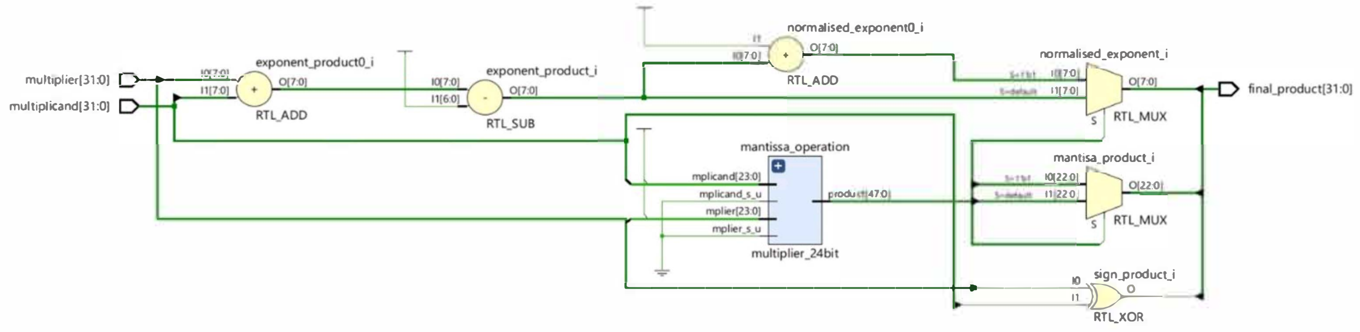
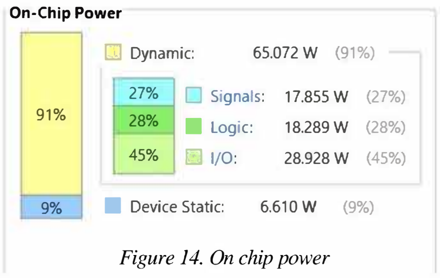


Figure 1.3 RTL schematic of single precision floating point multiplier


On-Chip Power

91%	Dynamic:	65.072 W	(91%)
27%	Signals:	17.855 W	(27%)
28%	Logic:	18.289 W	(28%)
45%	I/O:	28.928 W	(45%)
9%	Device Static:	6.610 W	(9%)

Figure 14. On chip power

IX. CONCLUSION

This paper presented the design and implementation of a high-efficiency 32-bit single-precision floating-point multiplier based on the IEEE 754 standard. By integrating the Booth recoding algorithm for mantissa multiplication, the architecture successfully reduced the total number of partial products, leading to a more streamlined addition stage. The performance evaluation confirms that the proposed design effectively optimizes hardware resource utilization while maintaining architectural integrity for floating-point arithmetic. The implemented unit provides a reliable solution for high-speed digital signal processing and embedded engineering applications where balanced power consumption and FPGA area efficiency are critical

REFERENCES.

[1] Saha, P., Banerjee, A., Bhattacharyya, P., and Dandapat, A. (2011, January). "High speed ASIC design of complex multiplier using vedicmathematics". In Students' Technology Symposium (TechSym), 2011 IEEE (pp. 237-241). IEEE.

[2] Ankush Nikam, Swati Salunke, Sweta Bhurse. "Design and Implementation of 32bit Complex Multiplier using Vedic Algorithm" IJERT ,2015 March,Vol 4

[3] M. Morris Mano, "Digital Design",5th edition, Prentice Hall,2002.

[4] Piyush Pati. "FPGA Implementation of Single Cycle Signed Multiplier using Booth Recoding Algorithm" IJERT ,2023 March,Vol 12, issue 03.

[5] Prity Mishra," FPGA Realization of Single-Cycle, 32-Bit Booth Recoding Signed Multiplier Enhanced by High-Speed Compressors" IJERT ,2024 March,Vol 13, issue 03.

[6] IEEE. (2019). *IEEE Standard for Floating-Point Arithmetic* (IEEE std 754-2019)IEEE.