

Design of Low Power and High Speed Multiplier using 8-Bit Additive Multiply Module(AMM)

**D. SWATHI¹, N. MADHU², N. SOBHA RANI³ P. SATYA SRI⁴,
K. Mahesh Babu⁵**

^{1,2,3,4}B.Tech Scholars, Department of ECE,

⁵Assistant Professor, Department of ECE, Aditya College of Engineering and Technology, Surampalem,
Kakinada, AP, India.

ABSTRACT

Multiplier is the most essential and primitive element of multiply and accumulate (MAC) unit, and is typically found in many digital signal processing (DSP) applications. Conventionally the multiplication operation is carried out through repeated addition, hence multipliers use extensively the full adders (FA's) for the addition process. The energy efficiency of the multiplier is determined in terms of power delay consumed per bit operation. Thus, FA plays a vital role in determining the overall efficiency of the multiplier. In this paper, a study, design, and simulation of an 8-bit additive multiply module (AMM) using different FA circuit architectures is presented. Further, the designed AMM is compared against the traditional Braun multipliers in terms of design metrics. To perform comparison of all the multipliers, they are described using RTL codes, simulated and verified using Xilinx tool.

KEYWORDS: Full Adder, Multiply and Accumulate, Braun Multiplier, 8 Bit Additive Multiply Module (AMM), Xilinx.

I. Introduction

The main objective of this project is to design and implement a low-power, high-speed multiplier using the 8-bit Additive Multiply Module (AMM) for Digital Signal Processing (DSP) applications. Multipliers play a critical role in multiply-and-accumulate (MAC) operations, which are widely used in DSP systems, microprocessors, and embedded devices. Traditional multipliers, such as Braun and Booth multipliers, consume more power and introduce higher computational delays due to their complex architecture. To address these limitations, the AMM-based multiplier is designed to optimize performance by dividing the multiplicand and multiplier into smaller parts, enabling parallel processing of partial products.

This approach reduces propagation delay and minimizes power consumption, making the system more efficient. The project aims to compare the performance of the AMM multiplier with traditional multipliers by evaluating metrics such as speed, power efficiency, and resource utilization through HDL (Hardware Description Language) simulations using tool Xilinx ISE 14.7. By achieving faster computation and lower power consumption, the proposed AMM multiplier is intended to enhance the efficiency of DSP systems and other high-performance computing applications. The Additive Multiply Module (AMM) is introduced as an alternative to traditional multiplier architectures, offering improved performance by reducing the complexity of partial product generation and accumulation. By dividing the 8-bit multiplicand and multiplier into smaller segments and utilizing parallel processing, the AMM architecture minimizes delay and hardware complexity. This approach not only enhances computational efficiency but also reduces power consumption, making it ideal for resource-constrained environments such as embedded systems and IoT devices.

The motivation to develop the AMM-based multiplier is further driven by the need to improve the performance of Arithmetic Logic Units (ALUs) in processors and enhance the efficiency of high-performance computing applications. With the increasing prevalence of battery-powered devices, the importance of energy-efficient and high-speed arithmetic units has become more critical, motivating the adoption of innovative architectures like AMM to address these challenges effectively.

II. Related work

➤ M. C. Parameshwara [1] and M. Nagabushanam [5] proposed a 4 novel reversible FAs which are based on cutting-edge FA architecture design. The merits of proposed RFAs are compared to those of reported reversible FAs on the basis of QGMs. According to the result comparison, the proposed RFAs are efficient in terms of the number of constant inputs/garbage outputs and the quantum cost.

- P. Choppala et al.[2] proposed a multiplier design that is area efficient, low power, and Full Swing Hybrid Multipliers. This research paper focuses on Gate Diffusion Input based hybrid FA of a 1-bit with only 14 transistors, within the standard array and WT multipliers. This work implements a hybrid adder and Wallace Tree multipliers within the array; thus the names H-A and H-WT are kept. The use of WT raises power consumption and is compensated by the use of Gate Diffusion Input technology, which also decreases the area. But as a limitation that the GDI structures fail to achieve full swing, it can be overcome by the use of H-A & H-WT multipliers. Ultimately partial products are added by CSA. Tanner EDA tool was used to implement the proposed designs with 250 nm technology and power consumption for the designs is 583 μ W and 693 μ W.
- A. Jain et al. [3] implemented an 8 \times 8 Multiplier which is an integration of Vedic Mathematics and Booth WT Multiplier. It uses the Radix-4 Booth algorithm, Wallace Tree algorithm, and fusion of Modified Booth and Wallace algorithms. It also uses the Vedic multiplication algorithm and it combines the Vedic-Booth-Wallace algorithms and the new proposed multiplier are designed which results in a reduction in delay and area.
- M. Munawar et al. [4] implemented a modified version of the DT multiplier which uses a carry select adder and binary to an excess-1 converter, as a result, it increases its efficiency in terms of power and speed. The modified multiplier is Implemented and compared with existing multipliers in terms of delay, dynamic power, and PDP. The analysis depicts that the proposed multiplier outperforms other multipliers in terms of performance.
- Che-Wei Tung et al. [6] proposed pipeline MAC architecture with high speed and low power design. Where carry propagations of additions frequently result in higher power consumption and path delay in a conventional MAC. Additions to the partial product reduction process were incorporated to address these issues. The implemented MAC design does not perform the operation of addition and accumulation of most significant bits until the partial product (pp) reduction process of the following multiplication. To properly deal with an overflow in the PP reduction process, an adder with lower bit size is used to accumulate the carries. Experiment analysis reveals that the implemented MAC design can significantly decrease both the power consumption of the circuit and the area occupied by it under the same timing constraint as compared with existing works.

III. DESIGN & IMPLEMENTATION

DESIGN OF PROPOSED SYSTEM

AN ABSTRACT MODEL OF 4 \times 2 AMM

The multiplication of two eight numbers to get a final product $M=X \times Y$ is explained as follows. To multiply X with Y using the AMM, the multiplicand X is halved into two equal parts namely lower half bits (XL) and higher half bits (XH) respectively. The multiplier is also divided into four equal parts viz. Y_1, Y_2, Y_3 , and Y_4 , where the weight of the bits will increase from Y_1 to Y_4 . The multiplication of XL and XH with each part of Y is carried out using an independent additive multiply block as shown in Fig.5.1. The pictorial representation of the process of multiplying two numbers using AMM is illustrated with an example of 8 \times 8 multiplication in Fig.5.2, along with an example shown in Fig.5.3. In Fig.5.2 and Fig.5.3, the X and Y represent multiplicand and multiplier respectively.

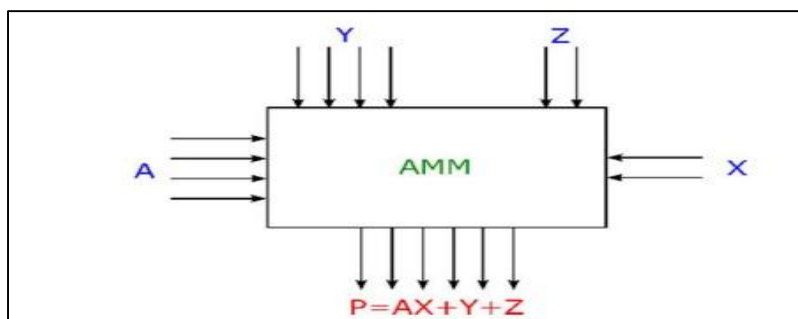


Figure : An abstract model of 4 \times 2 AMM

Beneath the abstract model, the functional behavior of the AMM is explicitly defined by the equation " $P = AX + Y + Z$." This equation reveals that the AMM performs a multiplication operation between the input " A " and some internal multiplier, implicitly represented by the term " X " being dependent on " A " and the internal logic of the module. Crucially, the AMM also incorporates an additive function, summing the inputs " Y " and " Z " to the product of " A " and the internal multiplier. The caption "An abstract model of 4 \times 2 AMM" suggests that the module is designed to handle a 4-bit input for one operand and potentially produces an intermediate result with a 2-bit width before the addition stage, although these bit-widths are not explicitly shown in the diagram.

Considering the accompanying text which describes an 8 \times 8 multiplication using AMMs, this 4 \times 2 AMM model represents a fundamental building block in that larger process. In the context of multiplying two 8-bit

numbers, where the multiplicand (X) is divided into a 4-bit lower half (XL) and a 4-bit higher half (XH), and the multiplier (Y) is divided into four 2-bit parts (Y1 to Y4), the input "A" to this AMM would correspond to either XL or XH. The internal multiplication within the AMM would then involve one of the 2-bit segments of the multiplier. The "additive" nature of the AMM, through the inputs "Y" and "Z," is crucial for accumulating partial products and intermediate sums generated by multiple AMM blocks within the 8x8 multiplication architecture, ultimately contributing to the final 16-bit product. The output "X" from this individual AMM would then be further processed and combined with the outputs of other AMMs to yield the complete multiplication result.

BLOCK DIAGRAM OF 8X8 AMM MODULE

An 8-bit AMM (Additive Multiply Module) multiplier was built using Verilog's 4x2 AMMs. The primary goal of this work is to optimize the power consumption of 8-bit multiplier. The block diagram shown in Figure 2 represents the proposed system architecture of the 8x8 Additive Multiply Module (AMM) multiplier. In this architecture, the 8-bit multiplicand (X) is divided into two equal parts: the lower half (XL) and the higher half (XH). Similarly, the 8-bit multiplier (Y) is split into four equal parts: Y1, Y2, Y3, and Y4, where the weight of the bits increases progressively from Y1 to Y4. Each part of X is multiplied with each part of Y using an independent 4x2 AMM module, which is responsible for generating partial products. As shown in the figure, these partial products are computed in parallel, significantly reducing computation time.

The partial products are then accumulated using Carry-Save Adders (CSA) and RippleCarry Adders (RCA), ensuring that the results from the intermediate modules are efficiently combined. The final output, represented by P0 to P15, is generated by adding the accumulated partial products from each stage. The modular and hierarchical design of the AMM architecture reduces hardware complexity and computational delay, making the system more efficient in terms of power consumption and speed. This approach is particularly beneficial for high-speed digital signal processing applications where low power and high efficiency are essential.

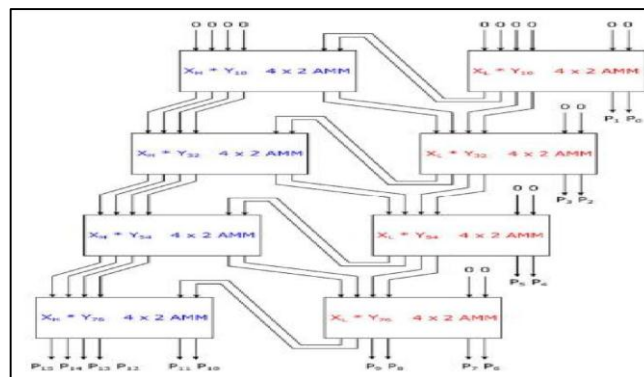


Figure: Block diagram of binary 8x8 AMM

The partial products are then accumulated using Carry-Save Adders (CSA) and RippleCarry Adders (RCA), ensuring that the results from the intermediate modules are efficiently combined. The final output, represented by P0 to P15, is generated by adding the accumulated partial products from each stage. The modular and hierarchical design of the AMM architecture reduces hardware complexity and computational delay, making the system more efficient in terms of power consumption and speed. This approach is particularly beneficial for high-speed digital signal processing applications where low power and high efficiency are essential.

8 X 8 MULTIPLICATION WITH AN EXAMPLE

All the generated intermediate PP are shifted and added as per their respective weights as shown in Fig.5.3. An example of $M=187 \times 235=43945$ using a binary 8x8 AMM is illustrated in Fig.5.4. The process of addition and pipelining of all intermediate partial products generated from each sub-module of AMM is pictorially represented in the block diagram shown in Fig.

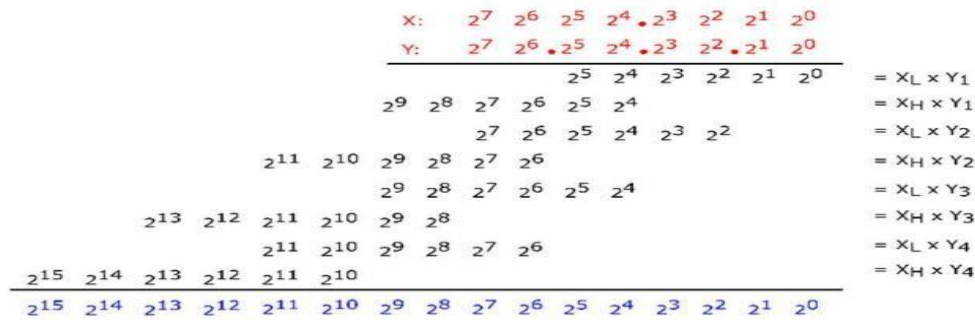


Figure: A pictorial representation of an 8 × 8 multiplication via 4 × 2 AMM

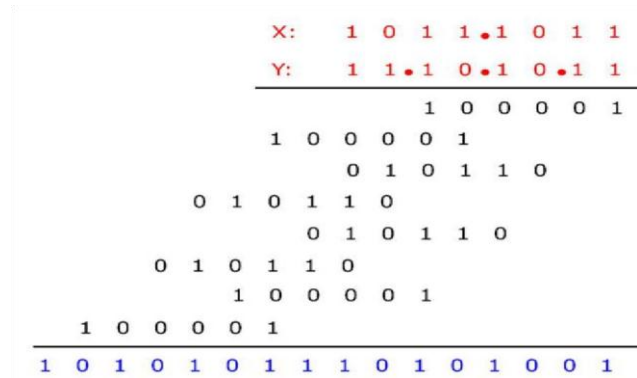


Figure: An example of 8x8 multiplication using AMM module

All the generated intermediate partial products (PP) are systematically shifted and summed according to their respective weights, as illustrated in Fig. 5.3. This step ensures that the contribution of each bit position is correctly aligned in the final summation process. An example of an 8x8 Additive Multiply Module (AMM) performing multiplication for $M = 187 \times 235 = 43945$ is depicted in Fig. 5.4. The multiplication is executed in binary representation, where each bit of the multiplier interacts with the multiplicand to produce partial products.

These partial products are then appropriately aligned based on the weight of the corresponding bit positions before being added together to obtain the final result.

Fig. 5.4 further elaborates on this process by representing the operands in both binary and exponential notation. The alignment of partial products follows the significance of each bit, ensuring that each multiplication contributes to the correct positional value in the summation process. The diagram effectively illustrates how the higher-order and lower-order bits interact, and how each sub-module within the AMM contributes to the computation.

IV. IMPLEMENTATION

MIGRATING PROJECTS FROM PREVIOUS ISE SOFTWARE RELEASES:

When you open a project file from a previous release, the ISE® software prompts you to migrate your project. If you click Backup and Migrate or Migrate Only, the software automatically converts your project file to the current release. If you click Cancel, the software does not convert your project and, instead, opens Project Navigator with no project loaded. **Note:** After you convert your project, you cannot open it in previous versions of the ISE software, such as the ISE 11 software. However, you can optionally create a backup of the original project as part of project migration, as described below.

To Migrate a Project

1. In the ISE 12 Project Navigator, select **File > Open Project**.
 2. In the Open Project dialog box, select the .xise file to migrate.
- Note** You may need to change the extension in the Files of type field to display .npl (ISE 5 and ISE 6 software) or .ise (ISE 7 through ISE 10 software) project files.
3. In the dialog box that appears, select **Backup and Migrate** or **Migrate Only**.
 4. The ISE software automatically converts your project to an ISE 12 project. **Note** If you chose to Backup and Migrate, a backup of the original project is created at project_name_ise12migration.zip.
 5. Implement the design using the new version of the software.
- Note** Implementation status is not maintained after migration.

PROPERTIES:

For information on properties that have changed in the ISE 12 software, see ISE 11 to ISE 12 Properties Conversion.

IP MODULES:

If your design includes IP modules that were created using CORE Generator™ software or Xilinx® Platform Studio (XPS) and you need to modify these modules, you may be required to update the core. However, if the core netlist is present and you do not need to modify the core, updates are not required and the existing netlist is used during implementation.

OBSOLETE SOURCE FILE TYPES:

The ISE 12 programming backings the greater part of the source sorts that were upheld in the ISE 11 programming. On the off chance that you are working with undertakings from past discharges, state graph source documents (.dia), ABEL source records (.abl), and test seat waveform source documents (.tbw) are no more upheld. For state outline and ABEL source records, the product discovers a related HDL document and adds it to the task, if conceivable. For test seat waveform documents, the product consequently changes over the TBW record to a HDL test seat and adds it to the venture. To change over a TBW record after task relocation, see Converting a TBW File to a HDL Test Bench.

USING ISE EXAMPLE PROJECTS:

To help familiarize you with the ISE® software and with FPGA and CPLD designs, a set of example designs is provided with Project Navigator. The examples show different design techniques and source types, such as VHDL, Verilog, schematic, or EDIF, and include different constraints and IP.

To Open an Example

1. Select **File > Open Example**.
2. In the Open Example dialog box, select the Sample Project Name.

Note To help you choose an example project, the Project Description field describes each project. In addition, you can scroll to the right to see additional fields, which provide details about the project.

3. In the Destination Directory field, enter a directory name or browse to the directory.
4. Click **OK**.

The example project is extracted to the directory you specified in the Destination Directory field and is automatically opened in Project Navigator. You can then run processes on the example project and save any changes.

Note If you modified an example project and want to overwrite it with the original example project, select **File > Open Example**, select the Sample Project Name, and specify the same Destination Directory you originally used. In the dialog box that appears, select **Overwrite the existing project** and click **OK**.

V. SIMULATION RESULTS

RESULTS OF EXISTING SYSTEM

SIMULATION RESULTS OF BRAUN MULTIPLIER

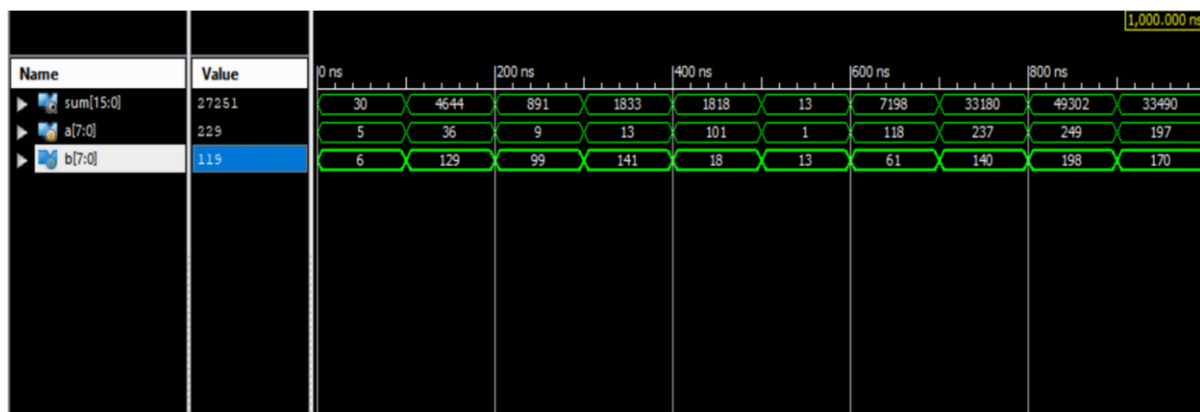


Figure: Simulation result of the 8 bit Braun multiplier

SUMMARY REPORT OF BRAUN MULTIPLIER

braun_mul Project Status (03/25/2025 - 21:45:28)			
Project File:	existing.xise	Parser Errors:	No Errors
Module Name:	braun_mul	Implementation State:	Synthesized
Target Device:	xc3s50a-5tq144	*Errors:	No Errors
Product Version:	ISE 14.7	*Warnings:	1 Warning (0 new)
Design Goal:	Balanced	*Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	*Timing Constraints:	
Environment:	System Settings	*Final Timing Score:	

Device Utilization Summary (estimated values)				[1]
Logic Utilization	Used	Available	Utilization	
Number of Slices	69	704	9%	
Number of 4 input LUTs	120	1408	8%	
Number of bonded IOBs	32	108	29%	

Figure: Summary report of the 8 bit Braun multiplier

SYNTHESIS REPORT OF BRAUN MULTIPLIER

LUT4:I0->O	2	0.561	0.488	row1/Cell16/adder/Mxor_sum_Result1 (w1<5>)
LUT4:I0->O	2	0.561	0.382	row2/Cell15/adder/c_out1 (row2/w2<5>)
LUT4:I3->O	2	0.561	0.488	row2/Cell16/adder/c_out1 (row2/w2<6>)
LUT4:I0->O	2	0.561	0.488	row2/Cell7/adder/Mxor_sum_Result1 (w2<6>)
LUT4:I0->O	2	0.561	0.488	row3/Cell16/adder/c_out1 (row3/w2<6>)
LUT4:I0->O	2	0.561	0.488	row3/Cell7/adder/Mxor_sum_Result1 (w3<6>)
LUT4:I0->O	2	0.561	0.488	row4/Cell16/adder/c_out1 (row4/w2<6>)
LUT4:I0->O	2	0.561	0.488	row4/Cell7/adder/Mxor_sum_Result1 (w4<6>)
LUT4:I0->O	2	0.561	0.488	row5/Cell16/adder/c_out1 (row5/w2<6>)
LUT4:I0->O	2	0.561	0.488	row5/Cell7/adder/Mxor_sum_Result1 (w5<6>)
LUT4:I0->O	2	0.561	0.488	row6/Cell16/adder/c_out1 (row6/w2<6>)
LUT4:I0->O	2	0.561	0.488	row6/Cell7/adder/Mxor_sum_Result1 (w6<6>)
LUT4:I0->O	2	0.561	0.488	row7/Cell16/adder/c_out1 (row7/w2<6>)
LUT4:I0->O	1	0.561	0.357	row7/Cell7/adder/Mxor_sum_Result1 (sum_14_OBUF)
OBUF:I->O		4.396		sum_14_OBUF (sum<14>)

Total		26.582ns	(16.440ns logic, 10.142ns route)	
			(61.8% logic, 38.2% route)	

Figure: Synthesis report of the 8 bit Braun multiplier

VI. RESULT OF PROPOSED SYSTEM

SIMULATION RESULTS OF PROPOSED MULTIPLIER WITH AMM




		1,000.000 ns											
Name	Value	0 ns		200 ns		400 ns		600 ns		800 ns			
▶  p[15:0]	27251	250	4644	891	1833	1818	13	7198	33180	49302	33490		
▶  a[7:0]	229	10	36	9	13	101	1	118	237	249	197		
▶  b[7:0]	119	25	129	99	141	18	13	61	140	198	170		

Figure: Simulation result of the 8 bit proposed multiplier

SUMMARY REPORT OF 8 BIT MULTIPLIER WITH AMM MODULE

braun_mul_amm Project Status (03/26/2025 - 10:50:37)			
Project File:	proposed.xise	Parser Errors:	No Errors
Module Name:	braun_mul_amm	Implementation State:	Synthesized
Target Device:	xc7vx330t-3ffg1157	• Errors:	No Errors
Product Version:	ISE 14.7	• Warnings:	No Warnings
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:	System Settings	• Final Timing Score:	

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	128	204000	0%
Number of fully used LUT-FF pairs	0	128	0%
Number of bonded IOBs	32	600	5%

Figure: Summary report of the 8 bit proposed multiplier

SYNTHESIS REPORT OF THE 8 BIT PROPOSED MULTIPLIER

LUT4:I2->O	3	0.043	0.560	a4/Maddsub_j_Madd_lut<2>1 (a4/Maddsub_j_Madd_lut<2>)
LUT6:I0->O	3	0.043	0.552	a4/Maddsub_j_Madd_xor<3>11 (n0008<3>)
LUT5:I0->O	4	0.043	0.470	a6/Madd_n0015[4:0]_cy<1>11 (a6/Madd_n0015[4:0]_cy<1>)
LUT5:I1->O	1	0.043	0.542	a6/Maddsub_j_Madd_lut<3>1 (a6/Maddsub_j_Madd_lut<3>)
LUT6:I1->O	3	0.043	0.552	a6/Maddsub_j_Madd_xor<3>11 (n0009<3>)
LUT5:I0->O	4	0.043	0.442	a8/Madd_n0015[4:0]_cy<1>11 (a8/Madd_n0015[4:0]_cy<1>)
LUT3:I0->O	2	0.043	0.546	a8/Madd_n0015[4:0]_cy<2>11 (a8/Madd_n0015[4:0]_cy<2>)
LUT6:I1->O	2	0.043	0.347	a8/Maddsub_j_Madd_cy<3>1 (a8/Maddsub_j_Madd_cy<3>)
LUT5:I3->O	1	0.043	0.279	a8/Maddsub_j_Madd_xor<5>11 (p_15_OBUF)
OBUF:I->O		0.000		p_15_OBUF (p<15>)

Total		7.960ns	(1.108ns logic, 6.852ns route)	
			(13.9% logic, 86.1% route)	
=====				

Figure: Synthesis report of the 8 bit proposed multiplier

Based on the provided simulation and synthesis reports, a comparison between the existing Braun multiplier and the proposed multiplier utilizing the Additive Multiply Module (AMM) reveals notable differences in performance and resource utilization. The synthesis report for the existing Braun multiplier, implemented on a Xilinx device, indicates a critical path delay of 26.582 ns, with a significant portion (61.8%) attributed to logic and the remainder (38.2%) to routing. This design utilized 9% of the available slices and 8% of the 4 input LUTs. In contrast, the proposed multiplier, synthesized for a Xilinx device, demonstrates a considerably improved critical path delay of 7.960 ns and logic

VII. CONCLUSION & FUTURE SCOPE

CONCLUSION

The 8×8-bit WTM, DTM, and AMM multiplier schematics are designed using Verilog RTL codes based on generic FA architecture [1,5] and tested for various input data combinations and the functionality is verified, the proposed designs are also synthesized with Xilinx. The proposed multipliers are then implemented with different FA architectures viz. XOR-Mux, XNOR-Mux, and XOR/XNOR-Mux [5] to further study the power behavior of the multipliers under different FA architectures.

The results obtained from the simulations demonstrate that the proposed architectures exhibit significant improvements in terms of area, delay, and power consumption when compared to traditional multiplier designs. Among the various FA architectures analyzed, the XOR/XNOR-Mux implementation showed a balanced trade-off between speed and power efficiency, making it a suitable choice for low-power applications.

Additionally, the AMM multiplier, due to its unique Additive Multiply Module approach, offers enhanced performance, particularly in scenarios involving large data sets. The synthesized results also confirm that the proposed designs are scalable and can be efficiently extended to higher bit-widths without substantial degradation in performance.

8.2 FUTURE SCOPE

The proposed multiplier architectures, including WTM, DTM, and AMM designs implemented with different FA architectures, provide a strong foundation for further research and optimization. These designs can be extended to support higher bit-width multiplications to evaluate their scalability and performance under larger data sets. The potential for power and delay optimization can be explored by implementing advanced techniques such as clockgating and operand decomposition to further reduce switching activity and enhance energy efficiency.

In addition, incorporating fault-tolerant techniques can improve the reliability of these architectures, making them suitable for mission-critical applications. Error detection and correction mechanisms can be integrated to ensure data integrity in noise-prone environments. The proposed designs can also be implemented and tested on FPGA platforms to validate their performance in real-time scenarios, allowing for practical insights and refinements.

Furthermore, the application of these optimized multipliers in fields such as artificial intelligence, machine learning, and digital signal processing can be explored, where highspeed and low-power arithmetic operations are essential. Future research can also investigate the integration of these designs with emerging technologies such as approximate computing and neuromorphic systems, paving the way for next-generation computing applications. By addressing these aspects, the proposed multiplier architectures can be further refined and adapted for use in a variety of high-performance and low-power applications, ensuring their relevance in modern digital systems.

REFERENCES

- [1] Parameshwara, M.C., "Approximate full adders for energy efficient image processing applications" *Journal of Circuits, Systems and Computers*, vol. 30, issue 13, pp.1-17, Oct. 2021.
- [2] P. Choppala, V. Gullipalli, M. Gudivada and B. Kandregula, "Design of Area Efficient, Low Power, High Speed and Full Swing Hybrid Multipliers," 2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS), 2021, pp. 929-934.
- [3] A. Jain, S. Bansal, S. Khan, S. Akhter and S. Chaturvedi, "Implementation of an Efficient N×N Multiplier Based on Vedic Mathematics and Booth-Wallace Tree Multiplier" 2019 International Conference on Power Electronics, Control and Automation (ICPECA), 2019, pp. 1-5.
- [4] M. Munawar et al., "Low Power and High Speed Dadda Multiplier using Carry Select Adder with Binary to Excess-1 Converter," 2020 International Conference on Emerging Trends in Smart Technologies (ICETST), pp. 1-4, 2020.
- [5] Parameshwara, M.C., Nagabushanam, M. "Novel low quantum cost reversible logic based full adders for DSP applications." *International Journal of Information Technology*. 2021, vol. 13, pp.1755–1761.
- [6] C.W.Tung and S.H. Huang, "A High-Performance MultiplyAccumulate Unit by Integrating Additions and Accumulations into Partial Product Reduction Process," in *IEEE Access*, vol. 8, pp. 87367-87377, 2020.
- [7] K. B. Jaiswal, Nithish Kumar V, P. Seshadri and Lakshminarayanan G, "Low power Wallace tree multiplier using modified full adder," 2015 3rd International Conference on Signal Processing, Communication and Networking (ICSCN), pp. 1-4, 2015.
- [8] S. Venkatachalam and S. Ko, "Design of Power and Area Efficient Approximate Multipliers," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1782-1786, May 2017.
- [9] P. Yadav, A. Pandey, M. R. K., R. P. K.J., V. M.H. and N. K. Y.B., "Low Power Approximate Multipliers with Truncated Carry Propagation for LSBs," 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 500-503, 2018.
- [10] Xiong, X., Lin, M. "Low Power 8-Bit Baugh–Wooley Multiplier Based on Wallace Tree Architecture," In: Sobh, T., Elleithy, K. (eds) *Emerging Trends in Computing, Informatics, Systems Sciences, and Engineering. Lecture Notes in Electrical Engineering*, vol. 151. Springer, New York, NY, 2013.