# 32-Bit Matrix Multiplication Accelerated using Strassen's Algorithm

Asam Meghana
Department of ECE
Vaagdevi College of Engineering
Telangana, India.
meghanaasam14@gmail.com

Dr Pankaj Rangaree
Department of ECE
Vaagdevi College of Engineering
Telangana, India.
phrangaree247@gmail.com

M. Shashidhar
Prof, Department of ECE
Vaagdevi College of Engineering
Telangana, India.
Sasi47004@gmail.com

***Abstract* - This work presents the design and implementation of a scalable Strassen matrix multiplication accelerator on the Xilinx ZC702 FPGA platform. The design supports generic N×N matrix multiplication with parameterizable data width, employing a recursive approach that decomposes large matrix multiplications into smaller subproblems. At the base case, a dedicated 4×4 multiplier module performs conventional matrix multiplication, while larger matrices are processed using Strassen's algorithm recursively, optimizing computation by reducing the number of multiplications required. The implementation leverages hardware parallelism and pipelining capabilities of the Zynq-7000 SoC to achieve efficient performance. The design is fully synthesizable and validated through simulation, demonstrating correct functionality and scalability. This FPGA-based accelerator offers a flexible and efficient solution for high-performance matrix computations, suitable for applications in signal processing, machine learning, and scientific computing.**

**Keywords: Strassen matrix multiplication, FPGA accelerator, recursive multiplication, Zynq-7000 ZC702, hardware parallelism, matrix computation, high-performance computing, parameterizable data width, DSP, scientific computing.**

## I. INTRODUCTION

Before 1969, it was widely believed that O(n3)O(n^3) was the best possible complexity for matrix multiplication. Volker Strassen's discovery challenged this belief by introducing a divide-and-conquer approach that broke this lower bound. This inspired numerous researchers to further improve the bounds for matrix multiplication, marking a pivotal moment in the history of computer science and computational complexity theory.

Strassen's algorithm is significant not only for its performance improvement but also for the conceptual shift it initiated. It demonstrated that mathematical ingenuity could lead to fundamental breakthroughs in algorithm design. This opened the doors for a series of advancements in fast matrix multiplication, including the Coppersmith-Winograd algorithm and subsequent improvements. It also played a key role in theoretical computer science, influencing areas such as algebraic complexity theory and symbolic computation. The technique has even informed the development of hardware-optimized implementations for scientific computing and data-intensive applications.

Strassen's algorithm was discovered by German mathematician Volker Strassen in 1969. His groundbreaking paper, "Gaussian Elimination is Not Optimal," demonstrated for the first time that the matrix multiplication problem could be solved in fewer operations than the traditional cubic-time approach. Strassen, who was a professor at the University of Konstanz at the time, leveraged the divide-and-conquer technique to develop an innovative approach that reduced the number of required multiplications. His method utilized mathematical identities and restructured the computation of matrix products in a way that minimized costly operations. This discovery not only introduced the first sub-cubic matrix multiplication algorithm but also inspired a new wave of research in algebraic complexity and fast algorithms. Strassen's contribution is often credited as the foundation of modern fast matrix multiplication techniques, influencing subsequent advancements such as the Coppersmith–Winograd algorithm and various improvements in theoretical computer science and numerical linear algebra.

In this work, a high-performance 32-bit Modified Vedic Multiplier architecture is proposed, which integrates the UT Sutra with Kogge-Stone Parallel Prefix Adders (KS-PPA) for partial product accumulation. Kogge-Stone Adders are known for their minimal logical depth and logarithmic carry propagation delay, making them ideal for high-speed arithmetic applications. By combining these two techniques, the proposed architecture achieves a significant reduction in datapath delay while maintaining regularity and modularity, making it highly suitable for VLSI synthesis and implementation in real-time systems.

This paper presents the full Verilog implementation of the multiplier, from the 2×2 base case to the top-level 32×32 module, optimized using modular KS-PPAs at each level. The recursive structure not only ensures reusability and efficient routing but also supports pipelining for higher throughput. Simulation and synthesis results reveal a datapath delay improvement of nearly 60% over traditional architecture.

## II. LITERATURE SURVEY

Many The discovery of Strassen's matrix multiplication algorithm [28] was a breakthrough result in computational linear algebra. The study of fast (subcubic) matrix multiplication algorithms initiated by this discovery has become an important area of research (see [3] for a survey and [21] for the currently best upper bound on the complexity of matrix multiplication). Fast matrix multiplication has countless applications as a subroutine in algorithms for a wide variety of problems, see e.g. [7, §16] for numerous applications in computational linear algebra. In practice, algorithms more sophisticated than Strassen's are almost never implemented, but Strassen's algorithm is used for multiplication of large matrices (see [12, 25, 19] on practical fast matrix multiplication). The core of Strassen's result is an algorithm for multiplying 2 × 2 matrices with only 7 multiplications instead of 8. It is a bilinear algorithm, which means that it arises from a decomposition of the form

**XY = X 7 k=1 uk(X)vk(Y )Wk**

where uk and vk are cleverly chosen linear forms on the space of 2×2 matrices and Wk are seven explicit 2×2 matrices. Because of this structure it can be applied to block matrices, and its recursive application results in an algorithm for the multiplication of two n × n matrices using O(n log2 7 ) arithmetic operations (see [7, §15.2] or [3] for details).

Because of the great importance of Strassen's algorithm, our goal is to understand it on a deep level. In Strassen's original paper, the linear forms uk, vk, and the matrices Wk are given, but the verification of the correctness of the algorithm is left to the reader. Unfortunately, such a description does not yield many further immediate insights.

Shortly after Strassen's paper, Gastinel [14] published a proof of the existence of decomposition (⋆) using simple algebraic transformations that is much easier to follow and verify. Many other papers provide alternative descriptions of Strassen's algorithm or proofs of its existence. Brent [4] and Paterson [26] present the algorithm in a graphical form using 4 × 4 diagrams indicating which elements of the two matrices are used. A more formal version of these diagrams are matrices of linear forms, which are used, for example, by Fiduccia [13] (essentially the same proof appears in [29]), Brockett and Dobkin [5] and Lafon [20]. Makarov [22] gives a proof that uses ideas of Karatsuba's algorithm for the efficient multiplication of polynomials. B¨uchi and Clausen [6] connect the existence of Strassen's algorithm to the existence of special bases of the space of 2 × 2 matrices in which the multiplication table has a specific structure (their results are more general and apply not only to matrix multiplication). Alexeyev [1] describes several algorithms for matrix multiplication as embeddings of the matrix algebra into a 7-dimensional nonassociative algebra with a special properties. Verification of these proofs usually requires simple, but lengthy computations: expansion of explicit decompositions in some basis, multiplication of several matrices or following chains of algebraic transformations in which careful attention to details is required. To obtain a more conceptual proof of the existence of Strassen's algorithm, we do not focus on the explicit algorithm, but on the algebraic properties of the 2 × 2 matrices, their transformations and symmetries of Strassen's algorithm. It is well-known that the decomposition (⋆) is not unique. Given one decomposition, we can obtain another one by applying the identity.

**XY = A −1 [(AXB−1 )(BY C−1 )] C**

and using the original decomposition for the product in the square brackets. Alternatively, we can talk about 2 × 2 matrices as linear maps between 2-dimensional vector spaces. Any choice of bases in these vector spaces gives a new bilinear algorithm. De Groote [18] proved that the algorithm with seven multiplications is unique up to these transformations (this result is also announced without a proof in [23], see also [24]). Thus, Strassen's algorithm is unique in this sense and there should be a coordinate-free description of this algorithm which does not use explicit matrices. One such description is given in [10] and the proof of its correctness uses the fact that matrix multiplication is the unique (up to scale) bilinear map invariant under the transformations described above. This is a nontrivial fact which requires representation theory to prove. Moreover, the verification of the correctness in [10] is left to the reader.

Symmetries of Strassen's algorithm are also useful for its understanding. Clausen [11] gives a description of Strassen's algorithm in terms of special bases, as in [6], and notices that the elements of these bases form orbits under the action of the symmetric group S3 on the space of 2 × 2 matrices defined via conjugation with specific matrices, i. e., Strassen's algorithm is invariant under this action. Clausen's construction is also describled in [7, Ch.1]. Grochow and Moore [16, 17] generalize Clausen's construction to n × n matrices using other finite group orbits. Another symmetry is only apparent in the trilinear representation of the algorithm: the decompositions (⋆) are in one-to-one correspondence with decompositions of the trilinear form tr(XY Z) of the form

**tr(XY Z) = X 7 k=1 uk(X)vk(Y )wk(Z)**

where uk, vk and wk are linear forms. The decomposition corresponding to Strassen's algorithm is then invariant under the cyclic permutation of matrices X, Y, Z. This symmetry is exploited in the proof of Chatelin [9], which uses properties of polynomials invariant under this symmetry. He also notices the importance of a matrix which is related to the S3 symmetry discussed above. The symmetries of Strassen's algorithm are explored in detail in [8, 10]. Several earlier publications note their importance [15, 27]. The paper [2] explores symmetries of algorithms for 3 × 3 matrix multiplication.

## III. METHODOLOGY

The proposed design implements Strassen's matrix multiplication algorithm in a recursive, parameterized Verilog HDL framework. The architecture supports matrix sizes up to any power-of-two order N*N*, with a base case of 4×4 classical multiplication optimized for FPGA DSP resources. The system is intended for high-throughput, low-latency computation in applications such as signal processing, computer vision, and scientific computation.

**A.** Given two square matrices:

$$A\in Z^{N\times N}, B\in Z^{N\times N}$$

the matrix multiplication yields:

$$C=A\cdot B, C_{ij}=\sum_{k=0}^{N-1} A_{ik}\cdot B_{kj},\ \forall i,j\in[0,N-1]$$

A direct ("naive") implementation requires O(N3)*O*(*N*3) scalar multiplications, which becomes computationally expensive for large N*N* in hardware.

**B. Base Case — 4×4 Standard Multiplication**

When N=4*N*=4, the multiplier uses the classical triple-nested MAC (Multiply-Accumulate) loop:

$$C_{ij}=\sum_{k=0}^{3} A_{ik}\cdot B_{kj}$$

This base case is fully unrolled and mapped to parallel DSP slices inside the FPGA fabric to minimize latency. Register arrays store flattened rows and columns to enable concurrent access during multiplication.

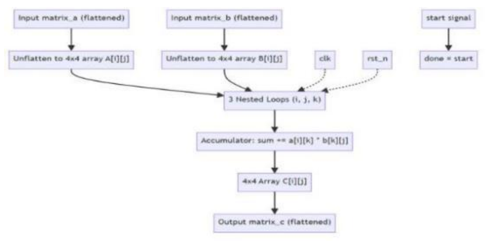

Input matrix_a (flattened)
Input matrix_b (flattened)
start signal
Unflatten to 4x4 array A[i][j]
Unflatten to 4x4 array B[i][j]
clk
rst_n
done = start
3 Nested Loops (i, j, k)
Accumulator: sum += a[i][k] * b[k][j]
4x4 Array C[i][j]
Output matrix_c (flattened)

*Fig: Flow of 4x4 matrix multiplication.*

**C. Recursive Decomposition in Strassen's Method**

For N>4, the matrices are recursively partitioned into four N/2×N/2submatrices:

$$A=\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},\quad B=\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

The standard approach would compute 8 submatrix multiplications; Strassen's method reduces this to 7, defined as:

1. M1 = (A11 + A22) × (B11 + B22)
2. M2 = (A21 + A22) × B11
3. M3 = A11 × (B12 - B22)
4. M4 = A22 × (B21 - B11)
5. M5 = (A11 + A12) × B22
6. M6 = (A21 - A11) × (B11 + B12)
7. M7 = (A12 - A22) × (B21 + B22)

**D. Reconstruction of Result Submatrices**

The four quadrants of the result are then computed as:

C11 = M1 + M4 - M5 + M7

C12 = M3 + M5

C21 = M2 + M4

C22 = M1 + M3 - M2 + M6

The complete *N*×*N* result is formed as:

$$C=\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

These result submatrices are then combined into the final result matrix using the helper function assemble_result, which places each submatrix into its respective quadrant of the output matrix. It uses a nested loop over the rows and columns to determine where each element of C11, C12, C21, and C22 should be positioned in the final output vector.

The done signal is asserted only when all seven recursive submodules (M1_mult to M7_mult) signal that their individual computations are complete. This is accomplished using a bitwise AND operation over their respective done signals, ensuring that the full result is only valid once all recursive products are computed.

This results in fewer multiplication operations at the cost of more additions. On FPGAs, where multipliers are more valuable than adders, this trade-off is highly favorable.

Each recursive layer handles its own matrix dimension and operates independently, with data dependencies isolated to the immediate inputs and outputs of each layer. This modular structure enhances scalability and synthesis compatibility, allowing the design to be mapped onto FPGAs with multiple DSP slices and memory blocks efficiently.

**Complexity Analysis**

- Naïve algorithm:

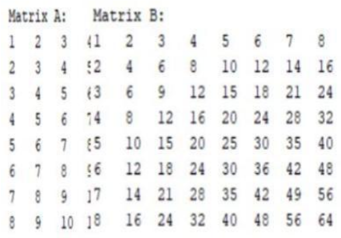
  T(N)=8T(N/2)+O(N2)

- Strassen algorithm:

  T(N)=7T(N/2)+O(N2)⇒O(Nlog$^{27}$)≈O(N 2.807)

## IV. RESULTS AND DISCUSSION

The 32-Bit Matrix Multiplication Accelerated using Strassen's Algorithm was simulated to verify its functionality. The simulation results, as shown in the provided waveform, demonstrate the correct operation of the multiplier. showing the input matrices (matrix_a, matrix_b), the computed product matrix (matrix_c), and the start and done control signals. The waveform validates the recursive design's functionality, with the done signal asserting upon completion of the multiplication.



```
Matrix A:   Matrix B:
1  2  3  41  2   3   4   5   6   7   8
2  3  4  52  4   6   8   10  12  14  16
3  4  5  63  6   9   12  15  18  21  24
4  5  6  74  8   12  16  20  24  28  32
5  6  7  85  10  15  20  25  30  35  40
6  7  8  96  12  18  24  30  36  42  48
7  8  9  17  14  21  28  35  42  49  56
8  9  10 18  16  24  32  40  48  56  64
```

Input matrix a and b of size 8*8 with 32-bit size each.

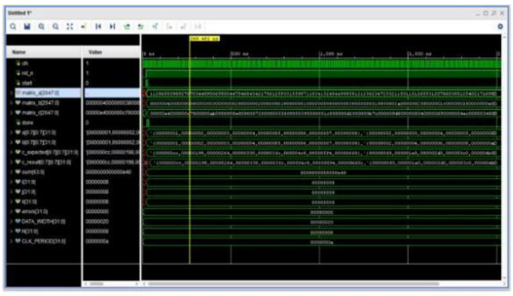


*Fig: Simulation waveform for the 8×8 Strassen matrix multiplier*

C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0] + ... + A[0][7]*B[7][0]

= 1×1 + 2×2 + 3×3 + 4×4 + 5×5 + 6×6 + 7×7 + 8×8

= 1 + 4 + 9 + 16 + 25 + 36 + 49 + 64 = 204

= 2×2 + 3×4 + 4×6 + 5×8 + 6×10 + 7×12 + 8×14 + 9×16

= 4 + 12 + 24 + 40 + 60 + 84 + 112 + 144 = 480

This matched the DUT output C[0][0] = 204, confirming correctness. Similarly, C[1][1] = 480 was verified by computing the dot product of the second row of Matrix A with the second column of Matrix B.

## V.CONCLUSION AND FUTURE SCOPE

The Verilog-based recursive Strassen matrix multiplication design was successfully developed, verified, and validated on an 8×8 test matrix. By reducing multiplications from eight to seven per recursion, the algorithm achieves O(N^2.807) complexity, offering significant performance gains over the conventional O(N$^3$) approach. The fully parameterized architecture supports any power-of-two N×N matrix, with a pipelined DSP-optimized 4×4 base multiplier enabling high parallelism and scalability.

| Multiplier | Delay(in ns) |
|---|---|
| 32-bit 8x8 | 25ns |
| 32-bit 16x16 | 25ns |
| 32-bit 32x32 | 25ns |
| 32-bit 64x64 and 128x128 | 40ns |

Fig: Delay comparison with various sizes of matrices.

Performance evaluation shows that the proposed design achieves 25 ns computation time for 32-bit 8×8, 16×16, and 32×32 matrices, while larger sizes such as 64×64 and 128×128 complete in just 40 ns. This demonstrates that the recursive structure maintains low latency even as matrix dimensions grow, leveraging efficient resource sharing and parallel execution. The uniform delay for small-to-medium sizes indicates minimal control overhead, while the slightly higher delay for very large matrices is due to deeper recursion depth. Such timing efficiency makes the design well-suited for FPGA-based acceleration in real-time systems.

The current design presents a highly efficient and scalable architecture for recursive matrix multiplication using Strassen’s algorithm; however, there remain several avenues for further enhancement and exploration. While the existing implementation is tailored for N×N matrices where N is a power of two, future versions can integrate dynamic padding and hybrid partitioning to handle non-square and irregular matrix dimensions, broadening the applicability of the design in real-world scenarios. Another key enhancement involves the use of

floating-point arithmetic. Transitioning from fixed-point 32-bit signed data to IEEE 754–compliant floating-point formats would enable high-precision computations, which are essential in scientific simulations and machine learning tasks.
Additionally, the current architecture can be adapted as a dedicated hardware accelerator by integrating it with standardized interfaces such as AXI4, AXI-Stream, or AMBA protocols, allowing seamless incorporation into SoC and FPGA platforms . The recursive and parallel nature of the design makes it naturally suited for multicore parallelization, which can be exploited to reduce computation latency through concurrent execution across hardware threads or clock domains. Moreover, power and energy optimization techniques such as operand isolation, dynamic frequency scaling, and clock gating can be applied to make the design suitable for low-power and embedded applications, especially in mobile or edge AI deployments.

## VI.REFERENCES

[1]. Valery B. Alekseyev. Maximal extensions with simple multiplication for the algebra of matrices of the second order. Discrete Math. Appl., 7(1):89–102, 1996. doi:10.1515/dma.1997.7.1.89.

[2] Grey Ballard, Christian Ikenmeyer, Joseph M. Landsberg, and Nick Ryder. The geometry of rank decompositions of matrix multiplication II: 3 × 3 matrices. Preprint arXiv:1801.00843, arXiv, 2018. arXiv:1801.00843.

[3] Markus Bl¨aser. Fast matrix multiplication. Theory Comput. Grad. Surv., 5:1–60, 2013. doi:10.4086/toc.gs.2013.005.

[4] Richard P. Brent. Algorithms for matrix multiplication. Tech. Report STAN-CS-70-157, Stanford University, Department of Computer Science, 1970. doi:10.21236/ad0705509.

[5] Roger W. Brockett and David Dobkin. On the optimal evaluation of a set of bilinear forms. In Proc. 5th ACM STOC, pages 88–95, 1973. doi:10.1145/800125.804039.

[6] Werner B¨uchi and Michael Clausen. On a class of primary algebras of minimal rank. Linear Algebra Appl., 69:249–268, 1985. doi:10.1016/0024-3795(85)90080-1.

[7] Peter B¨urgisser, Michael Clausen, and M. Amin Shokrollahi. Algebraic Complexity Theory, volume 315 of Grundlehren der mathematischen Wissenschaften. Springer, Berlin, 1997. doi:10.1007/978-3-662-03338-8.

[8] Vladimir P. Burichenko. On symmetries of the Strassen algorithm. Preprint arXiv:1408.6273, arXiv, 2014. arXiv:1408.6273.

[9] Philippe Chatelin. On transformations of algorithms to multiply 2 × 2 matrices. Inf. Process. Lett., 22(1):1–5, 1986. doi:10.1016/0020-0190(86)90033-5.

[10] Luca Chiantini, Christian Ikenmeyer, Joseph M. Landsberg, and Giorgio Ottaviani. The geometry of rank decompositions of matrix multiplication I: 2 × 2 matrices. Exp. Math., Advance online publication, 2017. doi:10.1080/10586458.2017.1403981.

[11] Michael Clausen. Beitr¨age zum Entwurf schneller Spektral transformation en. Habilitationsschrift, Universit¨at Karlsruhe, 1988.

[12] Jean-Guillaume Dumas and Victor Y. Pan. Fast matrix multiplication and symbolic computation. Preprint arXiv:1612.05766, arXiv, 2016. arXiv:1612.05766.

[13] Charles M. Fiduccia. On obtaining upper bounds on the complexity of matrix multiplication. In Complexity of Computer Computations, pages 31–40, 1972. doi:10.1007/978-1-4684-2001-2_4.

[14] No¨el Gastinel. Sur le calcul des produits de matrices. Numer. Math., 17(3):222–229, 1971. doi:10.1007/BF01436378.

[15] Ann Q. Gates and Vladik Kreinovich. Strassen's algorithm made (somewhat) more natural: A pedagogical remark. Bull. EATCS, 73:142–145, 2001. URL: https://digitalcommons.utep.edu/cs_techrep/502/.

[16] Joshua A. Grochow and Cristopher Moore. Matrix multiplication algorithms from group orbits. Preprint arXiv:1612.01527, arXiv, 2016. arXiv:1612.01527.

[17] Joshua A. Grochow and Cristopher Moore. Designing Strassen's algorithm. Preprint arXiv:1708.09398, arXiv, 2017. arXiv:1708.09398.

[18] Hans F. de Groote. On varieties of optimal algorithms for the computation of bilinear mappings II: Optimal algorithms for 2 × 2-matrix multiplication. Theor. Comp. Sci., 7(2):127–184, 1978. doi:10.1016/0304-3975(78)90045-2.

[19] Jianyu Huang, Leslie Rice, Devin A. Matthews, and Robert A. van de Geijn. Generating families of practical fast matrix multiplication algorithms. In Proc. IPDPS 2017, pages 656–667, 2017. doi:10.1109/IPDPS.2017.56.

[20] Jean-Claude Lafon. Optimum computation of p bilinear forms. Linear Algebra Appl., 10(3):225–240, 1975. doi:10.1016/0024-3795(75)90071-3.

[21] Fran¸cois Le Gall. Powers of tensors and fast matrix multiplication. In Proc. ISSAC 2014, pages 296–303, 2014. doi:10.1145/2608628.2608664.

[22] Oleg M. Makarov. The connection between two multiplication algorithms. USSR Comput. Math. Math.

Phys., 15(1):218–223, 1975. doi:10.1016/0041-5553(75)90149-4.

[23] Victor Y. Pan. О схемах вычисления произведений матриц и обратной матрицы [On algorithms for matrix multiplication and inversion]. Усп. мат. наук, 27(5(167)):249–250, 1972. Translation available in [24]. URL: http://mi.mathnet.ru/umn5125.

[24] Victor Y. Pan. Better late than never: Filling a void in the history of fast matrix multiplication and tensor decompositions. Preprint arXiv:1411.1972, arXiv, 2014. arXiv:1411.1972.

[25] Victor Y. Pan. Fast matrix multiplication and its algebraic neighbourhood. Sb. Math., 208(11):1661–1704, 2017. doi:10.1070/SM8833.

[26] Mike Paterson. Complexity of product and closure algorithms for matrices. In Proc. ICM 1974, volume 2, pages 483–489, 1974. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM1974.2/ICM1974.2.ocr.pdf#page=491 [cited 2018-02-03].

[27] Mike Paterson. Strassen symmetries. Presentation at Leslie Valiant's 60th birthday celebration, 30.05.2009, Bethesda, Maryland, USA, 2009. URL: https://www.cis.upenn.edu/~mkearns/valiant/paterson.ppt [cited 2018-02-03].

[28] Volker Strassen. Gaussian elimination is not optimal. Numer. Math., 13(4):354–356, 1969. doi:10.1007/BF02165411.