2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION JAPAN

# Impact of a 64-bit Vedic Multiplier on Processor, Multi-Core, and DSP Performance

Lakshya Miglani, Engineer – VLSI Design, Silicon Interfaces, Mumbai, India
(lakshya@siliconinterfaces.com)

Gopi Srinivas Deepala, Engineer – VLSI Design, Silicon Interfaces, Mumbai, India
(gopi@siliconinterfaces.com)

***Abstract*— The Urdhva Tiryagbhyam sutra-based algorithm revolutionizes multiplication by significantly reducing computational steps and enhancing Power, Timing, and Area (PTA) metrics. This paper introduces a 64-bit multiplier design that leverages the optimized Urdhva Tiryagbhyam algorithm for binary numbers, demonstrating improved efficiency and performance over traditional methods. Essential for Arithmetic Logic Units in processors, multi-core systems, and Digital Signal Processors (DSPs), rigorous testing has confirmed faster speeds and superior PTA metrics. Our Vedic multiplier deviates from conventional approaches by focusing on concatenation as the key operation for creating partial products. This method efficiently merges binary digits or bit strings crosswise, generating partial products that are then concatenated. An addition operation follows, producing a final 128-bit multiplication result. This innovative approach simplifies the multiplication process and reduces computational complexity, leading to better PTA outcomes. The paper showcases significant improvements in size, area, and power consumption, enhancing computational efficiency and resource utilization. This development represents a significant leap in multiplier design, paving the way for progress in various fields. Its integration into computing systems heralds a new era of digital computation, offering unmatched efficiency and leading to new levels of performance and power efficiency.**

**Keywords— Vedic Multiplier, power, area, time, processor.**

## I. INTRODUCTION

The Urdhva Tiryagbhyam sutra, highlighted by Bharati Krishna Tirtha, is a cornerstone of Vedic mathematics, rooted in ancient Indian teachings. This sutra introduces a novel approach to multiplication that simplifies the process by using “vertically and crosswise" multiplication. Instead of the numerous intermediate steps typical of traditional methods, this technique employs a pattern of vertical and horizontal operations to calculate the product. In essence, the Urdhva Tiryagbhyam method involves vertical and crosswise multiplication, combining these to form different vectors, which are then added together to derive the final product. This results in fewer steps and less complexity, leading to quicker and easier outcomes. The Vedic multiplier, which incorporates this sutra, therefore stands as a faster and more efficient alternative to conventional multiplication algorithms. Its streamlined approach makes it particularly useful for mental arithmetic and computational tasks, offering a significant advantage in speed and simplicity.

## II. VEDIC MULTIPLICATION

This research introduces an innovative 64-bit Vedic multiplier design, representing an advancement in the development of core components for the Arithmetic Logic Unit (ALU), especially for multicores. The new design, based on the Urdhva Tiryagbhyam sutra, significantly impacts a wide range of digital system architectures, leading to substantial performance gains. This method boosts multiplication speed while reducing execution steps. A comprehensive evaluation using design metrics and Power, Timing, and Area (PTA) reports reveals that this Vedic multiplier inherently improves PTA, reinforcing its advantages.

The 64-bit design employs the Urdhva Tiryagbhyam method, which involves vertical and crosswise multiplication at the binary bit level. This approach markedly increases multiplication speed while reducing computational steps. To demonstrate the effectiveness of this technique, the research provides a detailed step-by-step guide, clearly explaining how this sutra is applied in practice. The diagram in Figure 1 depicts the computation process for our Vedic multiplier.

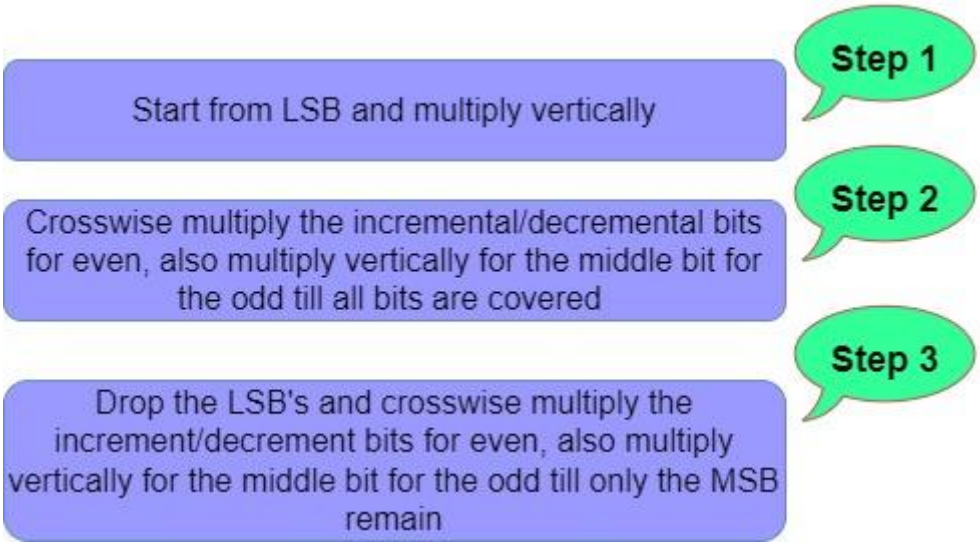


Figure 1: Steps in Vedic Multiplication

The calculation begins with the rightmost bits using vertical multiplication, where each bit is multiplied by its corresponding bit in the other operand. Next comes crosswise multiplication, involving diagonal multiplication of bit pairs from both operands. The results from these operations are concatenated to create partial products. This pattern continues iteratively, moving toward the most significant bit (MSB) in both operands. Simultaneously, a similar set of operations is carried out from the leftmost bits, ensuring a balanced and symmetrical calculation. By adhering to this sequence of steps, our Vedic multiplier achieves a balance of precision, efficiency, and scalability, offering a robust method for complex numerical computations.

## III. ADVANCED METHODOLOGY

The development of Vedic multipliers initially focused on native implementations for various configurations, such as 4*4, 8*8, 16*16, and 32*32, all based on the Urdhva Tiryagbhyam sutra. This ancient method, meaning "vertically and crosswise," simplifies multiplication by breaking down the process into smaller, more manageable steps. However, a detailed examination of these early implementations revealed a critical flaw: many did not support all possible bit combinations, which limited their overall effectiveness and reliability in practical applications. Recognizing this limitation, we undertook a comprehensive review and enhancement of the existing algorithms. Our goal was to optimize traditional methods to achieve superior performance while maintaining the fundamental principles of the Urdhva Tiryagbhyam sutra. This effort led to the development of a truly native implementation for a 64*64-bit multiplier.

In this article, we present the result of our enhancements: a refined and optimized 64*64-bit Vedic multiplier. This new algorithm addresses the shortcomings of previous versions, ensuring comprehensive support for all bit combinations and enhancing overall efficiency. The refinement process involved rigorous testing and iterative improvements, ensuring that the final design would deliver robust performance across various digital system architectures. To evaluate the performance of our 64*64-bit Vedic multiplier, we conducted a thorough comparison with other well-known multipliers, such as the Modified Booth and Array multipliers. Our analysis focused on key metrics, including crisscross multiplication efficiency, area occupancy, and power consumption. The results were compelling: our Vedic multiplier demonstrated a threefold increase in speed, a comparable reduction in area usage, and an exponential decrease in power consumption compared to traditional designs.

These improvements highlight the strategic advantages of our Vedic multiplier, making it a compelling alternative for large-scale digital designs. By offering remarkable efficiency, resource optimization, and exceptional performance, our solution sets the stage for a new era in computational efficiency. This advancement has the potential to transform the design and implementation of digital systems, providing a powerful tool for engineers and researchers working on complex numerical computations. In conclusion, our enhanced 64x64-bit Vedic multiplier not only overcomes the limitations of previous implementations but also offers significant benefits in terms of speed, area, and power efficiency. By demonstrating these advantages, we aim to inspire further innovations in the field and pave the way for more efficient and effective digital systems.

## IV. LOGICAL CALCULATION FOR VEDIC SUTRA

Figure 2 illustrates the step-by-step process of our Vedic multiplication, beginning with the least significant bit (LSB). The first step, shown in k[0], involves vertical multiplication, where each bit is multiplied by its corresponding bit in the other operand. Step 2, depicted in k[1] and k[2], incorporates crosswise multiplication for even positions and vertical multiplication for the middle bit in cases of an odd number of bits. This pattern continues until all bits are processed. Step 3, illustrated by k[4], focuses on crosswise multiplication of increment/decrement bits for even numbers and vertical multiplication for the middle bit for odd numbers, dropping the LSB and moving towards the Most Significant Bit (MSB).

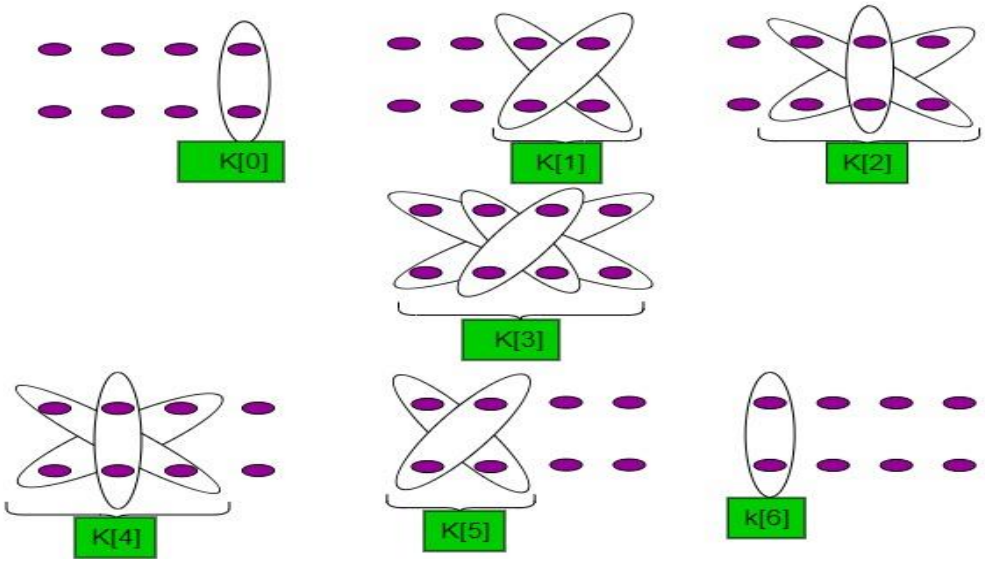


Figure 2. Line diagram for the native Urdhva Tiryagbhyam 4-bit algorithm.

Crisscross multiplication, a fundamental principle in Vedic multiplication, is effectively executed through bitwise AND operations between specific bits of `a` and `b`. This method breaks down the multiplication into smaller, manageable parts, simplifying the overall computation.

In the Verilog code shown in Figure 3, the `k` array stores the partial products generated during the Vedic multiplication process. Each `k[i]` value is calculated by concatenating the results of vertical and crisscross multiplications between specific bits of the input operands `a` and `b`. The initial partial product, `k[0]`, is obtained by applying a bitwise AND operation to the LSB's of both `a` and `b`, forming the first bit of the resulting product. Subsequent partial products are calculated similarly, involving different bit selections from `a` and `b`. For instance, `k[1]` is derived from the LSB of `a` and undergoes AND operation with the next bit of `b` and vice versa, resulting in a two-bit partial product. This iterative approach continues, working towards the MSB, with the partial products gradually combined into the final product of the Vedic multiplication process.

```
k[0] = a[0] & b[0];

k[1] = {(a[0] & b[1]) , (a[1] & b[0])};

k[2] = {(a[0] & b[2]) , (a[1] & b[1]) , (a[2] & b[0])};

k[3] = {(a[0] & b[3]) , (a[1] & b[2]) , (a[2] & b[1]) , (a[3] & b[0])};

k[4] = {(a[1] & b[3]) , (a[2] & b[2]) , (a[3] & b[1])};

k[5] = {(a[2] & b[3]) , (a[3] & b[2])};

k[6] = {(a[3] & b[3])};
```

Figure 3. K values for 4-bit Vedic Multiplier to generate Partial Product

Figure 4 demonstrates how individual bits from the crisscross multiplication of a 4-bit Vedic Multiplier are combined to form the final product. The result,`s`, is constructed from a sequence of partial products, `k`, with correct alignment and padding. Each `s[i]` corresponds to a specific bit position in the final output, from the LSB, i.e.,'s[0]', to the MSB, i.e.,'s[3]`. To form 's[0]`, the code concatenates the LSB's from `k[6]` to `k[0]`, placing each bit in its correct position. Other `s[i]` values are similarly constructed by extracting bits from the corresponding partial products. For example, `s[1]` is created from the second bits of `k[5]` to `k[1]`, while `s[2]` is derived from the third bits of these partial products, and so on. The code strategically adds `1'b0` values to ensure proper alignment and padding, maintaining the structure of the final output.

```
s[0] = {k[6][0],k[5][0],k[4][0],k[3][0],k[2][0],k[1][0],k[0][0]};

s[1] = {1'b0,k[5][1],k[4][1],k[3][1],k[2][1],k[1][1],1'b0};

s[2] = {1'b0,1'b0,k[4][2],k[3][2],k[2][2],1'b0,1'b0};

s[3] = {1'b0,1'b0,1'b0,k[3][3],1'b0,1'b0,1'b0};
```

Figure4. Individual bit concatenation of criss-cross multiplication for 4-bit Vedic Multiplier.

After constructing the `s` vectors, the final step as shown in equation 1 involves adding them together to obtain the final result. This follows traditional binary addition principles, starting from the LSB and moving towards the MSB, carrying over any overflow. For multiplying two 4-bit numbers, the code generates four `s` vectors, and the final product is computed as:

$$\text{Result (product of `a` and `b`)} = s[0] + s[1] + s[2] + s[3]; \quad (1)$$

This step-by-step addition of `s` vectors exemplifies the efficiency and speed of the Vedic multiplication process, allowing for quicker computation compared to traditional binary multiplication techniques.

## V. MULTIPLICATION STAGES FOR 64-BIT VEDIC MULTIPLIER

In this study, we conducted extensive simulations to evaluate the performance and accuracy of our Vedic multiplier implementation. Using a simulator tool, we ran the Verilog code with various 64-bit input values, covering a broad range of numerical scenarios. The simulation results demonstrated that the Vedic multiplier consistently produced accurate results for all tested input combinations, confirming its robustness and reliability across diverse numerical sets. The Vedic multiplier's efficiency and precision were evident regardless of the input values, proving its capability to handle multiplication operations effectively. Additionally, snapshots of the 64-bit Verilog code provide a detailed look into the step-by-step computation process, illustrating how bits are concatenated to form the intermediate `k` and `s` values. These snapshots highlight the systematic approach of the Vedic multiplier algorithm.

```
k[62] = { (a[0] & b[62]) , (a[1] & b[61]) , (a[2] & b[60]) , (a[3] & b[59]) , (a[4] & b[58]) , (a[5] & b[57]) , (a[6] & b[56]) , (a[7] & b[55]) , (a[8] & b[54]) , (a
[9] & b[53]) , (a[10] & b[52]) , (a[11] & b[51]) , (a[12] & b[50]) , (a[13] & b[49]) , (a[14] & b[48]) , (a[15] & b[47]) , (a[16] & b[46]) , (a[17] & b[45]) , (a[18] &
b[44]) , (a[19] & b[43]) , (a[20] & b[42]) , (a[21] & b[41]) , (a[22] & b[40]) , (a[23] & b[39]) , (a[24] & b[38]) , (a[25] & b[37]) , (a[26] & b[36]) , (a[27] & b
[35]) , (a[28] & b[34]) , (a[29] & b[33]) , (a[30] & b[32]) , (a[31] & b[31]) , (a[32] & b[30]) , (a[33] & b[29]) , (a[34] & b[28]) , (a[35] & b[27]) , (a[36] & b[26])
, (a[37] & b[25]) , (a[38] & b[24]) , (a[39] & b[23]) , (a[40] & b[22]) , (a[41] & b[21]) , (a[42] & b[20]) , (a[43] & b[19]) , (a[44] & b[18]) , (a[45] & b[17]) , (a
[46] & b[16]) , (a[47] & b[15]) , (a[48] & b[14]) , (a[49] & b[13]) , (a[50] & b[12]) , (a[51] & b[11]) , (a[52] & b[10]) , (a[53] & b[9]) , (a[54] & b[8]) , (a[55] &
b[7]) , (a[56] & b[6]) , (a[57] & b[5]) , (a[58] & b[4]) , (a[59] & b[3]) , (a[60] & b[2]) , (a[61] & b[1]) , (a[62] & b[0]) };

k[63] = { (a[0] & b[63]) , (a[1] & b[62]) , (a[2] & b[61]) , (a[3] & b[60]) , (a[4] & b[59]) , (a[5] & b[58]) , (a[6] & b[57]) , (a[7] & b[56]) , (a[8] & b[55]) , (a
[9] & b[54]) , (a[10] & b[53]) , (a[11] & b[52]) , (a[12] & b[51]) , (a[13] & b[50]) , (a[14] & b[49]) , (a[15] & b[48]) , (a[16] & b[47]) , (a[17] & b[46]) , (a[18] &
b[45]) , (a[19] & b[44]) , (a[20] & b[43]) , (a[21] & b[42]) , (a[22] & b[41]) , (a[23] & b[40]) , (a[24] & b[39]) , (a[25] & b[38]) , (a[26] & b[37]) , (a[27] & b
[36]) , (a[28] & b[35]) , (a[29] & b[34]) , (a[30] & b[33]) , (a[31] & b[32]) , (a[32] & b[31]) , (a[33] & b[30]) , (a[34] & b[29]) , (a[35] & b[28]) , (a[36] & b[27])
, (a[37] & b[26]) , (a[38] & b[25]) , (a[39] & b[24]) , (a[40] & b[23]) , (a[41] & b[22]) , (a[42] & b[21]) , (a[43] & b[20]) , (a[44] & b[19]) , (a[45] & b[18]) , (a
[46] & b[17]) , (a[47] & b[16]) , (a[48] & b[15]) , (a[49] & b[14]) , (a[50] & b[13]) , (a[51] & b[12]) , (a[52] & b[11]) , (a[53] & b[10]) , (a[54] & b[9]) , (a[55] &
b[8]) , (a[56] & b[7]) , (a[57] & b[6]) , (a[58] & b[5]) , (a[59] & b[4]) , (a[60] & b[3]) , (a[61] & b[2]) , (a[62] & b[1]) , (a[63] & b[0]) };

.....

k[126] = { (a[63] & b[63]) };
```

Figure 5. Snippet for multi stage criss-cross multiplication

```
s[0] = { k[126][0],k[125][0],k[124][0],k[123][0],k[122][0],k[121][0],k[120][0],k[119][0],k[118][0],k[117][0],k[116][0],k[115][0],k[114][0],k[113]
[0],k[112][0],k[111][0],k[110][0],k[109][0],k[108][0],k[107][0],k[106][0],k[105][0],k[104][0],k[103][0],k[102][0],k[101][0],k[100][0],k[99][0],k[98]
[0],k[97][0],k[96][0],k[95][0],k[94][0],k[93][0],k[92][0],k[91][0],k[90][0],k[89][0],k[88][0],k[87][0],k[86][0],k[85][0],k[84][0],k[83][0],k[82]
[0],k[81][0],k[80][0],k[79][0],k[78][0],k[77][0],k[76][0],k[75][0],k[74][0],k[73][0],k[72][0],k[71][0],k[70][0],k[69][0],k[68][0],k[67][0],k[66]
[0],k[65][0],k[64][0],k[63][0],k[62][0],k[61][0],k[60][0],k[59][0],k[58][0],k[57][0],k[56][0],k[55][0],k[54][0],k[53][0],k[52][0],k[51][0],k[50]
[0],k[49][0],k[48][0],k[47][0],k[46][0],k[45][0],k[44][0],k[43][0],k[42][0],k[41][0],k[40][0],k[39][0],k[38][0],k[37][0],k[36][0],k[35][0],k[34]
[0],k[33][0],k[32][0],k[31][0],k[30][0],k[29][0],k[28][0],k[27][0],k[26][0],k[25][0],k[24][0],k[23][0],k[22][0],k[21][0],k[20][0],k[19][0],k[18]
[0],k[17][0],k[16][0],k[15][0],k[14][0],k[13][0],k[12][0],k[11][0],k[10][0],k[9][0],k[8][0],k[7][0],k[6][0],k[5][0],k[4][0],k[3][0],k[2][0],k[1]
[0],k[0][0]};

.....

s[63] = {1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0
,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,k[63][63],1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,
1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0 };
```

Figure 6. Snippet for individual bit concatenation of criss-cross multiplication

The final product of multiplying two 64-bit numbers, `a` and `b`, is achieved by summing a series of intermediate results, known as 's' vectors. Each vector represents a different stage in the multiplication process, and their combined outcome yields the final product. In a 64-bit multiplication operation, the product is derived by breaking the process into smaller, manageable partial products. These partial products are created through bitwise operations on the bits of `a` and `b`. Each bit in one operand is multiplied by each bit in the other, resulting in a set of intermediate values. These values are then `grouped into a sequence of 's' vectors, each representing a specific stage in the multiplication. The final step in the 64-bit multiplication process, illustrated in Figure 7, involves adding all the 's' vectors to obtain the final result. This step is similar to binary addition, where carries from one bit position are propagated to the next as needed. The summed output is the complete product of the two 64-bit numbers, `a` and `b`.

```
Product of two 64 bit binary number = s[0] + s[1] + s[2] + s[3] + s[4] + s[5] + s[6] + s[7] + s[8] +
s[9] + s[10] + s[11] + s[12] + s[13] + s[14] + s[15] + s[16] + s[17] + s[18] + s[19] + s[20] + s[21] +
s[22] + s[23] + s[24] + s[25] + s[26] + s[27] + s[28] + s[29] + s[30] + s[31] + s[32] + s[33] + s[34]
+ s[35] + s[36] + s[37] + s[38] + s[39] +  s[40] + s[41] + s[42] + s[43] + s[44] + s[45] + s[46] + s
[47] + s[48] + s[49] + s[50] + s[51] + s[52] + s[53] + s[54] + s[55] + s[56] + s[57] + s[58] + s[59] +
s[60] + s[61]+ s[62] + s[63];
```

Figure 7. Result for multiplication of 64 bit numbers using ‘s’ vector

By breaking the multiplication of two 64-bit numbers into smaller calculations and reassembling the final product through the addition of 's' vectors, this structured approach provides an efficient method for large-scale multiplication in digital systems. The systematic summing of partial products ensures accuracy and consistency, offering a reliable solution for complex multiplication tasks.

## VI. APPLICATION

The enhanced 64*64-bit Vedic multiplier marks a significant milestone in computational technology, with far-reaching implications across critical fields. In digital signal processing, its swift and precise multiplication capabilities promise to enhance the speed and accuracy of signal manipulation and analysis tasks, impacting industries such as telecommunications and multimedia processing. Likewise, in image processing, the Vedic multiplier's efficiency in handling large datasets accelerates image transformation and enhancement processes, benefiting areas like medical imaging and computer vision.

In cryptography, its quick and secure multiplication processes strengthen encryption algorithms, ensuring the security of sensitive information in financial transactions and communications. Additionally, in high-performance computing, the Vedic multiplier's rapid execution of multiplication operations enhances computational speed and efficiency, advancing scientific simulations and data analysis tasks. Finally, in communication systems, it facilitates the seamless processing of signals and data streams, supporting the efficient transmission of information across various networks. Overall, the Vedic multiplier's versatility and effectiveness promise to drive significant advancements in computational efficiency and performance, shaping the future of digital systems and applications.

## VII. RESULT & CONCLUSION

The multiplication of `a` and `b`, with decimal values of 18,446,744,073,394,194,650 and 18,446,744,072,077,428,285 respectively, yields the product `re = 340,282,366,885,013,792,836,538,441,420,705,675,250` in decimal value. As shown in Figure 8, this result is consistent when employing Vedic multiplication for 64-bit numbers using the binary representations of `a` and `b`. The matching results in Figure 8 demonstrate that Vedic multiplication is reliable and flexible, effectively handling different numerical representations. These findings affirm the effectiveness of the Vedic multiplier in delivering precise multiplication outcomes, reinforcing its suitability for various applications requiring fast and accurate arithmetic operations.

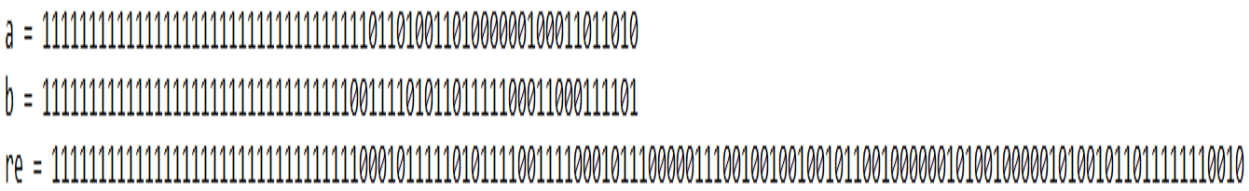


Figure 8: Simulation result Window

The proposed Vedic multiplier surpasses traditional designs by utilizing fewer gates, optimizing the logic for scalability and cost-effectiveness. It achieves superior area efficiency, maximizing integrated circuit space utilization. Furthermore, advancements in power consumption enhance energy efficiency, contributing to sustainable computing. The crisscross multiplication technique employed ensures each bit interacts uniquely, streamlining the computation process. Table 1 highlights the Vedic multiplier's advantages, including lower propagation delays, reduced power consumption, simplified timing complexities, and a smaller footprint compared to traditional 64-bit multipliers.

Table I. Comparison of previous work with proposed word

| Parameter | Booth encoded parallel multiplier | Modified Booth Encoding Multiplier | Vedic Multiplier |
|---|---|---|---|
| **Gate Count** | 20557 | 17196 | 4604 |
| **Area (µm2 )** | 102183 | 85983 | 23183 |
| **Power (mW)** | 13.7 | 12.1 | 3.24 |

Beyond these immediate benefits, the Vedic multiplier represents a breakthrough in digital design, offering a foundational component that has the potential to reshape the use of existing library elements. Its efficiency and adaptability make it a valuable building block for future developments in processors, multi-core architectures, and digital signal processors (DSPs). This combination of efficiency and versatility suggests that the Vedic multiplier could play a key role in driving innovation and redefining approaches to digital design.

## REFERENCES


[1] Shri Bharati Krishna Tirtha, “Vedic Mathematics”:

[2] Li-Rong Wang, Shyh-Jye Jou and Chung-Len Lee, "A well-structured modified Booth multiplier design," 2008 IEEE International Symposium on VLSI Design, Automation and Test (VLSI-DAT), Hsinchu, Taiwan, 2008, pp. 85-88, doi: 10.1109/VDAT.2008.4542418.

[3] D. Govekar and A. Amonkar, "Design and implementation of high speed modified booth multiplier using hybrid adder," 2017 International Conference on Computing Methodologies and Communication (ICCMC), Erode, India, 2017, pp. 138-143, doi: 10.1109/ICCMC.2017.8282661.

[4] S. Lee and Y. Kim, "Booth Fusion: Efficient Bit Fusion Multiplier with Booth Encoding," 2020 International SoC Design Conference (ISOCC), Yeosu, Korea (South), 2020, pp. 73-74, doi: 10.1109/ISOCC50952.2020.9332943.

[5] H. D. Tiwari, G. Gankhuyag, Chan Mo Kim and Yong Beom Cho, "Multiplier design based on ancient Indian Vedic Mathematics," 2008 International SoC Design Conference, Busan, Korea (South), 2008, pp. II-65-II-68, doi: 10.1109/SOCDC.2008.4815685.

[6] Neethu Johny, Mayur S. M."Design and Implementation of a Low Power Vedic Multiplier ", Volume 6, Issue IV, International Journal for Research in Applied Science and Engineering Technology (IJRASET) Page No: 4156-4162, ISSN : 2321-9653, DOI : 10.22214/ijraset.2018.4686

[7] G. S. Sai Venkatramana Prasada, G. Seshikala and N. Sampathila, "Performance Analysis of 64×64 bit Multiplier Designed Using Urdhva Tiryakbyham and Nikhilam Navatashcaramam Dashatah Sutras," 2018 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore, India, 2018, pp. 28-31, doi: 10.1109/DISCOVER.2018.8674125.

[8] P. Saha, A. Banerjee, P. Bhattacharyya and A. Dandapat, "High speed ASIC design of complex multiplier using Vedic Mathematics," IEEE Technology Students' Symposium, Kharagpur, India, 2011, pp. 237-241, doi: 10.1109/TECHSYM.2011.5783852.