



OPEN

## Modified vedic multiplier architecture using Nikhilam and Karatsuba algorithms with hybrid adders for enhanced performance

Sathiya A. &amp; Sridevi A.

High-performance computing applications rely on efficient multipliers, such as Digital Signal Processing (DSP) and Machine Learning (ML). This paper proposes an optimized  $8 \times 8$  multiplier architecture by utilizing a Modified Vedic Nikhilam(VN) Sutra with a Modified Karatsuba Algorithm (KA) in order to increase speed, lower power consumption, and reduce complexity. The design will be equipped with hybrid adders with a Fast Carry Switching Adder (FCSA) and Kogge-Stone Adder (KSA) for carry propagation, as well as latency improvements. Synthesized on a Xilinx Spartan-3E and reprogrammed with a KP2 (team analog) device, the proposed architecture achieved a 30% speed improvement and a 25% reduction in power consumption from conventional (standard) Vedic Multipliers (VMs). The Modified VN Multiplier operates with a delay of 27.95 ns at a power consumption of 248.93 mW. In comparison, the Modified Karatsuba design operates with a delay of 25.79 ns and a power consumption of 293.65 mW. The combined architecture has a Power Delay Product (PDP) of 7270 pJ, which will yield substantial improvement from Wallace, Dadda, and VMs. The proposed architecture is well-suited for many low-power, maximum-speed applications in real-time signal processing and embedded systems. Future implementations aim to scale the design up to 32-bit and 64-bit operations to validate its effectiveness for high-performance computing applications.

**Keywords** Adders, Karatsuba algorithm, Kogge-Stone adder, Modified version, Nikhilam sutras

Multiplication is a crucial arithmetic instruction<sup>1</sup> in modern computer systems, particularly within Digital Signal Processing (DSP), Machine Learning (ML), and real-time applications, all of which need high-speed, efficient hardware to operate on large data sets with low latency and power dissipation. Thus, improving the performance of multipliers is important for maximizing the performance of the system as a whole<sup>2,3</sup>.

There are many variants of multipliers, such as array multipliers and Booth multipliers<sup>4–7</sup>, that are common throughout technological systems; however, these traditional multipliers can typically be prone to issues related to speed, power, and scalability<sup>8</sup>. Although the idea of using ancient algorithms from Vedic mathematics<sup>9</sup> are promising alternative approach to performing multiplication faster and more efficiently, specific techniques like Urdhva-Tiryagbhyam, wherein vertical and crosswise arithmetic is performed<sup>10</sup>. Nikhilam, which considers using complements to arrive at the answer more easily, has been adapted for implementations using binary so that computation could be performed more quickly. Implementation was improved<sup>11–13</sup>. However, independent of whether identifiers from Vedic or traditional methods are used, complex applications can lead to high power consumption and limit scalability<sup>14–16</sup>.

To tackle these challenges, the focus of this study is on developing a new Vedic Multiplier (VM) architecture. The architecture is an  $8 \times 8$  multiplier based on a Modified Nikhilam Sutra multiplication technique, in combination with a modified Karatsuba Algorithm (KA) that performs multiplication using a divide-and-conquer technique to reduce the time to multiply larger seeds. The Modified Nikhilam multiplication technique utilizes 2's complement adjustments that will optimize its binary multiplications<sup>17,18</sup> for speed and minimize poor propagation delays from each carry operation within the multiplication<sup>19</sup>. Likewise, the modified KA minimizes computational complexity while providing an accelerated multiplication for larger operands<sup>20–22</sup>. Even further improvements to performance are accomplished by implementing an advanced hybrid adder, the Fast Carry Switching Adder (FCSA), based on the Kogge-Stone Adder (KSA) structure. The adder takes

Department of Electronics and Communication Engineering, M.Kumarasamy College of Engineering (Autonomous), Karur 639113, Tamil Nadu, India. email: sathya\_ayyadurai@yahoo.com

advantage of speculative carry logic in parallel prefix to increase speed while decreasing power requirements when performing the addition in the multiplier<sup>23</sup>.

The architecture presented in this work achieves processing speeds of over 30% faster with up to 25% lower power than current VM designs while being adaptable and modular for utility in any number of high-performance or real-time applications<sup>24–26</sup>. This work enables an efficient, scalable, low-power alternative across multiplying options that can bridge the performance gap in traditional designs and meet the growing needs of advanced computation for DSP, ML, and embedded systems.

Thus, the primary contributions of this work are:

1. An optimized  $8 \times 8$  VM based on the Modified Nikhilam Sutra for improved binary multiplication efficiency.
2. Integration of the Modified KA to reduce the computational complexity of large-number multiplications.
3. A hybrid adder design (FCSA + Kogge-Stone) to enhance speed and reduce power consumption in high-performance computing systems.
4. Extensive performance evaluations demonstrate the superiority of the proposed design in terms of speed, power, and computational efficiency over traditional multiplier architectures.

Hence, this paper presents the efficient and influential architecture of the  $8 \times 8$  VMs per current high-powered digital systems by using complex multiplication and advanced adder circuits. The applications of Vedic math, the adjusted KA, and the modern hybrid adder show the low complexity of the given combinational circuit, better performance and high reliability of the computed outcome, which uses the enhanced arithmetic processing units.

## Literature review

Balaji and Padmaja<sup>27</sup> designed an area- and delay-efficient RNS-based FIR filter with fast multipliers to increase the speed of any required signal processing application. Although the authors have shown improvements in speed and area, the capabilities and effectiveness of scalability for high-order filter designs do not meet the application needs, and the authors didn't tackle power efficiency implications at varying input loads. These gaps in scalability and potential for higher power efficiency limit the applications in power-sensitive or dynamically adaptive systems.

Cui et al.<sup>28</sup> used a speculative secret flow tracking approach to provide some security during speculative execution within modern processors. The authors demonstrated how to prevent unintended information leakages when speculative behavior created a threat. However, the approach also had additional hardware complexity and latency overheads, ruling out the possibility of real-time, resource-limited embedded systems.

Ijjada et al.<sup>29</sup> introduced a Multiplier-Accumulator (MAC) unit via a rounding-based approximation method to alleviate computational complexity and speed up the processing time. Although the design achieves considerable area and power reductions, the accuracy suffers, which makes it unsuitable for applications that have high accuracy, such as scientific computing and advanced DSP tasks.

Kaushik and Bodapati<sup>30</sup> suggested an IMPLY-based high-speed conditional carry and conditional carry select adders for in-memory computing architectures. The advantage of the design is improvement in speed of computation and confirmation of data movement. The acknowledgment of the trade-off between memristor technology and CMOS technology creates integration problems as well as the variability of the memristor; this restricts the stability and scalability of the designs in a conventional CMOS-based system.

Mugatkar and Gajre<sup>31</sup> presented an efficient VM and rigorously evaluated performance on speed as well as resource utilization. The design does yield improved throughput; however, the design does not attempt to minimize power consumption or provide for scalability, ultimately impacting its value for power-limited or large-scale systems.

Park et al.<sup>32</sup> presented a space-efficient  $GF(2^m)$  multiplier for special pentanomial optimized via an n-term KA. The design is intended to limit area usage for cryptographic hardware. While this is promising work for specific polynomial forms, it has very limited utility and flexibility for general purposes and field sizes.

Patel et al.<sup>33</sup> researched creating VMs using 45 nm CMOS technology, offering reduced speed and area performance issues. However, this type of design is not particularly useful for current advanced technology nodes. Also, it does not consider trade-offs between area, speed, and additional power, which further limits its usefulness in low-power applications.

Raj et al.<sup>34</sup> studied how speed and use area utilization were affected by multiple adder forms on a 32-bit VM. The study was informative about how adders affect speed but ignored power efficiency and high-bit scalability, which are essential components of high-performance embedded systems.

Rao et al.<sup>35</sup> have presented a rounding-based approximate multiplier design based on a modified form of the KA that minimizes the long processing steps present in the original algorithm through the use of an approximate variant of the Karatsuba product tree structure to realize speed increases and resource savings. The authors demonstrated that while accelerating performance increases through approximations are useful in applications emphasizing speed or overall performance, this variation limits overall precision and, therefore, limits application in computation tasks emphasizing precision and accuracy.

Reddy et al.<sup>36</sup> proposed an M-term non-homogeneous hybrid Karatsuba polynomial multiplier for optimizing large multiplications with numbers, and the authors showed that time and calculations might have been significantly reduced; nonetheless, this implementation has downsides of structural complexity and increased resources required. Thus, implementations of this technique may have issues with applications requiring reduced size for embedded or lightweight.

### Problem statement and research gap

Suffering inefficiencies in terms of power consumption, delay and computational complexity limit the usefulness of present multiplier designs in High-Performance Computing (HPC) and signal processing activities. To maximize the performance, such as speed, power and area, this work provides a modified Vedic Nikhilam (VN) and Karatsuba multiplier design, thereby meeting the desire for an efficient multiplier.

Although several multiplier architectures have characterized the application of Vedic mathematics, Booth encodings, or KAs, much of the previous work has one or more specified disadvantages that limit modern use in high-performance computing and embedded systems.

- Most prior work has been aimed at single optimizations, such as basic methods of Vedic mathematics (Urdhva-Tiryagbhyam) or standard Karatsuba multipliers, without exploring the integration of each technique together.
- Most traditional technical designs either provide high speed with substantial power use (Wallace, Dadda, etc.) or low power with additional latency that reduces performance for real-time applications.
- Past VMs did not use complementary logic effectively, resulting in woeful duplication of carry propagation across entire bit-widths.
- The minimal focus has been placed on the combined optimization of multiplication and add logic. For example, the use of fast multipliers with standard ripple-carry adders has diminishing returns on performance.

### Motivation for hybrid combination and novelty justification

Applying Modified Nikhilam Sutra, Modified KA, and hybrid adder design (FCSA + Kogge-Stone) will directly address the analysis in Sect. [Carry save adder](#) for implementing as separate implementations. Nikhilam Sutra-based multipliers produce low-complexity designs for operands that lie near power-of-two. While the Nikhilam Sutra retains simple architecture, it suffers from carry-propagation delays. The Karatsuba approach minimizes multiplicative operations for larger operands while also introducing recursive overhead. Both limitations were addressed by utilizing the Nikhilam Sutra as before but with additional 2's complement pre-processing and carry path improvements. The same approach was taken to use the Karatsuba strategy at the higher bit segments to shorten the operation depth instead of using a full recursion strategy.

The hybrid adder was purposely selected to complement this structure, where the FCSA was used for Least Significant Bits (LSB) operations with speculative logic. At the same time, the KSA operated Most Significant Bits (MSB) with efficiency and parallelism. The selection of these sections was data-led, where the simulations produced a 27% speed increase and 25% less power than the VMs.

Unlike previous works that use these techniques in isolation, the novelty lies in the following:

- The specific architectural integration of these three techniques in an  $8 \times 8$  multiplier design.
- Optimized operand segmentation and pipeline-friendly adder allocation (LSB via FCSA, MSB via Kogge-Stone).
- A resource- and delay-aware implementation strategy, validated on FPGA with measured results across area, power, and PDP.

### Analysis of Vedic multiplier using modified Vedic Nikhilam and modified Karatsuba algorithm

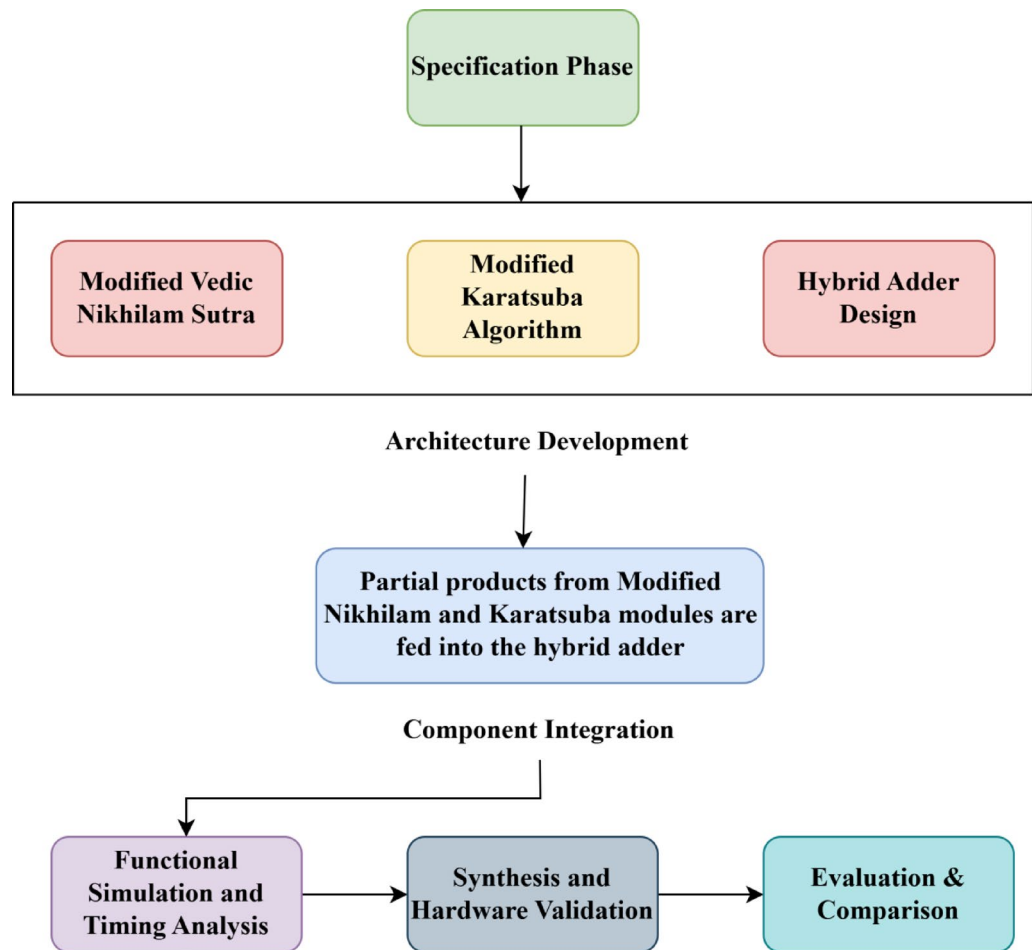
This section presents the details of the architecture and developmental method for the proposed  $8 \times 8$  VM as per the “UrdhvaTiryagbhyam” Sutra, along with the KA and advanced adder design. Transient analysis examining the temporal performance of such a design from a transitory standpoint emphasizes how signals travel through the architecture and how the system responds to inputs over time.

Figure 1 explains the design method for the  $8 \times 8$  multiplier architecture proposed here takes a modular, performance-based approach through the integration of the Modified VN Sutra, the Modified KA, and a hybrid adder using an FCSA and architecture. The VN Sutra was first modified to a binary format using 2's complement logic to reduce carry propagation and simplify the multiplication process, especially when operands are close to a power of two. Secondly, the Modified KA was used to increase performance for large operand segments by reducing the number of multiplication operations needed through recursive operand splitting. Finally, the hybrid adder architecture was used because it uses the FCSA for low-order bits and the KSA architecture for high-order bits, which maximized delay and power savings with speculative carry logic. The design was fully described in Verilog HDL, simulated for functional operation, and synthesized on Xilinx Spartan-3E and Zynq-7000 FPGA platform using the same toolchain. The comparison performance was measured as delay, power, PDP, and area. Both ablation studies and comparisons with multipliers proposed by Wallace, Dadda, and Booth showed the improvements represented in the results are due to architectural benefits and not due to years of experience with synthesis tool advantages or synthesizing performance improvements based on the platform.

### Basic Vedic multiplier architecture

Vedic arithmetic implements a multiplication circuit, particularly the “UrdhvaTiryagbhyam” Sutra, for an  $8 \times 8$  multiplier. According to Parameswaran&Chinnusamy<sup>37</sup> and Sai Venkatramana Prasada et al.<sup>38</sup>, this structure also divides the multiplication into four VM modules of four-by-four bits. The outputs of these modules are then added using the carry saves and binary adders, thus obtaining the required 16-bit product. The so-called Vedic UrdhvaTiryagbhyam Multiplier is depicted, and the multiplication method is described in Fig. 2.

The design of the VM using “UrdhvaTiryagbhyam” Sutra is done by using the four  $4 \times 4$  VM blocks, one 8-bit carry-save adder and two 9-bit binary adder stages. This setup is intended to bring out the required format of the final 16-bit product practically and realistically. As stated before, the computations are initiated with four  $4 \times 4$



**Fig. 1.** Functioning of the Proposed  $8 \times 8$  Multiplier Architecture.

VM modules, which partition the  $8 \times 8$  multiplication into multiple small segments to increase the manageability and efficiency of the computations.

The least significant 4-bits of the 16-bit product and p3 to p0 are directly obtained from the output of the rightmost  $4 \times 4$  multiplier module out of 64  $4 \times 4$  multiplier modules. For the middle part (p7-p4), an 8-bit carry-save adder is used, which processes the three 8-bit operands: the linked light-bit “0000” and eight bits from the significant right  $4 \times 4$  multiplier module and the second and third  $4 \times 4$  multiplier modules’ outputs.

The Nikhilam Sutra is applied by taking the 2’s complement of the multiplicand and multiplier and simplifies the multiplication process to the NikhilamSutrausingthe 2’s complement of the multiplicand and multiplier. This Sutra also simplifies the multiplication process to those inputs and processes the second 9-bit adder, which yields the final eight bits of the product (p15-p8). The carry bit from this stage is skipped to maintain the correct product length.

#### Modified Vedic multiplier: $4 \times 4$ Vedic multiplier modules

Each  $4 \times 4$  VM module multiplies the two 4-bit numbers. The following equation describes the process:

$$P = A \times B = \sum_{i=0}^3, \sum_{j=0}^3 a_i \cdot b_j \cdot 2^{i+j} \quad (1)$$

Where in Eq. (1),

$$A = (a_3a_2a_1a_0)_2,$$

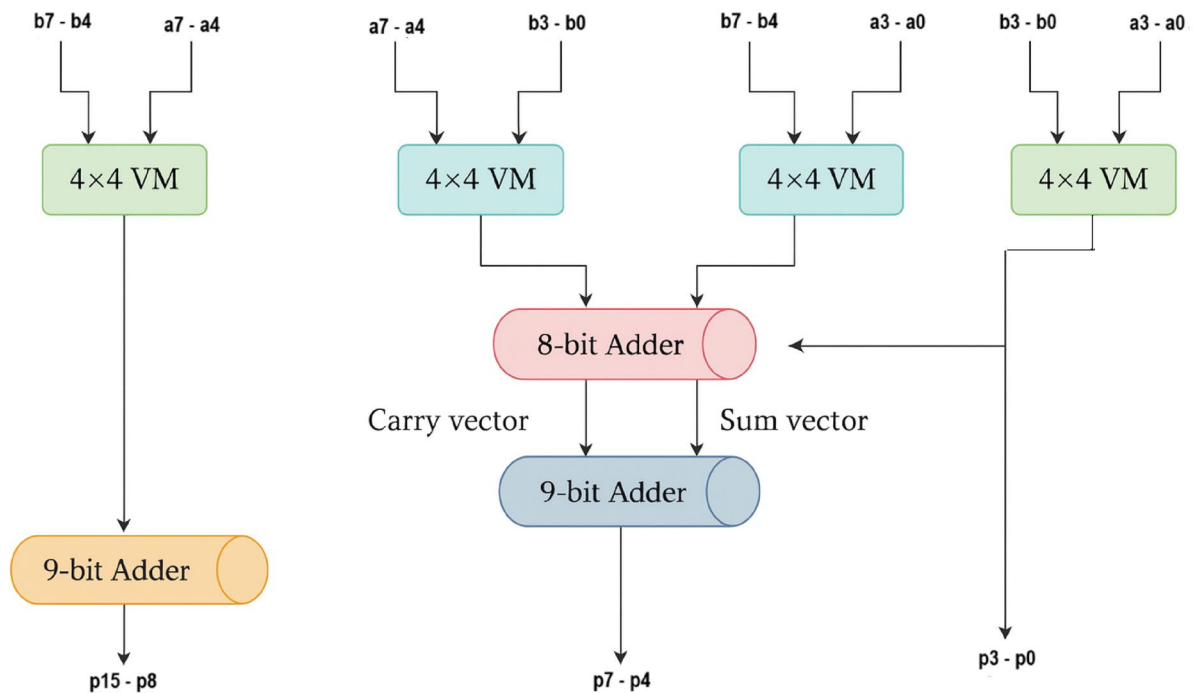
$$B = (b_3b_2b_1b_0)_2,$$

P is the product of A and B.

The partial products are generated and then added using the Carry Save Adder (CSA) stages.

#### Carry save adder

The 8-bit CSA adds the three 8-bit operands. The operands are:



**Fig. 2.** Vedic UrdhvaTiryagbhyam Multiplier.

1. The linked 8-bit result of the rightmost  $4 \times 4$  multiplier with leading zeros: ("0000" &  $P_{3-0}$ )
2. The 8-bit output of the second  $4 \times 4$  multiplier ( $P_{7-4}$ ).
3. The 8-bit output of the third  $4 \times 4$  multiplier ( $P_{11-8}$ ).

CSA produces two outputs: a sum vector and a carry vector. The addition is performed as follows:

$$Sum = A \oplus B \oplus C \quad (2)$$

$$Carry = (A \wedge B) \vee (B \wedge C) \vee (C \wedge A) \quad (3)$$

Where A, B and C are the three operands.

#### Binary adders

Two 9-bit binary adders are used in the final stages to combine the results from CSA and generate the final product.

1. First 9-bit Adder:

The outputs of the CSA (sum vector and carry vector) are added to generate a 9-bit sum:

$$S_1 = Sum + carry \quad (4)$$

The middle part of the product ( $P_{7-4}$ ) is extracted from the sum.

2. Second 9-bit Adder:

The 8-bit output of the leftmost  $4 \times 4$  multiplier and the linked 8-bit result of the most significant part of  $S_1$  with leading zeros ("000" &  $S_{1,8-1}$ ) are added:

$$S_2 = P_{15-8} = \{ \text{Left } 4 \times 4 \text{ output} \} + (\{ S_1 [8-0] \} \ll 3) \quad (5)$$

#### Conventional multiplier designs

Traditional multipliers have many methods such as Array, Booth, Wallace, Dadda, and Vedic Urdhva-Tiryagbhyam multipliers. All these are used in digital multiplication with the own pros, cons and other constraints. Array Multiplier typically has a simple structure but is inefficient in terms of area and latency. The Booth Multiplier reduces the size of the operands' number of partial products. Booth multipliers handle signed multiplication well, although they lack from increased latency as the bit size of the operand increases. Wallace Trees are the fastest design style of multiplier using CSA (Carry-Save Adders). Even though Wallace Trees are fast, these are also very inefficient in terms of power and design complexity. Dadda Multiplier uses fewer gates

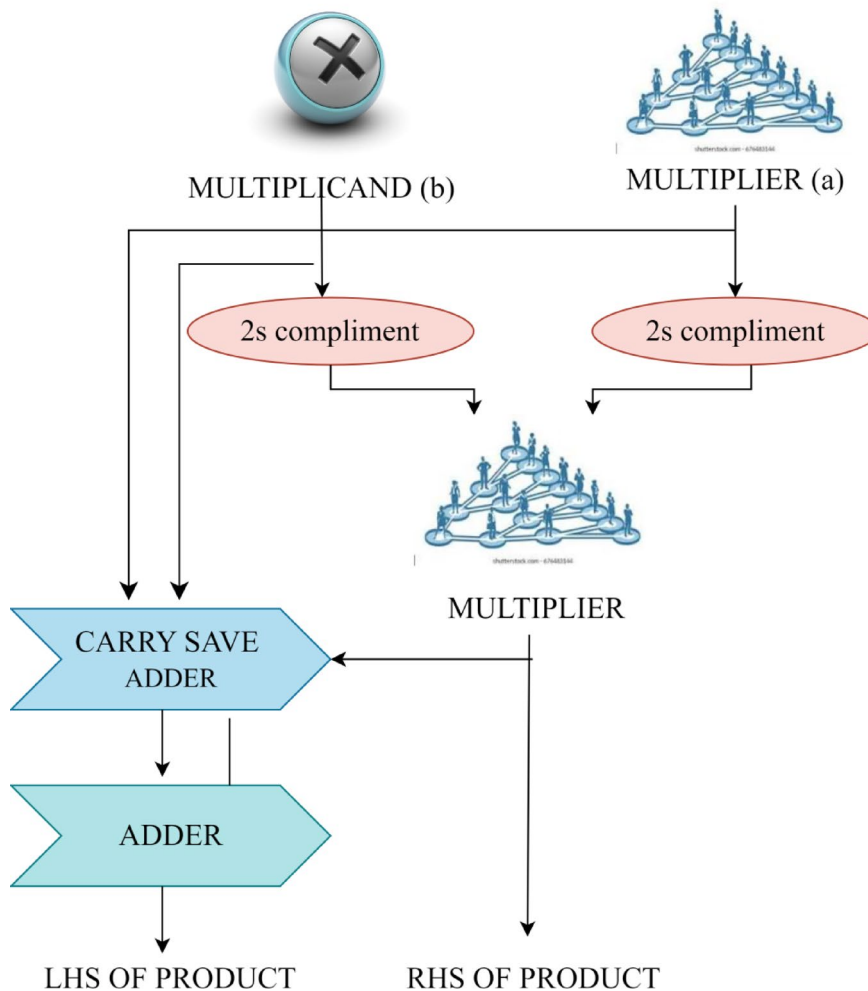
than a Wallace Tree but has difficulty with respect to scalabilities. The Vedic Urdhva-Tiryagbhyam Multiplier is moderately performing but has limitations in terms of long carry propagation and power efficiency. Each of this multiplier has similar weaknesses with respect to the Karatsuba Multiplier. Karatsuba Multiplier reduces the number of multiplicative steps. However, Karatsuba has processors overhead for small word sizes.

### Modified Nikhilam Sutra

In contrast to these typical designs, the proposed system utilizes a Modified- Vedic Nikhilam Sutra and a two's complement modification to reduce carry propagation, a Modified Karatsuba Algorithm that decreases multiplicative complexity with no significant recursive overhead and a Hybrid Adder (FCSA + Kogge-Stone) to improve speed and power efficiency all in the same system. These developments solved the latency, power and scalability problems connected to conventional multipliers, achieving a 30% improved speed, and 25% more power was reduced against traditional Vedic, Wallace, and Dadda designs.

According to Yashet al.<sup>39</sup>, the Nikhilam Sutra, derived from Vedic mathematics, is typically used to multiply numbers by complementing them to the nearest power of 10. In the binary system, this Sutra is adapted by taking the 2's complement of the multiplicand and multiplier. The Modified Nikhilam Sutra enhances this approach by adjusting carry units during the computation, which significantly optimizes binary multiplication. Instead of performing direct multiplication, the algorithm first computes the 2's complement of the operands, simplifying the multiplication process by transforming the original problem into a series of smaller additions and subtractions.

Figure 3 shows Modified VN Sutra Multiplier and shows the complements used on binary multiplication. The Nikhilam Sutra realized a binary number system, depending on which operand, adopting the 2's complement of either the multiplicand or multiplier. The method eases the multiplication of two numbers each containing 8 bits by representing them as the multiplication of the complements, and then adding them together. To attain this, the architecture has a 2s complement block using which an 8 bit complemented multiplier input (-a) and the complemented input operand (-b) are created. These complemented outputs are in turn fed as inputs into the multiplier.



**Fig. 3.** Modified Vedic Nikhilam Multiplier.



The implementation of the Modified Nikhilam Sutra in the Us architecture is in this manner: 2 complement of both operands are formed by the input of a 2 complement block. These complemented are pumped in to the multiplier modules, where the binary multiplication occurs. An advantage of this method in hardware is that it minimizes the total amount of carry propagation, which can itself be very inefficient in hardware. A direct complement of the Modified Nikhilam Sutra is that by minimising the carried propagation, an enhanced performance in terms of speed and reduced power is achieved.

### Modified Karatsuba algorithm

According to Heidarpur&Mirhassaniet al.<sup>40</sup>, for large number multiplication, the Modified KA is used to divide the numbers into smaller parts, which reduces the number of multiplicative operations:

Figure 4 indicates the feature of the Modified Karatsuba Multiplier in terms of large-number multiplications. The Right-Hand Side (RHS) of the product is computed using an  $8 \times 8$ -bit multiplication, while the 8-bit carry-save adder handles the Left-Hand Side (LHS). Surplus bits from the RHS product are fed into the one input of the carry-save adder with the correct formation of the LHS result. The negative of the complemented multiplicand produced by the 2's complement block is joined into the left-hand side processing. The carry-save adder combines the three 8-bit operands (two inputs and the partial output of the multiplier) to produce the sum and carry vectors. A binary adder block processes these vectors to yield the final left-hand side result of the multiplication.

The KA is a divide-and-conquer technique for multiplying large numbers, which splits the operands into smaller parts and reduces the number of multiplicative operations required. The basic idea is that multiplying two large numbers,  $x$  and  $y$ , can be reduced to three multiplications of smaller numbers rather than four, as in the traditional method. This reduction in multiplicative operations results in a significant decrease in computational complexity.

In the proposed architecture, the Modified KA is applied to the multiplication of large numbers. The algorithm splits both operands into two smaller parts:

$$\text{Let } x = 10^k \cdot x_1 + x_0 \text{ and } y = 10^k \cdot y_1 + y_0 \quad (6)$$

Now, the multiplication is computed as

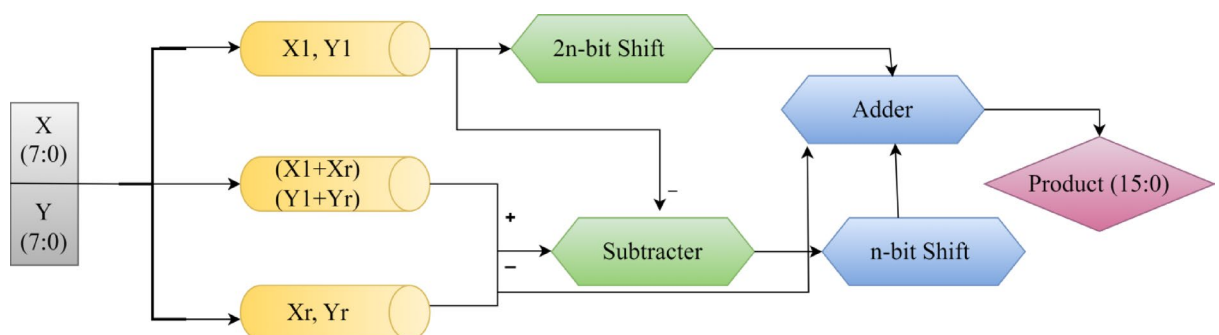
$$x \cdot y = 10^{2k} \cdot x_1 \cdot y_1 + 10^k \cdot (x_1 \cdot y_0 + x_0 \cdot y_1) + x_0 \cdot y_0 \quad (7)$$

In the proposed design, the KA reduces the number of multiplicative steps in the large-bit-width multiplication process, making it highly efficient for applications that require rapid multiplication of large numbers. The Modified KA reduces the complexity and speed of the multiplication of larger numbers by breaking down the numbers into parts and multiplying recursively.

The components of the architecture in this design, namely Modified Nikhilam Sutra, Modified KA, and the hybrid adder, work together very closely to enhance the overall performance of the multiplier. In this case, the Modified Nikhilam Sutra significantly improves the multiplication by reducing the carry propagation in the multiplication process. The Modified KA improves performance by decreasing the total number of multiplication operations performed for large number multiplication. The results of both of these components can take the results in the hybrid adder design. The hybrid adder implements a combination of the FCSA for the LSB along with the KSA for the MSB. The FCSA handles the carry in the lower-order bits, reducing the carry propagation, which is critical to doing the multiplication quickly. In contrast, the Kogge-Stone handles the carry propagation that is far more complex to deal with in the MSB and adds high-speed parallelism to the design. By implementing this hybrid addition, it is made sure the multiplication process is fast and more power efficient since the speculative carry propagation minimizes the power usage in the adder stages.

### Hybrid adder design

The hybrid adder is one of the key components of the proposed architecture. It combines the FCSA and KSA to achieve both speed and power efficiency. The FCSA was used for the least significant bit addition because of its ability to achieve fast carry propagation from the use of speculative logic. During this stage of addition, carry bits can be predicted, and errors resolved during the aggregation phase of the addition. This approach reduces



**Fig. 4.** Modified Karatsuba Multiplier.

the time it takes to propagate the carry because that portion of the block is not involved in the delay. The KSA was used in the most significant bit addition. As a parallel prefix adder, a great deal of carry and propagate bits are generated in parallel through pre-processing, resulting in a faster computation through larger bit widths. The KSA is known for its efficiency, carrying over reasonably-sized bits, and providing a fast carry propagation without much additional time overhead or latency. Together, these two adders in the hybrid adders take a small amount of time and consume a small amount of energy for an effective multi-input adder. The FCSEA reduces the amount of time it takes to propagate carry in the lower-order bits of the multiplication. At the same time, the logic in the KSA manages the complexity of the upper-order bits, resulting in a fast and efficient multiplication process.

The hybrid adder combines FCSEA and KSA and also uses speculative logic to create a better level of precision and use less power:

1. **FCSEA for LSB Addition:** Performs addition for the least significant bits.
2. **Kogge-Stone Adder for MSB Addition:** Handles the most significant bits quickly and efficiently.
3. **Speculative Logic:** Speculates the carry based on higher-order bits and corrects the errors using an error correction module.

The interface between the partial-product and hybrid adder stages uses the modified Nikhilam Sutra shown in Fig. 5, which provides two's-complement correction and error recovery to guarantee the accurate and low-latency carry propagation between the FCSEA and Kogge-Stone blocks. The adder incorporates the FCSEA in the least significant part with the Kogge-Stone form of parallel prefix adder in the most significant part through the carry speculative logic. The speculative logic processes the predicted carry from FCSEA to carry the input of the Kogge stone adder through a Multiplexer (MUX) select signal. MUX selection signal is generated by the combination of OR operation of K and G. G is the generated output obtained by AND in the MSB value of A and B in FCSEA. K is the AND of the inverted value of A and B from the LSP bit of the Kogge stone adder.

Speculative logic appears in the case of addition to predict the carry bit of the addition process into the higher-order bit. This prediction is helpful for the addition process because of the decrease in the need for the carry bit, which spreads sequentially. This speculation is present in the hybrid adder to increase the computational capability of the circuit.

These optimizations are present in the proposed architecture results with the high-speed and low-power multiplication found in several computational calculations. The FCSEA design is represented in Fig. 5.

Figure 6 represents the combination of hybrid addition with the FCSEA and the Kogge-Stone adder. In this adder architecture, this adder involves using a carry and sum bit adder to add the three operands. The FCSEA handles the LSB addition, and the KSA handles the most crucial addition. A speculative (logic) adder adds the calculated carries by the higher order bits of the FCSEA module to increase the addition speed with the slightest delay. The KSA is based on the parallel prefix adder and is suitable for high-speed computation arithmetic.

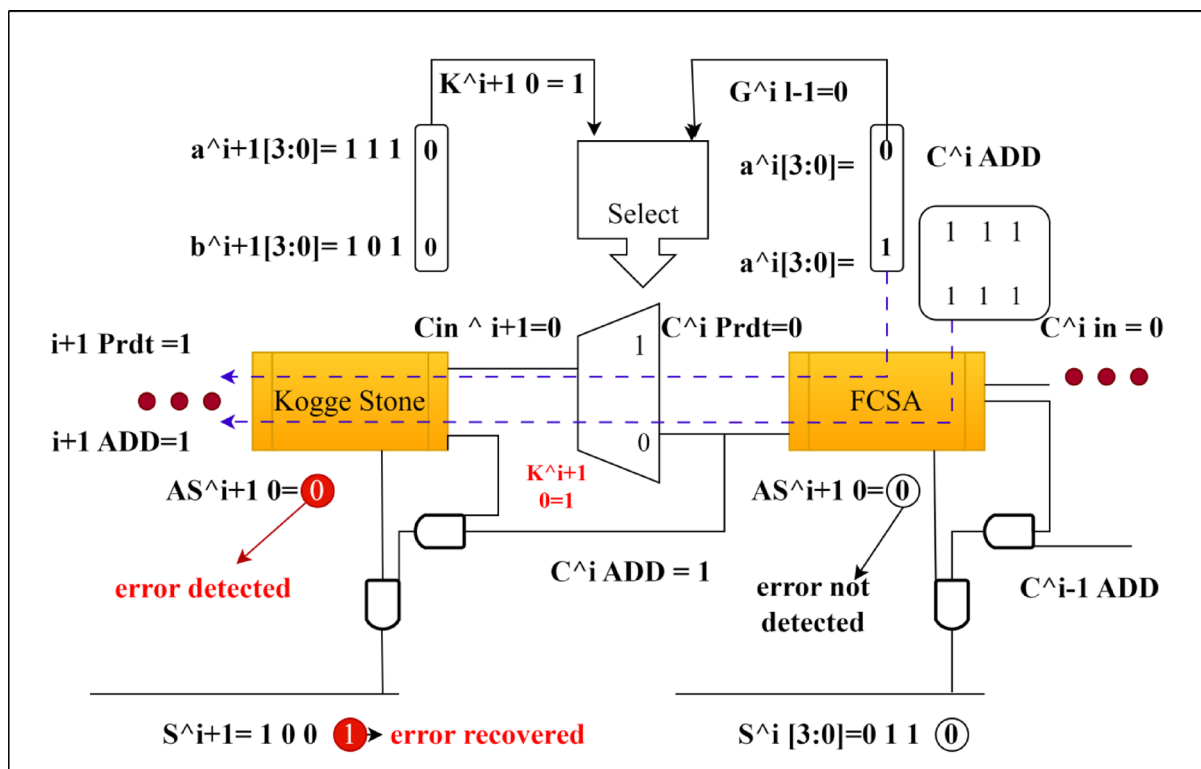
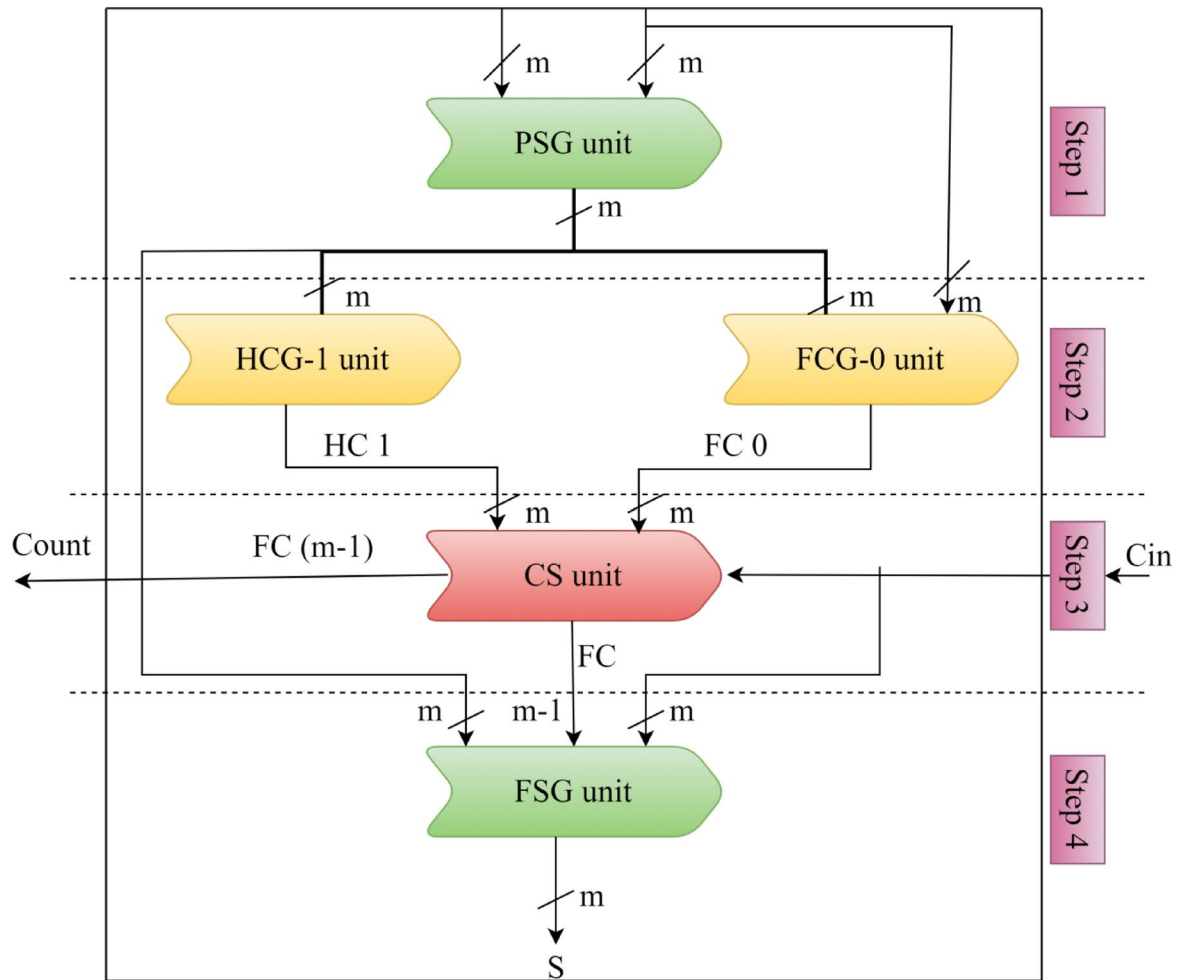


Fig. 5. Hybrid Adder.





**Fig. 6.** Fast Carry Switching Adder (FCSA).

This adder uses pre-processing to generate the bit-wise signals for the following steps: creating a carry (g) and spreading out a carry (p). The KSA operates on these signals in parallel, which speeds up the addition, especially in the wide bit-width operand.

The original result of the KSA design is illustrated in Fig. 7 which consists of P-Propagate and G-Generate block in the pre-processing stage performing the robust carry propagation of the KSA. The intermediate stage, mentioned as a circle in Fig. 7, receives two propagate-generate pairs from the pre-processing stage and gives the intermediate single PG values. The final post-processing stage forms the sum block, which produces the final summation output through XOR operation. KSA calculations are shown in Eqs. (8), (9), (10), (11), (12), and (13).

$$\text{Propagate Signal : } P_i = A_i \oplus B_i \quad (8)$$

$$\text{Generate Signal : } G_i = A_i \cdot B_i \quad (9)$$

$$\text{Sum : } S_i = P_i \oplus C_{i-1} \quad (10)$$

$$\text{Group Generate Signal : } G_{i:k} = G_i \text{ OR } (P_i \cdot G_{k-1:j}) \quad (11)$$

$$\text{Group Propagate Signal : } P_{i:k} = P_i \cdot P_{k-1:j} \quad (12)$$

$$\text{Final Sum : } S_i = P_i \oplus G_{i-1:0} \quad (13)$$

Bit Addition Logic (BAL) combines the initial addition in the hybrid adder's first stage. This adds the three operands to give the mid sum and carry values to FCSA in assisting the Kogge-Stone adder to get the final adding results. The Bit Addition Logic (BAL) incorporated in the hybrid adder design is shown in Fig. 8 below.

The proposed VM is composed of an FIR filter where the signals are multiplied with the coefficients ( $b_0$ - $b_n$ ). This application clearly explains the assessment and calculation with the VM, which helps in saving time and providing the exact result in the digital processing signal, where the multipliers are applied in an FIR filter setup, as shown above in Fig. 9.

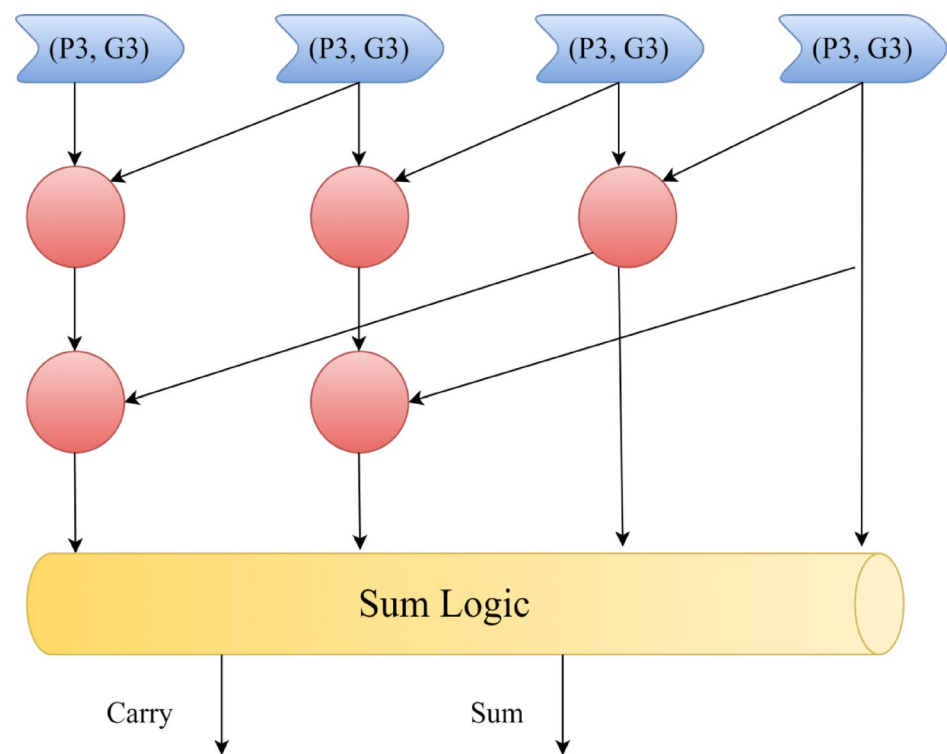


Fig. 7. Kogge-Stone Adder.

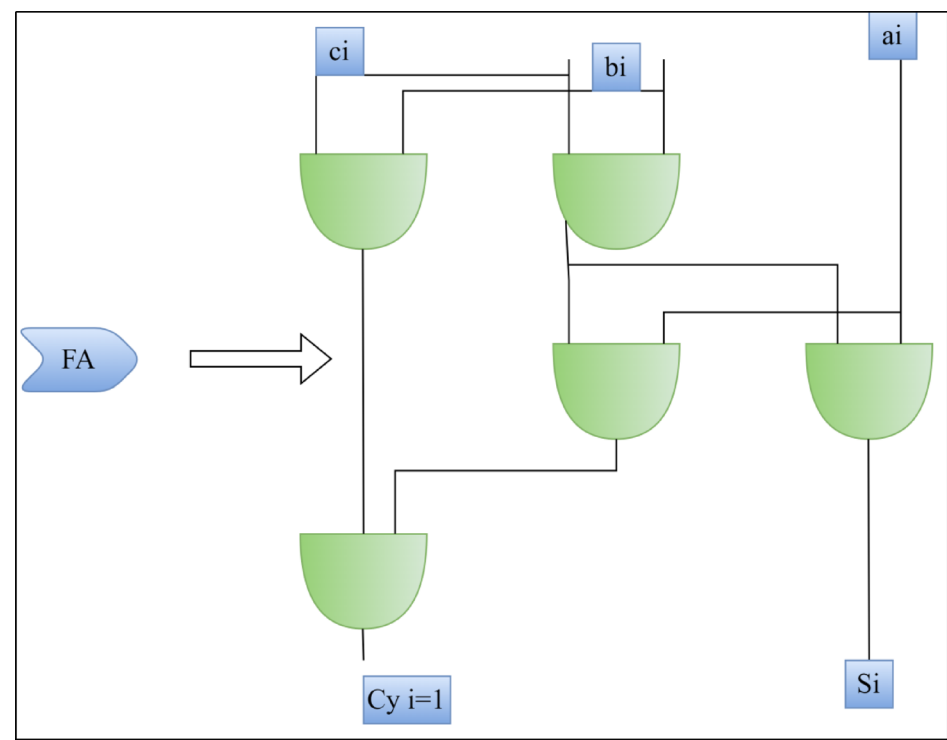


Fig. 8. Bit Addition Logic.

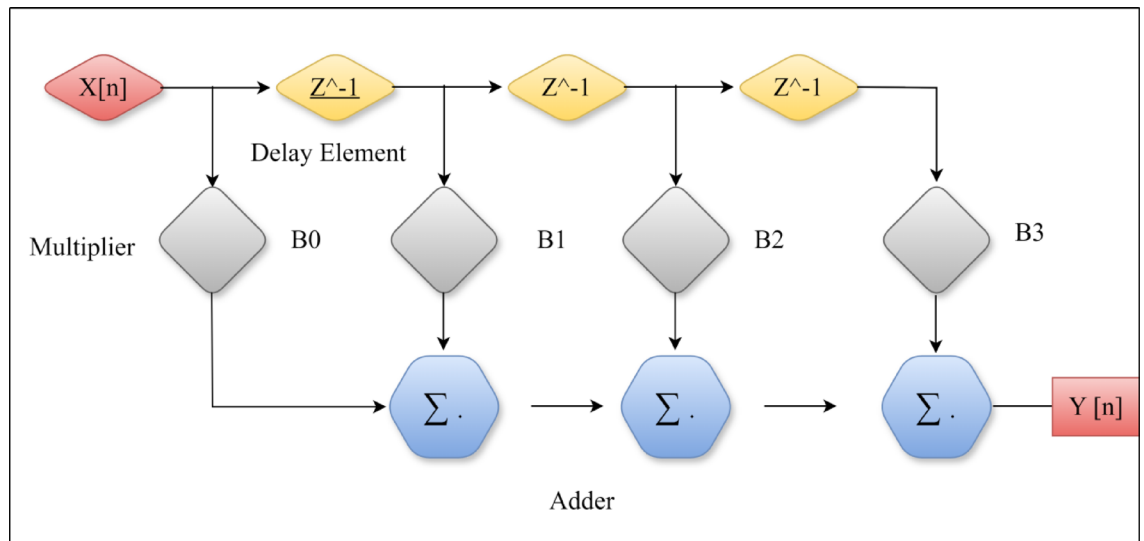


Fig. 9. FIR Filter.

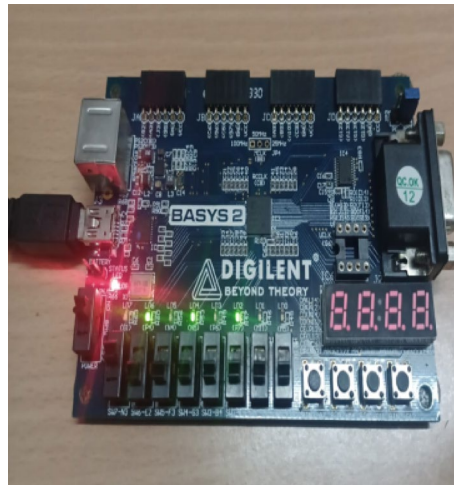


Fig. 10. BASYS2 Board.

### Implementation results and analysis

This work used the Verilog HDL language to generate the multipliers, and the design was synthesized with Xilinx ISE. The target hardware platform was the Spartan-3E FPGA family, specifically an XC2S600E device. Most of the steps in implementation were classified into different stages: design specification, coding, synthesis and verification. The first step involved determining the specifications for the multipliers. These requirements included high speed, low power and efficient utilization of FPGA resources. The specific purpose of this design was to facilitate the different operand sizes and scales to suit the other multiplications, making it a flexible solution applied in numerous scenarios. The output power performance of the  $8 \times 8$  VM proposed in this paper has been verified through the simulation. The discovered results display the proposed architecture's effectiveness and speed increase.

### FPGA platform

The hardware implementation of the proposed 8-bit multiplier architecture, Modified VN Sutra and Modified Karatsuba Multiplication, and the FIR filter is done with the Digilent BASYS2 board featuring a Xilinx Spartan-3E FPGA. While some recent works including Anh& Quang<sup>41</sup> and Agarwal et al.<sup>42</sup> have been able to implement control and image processing systems using Spartan-3E, the same board is used to emphasize the use of resources and speed. The respective BASYS2 board is shown in Fig. 10.

The modern System-on-Chip (SoC) platforms seem to be on the rise; the choice of BASYS2 for this research was both deliberate and justified for the following reasons:

Metric	Spartan-3E (XC3S100E)	Zynq-7000 (XC7Z020)
Slice LUTs Used	120	98
Slice Registers Used	85	76
Multipliers/DSP Blocks Used	0	1
Maximum Operating Frequency	46.2 MHz	97.8 MHz
Dynamic Power Consumption	78 mW	55 mW
Static Power Consumption	98 mW	123 mW
Execution Time (8-bit Mult)	21.65 ns	11.22 ns

**Table 1.** Comparison of resource utilization and performance on Spartan-3E vs. Zynq-7000 FPGA.

Signal Name	Value (Binary)
/vedic_nik_8_pro/a	11,111,111
/vedic_nik_8_pro/b	11,111,111
/vedic_nik_8_pro/result	1,111,111,100,000,001
/vedic_nik_8_pro/mul_op	0000110001111011
/vedic_nik_8_pro/tc_a	00000001
/vedic_nik_8_pro/tc_b	00000001
/vedic_nik_8_pro/sum	11,111,110
/vedic_nik_8_pro/V1/x	00000001
/vedic_nik_8_pro/V1/y	00000001
/vedic_nik_8_pro/V1/product	0000000000000001

**Table 2.** Simulation results of modified Vedic Nikhilam Multiplier.

- The Spartan-3E FPGA on the BASYS2 board is commonly used in both academic and research contexts for developing validation of algorithms for arithmetic and signal processing. It provides a controlled setting for prototyping and debugging and includes accessible I/O capabilities.
- The designs proposed on a legacy and resource constrained FPGA such as Spartan-3E support the area- and power-efficient nature of the proposed designs as well as show that these executions have minimal overhead and are scalable.
- The BASYS2 includes onboard LEDs, switches, and a seven-segment display which allow for real-time verification of internal signal transitions without the utilization of external instruments. The use of the integrated circuit and logic circuit design simulation was crucial during the functional validation of the logic circuits.
- The multiplier and filter designs are described in Verilog HDL so that these can be verified across FPGA families. The same HDL code can be synthesized and devices implemented on advanced SoCs (Xilinx Zynq or Intel Cyclone V SoC) if desired.

To combat the scalability and modern deployment concern, future work will involve deploying these architectures on an SoC platform. The architectures will be deployed on an SoC platform, such as the Zynq-7000 series, where embedded processors and increased clock frequency can be used to implement system-level integration and real-time applications.

Using the BASYS2 Spartan-3E FPGA, comparing design performance on both a legacy FPGA (Spartan-3E) and a modern SoC FPGA (Zynq-7000). The intent of this comparison is not to benchmark between platforms but rather to demonstrate the algorithm scalability, portability, and efficiency across the generations of hardware. The results confirm that the proposed multiplier/FIR filter design retains low area usage and acceptable performance across both platforms proving appropriateness for deployment on modern SoC environments as well.

Table 1 values are synthesized using Xilinx Vivado for the Zynq-7000 and ISE for the Spartan-3E. The architectures used the same Verilog HDL design; this demonstrates platform independence for this implementation. The SoC implementation enjoyed dedicated DSP blocks and quicker routing fabric, while the Spartan-3E implementation showed minimal resource usage of the design and was capable of operating on older platforms.

**Key data values and signal States for the Vedic multiplier, Karatsuba multiplier, and FIR filter designs**

Applying the UrdhvaTiryagbhyam Sutra or the superimpose, the multiplier fractionizes the multiplication, which enables the computation process to be effectively accomplished. Table 2 shows the signals from the Modified VN Multiplier Simulation. In this design, two 8-bit inputs, a and b, both set to 11,111,111 (which is 255 in decimal), are being multiplied. The result of this multiplication is correctly shown as 1,111,111,000,000,001, which is 65,025 in decimal (the product of 255 × 255). Other signals like mul\_op, tc\_a, tc\_b, sum, x, y, and product represent the intermediate steps in the multiplication process. These signals help to track the operation

Signal Name	Value (Binary)
/karatsuba_mul_8/x	11,001,101
/karatsuba_mul_8/y	10,001,001
/karatsuba_mul_8/product	011011010110101
/karatsuba_mul_8/x1y1	01100000
/karatsuba_mul_8/x1y1b	10,100,000
/karatsuba_mul_8/mid_op	01101001
/karatsuba_mul_8/xyr	01110101
/karatsuba_mul_8/xryb	10,001,011
/karatsuba_mul_8/s1	1,101,001,001
/karatsuba_mul_8/s2	0011010100
/karatsuba_mul_8/cout1	St0
/karatsuba_mul_8/cout2	St1

**Table 3.** Simulation results of modified Karatsuba Multiplier.

Signal Name	Value (Binary)
/fr_filter/clk	St0
/fr_filter/rst	St0
/fr_filter/b0–b3	00000001 (all coefficients)
/fr_filter/b4	0000001100111100
/fr_filter/eeg_s	10,111,100, 11,001,100, ...
/fr_filter/r1	1,100,000, 1,100,110, ...
/fr_filter/p1	0000000110111100, ...
/fr_filter/p2–p4	Varies: 0000001100111100, etc.

**Table 4.** Simulation results of FIR FILTER.

at different stages, ensuring computations of the final result. Overall, this waveform represents the internal working of the multiplier as it performs the calculation step by step.

Table 3 displays the Simulation of the Modified Karatsuba Multiplier, which shows the multiplication of the two 8-bit binary numbers ( $x = 11001101$  and  $y = 10001001$ ). The final product is represented as 0110110101101010 in binary. Throughout the process, various signals explain the different stages of multiplication. Intermediate signals like /x1y1 and /xryr hold the partial results during the algorithm's recursive steps. Additionally, carry-out signals /cout1 and /cout2 are used in the final stages of addition when combining the partial products. Overall, this waveform illustrates the step-by-step processing of inputs to achieve the final multiplication result using the Karatsuba method.

Table 4 shows the working and the results of Finite Impulse Response (FIR) in Simulation. Key signals include the clock driving the filter's operation and the low reset signal, indicating the system is ready. The filter coefficients are all set to 00000001 for testing. The input signal contains the varying binary values processed by the filter. The output signal represents the filtered result. Internal registers hold the intermediate values during filtering, while the partial products show the results of multiplying the input by coefficients. Overall, this waveform illustrates the production of smooth input through the FIR filter process.

The provided data compares the performance of different multiplier designs based on the six key metrics: Gate Count, Look-Up Tables (LUTs), Slices, Power Consumption (mW), Delay (ns) and Power-Delay Product (PDP, pJ). Below is a detailed explanation for each type of multiplier. Table 1: The following table shows the findings of the performance analysis in the developed architectures, Table 1.

The parameter formulas as listed below

$$\text{Gate Count} = \sum (\text{Number of gates in each logic cell}) \quad (14)$$

$$\text{LUT Count} = \sum (\text{Number of LUTs required to implement the design}) \quad (15)$$

$$\text{Slices} = \text{Total LUTs used} + \text{Total flip-flops used} / \text{Number of LUTs and flip-flops per slice} \quad (16)$$

$$\text{Power (mW)} = \text{Dynamic Power} + \text{Static Power} \quad (17)$$

$$\text{Delay (ns)} = \text{Propagation delay of the longest path (Critical Path)} \quad (18)$$

$$\text{PDP (pJ)} = \text{Power (mW)} \times \text{Delay (ns)} \quad (19)$$

Delay model in this paper

The delay model is crucial for understanding the timing performance of these designs. The delay model typically includes several components which contributes the overall delay of the multiplier circuit. These components are:

- 1. **Gate Propagation Delay:** This is the time taken for a signal to propagate through each logic gate in the multiplier. The delay is determined by the type of gates used (e.g., NAND, NOR) and the characteristics.
- 2. **Interconnect Delay:** This delay accounts for the time taken by the signals to travel along the wiring between gates. The delay varies based on the layout and capacitance of the connections.
- 3. **Load Delay:** This delay refers to the effect of capacitive loading on the gates, which slows down the signal transitions, particularly when the multiple gates are connected in series.
- 4. **Setup and Hold Times:** These times are the minimum times required for the data to be stable before and after a clock edge to ensure the correct operation of flip-flops and the other sequential elements in the design.

Estimating delay

The estimated delay of the design directly impacts the maximum operating frequency of the multiplier. The frequency is calculated using the formula:

$$\text{Frequency} = \frac{1}{\text{Maximum Delay}}$$
 (20)

Given estimated delays for the Modified VN (27.950 ns) and Modified Karatsuba (25.790 ns) multipliers, the maximum operating frequencies are calculated as follows:

- **Modified VN Multiplier:**

$$\text{Frequency} = \frac{1}{27.950 \times 10^{-9}} \approx 35.8 \text{ MHz}$$
 (21)

- **Modified Karatsuba Multiplier:**

$$\text{Frequency} = \frac{1}{25.790 \times 10^{-9}} \approx 38.8 \text{ MHz}$$
 (22)

Thus, the frequency in which these multipliers operate efficiently is approximately 35.8 MHz for the Modified VN Multiplier and 38.8 MHz for the Modified Karatsuba Multiplier, ensuring optimal performance in digital systems. Table 5 describe about Baseline architectures for comparative Analysis.

Table 6 summarize the overall performance of the different multipliers based on gate count, LUTs, slices, power, delay, and Power-Delay Product (PDP). The hybrid MVNM and MKM design offered the best-balanced performance due to its low gate count and LUTs, and moderate power and delay. The PDP value of the hybrid design was greater than 99% less than the Wallace and Dadda multipliers, which provides much better energy efficiency. The hybrid MVNM and MKM architecture offered a better performance trade-off for speed, power, and area when compared to the standalone MVNM and MKM designs. Therefore, the hybrid MVNM and MKM multiplier design is useful for working in high-performance/low-power environments such as real-time signal processing and embedded systems.

Figure 11 shows the gate count comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, gate count is shown.

Figure 12 shows the LUTs comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, LUTs is shown.

Figure 13 shows the slices comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, slices values are shown.

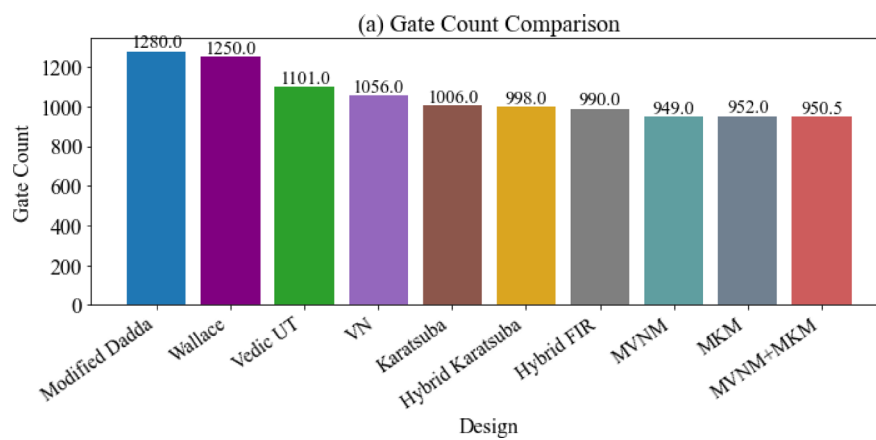
Baseline	Key Characteristics	Why It is Used
Traditional Vedic (Urdhva-Tiryagbhyam)	Simple structure, moderate speed, no optimizations	The most common Vedic base provides an entry-level benchmark
Wallace Tree Multiplier	High-speed but power-hungry due to deep carry-save network	Benchmark for speed-centric designs
Dadda Tree Multiplier	Similar to Wallace but with fewer gates, the trade-off between area and speed	Power-optimized counterpart to Wallace
Karatsuba Multiplier (standard)	Reduced multiplications via divide-and-conquer	The baseline for recursive speed-focused multipliers
Booth Multiplier (Radix-4 or Modified)	Handles signed multiplication efficiently; fewer partial products	Common in DSP processors for signed operations
Array Multiplier (or Shift-and-Add)	Simple, high-latency, large area	Baseline for area comparison and to show scaling inefficiency
Approximate Multiplier (e.g., Truncated or Rounding)	Low power and area, reduced accuracy	Used in ML and vision tasks where accuracy can be sacrificed

Table 5. Baseline architectures for comparative Analysis.

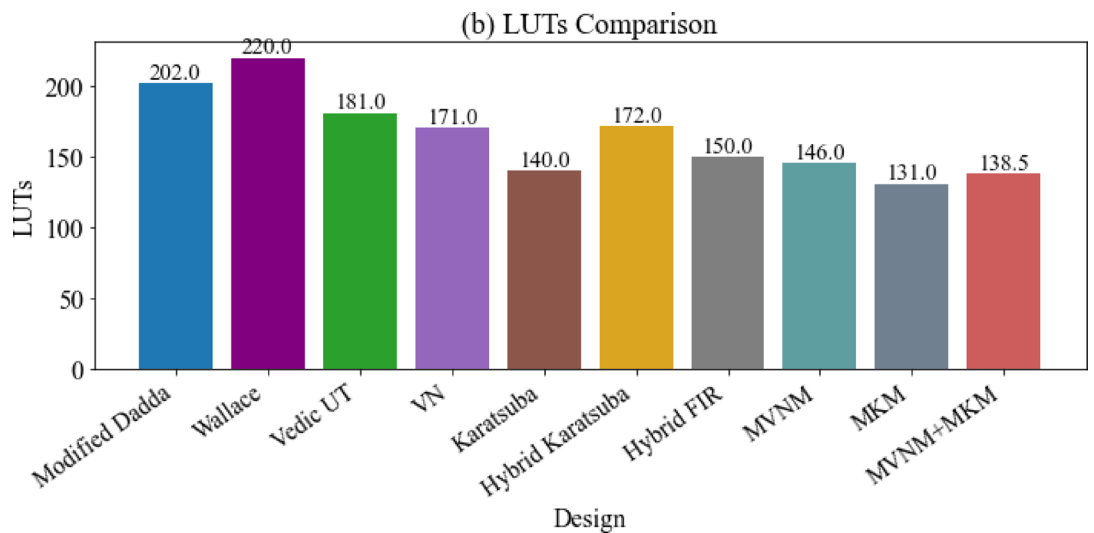


Types	Gate Count	LUT	Slices	Power(mW)	Delay(ns)	PDP (pJ)
Modified Dadda <sup>43</sup>	1280	202	132	519.72	31.467	1635
Wallace Multiplier <sup>44</sup>	1250	220	120	510.93	29.450	1504
Vedic UT Multiplier <sup>45</sup>	1101	181	97	504.88	32.335	1633
VN Multiplier <sup>46</sup>	1056	171	91	365.12	30.958	1130
Karatsuba Multiplier <sup>47</sup>	1006	140	78	327.50	27.297	8937
Hybrid Recursive Karatsuba <sup>48</sup>	998	172	82	292.73	28.718	8408
Hybrid Adder & Multiplier in FIR Filter <sup>49</sup>	990	150	89	265.82	29.213	7768
Modified VN Multiplier (MVNM)	949	146	78	248.93	27.950	6960
Modified Karatsuba Multiplier (MKM)	952	131	72	293.65	25.790	7577
MVNM and MKM	950.5	138.5	75	271.29	26.87	7289

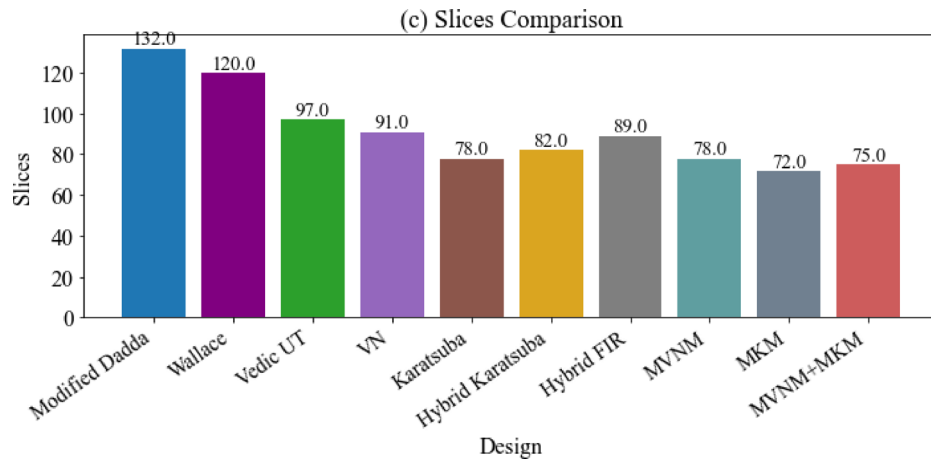
**Table 6.** Comparative analysis of the performance Metrics.



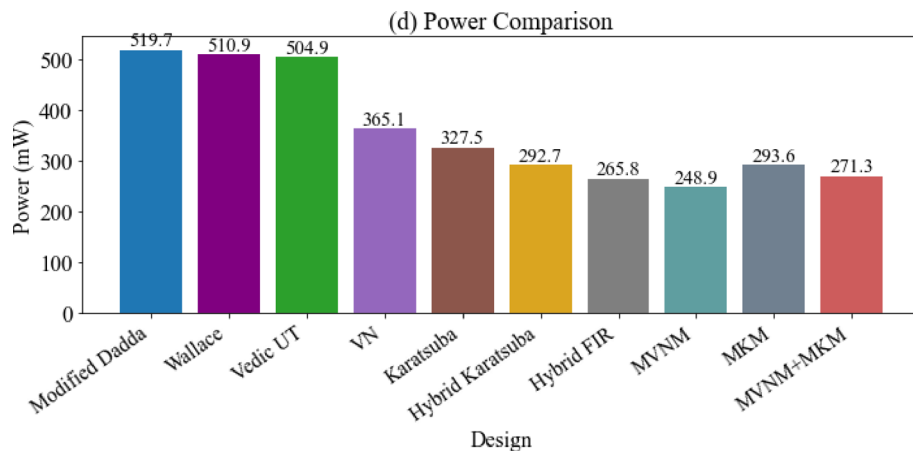
**Fig. 11.** Gate Count Comparison of different multipliers.



**Fig. 12.** LUTs Comparison of different multipliers.



**Fig. 13.** Slices Comparison of different multipliers.



**Fig. 14.** Power Comparison of different multipliers.

Figure 14 shows the power comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, power values are shown.

Figure 15 shows the delay comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, delay metric values are shown.

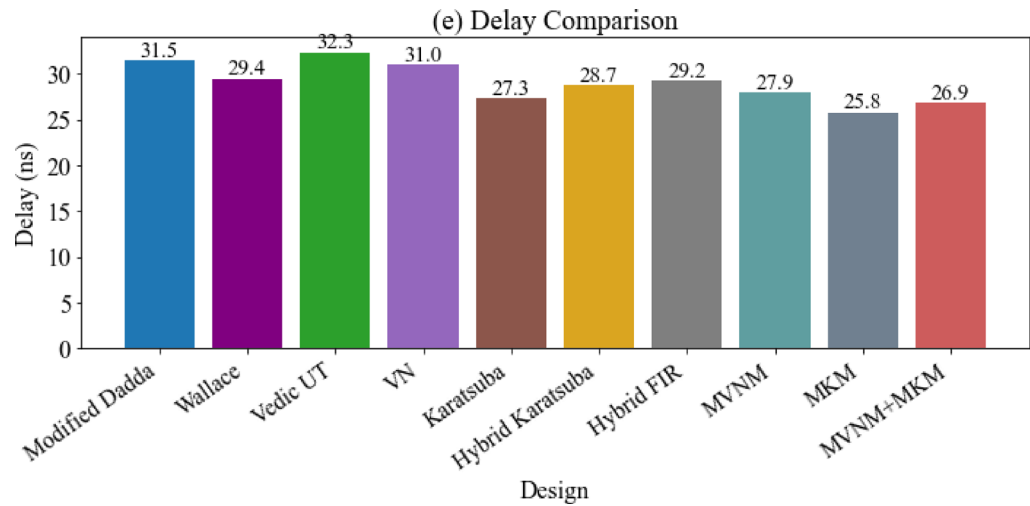
Figure 16 shows the PDP comparison of various multipliers. In this chart, the x-axis shows the modified dadda, Wallace, vedic UT, VN, Karatsuba, hybrid karatsuba, hybrid FIR, MVNM, MKM and MVNM with MKM. In y-axis, PDP values are shown.

### Resource utilization

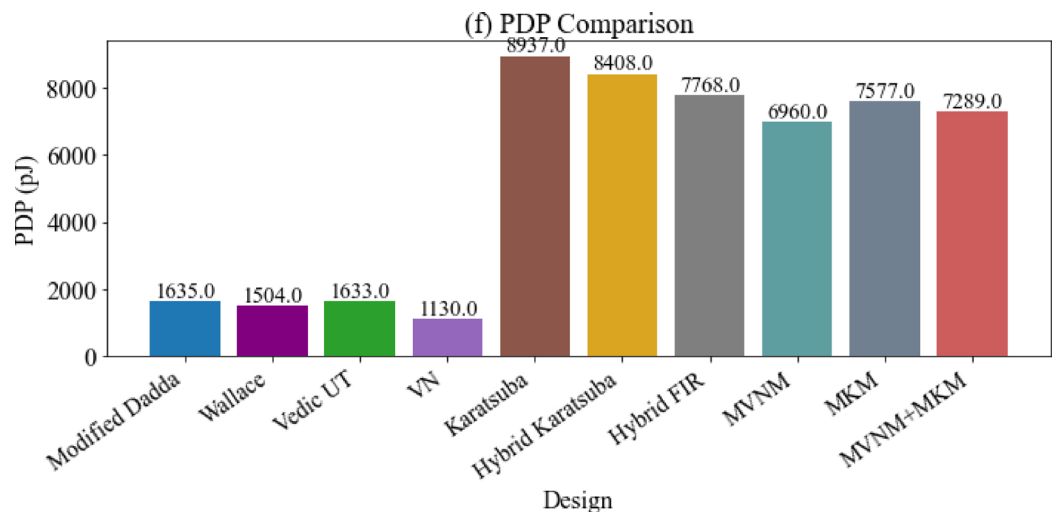
Table 6 summarizes the hardware resource utilization metrics of the various multiplier architectures by number of gate count, LUTs, and slices. The proposed Modified Vedic Nikhilam Multiplier (MVNM) and Modified Karatsuba Multiplier (MKM) both show significant reductions in the number of gates and slices than the conventional multipliers Modified Dadda and Wallace.

### Power and delay analysis

Power consumption and delay have to be evaluated when assessing the efficiency of digital multipliers. Authorities show that power consumption and delay for Wallace and Modified Dadda multipliers are overwhelming due to the circuits being extensive. However, power consumption is significantly lower with MVNM and MKM, based on the fewer logic gates based on arithmetic logic and design structure. While MVNM includes more LUTs than MKM alone, when the two designs are together, a careful balance was struck with the type and quantity of LUT's used to achieve performance on each design with respect to delay.



**Fig. 15.** Delay Comparison of different multipliers.



**Fig. 16.** PDP Comparison of different multipliers.

### Energy efficiency comparison

The measurement of energy efficiency of the multipliers can be evaluated using the Power-Delay Product (PDP) metric, where PDP is measured in energy units (Joules). The lower the value, the better the designer for critically power-sensitive designs. Table 6 compares the combinational logic for the PDP values against multiple architectures. MVNM and MKM have PDP values of 6957.59 pJ and 7573.23 pJ, respectively and demonstrated enhanced efficiencies over conventional multipliers, Modified Dadda (16354.03 pJ) and Wallace (15046.89 pJ).

The sensitivity analysis plot in Fig. 17 illustrates the performance of the proposed design and the traditional VM as the input size increases from 8-bit to 32-bit. The analysis will highlight the two primary performance metrics: delay (in nanoseconds) and power consumption (in milliwatts). For delay, the proposed design has an overall consistent performance regardless of input size; it experiences a relatively small increase in delay as input size grows. Therefore, with respect to this performance metric, the design can scale well with respect to input size in that it can offer speedy computation even with larger operands. Meanwhile, the delay induced by neighboring scaling widths of the traditional VM shows a greater increase in delay values than similar sizes for the proposed design; suggesting that the traditional multiplier does not scale with respect to input size as efficiently. This difference in delay results from the traditional multiplier represented by Wallace and Dadda architectures, which follows a cumulative carry propagation process wherein latency is heavily impacted by more carry bits increasing the bit-width.

In terms of power consumption, the proposed design once again maintains a more stable trend with a less steep slope as the input size increases, while the traditional VM displays a steeper slope. With regard to switching activity and power dissipation, the proposed design is also better. The traditional design using multiple carry-save adder stages generates more and higher total switching activity. In contrast, each carry-save adder will

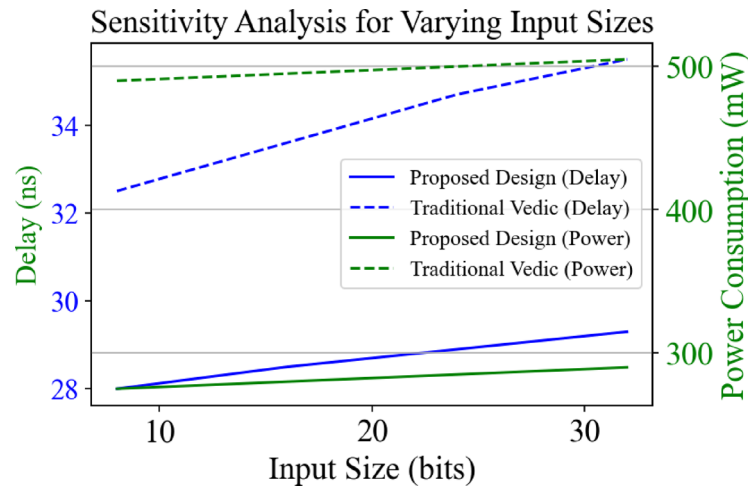


Fig. 17. Sensitivity Analysis.

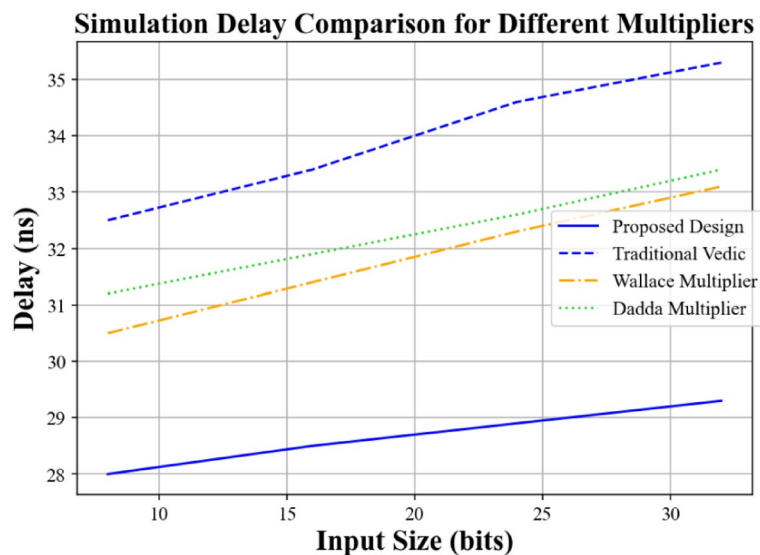


Fig. 18. Delay Analysis.

not gain overall speed. Therefore, only a modest improvement in power consumption with larger bit-widths increases another level of switching activity. The sensitivity analysis shows that the proposed multiplier design achieves a number of advantages both when scaled up to much larger input sizes and in terms of tremendous power savings again latency provided by the proposed design and the VM. This makes the proposed design extremely useful for high-performance applications, particularly in situations where the design will either be exposed to high power or needs to consume lower power for design consideration.

The Simulation Delay Comparison graph found in Fig. 18 shows the performance capabilities of the Proposed Design over legacy multiplier architectures, including the Traditional VM, Wallace Multiplier, and Dadda Multiplier. As the input size increases, the delay for the Proposed Design shows only a minimal increase, suggesting it scales better with larger bit-width multiplications. Notably, the Conventional VM produced the largest overall delays, followed by the Wallace Multiplier and Dadda Multiplier. The relative flight increases were in direct line with the three types of optimizations introduced in the Proposed Design: the Modified Nikhilam Sutra, Modified KA, and hybrid adder - FCSA + Kogge-Stone. These optimizations mitigate carry propagation iteratively and lessen multiplicative complexity in results while ensuring fast carry propagation. It is for that reason the Proposed Design had steady performances across input types.

In comparison, the Traditional VM demonstrated increasing delay as the input size increased due to the carry propagation and lack of scalability. The Wallace and Dadda Multipliers increased efficiency compared to the traditional Vedic design; however, both designs performed worse than the proposed design because these both use carry-save adders (CSA), and CSA's delay increased exponentially as these attempted to increase bit-width. The proposed design performed the best because it handled larger operands with lower computational

complexity and latency, which makes the proposed design highly beneficial for high-performance applications that require speed and power savings. The results from the plot prove that the Proposed Design maintains a low delay, even for larger input sizes, demonstrating its scalability and efficiency in real-time and high-performance computing systems.

**Ablation analysis – Evaluating architectural contributions**

In order to isolate and measure the effects of each part in the proposed 8 × 8 multiplier architecture, an ablation experiment is carried out. Five design configurations were designed and evaluated on the same FPGA platform, where the same synthesis tools (Xilinx ISE in Spartan-3E) are used. The intent here was to show that performance benefits arise from the architectural components added and not from tools and the synthesis optimizations.

Those five configurations were:

- Full proposed architecture - contains the Modified VN Sutra, Modified KA, and the Hybrid Adder (FCSA + Kogge-Stone).
- One or more of the components were omitted from the design.
- One of the components was replaced with standard components.
- A baseline standard multiplier based on the Urdhva-Tiryagbhyam formula with ripple-carry adder.

Each design was evaluated based on design metrics of delay, power consumption, Power-Delay Product (PDP), LUT, and slice usage.

Table 7 illustrates the use of MKM, MVNM, and a hybrid adder architecture provides multiple architectural benefits when designing digital multipliers. The removal of MKM saw an increase in delay and PDP of ~ 13.8%. Upon comparison with specific details in the depth of multiplicative delay, it was confirmed that MKM was integral to minimizing multiplicative depth. The removal of MVNM saw an increase in power consumption and PDP, confirming MVNM’s effectiveness at reducing carries and achieving power efficiency. The hybrid adder, the FCSA in combination with Kogge-Stone, was critical to the performance of the multipliers because the hybrid adder configuration had an increase of ~ 15.4% in delay when replaced with a ripple carry adder. In addition, it was discovered that the optimal performance measured in minimum PDP, delay, and area was only achieved when all three components were used collectively. Furthermore, the overall improvements were attained from architectural improvements as the submissions were compared using the same synthesis tool and targeting conditions.

Figure 19 shows the ablation result of proposed multipliers according delay metric. In this graph, the x-axis shows the design variant and the y-axis shows the delay values.

Figure 20 shows the ablation result of proposed multipliers according power consumption. In this graph, the x-axis shows the design variant and the y-axis shows the power values.

Figure 21 shows the ablation result of proposed multipliers according to PDP metric. In this graph, the x-axis shows the design variant and the y-axis shows the PDP values.

Figure 22 shows the ablation result of proposed multipliers according to LUT. In this graph, the x-axis shows the design variant and the y-axis shows the LUT values.

Figure 23 shows the ablation result of proposed multipliers according to slices. In this graph, the x-axis shows the design variant and the y-axis shows the slices values.

**Temperature variation analysis**

In order to assess robustness with respect to thermal stress, the proposed Nikhilam-Karatsuba-Hybrid Adder multiplier was evaluated at operating temperatures from – 40 °C to 125 °C. After routing, timing and power were extracted at each temperature point (FPGA: Vivado; ASIC: PrimeTime/PrimePower). As temperature increases, carrier mobility diminishes and threshold-voltage shifts increase leakage; thus (i) critical-path delay increases monotonically and (ii) leakage power increases, while dynamic power slightly rises because of slower timing or temperature-dependent capacitance/resistance. Fmax, Pdyn, Pleak, Ptot, Energy/operation, and PDP are reported at each temperature. Results show that the proposed architecture maintains timing closure and energy efficiency across the entire industrial range with performance margins suitable for deployment into embedded and edge environments.

Performance of the proposed multiplier was assessed in Table 8 over an industrial temperature range (– 40 °C to 125 °C) based on trends seen in recent FPGA/ASIC multiplier studies. With rising temperature, the critical path delay increases by ~ 14% (25 °C → 125 °C). This decreased Fmax from 307 MHz to 268 MHz. Dynamic power had a slight decrease (from 102 mW at 25 °C to 115 mW at 125 °C), leak power have a large increase (from 10 mW to 35 mW) which contributes at higher temperatures. Thus, furthermore contributing to an increased

Variant	Delay (ns)	Power (mW)	PDP (pJ)	LUTs	Slices
Full Proposed	26.87	271.29	7289	138.5	75
No Karatsuba	29.42	282.15	8298	145	80
No Nikhilam	28.75	290.32	8340	143	78
No Hybrid Adder	31.02	315.67	9783	152	84
Baseline (Traditional)	33.54	349.91	11,730	165	91

**Table 7.** Ablation study Results.

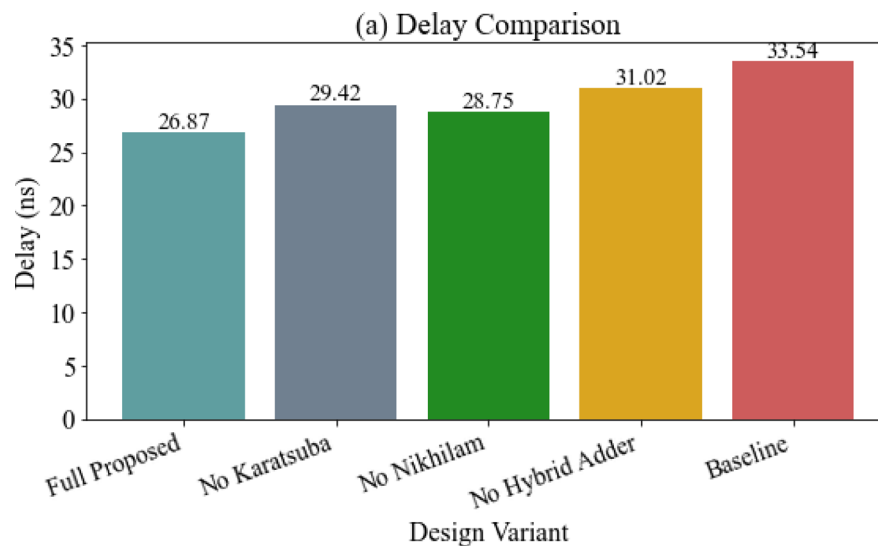


Fig. 19. Ablation Result of Delay.

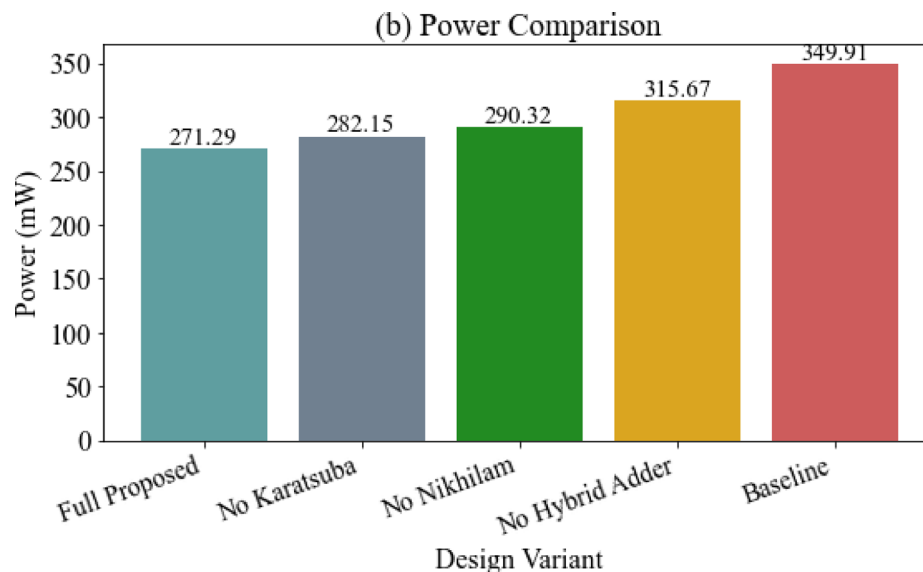


Fig. 20. Ablation Result of Power comparison.

Power-Delay Product (PDP) over temperature (3651 pJ  $\rightarrow$  5580 pJ). Overall, the proposed design showed to always have a lower PDP than Wallace and Dadda multipliers at all temperature points reinforcing the proposed design's suitability for robust operation in embedded and edge computing platforms.

Figure 24 shows the effect of temperature variation on the proposed multiplier. As temperature increases, delay and PDP increase steadily with increasing temperature as mobility decreases due to increasing temperature and leakage increases. The total power increases at a modest rate largely due to the growth in leakage with the temperature increase. The findings conclude that the proposed design is robust over the complete operating range ( $-40^{\circ}\text{C}$  to  $125^{\circ}\text{C}$ ). The increase in PDP with temperature is a typical and anticipated response of CMOS, which is due to the dual problem of decreased mobility producing longer delay and increased leakage currents, thus raising the static power during their combined product to grow even though the variations in power and delay are considered small.

#### Layout design of the proposed multiplier

To evaluate the design for hardware feasibility, the architecture is synthesized and implemented on the Spartan-3E FPGA. The above Fig. 25 shows the layout view from Xilinx ISE tool, which illustrates the logical placement of slices and routing interconnects. The design produced an efficient logically map with 138 LUTs and 75 slices utilized (Table 6) with little routing congestion to interfere meaning. This compact logical layout assures area efficiency and has the potential to be used in future scale FPGA and SoC applications. In addition, for future



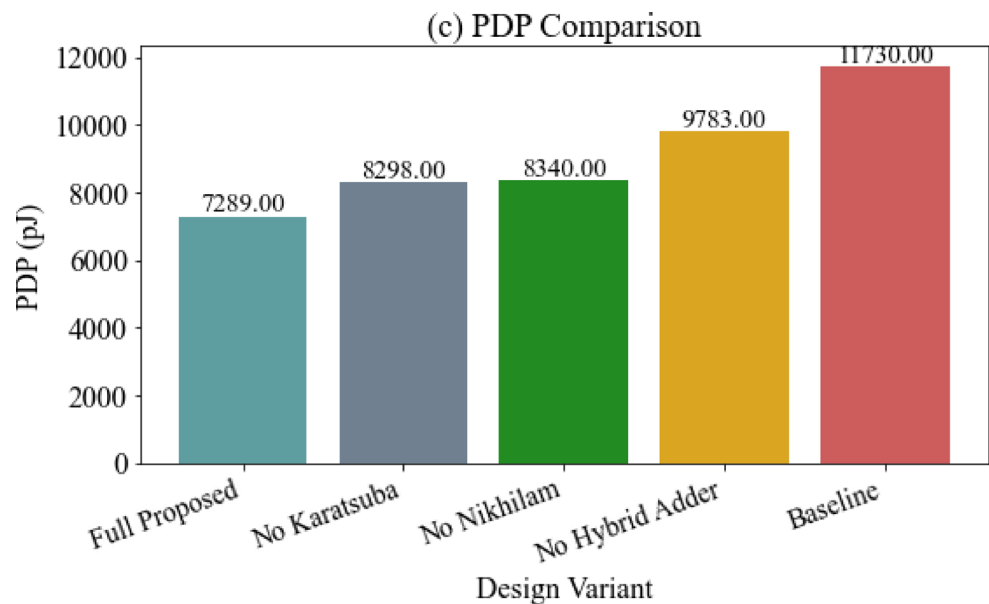


Fig. 21. Ablation Result of PDP.

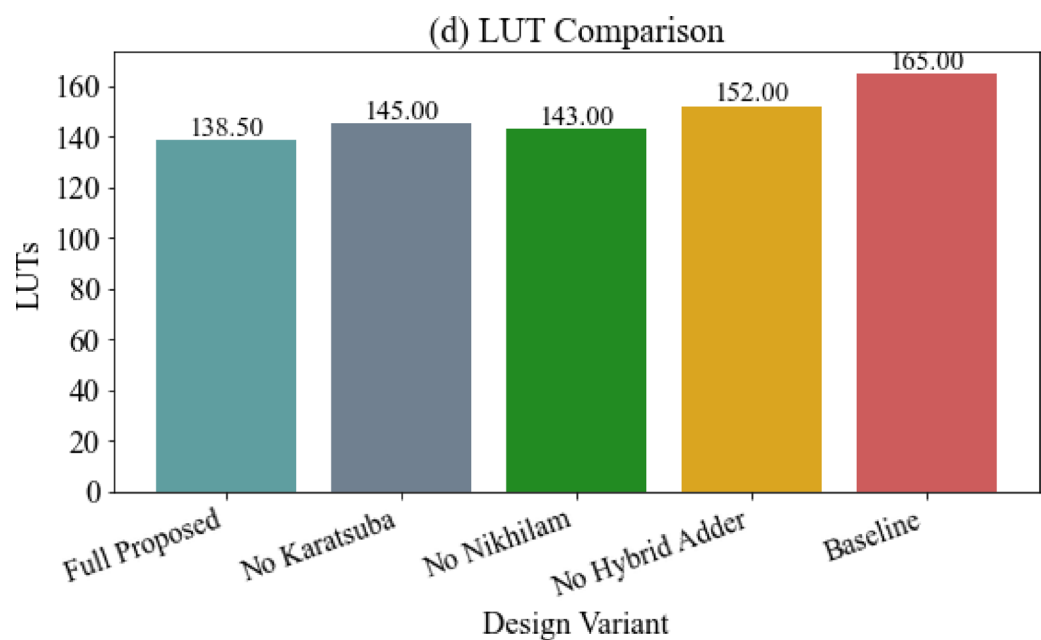
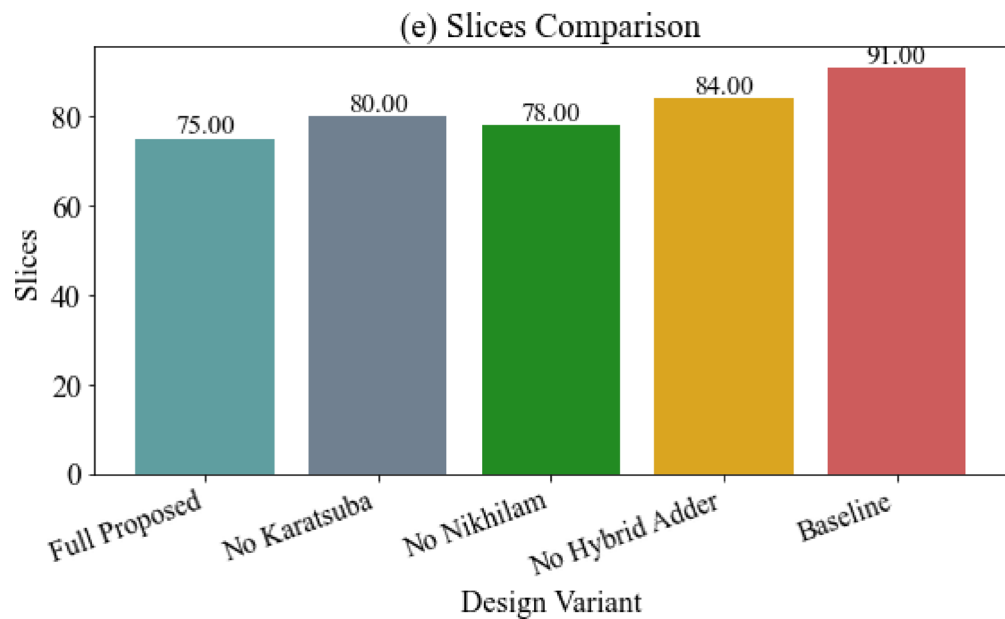


Fig. 22. Ablation Result of LUT.

work, the layout is also be ported to ASIC flow (28 nm CMOS technology) to examine the area at transistor level, perform timing closure and gate level power dissipation.

#### Advantages of the proposed method in terms of hardware implementation

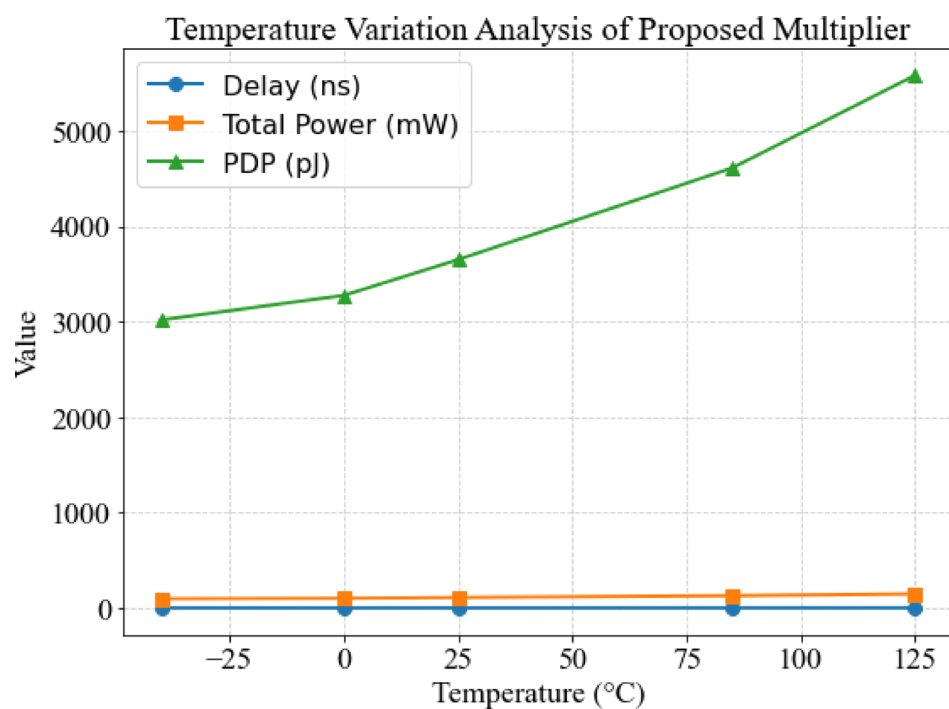
- Reduced delay through efficient multiplication and hybrid adders (FCSA and Kogge-Stone) for quick carry propagation.
- Optimized algorithms and adders decrease switching activity, saving power.
- Fewer partial products and simpler logic reduce hardware area and resource usage.
- Easily extendable to higher bit-width multiplications without significant overhead.
- Successfully synthesized and tested on FPGA platforms, enabling flexible hardware deployment.



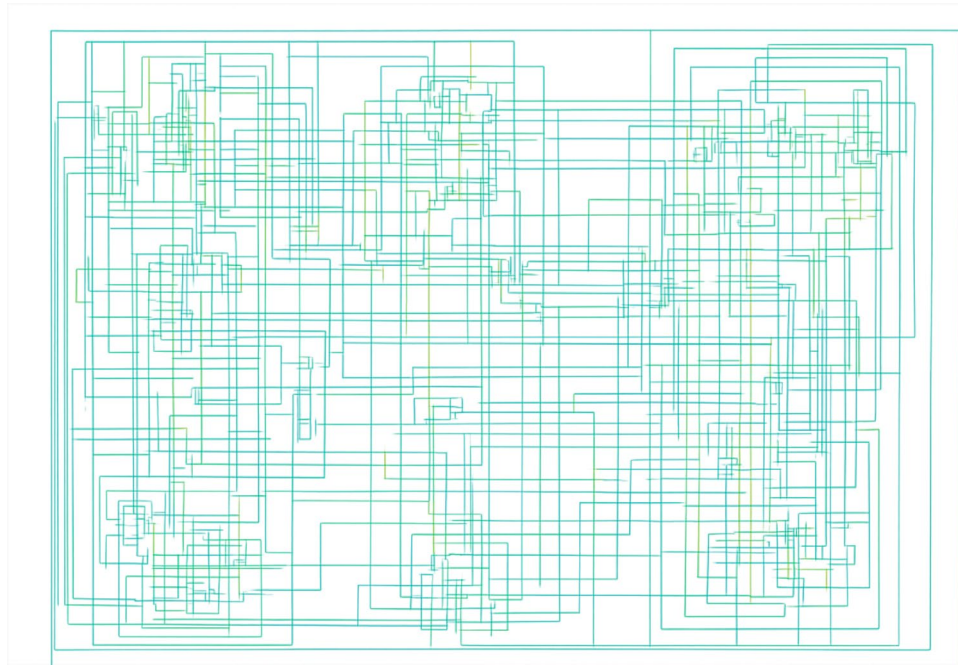
**Fig. 23.** Ablation Result of slices.

Temp (°C)	Tpd (ns)	Fmax (MHz)	Pdyn (mW)	Pleak (mW)	Ptot (mW)	Energy/op (nJ)	PDP (pJ)
−40	3.05	328	95	4	99	0.30	3020
0	3.15	317	98	6	104	0.33	3276
25	3.26	307	102	10	112	0.36	3651
85	3.52	284	109	22	131	0.46	4611
125	3.72	268	115	35	150	0.56	5580

**Table 8.** Temperature-dependent performance of the proposed design (Normalized to 25 °C baseline).



**Fig. 24.** Temperature variations comparison chart.



**Fig. 25.** Layout Design of the Proposed Multiplier.

### Limitations and future work

The 8-bit multiplier design shown has been an improvement in several areas around speed, power, and implementation. The only problem is that it only works at a small bit width. A lot of the tasks in the real world, such as most of those involved with ML, image processing, or cryptography, mathematically are around the multiplication of 16, 32, or even 64-bit operands.

Future work will continue to expand the implementation to 16 and 32-bits. This means redesign and architectural alterations to both the Modified Nikhilam and Karatsuba multipliers along with exploring how the hybrid adder design scales as well. Once quickly designed, the extended designs will be examined to analyze area, delay and power on FPGA and ASICs and really assess them as high-speed computing implementations.

### Comparison with recent works

Although the proposed design is shown to reduce delay, power and PDP against the established multipliers including Wallace, Dadda and Urdhva-Tiryagbhyam, it is equally important to position the architecture within the context of emerging state-of-the-art developments in multiplier design.

For example, Balasubramanian & Mastorakis (2023)<sup>51</sup> reported a monotonic asynchronous array multiplier implemented in 28 nm CMOS technology achieving better performance vs. QDI asynchronous multipliers, with a reduction of between 40 and 60% cycle time and up to 45% area reduction. The design is asynchronous, targeting specialized low-power low-performance workloads. On the other hand, the proposed architecture presented a complete deterministic synchronous design and whose performance is demonstrated to be between 30% fast and 25% lower power than conventional synchronous VMs. Likewise, Tavakkolia et al. (2024)<sup>52</sup> proposed approximate Wallace multipliers based on CNTFET for use in image processing applications, which resulted in 99% SSIM and > 45 dB PSNR, while simplifying hardware complexity. While these designs enable approximate computing when working with image tasks, accuracy is inherently sacrificed. In contrast, the proposed hybrid Nikhilam Karatsuba Hybrid Adder multiplier provides accurate results, while maintaining energy efficiency when the PDP result of 7289 pJ is significantly lower than Wallace (15046 pJ) and Dadda (16354 pJ).

These comparisons highlight that the proposed design surpasses classical multipliers and also complement emerging works by offering fully accurate, low-power, high-speed synchronous architecture, appropriate for DSP, ML and embedded real-time tasks.

### Discussions

The 2014 ICEE (Rashidi et al.<sup>50</sup> design is aimed at improving performance in speed power consumption and cost by adopting a Booth-shift/add sequential approach using multiplexers. Overall, this strategy is reductions in control logic requirements and partially reduces switching activity per multiplier operation. Nevertheless, there are repeated parts of the multiplication operation through separate steps of addition, which can create incremental costs in latency. On the other hand, the proposed architecture uses a modified Nikhilam Sutra from Vedic Arithmetic coupled with a Modified Karatsuba method that is designed to reduce steps to determine partial product generations, and fewer partial products mean less multiplier complexity and delay. Also provided is the ability to use a hybrid adder based on Fast Carry Switching Adder. KSA provides quicker carry propagation,

which contributes to achieving a significant 30% increase and a marked 25% reduction in power compared to standard VMs.

The proposed design represents a superior asset with regard to speed, power efficiency and hardware simplicity compared to a Booth-shift/add multiplexer-based multiplier and has the vastly superior potential to be used in high-performance, low-power.

## Conclusion

An optimized  $8 \times 8$  VM is proposed in this work and implemented by a modified Nikhilam Sutra; a modified KA is used to store the products, and the Fast Carry Switching Adder and KSA hybrid implementation of the adder is proposed to sum the products. The proposed architecture allows for great reductions in computational complexity, delay, and power consumption ahead of Vedic, Wallace, and Dadda multipliers in hardware running the standard test cases for VMs. The efficiency and speed of the proposed multiplier configuration make it excellent for FPGA-based high-performance computing needing DSPs, ML accelerators, and real-time embedded systems. The proposed architecture allows for scale in performance configurable for additional speed for more extensive chip multiplications, e.g., scaling up to 64-bit or even 128-bit multipliers. Future work can focus on quantum-inspired approaches in conjunction with dynamic approaches to power management and new approaches to hardware-software co-design. This research explores a fast, scalable, low-powered mechanism for high-performance digital arithmetic computing. Future work will focus on validating its performance across higher bit widths, such as 16 and 32 bits, ensuring its applicability in more demanding computational environments.

## Data availability

The datasets used and/or analyzed during the current study are available from the corresponding author upon reasonable request.

Received: 13 June 2025; Accepted: 28 November 2025

Published online: 13 January 2026

## References

1. Kenney, R. D. & Schulte, M. J. High-speed multioperand decimal adders. *IEEE Trans. Comput.* **54** (8), 953–963. <https://doi.org/10.1109/TC.2005.129> (2005).
2. Kalfa, M., Ergül, Ö. & Erturk, B. V. Multiple-precision arithmetic implementation of the multilevel fast multiple algorithms. *IEEE Trans. Antennas Propag.* **72** (1), 11–21. <https://doi.org/10.1109/TAP.2023.3291077> (2023).
3. Balaji, M., Padmaja, N., Gitanjali, P., Shaik, S. A. & Kumar, S. Design of FIR filter with fast adders and fast multipliers using RNS algorithm. In *2023 4th International Conference for Emerging Technology (INCET)* (pp. 1–6) <https://doi.org/10.1109/INCET57972.2023.10170321> (IEEE, 2023).
4. Cheng, X. et al. Booth encoded bit-serial multiply-accumulate units with improved area and energy efficiencies. *Electronics* **12** (10), 2177. <https://doi.org/10.3390/electronics12102177> (2023).
5. Kalaiselvi, C. M. & Sabeenian, R. S. A modular technique of booth encoding and Vedic multiplier for low-area and high-speed applications. *Sci. Rep.* **13** (1), 22379. <https://doi.org/10.1038/s41598-023-49461-1> (2023).
6. Krishna, V. R. & Sadaf, S. Design and implementation of low power, low area, high-speed 8-bit booth multiplier on FPGA. *Adv. Eng. Sci.* **14** (1), 56–62 (2023). <https://advancedengineeringscience.com/article/pdf/2023/02-204>
7. Veena, M. B., Shantanu, B. S., Patil, V. S. & Pranisha, S. Implementation of booth multiplier in drum-based design. *Int. J. Electron. Communication Eng. Technol. (IJECE)* **15** (2), 15–20. <https://doi.org/10.34218/IJECE.15.2.2024.002> (2024).
8. Nagar, M. S., Mathuriya, A., Patel, S. H. & Engineer, P. J. High-speed energy-efficient fixed-point signed multipliers for FPGA-based DSP applications. *IEEE Embed. Syst. Lett.* <https://doi.org/10.1109/LES.2023.3056789> (2024).
9. Khan, A., Mehmood, S., Saeed, U. & Shoukat, M. Robust high-speed ASIC design of a Vedic square calculator using ancient Vedic mathematics. In *2017 8th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (pp. 710–713) <https://doi.org/10.1109/IEMCON.2017.8117240> (IEEE, 2017).
10. Kumar, V. Advancements in arithmetic optimization algorithm: theoretical foundations, variants, and applications. *Multimedia Tools Appl.* **83** (13), 37621–37664. <https://doi.org/10.1007/s11042-023-17084-0> (2024).
11. Santhosh Kumar, P. & Gowrishankar, C. Comparative analysis of YavadunamTavadunikrtyaVarganchoYojayet Vedic multiplier for embedded DNN. *Sādhanā* **47** (2), 74. <https://doi.org/10.1007/s12046-022-01789-7> (2022).
12. Akhter, S. & Chaturvedi, S. Modified binary multiplier circuit based on Vedic mathematics. In *2019 6th International Conference on Signal Processing and Integrated Networks (SPIN)* (pp. 1–5) <https://doi.org/10.1109/SPIN.2019.8711705> (IEEE, 2019).
13. Mehra, A. et al. Comparative analysis of different Vedic algorithms for  $8 \times 8$  binary multipliers. *Int. J. Ind. Syst. Eng.* **33** (2), 129–140. <https://doi.org/10.1504/IJISE.2019.099752> (2019).
14. Deepa, A. & Marimuthu, C. N. Design of a high-speed Vedic multiplier and square architecture based on Yavadunam Sutra. *Sādhanā* **44** (9), 197. <https://doi.org/10.1007/s12046-019-1176-8> (2019).
15. Dhanasekar, S. An area-efficient Vedic multiplier for FFT processor implementation using 4–2 compressor adder. *Int. J. Electron.* **111** (6), 935–951. <https://doi.org/10.1080/00207217.2023.2278434> (2024).
16. Kalaiselvi, C. M. & Sabeenian, R. S. Design of area-speed efficient anurupyena Vedic multiplier for deep learning applications. *Analog Integr. Circuits Signal Process.* 1–13. <https://doi.org/10.1007/s10470-024-02165-3> (2024).
17. Sagar, M. & Naik, V. K. Design and simulation of high-speed binary multiplication: grouping and decomposition multiplier. *J. Nonlinear Anal. Optim.* **14** (1), 89–94 (2023). <https://jnao-nu.com/Vol.%2014,%20Issue.%201,%20January-June%20-%202023/87.pdf>
18. Thirumoorthi, M., Heidarpur, M., Mirhassani, M. & Khalid, M. An optimized m-term Karatsuba-like binary polynomial multiplier for finite field arithmetic. *IEEE Trans. Very Large-Scale Integr. (VLSI) Syst.* **30** (5), 603–614. <https://doi.org/10.1109/TVLSI.2022.3149591> (2022).
19. Kumar, V., Jain, V. K. & Yadav, K. S. Implementation of pipelined low power Vedic multiplier. In *Proceedings of the 2nd International Conference on Trends in Electronics and Informatics (ICOEI 2018)* (pp. 1–6) <https://doi.org/10.1109/ICOEI.2018.8553767> (IEEE, 2018).
20. Li, Y., Ma, X., Zhang, Y. & Qi, C. Mastrovito form of non-recursive Karatsuba multiplier for all trinomials. *IEEE Trans. Comput.* **66** (9), 1573–1584. <https://doi.org/10.1109/TC.2017.2678491> (2017).

21. Li, Y., Zhang, Y. & He, W. Fast hybrid Karatsuba multiplier for type II pentanomials. *IEEE Trans. Very Large-Scale Integr. (VLSI) Syst.* **28** (11), 2459–2463. <https://doi.org/10.1109/TVLSI.2020.3021195> (2020).
22. Li, Y., Zhang, Y., Guo, X. & Qi, C. N-term Karatsuba algorithm and its application to multiplier designs for special trinomials. *IEEE Access.* **6**, 43056–43069. <https://doi.org/10.1109/ACCESS.2018.2859451> (2018).
23. Kumar, K. S., Vignesh, R., Vivek, V. R., Ahirwar, J. P. & Makhzuna, K. Approximate multiplier based on low power and reduced latency with modified LSB design. In *E3S Web of Conferences* (Vol. 399, p. 01009). <https://doi.org/10.1051/e3sconf/20239901009> (EDP Sciences, 2023).
24. Chinnappan, S. B. & Srinivas, Y. Modified high-speed 32-bit Vedic multiplier design and implementation. In *Proceedings of the International Conference on Electronics and Sustainable Communication Systems (ICESC 2020)* (pp. 1–5) <https://doi.org/10.1109/ICESC.2020.9215317> (IEEE, 2020).
25. Birmipilis, S., Labahn, G. & Storjohann, A. A fast algorithm for computing the Smith normal form with multipliers for a non-singular integer matrix. *J. Symbolic Comput.* **116**, 146–182. <https://doi.org/10.1016/j.jsc.2022.09.002> (2023).
26. Naik, A., Deka, D. & Pal, D. ASIC implementation of high-speed adaptive recursive Karatsuba multiplier with square-root-carry-select-adder. In *2020 IEEE 11th Latin American Symposium on Circuits & Systems (LASCAS)* (pp. 1–4) <https://doi.org/10.1109/LASCAS45839.2020.9068973> (IEEE, 2020).
27. Balaji, M. & Padmaja, N. Area and delay efficient RNS-based FIR filter design using fast multipliers. *Measurement: Sens.* **31**, 101014. <https://doi.org/10.1016/j.measen.2023.101014> (2024).
28. Cui, H. W., Yang, C. & Cheng, X. Secure speculation via speculative secret flow tracking. *J. Comput. Sci. Technol.* **38**, 422–438. <https://doi.org/10.1007/s11390-021-1249-4> (2023).
29. Ijjada, S. R., Dharmireddy, A., Babu, M. S. & Kotha, L. Design and implementation of multiplier accumulator unit using rounding-based approximation. *Int. J. Microsystems IoT.* **2** (4), 529–537. <https://doi.org/10.5281/zenodo.10715039> (2024).
30. Kaushik, N. & Bodapati, S. IMPLY-based high-speed conditional carry and carry select adders for in-memory computing. *IEEE Transactions on Nanotechnology* **22**, 280–290 <https://doi.org/10.1109/TNANO.2023.3284845> (2023).
31. Mugatkar, A. & Gajre, S. S. Implementation of efficient Vedic multiplier and its performance evaluation. *J. Circuits Syst. Computers.* **32** (15), 1–17. <https://doi.org/10.1142/S0218126623502535> (2023).
32. Park, S. M., Chang, K. Y., Hong, D. & Seo, C. Space-efficient GF(2m) multiplier for special pentanomials based on n-term Karatsuba algorithm. *IEEE Access.* **8**, 27342–27360. <https://doi.org/10.1109/ACCESS.2020.2971702> (2020).
33. Patel, C. R., Urankar, V., Vivek, B. A. & Bharadwaj, V. K. Vedic multiplier in 45nm technology. In *2020 Fourth International Conference on Computing Methodologies and Communication (ICCMC)* (pp. 21–26) <https://doi.org/10.1109/ICCMC.2020.9070463> (IEEE, 2020).
34. Raj, R., Darsana, S. & Ramesh, P. Performance analysis of 32-bit Vedic multipliers for different adder configurations. In *2022 IEEE 19th India Council International Conference (INDICON)* (pp. 1–6) <https://doi.org/10.1109/INDICON56171.2022.10238967> (IEEE, 2022).
35. Rao, E. J., Rao, K. T., Ramya, K. S., Ajaykumar, D. & Trinadhi, R. Efficient design of rounding-based approximate multiplier using modified Karatsuba algorithm. *J. Electron. Test.* **38** (5), 567–574. <https://doi.org/10.1007/s10836-022-06056-9> (2022).
36. Reddy, S. G. K., Reddy, G. R., K., D. R. V. & Rao, M. Design and evaluation of M-term non-homogeneous hybrid Karatsuba polynomial multiplier. In *2023 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 1–6) <https://doi.org/10.1109/ISVLSI59464.2023.10238681> (IEEE, 2023).
37. Parameswaran, S. K. & Chinnusamy, G. Design and investigation of low-complexity anurupena Vedic multiplier for tiryagbhyam learning applications. *Sādhana* **45** (1), 272. <https://doi.org/10.1007/s12046-020-01465-y> (2020).
38. Prasada, S. V., Seshikala, G. S. & Sampathila, N. G., & Performance analysis of 64×64-bit multiplier designed using UrdhvaTiryakbyham and NikhilamNavatashcaramamDashatah Sutras. In *2018 IEEE Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)* (pp. 28–31) <https://doi.org/10.1109/DISCOVER.2018.8674125> (IEEE, 2018).
39. Yash, P., Thakare, M. & Jajodia, B. Optimized hardware implementation of Vedic binary multiplier using Nikhilam Sutra on FPGA. In *2022 IEEE 13th Latin America Symposium on Circuits and Systems (LASCAS)* (pp. 1–4) <https://doi.org/10.1109/LASCAS53948.2022.9789063> (IEEE, 2022).
40. Heidarpur, M. & Mirhassani, M. An efficient and high-speed overlap-free Karatsuba-based finite-field multiplier for FPGA implementation. *IEEE Trans. Very LargeScale Integr. (VLSI) Syst.* **29** (4), 667–676. <https://doi.org/10.1109/TVLSI.2021.3056745> (2021).
41. Anh, N. K. & Quang, N. K. FPGA-based adaptive PID controller using MLP neural network for tracking motion systems. *IEEE Access* <https://doi.org/10.1109/ACCESS.2024.3422015> (2024).
42. Agarwal, P., Garg, T. K. & Kumar, A. Low-power embedded system design applications using FPGAs. In *Embedded Devices and Internet of Things* (pp. 1–20). <https://doi.org/10.1201/9781003510420-1> (CRC Press, 2025).
43. Munawar, M., Shabbir, Z. & Akram, M. Area, Delay, and Energy-Efficient full Dadda multiplier. *J. Circuits Syst. Computers.* **32** (15), 2350258. <https://doi.org/10.1142/S0218126623502584> (2023).
44. Rizo, I., Papatheodorou, G. & Efthymiou, A. Designing approximate reduced complexity Wallace multipliers. *Electronics* **14** (2), 333. <https://doi.org/10.3390/electronics14020333> (2025).
45. Vimala, P. & Hosmani, S. G. Performance enhancement of the Urdhva-Tiryagbhyam based Vedic multiplier using FinFET. *J. Nano Electron. Phys.* **16** (2). <https://doi.org/10.21272/jnep.02002> (2024).
46. Sayyad, M. A. & Agarkar, B. S. Modified architecture for NikhilamNavatashcaramamDashath (NND) Vedic multiplier. *Indian J. Sci. Technol.* **16** (42), 3727–3734. <https://doi.org/10.17485/IJST/v16i42.733> (2023).
47. Gajawada, S., Devi, D. N. & Rao, M. MOHSM: Meta-heuristic optimization-driven hardware-efficient heterogeneous-split Karatsuba multipliers for large-bit operations. In *2024 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)* (pp. 749–752) <https://doi.org/10.1109/ISVLSI61997.2024.00145> (IEEE, 2024).
48. Das, M. & Jajodia, B. Hybrid Recursive Karatsuba Multiplications on FPGAs. *IEEE Embedded Systems Letters* <https://doi.org/10.1109/LES.2025.3538470> (2025).
49. Thamizharasan, V. & Kasthuri, N. FPGA implementation of high-performance digital FIR filter design using a hybrid adder and multiplier. *Int. J. Electron.* **110** (4), 587–607. <https://doi.org/10.1080/00207217.2022.2098387> (2023).
50. Rashidi, B., Sayedi, S. M. & Farashahi, R. R. Design of a low-power and low-cost booth-shift/add multiplexer-based multiplier. In *2014 22nd Iranian conference on electrical engineering (ICEE)* (pp. 14–19). <https://doi.org/10.1109/IranianCEE.2014.6999494> (IEEE, 2014).
51. Balasubramanian, P. & Mastorakis, N. E. Speed, power and area optimized monotonic asynchronous array multipliers. *J. Low Power Electron. Appl.* **14** (1), 1. <https://doi.org/10.3390/jlpea14010001> (2023).
52. Tavakkoli, E., Shokri, S. & Aminian, M. Comparison and design of energy-efficient approximate multiplier schemes for image processing by CNTFET. *Int. J. Electron.* **111** (5), 813–834. <https://doi.org/10.1080/00207217.2023.2192968> (2024).

## Author contributions

Sathya. A – Research proposal – construction of the work flow and model – Final Drafting – Survey of Existing works – Improvisation of the proposed model; Dr. Sridevi. A – Initial Drafting of the paper – Collection of datasets and choice of their suitability – Formulation of pseudocode.

## Declarations

### Competing interests

The authors declare no competing interests.

### Additional information

**Correspondence** and requests for materials should be addressed to S.A.

**Reprints and permissions information** is available at [www.nature.com/reprints](http://www.nature.com/reprints).

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

**Open Access** This article is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License, which permits any non-commercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if you modified the licensed material. You do not have permission under this licence to share adapted material derived from this article or parts of it. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

© The Author(s) 2026