INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY

RESEARCH ARTICLE

Modified Architecture for Nikhilam Navatshcaramam Dashath (NND) Vedic Multiplier

M A Sayyad[1]*, B S Agarkar[1]

1 Department of Electronics and Telecommunication Engineering, Sanjivani College of Engineering, Kopargaon, Maharashtra, India

OPEN ACCESS

Received: 29-03-2023

Accepted: 29-09-2023

Published: 12-11-2023

Citation: Sayyad MA, Agarkar BS (2023) Modified Architecture for Nikhilam Navatshcaramam Dashath (NND) Vedic Multiplier. Indian Journal of Science and Technology 16(42): 3727-3734. https://doi.org/10.17485/IJST/v16i42.733

* **Corresponding author**.

sayyadma@yahoo.com

Funding: None

Competing Interests: None

Published By Indian Society for Education and Environment (iSee)

ISSN
Print: 0974-6846
Electronic: 0974-5645

Abstract

Objectives: Speed of multiplication in Digital Signal Processing (DSP) applications plays an important role in generating the result quickly. There is scope for reducing the propagation delay in multiplication by designing the multiplier circuit based on the Vedic mathematics (formulas) sutras. This study aims to design the multiplier circuit based on Nikhilam Navatshcaramam Dashath (NND) of Vedic mathematics for improvement in speed, power, and area. **Methods:** The multiplier circuit based on NND method is designed for the multiplier and multiplicand less than as well as greater than the nearest base in the binary number system. The architecture is designed for both. The proposed multiplier is implemented with VHDL on Vertex-7, Device: XC7VX485T Package: FFG1157, FPGA board using Xilinx ISE 14.7, and its power dissipation is calculated using XPower analyzer. The performance of the proposed multiplier is compared with the conventional array multiplier, Vedic multiplier and also compared with the architecture reported in the literature**. Findings:** In the proposed architecture n bit multiplier and multiplicand are divided into n/2 bits, two parts, and processed through n/2 bit multiplier and n bit adder. This method converted the n bit multiplication into n/2 bit multiplication and n bit addition. The proposed architecture is efficient in terms of area, delay and power as compared to the array and Vedic Urdhva Tiryakbyham (UT) multiplier. The 4 bit NND multiplier is 26.72 % delay and 47.05 % area efficient as compared to the reported architecture[(1)]. To compare with other reported architecture, the results are also taken on Artix-7, Device: XC7A100T Package: CSG324, and Spartan-6, Device: XC6SLX4 Package: TQG144 FPGA. The results demonstrate an improvement in processing speed as well as power consumption. This method is a special case of multiplication and is efficient if the multiplicand and multiplier are close to the base value. In this implementation, the multiplier and multiplicand are split into two parts and processed; hence, the algorithm will give the correct result for a specific range of multipliers and multiplicands. The accuracy analysis of the proposed multiplier is also performed for multipliers and multiplicands far away from the base value. The maximum error is 7.94%. **Novelty:** The architecture used in this system converts n-bit multiplication into n bit addition and

n/2 bit multiplication, which can be used for the numbers below the base value. For values greater than the base value, on which very little work has been documented in the literature, the NND multiplier is implemented in this study.

Keywords: Array Multiplier; Vedic Multiplier; Urdhva Tiryakbyham(UT) Multiplier; Nikhilam Navatshcaramam Dashath (NND); Filed Programmable Gate Array (FPGA)

1 Introduction

A high speed processor depends greatly on the multiplier and adder as they are the basic building blocks of the ALU. Most DSP processors use the multiplier block. Hence the speed of the multiplier is a key factor. Vedic multipliers provide the solution to getting the output at a faster speed. There are 16 main sutras with 13 upasutras, among these UrdhvaTiryakbyham (UT) and Nikhilam Navatashcaramam Dashatah (NND) are most widely used by researchers. The performance of these multipliers can be verified by implementing them on reconfigurable hardware with the help of hardware description language (HDL). 16-bit multiplication [(2)] with UrdhvaTiryakbyham method of Vedic multiplier is implemented. In UT multiplier the generated cross products are to be added to get the result. There is significant improvement in delay by using different methods to add these partial products. Use of Increment by one (IBO) and carry save adder(CSA) [(3)], carry select adder [(4)], and ripple carry adder [(5)] for DSP applications such as FIR filter is successfully implemented for improvement in delay. The method of arithmetic adder [(6)] and multiplexer based half adder with modified full adder [(7)] is used for improvement in delay of Vedic UT multiplier. The performance of Vedic multiplier [(8)] and divider along with convolution in DSP [(9)] is implemented on reconfigurable hardware. The method of Nikhilam Navatashcaramam Dashatah [(10)] with the algorithm and flow chart for software implementations are proposed. This method uses NND multiplier with base as 2^n and 2^{n-1} and recommends the use of Nikhilam sutra. The NND multiplier, UrdhvaTiryakbyham and Sampoornam Vedic multipliers are successfully implemented on Artix-7 FPGA [(11)]. MAC unit is designed with Nikhilam Navatashcaramam Dashatah multiplier for improvement in speed of multiplication [(12)]. The use of DEMUX based adders [(13)] instead of conventional adder for implementation of Vedic multiplier resulted in improvement of power consumption as well as delay. The UT Vedic multiplier with Quaternary Signed Digit (QSD) [(14)] is implemented, this is best suited for DSP operation such as Fast Fourier Transform, Convolution and Filtering, etc. Palak Yesh et. al [(1)] have proposed the architecture of NND multiplier. This architecture uses the two's complement, multiplier, binary adder and subtractor and left shifter blocks. The two's complement block is used to find the deficit form the base value. The designed NND multiplier is implemented on Artix-7 FPGA and the results are compared with other reported architecture. This paper aims to propose the architecture for the NND multiplier. The proposed architecture uses n/2 bit multiplier and n bit adder for its implementation. This resulted in improvements in area (in terms of number of lookup tables (LUTs), delay as well as power consumption. Most of work reported in the literature is related to the multiplication for numbers smaller than the base value. In this work the architecture for the multiplication based on NND multiplier for the numbers greater than the base value is also designed.

2 Methodology

The multiplier is based on the ancient Indian Vedic Mathematics formula Nikhilam Navatashcaramam Dashatah (all from 9 and last from 10). The NND Sutra is a unique multiplication formula that can be used when the multiplier

and multiplicand are close to the base value. It is required to get the shortfalls (deficits) of the multiplier and multiplicand form the base value. The Anurupyena Sutra [15] is implemented for machine learning applications on Cyclone V FPGA using intel Quartus 17.0 software. The architecture proposed in [16,17] uses the adder, two's complement, n bit multiplier and mux for the implementation of Yavadunam Sutra.

In this work the architecture is developed for Nikhilam Navatashcaramam Dashatah multiplier. Two architectures are proposed as follows.

- For the multiplier and multiplicand smaller than base value, here the multiplier and multiplicand are considered to be presented by 4,8 and 16 bits and the base value is 2^n where n is the number of bits.
- For the multiplier and multiplicand greater than base value represented by 5, 9 and 17 bits considering the base value as 2^n where n is the number of bits minus one.

2.1 Nikhilam Navatashcaramam Dashatah Multiplier

2.1.1 Decimal Number System

The multiplier is based on an algorithm Nikhilam Navatashcaramam Dashatah (all from 9 and last from 10) of ancient Indian Vedic Mathematics. NND Sutra is a special multiplication formula applicable to the cases of multiplication where the multiplier and multiplicand are near the base value. The base for the decimal number system is 10^n. Where, n is the number of digits. The following example shows NND applied for decimal numbers.

Example 1. 91× 93 =
For both numbers, the nearest base is 100 and the numbers less than the base value
91 -9 (difference from base)
× 93 -7 (difference from base)
The lower two digits of result will be -9 × -7 = 63
The upper two digits will be 91 + (-7) = 84 or 93 + (-9) = 84
The result of multiplication is 91× 93 = 8463
Example 2. 981× 973 =
For both numbers, the nearest base is 1000 and the numbers less than the base value
981 -19 (difference from base)
× 973 -27 (difference from base)
The lower three digits of result will be -19 × -27 = 513
The upper three digits will be 981 + (-27) = 954 or 973 + (-19) = 954
The result of multiplication is 981× 973 = 954513
Example 3. 106× 115 =
For both numbers, the nearest base is 100 and the numbers are larger than the base value
106 6 (difference from base)
× 115 15 (difference from base)
The lower two digits of result will be 6 × 15 = 90
The upper digits will be 106 + 15 = 121 or 115 + 6 =121
The result of multiplication is 106× 115 = 12190
Example 4. 1018× 1025 =
For both numbers, the nearest base is 1000 and the numbers are larger than the base value
1018 18 (difference from base)
× 1025 25 (difference from base)
The lower three digits of result will be 18 × 25 = 450
The upper digits will be 1018 + 25 = 1043 or 1025 +18 = 1043
The result of multiplication is 1018 × 1025 = 1043450

2.1.2 Binary Number System

In the binary number system base is 2^n. Where n is the number of bits in the multiplier or multiplicand. The method of NND will be the same as that of the decimal number system. The following examples 5 and 6 will illustrate 4 bit and 8 bit multiplications and examples 7 and 8 will illustrate 5 bit and 9 bit multiplications.

Example 5. 1101 × 1110 =
For both numbers, the nearest base is 2^4 = 16 and the numbers are smaller than the base value.

1101 - 0011 (difference from base)
× 1110 - 0010 (difference from base)
The lower half digits of result will be -11 × -10 = 0110
The upper half digits will be 1101 + (-0010) = 1011 or 1110 + (-0011) = 1011
The result of multiplication is 1101× 1110 = 10110110
Example 6. 11110011× 11111010 =
For both numbers, the nearest base is 2^8 = 256 and the numbers are smaller than the base value.
11110011 -1101 (difference from base)
× 11111010 -0110 (difference from base)
The lower three digits of result will be -1101 × -0110 = 01001110
The upper three digits will be 11110011 + (-0110) = 11101101 or 11111010 + (-1101) = 11101101
The result of multiplication is 11110011× 11111010 = 1110110101001110
Example 7. 10010× 10011 =
For both numbers, the nearest base is 2^4 = 16 and the numbers are larger than the base value.
10010 10 (difference from base)
× 10001 01 (difference from base)
The lower half digits of result will be 10 × 01 = 0010
The upper half digits will be 10010 + 01 = 10011 or 10001 + 10 = 10011
The result of multiplication is 10010× 10001 = 100110010
Example 8. 100001011 × 100000101 =
For both numbers, the nearest base is 2^8 = 256 and the numbers are larger than the base value.
100001011 1011 (difference from base)
× 100000101 0101 (difference from base)
The lower half digits of result will be 1011 × 0101 = 00110111
The upper half digits will be 100001011 + 0101 = 100010000 or 100000101 + 1011 = 100010000
The result of multiplication is 100001011 × 100000101 = 10001000000110111

2.2 Proposed System for Binary multiplication using NND

The implementation of NND is divided in two parts

2.2.1 When the multiplier and multiplicand are smaller than the base value

The base value for the n-bit number will be 2^n. The numbers which are less than 2^n are represented with the help of n bits.

Consider the multiplication of two numbers x and y smaller than the base value 2^n, the difference from the base value is represented by a and b, which is given by

$$a = (2^n - x) \tag{1}$$

$$b = (2^n - y) \tag{2}$$

The multiplication result of x and y is given as follows

$$xy = (2^n - a)(2^n - b)$$

$$xy = 2^n 2^n - 2^n (a + b) + ab$$

$$xy = 2^n (2^n - (a + b)) + ab \tag{3}$$

The equation 3 represents the result of multiplication. This represents the two parts of result, the upper half (n bits $[2n-1 \ : \ n]$) and lower half (n bits$[n-1 \ : \ 0]$) of the result. The proposed system calculates the lower half of result and upper half of

result using the architecture shown in figure 1. The upper half of the result of multiplication is calculated using n-bit adder and neglecting the carry generated. The upper half of the result is the common difference. It is to be derived as multiplier – (difference of multiplicand from base). To find the lower part of the result we have to multiply the difference of multiplier and multiplicand from the base. To get the difference from the base take the 2's complements of the lower half bits of multiplier and multiplicand as shown in Figure 1.

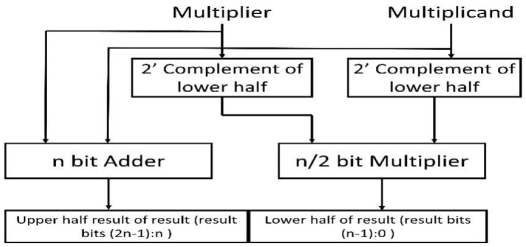

Fig 1. n bit NND Multiplier

2.2.2 When the multiplier and multiplicand are larger than the base value

Similar architecture is proposed for numbers larger than base value. The number of bits for this case will be $n+1$. The multiplier is $n+1$ bit multiplier.

Consider the multiplication of two numbers p and q greater than the base value 2^n, the difference from the base value m and n which is given by

$$p = (2^n + m)$$

$$q = (2^n + n)$$

The multiplication result of x and y is given as follows

$$pq = (2^n + m)(2^n + n)$$

$$pq = 2^n 2^n + 2^n(m+n) + mn$$

$$pq = 2^n(2^n + (m+n)) + mn \quad (4)$$

The equation 4 represents the result of multiplication. This represents the two parts of result the upper half ($n+1$ bits $[2n : n]$) and lower half (n bits$[n-1 : 0]$) of the result. The proposed system calculates the lower half of result and upper half of result using the architecture shown in Figure 2.

3 Results and Discussion

Synthesis of the proposed NND multiplier is done using Xilinx ISE 14.7 with the Family: Vertex 7, Device: XC7VX485T Package: FFG1157 FPGA. The implementation results related to the number of 4 input LUTs and delay are presented for 4, 8 and 16 bit multipliers in Table 1. For calculation of power dissipation, the XPower analyzer tool is used, and the switching activity data can be written at the time of simulation after place and route (PAR). This switching activity data file is used by the XPower analyzer to calculate total power. The power calculations will help to optimize the power during design. In our design, the simulation inputs are kept the same for array, Vedic UT multiplier and NND multiplier so that there should be the same effect of dynamic power on the circuits. The total power dissipation of all the designed multipliers for 4,8 and 16 bit are shown in Table 1. All the designs are simulated using Xilinx ISIM simulator. The simulation output of 16 bit NND multiplier is shown in Figure 3.

It is considered that the number of LUT is equivalent to the area. The results of implementation show that the proposed 16 bit Vedic NND multiplier is 53.9% and 70.0% area, 71.1% and 37.9 % delay efficient as compared to array and Vedic UT multipliers respectively. The power of proposed NND multiplier is also improved by 7.14%.

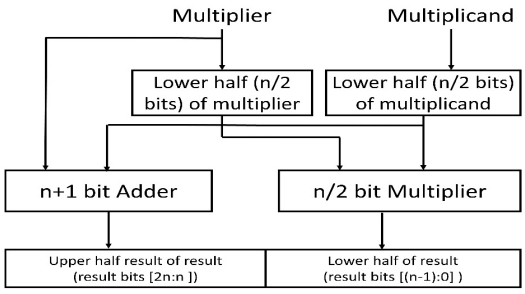

Fig 2. $n+1$ bit NND Multiplier

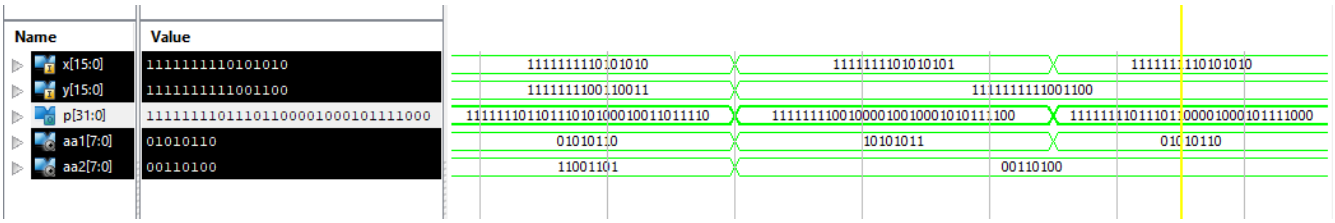

Fig 3. Simulation of16 bit NND multiplier

Table 1. Comparison of proposed NND multiplier with array and UT multiplier with Vertex7 FPGA implementation

Implementation method	LUTs	Delay(ns) Total	Delay(ns) Logic	Delay(ns) Route	Power(W)
A] 4-bit multiplier					
Array multiplier	23	3.527	0.258	3.270	0.003
Vedic UT multiplier	26	3.101	0.215	2.886	0.003
Proposed NND multiplier 4 bits	9	1.209	0.086	1.123	0.003
Proposed NND multiplier 5 bits	9	1.173	0.086	1.087	0.003
B] 8-bit multiplier					
Array multiplier	93	9.444	0.731	8.713	0.007
Vedic UT multiplier	128	5.813	0.473	5.340	0.007
Proposed NND multiplier 8 bits	42	3.261	0.215	3.046	0.006
Proposed NND multiplier 9 bits	37	3.118	0.215	2.903	0.005
C] 16-bit multiplier					
Array multiplier	365	20.339	1.591	18.748	0.014
Vedic UT multiplier	570	9.471	0.86	8.611	0.014
Proposed NND multiplier 16 bits	168	5.874	0.473	5.401	0.013
Proposed NND multiplier 17 bits	152	5.74	0.473	5.267	0.012

To have a comparison with the reported architecture, the proposed NND architecture is also implemented on Artix-7, Device: XC7A100T, Package: CSG324 FPGA and Sparten-6, Device: XC6SLX4, Package: TQG144 FPGA. The results of the comparison are shown in Table 2.

The proposed 4 and 8 bit multipliers are 26.72 % and 16.16% delay efficient as compared to reported architecture [(1)] respectively. With the implementation on Artix-7 the proposed design of 4 bit multiplier is 28.43% efficient in comparison with [(1)] and 78.7% efficient than [(11)].

The 8 bit NND multiplier implemented on Spartan-6 is 14.4 % delay and 38.23% area efficient than reported architecture [(12)].

3.1 Accuracy analysis of proposed NND multiplier

The proposed n bit NND multiplier uses the adder to get the upper half of the results and n/2 bit multiplier to get the lower half of the results. Here it is expected that the product of the difference from the base value (denoted by 'a' and 'b' in equation-1 and

Table 2. Comparison of proposed NND multiplier with multiplier architecture reported

FPGA : Family: Vertex 7, Device : XC7VX485T Package: FFG1157		
	Parameters	
Implementation method	**Number of LUTs**	**Delay(ns) Total**
4 bit NND multiplier [1]	17	1.65
4 bit Proposed NND multiplier	9	1.209
8 bit NND multiplier [1]	42	3.89
8 bit Proposed NND multiplier	42	3.261

Table 3. FPGA : Family: Artix 7, Device : XC7A100T Package: CSG324

	Parameters	
Implementation method	**Number of LUTs**	**Delay(ns) Total**
4 bit NND multiplier [1]	17	2.05
4 bit NND multiplier [11]	34	5.749
4 bit Proposed NND multiplier	9	1.467
8 bit NND multiplier [1]	42	4.58
8 bit Proposed NND multiplier	42	4.202

Table 4. FPGA : Family: Sparten 6, Device : XC6SLX4 Package: TQG144

	Parameters	
Implementation method	**Number of LUTs**	**Delay(ns) Total**
8 bit NND multiplier [12]	68	12.51
8 bit Proposed NND multiplier	42	10.707

equation-2) of the multiplicand and multiplier should be 8 bits, then the accuracy will be 100%. It is required to perform the accuracy analysis of the n bit NND multiplier so that an error can be found for the multiplicand and multiplier away from the base value. Table 5 represents the accuracy of the8 bit NND multiplier.

Table 5. Accuracy analysis of proposed8 bit NND multiplier

Sr. No.	**Multiplier x in binary**	**Multiplier x in decimal**	**Multiplicand y in binary**	**Multiplicand y in decimal**	**Average % error in result**
1	11110000 to 11111111	240 to 255	11110000 to 11111111	240 to 255	0.00
2	11110000 to 11111111	240 to 255	11100000 to 11101111	224 to 239	0.27
3	11110000 to 11111111	240 to 255	11010000 to 11011111	208 to 223	0.43
4	11110000 to 11111111	240 to 255	11000000 to 11001111	192 to 207	0.88
5	11100000 to 11101111	224 to 239	11110000 to 11111111	240 to 255	0.27
6	11100000 to 11101111	224 to 239	11100000 to 11101111	224 to 239	1.03
7	11100000 to 11101111	224 to 239	11010000 to 11011111	208 to 223	1.90
8	11100000 to 11101111	224 to 239	11000000 to 11001111	192 to 207	2.91
9	11010000 to 11011111	208 to 223	11110000 to 11111111	240 to 255	0.43
10	11010000 to 11011111	208 to 223	11100000 to 11101111	224 to 239	1.90
11	11010000 to 11011111	208 to 223	11010000 to 11011111	208 to 223	3.45
12	11010000 to 11011111	208 to 223	11000000 to 11001111	192 to 207	5.24
13	11000000 to 11001111	192 to 207	11110000 to 11111111	240 to 255	0.88
14	11000000 to 11001111	192 to 207	11100000 to 11101111	224 to 239	2.91
15	11000000 to 11001111	192 to 207	11010000 to 11011111	208 to 223	5.24
16	11000000 to 11001111	192 to 207	11000000 to 11001111	192 to 207	7.94

Table 5 shows that the error is 0% for multiplicand and multiplier values between 240 and 255. The accuracy is 100% for the input range of 240 to 255 since the multiplication (n/2 bit multiplier) result of difference from the base is of 8 bits. The

inaccuracy also grows as the multiplier and multiplicand move farther from the base number. On average, Table 5 contains 25% of all potential multiplication combinations. The proposed 8 bit NND multiplier's maximum error is 7.94%. In comparison to an array and a Vedic UT multiplier, the proposed 8 bit NND multiplier is 54.83%, 67.18% efficient in area and 65.47%, 43.90% efficient in delay, respectively.

4 Conclusion

The proposed architecture employed one multiplier and one adder in comparison to the reported architecture, which removed the binary shifter and binary subtractor by splitting the given binary number into two halves. This resulted an improvement in performance parameters. When implemented on Vertex-7, the proposed 4 and 8 bit multipliers are 26.72 % and 16.16% delay efficient as compared to reported architecture [(1)] respectively. The proposed 17 bit NND multiplier is 58.3%, 73.3 % area and 71.77%, 39.39% delay efficient as compared to array and Vedic UT multipliers respectively. The proposed multiplier is efficient in terms of area, propagation delay and power. The NND multiplier has limitation that it is applicable for special cases of multiplication. The proposed multiplier is very effective if the multiplier and multiplicand have values near to the base value which is 2^n where the n represents the number of bits in multiplier and multiplicand. The accuracy analysis of the NND multiplier must be done in the case where the numbers are far away from the base value. Still, there are applications where the probability of multiplier and multiplicand are close to the base, then the proposed multiplier will be the best choice for implementation.

References

1) 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 System (LASCAS). IEEE. 2022;p. 1–4. Available from: https://doi.org/10.1109/LASCAS53948.2022.9789063.
2) Chowdary KKS, Mourya K, Teja SR, Babu GS, Priya SSS. Design of Efficient 16-bit Vedic Multiplier. In: 2021 3rd International Conference on Signal Processing and Communication (ICPSC). IEEE. 2021;p. 214–218. Available from: https://doi.org/10.1109/ICSPC51351.2021.9451757.
3) Akhter S, Chaturvedi S. Modified Binary Multiplier Circuit Based on Vedic Mathematics. In: 2019 6th International Conference on Signal Processing and Integrated Networks (SPIN). IEEE. 2019;p. 234–237. Available from: https://doi.org/10.1109/SPIN.2019.8711583.
4) Yaswanth D, Nagaraj S, Vijeth RV. Design and analysis of high speed and low area vedic multiplier using carry select adder. In: 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE). IEEE. 2020;p. 1–5. Available from: https://doi.org/10.1109/ic-ETITE47903.2020.369.
5) Samyuktha S, Chaitanya DL. VLSI design of efficient FIR filters using Vedic Mathematics and Ripple Carry Adder. *Materials Today: Proceedings*. 2020;33:4828–4832. Available from: https://doi.org/10.1016/j.matpr.2020.08.391.
6) Kumari S, Sharma K. Implementation of Nobel Vedic Multiplier Using Arithmetic Adder. In: Jacob, J I, Shanmugam K, S, Bestak, R, editors. Data Intelligence and Cognitive Informatics. Springer Nature Singapore. 2022;p. 209–216. Available from: https://doi.org/10.1007/978-981-16-6460-1_15.
7) Kumar U, Sindhuri NB, Subbalakshmi K, Kiranmayi U, P. Performance Evaluation of Vedic Multiplier Using Multiplexer-Based Adders. *Lecture Notes in Electrical Engineering*. 2019;521. Available from: https://doi.org/10.1007/978-981-13-1906-8_36.
8) Pasuluri BS, Sonti VJKK. Performance Analysis of 8-Bit Vedic Multipliers Using HDL Programming. In: Kumar, A, Paprzycki, M, Gunjan, V, editors. Lecture Notes in Electrical Engineering;vol. 601. Springer Singapore. 2020;p. 1036–1046. Available from: https://doi.org/10.1007/978-981-15-1420-3_114.
9) Biji R, Savani V. Performance analysis of Vedic mathematics algorithms on re-configurable hardware platform. *Sādhanā*. 2021;46(2). Available from: https://doi.org/10.1007/s12046-021-01605-4.
10) Sona M, Somasundaram V. Vedic Multiplier Implementation in VLSI. *Materials Today: Proceedings*. 2020;24. Available from: https://doi.org/10.1016/j.matpr.2020.03.748.
11) Yuvaraj M, Kailath BJ, Bhaskhar N. Design of optimized MAC unit using integrated vedic multiplier. In: 2017 International conference on Microelectronic Devices, Circuits and Systems (ICMDCS). IEEE. 2017;p. 1–6. Available from: https://doi.org/10.1109/ICMDCS.2017.8211704.
12) Naregal K, Hebbar PK, Chandu Y. Design and implementation of high efficiency vedic binary multiplier circuit based on squaring circuits. In: 2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT). IEEE. 2017;p. 973–977. Available from: https://doi.org/10.1109/RTEICT.2017.8256743.
13) Kumar BY, Kharwar S, Singh S, Mohammed MKA, Dauwed M. Design and FPGA Implementation of Matrix Multiplier Using DEMUX-RCA-Based Vedic Multiplier. In: Lecture Notes in Networks and Systems;vol. 573. Springer International Publishing. 2023;p. 216–224. Available from: https://doi.org/10.1007/978-3-031-20429-6_21.
14) Rani D, Govindarajulu. Implementation of Modified Vedic Multiplier using On Quaternary Signed Digit Number System. *Journal of Engineering Sciences*;13(12):111–122. Available from: https://www.jespublication.com/upload/2022-V13I12013.pdf.
15) Parameswaran SK, Chinnusamy G. Design and investigation of low-complexity Anurupyena Vedic multiplier for machine learning applications. *Sādhanā*. 2020;45(1):272. Available from: https://doi.org/10.1007/s12046-020-01500-4.
16) Deepa A, Marimuthu CN. Design of a high speed Vedic multiplier and square architecture based on Yavadunam Sutra. *Sādhanā*. 2019;44(9). Available from: https://doi.org/10.1007/s12046-019-1180-3.
17) Deepa A, Marimuthu CN, Murugesan C. An efficient high speed squaring and multiplier architecture using yavadunam sutra and bit reduction technique. *Journal of Physics: Conference Series*. 2020;1432(1):012080. Available from: https://doi.org/10.1088/1742-6596/1432/1/012080.