# Design of an efficient vedic binary squaring circuit

*Nithyashree S*
*Electronics and Communication Engineering*
RV College of Engineering
Bangalore, India
nshree1596@gmail.com

*Chandu Y*
*Electronics and Communication Engineering*
PES Institute of Technology
Bangalore, India
ychandu1101@gmail.com

***Abstract*— High speed and less area have always been a major concern in VLSI design. With this as a constraint, in this paper a dedicated architecture which is exclusively used for squaring operation has been proposed. Squaring plays a vital role in many signal processing applications and probabilistic analysis in communication systems, where, quite often general multipliers are used although squaring has to be done. This unnecessarily increases the area of the design and also increases the computation time. The principles of Nikhilam Sutra have been leveraged and is extended for squaring binary bits. A comparative study of the projected architecture and the prevailing multipliers, on the premise of delay and space utilization is presented. The simulation outcome proves that the design projected employing Nikhilam sutra improves the performance significantly. The 8-bit architecture has been developed by Verilog HDL and the synthesis is completed using Xilinx ISE – 14.5 software.**

**Keywords— Vedic mathematics; squaring; Nikhilam sutra; Area; Delay.**

## I. Introduction

Multiplication is one of the most important and frequently used operations in any digital system [1]. Consequently, the complete efficiency of the system is governed by the performance of the multipliers used. Considering high speed, less area and low power consumption as the major design targets, many high-performance designs are being proposed [1][2][3]. Various types of multipliers like Array multiplier, Wallace multiplier, Bypassing multiplier, Modified booth multiplier and Vedic multipliers were witnessed during the evolution of the multipliers [4]. Many technical papers are being published which substantiate that the Vedic multipliers outperform the other existing types.

However, in several signal processing applications involving statistical and probabilistic analysis, oftentimes there is a need to evaluate the square of the amplitudes of incoming signals, for example, Computation of energy of the signal needs the calculation of its variance, that is found by squaring the signal amplitude. In addition, squaring operation is also used in Image processing, Adaptive filtering, Decoding, Least mean squaring, Computation of Euclidean distance among pixels in graphic processors, transformation from rectangular to polar coordinate etc., [5]. As squaring is considered to be a particular case of multiplication, general multipliers are conventionally being used to compute the squares. The most naïve method of multiplication of two n bit numbers is using classical or long multiplication which exhibits a complexity of $O(n^2)$. To improvise the $O(n^2)$ bound of multiplication, several Vedic principles can be used [6].

A novel dedicated circuit for squaring the base-2 numbers is presented in this paper, and its efficiency is compared with the present multipliers on the premise of space utilization and operation speed. The proposed architecture uses Nikhilam Sutra for the said purpose. Outline of the paper is as follows.

Section II discusses the literature survey, section III discusses the method for computing the square using Nikhilam Sutra. Section IV covers the design of the architecture. Section V demonstrates the results obtained in the simulation and shows a comparative study while Section VI presents the conclusion and future scope of the work done.

## II. Literature Survey

With the prominent impact that the Vedic principles have continued to create in the field of computation, their use is not just limited to mental math, but it is exhaustively being used in the world of electronics. Various works have been published in this genre. The following gives an insight into the same:

SV Mogre and DG Bhalke have proposed the implementation of high-speed matrix multiplication on FPGA using "Urdhava Trigyagbhyam" Sutra. This design shows the significant reduction in Area/Speed Ratio, increases the running frequency of the multiplier and also makes the system energy efficient [1].

Kunal Jadhav, Aditya Vibhute et al., have suggested Vedic principles as a solution to increase the speed of the Arithmetic Logic Unit (ALU), which was designed by employing reversibility in the circuits, in order to reduce power dissipation the cost of reduced speed. Thereby coming up with a power efficient and faster ALU [3].

Inspired by the working of the human brain, Anshika, Yamuna et al., have designed a Vedic neuron, using "Urdhava Trigyagbhyam" Sutra, which outperforms the standard neurons in terms of speed and energy efficiency [10].

Premananda B.S., Samarth S. Pai, et al., have designed an 8-bit Vedic multiplier using Urdhva Tiryagbhyam sutra and carry skip addition technique. Results prove that the performance of their design is better than the conventional multipliers [8].

Through the survey, it is observed that a general multiplier is used even in applications where squaring is exclusively done. This needlessly requires higher area and large computation time. This subject is addressed in the paper and a dedicated squaring unit is proposed whose design incorporates one of the Vedic sutras.

## III. Procedure of Nikhilam sutra based squaring

"Vedic Mathematics" implies a way of calculation supported by a collection of sixteen Sutras, or maxims and their Upa-sutras or outcomes obtained from these Sutras [7]. These concepts were being used as timesavers in cerebral calculation for resolving complex questions. It is enthralling to examine the use of those early techniques in recent advancements to provide optimized results. Out of the sixteen sutras, Urdhva Triyagbhyam (UT) Sanskrit literature (sutra) and Nikhilam Sanskrit literature are the commonly used techiques in processor technology. However, UT sutra is highly fitted for multiplication of two numbers while Nikhilam sutra is best applicable when both the multiplier and multiplicand are nearer to the same base. In case of squaring a number, inevitably since the multiplier is same as the multiplicand, both will be closer to the same base. Therefore, in the proposed design Nikhilam sutra is used.

The algorithm for Nikhilam sutra is illustrated in two stages. Firstly, it is described for base-10 number system and later its manifestation in base-2 number system is demonstrated.

### *A. Method for base-10 number system*

The algorithm is explained by considering an example. Taking 24 as input, its square is computed along these lines.

- Establish the closest base for the considered input. Multiples of ten are thought-about as bases in the base-10 number system. 24 is near to 20, therefore,

  Base = 20.

- Difference of the input and the base has to be found. It is calculated by deducting the base from the input.

  Difference = 24 -20 = 4.

- Add the difference obtained in the previous step to the given input. Let the result of this step be 'L'. Therefore, we get
  L = 24 + 4 =28.

- Division of the base by 10 yields a number, say k.
  k= 20/10 = 2

  **NOTE:** In general, if the base is of N digits, then, the base is divided by $10^{(N-1)}$. In the example considered N = 2. Therefore, base is divided by $10^{(2-1)}$ = 10.

- The left part of the result is got by multiplying the outcomes of the previous two steps. Therefore,
  Left part = L * k
  = 28 * 2 = 56.

- Right part of the outcome is obtained by squaring the difference. The number of digits in the right portion after squaring the difference, must be equal to that of the zeros in the base. If $(\text{difference})^2$ results in higher number of digits, i.e., greater than that of zeros in the base, then the extra numerals will be considered as carry to the left part of the result. On the other hand, if it is lesser, essential number of zeros will be headed.

| | |
|---|---|
| $\text{Difference}^2$ | $= 4^2 = 16$ |
| Right part | = 6 |
| Carry | = 1 |

- If at all a carry is created in the previous step, it gets added to the left part. The final value is got by joining the left and right portions obtained earlier.

  Result = (Left part + carry ) || Right part
  = (56 + 1) || 6
  = 576

  Hence, we have found $24^2 = 576$.

### *B. Method for Base-2 number system*

Calculating the square of a base-2 number with the help of Nikhilam sutra takes a repetitious method. The algorithmic program is shown for a four-bit number; nevertheless, the same method can be extrapolated for any bit length.

Taking the example of an input to be P = $(1101)_2$, the following steps are followed to get the square of P.

- Deduct the closest base from the input to calculate the first difference(A). In the base-2 number system, powers of 2 are taken as the bases.
  A = $(1101)_2 - (1000)_2 = (101)_2$

- The consecutive differences (B, C and so on) are found till the final bit arrives. Understandably, the consecutive differences are found by removal of the Most significant bit (MSB) from the preceding difference.

TABLE I
4- BIT BINARY SQUARING OF $(1011)_2$

| | Input | First Difference | Next Difference | Last difference |
|---|---|---|---|---|
| | $P = 1101_2$ | $A = P[2:0] = 101_2$ | $B = A[1:0]= 01_2$ | $C =B[0]= 1_2$ |
| **S0** | | | | $1_2$ |
| **S1** | | | $S0 = 1_2$ (B = C) | |
| **S2** | | $(101_2 + 1_2)*100_2 + S1= 11001_2$ | | |
| **S3** | $(1101_2 + 101_2)*1000_2 + S2= 10101001_2$ | | | |
| **Result** | $10101001_2$ | | | |

$B = A[1:0] = (01)_2$; Removing the MSB of A gives B;
$C = B[0] = (1)_2$ ;Removing the MSB of B gives C;

C is the final difference since the last bit is reached

•

i) Retain the Least significant bit (LSB) of the input as the LSB of the result (S0).

$S0 = C$
$= (1)_2$

ii) As per the rule, if any two successive differences are equal, then the partial product at that specific stage is same as the preceding one.
Hence,

$S1 = S0 = (1)_2$ ; Since B and C are equal.

iii) Now, add the next two successive differences, (A and B) left shift the result by 2 bits and add it to the partial product found in the preceding step i.e., S1, to obtain the next partial product S2.

$$\begin{aligned} S2 &= (A+B)*(100)_2 + S1 \\ &= ((101_2 + 1_2)* (100_2)) + 1_2 \\ &= (11001)_2 \end{aligned}$$

iv) When the successive differences are not equal, then the partial product of that particular stage is calculated as in the previous step. But the number of shifts will rise by 1 bit with every successive stage.

Therefore, in this step three-bit left shifting is done to get S3.

$$\begin{aligned} S3 &= (P+A) * (1000)_2 + S2 \\ &= ((1101_2 + 101_2)* (1000)_2) +11001_2 \\ &= (10101001)_2 \end{aligned}$$

At this stage, the process ends because the successive differences in calculating S3 are the input (P) and the first difference (A). Hence the final result of squaring P is S3 = $(10101001)_2$. This process is shown systematically in TableI [6].

S0, S1, S2 and S3 denote the partial products. Generalizing the process, for a K bit input, K repetitions should be conducted to find the square. Thereby, the complexity is given as O(n).

## IV. DESIGN OF NIKHILAM SQUARING CIRCUIT

The architecture of a K-bit squaring unit is implemented using adders/subtractors, comparators, shifters and 2:1 Multiplexers (Mux).

The Nikhilam Sutra for squaring a binary number uses a repetitive process as described in the previous section. Thus, the squaring architecture for 2-bit number is described in detail this paper, however, it can be extrapolated for base-2 numbers of any bit length. The abstract architecture of 8-bit squaring circuit is also presented in this section.

For a 2-bit (a[1:0]) input, initially, the first difference is found. The consecutive differences are obtained logically by removing the MSB. Thus, the first difference in this example is a[0]. As the Least Significant Bit of the input is encountered, the process stops. In case of longer inputs, the consecutive differences will be computed until LSB of the input is reached and the differences computed will be saved in different register units. The LSB of the final outcome (S[0]) is retained to be the last difference (a[0]). The comparator unit is employed to compare the final difference (a[0]) with its preceding difference (a[1:0]). The comparator generates a 1-bit output which identifies if its inputs are the same or otherwise; this result is used as a control line for the mux. Intermediate value S[1] is calculated by adding the inputs of the comparator, shifting the outcome left by one bit and adding this shifted output to S[0]. Based on the result of the comparator unit, the mux output is selected between S[0] and S[1], for equal and unequal output of the comparator output, respectively, to obtain the final result. The process followed in implementing a 2-bit squaring unit is depicted in Fig. 1.

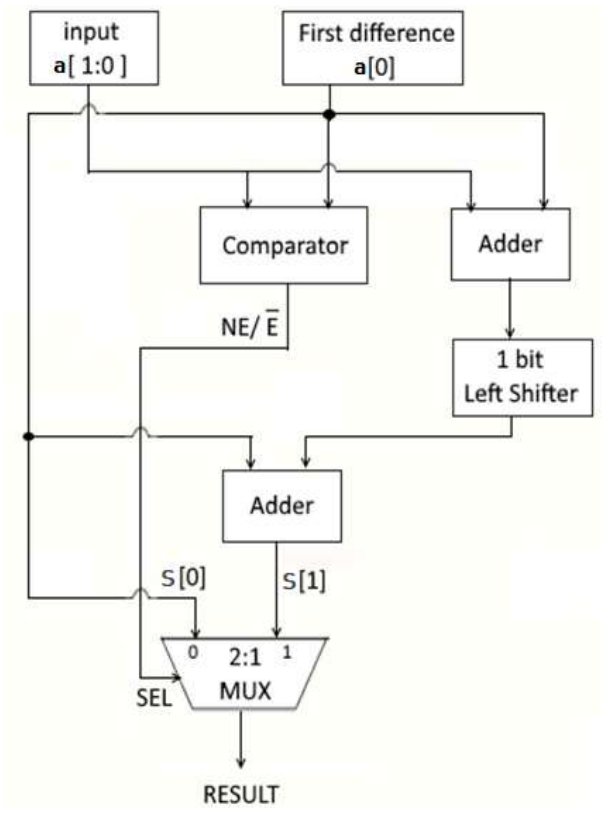

Figure 1: Proposed Architecture for 2-bit input

Further, considering the 2-bit architecture as a primitive unit, the architecture for the 8-bit squaring unit is obtained as shown in Fig. 2.

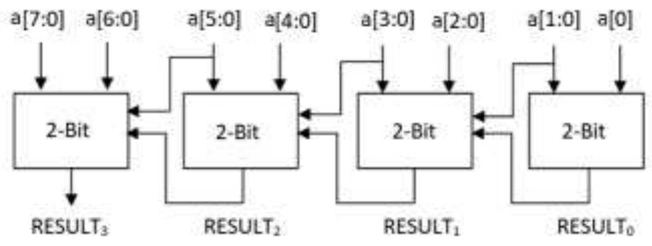

Figure 2: Architecture for 8-bit input

Extending the 2-bit squaring unit, the 8-bit squaring unit is obtained. The successive differences are given as inputs to the blocks and the partial products are calculated at each stage that are used by the succeeding blocks to evaluate the final result.

## V. Execution and Results

Verilog hardware description language (HDL) is used to design the projected binary squaring unit which uses the principles of Nikhilam Sutra. The design is implemented for 8-bit input. Xilinx ISim simulator and Xilinx XST are used to simulate and synthesize the design respectively for package TQG144 FPGA, XC6SLX4 device with -3 speed that belongs to Spartan 6 family. The functionality is verified for various input cases. Verilog HDL test bench is used to create the inputs. The simulation outcome of 8-bit squaring unit is depicted in Fig 3.

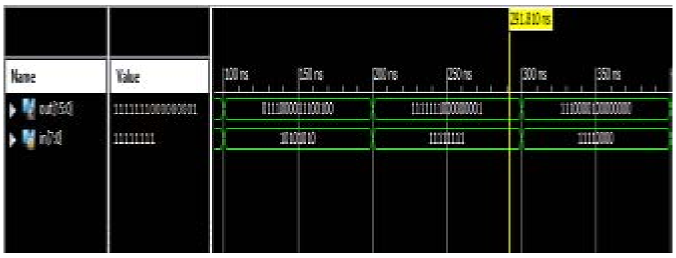

Figure 3: Results for various 8-bit input combination

|  |  |  |  |
|---|---|---|---|
| Case 1: | in [7:0] | = | 1010_1010 |
|  | out[15:0] | = | 0111_0000_1110_0100 |
| Case 2: | in [7:0] | = | 1111_1111 |
|  | out[15:0] | = | 1111_1110_0000_0001 |
| Case 3: | in [7:0] | = | 1111_0000 |
|  | out[15:0] | = | 1110_0001_0000_0000 |

Table II represents the HDL synthesis result of the 8-bit binary squaring circuit. Fig. 4 shows the technological view for the proposed architecture.

TABLE II
HDL SYNTHESIS RESULT OF THE PROPOSED DESIGN

| Hardware | Number used |
|---|---|
| Adders/Subtractors | 14 |
| Comparators | 7 |
| 2:1 Multiplexers | 7 |

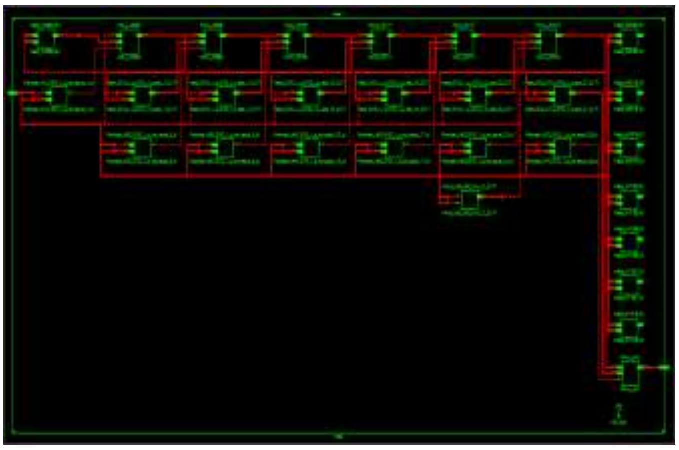

Figure 4: Technological view of the 8-bit Squaring unit

In general, for an N-bit input, the number of hardware units used are summarized as follows:

|  |  |
|---|---|
| No. of Adders/Subtractors | = 2*(N-1) |
| No. of Comparators | = (N-1) |
| No. of 2:1 Multiplexers | = (N-1) |

A comparative study of the performance parameters like speed and space utilization of the proposed design and the existing design is presented. The resulting analysis is shown in Table III

TABLE III
COMPARISON OF PERFORMANCE PARAMETERS: SPEED AND AREA

| **Parameter** | **Existing Vedic multipliers (8*8)** | **Proposed Nikhilam 8-bit unit** | **Percentage change** |
|---|---|---|---|
| Worst case delay(ns) | 17.430 [8] | 12.514 | 28.2% decrease |
| No. of bonded IOBs | 32 | 24 | 25% decrease |
| No. of slice LUTs | 225 out of 9312 [2] | 68 out of 2400 | 69.8% decrease |

It is now a demonstrated truth from the literature review that the Vedic multipliers are the most effective type of multipliers [4]. Hence, a comparative study is made with the current Vedic multipliers and the projected architecture. A significant contrast can be understood between the projected design and the current one, on the premise of space utilization and speed. This is because Nikhilam sutra uses lesser number of iterations for squaring than the other multiplication techniques. Also, the number of input-output buffers (IOBs) in the proposed squaring circuit is lesser by 8. This is because the squaring circuit uses a single input (8-bit) and generates the output (16-bit) while the general multipliers use two inputs (8+8 bits) resulting in higher area. In the proposed circuit, there is reduction in area and increase in the speed. Therefore, its speed/space-utilization proportion is considerably high when compared to the present multipliers.

## VI. CONCLUSION AND FUTURE SCOPE

As illustrated in the results, the Nikhilam sutra - squaring circuit lowers the space utilized by approximately 70% and improves the operation speed by around 28%. Therefore, it can be considered as a better choice in place of a general multiplier, where the applications solely demand squaring. The proposed design can be extended as a general multiplier as well, because multiplication of two numbers can be achieved through squaring, as shown by the equation 1 [6].

$$x*y = \frac{(x + y)^2 - (x - y)^2}{4} \qquad (1)$$

To further improve the efficiency of the proposed design, the adder used in the architecture can be replaced by fast Adders or Subtractors like Kogge stone adder [9]. Thus, increasing the speed of operation.

## REFERENCES

[1] Ms.S.V.Mogre, Mr.D.G.Bhalke , "*Implementation of High Speed Matrix Multiplier using Vedic Mathematics on FPGA*", International Conference on Computing Communication Control and Automation 2015, pp. 959- 963.

[2] Kapil Ram Gavali, Poonam Kadam," *VLSI Design of High Speed Vedic Multiplier for FPGA Implementation*", 2nd IEEE International Conference on Engineering and Technology (ICETECH), March 2016.

[3] Kunal Jadhav, Aditya Vibhute, Shyam Iyer, Asst. Prof. R. Dhanabal, "*Novel Vedic Mathematics Based ALU Using Application Specific Reversibility*", IEEE Sponsored 9th International Conference on Intelligent Systems and Control (ISCO) 2015.

[4] Khuraijam Nelson Singh , H. Tarunkumar, *"A Review on Various Multipliers Designs in VLSI",* IEEE INDICON 2015.

[5] Medimi Rani, SD.NageenaParveen, "*An FPGA Implementation of Low Power Square and Cube Architectures using Nikhilam Sutra*" International Journal & Magazine of Engineering, Technology, Management and Research,Volume No:2,Issue No:12, pp. 1748-1756, December 2015.

[6] Shri Prakash Dwivedi, "*An Efficient Multiplication Algorithm Using Nikhilam Method*", Fifth International Conference on Advances in Recent Technologies in Communication and Computing (ARTCom 2013), 10 Jul 2013.

[7] Jagadguru Swami Sri Bharati Krisna Tirthaji Maharaja, *"Vedic Mathematics: Sixteen Simple Mathematical Formulae from the Veda*," Motilal Banarasidas Publishers, Delhi, 2009, pp. 5-45.

[8] Premananda B.S. , Samarth S. Pai, Shashank B, Shashank S. Bhat," *Design and Implementation of 8-Bit Vedic Multiplier*",International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, Vol. 2, Issue 12, pp. 5877-5882, December 2013.

[9] Anjana.R, Abishna.B, Harshitha.M.S, Abhishek.E, Ravichandra.V, Dr. Suma M S, "*Implementation of Vedic Multiplier using Kogge-Stone Adder",* International Conference on Embedded Systems - (ICES 2014), pp. 28- 31, 2014.

[10] Anshika, Yamuna SV, Nidhi Goel, S.Indu, *" Neuronal Logic Gates Realization using Vedic Mathematics",* 1st International Conference on Next Generation Computing Technologies (NGCT-2015), pp.450-455.