Novel High speed Vedic Multiplier proposal incorporating Adder based on Quaternary Signed Digit number system

Preyesh Dalmia, Vikas, Abhinav Parashar, Akshi Tomar and Dr. Neeta Pandey
Dept. of Electronics and Communication Engineering
Delhi Technological University (formerly Delhi College of Engineering)
New Delhi, India
{preyeshdalmia, vikas.dce2016, abhinavparashar.1810, akshitomar274}@gmail.com, neetapandey@dce.ac.in

Abstract— **This paper presents a high-speed Vedic multiplier based on the Urdhva Tiryagbhyam sutra of Vedic mathematics that incorporates a novel adder based on Quaternary Signed digit number system. Three operations are inherent in multiplication: partial products generation, partial products reduction and addition. A fast adder architecture therefore greatly enhances the speed of the overall process. A Quaternary logic adder architecture is proposed that works on a hybrid of binary and quaternary number systems. A given binary string is first divided into quaternary digits of 2 bits each followed by parallel addition reducing the carry propagation delay. The design doesn't require a radix conversion module as the sum is directly generated in binary using the novel concept of an adjusting bit. The proposed multiplier design is compared with a Vedic multiplier based on multi voltage or multi value logic [MVL], Vedic Multiplier that incorporates a QSD adder with a conversion module for quaternary to binary conversion, Vedic multiplier that uses Carry Select Adder and a commonly used fast multiplication mechanism such as Booth multiplier. All these designs have been developed using Verilog HDL and synthesized by Synopsys Design Compiler. They have been realized using the open source NAN gate 15nm technology library. The proposal shows a maximum of 88.75% speed improvement with respect to Multi Value logic based 128x128 Vedic multiplier while the minimum is 17.47%.**

Keywords-Multiplier; Quaternary Signed Digit adder [QSD]; Urdhva Tiryagbhyam; Vedic Mathematics

I. Introduction

One of the primary features that help us determine the computational power of a processor is the speed of its arithmetic unit. An important function of an arithmetic block is multiplication because, in most mathematical computations, it forms the bulk of the execution time. Thus, the development of a fast multiplier has been a key research area for a long time.

Some of the important algorithms proposed for fast multiplication in literature are Array, Booth and Wallace multipliers [1]-[5]. Vedic Mathematics [6, 7] is a methodology of arithmetic rules that allows for more efficient implementations regarding speed. Multiplication in this methodology consists of three steps: generation of partial products, reduction of partial products, and finally carry-propagate addition. Multiplier design based on Vedic mathematics has many advantages as the partial products and sums are generated in one step, which reduces the carry propagation from LSB to MSB. This feature helps in scaling the design for larger inputs without proportionally increasing the propagation delay as all smaller blocks of the design work concurrently. References [8], [9] and [11] compared Vedic Multiplier with other multiplier architectures namely Booth, Array and Wallace on the basis of delay and power consumption. Vedic multiplier showed improvements in both the parameters over other architectures. Thus, many implementations of multiplication algorithms based on Vedic sutras have been reported in literature [10]-[12]. Vedic multiplier schemes proposed in literature are based on Urdhva Tiryagbhyam and Nikhilam sutras of Vedic Mathematics. As Nikhilam sutra is only efficient for inputs that are close to the power of 10, in this paper a design to perform high-speed multiplication based on the Urdhva Tiryagbhyam sutra of Vedic Mathematics which is generalized method for all numbers, has been presented.

The final step, carry-propagate addition, requires a fast adder scheme because it forms a part of the critical path. A variety of adder schemes have been proposed in literature to optimize the performance of Vedic multiplier [13]. Adder based on QSD shows an improvement in speed over other state of the art adders [14, 15]. Earlier implementations of QSD adder were based on Multi Voltage or Multi Value Logic (MVL) [16]. The difficulty in application of quaternary addition outside MVL (Multi Voltage logic) is that, the adder is only a small unit of the design whose outputs will needed to be converted back to binary for further processing. However, use of a conversion module undermines the advantages gained in speed by using QSD. In this paper, a novel implementation of an adder based on QSD is proposed, which reduces the carry propagation delay in the design by making use of carry free arithmetic. The proposed adder design works on a hybrid of binary and quaternary number systems wherein the sum is directly generated in binary using the concept of an adjusting bit, eliminating the conversion module. The design can be scaled to larger bit implementations such as 32, 64, 128 or more with minimal increase in propagation delay owing to the parallelism prevalent in the design. We have compared our design with a Vedic multiplier based on MVL logic that uses a ripple carry adder [16], Vedic Multiplier that incorporates a QSD adder and a conversion module for quaternary to binary conversion, Vedic multiplier that uses state of the art fast adder scheme such as Carry select adder [17] and a commonly used fast multiplication mechanism such as Booth multiplier [18], to prove the feasibility of our design across important comparison points.

This paper is organized as follows. Section II describes the Basic Terminology associated with our design. Section III describes the Proposed Multiplier architecture based on Vedic multiplication and Quaternary addition. Section IV comprises of Result in which device utilization summary and computational path delay obtained for the proposed Vedic multiplier (after synthesis) is discussed and Section V consists of Conclusion.

II. Basic Terminology

A. *Urdhva Tiryagbhyam (UT) Sutra*

The UT sutra is an ancient Vedic Mathematics sutra that can be used for multiplication of two numbers in any number system. It is based on "Vertical and Crosswise" multiplication. A 2x2 multiplier based on UT sutra is depicted in Fig. 1 and Fig. 2, where X and Y represent inputs while Z corresponds to output. Stepwise procedure is outlined below.

Step1: Vertical Multiplication: The least significant digits of the multiplicand and the multiplier are multiplied, as in (1).

$$Z0=X0.Y0 \tag{1}$$

Step2: Crosswise Multiplication and Addition: Z1, in (2), is obtained by cross multiplying X1 and Y0, and Y1 and X0 and subsequently adding the two products. In this stage a carry C1, as in (3), might be generated, that is propagated to the next step.

$$Z1= (X0.Y1) \oplus (X1.Y0) \tag{2}$$

$$C1=X0.X1.Y0.Y1 \tag{3}$$

Step3: Vertical Multiplication and Addition: The most significant digits of the multiplicand and the multiplier are multiplied, and the product is added with the carry of the previous step to obtain Z3 and Z2, as in (4) and (5) respectively.

$$Z2= (X1.Y1) \oplus C1 \tag{4}$$

$$Z3= X0.X1.Y0.Y1 \tag{5}$$

The final result is concatenation of Z3, Z2, Z1 and Z0.

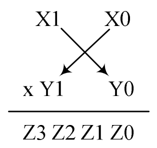


Fig. 1. Vertical and Crosswise multiplication

The logic circuit for 2x2 UT multiplier is shown Fig. 2.

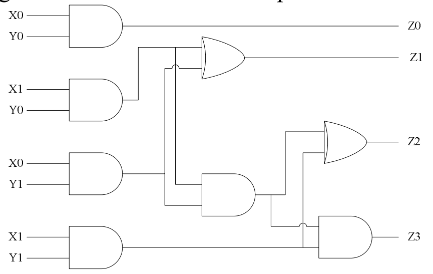


Fig. 2. 2x2 UT multiplier

B. *Quaternary Signed Digit (QSD number system)*

The QSD is a radix-4 number system that provides the benefit of faster arithmetic calculations over binary computation, as it eliminates rippling of carry during addition. Every number in QSD can be represented using digits from the set {-3,-2,-1, 0, 1, 2, 3}. Being a higher radix number system it utilizes less number of gates and hence saves on time and reduces circuit complexity. The stages involved in addition of two numbers in QSD are:

Stage1: Generation of intermediate carry and sum: When two digits are added in QSD number system, the resulting sum ranges between -6 to +6. Numbers with magnitude higher than 3 are represented by multiple digits with least significant digit representing sum and the next digit corresponds to carry. Also, every number in QSD can have multiple representations [14, 15]. The representation is chosen such that the magnitude of sum digit is 2 or less than 2 and the magnitude of carry digit is 1 or less than 1, the reason for which is explained in the next stage.

Stage2: The intermediate sum and carry have a limit fixed on their magnitude because this allows carry free addition in the second step. The result can be obtained directly by adding the sum digit with the carry of the lower significant digit [14, 15].

III. Proposed design

A. *4x4 Multiplier*

Block diagram of a 4x4 multiplier is shown in Fig. 3. In this multiplier, four 2x2 multipliers are arranged systematically. Each multiplier accepts four input bits; two bits from multiplicand and other two bits from multiplier. Addition of partial products is done using two four bit Quaternary adders, a two-bit adder and a half adder. The final result is obtained by concatenating the least significant two bits of the first multiplier, four sum bits of the second four-bit Quaternary adder and the sum bits of two-bit adder.

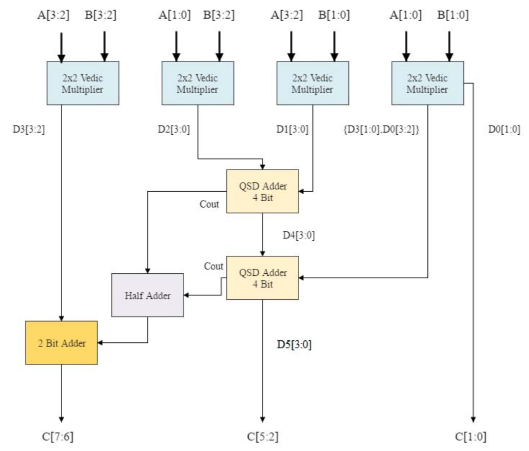


Fig. 3. Proposed 4x4 Multiplier

Table I shows all intermediate and final results involved in the multiplication process of two binary numbers, A = $(1111)_2$ and B = $(1001)_2$.

The data flow in the proposed 4x4 multiplier is given below:

1) A[1:0] and B[1:0], A[3:2] and B[1:0], A[1:0] and B[3:2], and A[3:2] and B[3:2] are multiplied by 2x2 Vedic multipliers, giving output D0[3:0], D1[3:0], D2[3:0] and D3[3:0] respectively.

2) D1 [3:0] and D2[3:0] are added by the proposed 4 bit QSD adder, giving D4[3:0] and a carry out as the outputs.

3) D4[3:0] and {D3[1:0], D0[3:2]} are added by the second 4 bit QSD adder, giving D5[3:0] and a carry out as the outputs.

4) The half adder is used to add the carry outs of the QSD adders. The output obtained is fed to the 2 Bit Adder along with D3[3:2].

5) The result, C, in binary is obtained by concatenation of output of 2 Bit Adder, D5[3:0] and D0[1:0].

The proposed design can be extended to multiply both negative and positive integers by an addition of a sign bit in both inputs. An XOR logic can then be used to compute the sign bit of the final output. The multiplication of the magnitudes will proceed simultaneously in a similar manner to the example described above.

TABLE I. MULTIPLICATION RESULT OF TWO 4 BIT BINARY NUMBERS USING THE PROPOSED DESIGN

	Binary equivalent	**Decimal equivalent**	**Explanation**
A	$(1111)_2$	15	Input 1
B	$(1001)_2$	9	Input 2
D0	$(0011)_2$	3	Output of 2x2 Vedic Multiplier 1
D1	$(0011)_2$	3	Output of 2x2 Vedic Multiplier 2
D2	$(0110)_2$	6	Output of 2x2 Vedic Multiplier 3
D3	$(0110)_2$	6	Output of 2x2 Vedic Multiplier 4
D4	$(01001)_2$	9	Output of 4 bit QSD adder 1 (D1+D2)
D5	$(10001)_2$	17	Output of 4 bit QSD adder 2 (D4 +{D3[1:0],D0[3:2]})
C[1:0]	$(11)_2$	3	D0[1:0]
C[5:2]	$(0001)_2$	1	D5[3:0]
C[7:6]	$(10)_2$	2	Output of 2 Bit Adder (D3[3:2]+D4[4]+D5[4])
C[7:0]	$(10000111)_2$	135	Final Result

B. 32x32 multiplier

The 4x4 multiplier design can be scaled to multiply larger numbers as shown in Fig. 4, where the design is scaled up for a 32 bit multiplier.

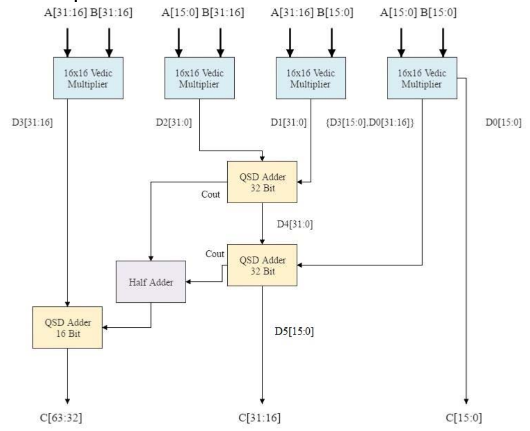


Fig. 4. Proposed 32x32 Multiplier

C. Proposed adder design based on QSD

In this paper, a novel idea of an adder, based on QSD (Quaternary Signed Digit) is proposed. The algorithm for the proposed adder uses a hybrid of quaternary and binary number systems. The outputs from smaller multipliers are obtained as binary strings. Inside the addition module, this string is broken into quaternary digits of two bits each. Addition using QSD allows us to reduce the carry propagation delay by making use of carry free arithmetic i.e. the carry doesn't ripple past the subsequent quaternary digit. Especially for higher bit input strings this method is extremely efficient.

The difficulty in application of quaternary addition outside MVL (Multi Voltage logic) is that the least significant 2 bits of the binary representation of the quaternary digits can't be directly concatenated to form an output binary string for every case as depicted in Table II. Each string would have to be read individually and a conversion module that converts quaternary to binary would have to be employed. To overcome this limitation, the concept of an adjusting bit has been introduced.

TABLE II. CONVERSION OF A QUATERNARY NUMBER TO BINARY NUMBER SYSTEM

Quaternary number A	2 1 $\rightarrow$ 010_001	Quaternary number B	2 $\bar{1}$ $\rightarrow$ 010_111
Binary equivalent of A	1001	Incorrect Binary equivalent of B	1011
Decimal equivalent of A	9	Incorrect Decimal equivalent of B	11

The Intermediate sum lies in the range [0, 6], as the operands are unsigned numbers. From [16], for quaternary addition to be carry free beyond the first stage, the intermediate sum can't be greater than 2. To ensure this stipulation holds true, the $(1\bar{1})_4$ representation of 3 needs to be chosen while adding. However, this represents a blocking case when converting the final output string back into binary as it prohibits us from simply concatenating the lower two bits of quaternary output strings to get the binary equivalent.

For addition of unsigned numbers, if the $(03)_4$ representation would have been used, direct concatenation of results could have been possible. But, then that wouldn't have always been carry free after the initial stage. Thus, the concept of an adjusting bit has been devised to solve the dilemma of which representation of 3 to use, such that both carry free addition and concatenation of output string bits to get the final output can be realized in the same design.

The solution to the problem described above, is that the $(03)_4$ representation of 3 is required to be taken instead of the $(1\bar{1})_4$ representation in some cases. But, determining when such a change is required before proceeding with the addition will increase the delay of the design and be counter-productive. Thus, the $(1\bar{1})_4$ representation of 3 is always selected in stage 1, to satisfy necessary conditions for carry free arithmetic. While necessary adjustments are made in stage 2 if $(03)_4$ representation was to be taken, the need for such an adjustment is determined via an adjusting bit.

OBSERVATION 1: In both quaternary representations of 3, $(03)_4$ or $(1\bar{1})_4$, the two least significant bits of the least significant digit are 11. Thus, regardless of which representation was supposed to be taken, the lower two bits of the intermediate sum will remain same and these are the two-bit positions that would be concatenated in the end.

The problem of incorrect representation would come under certain cases. To better understand these cases an example is described. The example uses two numbers as inputs represented using three quaternary digits each:

Input A= $(X_3X_2X_1)_4 = (A_8A_7A_6A_5A_4A_3A_2A_1A_0)_2 = (030)_4$

Input B = $(Y_3Y_2Y_1)_4 = (B_8B_7B_6B_5B_4B_3B_2B_1B_0)_2 = (001)_4$

The Base case: For addition of X_2 and Y_2, if the intermediate sum comes out to be 3, as stated above $(1\bar{1})_4$ representation will be chosen in stage 1. The Intermediate sum for this digit addition becomes $\bar{1}$ or $(111)_2$. If then the intermediate carry from the addition of X_1 and Y_1 is 0, the final output after stage 2 for this addition would be $\bar{1}$. The intermediate carry that will be added to addition of X_3 and Y_3 would be 1. The binary output thus received after the concatenation of lower two bits will be wrong, as shown in Table III.

As established above, this problem wouldn't have been there if the $(03)_4$ representation of 3 would have been used. According to the findings of observation 1, the intermediate carry from addition of X_2 and Y_2 needs to be negated for the correct result because for $(03)_4$ there would have been no carry. This negation will be done by the adjusting bit.

TABLE III. EXAMPLE OF QUATERNARY ADDITION USING ORIGINAL LOGIC

$X_3X_2X_1$	0 3 0 → 000_011_000	Input A
$Y_3Y_2Y_1$	0 0 1 → 000_000_001	Input B
U1	0 $\bar{1}$ 1 → 000_111_111	Stage 1 output (Intermediate sum)
U2	0 1 0	Stage 1 output (Intermediate carry)
R	1 $\bar{1}$ 1 → 001_111_001	Result (Before concatenation)
R'	$(01\ 11\ 01)_2$	Incorrect Result (After concatenation)

Mathematically this can be written as:

Final output = Intermediate sum + Intermediate carry – Adjusting bit.

Thus, adjusting bit can be said to be 1 when $(S_{n-1}.\bar{c})$ is true where S_{n-1} and $\bar{c}$ are defined as:

S_{n-1}: True if n-1th intermediate sum digit is 3.

$\bar{c}$: True if there is no carry from n-2th digits sum.

Secondly, another special case could arise when the intermediate sum for addition of X_2 and Y_2 and X_1 and Y_1 are both 3. For example if A = $(030)_4$ and B = $(003)_4$. Now as per previously devised logic the addition would have proceeded as in Table IV.

Thus, the final result as shown in Table IV, would have been $(01\ 11\ 11)_2$ which is incorrect. The intermediate carry from the addition of X_2 and Y_2 hasn't been negated while carry from addition of X_1 and Y_1 has. This is because intermediate carry from X_1 and Y_1 is taken as 1 while calculating the adjusting bit for X_3 and Y_3 While an adjustment is made to it later to negate it to 0. This adjustment hasn't been factored into the formula. Thus, the modified and complete formula for adjusting bit becomes as in (10).

TABLE IV. EXAMPLE OF QUATERNARY ADDITION USING INITIAL MODIFIED LOGIC

$X_3X_2X_1$	0 3 0 → 000_011_000	Input A
$Y_3Y_2Y_1$	0 0 3 → 000_000_011	Input B
U1	0 $\bar{1}$ $\bar{1}$ → 000_111_111	Stage 1 output (Intermediate sum)
U2	0 1 1	Stage 1 output (Intermediate carry)
A	0 1 0	Adjusting Bit
R	1 $\bar{1}$ $\bar{1}$ → 001_111_111	Result (Before concatenation)
R'	$(01\ 11\ 11)_2$	Incorrect Result (After concatenation)

$$\text{Adjusting bit} = S_{n-1}.(S_{n-2}+\bar{c}) \tag{10}$$

Where S_{n-2} is true if n-2th intermediate sum digit is 3. This formula can cover the problem of n consecutive 3's in a similar manner.

The adjusting bit can be predicted based on the initial inputs to the adders itself. It can be computed in parallel with Stage 1. Thus, effect on delay of the adder is minimal. The above example is revaluated with the modified formula:

Input A= $(X_3X_2X_1)_4 = (A_8A_7A_6A_5A_4A_3A_2A_1A_0)_2 = (030)_4$

Input B = $(Y_3Y_2Y_1)_4 = (B_8B_7B_6B_5B_4B_3B_2B_1B_0)_2 = (003)_4$

Adjusting Bit for addition of X_n and Y_n is $S_{n-1}.(S_{n-2}+\bar{c})$. As can be seen from the flow of data shown in Table V. The modified formula gives the correct binary output after concatenation.

The proposed adder works in two stages, as shown in Fig. 5.

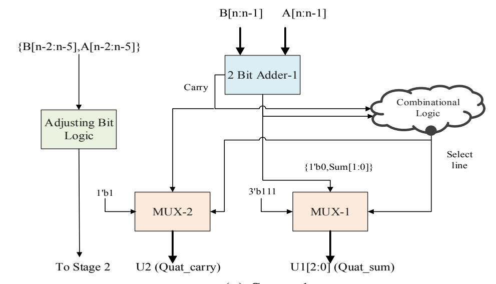
1) In the first stage, as in Fig. 5(a), every individual digit at the same position in the quaternary representation of two n-bit numbers A and B is added using a 2 Bit Adder to generate a sum. This sum lies in the range [0, 6]. From the sum obtained from the adder, the intermediate sum and intermediate carry for the next stage are calculated in parallel using 2x1 multiplexers. The logic for the selection of the representation of sum and carry has been explained in [16]. The adjusting bit is also computed in parallel with the addition process. The input to the adjusting bit calculation block for every quaternary digit addition are the previous two quaternary digits of A and B signified by [n-2: n-5].

2) Second stage has two modules as shown in Fig. 5(b). One is a one-bit module that performs the computation (A+B-C). In this case A would be LSB of intermediate sum, B would be carry from the previous quaternary digit addition and C would be the adjusting bit. The other module will be a half adder which will add the carry from the (A+B-C) module and the bit to the left of the least significant bit of the intermediate sum. As for the final concatenation, the sign bit would not be

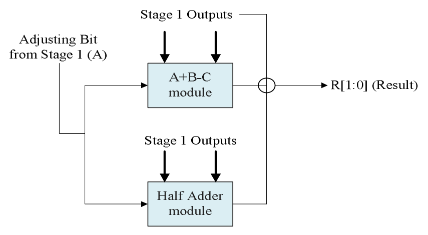
used owing to the adjustments proposed in the design. Thus, its final value is not computed.

TABLE V. EXAMPLE OF QUATERNARY ADDITION USING PROPOSED LOGIC

A	$(1100)_2$	Input 1
B	$(0011)_2$	Input 2
Q1	11_00 → 3 0	Quaternary representation of Binary number A
Q2	00_11 → 0 3	Quaternary representation of Binary number B
X_2X_1	3 0 → 011_000	Input A
Y_2Y_1	0 3 → 000_011	Input B
U1	$\bar{1}\,\bar{1}$ → 111_111	Stage 1 output (Intermediate sum)
U2	1 1	Stage 1 output (Intermediate carry)
S2	1	2nd intermediate sum digit is 3
S1	1	1st intermediate sum digit is 3
S0	0	0th digits do not exist
C1	1	Carry from sum of 2nd digits is 1
C0	1	Carry from sum of 1st digits is 1
A2	$S2.(S1+\overline{C1}) = 1$	2nd Adjusting Bit
A1	$S1.(S0+\overline{C0}) = 0$	1st Adjusting Bit
A	1 0	Stage 1 output (Adjusting Bit)
U3	001_111_111	Stage 2 output (Before Adjusting Bit logic)
U4	000_111_111	Stage 2 output (After Adjusting Bit logic)
R	$(1111)_2$	Result after concatenation



(a) Stage 1



(b) Stage 2

Fig. 5. Proposed Adder

IV. RESULTS

In this section, we present a comparison between proposed design of multiplier and existing architectures namely Vedic multiplier based on MVL logic that uses a ripple carry adder [16], Vedic Multiplier that incorporates a QSD adder with a conversion module for quaternary to binary conversion, Vedic multiplier based on a different fast adder scheme such as Carry select adder [17] and a known fast multiplication scheme such as Booth multiplier [18]. These four architectures were chosen and implemented to verify the viability of proposed design across all domains it's pertinent to. All architectures are described using Verilog HDL and all the possible states including corner cases for digit by digit multiplication blocks are verified using simulation with Xilinx ISim simulator. The design synthesis has been carried out using Synopsys Design Compiler, using the open-source NAN gate 15nm technology library[19]. Table VI shows that proposed design has made substantial improvements in terms of speed over the existing designs. The total delay of 128x128 Multiplier based on Proposed Design comes out to be 578.85 ps which is 88.75% faster than booth multiplier,71.35% faster than MVL multiplier based Multiplier, 17.47% faster than Carry select adder based Multiplier and 51.69% faster when compared with QSD Adder based Vedic Multiplier using conversion module.

Proposed 128x128 design has 7.7% lower implementation area then CSA based Vedic multiplier but shows an increase in area over other three designs, as shown in Table VII, it can be considered as a tradeoff for the substantial improvement in speed over those designs. As shown in Table VIII, for 16 input bit value the proposed design consumes the lowest power amongst the designs compared. Whereas, for the larger input sizes, the power consumed by proposed designs is 25.14% and 20.64% more than the lowest recorded power amongst the designs compared for 64 bit and 128 bit respectively.

TABLE VI. COMPARISON OF PROPOSED DESIGN WITH OTHER MULTIPLIER ARCHITECTURES ON THE BASIS OF TOTAL DELAY

Type Of Multiplier	**Delay (ps)**			
	16 bit	***32 bit***	***64 bit***	***128 bit***
Proposed Design	266.13	422.65	501.68	578.85
QSD Adder based Vedic Multiplier with conversion module	308.27	506.96	878.25	1198.21
CSA based Binary Vedic Multiplier	362.61	484.88	595.11	701.43
MVL Multiplier	431.18	949.25	1763.98	2020.71
Booth Multiplier	637.57	1259	2604	5148.56

TABLE VII. COMPARISON OF PROPOSED DESIGN WITH OTHER MULTIPLIER ARCHITECTURES ON THE BASIS OF AREA

Type Of Multiplier	**Area (No. of Cells)**			
	16 bit	***32 bit***	***64 bit***	***128 bit***
Proposed Design	768	3475	14440.6	58842.6
QSD Adder based Vedic Multiplier with conversion module	678.9	2660.8	12303	50181.9
CSA based Binary Vedic Multiplier	938	3884.2	15801	63767
MVL Multiplier	432	1765.1	7212	29464.2
Booth Multiplier	605	2332	8977	35987

TABLE VIII. COMPARISON OF PROPOSED DESIGN WITH OTHER MULTIPLIER ARCHITECTURES ON THE BASIS OF POWER

Type Of Multiplier	**Power (mW)**			
	16 bit	***32 bit***	***64 bit***	***128 bit***
Proposed Design	737.72	4446.7	24132	99630
QSDA based Vedic Multiplier with conversion module	774.35	4066.3	20958	87176
CSA based Binary Vedic Multiplier	1024.4	4558.7	19284	82580
MVL Multiplier	912.5	4722.2	21189	96478
Booth Multiplier	800.62	4627.1	19659	86556

V. CONCLUSION

It can be concluded that the design when scaled to higher bits only shows a marginal rise in delay due to its core strengths. Firstly, the parallelism involved in its partial product generation. Secondly, reduction of carry propagation delay in the novel adder it incorporates. Due to the use of QSD, the design is able to incorporate carry free arithmetic while eliminating radix conversion module speed overhead by integrating concept of adjusting bit logic in its architecture.

The proposed design showed an increase in implementation area over some designs due to increased parallelism even in finer nuances of the architecture. The proposed design is targeted towards digital systems requiring high throughput and low latency at the cost of area overhead. For example, in a DSP system, operations such as Fast Fourier Transform, Convolution, Filtering and Discrete Wavelet transform etc. Multipliers play a key role in determining the speed of the system. Similarly, this architecture would be a good candidate to be implemented as a large part of systems like DCT, Central Processing Unit (CPU), MAC (Multiply and Accumulate) Unit, Image Processors where high-speed multiplications are critical to the performance of the system.

It can also be observed that despite the objective of decreasing the delay, the proposed design performs better than most designs compared in terms of power for lower input bit sizes [16 and 32 bit]. Although it consumes more power than other designs higher input bit sizes [64 and 128 bit], it is justifiable when factored in with advantages gained in speed for higher input bits.

REFERENCES

[1] M. Rabaey, A. Chandrakasan, and B. Nikolic, “Digital Integrated Circuits A Design Perspective,” PHI, 2003.

[2] B. Pahrami, “Computer Arithmetic and Hardware Design,” New York, Oxford University Press, 2000.

[3] M. Ercegovac, and T. Lang, “Digital Arithmetic, San Francisco, Morgan Kaufmann,” 2004.

[4] C S Wallace, “A Suggestion for a Fast multiplier”, IEEE Transactions on Electronic Computers, Vol. EC-13, Issue 1, pp. 14-17, 1964.

[5] K. Choi and M. Song, “Design of a high performance 32x32-bit multiplier with a novel sign select booth Encoder,” in IEEE International Symposium on Circuits and Systems, Volume 2, 2001, pp. 701-704.

[6] J. Swami S. B. K.Tirthaji Maharaja, “Vedic Mathematics: Sixteen Simple Mathematical Formulae from the Veda,” Delhi, 1965.

[7] S. N. A and K. N, “Implementation of Power Efficient Vedic Multiplier,” International Journal of Computer Applications (0975 – 8887), Vol. 43– No.16, 2012, pp. 21-24.

[8] S. Vaidya and D. Dandekar, “Delay-Power Performance Comparison of Multipliers In VLSI Circuit Design,” International Journal of Computer Networks & Communications (IJCNC), Vol.2, No.4, July 2-010.

[9] H. Thapliyal and M. B. Srinivas, “High Speed Efficient N X N Bit Parallel Hierarchical Overlay Multiplier Architecture Based On Ancient Indian Vedic Mathematics”, Enformatika (Transactions on Engineering, Computing and Technology),Vol. 2, Dec 2004, pp. 225-228.

[10] H. D. Tiwari, G. Gankhuyag, C. M. Kim and Y. B. Cho, “Multiplier design based on ancient Indian Vedic Mathematics,” International SoC Design Conference, 2008, pp. 65-68.

[11] D. Jaina, K. Sethi and R. Panda, “Vedic Mathematics based multiply accumulate Unit,” IEEE International Conference on Computational Intelligence and Communication Systems.2011, pp. 754-757.

[12] S. Jinesh, P. Ramesh and J. Thomas, “Implementation of 64 bit high speed multiplier for DSP application- based on Vedic mathematics,” in IEEE TENCON,2015, pp. 1-5.

[13] J. Thomas, R Pushpangadan, S Jinesh, “Comparative study of performance of vedic multiplier on the basis of adders used,” IEEE-WIECON, 2015, pp. 325-328.

[14] R. Rani, L. K. Singh and N. Sharma, “FPGA Implementation of Fast Adders using Quaternary Signed Digit Number System,” 2009 International Conference on Emerging Trends in Electronic and Photonic Devices & Systems (ELECTRO-2009), pp. 132 - 135.

[15] Nagamani A. N, Nishchai S, "Quaternary High Performance Arithmetic Logic Unit Design", 14th Euromicro Conference on Digital System Design 2011 IEEE.

[16] A. S. Shende, M. A. Gaikwad and D. R. Dandekar, “Design of efficient 4X4 Quaternary Vedic Multiplier Using Current Mode Multi Valued Logic,” Int. J. on recent Trends in Engineering and Technology,Vol 10, No 2, Jan. 2014, pp. 59-69.

[17] G. R. Gokhale, S. R. Gokhale, “Design of area and delay efficient Vedic multiplier using Carry Select Adder,” International Conference on Information Processing (ICIP), 2015, pp. 295 – 300.

[18] S. Kim and K. Cho, “Design of High-speed Modified Booth Multipliers Operating at GHz Ranges,” World Academy of Science, Engineering and Technology 61, 2010, pp. 1-4.

[19] M. Martins, J. M. Matos, R. P. Ribas, A. Reis, G. Schlinker, L. Rech and J. Michelsen, “Open Cell Library in 15nm FreePDK Technology”, In Proceedings of the International Symposium on Physical Design (ISPD), 2015.