# Memory Based Multiplier Design in Custom and FPGA Implementation

M. Mohamed Asan Basiri and S.K. Noor Mahammad

**Abstract.** The modern real time applications like signal processing, filtering, etc., demands the high performance multiplier design with fewer look up tables in FPGA implementation. This paper proposes an efficient look up table based multiplier design for ASIC as well as FPGA implementation. In the proposed technique, both the input operands of the multiplier are considered as variables and the proposed $LUT$ based multiplier design is compared with other schemes like $LUT$ counter, $LUT$ of squares and $LUT$ of decomposed squares based multiplier designs. The performance results have shown the proposed design achieves better improvement in depth and area compared with existing techniques. The proposed $LUT$ based $12 \times 4$-bit multiplier achieves an improvement of 34.61% in depth compared to the counter $LUT$ based architecture. The $16 \times 16$-bit proposed $LUT$ based multiplier achieves an improvement factor of 76.84% in the circuit depth over the square $LUT$ based multiplication technique using 45 *nm* technology.

## 1 Introduction

In general, multipliers are the crucial components of cryptography systems like elliptic curve cryptography algorithms [1] and digital signal processing algorithms like fast Fourier transform (FFT) [2]. The multipliers are basic part of multiply accumulate circuit (MAC) [3], which is the heart of digital signal processor, where two input operands are multiplied and the present multiplication result is added to the previous MAC result. Any digital filter application like finite impulse response (FIR) requires a multiplier to perform the operation. The multiplier with lesser depth, lower power requirement and lower area impact the digital system to

M. Mohamed Asan Basiri · S.K. Noor Mahammad
Department of CSE, Indian Institute of Information Technology Design and Manufacturing Kancheepuram
e-mail: {asanbasiri,noorse}@gmail.com

achieve high performance arithmetic. A digital multiplier has two parts, they are partial product generation and addition of partial products. The partial product generation will take $\Theta(1)$ time complexity and depth of the addition of partial products will be varied according to the multiplier structure. In general, carry save multipliers are having two parts in their partial product addition, they are carry save addition tree and final addition. The depth of carry save adder (csa) is $O(1)$. The depth of the carry save addition tree is $O(log_2\, n)$ for Wallace tree [4] multiplier and $O(n)$ for Braun multiplier, where $n$ is the number of bits. The number of carry save stages will be varied in both the above mentioned multipliers. The *sum* and *carry* from the last carry save stage of the multiplier is further added with final adder, where ripple carry adder (RCA) is used with $O(n)$ circuit depth or recursive doubling based carry look ahead adder (CLA) is used with $O(log_2\, n)$ depth.

The $n$-bit multiplier produces a $n$ number of partial products. If the number of partial products is getting reduced, then the depth of multiplier will be reduced. For this purpose, the higher radix modified Booth algorithm [5] is used. In this case, Booth encoder itself, will cause an increase in multiplier circuit depth. The paper [6] shows the basic complex number multiplier structure, where four multipliers are involved to perform one complex number multiplication. The paper [7] shows the Baugh Wooley based signed multiplication scheme, which is similar to array multiplier with circuit depth $O(n)$. In modern technology, vector processors [8] are playing a major role to achieve data level parallelism (DLP). Here multiple operands are following multiple data paths in the same hardware. So only one instruction is used to perform multiple operations on the vector of data. All the operands are processed in parallel, thus improves the speed of the operation. The twin precision based array multiplier is explained in [9], where the full precision multiplier is used to perform two half precision multiplications with circuit depth of $O(n)$ to achieve DLP. This means that, 8-bit multiplier is used to perform two 4-bit multiplications or one 8-bit multiplication at a time.

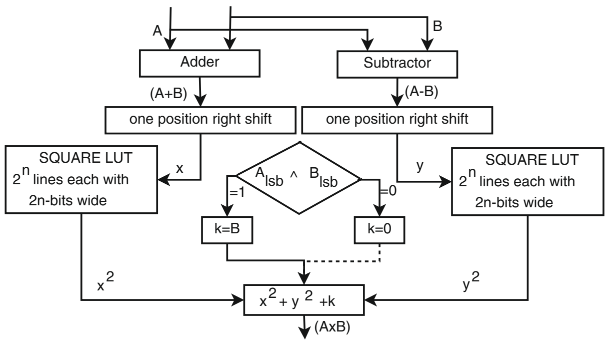

**Fig. 1** LUT of squares based multiplier

In the last two decades, look up table ($LUT$) based multiplier design is becoming an emerging technology in high performance arithmetic circuit design. For example, [10] shows the implementation of $LUT$ based discrete cosine transform, FIR filtering. The papers [11], [12] and [13] explain the $LUT$ based digital signal processing operations. In general $LUT$ based multiplication of two input operands $A$ and $B$ can be found by,

$$x = (A+B)/2 \tag{1}$$

$$y = (A-B)/2 \tag{2}$$

$$AB = x^2 - y^2 + k \tag{3}$$

Here $k$ is equal to 0, if $A$ and $B$ both are even/odd and $k$ is equal to $B$ if any one of $A$ or $B$ is even. The squared values $x^2$ and $y^2$ are obtained from the $LUT$. If the input operands $A$ and $B$ are $n$-bit wide, then the $x$ and $y$ also will become a $n$-bit wide. Hence the number of $LUT$ entries will be $2^n$ and each with $2n$-bits wide. So the size of $LUT$ will be $(2^{n+1}n)$ bits. The major drawback with this approach is the requirement of larger $LUT$. The Fig. 3 shows the above mentioned LUT based multiplication scheme, where the $xor$ value of least significant bit ($lsb$) of $A$ and $B$ ($A_{lsb}$ and $B_{lsb}$) are used to find the $k$ value. The paper [14] shows that the $LUT$ size reduction techniques for the above mentioned square based multiplication. In the $LUT$ of squares based multiplication, the least $n$-bits of the squared value ($2n$-bits wide) are periodic with the period $(2^n - 1)$ where the input of the $LUT$ is considered as $n$-bits wide.

The paper [15] shows the $LUT$ counter based multiplication scheme. If the multiplicand is $m$-bits wide and multiplier is $p$-bits wide, then the number of partial products will be $p$ each with $m$-bits wide, which tends to produce $m+p-1$ columns of partial products. According to [15], $t$ columns of $p$ partial products are sent to the $LUT$. The number of ones in each column is counted. The counted value is added to the other columns. The outputs from all the $LUTs$ are added to compressor tree. So the size of the $LUT$ will be $2^{pt}q$ bits, where $q$ is the number of bits in the output

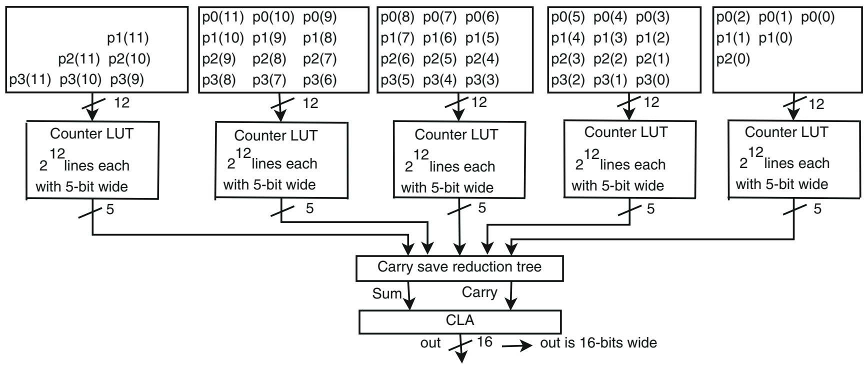

**Fig. 2** $LUT$ counter based 12x4 multiplier design

from an each $LUT$. The Fig. 2 shows the $LUT$ counter based 12x4 multiplier design, where 4 partial products each with 12 bits wide are added using five counter $LUTs$. Here 3 columns (12 bits) of partial products are sent to the counter $LUT$. The maximum number of 1's in each column is 4. So the $LUT$ is having $2^{12}$ lines each with 5-bits wide. All the three columns should be added in such a way, where the second column (one position left shifted) is added to the third column (two positions left shifted) and the first column. Here the first column is not shifted. If all the three columns are having four 1's, then the shifted addition value three columns will be 11100. So each of the counter $LUT$ is having the 5-bit output line. In the next step, all the output lines of the counter $LUTs$ are added through carry save reduction tree to get the final result. The drawback with this approach is that the $LUT$ size will be increased if the more number of columns are sent to the counter $LUT$. The number of $LUTs$ will be increased if the less number of columns are sent. This will increase the stages of carry save reduction tree. The paper [16] proposed $LUT$ based FIR filter design, where the filter co-efficients are considered as constant values. If the input signal sample value $x$ is considered as $n$-bit value, then the number of possible multiplication results between $x$ and the filter co-efficient $(A)$ will be $2^n$, where $A$ is assumed as $m$-bit constant. Hence the $LUT$ will contain $2^n$ entries, each with $(m+n)$-bits wide.

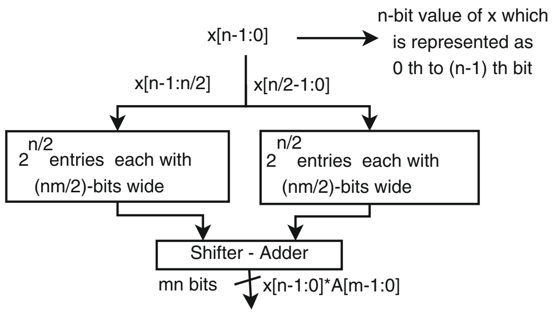
x[n-1:0]
n-bit value of x which is represented as 0 th to (n-1) th bit
x[n-1:n/2]
x[n/2-1:0]
2^n/2 entries each with (nm/2)-bits wide
2^n/2 entries each with (nm/2)-bits wide
Shifter - Adder
mn bits
x[n-1:0]*A[m-1:0]

**Fig. 3** Look up table based constant multiplication

The Fig. 3 has shown the above mentioned multiplication, where the $n$-bit input operand $x$ is divided into two parts. So the number of $LUT$ entries will be $2^{n/2}$ and each with $(m+(n/2))$-bits wide and finally the outputs from both the $LUTs$ are added to shifter-adder circuit to get the multiplication result. The drawback with this approach is that, it's not suitable for general purpose hardware design where the filter co-efficients are not considered as constants. If the filter co-efficient is a large number, then the $LUT$ size will be increased. Here the size of the $LUT$ is based on the value of $m$ and $n$. The value of $m$ will be higher for large values of $A$.

### 1.1 Contribution of This Paper

The drawbacks on the existing LUT based multiplication from the above literature gives the motivation towards the proposed $LUT$ based multiplication scheme. Here the size of the $LUT$ and delay, both are considered. The proposed technique uses the smaller $LUTs$, where both the input operands are considered as lesser number of bits. These smaller $LUTs$ are used to build $n \times n$-bit multiplication, where $n$ is considered as larger number bits. This means that, the given input operands are decomposed into smaller multiplications, where each result of the smaller multiplication can be obtained by using the smaller $LUTs$. In the final step, carry save reduction tree is used to add the results from all the smaller LUTs to obtain the desired result. Hence the size of the $LUT$ can be maintained as small. Due to the carry save reduction tree, the depth of the circuit will be in the time bound of $\Theta(log_2 n)$. This proposed $LUT$ based multiplier is used for general purpose multiplication, where both the input operands are considered as $n$-bit variables. The experimental results show that the proposed $LUT$ based multiplier gives better performance (in terms of delay and area) than the existing $LUT$ based techniques like $LUT$ counter, $LUT$ of squares and $LUT$ of decomposed squares based multipliers. The rest of the paper is organized as, section 2 states the proposed $LUT$ based multiplication scheme. Design modeling, implementation and results are discussed in section 3, followed by a section 4 conclusion.

## 2 The Proposed $LUT$ Based Multiplication

The proposed $3 \times 3$-bit multiplier is shown in Fig. 4, where three $LUTs$ are used to store all the possible results. The 3-bit input operands are $A$ and $B$. The address line for each LUT will be $B$. The 8 to 1 multiplexer is used to get all the possible results. The select line for the multiplexer is $A$. The $LUTs$ (connected to 3, 5 and 7 th input line of multiplexer) are used to store the multiplication result of $A$ by 3, 5 and 7 respectively. Since it is 3-bit multiplication, the line width of each $LUT$ is 6 bits. The number of lines in each $LUT$ will be 8. The data at the $2^i$ th input of multiplexer can be obtained by left shifting ($i$ times) of $A$, where $i = 0, 1, 2, \ldots$. The data at the even number (other than $2^i$) input line of the multiplexer can be obtained by left shifting the output of the $LUT$ to the corresponding odd position. Here the left shifting can be implemented by hardwire connection and hence it doesn't require any shifting units. In general, number of $LUTs$ used for $n \times n$ multiplier is $(2^{n-1} - 1)$ and each $LUT$ is consisting of $2^n$ lines each with $2n$-bits wide. So the size of each $LUT$ will be $(2^{n+1}n)$ bits.

A $2^n$ to 1 multiplexer is required to design the $n \times n$-bit $LUT$ based multiplier. If $n$ is large, then the requirement of larger multiplexer will cause an overhead of the design. The table 1 shows the comparison between the various $LUT$ based multipliers, which clearly shows about the number of $LUTs$, number of lines per $LUT$, line width of $LUT$ in bits, the size of each $LUT$ in bits and multiplexer involved in the

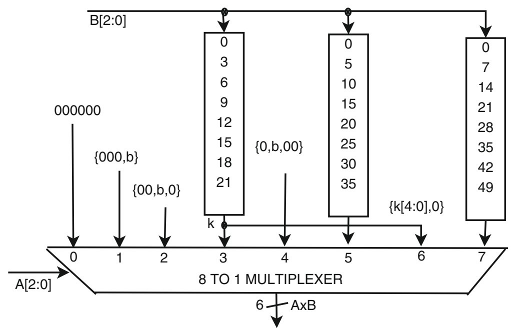

**Fig. 4** Proposed *LUT* based $3 \times 3$-bit multiplier

**Table 1** Comparison of *LUT* based multipliers

| *LUT* based multiplier | No. of *LUTs* used | No. of lines per *LUT* | Line width of *LUT* (bits) | Size of each *LUT* (bits) | Multiplexer used |
|---|---|---|---|---|---|
| $2 \times 2$ | 1 | 4 | 4 | 16 | 4 to 1 |
| $2 \times 3$ | 1 | 8 | 5 | 40 | 4 to 1 |
| $2 \times 4$ | 1 | 16 | 6 | 96 | 4 to 1 |
| $2 \times 5$ | 1 | 32 | 7 | 224 | 4 to 1 |
| $3 \times 3$ | 3 | 8 | 6 | 48 | 8 to 1 |
| $3 \times 4$ | 3 | 16 | 7 | 112 | 8 to 1 |
| $3 \times 5$ | 3 | 32 | 8 | 256 | 8 to 1 |
| $4 \times 4$ | 7 | 16 | 8 | 128 | 16 to 1 |
| $4 \times 5$ | 7 | 32 | 9 | 288 | 16 to 1 |
| $5 \times 5$ | 15 | 32 | 10 | 320 | 32 to 1 |

particular multiplier design. These simpler *LUT* based multipliers are used in the larger multiplier design and hence the size of the LUT will be maintained as small.

The Fig. 5 shows the implementation of $16 \times 16$-bit multiplier using $4 \times 4$-*LUT* based multipliers. The input operands $A$ and $B$ both are considered as 16-bits wide. So they can be decomposed into 4 parts, each with 4-bits wide. This is means that, $A$ is decomposed into $a0$, $a1$, $a2$ and $a3$. Similarly $B$ is decomposed into $b0$, $b1$, $b2$ and $b3$. The multiplication results $a0b0$, $a1b0$, $a2b0$, $a3b0$, $a0b1$, $a1b1$, $a2b1$, $a3b1$, $a0b2$, $a1b2$, $a2b2$, $a3b2$, $a0b3$, $a1b3$, $a2b3$ and $a3b3$ are obtained from $4 \times 4$-bit *LUTs*. The Fig. 5(a) shows the arrangement of output values obtained from the sixteen $4 \times 4$ *LUTs*. The Fig. 5(b) shows carry save reduction tree for adding all the partial results $p0$, $p1$, ...$p7$, where *csa* represents the carry save adder with time complexity $O(1)$. So the depth of the carry save reduction tree will be reduced compared to the conventional Wallace structure because the conventional $16 \times 16$-bit Wallace structure contains 16 partial products. In the proposed design, the number of partial results to the carry save reduction tree is depending on the

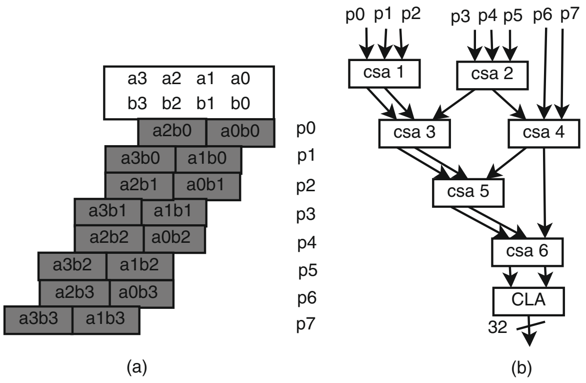

**Fig. 5** Proposed *LUT* based 16 × 16-bit multiplier (a) Arrangement of output from sixteen 4 × 4 *LUTs* (b) Carry save reduction tree to add outputs from all the *LUTs*

number of decompositions of the multiplier/multiplicand and the width of the multiplier/multiplicand. In case of 16 × 16-bit multiplier, the number of decompositions of the multiplier/multiplicand is 4 and which is shown in Fig. 5.

In the Fig. 5, both the operands $A$ and $B$ are decomposed by 4. So all the outputs from the *LUTs* are arranged in exactly half of the previous result. In some cases, the decomposition can be done by 2 or 3 or 4 or 5 or any other combination. In this case, the alignment of the output from *LUTs*, tends to give an important role. For example, Fig. 6 shows the various possible decompositions for 10-bit multiplicand ($A$) which is multiplied by 4-bit multiplier ($B$). The Fig. 6(a) shows the decomposition of the multiplicand by 3. This means that, multiplicand ($A$) is decomposed into $A[2:0]$, $A[5:3]$ and $A[9:6]$. The arrangement of outputs from the *LUTs* is having 3 partial results, they are $p0$, $p1$ and $p2$. Here the 0-th bit of the $A[5:3] \times B[3:0]$ should be aligned with 3-rd bit of $A[2:0] \times B[3:0]$. Similarly the 0-th bit of the $A[9:6] \times B[3:0]$ should be aligned with 6-th bit of $A[2:0] \times B[3:0]$.

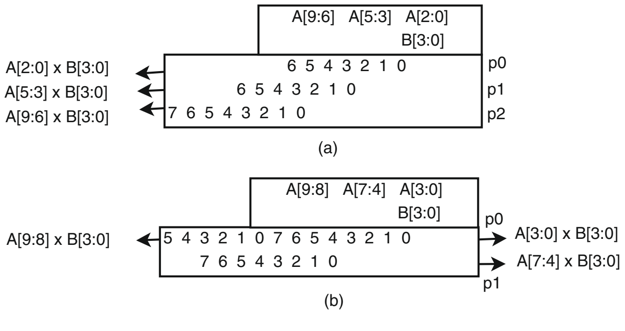

**Fig. 6** Proposed *LUT* based 10 × 4-bit multiplication (a) Decomposition with 3 partial results (b) Decomposition with 2 partial results

In Fig. 6(b), multiplicand ($A$) is decomposed into $A[3:0]$, $A[7:4]$ and $A[9:8]$. The arrangement of outputs from the $LUTs$ is having 2 partial results, they are $p0$ and $p1$. Here the 0-th bit of $A[7:4] \times B[3:0]$ is aligned with the 4-th bit of $A[3:0] \times B[3:0]$. Similarly the 0-th bit of $A[9:8] \times B[3:0]$ is aligned with 4-bit of $A[7:4] \times B[3:0]$. All the partial results are obtained from the $LUTs$. Due to an extra partial result ($p2$), the depth of the multiplier in Fig. $6(a)$ is higher than the multiplier in Fig. $6(b)$. In both the cases, the multiplier is treated as zero decomposition. So in general, the multiplier and multiplicand can be decomposed into $n1$ and $n2$ parts respectively. So the whole multiplicand can be multiplied by $n2$ times of full multiplication. Each full multiplication is consisting of $n1$ number of half multiplications. The decompositions should be in such a way that, each full multiplication should give maximum of two partial results to achieve lesser depth of carry save reduction tree. This will give the way of selecting smaller $LUT$ based multipliers for the decomposition of the required $n \times n$ multiplier.

## 2.1 Time Complexity Analysis of Proposed *LUT* Based Multiplier

If the number of decompositions of the multiplier is $d$ (where $d > 1$), then the number of partial results to the carry save reduction tree will be $kd$. Here the proposed logic is considered as the $n \times n$-bit multiplier is made up of basic $LUTs$ mentioned in table 1 only. The value of $k = 2$ if the number of decompositions in multiplicand is more than 1 and $k = 1$ if the number of decompositions in multiplicand is equal to 1. So the depth of the carry save reduction tree will become $O(log_2\ kd)$, which is lesser than the depth of the conventional $n \times n$-bit Wallace tree multiplier ($O(log_2\ n)$). The important thing is to align the multiplication result from the basic $LUTs$ before carry save addition. So the time complexity for the proposed $n \times n$-bit multiplier is $T(n) = T(LUT) + T(mux) + O(log_2\ kd) + O(log_2\ (2n-1))$, where $T(LUT)$ represents the time taken by accessing the $LUT$, $T(mux)$ represents the depth of the multiplexer used, $O(log_2\ kd)$ shows the depth of the carry save reduction tree and $O(log_2\ (2n-1))$ shows the time complexity for the recursive doubling based carry look ahead adder (CLA) used in the last stage of the multiplier. The basic $LUT$ based multipliers mentioned in that table are varied from $2 \times 2$ to $5 \times 5$. If the requirement of multiplier goes beyond $5 \times 5$, then the number of $LUTs$ used to design the particular multiplier will be high and this causes a huge memory requirement and this also causes a requirement of larger multiplexer. So any $n \times n$-bit multiplier can be designed using the proposed technique with the basic $LUT$ based multipliers mentioned in Table 1.

## 3 Design Modeling, Implementation and Results

The proposed and existing designs are modeled in Verilog HDL. These Verilog HDL models are simulated and verified using the Xilinx ISE simulator. The timing, area and power analysis of this implementation has been done with Cadence 6.1 ASIC design tool. All the designs are implemented for 45 *nm* technology, where the library $tcbn45gsbwpbc088\_ccs.lib$ is used for estimating the timing/area/power details.

**Table 2** Performance analysis of $12 \times 4$ multiplier using 45 *nm* technology

| | $LUT$ counter based | Proposed $LUT$ based |
|---|---|---|
| Worst path delay ($ps$) | 439.4 | 287.3 |
| Total area ($\mu m^2$) | 758.71 | 537.49 |
| Net power ($nw$) | 9520.08 | 6890.34 |

The Table. 2 shows the worst path delay, total area and net power comparison between the counter and proposed $LUT$ based $12 \times 4$-bit multiplier using 45 *nm* technology. Here depth of the $LUT$ counter based $12 \times 4$-bit multiplier seems to be higher than the proposed $LUT$ based multiplier due to the increase in the depth of the carry save tree which is mentioned in section 1. The proposed $LUT$ based $12 \times 4$-bit multiplier achieves an improvement of 34.61% in depth compared to the counter $LUT$ based architecture. The Table 3 shows the details about the $LUT$ of decomposed squares for the multipliers $4 \times 4$, $8 \times 8$, $12 \times 12$ and $16 \times 16$-bit multipliers. Here the conventional square $LUT$ is decomposed into two $LUTs$ and both are having different address lines as the inputs. The number of lines and the line width (in bits) for both the $LUTs$ of decomposed squares are varied according to the each multiplier, these details are mentioned in the Table 3. In Table 3, the input address line is mentioned as *in* and the output from the $LUT$ is mentioned as *out*. The proposed $LUT$ based $4 \times 4$, $8 \times 8$, $12 \times 12$ and $16 \times 16$-bit multipliers are compared with the LUT of squares based and decomposed LUT of squares based [14] techniques.

**Table 3** Decomposed square $LUT$ based multipliers

| | First $LUT$ of decomposed squares | | | | Second $LUT$ of decomposed squares | | | |
|---|---|---|---|---|---|---|---|---|
| | Input address | No. of lines | Line width | Output | Input address | No. of lines | Line width | Output |
| $4 \times 4$ | in[1:0] | 4 | 2 | out[1:0] | in[3:0] | 16 | 6 | out[7:2] |
| $8 \times 8$ | in[4:0] | 32 | 5 | out[4:0] | in[7:0] | 256 | 11 | out[15:5] |
| $12 \times 12$ | in[6:0] | 128 | 7 | out[6:0] | in[11:0] | 4096 | 17 | out[23:7] |
| $16 \times 16$ | in[8:0] | 512 | 9 | out[8:0] | in[15:0] | 65536 | 23 | out[31:9] |

The Fig. 7, 8 and 9 are showing the worst path delay, area and net power comparison between the proposed with an other existing $LUT$ based $4 \times 4$, $8 \times 8$ and $12 \times 12$-bit multipliers using 45 *nm* technology respectively. In these cases, the proposed $LUT$ based multiplier seems to be better than the other existing $LUT$ based multiplication techniques, they are $LUT$ of squares based and $LUT$ of decomposed squares based multipliers. The Table 4 is showing the worst path delay, total area and net power comparison between the $16 \times 16$-bit proposed $LUT$ based multiplier with the other existing $LUT$ based techniques. The $16 \times 16$-bit proposed $LUT$ based multiplier achieves an improvement factor of 76.84% in the circuit depth over the square $LUT$ based multiplication technique using 45 $nm$ technology and the same achieves an improvement of 95.2% in area reduction over the square $LUT$ based multiplication.

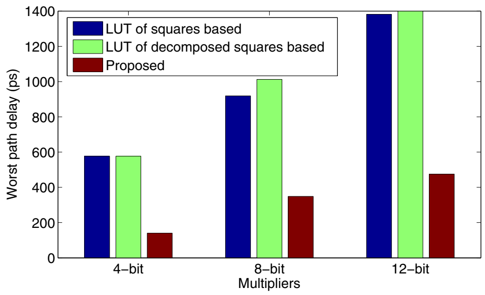

**Fig. 7** Worst path delay ($ps$) comparison for $4 \times 4$, $8 \times 8$ and $12 \times 12$-bit multiplier using 45 $nm$ lib

The above mentioned whole designs, are implemented with FPGA, where the device $EXC7A100T$ from the family of $Artix$ 7 with package $CSG324$ is used. The Table 5 shows the comparison between the counter based and proposed $LUT$ based $12 \times 4$-bit multiplier in FPGA implementation. The number of $LUTs$ used in LUT counter based and proposed $LUT$ based $12 \times 4$-bit multiplier are 294 and 134 respectively. The Table 6 shows the comparison of delay and number of $LUTs$ for $4 \times 4$, $8 \times 8$, $12 \times 12$ and $16 \times 16$-bit multipliers using the existing and proposed $LUT$ based multiplier designs. In all the cases, the proposed technique requires a lesser number of $LUTs$ than other techniques.

**Table 4** Performance analysis of $16 \times 16$ multiplier using 45 $nm$ technology

| | $LUT$ of squares based | $LUT$ of decomposed squares based | proposed |
|---|---|---|---|
| Worst path delay ($ps$) | 2331.8 | 2228.9 | 540.9 |
| Total area ($\mu m^2$) | 95876.4 | 96668.08 | 4219.31 |
| Net power ($nw$) | 1684823.36 | 1709193.57 | 75155.92 |

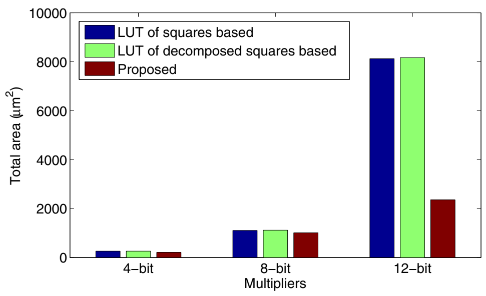

**Fig. 8** Total area ($\mu m^2$) comparison for $4 \times 4$, $8 \times 8$ and $12 \times 12$-bit multiplier using 45 *nm* lib

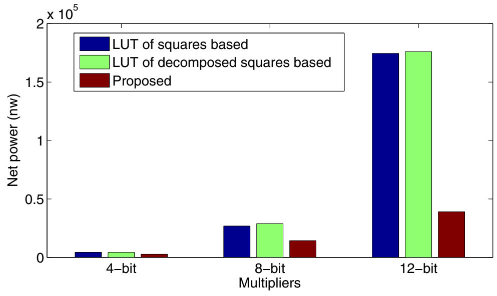

**Fig. 9** Net power (*nw*) comparison for $4 \times 4$, $8 \times 8$ and $12 \times 12$-bit multiplier using 45 *nm* lib

**Table 5** Comparison of number of $LUTs$ and delay in FPGA implementation for $12 \times 4$ multiplier

| | Delay (*ns*) | No. of $LUTs$ |
|---|---|---|
| $12 \times 4$-bit $LUT$ counter based | 6.975 | 294 |
| $12 \times 4$-bit proposed $LUT$ based | 5.253 | 134 |

**Table 6** Comparison of number of $LUTs$ and delay (*ns*) in FPGA implementation

| | $LUT$ squares | | $LUT$ of decomposed squares | | Proposed $LUT$ | |
|---|---|---|---|---|---|---|
| | No. of $LUTs$ | Delay | No. of $LUTs$ | Delay | No. of $LUTs$ | Delay |
| $4 \times 4$ | 44 | 5.047 *ns* | 44 | 5.0447 *ns* | 27 | 2.368 *ns* |
| $8 \times 8$ | 224 | 10.901 *ns* | 229 | 10.968 *ns* | 153 | 5.581 *ns* |
| $12 \times 12$ | 1670 | 13.295 *ns* | 1670 | 13.295 *ns* | 417 | 8.160 *ns* |
| $16 \times 16$ | 64483 | 19.046 *ns* | 61679 | 18.915 *ns* | 723 | 9.378 *ns* |

## 4 Conclusion

In this paper, an efficient $LUT$ based multiplier design is proposed, where both the input operands are treated as $n$-bit variables. So this proposed architecture has the advantage over an existing $LUT$ based multiplier design, where one of the operands is considered as constant. This proposed $LUT$ based multiplier design is compared with other schemes like $LUT$ counter, $LUT$ of squares and $LUT$ of decomposed squares based multiplier designs. The performance results have shown the proposed design achieves better improvement in depth and area compared with an existing technique.

## References

1. Koblitz, N.: Elliptic curve crptosystems. Mathematics of Computation 48(177), 203–209 (1987)
2. Smith, S.W.: The Scientist and Engineers Guide to Digital Signal Processing, pp. 551–566. California Technical Publishing (1997)
3. Elguibaly, F.: A fast parallel multiplier accumulator using the modified Booth algorithm. IEEE Transactions on Circuits Systems 27(9), 902–908 (2000)
4. Wallace, C.S.: A suggestion for a fast multiplier. IEEE Transactions on Electronic Computers EC-13(1), 14–17 (1964)
5. Madrid, P.E., Millar, B., Swartzlander, E.E.: Modified booth algorithm for high radix multiplication. In: IEEE International Conference on Computer Design: VLSI in Computers and Processors, pp. 118–121 (1992)
6. Ismail, R.C., Hussin, R.: High Performance Complex Number Multiplier Using Booth-Wallace Algorithm. In: IEEE International Conference on Semiconductor Electronics, pp. 786–790 (2006)
7. Sjalander, M., Larsson-Edefors, P.: High-Speed and Low-Power Multipliers Using the Baugh-Wooley Algorithm and HPM Reduction Tree. In: IEEE International Conference on Electronics, Circuits and Systems, pp. 33–36 (2008)
8. Kozyrakis, C.E., Patterson, D.A.: Scalable, vector processors for embedded systems, Micro. IEEE Journals and Magazines 23(6), 36–45 (2003)
9. Sjalander, M., Larsson-Edefors, P.: Multiplication acceleration through twin precision. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 17(9), 1233–1246 (2009)
10. Kim, H., Somani, A.K., Tyagi, A.: A Reconfigurable Multifunction Computing Cache Architecture. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 9(4), 509–523 (2001)
11. Guo, J.I., Liu, C.M., Jen, C.W.: The efficient memory-based VLSI array design for DFT and DCT. IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process 39(10), 723–733 (1992)
12. Chiper, D.F.: A systolic array algorithm for an efficient unified memory-based implementation of the inverse discrete cosine transform. In: IEEE International Conf. Image Process, pp. 764–768 (1999)
13. Chiper, D.F., Swamy, M.N.S., Ahmad, M.O., Stouraitis, T.: Systolic algorithms and a memory-based design approach for a unified architecture for the computation of DCT/DST/IDCT/IDST. IEEE Trans. Circuits Syst. I, Reg. Papers 52(6), 1125–1137 (2005)

14. Vinnakota, B.: Implementing Multiplication with Split Read-only Memory. IEEE Transactions on Computers 44(11), 1352–1356 (1995)
15. Mora-Mora, H., Mora-Pascual, J., Sanchez-Romero, J.L., Chamizo, J.M.G.: Partial product reduction by using look-up tables for $M \times N$ multiplier. Integration, the VLSI Journal 41, 557–571 (2008)
16. Meher, P.K.: New Approach to Look-Up-Table Design and Memory-Based Realization of FIR Digital Filter. IEEE Trans. Circuits Syst. II, Regular Papers 57(3), 592–603 (2010)