International Journal of Computer Science and Network (IJCSN)
Volume 1, Issue 2, April 2012 www.ijcsn.org ISSN 2277-5420

High-speed Multiplier Design Using Multi-Operand Multipliers

[1,2]Mohammad Reza Reshadi Nezhad, [3]Kaivan Navi

[1] Department of Electrical and Computer engineering, Shahid Beheshti University, G.C., Tehran, Tehran 1983963113, Iran

[2] Faculty of Department of Computer engineering, University of Isfahan, Isfahan, Isfahan 8174673440, Iran

[3] Department of Electrical and Computer engineering, Shahid Beheshti University, G.C., Tehran, Tehran 1983963113, Iran

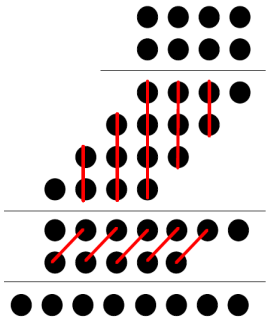
Abstract
Multiplication is one of the major bottlenecks in most digital computing and signal processing systems, which depends on the word size to be executed. This paper presents three deferent designs for three-operand 4-bit multiplier for positive integer multiplication, and compares them in regard to timing, dynamic power, and area with classical method of multiplication performed on today architects. The three-operand 4-bit multipliers structure introduced, serves as a building block for three-operand multipliers in general

Keywords: Dadda's multiplier, digital multipliers, fast multipliers, parallel multipliers, Wallace's multipliers.

1. Introduction

Multipliers are used in most arithmetic computing systems such as 3D graphics, signal processing, and etc. It is inherently a slow operation as a large number of partial products are added to produce the result. There has been much work done on designing multipliers [1]-[6]. In first stage, Multiplication is implemented by accumulation of partial products, each of which is conceptually produced via multiplying the whole multi-digit multiplicand by a weighted digit of multiplier. To compute partial products, most of the approaches employ the Modified Booth Encoding (MBE) approach [3]-[5], [7], for the first step because of its ability to cut the number of partial products rows in half. In next step the partial products are reduced to a row of sums and a row of caries which is called reduction stage. There are different schemes to be used in this step such as: Wallace trees [6], [7] or taking the advantages of compressor trees like [5], [8], [9] to reduce the number of partial products to two rows of sum and caries. In this reduction, one could consider using high speed carbon nanotube full adders to ensure a faster, low power consumption design [10]-[12], which is a new document promising technology for coming years. Finally in the last stage, using some adder approach [13], [14], to add the two rows of step two and compute the final product. Most recent publications have focused on reduction of partial products to achieve better multipliers [3], [4], [9], in other words, they have tried to optimize the second stage of multiplication to design a faster multiplier.

Fig. 1 illustrates the three steps involved as discussed above for a *4* by 4 bit multiplication. This is down by 4^2



bitwise products $x_i y_j$ (logical AND terms) and then using bit reduction and a final addition [13].

Fig. 1. Dot notation of a 4 by 4 bit multiplication

In this paper, we offer the design details of a three-operand multiplier in three different methods that is proposed. Robert McIlhenny and Miloˇs D. Ercegovac [15] introduced implementation of three-operand

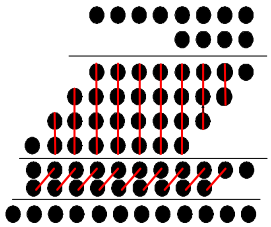
multipliers, and proposed three different methods in their implementation of three-operand multiplier: (1) cascade method; (2) ROM method; and (3) their proposed method. The cascade method consists of two multipliers in series, the first one multiplies the two 4-bits operands and the result which is 8-bits is then multiplied by the third 4-bit operand and 12-bit product is computed. The total delay using this method is equal to the delay of 14 exclusive or gates, which is shown by 14δXOR. The ROM method presented in their paper, consisting of utilizing the operands to address 256 by 8-bit ROM modules and producing the appropriate table-lookup result. The delay corresponding to this method was calculated and stated equal to 12δXOR. In their proposed method, they used Initial two-level recoding for three-operand multiplication. At the first stage of the proposed approach, the four bits of one operand are recoded, and the four bits of another operand are used to select the appropriate partial product bits. This generates two 5-bit words. At the second stage, the four bits of the third operand are recoded, and the bits of the two 5-bit words are used to select the appropriate new partial product bits. This generates four 6-bit words. Thus the total number of partial product bits generated is 24. The third stage consists of array reduction with height of 4 which needs a 4 to 2 compressor. In the last stage, a carry propagation adder is used to compute the final result. This method also has a delay of 12δXOR.

The outline of the paper is as follows. Section 2 gives the fundamental aspects of two-operand multipliers. In section 3 we will propose three models of three-operand multiplier. Then, section 4 represents results, including latency, area, and power for the proposed designs. This section is dedicated to comparisons of proposed designs against two-operand multipliers which we call it classical multiplier, where four different multipliers are synthesized based on FPGA technology. The target technology is a Xilinx Virtex5 FPGA. Finally, section 5 contains our concluding remarks.

2. Tow –Operand Multiplier

Most contributions have been made to design of multi-operand addition and parallel multiplication [1], [4], [6]. As mentioned in previous section, three-operand multipliers were presented in [15].

In this paper, we emphasis on three-operand multipliers and for future works we will extend our work to multi-operand multiplication. But here, we first show how a two operand multiplier works. The multiplication of two unsigned binary numbers X and Y, where $X=x_{n-1} \ldots x_1 x_0$ and $Y= y_{n-1} \ldots y_1 y_0$, then the product p is computed as $P= p_{n-1} \ldots p_1 p_0$. The architect for a 4-bit multiplier is shown in fig. 1. Now, if it is desired to multiply the result by a third operand, we need a *m* by *n* multiplier architecture to



do the task. The dot notation architect for an 8 by 4 bits multiplication is shown in figure 2, and the result multiplication is 12 bit long.

Fig. 2. Multiplication of third operand by the result of first and second operand multiplication

Let's suppose δ is used to represents the delay of a component in a given architecture. For a *n* by *n* bit multiplier we drive an expression to indicate the latency of the circuit. As mentioned before each multiplication consists of three stages. The delay of the first stage is equal to latency of an AND gate which is computed by δ(AND). The second stage which is called $\lceil \log_2 n \rceil * \delta(4:2)$, $\lceil \log_2 n \rceil$ partial product reduction stage has a delay of in which, is the hight of computed partial products, and δ (4:2) is the delay of a 4 to 2 compressor. The last stage

$$T_1 = \delta(AND) + \lceil \log_2 n \rceil * \delta(4:2) + \delta_{CPA}(2n-3) \quad (1)$$

delay corresponds to latency of a carry propagation adder circuit which is computable by $\delta_{CPA}(2n-3)$ according to architecture shown in fig. 1. Total delay of a *n* by *n* bit multiplier is the sum of the delays computed for each stage of multiplication. Therefore, the corresponding delay of Fig. 1 is defined as T_1 and is shown in equation (1).

$$T_2 = \delta(AND) + \lceil \log_2 n \rceil * \delta(4:2) + \delta_{CPA}(3*n-3) \quad (2)$$

The result of a *n* by *n* bit multiplication is equal to *m*=2*n* bit. In order to have a three operand-multiplier, we have to multiply *m* bit by another *n* bit operand as it is shown in Fig. 2. The same procedure is down for this

$$T_{clasic} = 2*\delta(AND) + 2*\lceil \log_2 n \rceil * \delta(4:2) + \delta_{CPA}(2*n-3) + \delta_{CPA}(3*n-3) \quad (3)$$

$$Hight\ of\ partial\ products = \frac{3*n^2}{4} \quad (4)$$

multiplication to compute the total delay. Hence, the total delay for the $m \times n$ multiplier is denoted by T_2 and written as equation (2).

In order to calculate the latency of a three-operand multiplication in today's architectures, we have to add up the delay expression (1) and (2) to get the total delay. We name this delay as classic three-operand multiplier delay $T_{classic}$, which is shown in (3).

3. Proposed Three-Operand Multiplier

In this paper we introduce three different design implementations for three-operand multipliers. Figure 3 shows the general idea behind the three-operand n-bit multiplication.

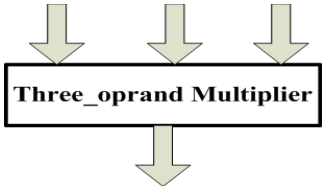


Fig. 3. Three-operand multiplier cell

As it is shown in the figure the architect has three separate inputs and in that block the partial products can be computed. Then, the partial product reduction is performed and, finally the carry propagation adder is used to compute the result. The schematic of the first design which, in this paper is referred to as proposed design I for 4-bit operands as a case study is depicted in figure 4.

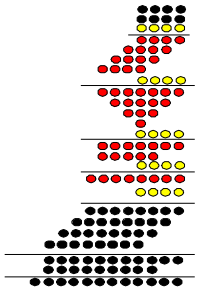


Fig. 4. proposed design I for three-operand multipliers

In this design, the first two operands are multiplied to each other and the result which is an eight bit long operand is calculated. Specifying that, the multiplications are performed in a whole cell, that is, the third operand is multiplied to the calculated result without of going out of the multiplication cell. The delay corresponding to this design can be calculated by equation (3), but because we perform the multiplications in a whole structure the synthesized results shows that its delay is better than what is expected.

The next implementation structure is proposed design II and is shown in figure 5. In this design we multiply the first two operands together and compute all the partial products. The trick is that we keep the partial products computed and multiply each bit of the third operand by the whole partial products as it is shown in the figure 5. It is easy to see that the final partial product for this design can be calculated by the use of 3-input AND gates. Using this design method we had to derive an expression to calculate the total delay of the proposed design. The delay of computing partial products is equal to 2δ(AND). In order to calculate the delay for reduction of partial products we had to come up with an expression to find the depth of partial products for any n-bit three-operand multiplier. This hight for any n-bit three-operand multiplier is given by equation (4).

$$Hight\ of\ \ partial\ products = 2*n \quad (6)$$

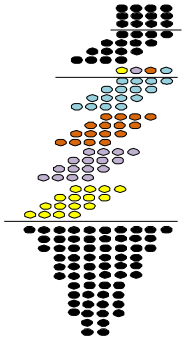


Fig. 5. proposed design II for three-operand multipliers

Knowing the hight of partial products, we are able to calculate the corresponding delay using 4 to 2 compressors. As it was done before multiplying (4) by delay of 4 to 2 compressor will give us the delay for reduction. Finally, the delay of carry propagation adder has to be calculated. By adding all the computed delays,

$$T_3 = 2 * \delta(AND) + \left\lceil \log_2\left(\frac{3*n}{4}\right)\right\rceil * \delta(4:2) + \delta_{CPA}(3*n-3) \quad (5)$$

we have expression (5) which calculates the latency of an n-bit three-operand multiplier using proposed architecture.

The last proposed implementation is named proposed design III and the dot product architecture of the design is depicted in figure 6. As it is shown, the first two operands are multiplied and the partial products are computed. Then in the reduction stage, the partial products are reduced to a row of sum and a row of carry. Following that, each bit of the third operand is multiplied by the two rows of sum and carry to build the final partial products. Finally, after reducing the partial products by the use of 4 to 2 compressors, we use an appropriate carry propagation adder to compute the result. To compute the latency of proposed architecture we have to talk the same steps taken in proposed design II. The depth of partial products after the second multiplication is given by equation (6).

Above equation shows the hight of partial product for any n-bit three-operand multiplier, using proposed design III architecture. The delay summation of each stage of the proposed multiplier is computed and is shown by equation (7).

$$T_4 = 2*\delta(AND) + \lceil \log_2(2*n)\rceil * \delta(4:2) + \delta_{CPA}(3*n-3) \quad (7)$$

Figure 6. proposed design III for three-operand multipliers

4. Delay, Area, and Power comparison

Comparison between n-bit classic three-operand and proposed n-bit Three-operand multiplier can be determined by subtracting the delays computed by each of the designs. Equation (3) is the corresponding delay for three-operand multipliers using classic method of multiplication, in today's architectures. Subtracting computed delay of each design from equation (3) would tell us which approach is faster. In case of proposed design I, as it was mentioned the delays are equal but because of cellular architecture used in proposed design I, we see that it is faster than classic method of multiplication. Subtracting equation (5) from (3) will tell us which approach is faster, comparing classic three-operand multiplication and proposed design II, and the difference is shown by equation (8). As it is evident from the derived equation, the proposed design II is faster by number computed by equation (8) with respect to classical method of multiplication.

$$T_{T_{classic}-T_3} = \left\lceil \log_2\left(\frac{4}{3}\right)\right\rceil * \delta(4:2) + \delta_{CPA}(2*n-3) \quad (8)$$

Performing the same procedure as proposed design II for proposed design III and subtracting equation (7) from (3), will give us the difference of the two equations. The

$$T_{T_{classic}-T_4} = \delta(4:2) + \delta_{CPA}(2*n-3) \quad (9)$$

resulted difference is shown in equation (9), which means that proposed design III is faster than classic multiplication by the value computed by equation (9).

For performance evaluation and comparison, we use logical effort and will show the delay of each proposed design. In this case, delay of an AND gate is delay of one gate shown by δ(AND), the delay of a 4:2 compressor is equal to 3 gates denoted by δ(4:2), and latency of a XOR is 2 gate delay, indicated by δ(XOR). In order to ease the comparison, figure 7 is produced to show the practical delay based on logical effort analysis. The figure 7 confirms that all the proposed designs have better delay compared to classical two-operand multipliers.

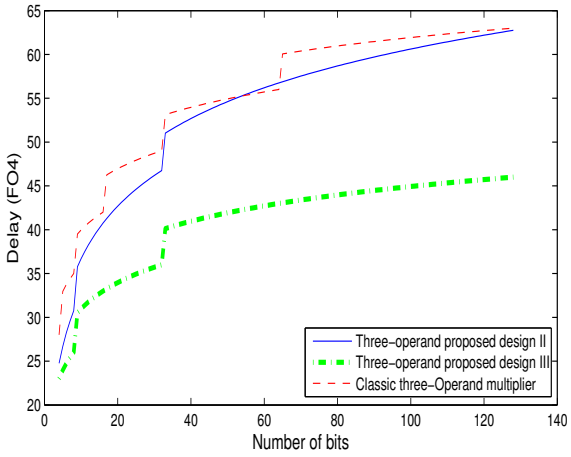
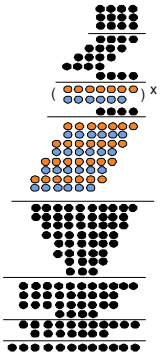


Fig. 7. Delay comparison of different proposed designs

However, to achieve precise estimations for area and delay, the proposed designs and other two-operand multipliers were described in VHDL, and implemented using FPGA technology. The target technology is a Xilinx

Virtex5 FPGA and the area is evaluated by the number of occupied slices. Table 1 compares the area and delay of proposed designs against classical three-operand multiplier.

Table 1: Implementation results of the three-operand multipliers on FPGA



	Fundamental parameters	
Multipplier	**Delay (*ns*)**	**Area (slices)**
Tow-operand-4×4	5.338	19
Tow-operand-8×4	9.087	51
Classic	14.425	70
Three-operand design I	10.692	71
Three-operand design II	9.708	92
Three-operand design III	8.516	68

In this table, the delays of two-operand 4×4 and two-operand 8×4 are added to come up with the delay of classical multiplier. Table 1 confirms that the proposed three-operand multipliers have better performance regarding latency, but ther is not noticeable improvement in the area parameter, which is expected. According to table 1 and also figure 7, proposed design III has a better performance regarding delay and area.

5. Conclusions

We have presented three simple, high performance and efficient n-bit three-operand multiplier architectures. The simulation results have confirmed that the delay and area improvement is reachable by the proposed multi-operand multiplier designs introduced. The presented results show that the design approach considered is a viable solution for high performance VLSI implementation.

References

[1] L. Dadda, "Some schemes for parallel multipliers", *Alta Frequenza*, vol. 34, 1965, pp. 349-356.

[2] A. D. Booth, "A Signed Binary Multiplication Technique", *Quarterly J. Mechanical and Applied Math.,* vol. 4, 1951, pp. 236-240.

[3] F. Elguibaly, "A Fast Parallel Multiplier-Accumulator Using the Modified Booth Algorithm", *IEEE Trans. Circuits and Systems,* vol. 47, no. 9, pp. 902-908, 2000.

[4] W. C. Yeh and C.-W. Jen, "High-Speed Booth Encoded Parallel Multiplier Design", *IEEE Trans. Computers*, vol. 49, no. 7,2000, pp. 692-701.

[5] J. Y. Kang and J. L. Gaudiot, "A Fast and Well Structured Multiplier", *EUROMICRO Symp. Digital System Design*, 2004, pp. 508-515.

[6] C. S. Wallace, "A Suggestion for a Fast Multiplier", *IEEE Trans.Computers*, vol. 13, no. 2, 1964, pp. 14-17.

[7] J. Fadavi-Ardekani, "M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees", *IEEE Trans. Very Large Scale Integration*, vol. 1, no. 2, 1993, pp. 120-125,.

[8] J. Y. Kang, W. H. Lee, and T. D. Han, "A Design of a Multiplier Module Generator Using 4-2 Compressor", *Fall Conf.*, vol. 16, 1993, pp. 388-392.

[9] V. G. Oklobdzija, D. Villeger, and S. S. Liu, "A Method for Speed Optimized Partial Product Reduction and Generation of Fast Parallel Multipliers Using an Algorithmic Approach", *IEEE Trans.Computers,* vol. 45, no. 3, 1996, pp. 294-306.

[10] K. Navi, A. Momeni, F. Shari, P. Keshavarzian, "Two novel ultra high speed carbon nanotube Full-Adder cells", *IEICE Electronics Express,* Vol. 6 No. 19, 2009, pp.1395-1401.

[11] K. Navi, Fazel Shari, Amir Momeni, Peiman Keshavarzian, "High Speed CNFET Full-Adder Cell Based on Majority Gates", *IEICE Electronics Express,* 2010, PP. 932-934.

[12] M. R. Reshadinezhad, M. H. Moaiyeri, K. Navi "An Energy Efficient Full Adder Cell Using CNFET Technology", *IEICE Electronics Express,* Vol.E95, o.4, Apr. 2012 to be published.

[13] B. Parhami, Computer arithmetic: algorithms and hardware designs, New York : *Oxford University Press*, 2000.

[14] W. Stenzel, W. Kubitz, and G. Garcia, "A compact high speed parallel multiplication scheme," *IEEE Transactions on Computers*, 1977, pp.948–957.

[15] R. McIlhenny, M. D. Ercegovac, "On the Implementation of a Three-operand Multiplier," *signals,systems & computers*, vol.2, 1997, PP. 1168 – 1172.

Mohammad Reza Reshadinezhad: He was born in Isfahan, Iran, in 1959. He received his B.S. and M.S. degree from the Electrical Engineering Department, University of Wisconsin Milwaukee, USA in 1982 and 1985,respectivly. He has been in position of lecturer as faculty of computer engineering in University of Isfahan since 1991. He is currently pursuing the Ph.D. degree in the school of Electrical and Computer Science, Shahid Beheshti University, Tehran, Iran. His research interests are digital arithmetic, Nanotechnology concerning CNFET, VLSI implementation and logic circuits.

Kaivan Navi: He received M.Sc. degree in electronics engineering from Sharif University of Technology, Tehran, Iran in 1990. He also received the Ph.D. degree in computer architecture from Paris XI University, Paris, France, in 1995. He is currently Associate Professor in Faculty of Electrical and Computer Engineering of Shahid Beheshti University. His research interests include Nanoelectronics with emphasis on CNFET, QCA and SET, Computer Arithmetic, Interconnection Network Design and Quantum Computing and cryptography. He has published over 50 ISI and research journal papers and over 70 IEEE, international and national conference paper.