An Optimised 3x3 Shift and Add Multiplier on FPGA

Aneela Pathan
PhD Scholar
[1]Institute of Information and Communication Technology
Mehran University of Engineering and Technology,
Jamshoro, Pakistan
[2]Department of Electronic Engineering,
Quaid-e-Awam University College of Engineering, Science and Technology (QUCEST), Larkana, Pakistan
pathan_aneela@quest.edu.pk

Tayab D Memon
Department of Electronic Engineering,
Mehran University of Engineering and Technology,
Jamshoro, Pakistan
tayabuddin.memon@faculty.muet.edu.pk

***Abstract*—Shift and add is conventional multiplication technique used at most, due to its simplest architecture. This simplicity becomes the bottleneck, when its hardware implementation takes more resources, when implemented on FPGAs. Though FPGA is, taken as an efficient implementation tool, but limited resources are the design hurdle observed many times. Optimizations is the way, opt, to design large circuits, especially whole system on chip (SOC), or network on chip (NOC) on this device. So many methods of multiplier optimization are found with some modifications in conventional methods along their implementation and testability on FPGA. In this paper, implementation of fixed point finite length 3x3 unsigned integer shift and add multiplier is shown, by introducing some changes in procedure. The modified version results in less resource utilizations in terms of Lookup Tables (LUTs) and produce less delay, due to less levels of logics, compared with conventional method.**

***Keywords* FPGA; Lookup Tables, Multiplier; Optimization.**

I. INTRODUCTION

Multiplication is an important fundamental function in arithmetic logic operation [1].It is a mathematical operation that at its simplest is an abbreviated process of adding an integer to itself a specified number of times [2]. But in convention, multiplication involves three main steps: Partial product generation; Partial product reduction; Final addition [18].

Computational performance of a DSP system is limited by its multiplication performance [3]. Typically, an instruction in a microprocessor involves three distinct steps: fetch, decode, and execute. In the design of multiplier, the fetching time is related to the size of multiplier and multiplicand. So, currently, multiplication time is still the dominant factor in determining the instruction cycle time of a DSP chip [4] and since, multiplication dominates the execution time of most DSP algorithms [5], therefore high-speed multiplier is much desired [6].

The old and straight forward method of multiplication is shift and adds. This is familiar, due to its simple approach. In this method, shifting is performed at first and then addition is carried out. The flow diagram of shift and add multiplier is given in Fig: 1. [7].

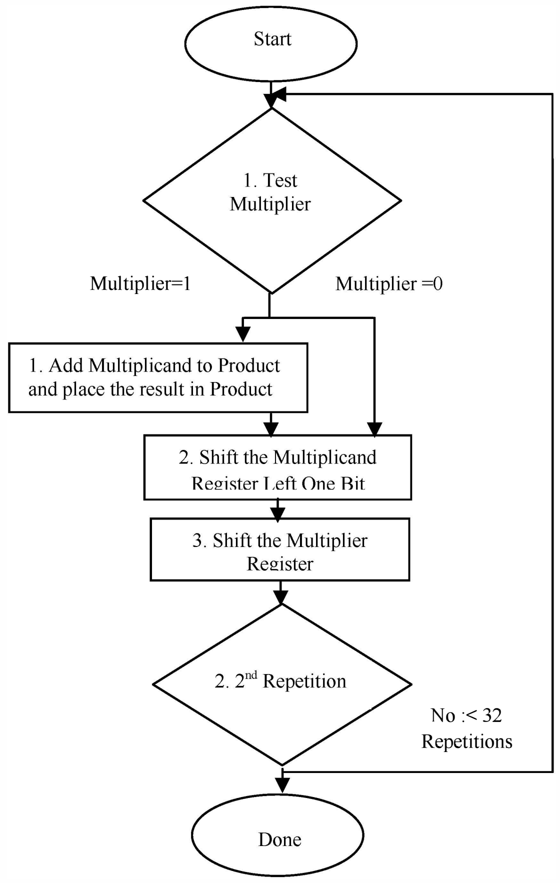
Start
1. Test Multiplier
Multiplier=1
Multiplier =0
1. Add Multiplicand to Product and place the result in Product
2. Shift the Multiplicand Register Left One Bit
3. Shift the Multiplier Register
2. 2nd Repetition
No :< 32 Repetitions
Done

Fig.1. Shift and Add Multiplier

Though this is found to be easy at one side, but when there come the hardware based implementation, it consumes more resources. It implements multipliers using logic; via shift and add operations [8]. Also traditionally shift and add algorithm has been implemented to design however this is not suitable for VLSI implementation and also from delay point of view [9]. This is specifically used when the number of DSP elements on an FPGA are exhausted or when we do not care a lot about throughput. [10]

Several works are reported on multiplier optimization with some modifications in conventional methods along their implementation and testability on FPGA. In [11] C.S. Wallace (1964), proposed a new fast parallel multiplying scheme in which sequential adding stages is to reduce the partial product accumulation. Moises E. Robinson and Earl Swartzlander, Jr. (1998) [12] proposed a reduction scheme of Wallace multiplier in which the reduction process of the first layer is redefined and uses a single 4:3 counter for optimizing the reduction process. In this work, when the number of bits of the input was equal to 5, 14, 20 or 29, the delay of the multiplier was reduced by 8-10% without increasing multiplier complexity. Similarly, Parth Mehta and Dhanashri Gawali (2009) [13] compared the conventional and the Vedic multiplier based on Urdhva Tiryakbhyam Sutra and turns out that both the multipliers have the same hardware expenses and the same speed. Vaijyanath Kunchigi et al. (2012) proposed a pipelined architecture based on vedic arithmetic. The purposed multiplier consists of 3 stages. First stage consists of 4 bit Vedic multiplier units, second stage consist of partial products reduction stage and the third stage consist of addition of the partial products. The multiplier architecture can be applied for large word length input such as 16 bit, 32 bit, 64 bit etc. and it was found out that the purposed multiplier is efficient in terms of speed and power when compared to multiplier such as array multiplier, booth multiplier etc. Jalil Fadavi Ardekani (1993) [14] designed a booth encoded [15], [16] parallel multiplier architecture in which the input are encoded into booth equivalent and a Wallace tree [11] is used to optimized the partial products. The partial products are then added by using a carry select adder. M.J. Liao et al. (2002) [17] presented a Booth encode Wallace tree multiplier using a partitioning carry select adder algorithm. They partitioned the carry select adders into number of blocks, so that, the overall delay is minimized. Experimentally, they found out that their architecture have an average of 9.12% less delay and with less than 1% overhead.

In this paper, implementation of fixed point finite word length 3x3 unsigned integer shift and add multiplier is shown, by introducing some changes in procedure. The modified version results is less resource utilizations in terms of Lookup Tables (LUTs) and produce less delay, due to less levels of logics, compared with conventional method. This paper is further divided as follows: Section II gives the understanding of proposed Algorithm. Section III is about FPGA Based Design of proposed Algorithm. Results are discussed in Section IV where as the work is concluded in Section V.

II. PROPOSED SHIFT AND ADD ALGORITHM

The binary representation gives eases of handling and manipulating the data. Like in signed number representation the negative number of any data may be achieved by analyzing the MSB. (1 at MSB position shows the negative number and 0 positive number). Similarly, the doubling of data may be achieved, just by appending zero in the last, (i-e after LSB) that is equal to shifting the data to right. Taking the benefits of these features, a modified shift and add algorithm is proposed here.

Decimal	Binary	2^n Form
0	000	--------
1	001	2^0
2	010	2^1
3	011	2^1+1
4	100	2^2
5	101	2^2+1
6	110	2^2+2^1
7	111	2^3-1
8	1000	2^3
9	1001	23 +1
10	1010	2^3+2^1
11	1011	2^3+2^1+1
12	1100	2^3+2^2
13	1101	2^3+2^2+1
14	1110	2^4-2^1
15	1111	2^4-1

Table 1: binary Representation of Data in 2^n form

In this proposed algorithm, binary data is represented in to 2^n for as shown in table 1. Here n is taken as the integer, which shows the number of shifts needed to give to particular data. As in conventional shift and add method, the data is shifted to the right, for everyone in the multiplicand value .Where as once the data is represented in 2^n form, all the n shifts are performed first, and then addition or subtraction is carried out if needed. This makes the overall shift and adds procedure easy. Proposed algorithm works as follows:

A. Algorithm

- For 3x3 case, the Multiplier and Multiplicand will take the values as; 0, 1 2, 3, 4, 5, 6, and 7.
- For the Multiplier value 0, the Product would result in zero.
- For Multiplier 1, the Product is same to the value of Multiplicand.

For other five values, 2 to 7 steps are here.

Step 1: For any value of Multiplicand, see if the multiplier may be written in 2^n form. If so, append n bits in the last to the multiplicand, to get the product.

For example: let the Multiplier be 2. This can be written in the 2^n form. (Here n=1 as 2^1=2.) . So give single shift to multiplicand to get the required output.

Example: Let multiplier be 2 and multiplicand 5. Binary, representation of 5=101. By appending the zero in the last will give 1010, that is equivalent to decimal value 10.

Step 2: see if the multiplier may be written in 2^n+1 form. In this case, add n zero in the last to the multiplicand, and then add the zero/s added multiplicand to the actual multiplicand value.

Example: Let multiplier be 5 and multiplicand 7. 5 may be written in 2^n+1 form, for n=2. ($2^2+1=5$). So here add 2 zero in the binary representation of 7 that is 11100. Now add that value in actual value, 111. We get:

$$\begin{array}{ccccccc} & & 1 & 1 & 1 & 0 & 0 \\ + & & & & 1 & 1 & 1 \\ & 1 & 0 & 0 & 0 & 1 & 1 \end{array}$$

Step 3: see if the multiplier may be written in $2^{n1}+2^{n2}$ form. Here is this case the value of n1 is different from n2. Append n1 zeros to multiplicand and add it with n2 zero appended multiplicand.

Example: Let multiplier be 6 and multiplicand 7. 6 may be written in $2^{n1}+2^{n2}$ form, for n1=2 and n2=1 ($2^2+2^1=6$). So in zero appended value, 1110. We get:

$$\begin{array}{ccccccc} & & 1 & 1 & 1 & 0 & 0 \\ + & & & 1 & 1 & 1 & 0 \\ & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

Step 4: see if the multiplier may be written 2^n-1 form. Then append n zeros to the multiplicand and subtract the actual value of multiplicand from it.

Example let the multiplier be 7. This may be written as 2^n-1; for n=3 ($2^3-1=7$). For the value of multiplicand be 6, we get the result as: 6=110; by adding 3 zero we get 110000. Now subtract the actual multiplicand from bits appended data.

$$\begin{array}{ccccccc} & 1 & 1 & 0 & 0 & 0 & 0 \\ - & & & & 1 & 1 & 0 \\ & 1 & 0 & 1 & 0 & 1 & 0 \end{array}$$

B. Flow Diagram

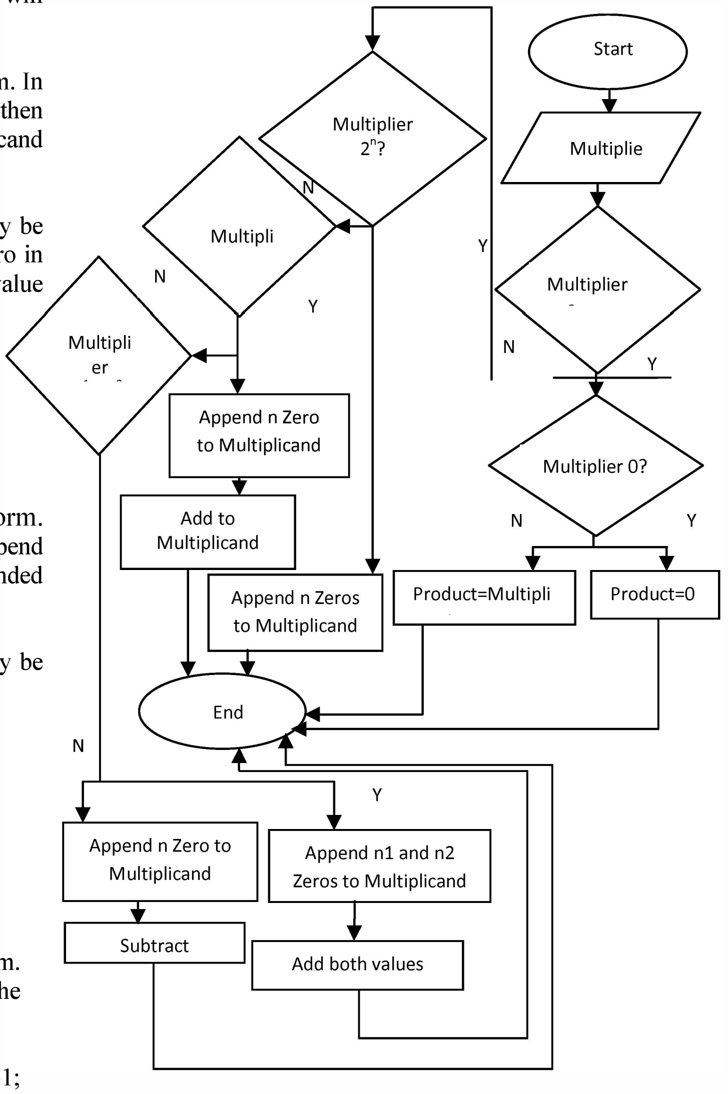

Fig: 2. Flow Diagram

III. FPGA BASED IMPLIMENTATION

Figure as under is the block diagram of proposed design being implemented on FPGA. As we can see that the figure shows only the multiplicand part, because, according to proposed algorithm all the operations are performed by multiplicand. The Multiplier only tells that what is done to the multiplicand.

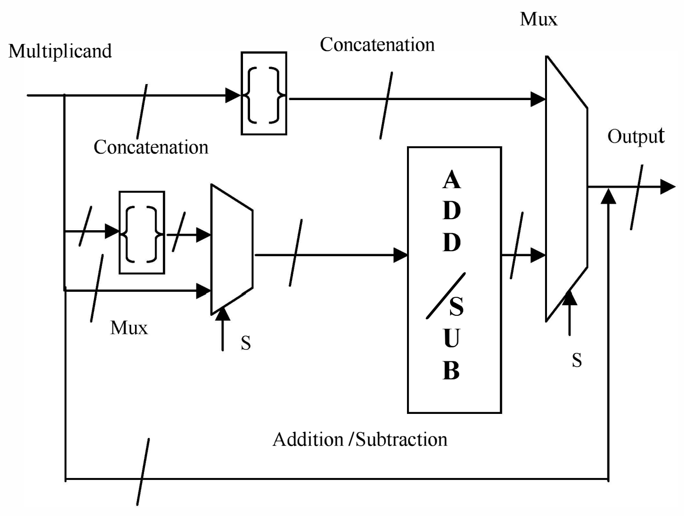

Fig: 3. FPGA Based Design Diagram

The three Logic Blocks of the design work as follow:

1. ***The Concatenation Operator:*** This is the vital part of the design. Most of the Multiplier functionality is achieved with the help of concatenation. This operator does not take any hardware recourse other than the wires. The two concatenations blocks work in parallel, hence producing no additional delay. The input to this block is 3 bit wide. The maximum bits, to be appended are restricted to 3, as the final output of the multiplier is 6 bit wide.

2. ***The Multiplexer:*** This block, works a selector for the data. The two inputs to the first multiplexer are the Multiplicand value and the bit/s concatenated data. The selection is based on the value of S bit. The first multiplexer is used to full fill the second, third and fourth step of the algorithm. The second step of the algorithm is to check if multiplier may be written in 2^n+1 form. In this case, the S bit will be set to one, and the multiplicand data will be passed to the ADD/SUB block, where the 2^n data will come from the upper concatenation branch and addition will be done. The third step tells, if the multiplier may be written in $2^{n1}+2^{n2}$ form. In this case, the S bit will be set to zero and n1 times concatenated data will be passed from the first multiplier and n2 concatenated data will come from upper branch and both the data will go to ADD/SUB block for being added. The forth steps tells, if the data may be written in to 2^n-1 form. Here the S bit will be kept to one, and the multiplicand data will be passed to ADD/SUB unit where the concatenated data will come from first branch. The second multiplexer is used to fulfill the first step of the algorithm. If the S bit is one, the 2^n concatenated data will pass to the output, if S is zero, then the algorithm will see the other three steps.

3. **Addition/Subtraction Block:** There are two input data to this block. One comes from the concatenation block and other from the multiplexer. This block works in two modes. Depending upon the Multiplicand, either this block adds two inputs, or subtracts the smaller input from the bigger one. The output of this block goes to another multiplexer, to be selected as a wanted output.

IV. RESULTS

The design was synthesized on Spartan 6 FPGA, using Xilinx ISE 13.2 version. .

Factor	Conventional	Proposed
LUTs	11 out of 1503	6 out of 1503
Macro Statistics	6-bit adder:2 2 to1multiplexer:3 (6 bit)	5-bit adder:1 6-bit addsub:1 2-to-1 multiplexer:1 (1 bit) 2-to-1 multiplexer :9 (6 bit)
Delay (ns)	9.049	5.684
Logic Level	7	3

Table 2: FPGA Based Implementation Results of Proposed Algorithm

The results given in the table are taken from the design synthesis report hence may vary a little, after performing post place and route step. In the initial phase, the parallel approach of the design was followed. I-e, a separate branch for each addition or subtraction was given, hence to reduce the sequential processing delay.

In the case of fixed point finite word length 3x3 unsigned integers shift and add multiplier, total two combinational circuits, one adder and one adder/subtractor is synthesized in the given approach. Also in conventional design approach two adders are used. Though the proposed algorithm did not reduce the adder/subsector, but the overall delay observed is less. (If the design is carried out in serial approach, one adder/subtractor would be needed) Also the results show that, the shift and Add Method of Multiplication occupies, almost the double number of Lookup Tables, where as the proposed algorithm saves the recourse by 45%. The delay observed in the design is reciprocal to the maximum frequency that may be achieved. Comparing the delay observed, we can calculate the frequency for conventional shift and add multiplier, to be 110 MHz approximately, where as the proposed design can give the frequency of about 175 MHz. The synthesis report generated after implementing the design using Xilinx Ise software shows the logic levels observed by the conventional design and proposed design. It may be seen that, the performance of proposed algorithm is much better than the conventional shift and add method. Whereas the number of multiplexer in proposed design is more that the conventional.

V. CONCLUSION

In this paper, we have presented a modified approach of shift and add multiplication. The example here taken was, fixed point finite word length 3x3 unsigned integer shift and add multiplier. Though; the number of adders, observed in both approaches is similar; whereas, substantial improved performance is obtained in the proposed algorithm in the sense of maximum operating frequency. This performance factor is highly expected to increase by increasing the order of multiplier and much difference should be observed in adder number utilization.

Point to the future is to carry out that at the large multipliers, by making some changes in the algorithm, proposed as above. Hence, in the design mentioned in this paper, the parallel approach is followed, that resulted in an individual adder/subtractor usage for each condition. While in future the approach will be made to use only single adder subtractor block for all branches. Also in the second phase of the design, along with higher order multiplier, the proposed design would be incorporated in adaptive multipliers.

Apparently, similarly, the comparative analysis of the modified algorithm will also be carried out, with other existing serial and parallel multipliers, like Daada, Both's and Wallace three algorithms.

REFEREANCES

[1] Vaidya, Sumit Dandekar, Deepak," DELAY-POWER PERFORMANCE COMPARISON OF MULTIPLIERS IN VLSI", International Journal of Computer Networks & Communications (IJCNC), Vol.2, No.4, July 2010 DELAY-POWER pp47-56.

[2] Laxman, S R, Darshan Prabhu Shetty, Mahesh Bm, Manjula Sharma, Chirag," FPGA Implementation of Different Multiplier Architectures", International Journal of Emerging Technology and Advanced Engineering Website: www.ijetae.com (ISSN 2250-2459, Volume 2, Issue 6, June 2012),pp 292-295.

[3] "ASIC Implementation of 4 Bit Multipliers" Pravinkumar Parate ,IEEE Computer society. ICETET,2008.25.

[4] Kiat-seng Yeo and Kaushik Roy "Low-voltage,low power VLSI sub system" Mc Graw-Hill Publication. Jong Duk Lee, Yong Jin Yoony, Kyong Hwa Leez and Byung-Gook Park "Application of Dynamic Pass Transistor Logic to 8-Bit Multiplier" Journal of the Korean Physical Society, Vol. 38, No. 3, pp.220-223,March 2001

[5] "Low power and high speed 8x8 bit Multiplier Using Non-clocked Pass Transistor Logic" C.Senthilpari, Ajay Kumar Singh and K. Diwadkar, 1-4244-1355-9/07, 2007, IEEE.

[6] Kiat-seng Yeo and Kaushik Roy "Low-voltage,low power VLSI sub system" Mc Graw-Hill Publication.

[7] Babulu, K Parasuram, G "FPGA Realization of Radix-4 Booth Multiplication Algorithm for High Speed Arithmetic Logics", K. Babulu et al/ (IJCSIT) International Journal of Computer Science and Information Technologies, ISSN:0975-9646 Vol. 2 (5) , 2011, 2102-2107.

[8] B.Parhami, Computer arithmetic algorithms and hardware designs, Oxford Univ. Press, 2000.

[9] Vaidya, Sumit Dandekar, Deepak," DELAY-POWER PERFORMANCE COMPARISON OF MULTIPLIERS IN VLSI", International Journal of Computer Networks & Communications (IJCNC), Vol.2, No.4, July 2010 DELAY-POWER pp47-5.

[10] Aly, Mohamed Sayed, Ahmed," A Study of Signed Multipliers on FPGAss", 2012 IEEE International Conference on Electronics Design, Systems and Applications (ICEDSA) , 33-38.

[11] Wallace, C.S., "A Suggestion for a Fast Multiplier", IEEE Trans.Electron. Compt., vol. 13, no. 1, pp. 14-17, Feb., 1964.

[12] Moises E. Robinson and Earl Swartzlander, Jr., "A Reduction Scheme toOptimize the Wallace Multiplier", IEEE Proc. Int. Conf., Comput. Design: VLSI in Comput. and Processors, pp. 122-127, Oct. 1998.

[13] Parth Mehta, Dhanashri Gawali, "Conventional versus Vedicmathematical method for Hardware implementation of a multiplier", IEEE Int. Conf. on Advances in Computing, Control, and Telecommun. Technologies, pp. 640-642, 2009.

[14] Jalil Fadavi Ardekani, "M x N Booth Encoded Multiplier Generator Using Optimized Wallace Trees", IEEE Int. Tran. on Very Large Scale Integration Sys., vol. 1, no. 2, pp. 120-125, June, 1993.

[15] A.D. Booth, "A Signed Binary Multiplication Technique," Jour. Of Mech. Appl. Math., vol. 4, pp. 236-240, Oxford University Press,1951.

[16] O.L. Macsorley, "High-Speed Arithmetic in Binary Computers", Proc. IRE, vol. 49, no. 1, pp. 67-91, Jan., 1961.

[17] M.J. Liao, C.F. Su, C.Y. Chang and Allen C.H. Wu, "A Carry-Select- Adder Optimization Technique for High Performance Booth-EncodedWallace-Tree Multipliers", IEEE Int. Symp. on Circuits and Sys., vol. 1, pp. 81-84, 2002.

[18] Nair.S, Khade.R.H., Saraf.A"Design and Analysis of Various 32bitMultipliers in an Approach towards a Fast Multiplier", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering, ISSN 2320 – 3765, Vol. 4, Issue 7, pp.6649-6658 July 2015.