An Approach to LUT Based Multiplier for Short Word Length DSP Systems

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

Aneela Pathan
[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

***Abstract*—Short Word Length (SWL) DSP systems offer good performance as they process less data- typically up to three bits. Short Word Length systems may be designed using the FPGAs. FPGAs come with many built-in primitives like Look-up tables, Flip-flops, additional Carry logic, Memories and DSP elements.All these primitives give alternative approaches for FPGA based system design. This paper presents a way to use the Look-up tables to design three bit (3×3) constant coefficient unsigned integral multiplier for Short Word Length DSP systems.Besides, the feasibility of using Block ram and DSP elements for Short Word Length DSP system (multiplier) is also carried out as an alternative implementation approach.Result suggests the proposed way be the better one when compared with other two implementations.**

Keywords—Built-in Core; Combinational Logic Blocks; Multiplier; FPGA.

I. INTRODUCTION

FPGAs consist of basic building blocks, including Look-Up Tables (LUT's), Flip-Flop's (FF's) and Additional Carry Logic, besides more complex on-chip block, as Memories (e.g., block RAM'S), DSP elements (e.g., Multiplier) and High Speed Transceivers [1].

Along with the power that FPGA consumes and the path delay observed, the overall resource usage is also an important measure of hardware cost [2].

It is better to use built-in primitives that give optimized performance when compared with implementations on configuration logic blocks (CLBs)-LUTs [3, 4]. But in all means, CLBs play a superior role in many implementations when an application needs more resources than on FPGA board. Like system on chip (SOC), network on chip (NOC), or even in a large DSP system design, constrains are observed if resources accede the available limit [5].

Besides, it is convenient to use the LUTS for the Short Word Length (SWL) DSP systems where the data is represented in three bits-as the design based on those complex primitives may result in resource wastage.

In SWL systems, the primary component is Sigma-Delta Modulator (SDM) that operates at higher oversampling ratio (OSR) than Nyquist rate, and was first reported in [6] for FIR filter coefficients optimization. Recently much work is reported on logic utilization in general purpose DSP algorithms through reduction in multiplier complexity- known as short word length (SWL) DSP systems, including LMS-like algorithms [7-9].

Besides SWL, literature reveals various implementations of conventional DSP systems, specially the multiplier, using the built-in DSP elements (DSP48) and block RAM.

For example, in [10] authors have presented FPGA based hardware architecture for floating point multiplier (single precision (SP),double precision (DP), double-extended precision (DEP) and quadruple precision (QP) implementation).In the design authors have used 25x18 DSP48E blocks available on the Xilinx Virtex-5 for the mantissa multiplication.

Similarly, in [11] author is proposing FPGA based hardware architecture for quadruple precision (QP) division arithmetic. Here the mantissa division is employed using a series expansion methodology integrated with a wide integer multiplier (further optimized for FPGA implementations) facilitating the built-in DSP blocks efficiently.

In [12] author has used DSP blocks in a compact FPGA-based micro-coded coprocessor for exponentiation in asymmetric encryption.

Memory based approach, yielding faster output uses memories (RAMs, ROMs) or Look-up tables (LUTs) that store pre-computed values and can be readout for multiplication operation [13]. The constant coefficient multiplier method and very well known Distributed arithmetic are the prime examples of memory-based multiplier methods [14]. For example, Xilinx in [15] has shown an efficient technique to implement fixed coefficient multipliers using Look-up tables (LUTs) of FPGA's .

This paper presents a way to use the Look-up tables (configured as memory blocks) to design three bit (3×3)

978-1-5386-5689-1/18/$31.00 ©2018 IEEE

constant co-efficient unsigned integral multiplier for Short Word Length DSP systems. Besides, the feasibility of using block RAM and DSP elements for Short Word Length DSP system (multiplier) is also carried out as an alternative implementation approach.

As the design is carried out for Short Word Length DSP systems, the length of multiplier and multiplicand is set to three bits (3×3) yielding them to be fixed in the range 0-7 (2^3=8). Furthermore, the coefficients are set to be unsigned integers; hence resulting in positive product values.

The major difference between the conventional constant coefficient memory based multiplier and proposed one is the amount of memory consumed. In the conventional (3×3) design for example, for each constant multiplier (0-7), respective product values would be pre-calculated and stored in eight block memories. While, in proposed design, only one LUT based memory module is consumed. This LUT based memory holds only the product values of the fixed coefficient multiplier 2 and for other coefficients same product values are modified at the output as per proposed algorithm steps.

II. Conventional and Proposed Algorithms

In subsequent paragraphs conventional and proposed algorithms are described.

A. Conventional Memory Based Multiplire Operation

Let X to be a positive binary number with word-length L; there can be 2^L possible values of X, and accordingly, there can be 2^L possible values of product C = A • X. Therefore, for memory-based multiplication, a LUT of 2^L words; consisting of pre-computed product values corresponding to all possible values of X is conventionally used.

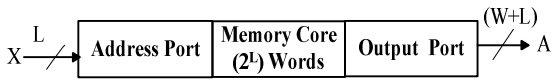

Fig.1. Typical approach to memory based multiplier

The product-word $A \bullet X_i$ is stored at the location whose address is the same as X_i for $0 \leq 2^L - 1$, such that if L-bit binary value of X_i is used as address for the LUT, then the corresponding product value $A \bullet X_i$ is available as its output [16-18].

TABLE I. Decimal and Binary Representation of the Data

Decimal Value	BR4BD	Decimal Value	BR4BD
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

BR4BD: Binary representation of 4 bit data

In the approach described above, the memory holds the product value of a constant multiplier X. If the X is kept changeable or adaptive, the total memory needed would depend upon the values which the variable X may take. For example, in our design X can run from 0 to 7; hence needing eight different memories.

Furthermore, in binary representation, new data may be obtained easily by just shifting the bits either left or right. Other goodness in base2 systems is that, the doubling of any value is easy to get by post fixing the zero as the LSB, as shown in table 1: moving from 2 to 4 in decimal is possible in binary with appending zero in the end of base2 representation of 2; similar is the case with moving from 3 to 6, 6 to 12, and 5 to 10 so on so forth.

This aspect gives us the opportunity to design memory based area optimized systems, especially multiplier, as with storing the pre-calculated product values of constant multiplier like 2 or 3 gives us the opportunity to get the product values of higher factors of constant multiplier. This scheme works very well for some data but issue is with the numbers whose least common factor is not available for example 5, 7, 9, 11.

B. Proposed Algorithm

In the proposed algorithm decimal 2 is taken as the least multiple of all the data.

TABLE II. Product Value for Multiplier 2 and Multiplicands 0-7

Multiplier	Multiplicand	Product Value
2	0	0
2	1	2
2	2	4
2	3	6
2	4	8
2	5	10
2	6	12
2	7	14

Hence, in memory pre-calculated product values for the constant multiplier 2 are stored and the product of all other multipliers (0, 1, 3, 4, 5, 6, and 7) is achieved by modifying the output in some ways using two combinational functions: not and concatenation.

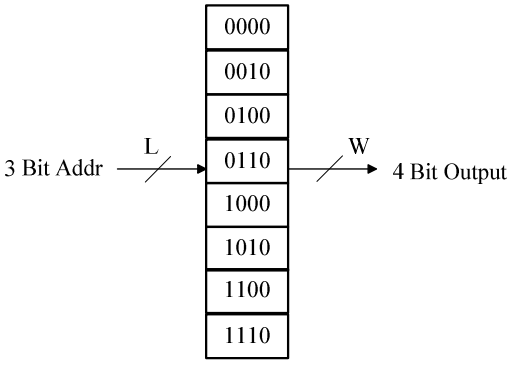

Fig.2. Block diagram of LUT based memory to store the product value

In Fig.2, LUT based memory, storing pre-calculated product values for constant multiplier 2 is shown. The memory address is 3-bit wide represented by W that has maximum 8 address locations with 2^n approach. Each product value stored is of four bits wide represented with L; hence, making total of 32 bits for total 8 addresses.

As for 3x3 multiplications the output should of 6 bits (W+L). But in our approach, we only need 4 bits; as shown in fig.2. Therefore, we can apply 2 bits at last as appendix to get the required output. The proposed algorithm for this type of multiplier is given as under.

Algorithm

This algorithm starts by storing product value in LUT memory where multiplier is 2 and multiplicand is 0, 1, 2, 3, 4, 5, 6, and 7. The Address is the value of Multiplicand.

Step 1: If the Multiplier is 2 and multiplicand is any value from the range 0-7, then the product value stored at that particular memory location (defined by multiplicand) is net output. But, if the Multiplier is other than 2 and multiplicand is from 0-7 then re-look in memory to find if the required output is already calculated. If yes, take the output from that particular address.

Example: let's suppose the multiplier be 2 and multiplicand be 5, then 2x5=10 and 10 $(10101)_2$ is already stored at memory location 5. But if the multiplier is not 2 and product value is still available in memory (like 3x4=12 $(1100)_2$ already stored at location 6), then simply get the product value at output stored at that particular location.

Step 2: If the Multiplier is other than 2 and the value is not available in memory; look for nearby value (one less or one greater of expected product) and flip the last bit.

Example: 3x5=15 and this value is not available in memory, so take the nearby value 14 $(1110)_2$ and flip the last bit to make 14$(1110)_2$ to 15$(1111)_2$.

Step 3: If the nearby value is not present, look for any least factor of the product value and append the bit(s) in the last. For example,4x5=20. Here 10 $(1010)_2$ is the least factor of 20 $(10100)_2$. So take the output 10 available at memory location 5 and append zero in the last to get the double of 10 that is 20.

Similarly, suppose multiplier is 3 and multiplicand is 7. Here the product would be 21$(10101)_2$. But 21 are not available in memory. So, this can be achieved by taking 10 $(1010)_2$ at the output and then appending 1 in the last. This will transform the 10 $(1010)_2$ in to 21$(10101)_2$.

Step 4: If above steps does not give the required output, append two bits in the last to get the required data. For example,in case of 7x5 resulting product value is 35 $(100011)_2$ and when we append $(11)_2$in the last to 8 $(1000)_2$, we can easily get the output 35(100011).

Flow diagram

The flow diagram for the addressing scheme and combinational logic is given as under:

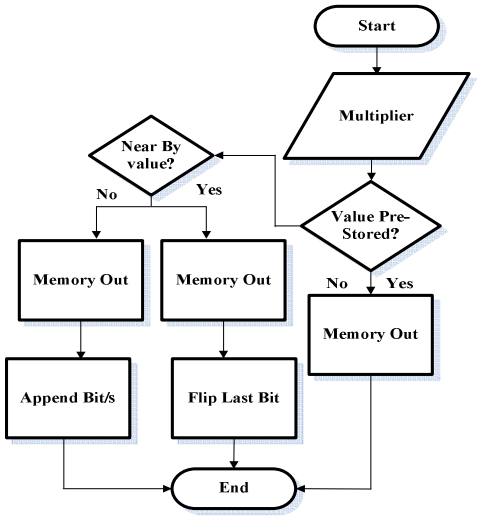

Fig.3. Flow diagram of proposed algorithm

III. SYSMEM DESIGN IN FPGA

Three FPGA based designs and implementations are discussed as follows:

Proposed LUT Based Design

In Fig.4, FPGA based system design is shown. It can be observed that the circuit consists of 4 functional elements: Memory module, Multiplexer, Not gate and Concatenation operator.

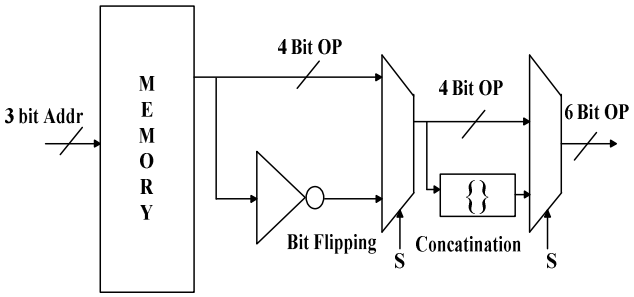

Fig.4. FPGA based system design

A. The Memory Module

The Look-up table based memory has the capacity of 32 bit. Total 8 product values of constant multiplier 2 are stored at 8 memory locations of Look-up table. The input to this memory module is generated by 3 bit address. The address is the multiplicand being selected from the range of 0 to 7 in the case of 3×3 multiplier.

B. The Multiplexer

In the design, two multiplexers are used. The function of first multiplexer is to select between the pre-calculated product and

the bit flipped value (when the nearby value of the product is present). The second multiplier is used if the algorithm comes in its third step, i-e, when the bit filliping does not work, but the bit concatenation is required in order to double the data.

C. The Not Gate

Amongst the two combinational Logics, one is the not gate. The function of not gate in this design is just to bring the flip in the last bit of the data taken from the LUT based memory. Flipping the last bit brings the effect of adding one in the memory output data.

D. The Concatenation Operator

The second combinational operation is to concatenate. As it is discussed above that for 3x3 multiplier the final output would be 6 bits. The word length of the data stored in the memory is 4. So, availing the opportunity to complete the final count to 6 bits, we can append 2 bits in the last of the data taken from the memory at maximum: to making the data double and even triple when required.

Block Ram based design

In block RAM based design, total six block RAMS were instantiated to store the product values for each constant multiplier 2, 3, 4, 5, 6 and 7. As it is obvious that for any multiplicand and zero multiplier the output would be zero and for one as multiplier the value of multiplicand will be the value of product. Hence, these two conditions were implemented using logic.

DSP48 based design

In this design strategy, the DSP48 block was instantiated to perform the required product. Here, contrary to above two implementations, the DSP48 was not restricted to carryout three bit multiplication.

IV. SIMULATION RESULTS

In this work, three different three bit (3×3) constant coefficient unsigned integral multiplier were designed; two using FPGA built in primitives a) Block Ram and b) DSP48 based Multiplier, and third with proposed design using LUT based implementation.

The designs were carried out on Xilinx Spartan 6 FPGA using Xilinx ISE 13.2. The implementation results of three different strategies are given in table 3.

In the design of block RAM multiplier, total of 8 memory modules are needed logically for 3x3 multiplier (considering 0, 1, 2, 3, 4, 5, 6, 7 as constant multipliers), but the design consists only 6 memory modules-as, for case of 0 and 1 the multiplier and multiplicand will be the result of product respectively.

TABLE III. RESULTS OF FPGA BASED DESIGNED MULTIPLIERS

Primitives	LUTs	DSP	RAM	Mux :	Max:delay (ns)	Freq: (MHz)
BRAM	3	0	6	0	4.372	228.72
DSP48	0	1	0	0	9.173	109.01
Proposed	9	0	0	85	3.648	274.12

Table 3 represents the simulation results of the designs. It is very much clear from the results that the proposed algorithm works very well in contrast to block RAM based designs. As, in that approach, even for the small bit width multipliers more memory elements are needed. Like in our design of 3x3 multiplier, six block RAMs are required to store all possible product results; while, our design consumes no block RAM. Furthermore, this design is also less efficient in terms of frequency achieved.

Similar is the case with DSP 48 based design. Though, this implementation consumes only one block, but results in resource wastage as DSP48 can work on large size of operands (18 bit log) but misfortunately for small size operands (similar to our case of 3 bit long) same resources are utilized.

Another important parameter is the delay observed in DSP48 based design that is much larger than the proposed design.

On other hands, the proposed design consumes 9 LUTs, but significant number of multiplexers; still it offering higher clock frequency and lower delay than other two designs.

V. CONCLUSION

This work presents three different implementations of three bit (3×3) constant coefficient unsigned integral multiplier for Short Word Length DSP Systems. All three designs were carried out using the built-in primitives of Xilinx Spartan 6 FPGA. The block RAM based design has the issue with the word length (as the bits of multiplier and multiplicand increase, the total memory would increase).

In the short word length systems, the concern with memory based design is the amount of minimum memory that we can configure in a given FPGA. For example, 32 bit memory needed to store the product values of constant multiplier 2 (considering eight multiplicand, i-e, 0-7) would be incorporated within no less then block RAM of 9K (The minimum configurable block Memory in Sprtan 6 FPGA); hence resulting in wastage of unused memory.

Similarly, as mentioned before, the DSP48 can effectively be used for larger multiplications; hence once used in Short Word Length Based system would result in in-efficient resource utilization.

So, to use memory based multiplication or using the DSP48 for Short Word Length systems is suggested as less feasible. Consequently, the choice is to use the customized (LUT based) implementations, as one proposed here.

The proposed multiplication algorithm may be used in any area of DSP, besides Short Word Length processing.

Point to the future work is to observe this multiplier by incorporating it in FIR Filter and Adaptive Filter design.

References

[1] C. Shi, J. Hwang, S. McMillan, A. Root, and V. Singh, "A system level resource estimation tool for FPGAs," *Field Programmable Logic and Application,* pp. 424-433, 2004.

[2] M. R. Singh and A. Rajawat, "A Review of FPGA-based design methodologies for efficient hardware Area estimation," *IOSR Journals (IOSR Journal of Computer Engineering),* vol. 1, pp. 1-6.

[3] C. Lavin, M. Padilla, S. Ghosh, B. Nelson, B. Hutchings, and M. Wirthlin, "Using hard macros to reduce FPGA compilation time," 2010, pp. 438-441.

[4] A. Palchaudhuri and R. S. Chakraborty, "A Fabric Component Based Approach to the Architecture and Design Automation of High-Performance Integer Arithmetic Circuits on FPGA," in *Computational Intelligence in Digital and Network Designs and Applications*, ed: Springer, 2015, pp. 33-68.

[5] A. Corporation, "AN 584: Timing Closure Methodology for Advanced FPGA Designs," 2014.12.19.

[6] N. Benvenuto, L. Franks, and F. Hill Jr, "Dynamic programming methods for designing FIR filters using coefficients-1, 0 and+ 1," *Acoustics, Speech and Signal Processing, IEEE Transactions on,* vol. 34, pp. 785-792, 1986.

[7] A. Z. Sadik and Z. M. Hussain, "Short word-length LMS filtering," 2007, pp. 1-4.

[8] T. D. Memon, P. Beckett, and A. Z. Sadik, "Power-area-performance characteristics of FPGA-based sigma-delta fir filters," *Journal of Signal Processing Systems,* vol. 70, pp. 275-288, 2013 2013.

[9] A. C. Thompson, *Techniques in Single-Bit Digital Filtering*: RMIT University, 20040., 2004.

[10] M. K. Jaiswal and H. K.-H. So, "DSP48E efficient floating point multiplier architectures on FPGA," in *VLSI Design and 2017 16th International Conference on Embedded Systems (VLSID), 2017 30th International Conference on*, 2017, pp. 1-6.

[11] M. K. Jaiswal and H. K.-H. So, "Architecture for quadruple precision floating point division with multi-precision support," in *Application-specific Systems, Architectures and Processors (ASAP), 2016 IEEE 27th International Conference on*, 2016, pp. 239-240.

[12] L. Rodriguez-Flores, M. Morales-Sandoval, R. Cumplido, C. Feregrino-Uribe, and I. Algredo-Badillo, "A compact FPGA-based microcoded coprocessor for exponentiation in asymmetric encryption," in *Circuits & Systems (LASCAS), 2017 IEEE 8th Latin American Symposium on*, 2017, pp. 1-4.

[13] K. Shanthi and N. Nagarajan, "HIGH SPEED AND AREA EFFICIENT FPGA IMPLEMENTATION OF FIR FILTER USING DISTRIBUTED ARITHMETIC," *Journal of Theoretical and Applied Information Technology,* vol. 62, 2014.

[14] H. P. Singh, R. Sarin, and S. Singh, "Implementation of high speed FIR filter using serial and parallel distributed arithmetic algorithm," *International Journal of Computer Applications,* vol. 25, pp. 26-32, 2011.

[15] K. Chapman, "Fast integer multipliers fit in FPGAs," *EDN magazinee's Desighn Ideas, www. ednmag. com,* 1993.

[16] A. Srinivasalu and G. R. Reddy, "Optimization of memory based LUT Multiplier," *Research and Development (IJECIERD),* vol. 3, pp. 125-132, 2013.

[17] B. Jeevanarani and T. Sreenivas, "Memory− Based Realiza− tion of FIR Digital Filter by Look− Up} Table Optimization," *International Journal of Engineering Research and Appti− cations, vQl,* vol. 2, pp. 1003-1009, 2012.

[18] R. H. Turner and R. F. Woods, "Highly efficient, limited range multipliers for LUT-based FPGA architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems,* vol. 12, pp. 1113-1118, 2004.