FPGA based area-power-performance analysis of LMS filter using conventional and proposed multipliers

A Pathan¹,², T D Memon³,⁴, F Khan Sohu¹

¹Department of Electronic Engineering, Quaid-e-Awam University of Engineering, Science and Technology (QUEST), Larkana, Pakistan
²Institute of Information and Communication Technologies, Mehran University of Engineering and Technology, Jamshoro, Pakistan
³Department of Electronic Engineering, Mehran University of Engineering and Technology, Jamshoro, Pakistan
⁴National Center for Robotics and Automation, HHCMS Lab, MUET, Jamshoro, Pakistan

pathan_anseela@quest.edu.pk

Abstract. This paper compares LMS filter-based adaptive channel equalizer, implemented on Vertex 7 FPGA using several existing and proposed multipliers. The three main performance parameters taken into consideration are the consumed resources (in terms of lookup tables), utilized power (among various design components), and the performance achieved (in terms of observed frequency). The results show the area-power-performance tradeoff amongst the conventional and proposed multipliers based designs. It is to mention that the proposed multipliers result in more compact systems due to less utilization of resources in all means.

1. Introduction

VLSI is good for rapid prototyping and fast implementation of complex DSP functions at one side; on the other hand, it is a challenging task to bring out those complex DSP systems into compact form along with optimal performance and minimal power usage, especially battery dependent portable devices like mobile phones. The researchers around the world have accepted that challenge and are working to give the preeminent solutions [1-4] to fulfill huge competing requirements [5].

In Digital Signal Processing (DSP), FIR, IIR, FFT and others need multipliers in their implementation and despite of various optimization techniques still the multiplier consumes more resources, power, and area along with observing more delay [6].

Some of well-known multiplier encoding techniques reported for simplifying the multiply and accumulate (MAC) operation are the Booth’s Algorithm [7], Wallace Tree Multiplier [8], DADDAD Multiplier [9], and Vedic Multiplier [10].

In FPGA based design, three main factors that play an important role are the targeted FPGA architecture, Electronic Design Automation (EDA) tools, and design techniques employed at the algorithmic level using hardware description languages [11].
The algorithm level optimization results in overall resource optimization along with good performance in terms of achieved frequency.

Some algorithm level optimized architectures of multipliers are already proposed and published by authors [12, 13] but those are limited to 3X3 multipliers.

The novelty of the paper is to extend the proposed multipliers to 8X8 and use them in implementing LMS based adaptive channel equalizer and compare the results with conventional multipliers based implementation.

The comparison is done based on some parameters including the Lookup tables, built-in adder/subtractor cores, multiplexer primitives, memory module, and DSP blocks. Also the performance characteristics listed as the delay observed, achieved frequency, logic levels and power distribution are also compared and discussed.

The results show the area-power-performance tradeoff amongst the conventional and proposed multipliers based designs. It is to mention that proposed multipliers result in more compact systems due to less utilization of resources in all means and observe the complexity level of O (n) that is not evident in some conventional multiplier algorithms. The paper further proceeds as follows: in section II proposed multipliers for LMS filter-based adaptive channel equalizer are given and discussed. In section III, FPGA based implementation results of channel equalizer using conventional and proposed multiplier architectures are compared and analyzed. Power analysis is done in IV; whereas paper is concluded in section V.

2. Proposed multipliers for LMS filter

2.1. An Optimization in Conventional Shift and Add Multiplier

Due to very simple approach, the shift and add is known as one of the easiest ways of multiplying two values.

In this method, depending on the value of the multiplier’s LSB bit, a value of the multiplicand is added and accumulated [14].

At each clock cycle, the multiplier is shifted one bit to the right and its value is tested. If it is a 0, then only a shift operation is performed. If the value is a 1, then the multiplicand is added to the accumulator and is shifted by one bit to the right.

A new approach of multiplying two 3×3 unsigned integers is proposed in [13] by the other by using the modified shift and add approach along its Sparta-6 FPGA based implementation.

That work is further extended for a 8×8 multiplier and is submitted for publication.

![FPGA Based Design Diagram](image)

**Figure 1.** FPGA Based Design Diagram

The proposed multiplier works with operands, be in base2 format, because binary data representation produce an ease in data handling and mitigating need of arithmetic for multiplication of
certain pair of operands (like to multiply any data with a constant value $2^k$ is as easier as just to append a binary zero to the multiplicand; whereas for all higher powers of 2, append the number of zeros equal to that power \[15\]. The algorithm for the proposed multiplier is given as under:

**Algorithm**
- **Case 1:** 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.
- **Case 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 add it with actual multiplicand value.
- **Case 3:** See if the multiplier may be written in $2^n+2^m$ form. If so, append $n$ zeros to multiplicand and add it with $m$ zero appended multiplicand.
- **Case 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.

### 2.2. 3-Input Lookup Table Based Signed Multiplier

Signed multiplication is used in several arithmetic operations in Digital Signal Processing. For example, convolution, correlation, and filtering typically involve negative as well as positive coefficients \[16\].

Implementation of a signed multiplier on hardware like FPGA is possible in alternative approaches. One of the most efficient ways is to utilize the built-in core of signed arithmetic (like DSP 48) or to use the mount on adders/subtractor configuration to get the required functionality \[17\].

Whilst, it is also a fact that FPGA is a resource-limited device and hence in large designs (NOC, SOC), it would be obvious to utilize look-up tables along with built-in Ip cores \[12\].

Vertex 7 FPGA’s lookup tables may be configured to work as 2-input or even 3–input combinational or sequential circuit \[18\].This configuration choice leads to area optimized implementation of any arithmetic operation, especially the multiplier.

The authors have proposed and submitted a design for a $8 \times 8$ signed multiplier that follows conventional shift and add method of multiplication \[13\].

After bottom to top approach of circuit design, at the first instant, $7 \times 2$ multiplier is designed and then it is further instantiated for the required number of times to perform $8 \times 8$ signed multiplications.

For designing $7 \times 2$ multiplier as a basic building block, we need a binary representation of 2-bit multiplier as given in Table I, and accordingly, the algorithm is four-step procedure discussed as below:

| Table 1. Decimal and binary representation of 2-bit data |
|--------------------------------------------------------|
| Based\(_{10}\) | Based\(_{2}\) |
| Representation | Representation |
|----------------|----------------|
| 0              | 00             |
| 1              | 01             |
| 2              | 10             |
| 3              | 11             |

**Algorithm**
- **Case 1:** If the multiplier is zero, the product value will be zero.
- **Case 2:** If the multiplier is one, the product value will be the same as that of the multiplicand.
- **Case 3:** If the multiplier is two, append the zero, at the end of multiplicand and hence that would be the net product value.
• **Case 4:** If the multiplier is three, append zero at the end of multiplicand and add it with the actual value of multiplicand.

The first three steps are possible with a simple multiplexer; whereas, to perform addition, that is obvious in step four, 3-input lookup tables are hardcoded as per conventional logic of addition and added serially to get 8x8 multiplier.

LUT3 # (  .INIT(8'b10010110) // Specify LUT Contents ) LUT3_inst3 (  .O(S0), // LUT general output .I0(C1), // LUT input .I1(X[0]), // LUT input .I2(Y[0]) // LUT input );

**Figure 2.** 3-Input LUT Hard-Coded for Sum Output

2.3. **FPGA’s Dual Port ROM Based Multiplier**

FPGA’s onboard Block memory may be programmed as a single or dual-port RAM/ROM module that leads to area efficient implementation of memory-based systems.

The research shows that in memory-based multiplier design, one of the operands is kept constant and hence leading the design to constant-coefficient multiplication.

To get out of this limitation and effectively utilizing the available resources, we have proposed Vertex 7 XC7vx330tffg1157 FPGA’s dual-port ROM based implementation of 8x8, unsigned integral multiplier that may be used in many simple to complex DSP applications.

In memory-based multiplier, pre-calculated product values are easy to obtain in minimal processing time, as they are stored at particular addresses masked with operands [12].

For a simple 2 × 2 multiplier the concept is as follows:

Let M, N be 2-bit operand, leading M0, N0 as LSB and M1, N1 be MSB bits of multiplicand and multiplier consecutively. Then two partial products, P1 and P2 are generated accordingly as an intermediate stage, and these partial products P1 and P2 are then added to get the final result. It may be observed that in the case of 2-bit operands, we need one full adder cascaded to one-half adders for summing P1 and P2.

\[
\begin{array}{c|c}
M_1 & M_0 \\
\hline
X & N_1 \end{array}, \begin{array}{c|c}
N_0 & M_1 \\
\hline
N_0M_1 & N_0M_0 PP1 \\
\hline
N_1M_1 & N_1M_0 X PP2 \\
\end{array}
\]

**Figure 3.** Conventional Multiplication Method

If this concept is extended to 8x8 multiplier we need 8 partial products P1, P2 . . . P8, and one 9-input 15-bit adder to add P1-P8 hence the circuit becomes a bit complex.

The same result may be obtained if we reduce the operand length from 8x8 to 4x4 and then perform some intermediate arithmetic [20].
And hence at the cost of extra adders, to replace a multiplier, the ROM size may further be reduced [1].

![Dual Port Rom Configuration](image)

**Figure 4.** Dual Port Rom Configuration

As shown in the figure, two addresses A and B, each 8 bit wide are given as an input to BROM along with a single common clock.

The address values are created as per partial product generation.

Each address is 8-bit wide (Four bits from the multiplier and four from multiplicand) and is given to memory module at port: Address A, and Address B consecutively and as consequences, we get two pre-calculated product values at ports Dout A and Dout B.

![General Architecture of Proposed Design](image)

**Figure 5.** General Architecture of Proposed Design

3. FPGA Based Implementations of LMS Algorithm Using Various Multipliers

In optimized FPGA based implementation count, besides consumed more lookup tables or achieved good frequency, one of the most worth considering parameter is the macro statistics. This tells about the consumption of available onboard resources, other than the familiars, like memory modules, clock, or DSP cores.

Many conventional and proposed multiplier architectures were incorporated in the design of LMS filter-based adaptive channel equalizer. To make the design easier, only a single coefficient filter was taken into account.

The implementation results are given in Table 2.
From the results, it may be observed that in terms of achieved frequency the Pipeline multiplier based design is better, but it consumes more worth-full resources, like DSP48 and the registers. The same is for the booth multiplier, as it gives good frequency at the cost of more adders and subtractors when compared to the proposed signed multiplier.

The proposed shift and add multiplier based design is also better than its conventional version in terms of consumed resources.

Table 2. FPGA based implementation results of LMS filter with 8 bit proposed multipliers

| Multiplier               | Macro Statistics                  | LUTS | Delay (ns) | Logic Level | Frequency (MHz) |
|-------------------------|-----------------------------------|------|------------|-------------|-----------------|
| Pipeline                | Add/Sub :2 Registers :152         | 40   | 2.624      | 10          | 381.134         |
|                         | Mux:4 DSP48E1s:4                  |      |            |             |                 |
| Proposed Memory Based   | Add/Sub:4 Registers:37 Mux:16     | 275  | 8.171      | 12          | 122.384         |
| Booth                   |                                    |      |            |             |                 |
| Proposed Signed         | Add/Sub:4 Registers:41 Mux:44     | 507  | 6.440      | 22          | 155.273         |
| Shift and add           | Add/Sub:32 Registers:13           | 690  | 5.62       | 33          | 177.81          |
| Proposed Shift and Add  | Mux:32                            |      |            |             |                 |
| Wallace Tree Serial     | Add/Sub:4 Registers:5 Xor:440     | 686  | 8.725      | 21          | 114.610         |
| Array                   | Add/Sub:4 Registers:5 Xor:416     | 823  | 11.98      | 27          | 83.475          |

Hence the results indicate that when the area is of concern, all three proposed approaches perform well, but when high-frequency systems are needed the Pipeline is a more obvious choice.

4. Power Analysis of the Design

Power requirements for programmable logic devices are highly dependent on the logic and functionality of the design. Early in the design cycle, we can enter the estimated activity and resources needed for our design in the Xilinx Power Estimator (XPE), which allows us to ensure that our design meets power requirements.

After synthesis and implementation, we can use the XPower Analyzer to get a detailed view of the power distribution for our design, broken down into individual device elements. Besides, we can import exact design activity from simulation. This allows us to determine whether the design meets power requirements.

Taking the benefit from this flavor of Power analysis, Tables III-IV show the power consumed in various elements of the LMS based adaptive filter for channel equalizer.

From the results, it may be observed that in terms of achieved frequency the Pipeline multiplier based design is better, but it consumes more worth-full resources, like DSP48 and the registers.

The same is for the booth multiplier, as it gives good frequency at the cost of more adders and subtractors when compared to the proposed signed multiplier.

The proposed shift and add multiplier based design is also better than its conventional version in terms of consumed resources.

Hence the results indicate that when the area is of concern, all three proposed approaches perform well, but when high-frequency systems are needed the Pipeline is a more obvious choice.
Table 3. Power estimation of LMS based adaptive channel equalization using various multipliers

| Multiplier       | Pipeline | Array | Wallace | Serial | Shift and | Proposed | Booth | Proposed | Signed Multiplier | Proposed | Dual Port Rom Based Design |
|------------------|----------|-------|---------|--------|-----------|----------|-------|----------|---------------------|----------|----------------------------|
|                  |          |       |         |        |           |          |       |          |                     |          |                            |
|                  |          |       |         |        | Add       | Add      |       | Add      |                     |          |                            |
|                  | On-Chip Power |        |         |        |           |          |       |          |                     |          |                            |
| Clocks           | 0.001    | 0.002 | 0.002   | 0.003  | 0.002     | 0.002    | 0.003 | 0.003    | 0.003               | 0.003    |                            |
| Logic            | 0.000    | 0.005 | 0.005   | 0.003  | 0.002     | 0.002    | 0.001 | 0.002    | 0.001               | 0.001    |                            |
| Signals          | 0.000    | 0.009 | 0.010   | 0.005  | 0.002     | 0.004    | 0.003 | 0.006    | 0.002               | 0.002    |                            |
| BRAMs            | -        | 0.082 | 0.038   | 0.081  | 0.076     | 0.096    | 0.045 | 0.004    |                     |          |                            |
| IOs              | 0.106    | 0.106 | 0.106   | 0.082  | 0.038     | 0.081    | 0.076 | 0.096    | 0.045               | 0.045    |                            |
| Leakage          | 0.143    | 0.144 | 0.144   | 0.143  | 0.143     | 0.143    | 0.143 | 0.144    | 0.143               | 0.143    |                            |
| Total            | 0.146    | 0.265 | 0.266   | 0.236  | 0.187     | 0.232    | 0.225 | 0.250    | 0.199               |          |                            |

It is obvious from the results that the Pipeline multiplier based design consumes less power and the Wallace based more. Whereas our proposed memory-based design has the power conception of order of Pipeline based design and also two other approaches are in an acceptable range.

5. Results and Conclusion
Being an emerging technology, FPGA is used to implement computation-intensive algorithms more efficiently in comparison to DSP or microprocessors [20, 21].

But limited resources are the design hurdle observed many times, and if the target is to get implementation of large systems, especially the whole system on chip (SOC), or network on chip (NOC) on the single device, it is required to optimize the design and make it compact [12].

So many methods of multiplication are found with some modifications in conventional methods along with their implementation and testability on FPGA.

This work shows the proposed area optimized and conventional multiplier algorithm based implementation of adaptive channel equalizer using LMS filter.

The conventional and proposed architecture was implemented in is Xilinx Vertex -7 FPGA with the EDA tool, Xilinx ISE 14.2.

The results indicate that when the area is of concern, in terms of macros or lookup tables, all the proposed design are more suitable than the conventional approaches, whereas, at the cost of more resources the Pipeline based system outperforms in speed.

References
[1] Cheng C and Parhi K K 2007 Low-cost parallel FIR filter structures with 2-stage parallelism IEEE Transactions on Circuits and Systems I: Regular Papers 54 pp 280-290
[2] Meher P K, Chandrasekaran S and Amira A 2008 FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic IEEE transactions on signal processing 56 pp 3009-3017
[3] Hawley R A, Wong B C, Lin T-j, Laskowski J and Samueli H 1996 Design techniques for silicon compiler implementations of high-speed FIR digital filters IEEE Journal of Solid-State Circuits 31 pp 656-667
[4] Shanthala S and Kulkarni S 2009 High speed and low power FPGA implementation of FIR filter for DSP applications European Journal of Scientific Research 31 pp 19-28
[5] Memon T D, Beckett P and Sadik A Z 2013 Power-area-performance characteristics of FPGA-based sigma-delta fir filters Journal of Signal Processing Systems 70 pp 275-288
[6] Memon T and Beckett P 2013 The impact of alternative encoding techniques on field
programmable gate array implementation of sigma-delta modulated ternary finite impulse response filters Australian Journal of Electrical and Electronics Engineering 10 pp 107-116

[7] Booth A D 1951 A signed binary multiplication technique The Quarterly Journal of Mechanics and Applied Mathematics 4 pp 236-240

[8] Wallace C S 1964 A suggestion for a fast multiplier IEEE Transactions on electronic Computers pp 14-17

[9] Dadda L 1965 Some schemes for parallel multipliers Alta frequenza 34 pp 349-356

[10] Maharaja J S S T 2009 Vedic mathematics: Sixteen simple mathematical formulae from the veda (Motilal Banarasidas Publishers) pp 5-45

[11] Rais M H 2009 Efficient hardware realization of truncated multipliers using FPGA International Journal of Engineering and Applied Sciences 5 pp 124-128

[12] Memon T D and Pathan A 2018 An approach to LUT based multiplier for short word length DSP systems 2018 International Conference on Signals and Systems (ICSigSys), pp 276-280

[13] Pathan A and Memon T D 2017 An optimised 3× 3 shift and add multiplier on FPGA 2017 14th International Bhurban Conference on Applied Sciences and Technology (IBCAST) pp 346-350

[14] Voronenko Y and Püschel M 2007 Multiplierless multiple constant multiplication ACM Transactions on Algorithms (TALG) 3 p 11

[15] de Dinechin F, Filip S-I, Forget L and Kumm M 2019 Table-Based versus Shift-And-Add constant multipliers for FPGAs 2019 IEEE 26th Symposium on Computer Arithmetic (ARITH) pp 151-158 IEEE

[16] Meyr H, Moeneclaey M and Fechtel S 1997 Digital communication receivers: synchronization, channel estimation, and signal processing (John Wiley & Sons, Inc)

[17] Milik A 2012 On Mapping of DSP48 Units for Arithmetic Operation in Reconfigurable Logic Controllers IFAC Proceedings Volumes 45 pp 249-254

[18] Plateau Jr R D, Wennekamp W E and Strader T H 2014 Memory controller with suspend feature ed: Google Patents

[19] Guo J-I, Liu C-M and Jen C-W 1992 The efficient memory-based VLSI array designs for DFT and DCT IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing 39 pp 723-733

[20] Todman T J, Constantines G A, Wilton S J, Mencer O, Luk W and Cheung P Y 2005 Reconfigurable computing: architectures and design methods IEE Proceedings-Computers and Digital Techniques 152 pp 193-207

[21] Zhuo L and Prasanna P K 2005 Sparse matrix-vector multiplication on FPGAs Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays pp 63-74