COMPACT AND HIGH-PERFORMANCE LUT-BASED MULTIPLICATION FOR EMBEDDED DSP SYSTEMS

[1]Ch.Ranadheer,[2]Santosh Kumar,[3]Lavanya
[123]Students
Department of ECE

ABSTRACT:
Multiplication is a fundamental operation in digital signal processing (DSP), but it remains a computational bottleneck in resource-constrained embedded systems. This paper presents a compact and high-performance look-up table (LUT)-based multiplier tailored for short word length DSP applications. By replacing traditional multiplier circuits with optimized LUT architectures, the design significantly reduces area and power consumption while maintaining acceptable computational accuracy and speed. The proposed design is particularly suited for embedded DSP systems where efficiency, speed, and silicon footprint are critical. Simulation and synthesis results on FPGA platforms demonstrate notable improvements in area-delay product (ADP) and energy efficiency compared to conventional array and booth multipliers, validating the viability of LUT-based approaches in modern low-power DSP applications.

1. INTRODUCTION

Digital Signal Processing (DSP) plays a pivotal role in embedded applications such as audio processing, communication systems, and biomedical signal analysis. Multiplication, being one of the most frequently used arithmetic operations, directly impacts the performance, area, and power consumption of DSP hardware implementations. Traditional hardware multipliers, such as array or Wallace tree multipliers, often demand substantial silicon area and power, which makes them unsuitable for short word length, resource-constrained environments.

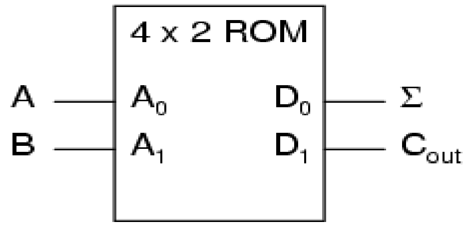
To address these limitations, Look-Up Table (LUT)-based multipliers offer a promising alternative by precomputing multiplication results and storing them in memory. This method reduces the need for complex arithmetic circuitry, enabling faster and more compact implementations. LUT-based designs are particularly beneficial in fixed-point DSP applications where operand ranges are known and limited.

This paper proposes an optimized LUT-based multiplier design that targets embedded DSP systems with strict constraints on size, power, and performance. By leveraging operand decomposition, memory-efficient storage, and pipelining techniques, the design achieves significant improvements in throughput and silicon utilization. The paper also discusses trade-offs in LUT size, word length, and accuracy, and provides comparative results to validate its practicality for embedded applications.

General LUT Design

It is possible to store binary data within solid-state devices. Those storage "cells" within solid-state memory devices are easily addressed by driving the "address" lines of the device with the proper binary values. A ROM memory circuit written, or programmed, with certain data, such that the address lines of the ROM served as inputs and the data lines of the ROM served as outputs, generating the characteristic response of a particular logic function. Theoretically, could ROM chip can program to emulate whatever logic function, wanted without having to alter any wire connections or gates.

Consider the following example of a 4 x 2 bit ROM memory programmed with the functionality of a half adder:



Address		Data	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

4X2 ROM

If this ROM has been written with the above data representing a half-adder's truth table, driving the A and B address inputs will cause the respective memory cells in the ROM chip to be enabled, thus outputting the corresponding data as the Σ (Sum) and C out bits. Unlike the half-adder circuit built of gates or relays, this device can be set up to perform any logic function at all with two inputs and two outputs, not just the half-adder function. To change the logic function, all we would need to do is write a different table of data to another ROM chip. EPROM chip can also program which could be re-written at will, giving the ultimate flexibility in function.

It is vitally important to recognize the significance of this principle as applied to digital circuitry. Whereas the half-adder built from gates or relays processes the input bits to arrive at a specific output, the ROM simply remembers what the outputs should be for any given combination of inputs. This is not much different from the "times tables" memorized in grade school: rather than having to calculate the product of 5 times 6 (5 + 5 + 5 + 5 + 5 + 5 = 30), school-children are taught to remember that 5 x 6 = 30, and then expected to recall this product from memory as needed. Likewise, rather than the logic function depending on the functional arrangement of hard-wired gates or relays (hardware), it depends solely on the data written into the memory (software).

Such a simple application, with definite outputs for every input, is called a look-up table, because the memory device simply "looks up" what the outputs should to be for any given combination of inputs states.

This application of a memory device to perform logical functions is significant for several reasons:

- Software is much easier to change than hardware.
- Software can be archived on various kinds of memory media (disk, tape), thus providing an easy way to document and manipulate the function in a "virtual" form; hardware can only be "archived" abstractly in the form of some kind of graphical drawing.
- Software can be copied from one memory device (such as the EPROM chip) to another, allowing the ability for one device to "learn" its function from another device.
- Software such as the logic function example can be designed to perform functions that would be extremely difficult to emulate with discrete logic gates.

The usefulness of a look-up table becomes more and more evident with increasing complexity of function. To build a 4-bit adder circuit using a ROM, A ROM is required with 8 address lines and 4 data lines.

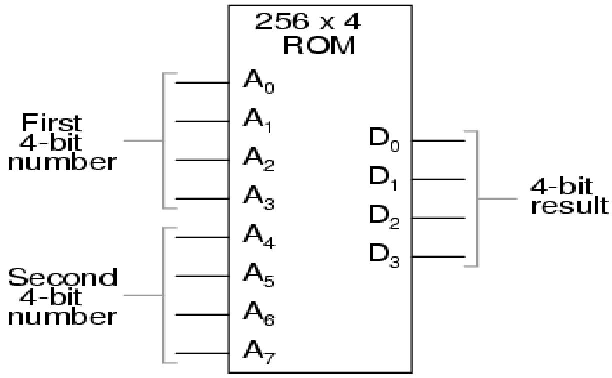


Figure 1. 256X4 ROM

Devices such as this, which can perform a variety of arithmetical tasks as dictated by a binary input code, are known as Arithmetic Logic Units, and they comprise one of the essential components of computer technology. Although modern ALUs are more often constructed from very complex combinational logic circuits for reasons of speed, it should be comforting to know that the exact same functionality may be duplicated with a "dumb" ROM chip programmed with the appropriate look-up table. In fact, this exact approach was used by IBM engineers in 1959 with the development of the IBM 1401 and 1620 computers, which used look-up tables to perform addition, rather than binary adder circuitry. The machine was fondly known as the "CADET," which stood for "**C**an't **A**dd, **D**oesn't **E**ven **T**ry."

2. LITERATURE SURVEY

The efficient memory-based VLSI array designs for DFT and DCT

Guo, J.-I.; Liu, C.-M.; Jen, C.-W Nat. Chiao Tung Univ., Hsinchu

Efficient memory-based VLSI arrays and a new design approach for the discrete Fourier transform and discrete cosine transform are presented. The DFT and DCT are formulated as cyclic convolution forms and mapped into linear arrays which characterize small numbers of I/O channels and low I/O bandwidth. Since the multipliers consume much hardware area, the designs utilize small ROMs and adders to implement the multiplications. Moreover, the ROM size can be reduced effectively by arranging the data in the designs appropriately. The arrays outperform others in the architectural topology , computing speeds, hardware complexity, the number of I/O channels, and I/O bandwidth. They benefit from the advantages of both systolic array and the memory-based architectures

On the design automation of the memory-based VLSI architectures for FIR filters

Lee, H.-R. Jen, C.-W. Liu, C.-M.

Dept. of Electron. Eng., Nat. Chiao Tung Univ., Hsinchu

An approach to automating the design of memory-based VLSI architectures for FIR filters has been developed. The automation is based on the exploration of the design space and schemes for efficient memory replacement, algorithm formulation, architecture design, and evaluation method. Various schemes and design considerations were integrated to produce a parameterized memory-based architecture that can easily be tuned to various hardware-speed requirements. This MBA is characterized by three design parameters. Differently configured MBAs result from specifying different values for these parameters. Hardware-speed evaluation formulas were established based on the required elements in MBAs. These elements include ROM, adders, and shift registers. These formulas and a cell library of a target technology can be used to design an optimally configured MBA by searching for the best values of the design parameters with the aid of a computer. Using the evaluation formulas and the parameterized architecture, an area-minimized architecture can be synthesized under a speed specification. Based on these results, an automatic synthesis too has been developed

A memory-efficient realization of cyclic convolution and its application to discrete cosine transform

The memory efficient design for realizing the cyclic convolution and its application to the discrete cosine transform. To adopt the method of distributed arithmetic computation, and exploit the symmetry property of DCT coefficients to merge the elements in the matrix of the DCT kernel and then separate the kernel to be two perfect cyclic forms to facilitate an efficient realization of 1-D N-point DCT using (N-1)/2 adders or subtractors, one small ROM module, a barrel shifter, and N-1/2+1 accumulators. The comparison results with the existing designs show that the proposed design can reduce delay-area product significantly.

Memory-based hardware for resource-constraint digital signal processing systems

The current trends of advancement of memory technology indicates reasonable scope to have efficient memory-based computing systems as promising alternative to the conventional logic-only computing in order to meet the stringent constraints and growing requirements of the digital signal processing systems in widely varying application environments. Several algorithms and architectures have been proposed in the literature to reduce the area- and time-complexities of commonly encountered computation-intensive cores of DSP functions by memory-based computing, but many more novel algorithms and architectures need to be developed to design flexible area-delay-power-efficient systems for various DSP applications.

3. EXISTING SYSTEM

A conventional LUT based multiplier is shown in Fig.1.1, where A is a fixed coefficient, and X is an input word to be multiplied with A. Assuming X to be a positive binary number of word length L, there can be 2L possible values of X, and accordingly, there can be 2L possible values of product C = A · X. Therefore, for memory-based multiplication, an LUT of 2L words, consisting of pre-computed product values corresponding to all possible values of X, is conventionally used. The product word A · Xi is stored at the location Xi for 0 ≤ Xi ≤ 2L − 1, such that if an L-bit binary value of Xi is used as the address for the LUT, then the corresponding product value A · Xi is available as its output. Several architectures have been reported in the literature for memory-based implementation of DSP algorithms involving orthogonal transforms and digital filters.

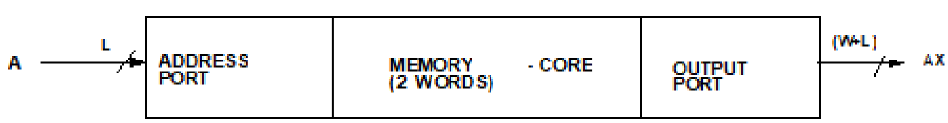


Fig 2. Conventional LUT Multiplier

- If the input bit size is 5 bits then the memory stores all the 2^5 possible

Outcomes which results in increase in LUT size. If the bits increase then the LUT Size increases exponentially

- Several architectures have been reported in the literature for memory-based implementation of DSP algorithms involving orthogonal transforms and digital filters .However, it is not found any significant work on LUT optimization for memory-based multiplication.

4. PROPOSED SYSTEM ARCHITECTURE

A new approach to LUT design is presented, where only the odd multiples of the fixed coefficient are required to be stored, which is referred to as the odd-multiple-storage *s*cheme in this brief. In addition, we have shown that, by the anti-symmetric product coding approach, the LUT size can also be reduced to half, where the product words are recoded as Anti-symmetric pairs.

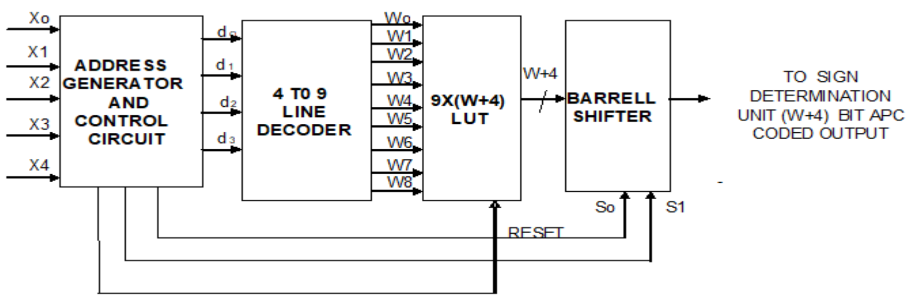


Fig 3. Proposed LUT Multiplier

if the input bit size= 5 then the memory stored is of 2^5/2 = 15 locations which results in a reduction in LUT size by factor of 2.

Objective

To build an LUT multiplier and the improvement is regarding the area consumption which increases exponentially in the existing LUT multipliers. Comparison of existing Booth multiplier with proposed LUT multiplier.

5. THE DESIGN FLOW

This section examines the flow for design using FPGA. This is the entire process for designing a device that guarantees that will not overlook any steps and that will have the best chance of getting backs a working prototype that functions correctly in the system.

The design flow consists of the steps in

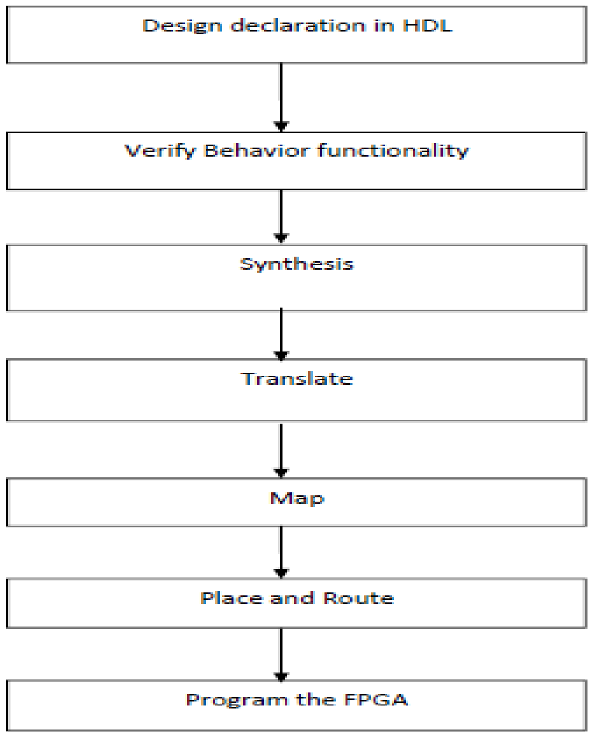


Figure.5. FPGA Design Flow

Design Entity

The basic architecture of the LUT is designed in this step which is coded in VHDL.

Behavioral Simulation

After the design phase, the LUT is verified using simulation software i.e. Xilinx ISE Simulator for different inputs to generate outputs and if it verifies then proceed further otherwise modification and necessary correction will be done in the HDL code. This is called as the behavioral simulation. Simulation is an ongoing process while the design is being done. Small sections of the design should be simulated separately before hooking them up to larger sections. There will be much iteration of design and simulation in order to get the correct functionality. Once design and simulation are finished, another design review must take place so that the design can be checked. It is important to get others to look over the simulations and make sure that nothing was missed and that no improper assumption was made. This is one of the most important reviews because it is only with correct and complete simulations that verify whether this chip will work correctly in the required system.

Proposed LUT APC Part

The structure and function of the LUT-based multiplier for $L = 5$ using the APC technique is shown in Fig.3.2 It consists of a four-input LUT of 16 words to store the APC values of product words as given in the sixth column of Table I, except on the last row, where $2A$ is stored for input $X = (00000)$ instead of storing a "0" for input $X = (10000)$. Besides, it consists of an address-mapping circuit and an add/subtract circuit. The address-mapping circuit generates the desired address (x3', x2', x1', x0'). A straightforward implementation of address mapping can be done by $X'L$ using $x4$ as the control bit. The address-mapping circuit, however, can be optimized to be realized by three XOR gates, three AND gates, two OR gates, and a NOT gate, as shown in Fig. 3.2 Note that the RESET can be generated by a control circuit (not shown in this figure) .The output of the LUT is added with or subtracted from $16A$, for $x4 = 1$ or 0, respectively, by the add/subtract cell. Hence, *x*4 is used as the control for the add/subtract cell.

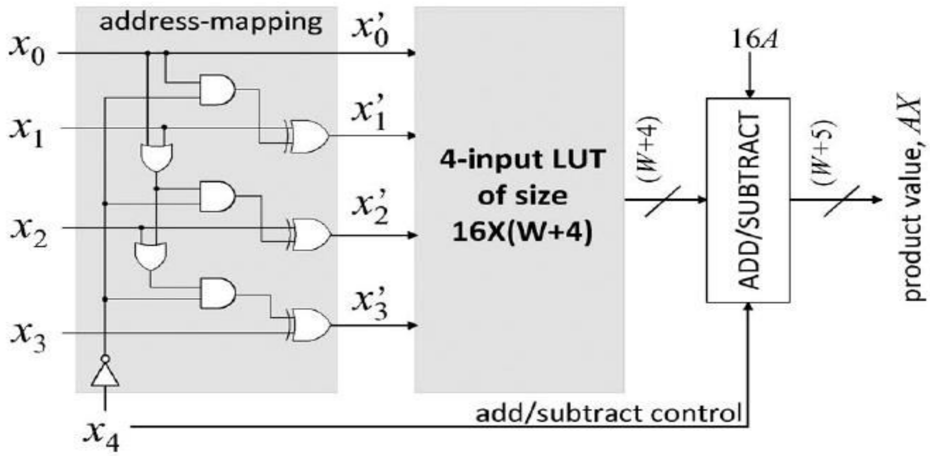


Figure 6. Proposed APC Part

For simplicity of presentation, it is assumed both X and A to be positive integers. The product words for different values of X for L = 5 are shown in Table I. It may be observed in this table that the input word X on the first column of each row is the two's complement of that on the third column of the same row. In addition, the sum of product values corresponding to these two input values on the same row is 32A. LUT based multiplier for L=5 using the APC technique

W = Width of A

L = Width of X

Table 1: Stored APC Words

APC Words for Different Input Values for $L = 5$

Input, X	product values	Input, X	product values	address $x'_3x'_2x'_1x'_0$	APC words
0 0 0 0 1	A	1 1 1 1 1	$31A$	1 1 1 1	$15A$
0 0 0 1 0	$2A$	1 1 1 1 0	$30A$	1 1 1 0	$14A$
0 0 0 1 1	$3A$	1 1 1 0 1	$29A$	1 1 0 1	$13A$
0 0 1 0 0	$4A$	1 1 1 0 0	$28A$	1 1 0 0	$12A$
0 0 1 0 1	$5A$	1 1 0 1 1	$27A$	1 0 1 1	$11A$
0 0 1 1 0	$6A$	1 1 0 1 0	$26A$	1 0 1 0	$10A$
0 0 1 1 1	$7A$	1 1 0 0 1	$25A$	1 0 0 1	$9A$
0 1 0 0 0	$8A$	1 1 0 0 0	$24A$	1 0 0 0	$8A$
0 1 0 0 1	$9A$	1 0 1 1 1	$23A$	0 1 1 1	$7A$
0 1 0 1 0	$10A$	1 0 1 1 0	$22A$	0 1 1 0	$6A$
0 1 0 1 1	$11A$	1 0 1 0 1	$21A$	0 1 0 1	$5A$
0 1 1 0 0	$12A$	1 0 1 0 0	$20A$	0 1 0 0	$4A$
0 1 1 0 1	$13A$	1 0 0 1 1	$19A$	0 0 1 1	$3A$
0 1 1 1 0	$14A$	1 0 0 1 0	$18A$	0 0 1 0	$2A$
0 1 1 1 1	$15A$	1 0 0 0 1	$17A$	0 0 0 1	A
1 0 0 0 0	$16A$	1 0 0 0 0	$16A$	0 0 0 0	0

For $X = (0\ 0\ 0\ 0\ 0)$, the encoded word to be stored is $16A$.

16 x (W+4) → 16 Locations and each location having (W+4) bits.

. Let the product values on the second and fourth columns of a row be u and v, respectively. Since one can write

u = [(u + v)/2 − (v − u)/2] and

v = [(u + v)/2 + (v − u)/2], for (u + v) = 32A,

U=16A+ [(V-U)/2]

V=16A-[(V-U)/2]

The product values on the second and fourth columns of Table I therefore have a negative mirror symmetry. This behavior of the product words can be used to reduce the LUT size, where, instead of storing u and v, only [(v − u)/2] is stored for a pair of input on a given row. The 4-bit LUT addresses and corresponding coded words are listed on the fifth and sixth columns of the table, respectively. Since the representation of the product is derived from the antisymmetric behavior of the products, we can name it as antisymmetric product code. The 4-bit address X_ = (x_3,x_2,x_1,x_0) of the APC word is given by

$$X' = \begin{cases} X_L, \text{if } x4=1 \\ X'_L, \text{if } x4=0 \end{cases}$$

Flow Chart For Proposed APC Part

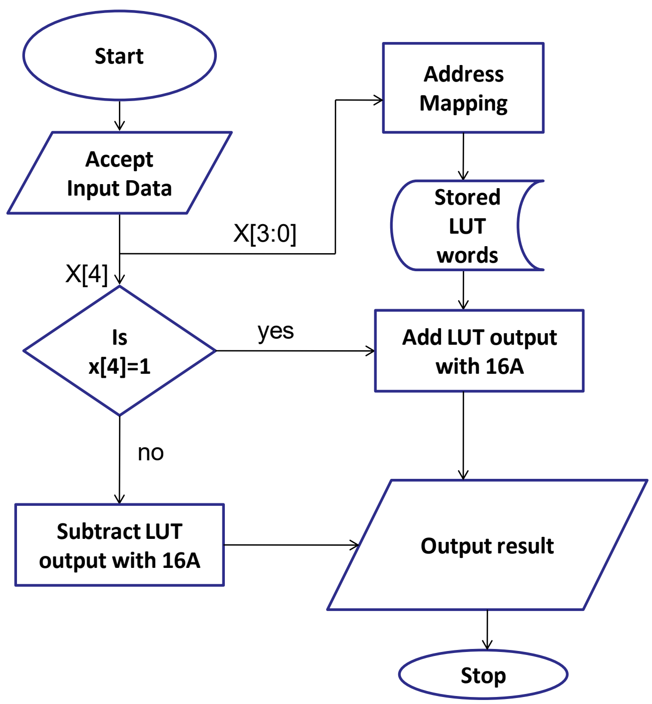


Figure.7. Flow Chart for Proposed LUT

Proposed APC-OMS Part

For the multiplication of any binary word X of size L, with a fixed coefficient A, instead of storing all the 2L possible values of C = A · X, only (2L/2) words corresponding to the odd multiples of A may be stored in the LUT, while all the even multiples of A could be derived by left-shift operations of one of those odd multiples. Based on the above assumptions, the LUT for the multiplication of an L-bit input with a W-bit coefficient could be designed by the following strategy.

1) A memory unit of [(2L/2) + 1] words of (W + L)-bit width is used to store the product values, where the first (2L/2) words are odd multiples of A, and the last word is zero.

2) A barrel shifter for producing a maximum of (L − 1) left shifts is used to derive all the even multiples of A.

3) The L-bit input word is mapped to the (L − 1)-bit address of the LUT by an address encoder, and control bits for the barrel shifter are derived by a control circuit.

Table 2: Stored APC-OMS Words

input X' $x'_3x'_2x'_1x'_0$	product value	# of shifts	shifted input, X''	stored APC word	address $d_3d_2d_1d_0$
0 0 0 1	A	0	0 0 0 1	$P0 = A$	0 0 0 0
0 0 1 0	$2 \times A$	1			
0 1 0 0	$4 \times A$	2			
1 0 0 0	$8 \times A$	3			
0 0 1 1	$3A$	0	0 0 1 1	$P1 = 3A$	0 0 0 1
0 1 1 0	$2 \times 3A$	1			
1 1 0 0	$4 \times 3A$	2			
0 1 0 1	$5A$	0	0 1 0 1	$P2 = 5A$	0 0 1 0
1 0 1 0	$2 \times 5A$	1			
0 1 1 1	$7A$	0	0 1 1 1	$P3 = 7A$	0 0 1 1
1 1 1 0	$2 \times 7A$	1			
1 0 0 1	$9A$	0	1 0 0 1	$P4 = 9A$	0 1 0 0
1 0 1 1	$11A$	0	1 0 1 1	$P5 = 11A$	0 1 0 1
1 1 0 1	$13A$	0	1 1 0 1	$P6 = 13A$	0 1 1 0
1 1 1 1	$15A$	0	1 1 1 1	$P7 = 15A$	0 1 1 1

Control Circuit

The control bits s0 and s1 to be used by the barrel shifter to produce the desired number of shifts of the LUT output are generated by the control circuit, according to the relations

$$s_0 = \overline{x_0 + \overline{(x_1 + \overline{x_2})}}$$
$$s_1 = \overline{(x_0 + x_1)}.$$

Note that ($s1s0$) is a 2-bit binary equivalent of the required number of shifts specified in Tables II. The RESET signal can alternatively be generated as ($d3$ AND $x4$).

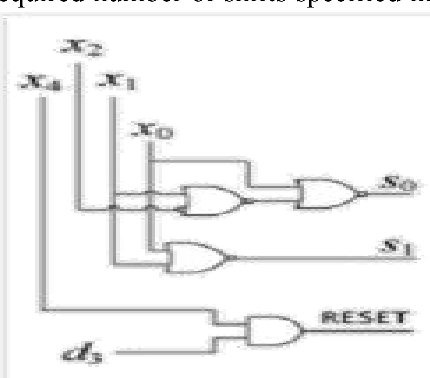


Figure 3.6 Control Circuit For Generation Of $S0$, $S1$, And Reset

The control circuit to generate the control word and RESET is shown in Fig. 3.7.The address-generator circuit receives the 5-bit input operand X and maps that onto the 4-bit address word (d3d2d1d0)

6. SIMULATION RESULTS & SYNTHESIS REPORTS

APC

Address Mapping

Figure 6.1 Simulation result of Address Mapping Unit

Address mapping unit converts the input to either two complement output or the input by seeing the LSB of the input

6.2. LUT APC

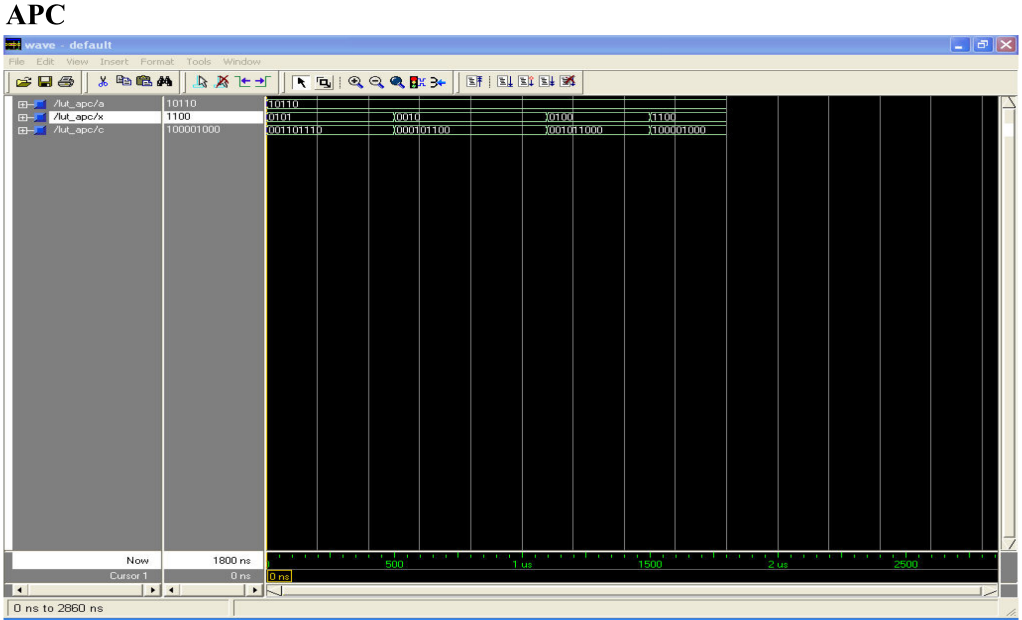


Figure 6.2 Simulation result of LUT APC Unit

LUT stores APC product word.

6.3. Add Sub Unit

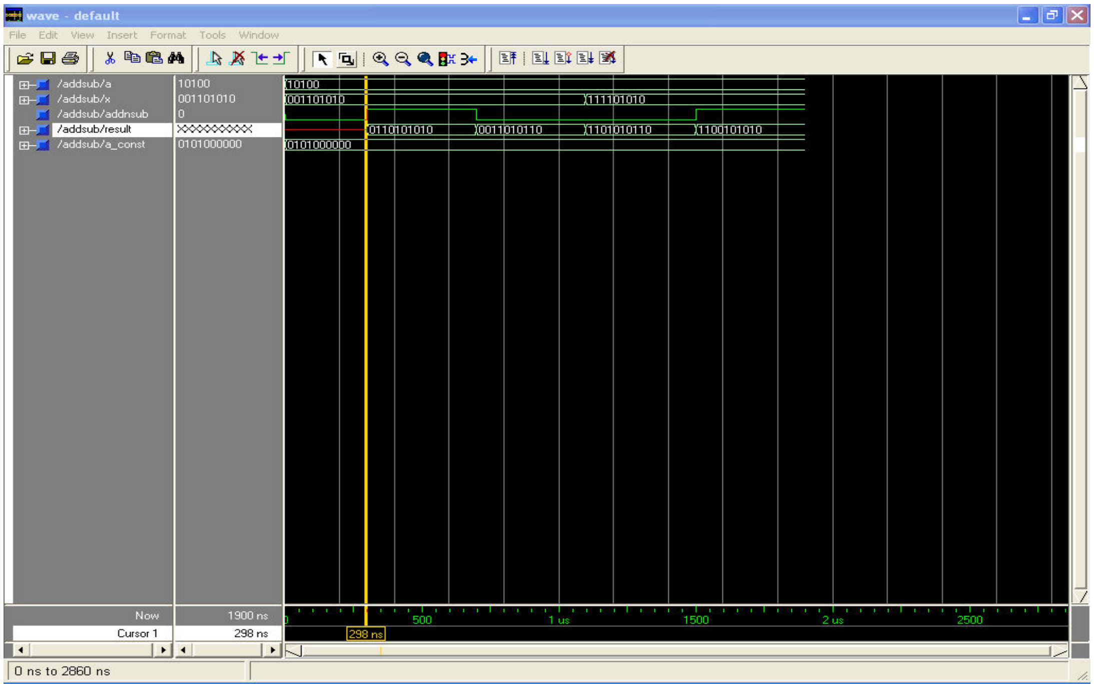


Figure 6.3 Simulation result of ADD SUB Unit

Add Sub unit either Add or Sub with 16A according to the value of the MSB of input

6.4.LUT APC Top Module

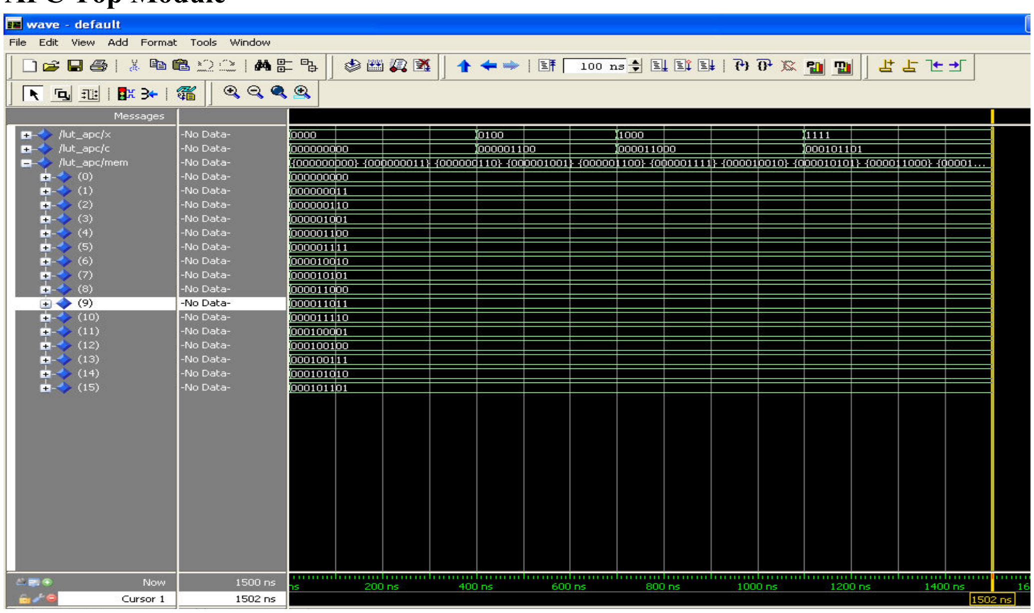


Figure 6.4 Simulation result of LUT APC Top Module

Output is equal to A.X and this module combines all the above three modules when A as constant

6.5. OMS

Address Generation

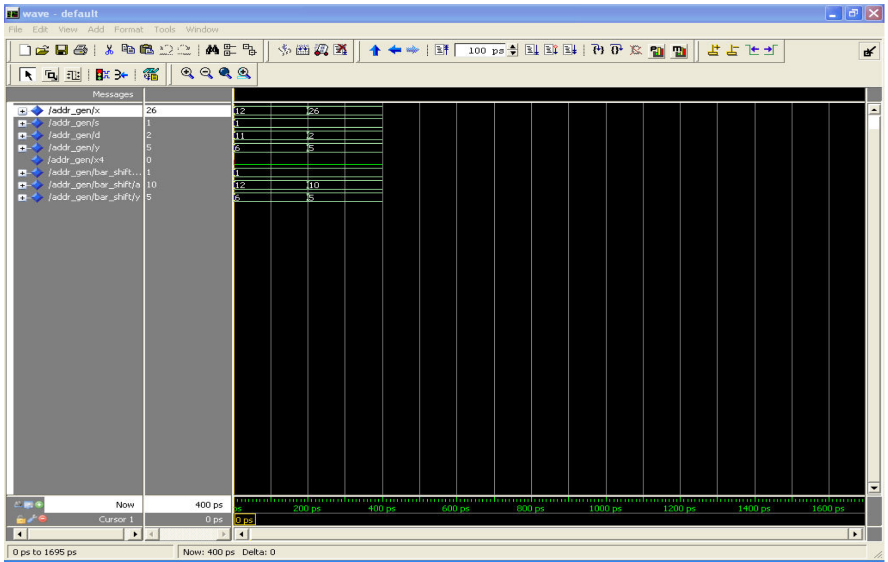


Figure 6.5 Simulation result of Address Generation Unit

6.6. Control Circuit

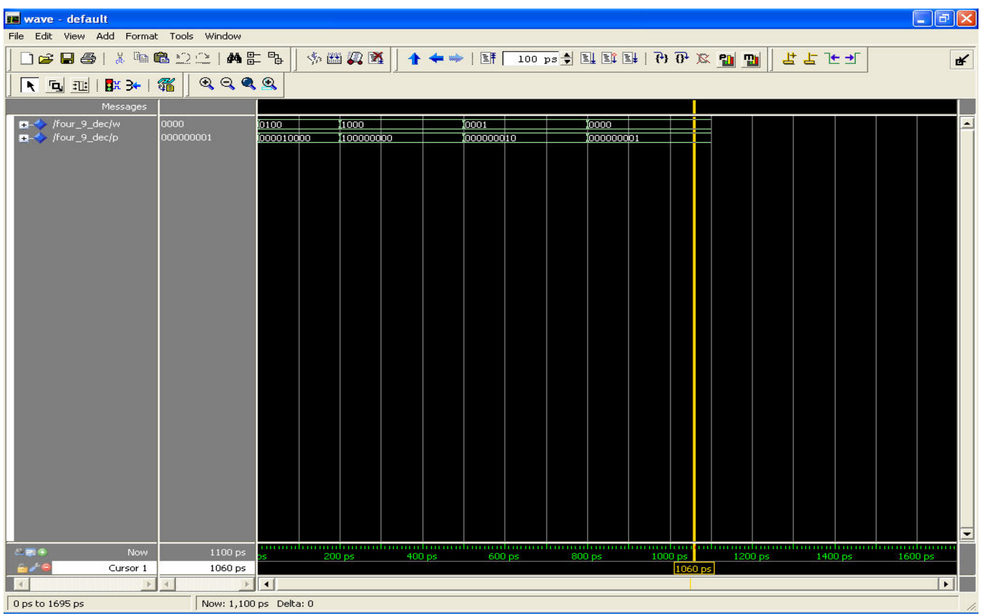


Figure 6.6 Simulation Result Of Control Circuit Unit

6.7. LUT APC & OMS

With A = 20

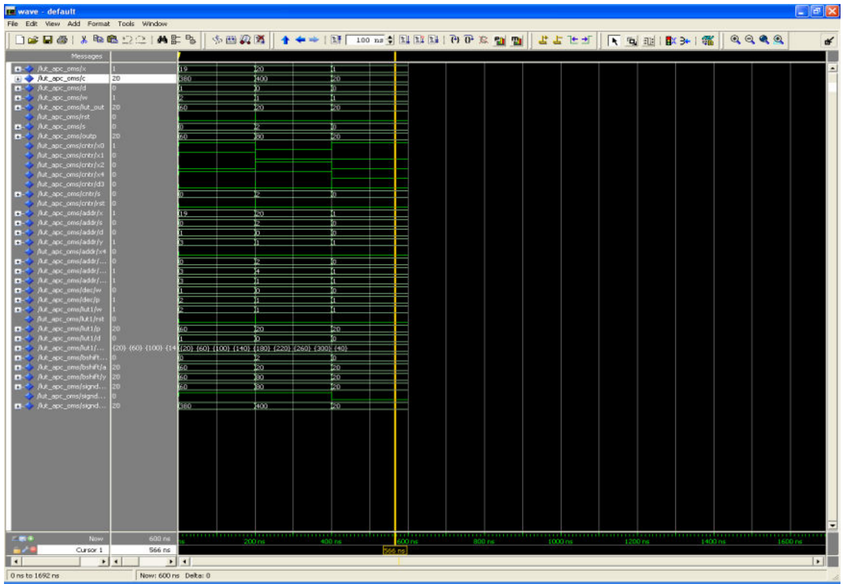


Fig 6.7 Simulation result of LUT APC & OMS Unit

6.8.FIR Filter With LUT

LUT With Constants {9, 17,13,1}

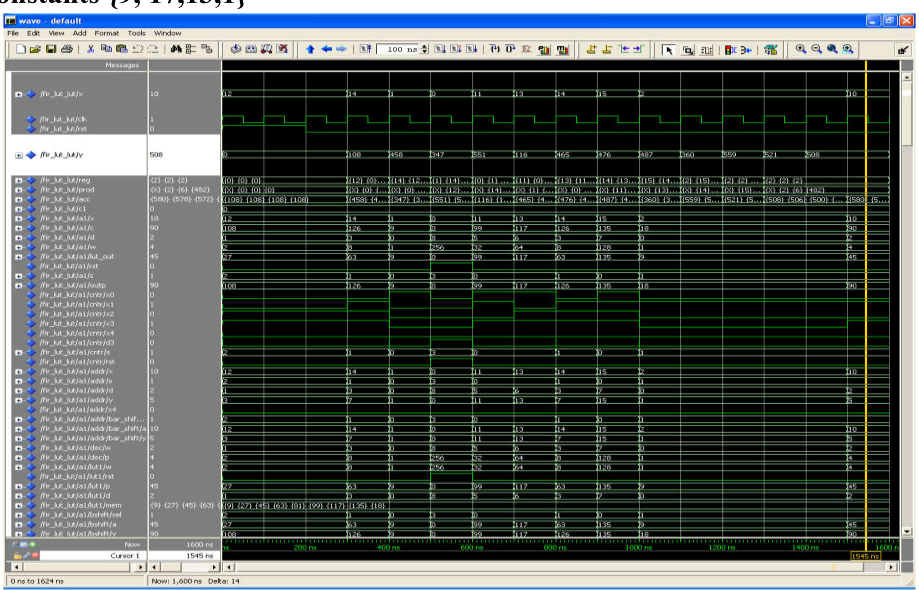


Fig .6.8. Simulation result of FIR Filter with LUT of Constants {9, 17,13, 1}

6.9. LUT 3 With Third Constant

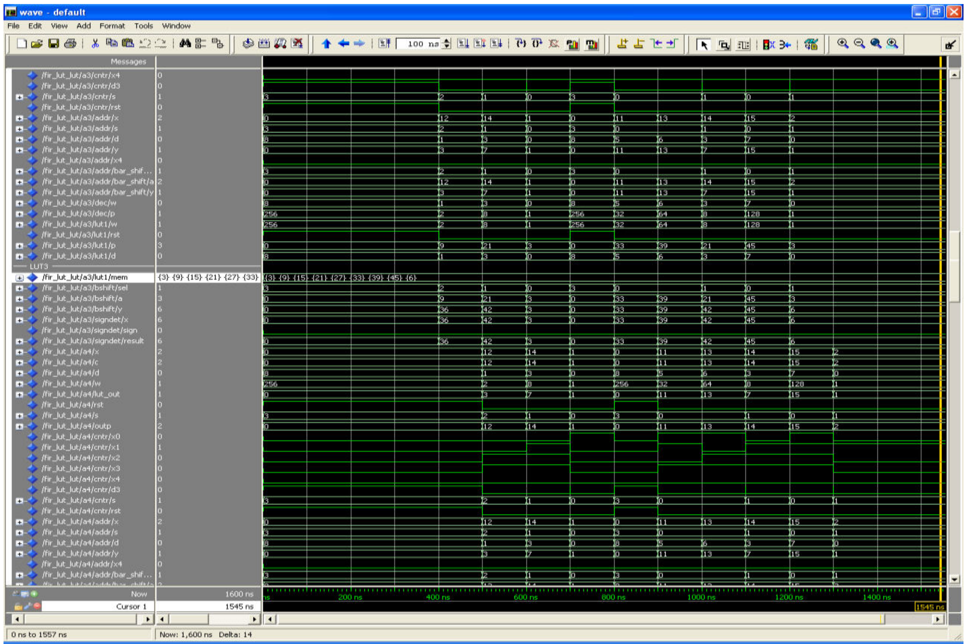


Fig 6.9. Simulation result of LUT 3 with Third Constant

6.10. LUT 4 With Fourth Constant

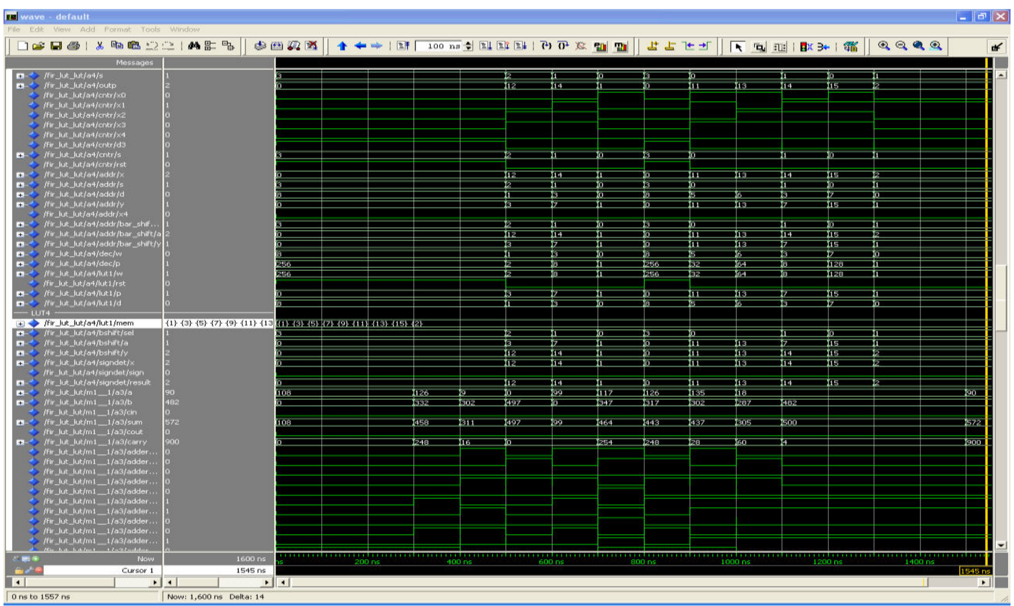


Fig 6.10. Simulation result of LUT 4 with Fourth Constant

7.CONCLUSION

This research presents a compact and high-performance LUT-based multiplier optimized for short word length embedded DSP systems. The proposed architecture significantly reduces hardware complexity and power consumption while maintaining competitive performance levels, making it suitable for real-time, low-power applications. Comparative analysis against conventional multiplication methods demonstrates the efficiency of the LUT approach in terms of area, delay, and energy metrics.

By tailoring the design for fixed-point DSP operations, the multiplier effectively balances accuracy and resource utilization. The results confirm that LUT-based multiplication is a viable solution for embedded system designers seeking efficient hardware acceleration without the overhead of traditional arithmetic units. Future work may explore adaptive LUT sizing and dynamic reconfiguration for broader application scalability and enhanced computational efficiency.

REFERENCES

[1] J.-I. Guo, C.-M. Liu, and C.-W. Jen, “The efficient memory-based VLSI array design for DFT and DCT,” IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process, vol. 39, no. 10, pp. 723–733, Oct. 1992.

[2] D. F. Chiper, M. N. S. Swamy, M. O. Ahmad, and T. Stouraitis, “A systolic array architecture for the discrete sine transform,” IEEE Trans. Signal Process., vol. 50, no. 9, pp. 2347–2354, Sep. 2002.

[3] H.-C. Chen, J.-I. Guo, T.-S. Chang, and C.-W. Jen, “A memory-efficient realization of cyclic convolution and its application to discrete cosine transform,” IEEE Trans. Circuits Syst. Video Technol., vol. 15, no. 3, pp. 445–453, Mar. 2005.

[4] P. K. Meher, “New approach to LUT implementation and accumulation for memory-based multiplication,” in Proc. IEEE ISCAS, May 2009, pp. 453–456.

[5] P. K. Meher, “New approach to LUT implementation and accumulation for memory-based multiplication,” in Proc. IEEE ISCAS, May 2009, pp. 453–456.

[6] P. K. Meher, “New look-up-table optimizations for memory-based multiplication,” in Proc. ISIC, Dec. 2009, pp. 663–666.