International Journal of Emerging Engineering Research and Technology
Volume. 2, Issue 6, September 2014, PP 72-80
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

LUT Design Using OMS Technique for Memory Based Realization of FIR Filter

L. Sujitha
PG scholar (VLSI)
Department of ECE, GRIET, JNTUH

M. Kiran
Associate Professor
Department of ECE, GRIET, JNTUH

Abstract: *The main objective of this paper is to develop an area efficient approach for the Look up table for the memory based realization of FIR filters. FIR filters are most often used in signal processing and image processing applications. Previously Distributed arithmetic (DA) is used which is popular for its potential for efficient memory based implementation of FIR filters. Here, we show that the look up table (LUT) multiplier based approach where the memory stores all possible values of filter coefficients. In digital signal processing applications multiplication is the major arithmetic technique. In ALU's the multiplier uses LUT as memory for their computation and it consumes much hardware area. To reduce the size of LUT odd-multiple-storage technique for LUT design for memory-based multipliers to be used in digital signal processing application. The design has been done in Verilog HDL, and is verified by Modelsim simulator and synthesized.*

Keywords: *Digital signal processing (DSP) chip, distributed arithmetic, Odd memory storage(OMS), FIR filter, memory-based computing, VLSI.*

1. Introduction

Filter is a frequency selective system which passes a desired range of frequencies and attenuates the remaining frequencies. Finite impulse response (FIR) digital filter is widely used as a basic tool in various signal processing because of their stability and linear phase. The order of an FIR filter primarily determines the width of the transition-band, such that the higher the filter order, the sharper is the transition between a pass-band and adjacent stop-band. Many applications in digital communication (channel equalization, frequency channelization), speech processing (adaptive noise cancellation), seismic signal processing (noise elimination), and several other areas of signal processing require large order FIR filters.

Finite impulse response (FIR) filters are digital filters having finite impulse response. FIR filters work only on past and current input values and are the simplest filters to design. FIR filters are also known as non- recursive filters.

Mathematically it can be given as

I(K)= 0 k>=k1

0 k>=k2

Where k_1 lies between -∞ to +∞, and k_2 lies between –∞ and ∞, I(n) denotes the impulse response of the digital filter, *n* is the discrete time index, and I_1 and I_2 are constants. A difference equation is the discrete time equivalent of a continuous time differential equation. The FIR filter has two basic properties

- Linearity
- Time invariance

The general difference equation for a FIR digital filter is

$$y(m) = \sum_{i=0}^{n-1} x(m-i)\, a_i$$

Where y(m) is the filter output at discrete time instance n, a_i is the i^{th} feed forward tap, or filter coefficient, and x(m-i) is the filter input delayed by k samples. The Σ denotes summation from i = 0 to i = n-1 where n is the number of feed forward taps in the FIR filter. FIR filters are the simplest filters to design. If a single impulse is present at the input of an FIR filter and all subsequent inputs are zero, the output of an FIR filter becomes zero after a finite time. Therefore, FIR filters are finite. The time required for the filter output to reach zero equals the number of filter coefficients. Equation describes the behaviour of the filter only in terms of current and past inputs. So FIR filter are also known as non -recursive filters.

In the processors like DSP core in FPGA applications MAC structures are replaced with look up tables. Rather than using conventional multipliers for complex multiplications it is best to use look up table approach. As the scaling in silicon devices has been increasing over the last four decades, semiconductor memory has become cheaper, faster and more power-efficient. According to the projections of the international technology roadmap for semiconductors (ITRS), embedded memories will continue to have dominating presence in the system-on-chip (SoC), which may exceed 90 %, of total SoC content. It has also been found that the transistor packing density of SRAM is not only high, but also increasing much faster than the transistor density of logic devices.

According to the requirement of different application environments, memory technology has been advanced in a large and significant manner. Radiation hardened memories for space applications, wide temperature memories for automotive, high reliability memories for biomedical instrumentation, low power memories for consumer products.

Memory based systems are the memory elements like RAM or ROM is used either as a part or whole of an arithmetic unit. Memory-based structures are more regular compared with the multiply-accumulate structures; and have many other advantages, e.g., greater potential for high-throughput and reduced-latency implementation and they have less dynamic power consumption due to less switching activities for memory-read operations compared to the conventional multipliers

Two ways of memory based techniques are distributed arithmetic for product computation and the another one is calculation of multiplication by look up table. In the LUT-multiplier-based approach, multiplications of input values with a fixed-coefficient are performed by an LUT consisting of all possible pre-computed product values corresponding to all possible values of input multiplicand, while in the previous approach, an LUT is used to store all possible values of inner-products of a fixed -point vector with any possible point bit-vector.

If the inner-products are caluculated in normal way, the memory-size of LUT multiplier based implementation increases exponentially with the word- length of input values, while that of the DA-based approach increases exponentially with the inner-product-length. Therefore several alternatives have done to reduce the memory-space in DA-based architectures using offset binary coding (OBC), and group distributed technique. But if in Distributed approach, if we reduce the memory size the delay increases, and also the number of adders and latches increases.

Tremendous work has done on DA-based computation of sinusoidal transforms and filters. In this paper, we designed two new approaches for LUT-multiplier-based implementation, in which memory-size is reduced to nearly half of the conventional approach. Besides, we find that instead of direct-form realization, transposed form realization of FIR filter is more efficient for the LUT-multiplier-based implementation. In the transposed form, a single segmented-memory core could be used instead of separate memory modules for individual multiplications in order to avoid the use of individual decoders for each of those separate modules.

2. LUT DESIGN FOR MEMORY-BASED MULTIPLICATION

The basic concept of memory based implementation is shown in fig.1. Let X be an input word to be multiplied with A and A be a fixed coefficient. If we assume X to be an unsigned binary number of word length M there can be 2^M possible combinations of X, and there are 2^M possible values of product C=A.X. Therefore, for the conventional implementation of memory based multiplication, a memory word of 2^M words is sufficient to be used as look-up-table consisting of previously calculated product values pertaining to all product values of X.

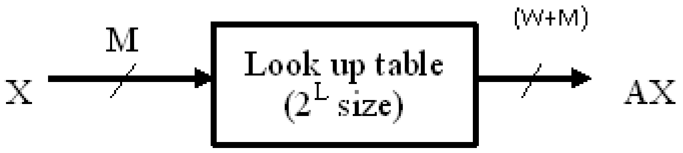


Fig1.*Conventional Memory Multiplier*

Therefore, for the conventional implementation of memory-based multiplication, a memory of 2^M words is used as look-up-table consisting of previously calculated product values corresponding to all possible values of the product-word, for $0 \leq X_i \leq 2^M - 1$, is stored at the memory location and the address is the same as the binary value of X_i, such that if M-bit binary value of X_i is used as address for the memory-unit, then the corresponding product value is read-out from the memory.

The 2^M possible values gives 2^M possible values of C=A.X, previously we have seen that only $(2^M/2)$ words corresponding to the odd multiples of A may only be stored in the LUT. We can explain this in Table I for M=4. At eight memory locations, eight odd multiples AX (2i+1) are stored as P_i for $0, 1, 2, \cdots, 7$. The even multiples of 2A,8A and 4A are derived by left shift operations of A. similarly 12A and 6A are derived by left shifting of 3A, while 10A is by shifting 5A and 14A is by 7A. $(2^M/2)$ odd multiple values are stored in the memory of the LUT, while the remaining (2^M/2-1) non-zero values could be derived by left shift operations of the stored values. The address X=(0000) corresponds to (A.X)=0, which is drawn by resetting LUT output. For an input multiplicand of word size M. Based on the above, an LUT for the multiplication of M bit input with W bit coefficient is derived by the following criteria:

Table1

Address d2 d1 d0	Word symbol	Stored value	Input a3a2a1a0	Product value	No of shifts	Control bits S1 S0
000	P0	A	0000	A	0	00
000	P0	A	0001	2^1xA	1	01
000	P0	A	0010	2^2xA	2	10
000	P0	A	0011	2^3xA	3	11
001	P1	3A	0100	3A	0	00
001	P1	3A	0101	2^1x3A	1	01
001	P1	3A	0110	2^2x3A	2	10
010	P2	5A	0111	5A	0	00
010	P2	5A	1000	2^1x5A	1	01
011	P3	7A	1001	7A	0	00
011	P3	7A	1010	2^1x7A	1	01
100	P4	9A	1011	9A	0	00
101	P5	11A	1100	11A	0	00
110	P6	13A	1101	13A	0	00
111	P7	15A	1111	15A	0	00

S0 and S1 are the control bits of Barrel Shifter

- A memory-unit of (2M/2) words of (W+M) bit width is used to store all the odd multiples of A.
- A barrel- shifter for producing a maximum of (M-1) left shifts is used to derive all the even multiplies of A.
- The M-bit input word is mapped to (M-1) bit LUT address by an encoder.

The control-bits for the barrel-shifter are derived by a control-circuit to perform the necessary shifts of the LUT output. Besides, a RESET signal is generated by the same control circuit to reset the LUT output when a=0.

3. PROPOSED LUT-BASED MULTIPLIER FOR 4-BIT INPUT

The proposed LUT-based multiplier for input word-size M=4 is shown in Fig. 3. It consists of a memory-array of (W+4) bit width with 8 words and a 3-to-8 line address decoder, AND-cell, a barrel-shifter, a 4-to-3 bit priority encoder to convert the 4-bit input operand to 3-bit LUT-address, and a control circuit for generating the control-word $(s_0 \quad s_1)$ for the barrel-shifter, and the RESET signal for the AND cell[1].

The 4 to 3 address encoder is shown in fig.2. It takes a 4 bit input word (a3 a2 a1 a0) and maps that on to the 3 bit address word (d2 d1 d0) according to the relations.

$d0 = \sim((\sim(a0\times a1)) \times (\sim(a1\times a2)) \times (a0+(\sim(a2\times a3))))$

$d1 = \sim((\sim(a0\times a2)) \times (a0+(\sim(a1\times a3))))$

$d2 = a0\times a3$

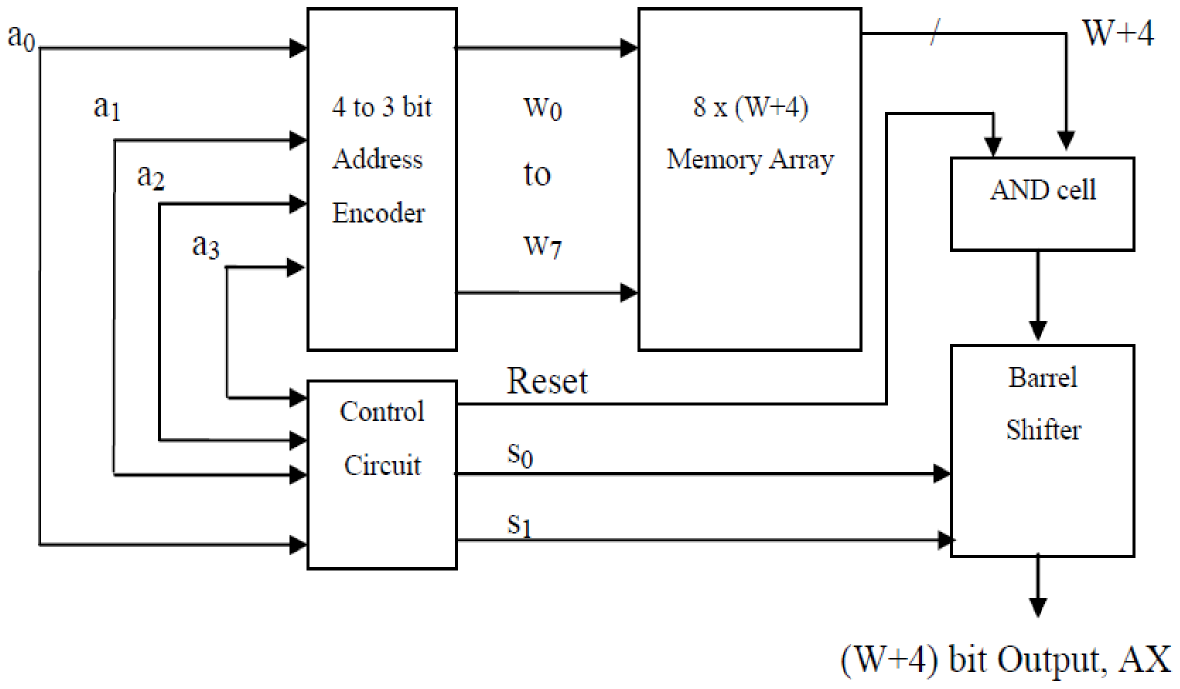


Fig2. *The Proposed LUT-based Multiplier*

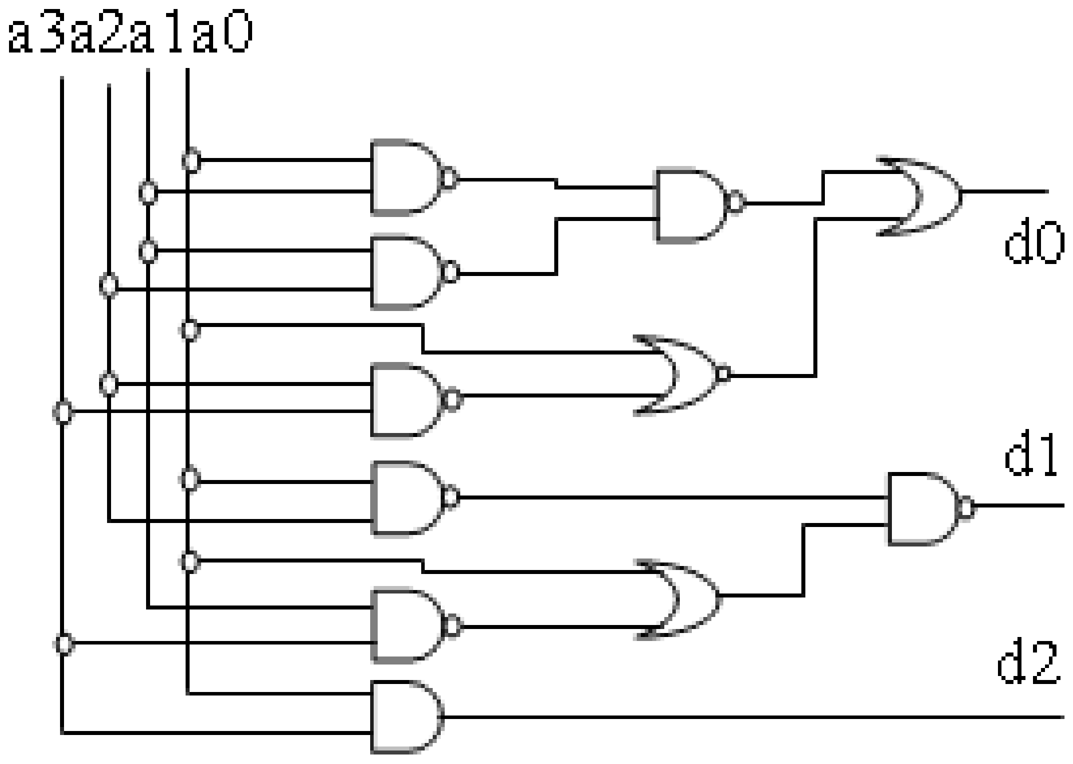
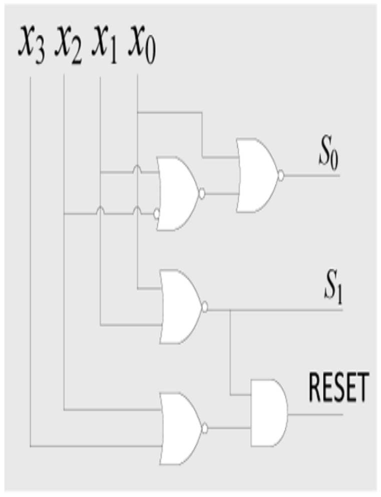
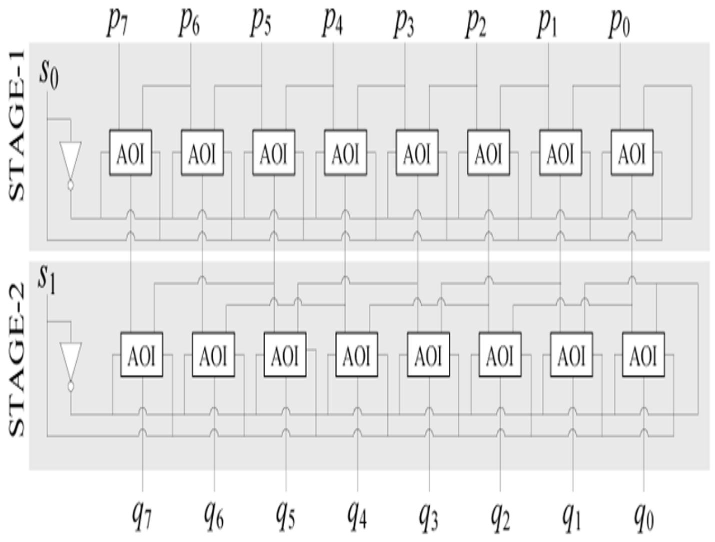


Fig3. *The 4-to-3 bits input Encoder*





[1]Fig4. *Control circuit* **[1]Fig5.** *Barallel shifter*

The pre-computed values of Ax(2i+1) are stored as P_i for i=0,1,2,3,4,5,6,7 at 8 locations of the memory-array as specified in Table I. The decoder takes the 3-bit address from the input encoder, and generates 8 word-select signals, $\{w_i, \text{for } 0 \leq i \leq 7\}$, to select the referenced-word from the memory-array. The output of the memory is either normal value or its sub-multiple in inverted form depending on the value of From Table I, we find that the LUT output is required to be shifted by 1 location to left when the input operand is one of the values{0010,0110,1010,1110}. Left-shifts of only two are required if is either (0 1 0 0) or (1 1 0 0). Only when the input word a=(1000), three shifts are required[1]. For the remaining input values, no shifts are required. Since the maximum number of left-shifts is three, a two-stage logarithmic barrel-shifter is sufficient to do the necessary left-shift operations.

The number of shifts needed to be done on the output is given in Table I. The control circuit [shown in Fig. 4] generates the control-bits are given by

$s_0 = !(a_0+(!(a_1+(!a_2))))$

$s_1 = !(a_0+a_1)$

A logarithmic barrel-shifter for W=M=4 is shown in Fig.5. It consists of two stages of 2-to-1 line bit-level multiplexers with inverted output, where each of the two stages involves(W+4)number of 2-input AND-OR-INVERT (AOI) gates. The control-bits and are fed to the AOI gates of stage-1 and stage-2 of the barrel-shifter, respectively. Since each stage of the AOI gates perform inverted multiplexing, after two stages of inverted multiplexing, outputs with desired number of shifts are produced by the barrel-shifter in (the usual) un-inverted form.

The input X= (0 0 0 0) implies to multiplication by a=0 which gives the product A*a=0. The reset function is implemented by a AND-cell consisting of (W+4) AND gates and an active-low RESET. So, when the input operand word a= (0 0 0 0), the output of the LUT is reset. The RESET bit is given as one of the inputs of all those AND gates, and the other input of (W+4) AND gates of AND cell are fed with (W+4) bits of LUT output in parallel. When a=(0 0 0 0), the control circuit in Fig.4, generates an active-low RESET according to the logic expression:

$\text{RESET}= !((!(a_0+a_1))\times (!(a_2+a_3)))$

When RESET=0, the outputs of all the AND gates become 0, so the barrel-shifter is fed with (W+4) number of zeros. When RESET=1, the outputs of all the AND gates are nothing but the output of the LUT which are fed to barrel shifter.

Now we have designed LUT multiplier for 8 bit. A dual port segmented memory is used. The 8 bit input is divided into two 4 bit and they are decoded using 4 to 3 decoder and control bits are generated. The shift adder performs the left shift operations of the output.

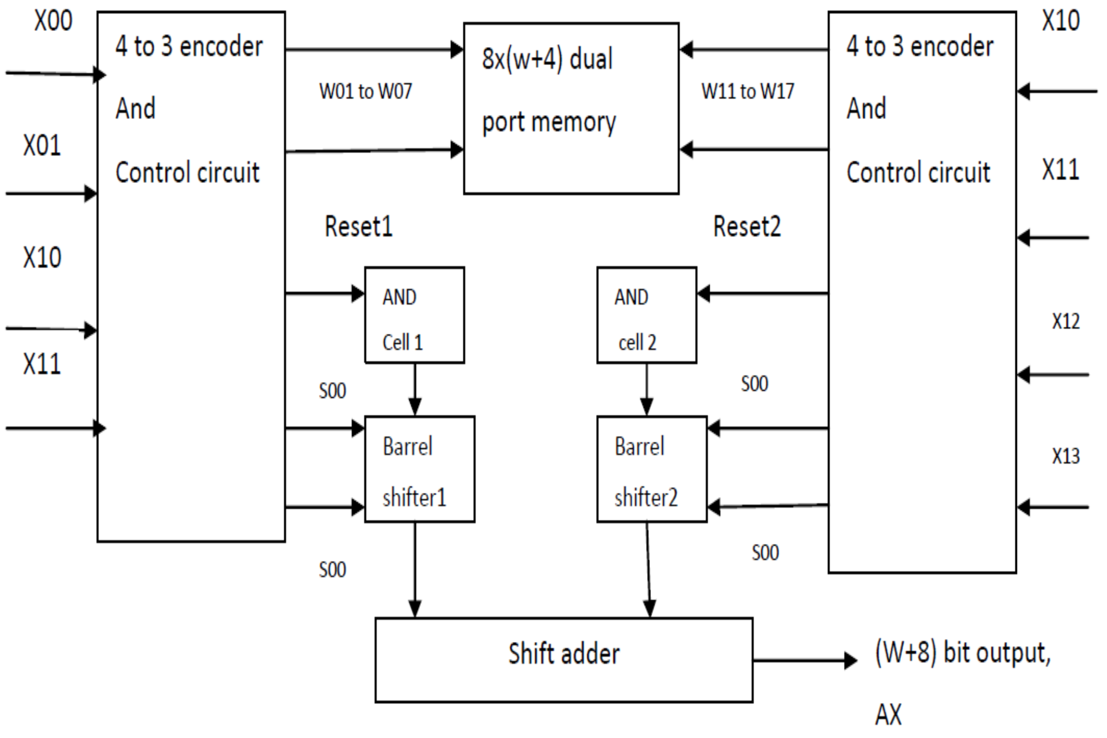
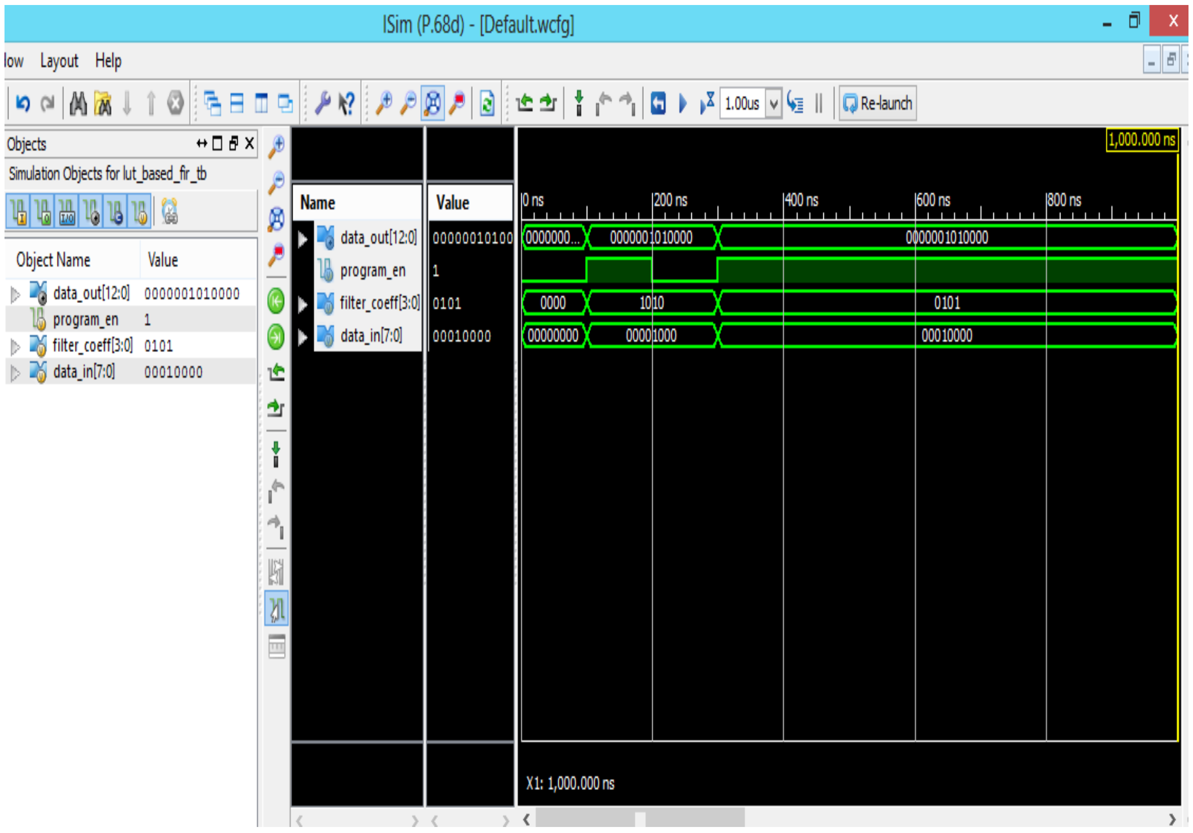


Fig6. *Memory based multiplier using proposed dual port memory for L=8*

4. Simulation Results

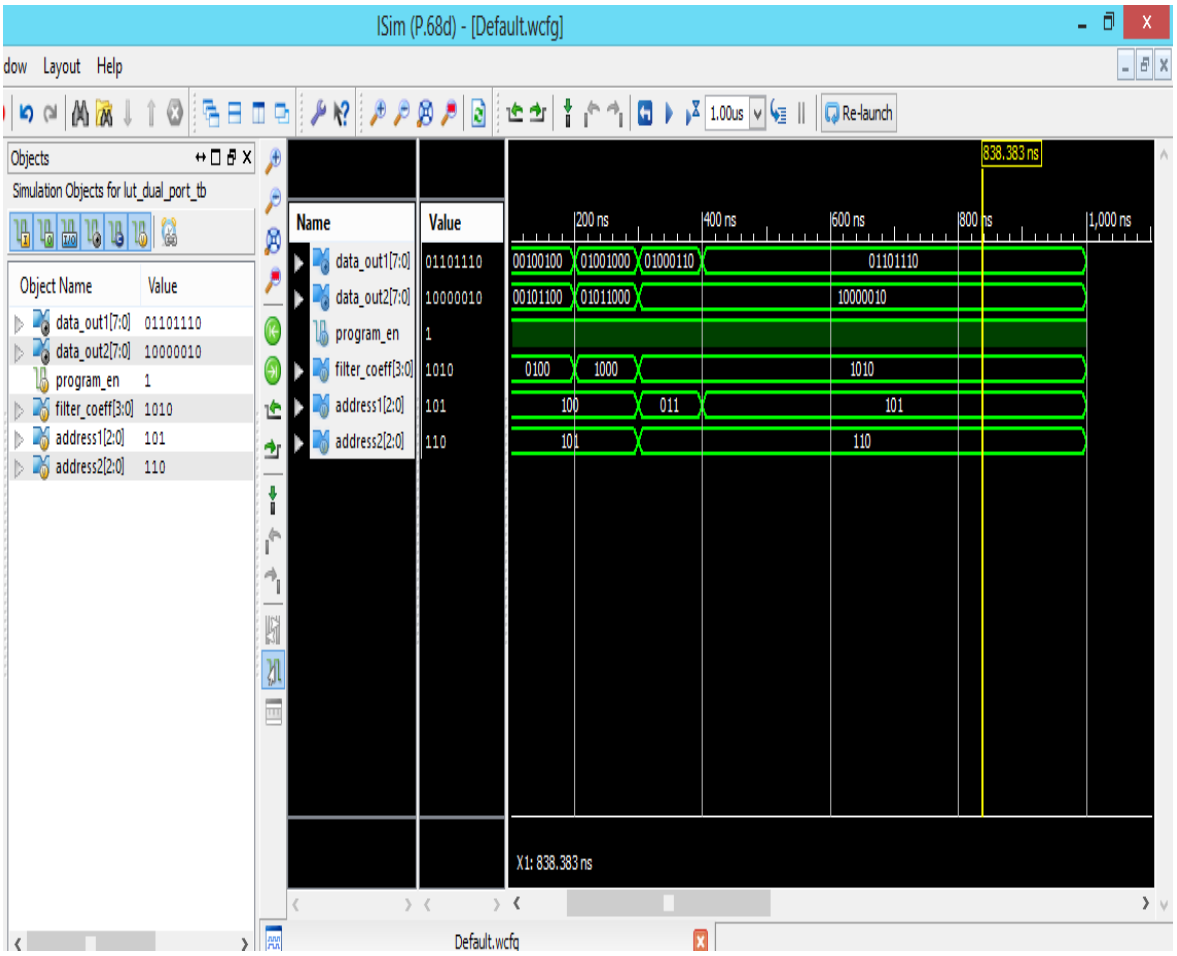
The memory based realization filter is Verilog coded and simulated on ModelSim to check the desired functionality. The filter specifications are 16 bit filter coefficients with input width of 8 bits. For comparison we have verilog coded the conventional filter structures. Figure 9 and Figure 10shows the Model sim snapshots of conventional and proposed FIR filter.



```
Object Name        Value
data_out[12:0]     0000001010000
program_en         1
filter_coeff[3:0]  0101
data_in[7:0]       00010000
```

```
Name               Value
data_out[12:0]     000000101000   0000000...  0000001010000  0000001010000
program_en         1
filter_coeff[3:0]  0101           0000  1010  0101
data_in[7:0]       00010000       00000000  00001000  00010000
```

Fig7. *Simulation result of LUT based Filter*



Object Name	Value
data_out1[7:0]	01101110
data_out2[7:0]	10000010
program_en	1
filter_coeff[3:0]	1010
address1[2:0]	101
address2[2:0]	110

ISim (P.68d) - [Default.wcfg]
Simulation Objects for lut_dual_port_tb
838.383 ns
X1: 838.383 ns
Default.wcfg
Re-launch

Fig8. *Simulation results of dual port memory*

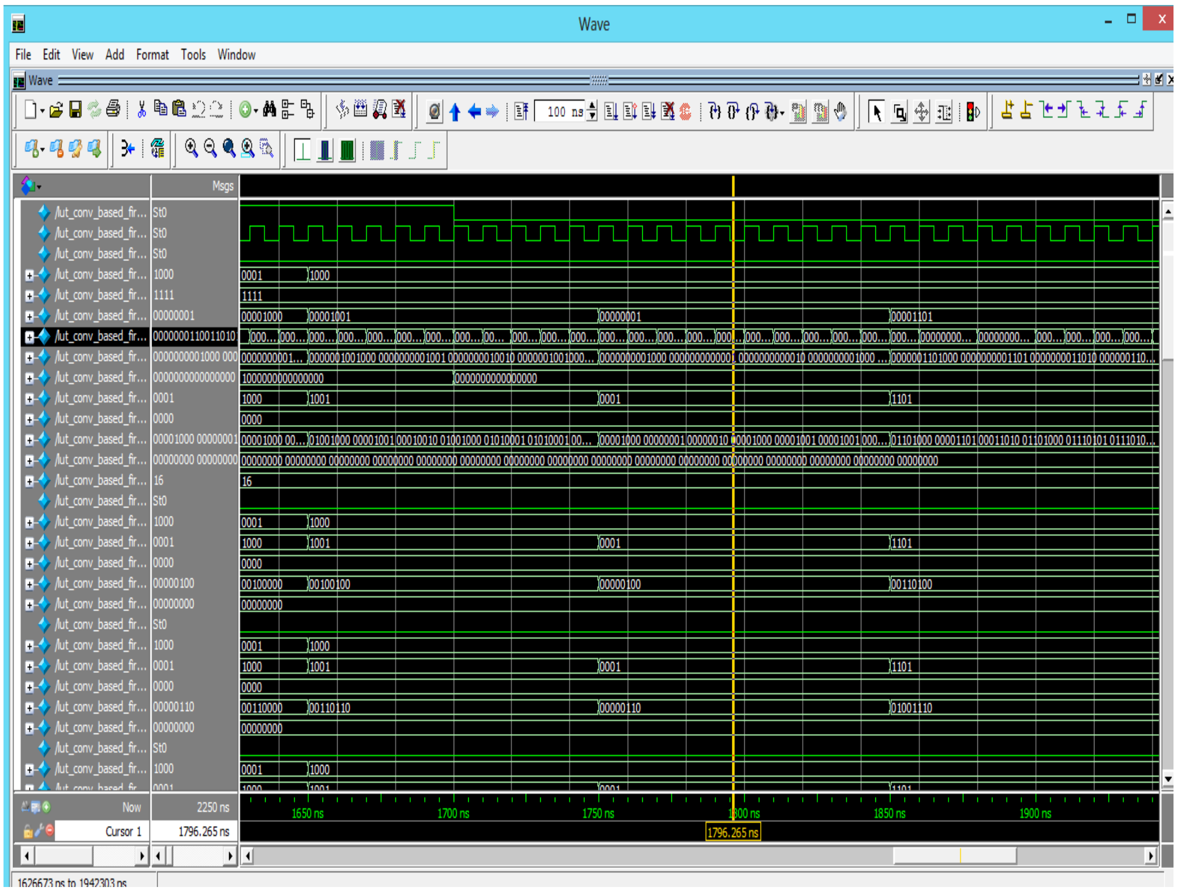


Fig9. *Simulation result of conventional FIR filter*

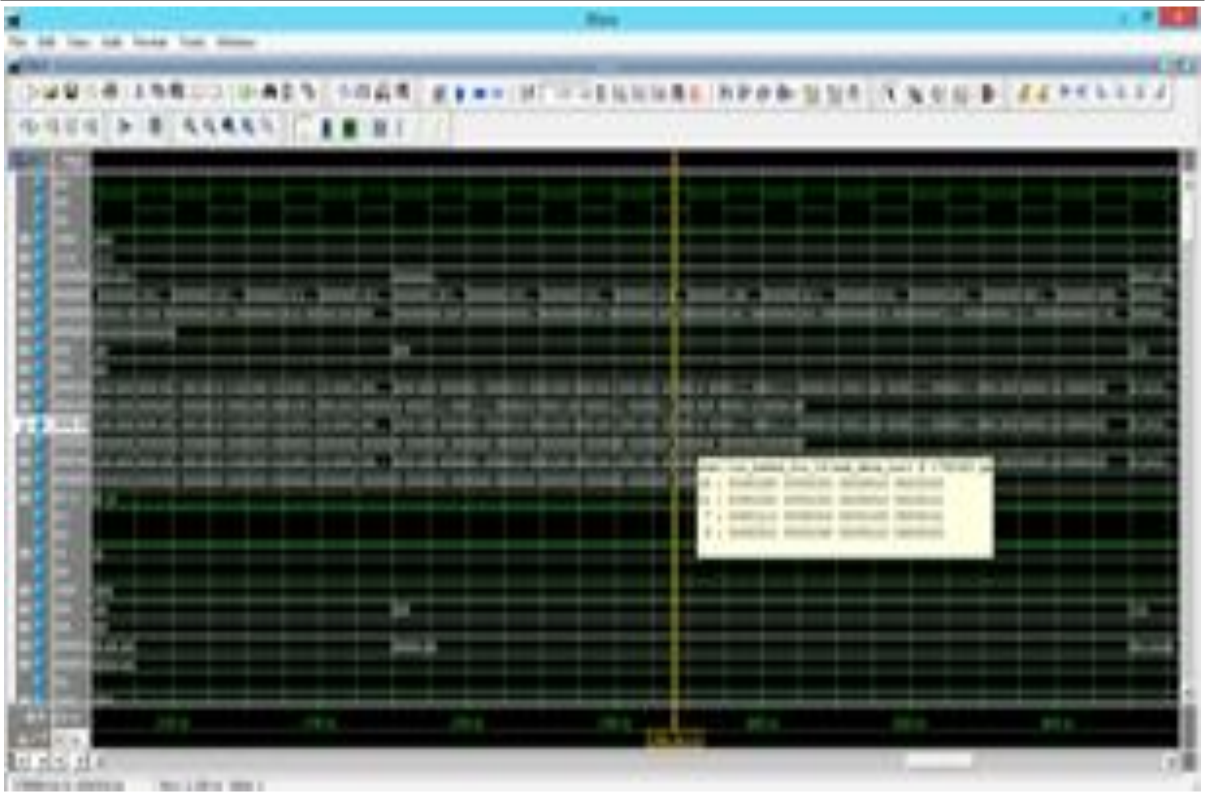


Fig10. *Simulation result of proposed FIR filter*

5. Conclusion

In this paper LUT design using odd-multiple-storage scheme, for address-length 8, the LUT size is reduced to half by using a barrel-shifter and the number of multipliers, registers, latches were reduced when compared with conventional based approach. The synthesis report is observes using Xilinx. Therefore the LUT-multiplier-based approach of FIR filter is more efficient than the conventional approach in terms of area-complexity for a given throughput and lower latency of implementation.

Future work

The LUT-multipliers could be used for memory-based implementation for linear and circular convolutions, sinusoidal transforms, and inner-product computation. The performance of memory-based structures, with different adder and memory implementations could be studied in future for different DSP applications. Further work is required to be carried out to find other possibilities of LUT-optimization with different address sizes for efficient memory-based multiplication.

References

[1] Pramod kumar mehar “New approach to Look up table design and memory based realization of FIR filter” in IEEE transactions of Circuits and systems june 2009 pp 1-12.

[2] M. Mehendale, S. D. Sherlekar, and G. Venkatesh, “Area-delay tradeoff in distributed arithmetic based implementation of FIR filters,” in Proc. 10th Int. Conf. VLSI Design, Jan. 1997, pp. 124–129.

[3] P. K. Meher, S. Chandrasekaran, and A. Amira, “FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic,” IEEE Trans. Signal Process., vol. 56, no. 7, pp. 3009–3017, Jul. 2008.

[4] P. K. Meher, “Unified systolic-like architecture for DCT and DST using distributed arithmetic,” IEEE Trans. Circuits Syst. I, Reg. Papers, vol. 53, no. 5, pp. 2656–2663, Dec. 2006.

[5] P. K. Meher and M. N. S. Swamy, “New systolic algorithm and array architecture for prime-length discrete sine transform,” IEEE Trans. Cir- cuits Syst. II, Exp. Briefs, vol. 54, no. 3, pp. 262–266, Mar. 2007.

[6] P. K. Meher, J. C. Patra, and M. N. S. Swamy, “High-throughput memory-based architecture for DHT using a new convolutional for- mulation,” IEEE Trans. Circuits Syst. II, Exp. Briefs, vol. 54, no. 7, pp. 606–610, Jul. 2007.

[7] J. P. Choi, S.-C. Shin, and J.-G. Chung, “Efficient ROM size reduc- tion for distributed arithmetic,” in Proc. IEEE Int. Symp. Circuits Syst. ISCAS, May 2000, vol. 2, pp. 61–64.

[8] P. K. Meher, “New approach to LUT implementation and accumula- tion for memory-based multiplication,” in Proc. 2009 IEEE Int. Symp. Circuits Syst., ISCAS’09, May 2009, pp. 453–456.

[9] P. K. Meher, “New look-up-table optimizations for memory-based multiplication,” in Proc. Int. Symp. Integr. Circuits (ISIC’09), Dec. 2009, pp. 663-666.

[10] K. Itoh, S. Kimura, and T. Sakata, “VLSI memory technology: Current status and future trends,” in Proc. 25th Eur. Solid-State Circuits Conference, ESSCIRC’99, Sep.1999, pp. 3–10.

[11] T. Furuyama, “Trends and challenges of large scale embedded memories,” in Proc. IEEE Conf. Custom Integrated Circuits, Oct. 2004, pp. 449–456.