Indian Journal of Science and Technology

FIR Filter Implementation using Modified Distributed Arithmetic Architecture

M. Yazhini[1*] and R. Ramesh[2]

[1]PG Scholar, Department of Electronics and Communication Engineering, Saveetha Engineering College, Tamil Nadu, 602105, India; yazhinimohan21@gmail.com

[2]Professor, Department of Electronics and Communication Engineering, Saveetha Engineering College, Tamil Nadu, 602105, India; raammesh1976@gmail.com

Abstract

In this project use Distributed Arithmetic (DA) technique for FIR filter. In this technique consist of Look Up Table (LUT), shift register and accumulator. Based on this technique multipliers in FIR filter are removed. Multiplication is performed through shift and addition operations. The LUT can be subdivided into a number of LUT to reduce the size of the LUT for higher order filter. Each LUT operates on a different set of filter taps. Analysis on the performance of various filter orders with different address length are done using Xilinx synthesis tool. The proposed architecture provides less latency and less area compared with existing structure of FIR filter.

Keywords: FIR, Distributed Arithmetic, LUT.

1. Introduction

A digital filter is a system that performs mathematical operations on a sampled or discrete time signal to reduce or enhance certain aspects of that signal. One type of digital filter is FIR filter. It is a stable filter. It gives linear phase response. Pipelining and parallel processing technique is used in FIR filter. Pipelining operation takes place in an interleaved manner. Pipelining done by inserting latches (delay element) in the system. It increases the overall speed of the architecture but the hardware structure and system latency will increases. Hardware structures increases due to inserting pipelining latches. For M level pipelining M-1 delay elements required. Latency is the difference between the availability of first output in the sequential system and the pipeline system[1]. At every clock cycle it will operate multiple inputs and produced multiple outputs is called parallel processing. It required extra hardware. Both pipelining and parallel processing has disadvantages. For FIR filters, output is a linear convolution of weights and inputs. For an Nth-order FIR filter, the generation of each output sample takes N+1 multiply accumulate (MAC) operations.

Multiplication is strongest operation because it is repeated addition. It require large portion of chip area. Power consumption is more. 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 are expected to have less dynamic power consumption due to less switching activities for memory-read operations compared to the conventional multipliers. Memory based structures are well-suited for many digital signal processing (DSP) algorithms, which involve multiplication with a fixed set of coefficients. For this Distributed Arithmetic architecture used in FIR filter.

Distributed arithmetic is one way to implement convolution with multiplier less unit, where the MAC operations are replaced by a series of LUT access and

* *Corresponding author:*
M.Yazhini (yazhinimohan21@gmail.com)

summations. Distributed Arithmetic is a different approach for implementing digital filters. The basic idea is to replace all multiplications and additions by a table and a shifter-accumulator. LUT are the kind of logic that used in SRAM based FPGAs. Basically each look table is a bunch of single bit memory cells storing individual bit values in each of the cells. Memory access time is less in SRAM, so speed of the static RAM is high. Distributed Arithmetic provides cost-effective and area-time efficient computing structures. Digital Finite Impulse Response (FIR) filters are essential building blocks in most Digital Signal Processing (DSP) systems. A large application area is telecommunication, where filters are needed in receivers and transmitters, and an increasing portion of the signal processing is done digitally [2, 3]. However, power dissipation of the digital parts can be a limiting factor, especially in portable, battery-operated devices. Scaling of the feature sizes and supply voltages naturally helps to reduce power. For a certain technology, there are still many kinds of architectural and implementation approaches available to the designer. Due to the advancement in Very Large Scale Integration (VLSI) technology, realization of FIR filters is done in Application Specific Integrated Circuits (ASIC) and Field-Programmable Gate Arrays (FPGA) platforms.

2. FIR Filter with Multiplier

In signal processing, a finite impulse response (FIR) filter is a filter whose impulse response is of finite duration, because it settles to zero in finite time. This is in contrast to infinite impulse response (IIR) filters, which may have internal feedback and may continue to respond indefinitely.

The Figure 1 shows discrete time FIR filter of order N. The top part is an N-stage delay line with N + 1 taps. Each unit delay is a z^{-1} operator in Z-transform notation. The output y of a linear time invariant system is determined by convolving its input signal x with its impulse response h. For a discrete-time FIR filter, the output is a weighted sum of the current and a finite number of previous values of the input. The operation is described by the following equation, which defines the output sequence y[n] in terms of its input sequence x[n]:

$$y(n) = b_0x(n)+b_1x(n-1)+....+b_Nx(n-N)$$
$$= \sum_{i=0}^{N} b_i x(n-i) \quad (1)$$

where,

x[n] is the input signal,

y[n] is the output signal,

b_i are the filter coefficients, also known as tap weights that make up the impulse response.

The transfer function of the FIR filter is

$$H(Z) = Z\{h(n)\}$$
$$= \sum_{n=0}^{N} h(n)\, z^{-n}$$
$$= \sum_{n=0}^{N} b_n\, z^{-n} \quad (2)$$

FIR filters are clearly Bounded Input Bounded Output (BIBO) stable system, since the output is a sum of a finite number of finite multiples of the input values. h (n) is the impulse response of the filter.

Third order FIR Filter (Figure 2):

N order filter has,

- Coefficients N+1
- Multiplier N+1
- Adder N.

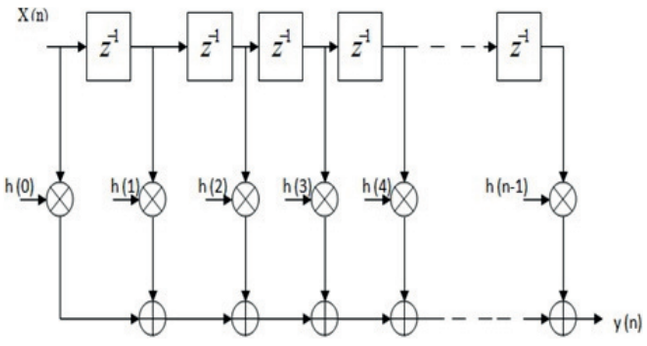

Figure 1. FIR filter with multiplier.

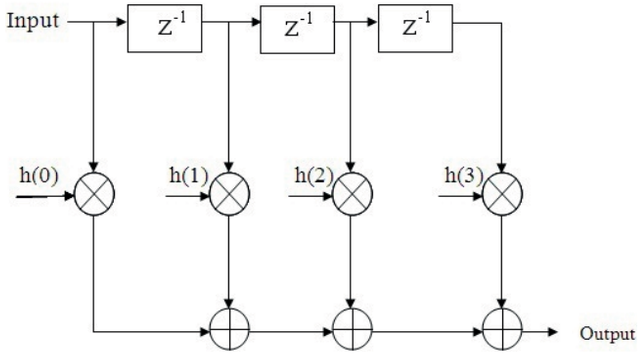

Figure 2. 3rd order FIR filter.

Output of 3rd order FIR filter is

$$y(n) = \sum_{k=0}^{3} h(k)x(n-k)$$

$$y(n) = h(0)\,x(n) + h(1)\,x(n-1) + h(2)\,x(n-2) + h(3)\,x(n-3) \quad (3)$$

For 3rd order FIR filter required 4 coefficients, 4 multipliers and 3 adders. For this output requires 4 multiplication and 3 addition operation. Multiplication is the strongest operation. It require large portion of chip area. Power consumption is more.

3. FIR Filter with Distributed Arithmetic Architecture

3.1 Distributed Arithmetic

Distributed Arithmetic (DA) technique is bit-serial in nature. It is actually a bit-level rearrangement of the multiply and accumulation operation. The basic DA is a computational algorithm that affords efficient implementation of the weighted sum of products, or dot product. DA is a bit-serial operation used to compute the inner (dot) product of a constant coefficient vector and a variable input vector in a single direct step and is given by

$$y = \sum_{k=1}^{K} A_k x_k \quad (4)$$

where,

y - Output response

A_k - Constant filter coefficients

X_k - Input data

Let X_k be a N-bits and can be expressed in scaled two's complement number as

$$x_k = -b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \quad (5)$$

Substituting X_k into equation y,

$$y = \sum_{k=1}^{K} A_k \left[-b_{k0} + \sum_{n=1}^{N-1} b_{kn} 2^{-n} \right]$$

$$y = -\sum_{k=1}^{K} b_{k0}.A_k + \sum_{k=1}^{K} \left[\sum_{n=1}^{N-1} (b_{kn}.A_k) 2^{-n} \right]$$

Rearranging the summation based on power terms and then grouping the sum of the products,

$$y = -\sum_{k=1}^{K} b_{k0}.A_k + \sum_{n=1}^{N-1} [b_{1n}.A_1 + b_{2n}.A_2 + \ldots + b_{Kn}.A_K]\, 2^{-n}$$

The final formulation,

$$y = -\sum_{k=1}^{K} A_k.b_{k0} + \sum_{n=1}^{N-1} \left[\sum_{k=1}^{K} A_k.b_{kn} \right] 2^{-n} \quad (6)$$

3.2 FIR Filter Realization using DA

The DA of FIR filter consists of Look Up Table (LUT), Shift registers and scaling accumulator. In DA all the cumulative partial product outcomes are precomputed and stored in a Look Up Table (LUT) which is addressed by the multiplier bits. A filter with N coefficients the LUT has 2^N values.

In the above equation, each term inside the brackets indicates a binary AND operation involving a bit of the input variable and all the bits of the constant. The plus signs denote arithmetic addition operations. The exponential factors denote the scaled parts of the bracketed pairs to the total sum. Using this, a look-up table can be constructed that can be addressed by the same type of scaled bit of all the input variables and can access the sum of the terms within each pair of brackets.

From equation 6, $\left[\sum_{k=1}^{K} A_k.b_{kn} \right]$ has only 2K possible values and it can be pre calculated for all possible values of b1n b2n …bKn… We can store these in a look-up table of 2K words addressed by K bits. For e.g., if the number of inputs is 4, then the LUT will have $2^4 = 16$ memory words.

Each product term consists of a variable (signal) and a constant (coefficient) both in fixed point binary format but not necessarily of the same word length; Rather than compute the product on a term by term basis, the partial products of all terms are computed simultaneously, and in the time it would take to compute a single partial product on bit by bit basis. These partial products are generally the filter coefficients. These partial product filter coefficients of all terms are cumulated on bit by bit basis .Finally all the cumulative partial products of each bit are added and the result is produced.

In DA, all the cumulative partial product outcomes are precomputed and stored in a look up table which is addressed by the multiplier bits. All input variables are

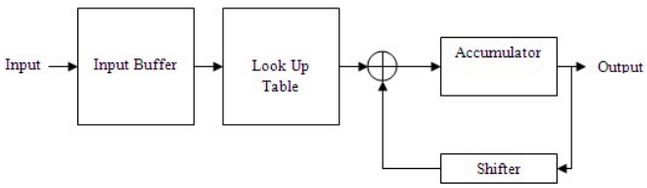

Figure 3. FIR filter using Look Up Table.

sequenced simultaneously, bit serial first to address the LUT; its outcome is added to the accumulated partial products. The complete dot product computation takes M clocks where M is the number of input variable bits, and is independent of the number of input variables. During the first iteration, the Least-Significant Bits x0(n), x0(n-1),..., of the K input samples form an K-bit address to the Look Up Table for f(x,0), and that table's output becomes the initial value of the accumulator. During the second iteration, the next-to-least significant bits x1(n), x1(n-1),..., x1(n-K+1) of the K input samples form another K-bit address to the lookup table for f(x,1), and the adder sums the Look up Table output to the contents of the accumulator shifted by one bit. This process continues until the last iteration, where the most-significant bits xN-1(n), xN-1(n- 1),..., xN-1 (n-K+1) of the K input samples form an K-bit address to the Lookup Table for f(x, N-1) and the adder sums the Look up Table output to the contents of the accumulator after shifting it to the corresponding position [5].

3.3 DA Technique for 3rd Order FIR Filter

Coefficients = 4

No.of inputs = 4

LUT size = 2^4 = 16 memory location

In this method possible outputs are pre computed and stored in LUT.LUT addressed through input of the filter. For 4 tap filter,4 tap represents the no. of co-efficient of the filter as well as it represents the no. of inputs to the filter and address bit for the LUT.

Each location has different output for the corresponding inputs .The possible inputs for this filter is 0(0000) - 15(1111).For each input the computation of output is easy by using this technique.

- Input = 1011 means
- Output = $1.h_0 + 0.h_1 + 1.h_2 + 1.h_3$
  $= h_0+h_2+h_3$
- Input = 1111 means
  Output = $h_0+ h_1+h_2+h_3$
- Input = 0101 means
  Output = h_1+h_3
- Input = 1010 means
  Output = h_0+h_2

It represents the addition of high level input co-efficient. We can easily find 16 output for corresponding input without any mathematical calculation. Table 1 shows the content of the LUT for 3rd order filter.

Table 1. LUT table for 3rd order filter

Address	Data
0000	0
0001	h_3
0010	h_2
0011	$h_2 + h_3$
0100	h_1
0101	$h_1 + h_3$
0110	$h_1 + h_2$
0111	$h_1 + h_2 + h_3$
1000	h_0
1001	$h_0 + h_3$
1010	$h_0 + h_2$
1011	$h_0 + h_2 + h_3$
1100	$h_0 + h_1$
1101	$h_0 + h_1 + h_3$
1110	$h_0 + h_1 + h_2$
1111	$h_0 + h_1 + h_2 + h_3$

For Example:

Input = X0, X1 ,X2 ,X3

X0 = 1011=11

X1 = 1101=13

X2 = 1010=10

X3 = 1001=9

h0 = h1 = h2 = h3 = 1

Step 1:

Store the values in input buffer.

X0[0] X1[0] X2[0] X3[0] =1101

X0[1] X1[1] X2[1] X3[1] =1010

X0[2] X1[2] X2[2] X3[2] =0100

X0[3] X1[3] X2[3] X3[3] =1111

Step 2:

Read the values from LUT for corresponding values in buffer.

Output of LUT:

O1 = 0011 = 3

O2 = 0010 = 2

O3 = 0001 = 1

O4 = 0100 = 4

Step 3:

If the value is multiplied by 2, it implies left shift.

Output = O1 + Shift the value of O2 one time + Shift the value of O3 2 times + Shift value of O4 3 times.

Output = 3 + 4 + 4 + 32 = 43.

Disadvantage

A filter with N coefficients the LUT has 2^N values. For higher order filter LUT size will increase, it required more memory space.

3.4 LUT Partitioning

The above technique holds good only when we go for lower order filters. For higher order filters, the size of the LUT also increases exponentially with the order of the filter. For a filter with N coefficients, the LUT have 2^N values. This in turn reduces the performance.

Therefore, for higher order filters, LUT size to be reduced to reasonable levels. To reduce the size, the LUT can be subdivided into a number of LUTs, called LUT partitions. Each LUT partition operates on a different set of filter taps. The results obtained from the partitions are summed [4].

Suppose the length LK inner product, then Eq.6 becomes

$$y = \sum_{k=1}^{LK} A_k x_k \tag{7}$$

Then the sum can be partitioned into L independent Kth parallel DA LUTs resulting in

$$y = \sum_{l=0}^{L-1} \left[\sum_{n=0}^{N-1} X_{LI} + A_{LI+n} \right. \tag{8}$$

For 3rd order filter

- Number of partition = 2
- 2 LUT tables are used .Each has 2 inputs
- Memory location = no .of partition * 2^n
  $= 2*2^2$
  $= 8$ location
- n = number of inputs of LUT

3.5 3rd Order FIR Filter with Partition Method:

LUT is divided into LUT 1&LUT 2.Each LUT has 2 inputs and 4 memory location. It shown in figure 4.

Input = 1011 means

First 2 bits are address bit of LUT 1, output becomes 10 = h_0

Remaining 2 bits are address bit of LUT 2, output becomes 11 = $h_2 + h_3$

Output = output of LUT1 + output of LUT 2 = $h_0 + h_2 + h_3$

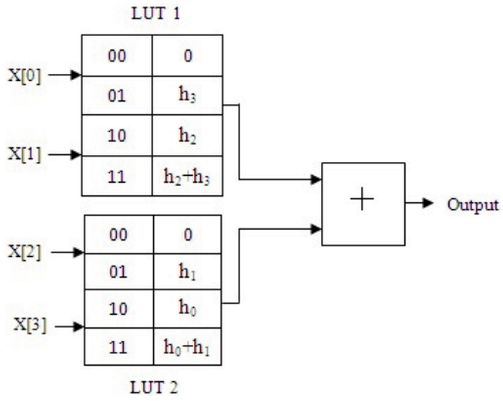

Figure 4. 3rd order FIR filter with partition method.

In this method the memory location reduced to 8 locations. Previous method 16 memory location required to produce the same output.

3.6 Summary

3rd order FIR filter:

Normal method use

4 multiplier

3 adders

In DA technique

Memory location = 2^4 =16 location

LUT partition method

Memory location = $2*2^2$ = 8 location

3.7 Realization of 16 Tap Fir Filter using Partial Tables

For 16-tap filter:

- No Partition
  Memory locations = 2^{16} = 65,536
- Partition 8
  Partial Tables = 8; each with 2 inputs
  Memory locations = 8* (2^2) = 32

It shown in Figure 5.

- Partition 4
  Partial Tables = 4; each with 4 inputs
  Memory locations = 4 * (2^4) = 64
- Partition 2
  Partial Tables = 2; each with 8 inputs
  Memory locations = 8 * (2^8) = 2048
  16 tap Fir filter:

Advantages:

- Power consumption is reduced.
- Memory access time is less than multiplication time.
- It reduces area and system latency.

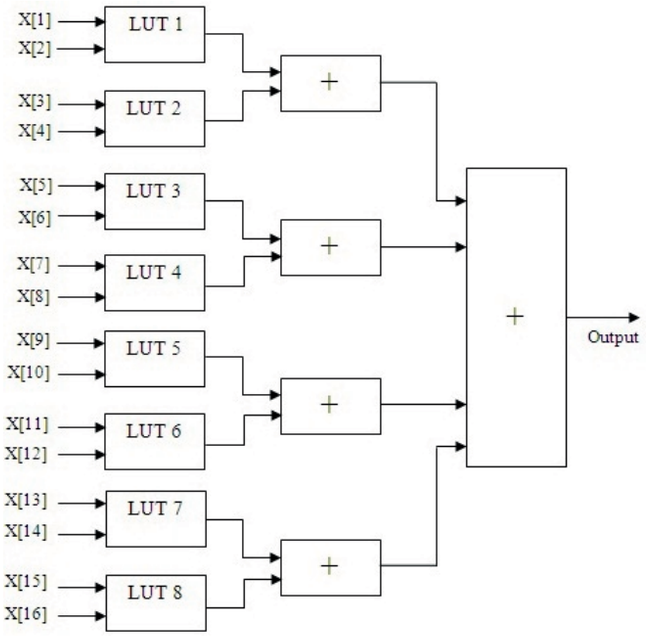

Figure 5. 16 tap Fir filter with partition 8.

4. Result Analysis

Table 2. Performance Analysis-LUT

Filter Tap	Address Size	No Partition	Partition 8	Partition 4	Partition 2
4	2	155	-	-	59
	4	255	-	-	102
	8	472	-	-	204
	16	892	-	-	408
8	2	3872	-	120	319
	4	6283	-	233	526
	8	11106	-	458	940
	16	20748	-	910	1768
16	2	-	261	647	7718
	4	-	508	1064	12495
	8	-	963	1882	22041
	16	-	1894	3534	41141

Table 2 represents the number of look up tables for various partitions.

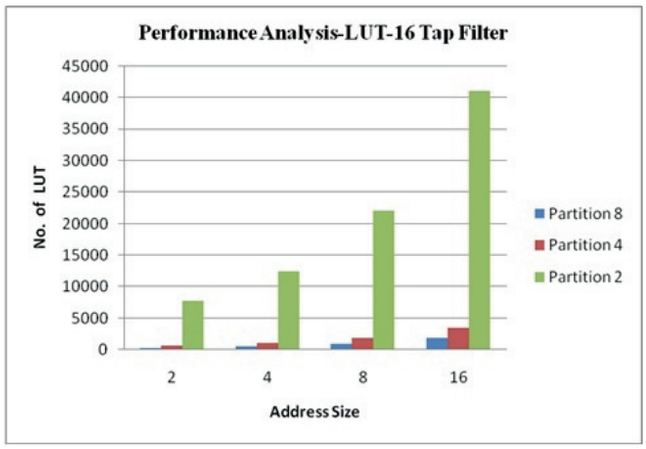

Figure 6. Performance Analysis-LUT.

The number of look up tables is reduced by using partition 8 method which shown in figure 6.

Table 3. Performance Analysis – Area (slice)

Filter tap	Address size	No partition	Partition 8	Partition 4	Partition 2
4	2	82	-	-	32
	4	134	-	-	55
	8	245	-	-	110
	16	460	-	-	221
8	2	2097	-	66	171
	4	3362	-	127	277
	8	5893	-	250	490
	16	10953	-	495	915
16	2	-	142	348	4203
	4	-	276	564	6731
	8	-	529	989	11781
	16	-	1038	1845	21886

Table 3 represents the number of slices (area) for various partitions.

The area of 16 tap FIR filter is reduced by using partition 8 method which is shown in Figure 7.

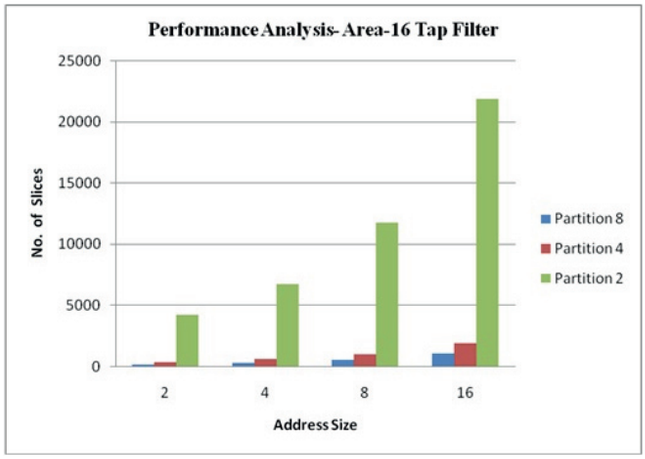

Figure 7. Performance Analysis-Area.

Table 4. Performance Analysis – Delay (ns)

Filter tap	Address size	No partition	Partition 8	Partition 4	Partition 2
4	2	24.587	-	-	19.409
	4	29.105	-	-	23.962
	8	39.853	-	-	33.128
	16	59.745	-	-	53.150
8	2	38.794	-	25.52	26.973
	4	44.825	-	30.472	32.203
	8	55.848	-	40.352	42.181
	16	76.621	-	59.904	62.217
16	2	-	29.922	30.729	40.472
	4	-	34.871	35.95	45.986
	8	-	43.69	45.873	56.884
	16	-	63.154	65.909	77.801

Table 4 represents the path delay for various partitions.

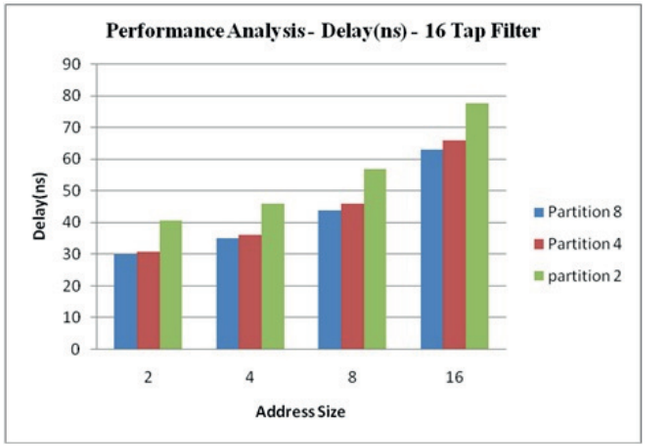

Figure 8. Performance Analysis-Delay.

The number of look up tables is reduced by using partition 8 method which shown in Figure 7.

The delay is reduced due to partition 8 method which is shown in Figure 8.

The area of 16 tap FIR filter is reduced by using partition 8 method which is shown in Figure 8.

The delay is reduced due to partition 8 method which is shown in Figure 9.

5. Conclusion and Future Work

Finite Impulse Response filter plays an important role in many Digital Signal Processing applications. In this method, the multiplier less FIR filter is implemented using Distributed Arithmetic which consists of Look Up Table and then partitioning is involved. Memory access time is less than multiplication time. LUT partition reduces memory requirements. This technique reduces the delay, area, power consumption. The performance can be further improved by pipelining all the partial tables. This architecture provides an efficient area-time power implementation which involves significantly less latency and less area-delay complexity when compared with existing structures for FIR Filter.

6. References

1. Kyung-Saeng K, Lee K (2003). Low-power and area efficient FIR filter implementation suitable for multiple tape, Very Large Scale Integration (VLSI) Systems, vol 11, No 1.
2. Meyer-Base U (2004). Digital Signal Processing with Field Programmable Gate Arrays, 2nd Edn., Chapter 2, 60–66.
3. Meyer-Base U (2004). Digital Signal Processing with Field Programmable Gate Arrays, 2nd Edn., Chapter 3, 112–113.
4. Meher P K (2006). Hardware efficient systolization of DA-based calculation of finite digital convolution of finite digital convolution, IEEE Transactions on Circuit and Systems II: Express Briefs, vol 53(8), 707–711.
5. Meher P K, Chandrasekaran S et al. (2008). FPGA realization of FIR filters by efficient and flexible systolization using distributed arithmetic, IEEE Transactions on Signal Processing, vol 56(7), 3009–3017.