International Journal of Scientific Research in Computer Science and Engineering
Research Paper Volume-3, Issue-4 ISSN: 2320-7639

Realization of FIR Filter using Distributed Arithmetic Architecture

Chandan Singh

Dept. of ECE, *Guru Gobind Singh Indraprashta University/India*
reachforchandan007@gmail.com

Available online at www.isroset.org

Received: 16 Jul 2015 Revised: 22 Jul 2015 Accepted: 23 Aug 2015 Published: 30 Aug 201

Abstract— **This work represents an efficient implementation of Finite Impulse response (FIR) filter using Distributed Arithmetic (DA) architecture. Distributed Arithmetic (DA) architecture is an efficient technique for calculation of inner product or multiply and accumulate (MAC) using Look-up tables. In absence of DA, the direct method of implementing MAC uses dedicated multipliers which are fast but consume considerable hardware.**

MAC operation is common in digital signal processing algorithms. Here DA replaces the explicit multiplications by ROM look-up tables which is an efficient technique to implement on FPGA. To save the memory units, we have divided the look-up tables. Area saving from using DA can be up to 80% in DSP hardware design. This work shows the implementation of an eight order FIR filter using DA architecture through VHDL coding and the code is finally simulated using Modelsim and then compared with MATLAB output.

Keywords:- Distributed Arithmetic (DA) architecture, VHDL. MATLAB, FIR Filter

I. Introduction

The term filter is commonly used to describe a device that discriminates according to some attribute of the objects applied to its input, what passes through it. For example, an air filter allows air to pass through it but prevents dust particles that are present in the air from passing through.
As we know that a linear time invariant system performs a type of discrimination or filtering among the various frequency components at its input. The nature of this filtering action is determined by the frequency response characteristics H(w), which in turn depends on the choice of the system parameters (e.g. the coefficients in the difference equation characterization of the system). Thus by proper selection of the coefficients, we can design frequency selective filters that pass signals with frequency components in some bands while they attenuate signals containing frequency components in other frequency bands.
In general, a linear time invariant system modifies the input signal spectrum X(w) according to its frequency response H(w) to yield an output signal with spectrum Y(w) = H(w). X(w). Here H(w) acts as a weighting function or a spectral shaping function to the different frequency components in the input signal.

Filtering is used in digital signal processing in a variety of ways. For example, removal of undesirable noise from desired signals, spectral shaping such as equalization of communication channels, signal detection signals, and so on.

Filters can be broadly defined into two categories.
(1) FIR (Finite Impulse Response) Filters and
(2) IIR (Infinite Impulse Response) filter.

As the name suggests an FIR filter is a one which has a finite duration impulse response i.e. it has an impulse response that is zero outside of some finite time interval whereas an IIR system is having an infinite duration impulse response.

An FIR filter of length M with input x(n) and output y(n) is described by the difference equation.

$$Y(n) = b_0x(n) + b_1x(n-1) + \ldots\ldots + b_{(M-1)}x(n-M+1)$$
$$= \sum_{k=0}^{M-1} b_kx(n-k)$$

Where $\{ b_k \}$ is the set of filter coefficient.

As we can see that the realization of the FIR filters needs a lots of multiplication and accumulation in its equation, a normal design of such a filter would require many multipliers and adders which can significantly increase the area on chip and consequently the cost of such a system. Therefore we require an efficient technique that will solve the area optimization problem and can reduce the area and thus the overall cost of the system.

One such technique is the **Distributed Arithmetic Architecture** which can significantly solve the need of such multipliers and adders.

II. DISTRIBUTED ARITHMETIC ARCHITECTURE

Distributed Arithmetic architecture is an efficient technique for calculation of inner product or multiply and accumulate (MAC). DA technique basically replaces the explicit multiplications by ROM look-up tables. Here multiplications are reordered (or shifted) and then added such that the arithmetic becomes distributed through the structure. It is a technique which is bit-serial in nature and hence can appear to be slow as compared to direct method of multiplier implementation. However when the number of elements in a vector is nearly the same as the word-size, DA is quite fast.

The K length FIR filter can be described as

$$y = \sum_{k=1}^{K} A_k x_k$$

eqn. 1

Let x_k be a N bit scaled 2's complement number.

$|x_k| < 1$

$x_k \{ b_{k0}\, b_{k1}\, b_{k2} \ldots . b_{k(N-1)} \}$

where b_{k0} is the sign bit

x_k can be written as

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

eqn. 2

Subsituting eqn.2 in eqn. 1

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

eqn. 3

$$y = -\sum_{k=1}^{K} (b_{k0} \bullet A_k) + \sum_{k=1}^{K} \sum_{n=1}^{N-1} (A_k \bullet b_{kn}) 2^{-n}$$

eqn. 4

Expanding the extreme right hand side part of eqn.4

$$y = -\underbrace{\sum_{k=1}^{K} (b_{k0} \bullet A_k)} + \underbrace{\sum_{k=1}^{K} \left[(A_k \bullet b_{k1}) 2^{-1} + (A_k \bullet b_{k2}) 2^{-2} + \cdots + (A_k \bullet b_{k(N-1)}) 2^{-(N-1)} \right]}$$

eqn. 5

$y = -[b_{10}.A_1 + b_{20}.A_2 + b_{30}.A_4 + b_{40}.A_5 \ldots\ldots\ldots\ldots\ldots\ldots\ldots b_{k0}.A_k]$

$+[(b_{11}.A_1)2^{-1} + (b_{12}.A_1)2^{-2} + (b13.A_1)2 \ldots\ldots\ldots (b_{1(N-1)}.A_1)2^{-(N-1)}]$

$+[(b_{21}.A_2)2^{-1} + (b_{22}.A_2)2^{-2} + (b_{23}.A_2)2^{-3} \ldots\ldots\ldots (b_{2(N-1)}.A_1)2^{-(N-1)}]$

.

.

.

$+[(b_{k1}.A_k)2^{-1} + (b_{k2}.A_k)2^{-2} + (b_{k3}.A_k)2^{-3} \ldots\ldots\ldots (b_{k(N-1)}.A_k)2^{-(N-1)}]$

Further simplification leads to (arranging coefficients with same places of decimal)

$y=-$

$[b_{10}.A_1 + b_{20}.A_2 + b_{30}.A_4 + b_{40}.A_5 \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots b_{k0}.A_k]$

$+[(b_{11}.A_1) + (b_{21}.A_2) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + (b_{k1}.A_k)]2^{-1}$

$+[(b_{12}.A_1) + (b_{22}.A_2) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots + (b_{k2}.A_k)]2^{-2}$

.

.

.

$+[(b_{1(N-1)}.A_1) + (b_{2(N-1)}.A_2) \ldots\ldots\ldots\ldots\ldots\ldots + (b_{k(N-1)}.A_k)]2^{-(N-1)}$

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

eqn. 6

Considering eqn. 6

$\left[\sum_{k=1}^{K} A_k b_{kn} \right]$ ……………………has only 2^k possible values.

$\sum_{k=1}^{K} A_k (-b_{k0})$ ……………………has only 2^k possible vaules.

Thus by storing these values in a LUT with address inputs as (kn)and after shifting it, depending upon the place of b i.e. k, and finally adding the sign bit as well, we can realize eqn. 6 which is actually a Distributed Arithmetic Architecture.

Thus for the above realization we need a ROM of size 2^k
Let's take an example to see the contents of LUT
Let number of taps k=4

Fixed coefficients are $A_1 = 2$, $A_2 = 3$, $A_3=4$, $A_4=5$

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

$2^k = 2^4 = 16$ Size ROM

$\sum_{k=1}^{K}A_k b_{kn}$ where k = 4

b_{1n}	b_{2n}	b_{3n}	b_{4n}	Contents in ROM
0	0	0	0	0
0	0	0	1	$A_4 = 5$
0	0	1	0	$A_3=4$
0	0	1	1	$A_3+A_4 = 9$
0	1	0	0	$A_2 = 3$
0	1	0	1	$A_2+A_4 = 8$
0	1	1	0	$A_2+A_3 = 7$
0	1	1	1	$A_2+A_3+A_4 = 12$
1	0	0	0	$A_1 = 2$
1	0	0	1	$A_1+A_4 = 7$
1	0	1	0	$A_1+A_3= 6$
1	0	1	1	$A_1+A_3+A_4 = 11$
1	1	0	0	$A_1+A_2 = 5$
1	1	0	1	$A_1+A_2+A_4 = 10$
1	1	1	0	$A_1+A_2+A_3 = 9$
1	1	1	1	$A_1+A_2+A_3+A_4= 14$

Table 1

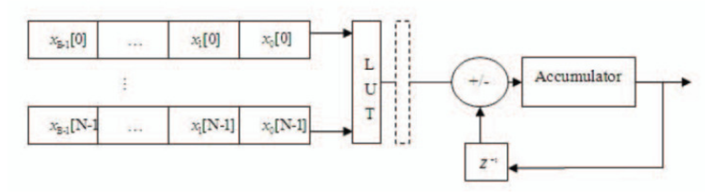

Fig. 1 Basic DA architecture.

III. DESIGN OF FIR FILTER

Now we will design a low pass filter of desired specifications as shown below using window technique.
Order of the filter = 8

Cut of frequency Fc = 1.5MHz

Sampling frequency Fs = 5MHz

Window used = Hamming window

First we will find the filter coefficients using Matlab command. We have used hamming window for this purpose.
h(n) = { 0.0022 -0.0320 0.0418 0.4880 0.4880 0.0418 -0.0320 0.0022}
These filter coefficients are first converted to 2's complement form to be used in the code. For this purpose we have taken binary values only upto 11 places of decimal.

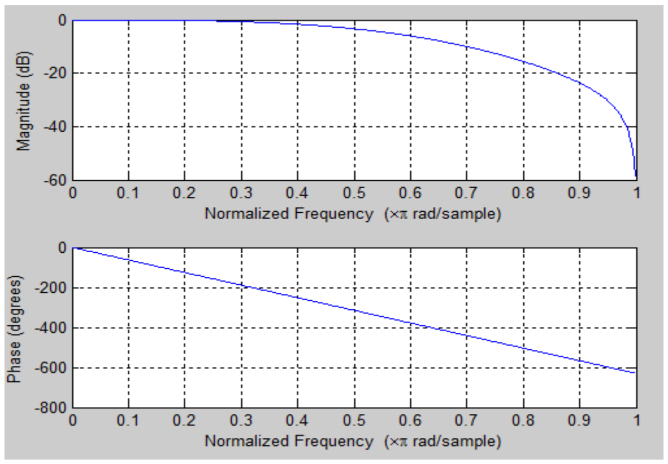

Fig 2. Frequency and phase response of the filter using Matlab

Now we will use these coefficients to design the filter using DA architecture.
The code for the same is written in VHDL which is then simulated using Modelsim. The results are then compared with Matlab output.

IV. SIMULATION OF FIR FILTER USING MODELSIM SE 6.4

Simulation of the filter is done by Modelsim SE 6.4.
Now if we closely examine the code what we have written and what should be the equation of the FIR filter, we will see a difference in the equation.

Code equation
$Y=\sum_{n=0}^{N}h_n x(n)$ eqn. 7
Actual Filter equation
$Y(n) = \sum_{k=0}^{K-1}h_k\, x(n-k)$ eqn. 8
By closely examining the Filter equation, it is apparent that the inputs given to the filter (X1-X7) should have some format.

For example,
Suppose the input signal given to the Filter is
xin = {x(0) x(1) x(2) x(3) x(4) x(0) x(1) x(2) x(3) x(4)……}
Where x(0) is the value of the Xin at 0 interval. So we will discard any value of Xin in the Filter equation which goes before 0 interval i.e. for negative value of time.
Output at different time intervals can be calculated using eqn. 8
Y(0) = h(0)x(0)
Y(1) = h(0)x(1) + h(1)x(0)
Y(2) = h(0)x(2) + h(1)x(1) + h(2)x(0)
Y(3) = h(0)x(3) + h(1)x(2) + h(2)x(1) + h(3)x(0)
Y(4) = h(0)x(4) + h(1)x(3) + h(2)x(2) + h(3)x(1) + h(4)x(0)
.
.
.
Y(10) = h(0)x(10) + h(1)x(9) + h(2)x(8) + h(3)x(7) + h(4)x(6) + h(5)x(5) + h(6)x(4) + h(7)x(3) #value of h(n) is only uptoh(0)- h(7) due the order of the filter i.e. order 8
.
.
Filter i/p Output Y(n)

	Y(0)	Y(1)	Y(2)	Y(3)	Y(4)	Y(5)	Y(6)	Y(7)	Y(8)	Y(9)	Y(10)	…
X(0)	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	…
X(1)	00	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	x(1)	x(2)	x(3)	x(4)	…
X(2)	00	00	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	x(1)	x(2)	x(3)	…
X(3)	00	00	00	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	x(1)	x(2)	…
X(4)	00	00	00	00	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	x(1)	…
X(5)	00	00	00	00	00	x(0)	x(1)	x(2)	x(3)	x(4)	x(0)	…
X(6)	00	00	00	00	00	00	x(0)	x(1)	x(2)	x(3)	x(4)	…
X(7)	00	00	00	00	00	00	00	x(0)	x(1)	x(2)	x(3)	…

Table 2

In the above table, each Y(n) is a sum of the values of X(0)-X(7) of the respective column.
By evaluating the above table, it is clear that the first input to Filter code i.e. X(0) is same as the input signal given for processing through the Filter and rest of the inputs to the Filter code i.e from X(1)-X(7) are just the delayed version of the same.
Thus we need to write an input testbench stimuli which should follow the above sequence while giving the inputs to the Filter w.r.t. to the original input signal taken for processing through the low pass filter.

V. Analysis of FIR filter with an Input signal

Signal taken as input to the Filter for processing is a square wave with frequency 1MHz and sampling frequency of 5MHz having amplitude 0.5.
The same signal is also analyzed using Matlab for comparison with Filter output.

We will have 5 samples per cycle. Inputs used are in hexadecimal
0.5 => 40
-0.5 => C0

Input Stimuli using Macro in Modelsim

```
## laoding design for simulation. Complied file is my_lib.fir
vsim my_lib.fir

## Enabing wave window
view wave

## Deleting previous waves from prior simulation, if any.
delete wave *

## Adding signals to wave window.
add wave *

##Setting zeorth  input X0, value changes after every 200ns.
force -deposit x0 16#00, 16#40 200, 16#40 400, 16#40 600, 16#C0 800, 16#C0 1000, 16#40 1200, 16#40 1400, 16#40 1600, 16#C0 1800, 16#C0 2000, 16#40 2200, 16#40 2400, 16#40 2600, 16#C0 2800, 16#C0 3000, 16#40 3200, 16#40 3400, 16#40 3600, 16#C0 3800, 16#C0 4000
```

```
##Setting first input X1, value changes after every 200ns.
force -deposit x1 16#00, 16#00 200, 16#40 400, 16#40 600,
16#40 800, 16#C0 1000, 16#C0 1200, 16#40 1400, 16#40
1600, 16#40 1800, 16#C0 2000, 16#C0 2200, 16#40 2400,
16#40 2600, 16#40 2800, 16#C0 3000, 16#C0 3200, 16#40
3400, 16#40 3600, 16#40 3800, 16#C0 4000

##Setting second input X2, value changes after every
200ns.
force -deposit x2 16#00, 16#00 200, 16#00 400, 16#40 600,
16#40 800, 16#40 1000, 16#C0 1200, 16#C0 1400, 16#40
1600, 16#40 1800, 16#40 2000, 16#C0 2200, 16#C0 2400,
16#40 2600, 16#40 2800, 16#40 3000, 16#C0 3200, 16#C0
3400, 16#40 3600, 16#40 3800, 16#40 4000

##Setting third input X3, value changes after every 200ns.
force -deposit x3 16#00, 16#00 200, 16#00 400, 16#00 600,
16#40 800, 16#40 1000, 16#40 1200, 16#C0 1400, 16#C0
1600, 16#40 1800, 16#40 2000, 16#40 2200, 16#C0 2400,
16#C0 2600, 16#40 2800, 16#40 3000, 16#40 3200, 16#C0
3400, 16#C0 3600, 16#40 3800, 16#40 4000

##Setting fourth input X4, value changes after every 200ns.
force -deposit x4 16#00, 16#00 200, 16#00 400, 16#00 600,
16#00 800, 16#40 1000, 16#40 1200, 16#40 1400, 16#C0
1600, 16#C0 1800, 16#40 2000, 16#40 2200, 16#40 2400,
16#C0 2600, 16#C0 2800, 16#40 3000, 16#40 3200, 16#40
3400, 16#C0 3600, 16#C0 3800, 16#40 4000

##Setting fifth input X5, value changes after every 200ns.
force -deposit x5 16#00, 16#00 200, 16#00 400, 16#00 600,
16#00 800, 16#00 1000, 16#40 1200, 16#40 1400, 16#40
1600, 16#C0 1800, 16#C0 2000, 16#40 2200, 16#40 2400,
16#40 2600, 16#C0 2800, 16#C0 3000, 16#40 3200, 16#40
3400, 16#40 3600, 16#C0 3800, 16#C0 4000

##Setting sixth input X6, value changes after every 200ns.
force -deposit x6 16#00, 16#00 200, 16#00 400, 16#00 600,
16#00 800, 16#00 1000, 16#00 1200, 16#40 1400, 16#40
1600, 16#40 1800, 16#C0 2000, 16#C0 2200, 16#40 2400,
16#40 2600, 16#40 2800, 16#C0 3000, 16#C0 3200, 16#40
3400, 16#40 3600, 16#40 3800, 16#C0 4000

##Setting seventh input X7, value changes after every
200ns.
force -deposit x7 16#00, 16#00 200, 16#00 400, 16#00 600,
16#00 800, 16#00 1000, 16#00 1200, 16#00 1400, 16#40
1600, 16#40 1800, 16#40 2000, 16#C0 2200, 16#C0 2400,
16#40 2600, 16#40 2800, 16#40 3000, 16#C0 3200, 16#C0
3400, 16#40 3600, 16#40 3800, 16#40 4000

## setting reset value which is set at 0.
force -deposit reset 0 0
##Running simulation for 4200ns
run 4200
```

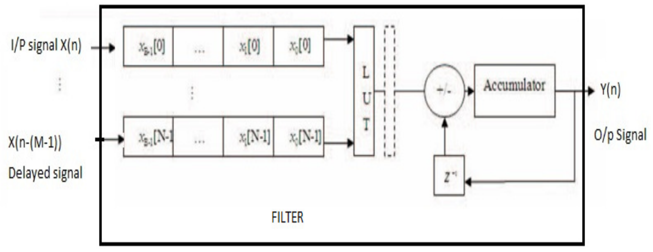

Fig. 3 FIR Filter using DA architecture

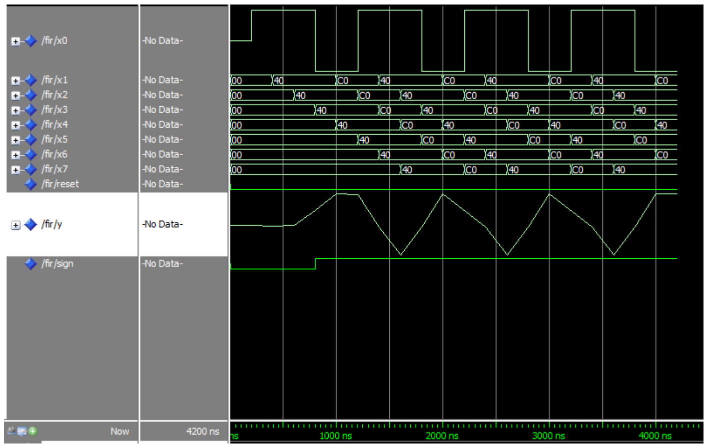

Fig. 4 Output waveforms of FIR filter from Modelsim

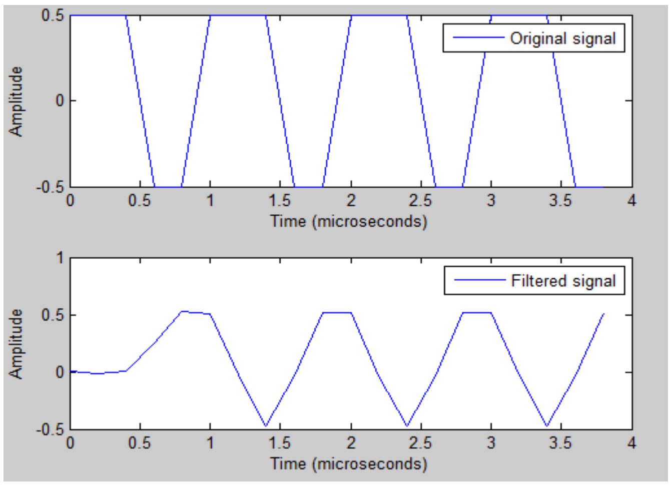

Fig. 5 Output waveforms of FIR filter from Matlab

VI. RESULT & CONCLUSION

As can be seen from the waveform comparison in the last section, the output waveform of the Modelsim (FIR filter) and the MATLAB matches with each other.

Thus the implemented FIR filter using DA architecture is suitable for the desired low pass filtering without using any multiplier. It can be used for any order low pass filter by changing the equation used and the corresponding change in the vhdl code. Further improvement can be done by taking into account the property of even symmetry of FIR filters and hence further reduction can be done in the requirement of memory units.

REFERENCES

[1]. Ramesh.R, Nathiya.R "Realization of FIR filter using Modified Distributed Arithmetic Architecture" An internation journal Vol **3.** No. **1**, February **2012**.

[2]. Yajun Zhou ; Sch. of Autom., HangZhou Dianzi Univ., Hangzhou, China ; Pingzheng Shi "Distributed Arithmetic for FIR Filter implementation on FPGA" IEEE 2011 International Conference on Multimedia Technology (ICMT), pp 294 - 297July **2011** Print ISBN: **978-1-61284-771-9**, DOI: 10.1109/ICMT.2011.6003032

[3]. Sang Yoon Park, Pramod Kumar Meher "Low-Power, High-Throughput, and Low-Area Adaptive FIR Filter Based on Distributed Arithmetic" IEEE Transactions on circuits and systems-II: Express briefs, Vol. **60**, N0. **6**, pp. 346-350 June **2013**.

[4]. M Surya Prakash, Rafi Ahamed Shaik "Low-Area & High-Throughput Architecture for an Adaptive filter using Distributed Arithmetic" IEEE Transaction on Circuit & System, Vol. **60**, No. **11**, pp. 781-785, Nov. **2013**