International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

# A VHDL Implementation of Direct, Pipelined and Distributed Arithmetic FIR Filters

**Sucharitha. L**
M.Tech. Student, CVSR College of Engineering
sucharithalekkala18@gmail.com

**M. Gopi**
Professor, CVSR College of Engineering
emtlstld@gmail.com

***Abstract* — Digital filters are typically used to modify or alter the attributes of a signal in the time or frequency domain. In this project, various FIR filter structures will be studied and implemented in VHDL.**

**Basic arithmetic blocks to carry out DSP on FPGAs will be discussed. The very popular LUT based approach for arithmetic circuit implementation will be presented. The conventional PDSP MAC and Distributed arithmetic MAC units will be implemented and their performance will be compared. Usage of Pipelining in multipliers for improving the speed will also be discussed. The ModelSim XE simulator will be used to simulate the design at various stages. Xilinx synthesis tool (XST) will be used to synthesize the design for spartan3E family FPGA (XC3S500E). Xilinx Placement & Routing tools will be used for backend, design optimization and I/O routing.**

***Keywords* — PDSP MAC, DA, IIR, DSP.**

## I. INTRODUCTION

In signal processing, the function of a *filter* is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. The following block diagram illustrates the basic idea.

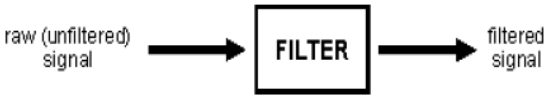



Fig.1.1. Filter basic Functionality

There are two main kinds of filter, analog and digital. They are quite different in their physical makeup and in how they work. An analog filter uses analog electronic circuits made up from components such as resistors, capacitors and op-amps to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, video signal enhancement, graphic equalizers in hi-fi systems, and many other areas.

There are well-established standard techniques for designing an analog filter circuit for a given requirement at all stages, the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialized DSP (Digital Signal Processor) chip.

The analog input signal must first be sampled and digitized using an ADC (analog to digital converter). The resulting binary numbers, representing successive sampled values of the input signal, are transferred to the processor, which carries out numerical calculations on them. These calculations typically involve multiplying the input values by constants and adding the products together. If necessary, the results of these calculations, which now represent sampled values of the filtered signal, are output through a DAC (digital to analog converter) to convert the signal back to analog form.

Note that in a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current. The following diagram shows the basic setup of such a system.

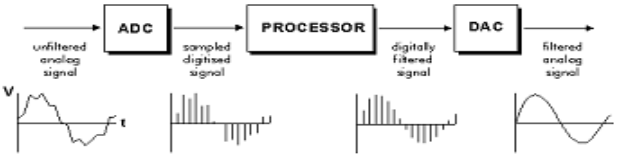



Figure1.2 Basic setup for Digital processor

### *Operation of Digital filters*

In this section, the basic theory of the operation of digital filters is presented. This is essential to an understanding of how digital filters are designed and used. Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function $V = x(t)$ where $t$ is time.

This signal is sampled at time intervals $h$ (the sampling interval). The sampled value at time $t = ih$ is $X_i = x(ih)$

Thus the digital values transferred from the ADC to the processor can be represented by the sequence $x_0$ , $x_1$ , $x_2$ , $x_3$ , ... corresponding to the values of the signal waveform at t = 0, h, 2h, 3h, ...and $t = 0$ is the instant at which sampling begins.

At time $t = nh$ (where $n$ is some positive integer), the values available to the processor, stored in memory, are $x_0$ , $x_1$ , $x_2$ , $x_3$ , ... $x_n$. Note that the sampled values $x_{n+1}$, $x_{n+2}$ etc. are not available, as they haven't happened yet! The digital output from the processor to the DAC consists of the sequence of values $y_0$ , $y_1$ , $y_2$ , $y_3$ , ... $y_n$ In general, the value of $y_n$ is calculated from the values $x_0$ , $x_1$ , $x_2$ , $x_3$ , ... $x_n$. The way in which the $y$'s are calculated from the $x$'s determines the filtering action of the digital filter.

The *order* of a digital filter is the number of *previous* inputs (stored in the processor's memory) used to calculate the current output. Filters may be of any order from zero upwards.

## II. FIR FILTER DESIGN WITH FDA TOOL

### *FIR Filter Structures*

In this project a low pass FIR filter with following specifications is chosen for implementation. Figure 4.2 shows the GUI of FDA tool with these specifications.

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

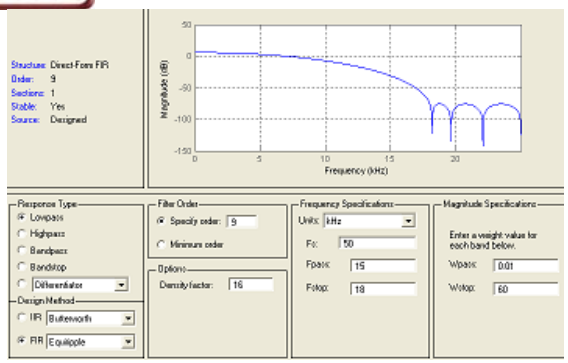



Fig.2.1. Filter Design Analysis tool with chosen low pass filter specifications

*Filter specifications*

Sampling Frequency $F_s$ = 50 KHz
Pass Band Edge Frequency = 15 KHz
Stop Band Edge frequency = 18 KHz
Weight in the pass band = 0.01
Weight in the stop band = 60
FIR filter type = Equiripple

The figure 2.1 shows the magnitude response of the filter. The phase response of the filer observed from FDA tool is given in the below figure

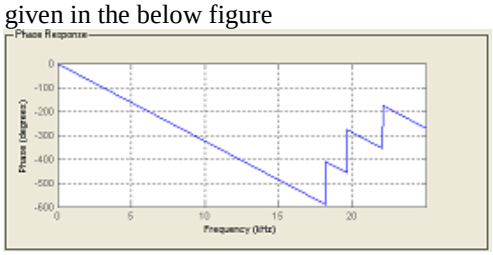



Fig.2.2. Phase response of the Low pass FIR filter

It can be observed from figure 2.2 the phase characteristic of the filter are linear as expected for FIR filter. The filter coefficients for the given specifications are observed from the FDA tool. The following figure gives the output of FDA tool showing the filter coefficients.

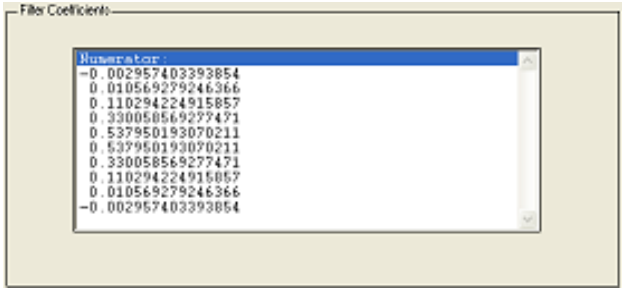



Fig.2.3. The filter coefficients observed from FDA tool

The impulse response and step response of the FIR filter from FDA tool are shown in the below figures.

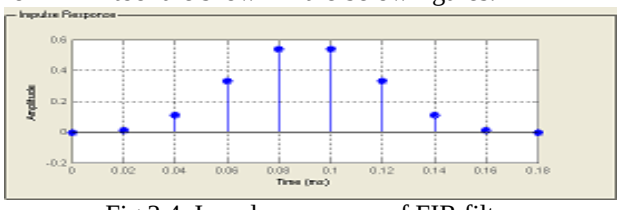



Fig.2.4. Impulse response of FIR filter (observed from FDA tool)

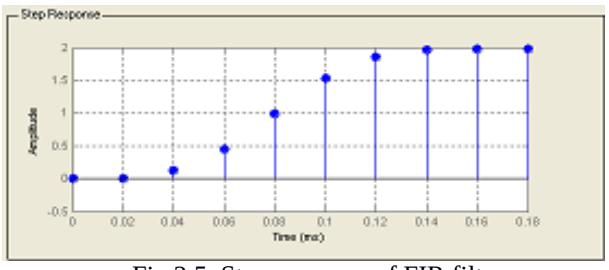



Fig.2.5. Step response of FIR filter (observed from FDA tool)

*Direct form FIR filter*

The direct form of FIR filter is standard linear convolution, which described the output as convolution of input and impulse response of the filter.

$$y[n]=x[n]*c[n]=\sum x[k]c[n-k]=\sum c[k]x[n-k]$$

Where c[n] values represent filter coefficients, and x[n] represents the input samples. The figure 1.5 shows the direct form FIR structure

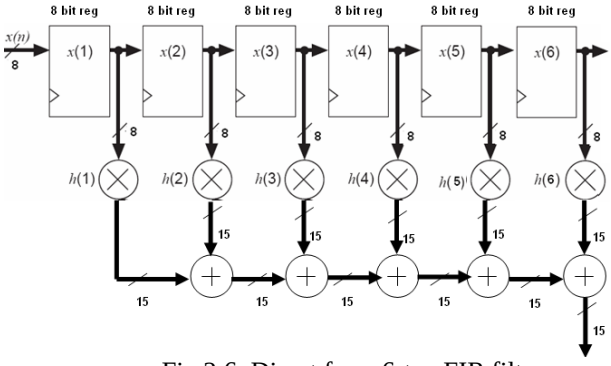



Fig.2.6. Direct form 6-tap FIR filter.

*Transposed form FIR structure*

A variation of the direct FIR model is called the transposed FIR filter. It can be constructed from the FIR filter in Fig 2.7 by:

1. Exchanging the input and output
2. Inverting the direction of signal flow
3. Substituting an adder by a fork, and vice versa

A transposed FIR filter is shown in Fig and is, in general, the preferred implementation of an FIR filter.

The benefit of this filter is that there is no need for an extra pipeline stage for the adder (tree) of the products to achieve high throughput.

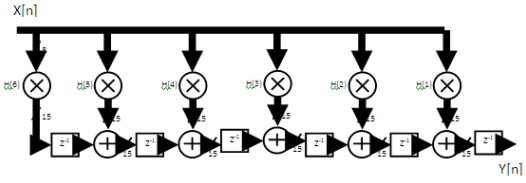



Fig.2.7. Transposed form structure for FIR filter.

The structure shown in figure implemented in VHDL. Since functionality wise the transposed structure is same as direct form structure the impulse response is same for both. Hence the transposed form filter behavior is same as direct form for any input. Due to this reason the simulated output for this structure are not shown here, only the synthesis report is presented here.

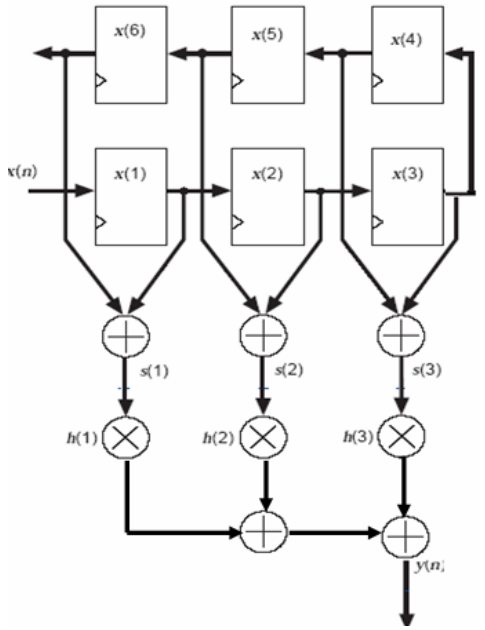
International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

*Symmetric FIR Filter structure*

The chosen FIR impulse response is symmetric, i.e., h(1)=h(6), h(2)=h(5), and h(3)=h(4). The output equation can be modified as below

Y[n] = (x[n] + x[n-5]) * h(1) + (x[n-1] + x[n-2]) * h(2) + (x[n-3] + x[n-4]) * h(3)

This approach leads to more efficient structure, with only three multiplications




2.8. Symmetric structure FIR filter

*FIR filter using Distributed Arithmetic*

A completely different FIR architecture is based on the distributed on the distributed arithmetic (DA) concept introduce in section chapter 4. In contrast to conventional sum-of-products architecture, in distributed arithmetic we always compute the sum of products of a specific bit b over all coefficients in one step. This is computed using a small table and an accumulator with shifter. The structure of Distributed Arithmetic FIR filter is shown in the below figure.

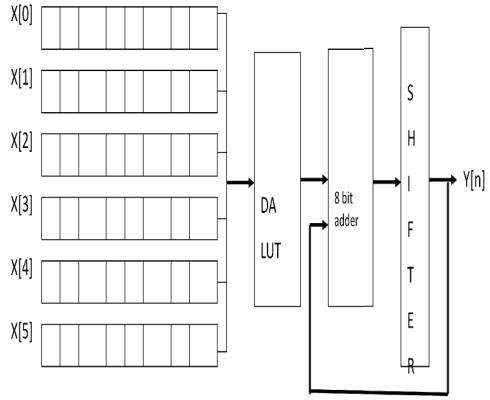



Fig.2.8 DA FIR filter structure

The structure shown in Figure 2.8 is implemented in VHDL. The shift and add unit which is main logic in Distributed arithmetic is implemented in structural style using n bit adder and PIPO register as components. The shift and add circuit shift the existing number one bit to right and adds the next number to it.

The top level FIR filter module is implemented in structural style using Look up Table and shift-and-add circuit as components.

## III. Distributed Arithmetic Fundamentals

Distributed arithmetic (DA) is an important FPGA technology. It is extensively used in computing the sum of products

$$y=(c,x)=\sum_{n=0}^{N-1}c[n]x[n]$$

Besides convolution, correlation, DFT computation can also be formulated as such a "Sum of products" (SOPs). Completing a filter cycle, when using a conventional arithmetic unit, would take approximately N MAC cycles. This amount can be shortened with pipelining but can, nevertheless, be prohibitively long. This is a fundamental problem when general purpose multipliers are used.

In many DSP applications, a general purpose multiplication is technically not required if The filter coefficients, c[n] are known apriori, then technically the partial product term c[n]x[n] becomes a multiplication with a constant (i.e., scaling) This is an important difference and is a prerequisite for a DA design.

To understand the DA design paradigm, consider the "sum of products" inner product shown below:

$$y=(c,x)=\sum_{n=0}^{N-1}c[n].x[n]=c[0]x[0]+c[1]x[1]+.....+c[N-1]x[N-1]$$

Assume further that the coefficients c[n] are known constants and x[n] is a variable. An unsigned DA system assumes that the variable x[n] is represented by:

$$x[n]=\sum_{b=0}^{B-1}x_b[n].2^b$$

with $x_b[n]\in[0,1]$, where $x_b[n]$ denotes the $b^{th}$ bit of x[n], i.e., the $n^{th}$ sample of x. The inner product y can therefore, be represented as:

$$y=\sum_{n=0}^{N-1}c[n].\sum_{b=0}^{B-1}x_b[n].2^b$$

Re-distributing the order of summation (thus the name "distributed arithmetic") results in:

$$\begin{aligned}y=\;&c[0]\left(x_{B-1}[0]2^{B-1}+x_{B-2}[0]2^{B-2}+...+x_0[0]2^0\right)\\&+c[1]\left(x_{B-1}[1]2^{B-1}+x_{B-2}[1]2^{B-2}+...+x_0[1]2^0\right)...\\&+c[N-1]\left(x_{B-1}[N-1]2^{B-1}+....+x_0[N-1]2^0\right)\end{aligned}$$

By taking $2^b$ common from the all terms, we get

$$\begin{aligned}y=\;&\left(c[0]x_{B-1}[0]+c[1]x_{B-1}[1]+....+c[N-1]x_{B-1}[N-1]\right)2^{B-1}\\&+\left(c[0]x_{B-2}[0]+c[1]x_{B-2}[1]+.....+c[N-1]x_{B-2}[N-1]\right)2^{B-2}\\&.....+\left(c[0]x_0[0]+c[1]x_0[1]+....c[N-1]x_0[N-1]\right)2^0\end{aligned}$$

The above equation can be rewritten as

$$y=\sum_{b=0}^{B-1}2^b.\sum_{n=0}^{N-1}c[n].x_b[n]$$

International Journal of Electronics Communication and Computer Engineering
Volume 4, Issue 2, ISSN (Online): 2249–071X, ISSN (Print): 2278–4209

implementation of the function f(c[n], $x_b$[n]) requires special attention. The preferred implementation method is to realize the mapping f(c[n], $x_b$[n]) using one LUT. That is a, $2^N$ –word LUT is preprogrammed to accept an N-bit input vector $x_b$ = [$x_b$[0],$x_b$[1],…,$x_b$[N-1]], and output f(c[n],$x_b$[n]). The individual mappings f(c[n],$x_b$[n]) are weighted by the appropriate power-of-two factor and accumulated. The accumulation can be efficiently implemented using a shift-adder as shown in Figure 3.2. After N look-up cycles the inner product y is computed.

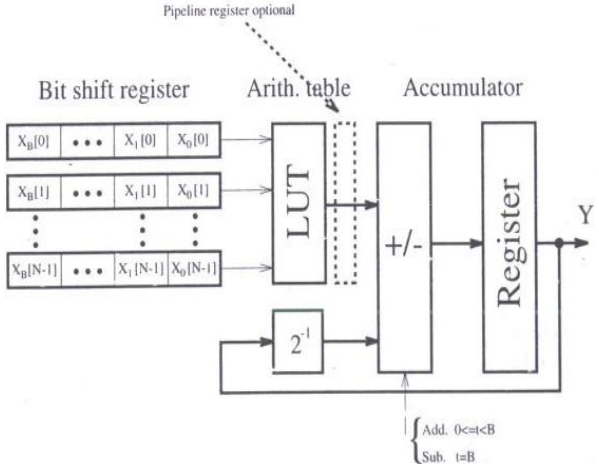



Fig.3.1. Distributed Arithmetic MAC unit.

With LUT f(c[n], $x_b$[n]) will be realized as below

$$f(c[n], x_b[n]) = \sum_{n=0}^{N-1} c[n].x_b[n]$$

Hence $y[n] = \sum_{b=0}^{B-1} 2^b . f(c[n], x_b[n])$

Consider an example. A third-order innerproduct is defined by the inner product equation

$$y = (c, x) = \sum_{n=0}^{2} c[n]x[n]$$

Assumes that the 3-bit coefficients have the Values c[0] = 2, c[1] = 3, and c[2] = 1. The resulting LUT, which implements f(c[n], $x_b$[n]), is defined below:

| $x_b$[2] | $x_b$[1] | $x_b$[0] | f(c[n],x[n]) |
|---|---|---|---|
| 0 | 0 | 0 | $1.0+3.0+2.0 = 0_{10} = 000_2$ |
| 0 | 0 | 1 | $1.0+3.0+2.1 = 2_{10} = 001_2$ |
| 0 | 1 | 0 | $1.0+3.1+2.0 = 3_{10} = 011_2$ |
| 0 | 1 | 1 | $1.0+3.1+2.1 = 5_{10} = 101_2$ |
| 1 | 0 | 0 | $1.1+3.0+2.0 = 1_{10} = 001_2$ |
| 1 | 0 | 1 | $1.1+3.0+2.1 = 3_{10} = 011_2$ |
| 1 | 1 | 0 | $1.1+3.1+2.0 = 4_{10} = 100_2$ |
| 1 | 1 | 1 | $1.1+3.1+2.1 = 6_{10} = 110_2$ |

The inner-product, with respect to x[n] = {x[0] = $1_{10}$ = $001_2$, x[1] = $3_{10}$ = $011_2$, x[2] = $7_{10}$ = $111_2$}, is obtained as follows:

| Step t | $x_t$[2] | $x_t$[1] | $x_t$[0] | f[t]+ACC[t-1] = ACC[t] |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | $6.2^0$+0= 6 |
| 1 | 1 | 1 | 0 | $4.2^1$+6= 14 |
| 2 | 1 | 0 | 0 | $1.2^2$+14= 18 |

We must get the same result by direct MAC operation

$$y = (c, x) = c[0]x[0] + c[1]x[1] + c[2]x[2]$$

= 2.1+3.3+1.7 = 18. (As expected)

For a hardware implementation, instead of shifting each intermediate value by b (which will demand an expensive barrel shifter) it is more appropriate to shift the accumulator content itself in each iteration one bit to the right. It is easy to verify that this will give the same results. The structure of Distributed Arithmetic MAC based on this type of adder is shown in figure 3.2.

Assume that a LUT and a general purpose multiplier have the same delay T = T(LUT) = T(MUL). The computational latencies are then BT(LUT) for DA and $N_T$(MUL) for the PDSP. In the case of small-bit-width B, the speed of the DA design can therefore be significantly faster than a MAC-based design. In the next chapter the speed of distributed arithmetic MAC is compared with conventional MAC.

## IV. RESULTS

*Comparison*

Area Comparison

| Components (Available) | FIR Structure | | | |
|---|---|---|---|---|
| | Direct Structure | Transposed Structure | Symmetric structure | Distributed Arithmetic Structure |
| Slices (4656) | 155 | 102 | 113 | 39 |
| Flip flops (9312) | 128 | 109 | 128 | 29 |

Speed Comparison

| | FIR Structure | | | |
|---|---|---|---|---|
| | Direct Structure | Transposed Structure | Symmetric structure | Distributed Arithmetic Structure |
| Minimum period | 28.335ns | 10.261ns | 19.167ns | 5.422ns*6=32.532ns |

In Distributed Arithmetic Structure we multiplied the minimum time period with Six because we will get the output after the six clock pulses where as in all other structures we will get in single clock cycle

## V. CONCLUSION

The present work implements few FIR filter structures on spartan3E FPGAs by analyzing the tradeoff between performance and chip area. The present implemented structures makes use of only VHDL constructs, hence can be ported on any family FPGAs. The work also brings out

fundamental design goals in FPGA based design. The quantization effects due to finite word lengths are also presented. The Distributed Arithmetic usage and its advantages over conventional arithmetic are clearly discussed.

In the following sections very popular applications of FIR filters (implemented on FPGAs) are presented. The future scope for extending the project is also discussed.

*Applications of FIR filters*

A few popular applications for FIR filters are listed below:

- Echo cancellation
  - Telecommunications
  - Data communications
  - Wireless communications
- Multi-path delay compensation
- Ghosting cancellation
  - HDTV
  - DTV
  - Video processing
- Speech synthesis
- Waveform synthesis
- Filtering
  - High-speed modems
- ISDN

## VI. Future Scope

(1) The implemented FIR structures at code level can be modified to make full benefit from the FPGA, such as using fast carry chains, Embedded Array Blocks etc.

(2) To achieve the peak performance fully parallel pipelined version can be implemented

(3) In the present work the DA based FIR filter is implemented with one LUT and without pipelining. This can be extended to full parallel implementation with more than one LUTs for high speed applications. The following figure shows a possible architecture.

## VII. References

[1] Digital Signal Processing with Field Programmable Gate Arrays, by U.Meyer-Baese, Springer Publications.

[2] Practical FIR Filter Design in MATLAB, Revision 1.1, Ricardo A. Losada The MathWorks, Inc. 3 Apple Hill Dr. Natick, MA 01760, USA

[3] A Guide to Using Field Programmable Gate Arrays (FPGAs) for Application-Specific Digital Signal Processing Performance by Gregory Ray Goslin Digital Signal Processing Program Manager Xilinx, Inc. 2100 Logic Dr. San Jose, CA 95124

[4] Optimization Techniques for Efficient Implementation of DSP in FPGAs. Mentor graphics Paper

[5] FPGA Optimized low power high speed FIR Filters For DSP Applications

[6] Estimating FPGA Requirements for DSP Applications from http://www.hunteng.co.uk.

[7] FPGAs for High-Performance DSP Applications form www.fpgajournal.com.

[8] Implementing FIR Filters in FLEX Devices, form altera application note73.

[9] ALTERA FLEX 10KE data sheet form www.altera.com