FPGA IMPLEMENTATION OF DIGITAL FILTERS USING DISCRETE TIME CONVOLUTION EQUATION

A dissertation submitted in partial fulfillment of the requirement for the award of the degree of

MASTER OF ENGINEERING

CONTROL & INSTRUMENTATION

By

MOHD. MAHFOOZ ALAM

Under the guidance of

Prof. PARMOD KUMAR

DEPARTMENT OF ELECTRICAL ENGINEERING
DELHI COLLEGE OF ENGINEERING

NEW DELHI – 110 042

DEPARTMENT OF ELECTRICAL ENGINEERING

DELHI COLLEGE OF ENGINEERING

CERTIFICATE

This is to certify that this report entitled," **FPGA IMPLEMENTATION OF DIGITAL FILTERS USING DISCRETE TIME CONVOLUTION EQUATION** " ,submitted by , Mohd.Mahfooz Alam in the partial fulfillment of the requirement for the award of the degree of **Master of Engineering** in **Control and Instrumentation** ,embodies the work done under my supervision.

Prof. Parmod Kumar
Head of Department
DCE, Delhi.

Date:

Abstract

FPGAs (Filed programmable Gate Array) have become a competitive alternative for high performance DSP applications, previously dominated by general purpose DSP and ASIC (Application Specific Integrated Circuit) devices. This thesis describes the benefits of using an FPGA as a DSP Co-processor, as well as, a stand-alone DSP Engine. Different type of filters namely Low Pass, High pass, Band pass, Band stop filters have been designed and implemented to illustrate the possibility of designing digital filters on Field Programmable Gate Array without losing its desired characteristics and performances. The whole work has been done in two parts both in theoretical approach and practical approach. Finally, a comparison has been done with the conventional implementation approach to clarify the situation where we should use FPGA based technique.

CONTENTS

Chapters

1 Introduction

1.1 Introduction to Field Programmable Gate Array

Field-Programmable Gate Arrays (FPGAs) are a revolutionary new type of user-programmable integrated circuits that provide fast, inexpensive access to customized VLSI. An FPGA consists of an array of logic cells that can be interconnected via programmable routing switches, where the routing structures are sufficiently general to allow the configuration of multiple levels of the FPGA's logic cells. FPGAs represent a combination of the features of Mask Programmable Gate Arrays (MPGAs) and Programmable Logic Devices (PLDs). From MPGAs, FPGAs have adopted a two-dimensional array of logic cells, and from PLDs the user-programmability. The work reported in this thesis is focused on FPGA based system design in one particular application area, Digital Filter design.

Following their introduction in 1985, by the Xilinx Company [Cart86], FPGAs have evolved considerably as various new devices have been developed [ElGa88] [ElGa89] [Plus90]. FPGAs have quickly gained widespread use, which can be attributed to the reduced manufacturing time and relatively low costs of these large-capacity user-programmable devices. As an implementation medium for customized VLSI circuits, FPGAs offer following unique advantages over the alternative technologies (MPGAs, standard cells, and full custom design):

(1) FPGAs provide a reduction in the cost of manufacturing a customized VLSI circuit from tens of thousands of dollars to about one hundred dollars.

(2) FPGAs reduce the manufacturing time from months to minutes.

These advantages, which are attributable to the user-programmability of FPGAs, provide a faster time-to-market and less pressure on designers, because multiple design iterations can be done quickly and inexpensively. However, user-programmability also has drawbacks: the logic density and speed performance of FPGAs is considerably lower than those of the alternatives. While developments over the last few years have shown significant improvements in FPGAs, much research is still needed before the best FPGA designs are discovered.

1.2 Motivation of Thesis

Digital Signal Processing is omnipresent in the modern world. Almost everything – from satellites to telephones, from household appliances to sophisticated instruments, from the medical world to the musical world, uses digital signal processing in some form or other.

Filtering is the most widely used and most important operation of Digital Signal Processing. Other operations include basic operations like amplification, summation, product, differentiation, integration, modulation etc. Although in this thesis the main concentration is on filtering.

Digital filters are versatile, immune to environmental changes like temperature, aging, etc. and can be reproduced in large quantities. Another main advantage is its small size – highly complex filters can be implemented on a small chip. One chip may contain a number of filters, or it may be timeshared among multiple signals. Thus, the reason for choosing a digital implementation over its analog counterpart is self-evident.

The accuracy of a digital system depends on the word length used by its data. This can be increased by increasing the word length. Digital systems are not affected by loading when cascaded. They almost always contain memory modules, which can be used to store processed data almost indefinitely for offline processing later. Also filter coefficients can be stored in such memory to facilitate adaptive control – in which the filter characteristics can be changed on the fly, depending on the change in conditions.

Digital Filtering involves the execution of a number of algorithms, which require many calculations. For example the basic algorithm of a FIR filter is given by

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

This evidently involves a number of multiplications, additions, and also memory elements to remember the last few values of the input, the coefficients, etc. Traditionally this calculation is done on a microprocessor called a DSP (Digital Signal Processor). However, a much faster and better implementation is obtained on an FPGA with some other added advantages.

FPGAs or Field Programmable Gate Arrays are a class of reconfigurable hardware device, which is increasingly being used to perform various digital signal-processing tasks. The digital processing algorithms are implemented using logic-gates, flip-flops, and other elementary hardware elements. In a DSP processor, algorithms are written in programming languages like 'C' and each instruction is crunched by the machine serially, whereas in an FPGA, being a hardware implementation, operations are performed in parallel. Herein lies the advantage of an FPGA – it provides the flexibility of software at the speed of hardware.

A well-made FPGA design is often as much as 1000 times faster than a corresponding DSP design. Also it consumes much less power, typically 20% of a microprocessor based DSP at the same sampling rate.

As FPGAs are a relatively new technology the necessary software tools are not always readily available. Traditional software algorithms cannot be used, or if used, result in poor performance, as FPGAs follow an entirely different philosophy. Good FPGA algorithms must take into account the parallel nature of the system – such algorithms have not yet reached the tried and tested maturity of the traditional algorithms. However with the rapid infiltration of FPGAs in the industry, such disadvantages will disappear as more experience is gathered in the field. *The focus of this thesis is to study the different algorithm available in the literature for digital filters, which suits the FPGA architecture and then implementing it practically using those algorithms without loss of its generality.* Then it can also be implemented in particular situation where it fits with added potentiality.

As mentioned earlier, traditional software algorithms are not very efficient in an FPGA implementation. For example, multiplication is a computationally intensive process in any platform. Digital Signal Processing operations like filtering requires a number of multiplication operations as is evident from the equation above. To speed up this process, a multiplication algorithm is being used in this thesis, which works very well on an FPGA – Discrete time convolution equation. Details of the Discrete time convolution equation algorithm will be discussed in chapter 3. The primary focus of this thesis is to show that various digital filters can be implemented on FPGA using the available algorithms with comparable performance as on DSPs. A theoretical study also

has been done to compare the use of FPGA based filter with the conventional DSP based filter design in terms of cost, speed (filter throughput) and adaptability.

1.3 Work Approach

FPGA based filters are studied in this thesis using both an experimental and a theoretical approach. For the theoretical study, a comparison has been established between conventional MAC (multiply and accumulation) based approach and a new algorithm, Discrete time convolution equation based approach in terms of filter throughput. Filter throughput basically determines how faster the filter is. For the experimental study, different kind of filter (Low pass, Band pass, Hi pass, Notch filters) has been designed using the new algorithm, discrete time convolution equation. The Xilinx' Virtex-E chip has been used for hardware implementation.

1.4 Organization of Thesis

This thesis is organized in 7 chapters. Chapters 2, 3 summarize the literature survey and necessary background on Digital Filters, Field Programmable Gate Arrays and Algorithms. Chapter 4 presents design and implementation of different filters on FPGA. Chapter 5 presents the simulated and experimental results on performance of the different filters. Chapter 6 and Chapter 7 are in nature of conclusion and future work.

Chapter 2 summarizes from available literature the necessary background information, including Field Programmable Gate Architecture and a brief introduction to Digital Filters. It also describes design, practical implementation concerns and application of digital filters.

Chapter 3 summarizes the basics of the Discrete time convolution equation algorithm from the available literature, especially for implementation of digital filters on FPGA. The algorithm is unique in that it solves the hardware constraints problems of FPGA. This chapter also describes different approach of using Discrete time convolution equation in filter implementation on different aspect, cost and speed. At the end an optimized algorithm is also described which takes care of both speed of the filter and the total hardware cost.

In this thesis the algorithm described in Chapter 3 has been used to design the filters. The design and implementation part is described in chapter 4 in detail. This chapter has two subsections Hardware and the Software. How different discrete chips are interfaced in particular mode has been described in detail. The software subsection describes the total FPGA design flow.

Chapter 5 presents the result of practical implementation of different filters with its characteristics. Different Filter has been studied and compare with the ideal response without truncating the coefficient width.

Chapter 6 provides concluding remarks.

Field Programmable Gate Array based design is completely new comparable to conventional approach. So the question comes here how much adaptability is there with this newer kind of devices, how much cost we need to implement it and obviously in which particular case we should use FPGA. Chapter 7 considers all these aspects. The scope of future work also has been suggested at the end of this Chapter. References and Datasheets are provided at the end of the thesis.

2 FPGA and Digital Filter Basics

2.1 Introduction

This chapter introduces the two main fields, FPGA architecture and Digital Filter Design. Section 2.2 provides some necessary background information on FPGA Architecture. Section 2.3 provides a brief introduction to Digital Filters and design of filters. At the end of this section application of digital filters has been given in brief.

2.2 General FPGA Architecture

Virtex-E devices feature a flexible, regular architecture that comprises an array of configurable logic blocks (CLBs) surrounded by programmable input/output blocks (IOBs), all interconnected by a rich hierarchy of fast, versatile routing resources. The

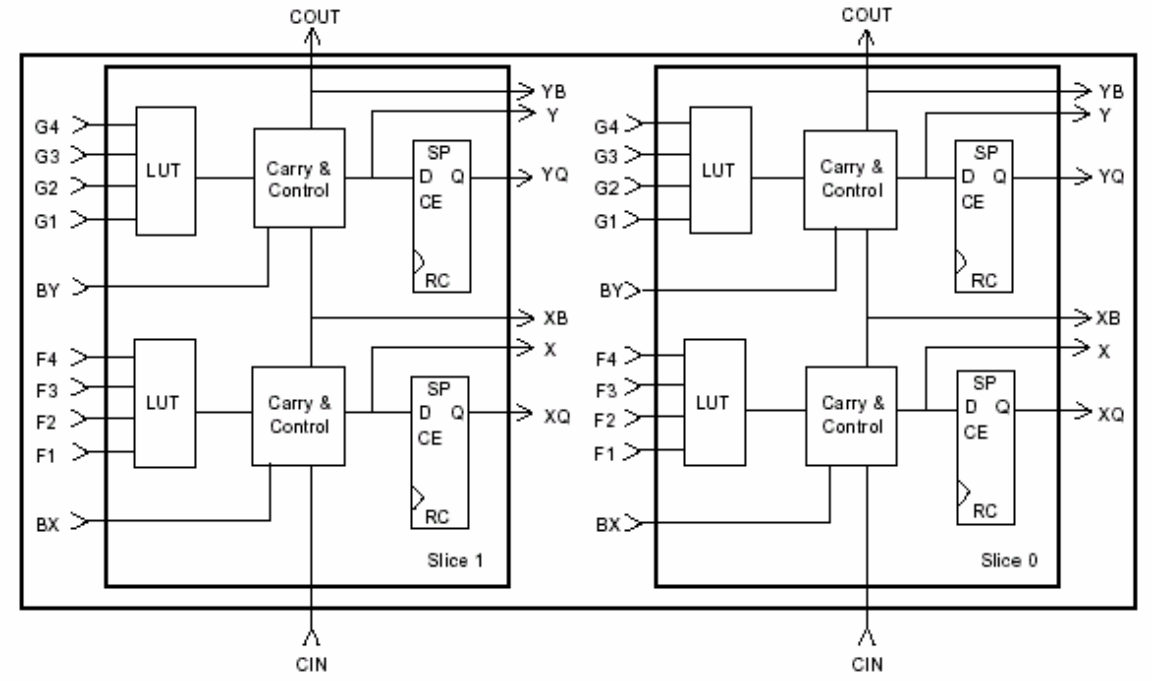


Figure 2.1: Simplified Block Diagram of a CLB

abundance of routing resources permits the Virtex-E family to accommodate even the largest and most complex designs. Virtex-E FPGAs are SRAM-based, and are

customized by loading configuration data into internal memory cells. Con- figuration data can be read from an external SPROM (master serial mode), or can be written into the FPGA SelectMAP™, slave serial, and JTAG modes). The standard Xilinx Foundation Series™ and Alliance Series™ Development systems deliver complete design support for

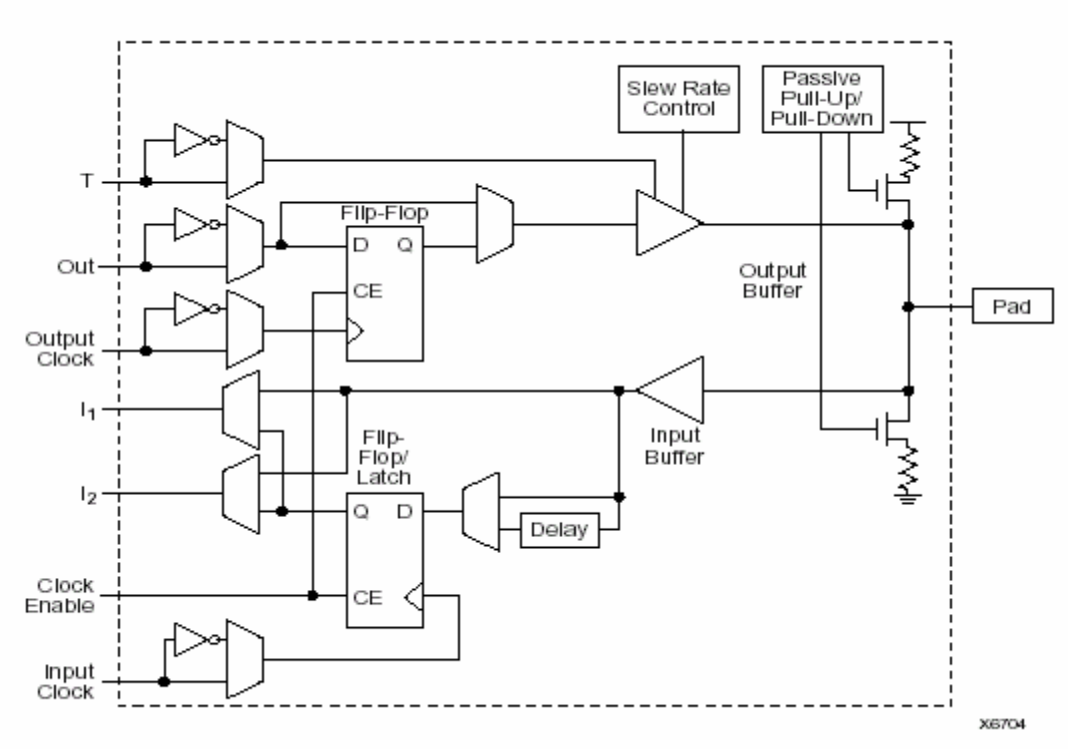


Figure 2.2: Simplified Block Diagram of a IOB

Virtex-E, covering every aspect from behavioral and schematic entry, through simulation, automatic design translation and implementation, to the creation and downloading of a configuration bit stream.

2.2.1 Features of Virtex-E Series FPGAs

Within one year of launching the original Virtex series, Xilinx has raised the bar yet again by introducing the next generation 1.8-volt Virtex-E family that enhances all aspects of the Virtex attributes. Fabricated on a leading edge 0.18 um, six-layer metal silicon process, the Virtex-E family has significantly increased both performance and density, while providing a high-performance system level feature set that further

addresses the bandwidth requirements of the next generation data communication and DSP applications. The advanced high-performance feature set of the Virtex series includes:

- Densities ranging from 50,000 to 3.2 million system gates
- Support for 20 I/O standards, including three differential signaling standards
- Over 311 Mbps single-ended I/O performance
- Up to 832 Kbits of internal True Dual-Port(TM) BlockRAM
- 8 DLLs for 311+ MHz clock management
- Up to 804 single-ended I/Os or 344 differential I/O pairs
- Direct interfacing to high performance memory devices

Here in this implementation Xilinx XCV300EPQ240 has been used for prototyping. The details of this device could be referred from [DB2000, Xilinx].

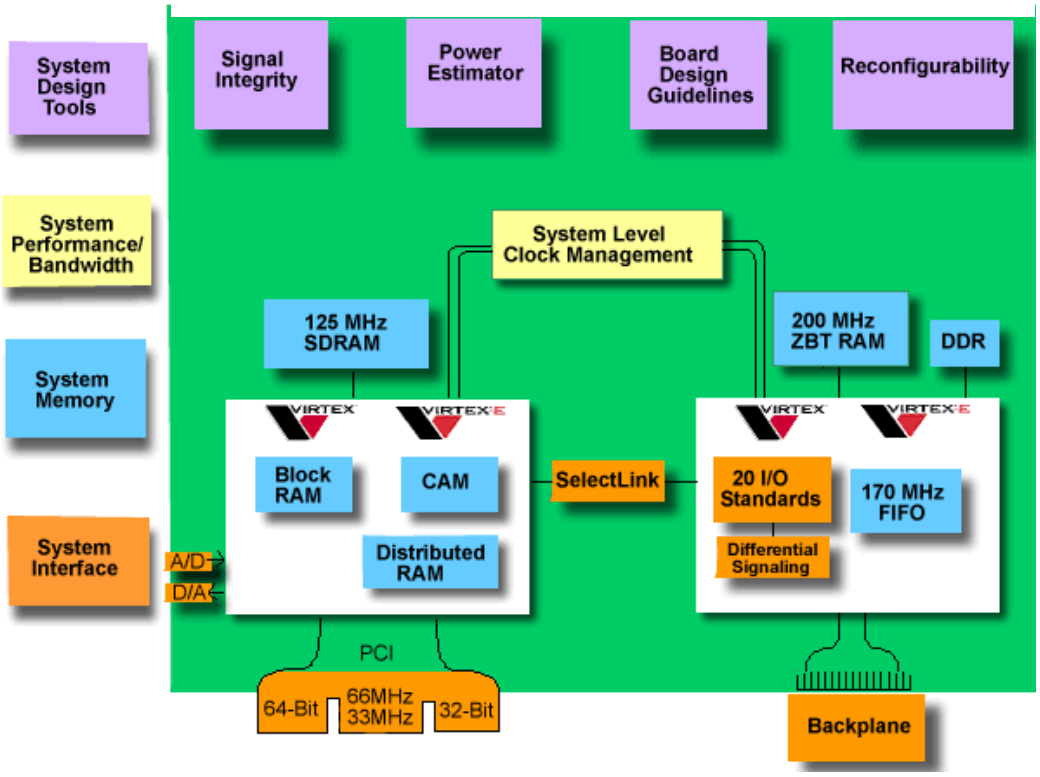


Figure 2.3: Functional Block Diagram of Virtex-E FPGAs

Since it is fully supported by Xilinx(TM) Alliance Series(TM) and Foundation Series(TM) software, as well as all of the EDA tools from Xilinx Alliance partners, the Virtex series is a complete solution ready to meet the challenges of next generation designs. The available software also includes the built-in CORE Generator(TM) tool with a variety of web downloadable Smart-IP(TM) BaseBlox(TM) cores. The Virtex solution helps system designers quickly create very complex designs with guaranteed results.

2.2.2 Delay Locked Loop

As FPGAs grow in size, quality on-chip clock distribution becomes increasingly important. Clock skew and clock delay impact device performance and the task of managing clock skew and clock delay with conventional clock trees becomes more difficult in large devices. The Virtex-E series of devices resolve this potential problem by providing up to eight fully digital dedicated on-chip Delay-Locked Loop DLL) circuits which provide zero propagation delay and low clock skew between output clock signals distributed throughout the device. Each DLL can drive up to two global clock routing networks within the device. The global clock distribution network minimizes clock skews due to loading differences. By monitoring a sample of the DLL output clock, the DLL can compensate for the delay on the routing network, effectively eliminating the delay from the external input port to the individual clock loads within the device. In addition to providing zero delay with respect to a user source clock, the DLL can provide multiple phases of the source clock. The DLL can also act as a clock doubler or it can divide the user source clock by up to 16. Clock multiplication gives the designer a number of designs alternatives. For instance, a 50 MHz source clock doubled by the DLL can drive an FPGA design operating at 100 MHz. This technique can simplify board design because the clock path on the board no longer distributes such a high-speed signal. A multiplied clock also provides designers the option of time-domain-multiplexing, using one circuit twice per clock cycle, consuming less area than two copies of the same circuit. Two DLLs in can be connected in series to increase the effective clock multiplication factor to four. The DLL can also act as a clock mirror. By driving the DLL output off-

chip and then back in again, the DLL can be used to de-skew a board level clock between multiple devices.

2.3 Introduction to Digital Filters

A filter is a system of network that selectively changes the wave shape, phase-frequency and amplitude – frequency characteristics of a signal as desired by the design Engineer. Filtering has the following effect on the input signal:

- Improves the quality of signal,
- Reduces noise,
- Extracts information,
- Separates multiple signals to get efficient information out of them.

Filtering can be done in two ways:

- Analog filtering.
- Digital filtering.

A digital filter is a mathematical algorithm implemented in hardware and software to operate on a digital input signal to produce a digital output thus achieving the above-mentioned objectives. Digital filters more often work on the digitized version of analog signals obtained after sampling or on just numbers, stored in the computer memory. The interface circuit performing the conversion of a continuous time analog signal (CTS) into its digitized version is called *analog to digital converters (A/D)* and the processed digitized output to its analog version is done by *digital to analog converters (D/A).* As A to D conversion takes some time (depending on the type of ADC used) it is necessary to keep the analog signal at the input of the ADC constant in amplitude until the conversion is complete for better results (at least at high frequencies>100Hz). This is achieved by the sample and hold (S/H) circuit which samples the input CTS at regular interval and holds the analog sampled data constant at its output for sufficient time to permit accurate conversion. The different AD, DA, S/H circuit specifications used in the work is discussed later on.

A block diagrammatic representation of the real time digital filter is shown in figure

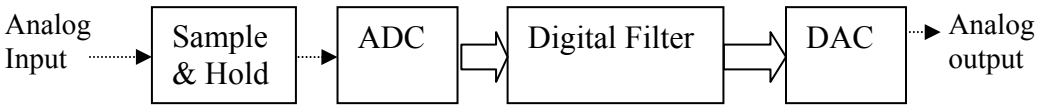


Figure 2.4: Block diagram of a digital filter

The S/H circuit consists of operational amplifiers, capacitors etc to provide better isolation and tracking of the input signal. The total time needed to switch from the hold mode to sample mode and to acquire the input with considerable accuracy is called *acquisition time* and it depends on RC time constant of the circuit. The converse of the above is known as *aperture time* .The A to D converter takes the output of the S/H circuit as its input. For signal processing applications the output of ADC is in binary code. The output is a sequence of words with each word representing a sample of sequence is a collection of bits, which limits the dynamic range, and accuracy of the converter.

The Sample and Hold circuit along with the Analog to Digital converter or the ADC produces a discrete time signal as most of the digital filters work on DTS. A discrete-time signal is a sequence of values that correspond to particular instants in time. The time instants at which the signal is defined are called the signal's *sample times*; traditionally, a discrete-time signal is considered to be undefined at points in time between these instants. For a periodically sampled signal, the equal interval between any pair of sample times is the signal's *sample period*, Ts. The *sample rate*, Fs, is the reciprocal of the sample period, or 1/Ts.The figure 2.5 will give an example of DTS.

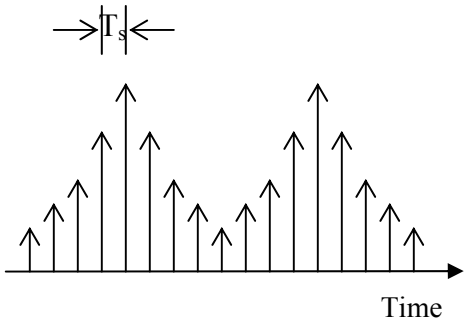


Figure 2.5: Example of DTS

Digital filters are now an integral part of Digital Signal Processing (DSP) where they are preferred compared to Analog filters in number of applications. The table below tabulates the different advantages and disadvantages of digital filters compared to their analog counterparts.

Advantages	Disadvantages
Truly linear phase response and performance repeatable from unit to unit.	Lower speed of response due to speed constraint of ADC and DAC.
Inert to environmental changes and can operate at very low frequencies.	The maximum bandwidth is in real time much lower than Analog filters.
Can work over wide range of frequencies just by changing the sampling frequency.	They are subjected to ADC noise, Quantization error, round off noise whose cumulative effect can make the filter unstable.
Single hardware can operate over a number of channels.	The hardware design and development is more time consuming.
They are more precise as their precision is limited by word length used.	

Digital filters are broadly divided into two classes:

- Finite impulse Response (FIR) filters.
- Infinite Impulse response (IIR) filters.

Both the type of filters can be represented by the formula,

$$y(n) = \sum_{k=0}^{N-1} h(k)x(n-k)$$

Where h(k) is the impulse response sequence (k=0, 1, 2, 3, ...) of the filters. In FIR filters h(k) is finite, i.e.

$h(k) = 0$ for $k > N_2$ and $k < N_1$ where $N_1 < N_2$.

In an IIR filter, however, *h(k)* is infinite.

The z-transform realization of such a filter is given by the transfer function H(z) and is represented in the z domain as

$$Y(z) = H(z)X(z) \text{ Where, } H(z) = \sum_{k=N_1}^{N_2} h[k]z^{-k}$$

and x[n] and y[n] are the input and output sequences and the filter is always BIBO stable over the whole frequency range of interest. This is because, all poles of the system lie on z = 0 in the z plane. Thus the region of convergence is the whole of the z plane except z=0.

As stated before, in IIR filters the impulse response is of infinite duration. Thus, the transfer function is generally expressed in a infinite series form.

$$H(z) = \frac{P(z)}{D(z)} = \frac{p_0 + p_1 z^{-1} + p_2 z^{-2} + \ldots\ldots\ldots p_N z^{-N}}{1 + d_1 z^{-1} + d_2 z^{-2} + \ldots\ldots\ldots\ldots\ldots d_N z^{-M}} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}}$$

$P(z) = \frac{W(z)}{X(z)}$ and $\frac{Y(z)}{X(z)} = \frac{1}{D(z)}$ where P(z) and $D(z)$ appears to be FIR filters separately.

In the time domain,

$$y(n) = \sum_{k=0}^{\infty} h(k)x(n-k) = \sum_{k=0}^{N} b_k x(n-k) - \sum_{k=1}^{M} a_k y(n-k) \quad \ldots\ldots\ldots\ldots\ldots\ldots\text{IIR}$$

An IIR filter of order N requires 2N+1 unique coefficients and requires 2N+1 multipliers and 2N adders for implementation. It can be seen from the above equation that the output y(n) depends on past outputs y(n-k) and past input samples x(n-k) which makes IIR filter some sort of feedback network.

IIR filters are not inherently stable like FIR filters. This is because they contain M poles; the region of convergence is exterior to the circle containing the pole farthest away from the origin. If the region of convergence includes the unit circle, the system is stable.

2.3.1 Filter Structures

The structure of a filter in its direct form from the equation mentioned above can be represented as,

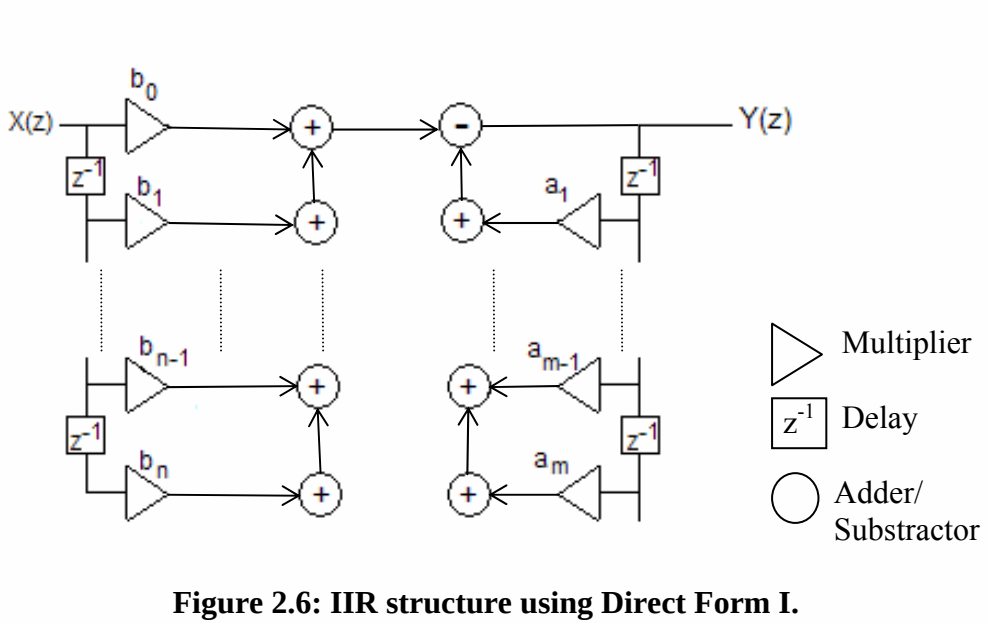


Figure 2.6: IIR structure using Direct Form I.

In FIR structure, the feedback loop block of the diagram is not present.

Another form, called the Direct form II reduces the delay elements by half. Filters can also be made with cascade and parallel combinations of filters of smaller, usually second, order. These implementations break the H(z) polynomial into forms like –

$$H(z) = p_0 \prod_k \left(\frac{1 + \beta_{1k} z^{-1} + \beta_{2k} z^{-2}}{1 + \alpha_{1k} z^{-1} + \alpha_{2k} z^{-2}} \right) \quad \text{.....Cascade Structure}$$

Or

$$H(z) = p_0 + \sum_k \left(\frac{\beta_{0k} + \beta_{1k} z^{-1}}{1 + \alpha_{1k} z^{-1} + \alpha_{2k} z^{-2}} \right) \text{……..Parallel Structure}$$

2.3.2 Comparison between FIR and IIR filters.

FIR	IIR
Exactly linear phase response with no phase distortion even when causal.	Non-linear phase response at the band edges.
Non-recursive realization makes them stable.	Stability not always guaranteed.
Effect of round off and coefficient quantization errors are less prominent.	Drastically changes the performance of the filter.
Transients have a finite duration.	Transients have infinite duration.
Requires more coefficients for sharp cutoff than IIR.	Analog filters can be readily transformed into digital IIR filters.

2.3.3 Filter selection criteria

- Sharp cutoff, high throughput - IIR with elliptic properties.
- Full phase linearity - FIR.

2.3.4 The Sampling Theorem and Nyquist Rate

The available frequency band for analog filters extends from zero to infinity, but for digital filters it varies from zero to the Nyquist frequency as governed by the sampling theorem which states that *the signal should be sampled at the rate of atleast* $2f_{max}$, *where* f_{max} *is the highest frequency component in a signal* to get proper out put.

$$F_s \geq 2f_{max}$$

Where, F_s is the sampling frequency or rate. The quantity $2f_{max}$ is also called Nyquist rate and the over sampling ratio is defined as:

Over sampling ratio= $\frac{F_s}{2 f_{\max}}$

In filter design approach the over sampling is more preferred.

2.3.5 Digital Filter Design

To design a digital filter, we must first obtain the specifications of the filter we have to design. The specifications of digital filter are based on the magnitude or phase response of the system to be designed. The phase criteria are usually corrected after meeting the magnitude specifications by cascading it with all-pass filter, called a phase equalizer. The magnitude specifications are de

An ideal filter has a "brick walled" response – it passes certain frequencies and do not pass others. It is not possible to realize this magnitude response in practice, because the impulse response of these filters are non-causal and have doubly infinite length. One of the ways to obtain an approximate roll-off is to truncate the coefficients. This results in a non-ideal response, which contains ripples in the passband and stopband. Also the sharp response is replaced by a gradual roll-off. Filter design involves the design of filters, which will conform, to certain tolerance levels, which are specified.

Filter specifications: Specification of a filter includes:

- **Order of the filter (N)** – This usually depends on the other specifications, but is very important, increasing order leads to an increasing computational complexity and hence additional burden on the resources.
- **The sampling frequency (F_s).** This must be such that the digital system is capable of handling the bandwidth of the signal, i.e. the maximum frequency of the signal must be less than the Nyquist frequency
- **Pass band edge frequency (f_p).** The value upto which frequencies can pass unattenuted through the filter
- **Stop band edge frequency (f_s).** The value after which frequencies are not allowed to pass at all.
- **Pass band ripple (Δ_p).** The maximum deviation allowed in the pass band.
- **Stop band ripple (Δ_s).** The maximum deviation allowed in the stop band.

The specifications are visually described using the figure 2.7, which shows the specifications for a low pass filter.

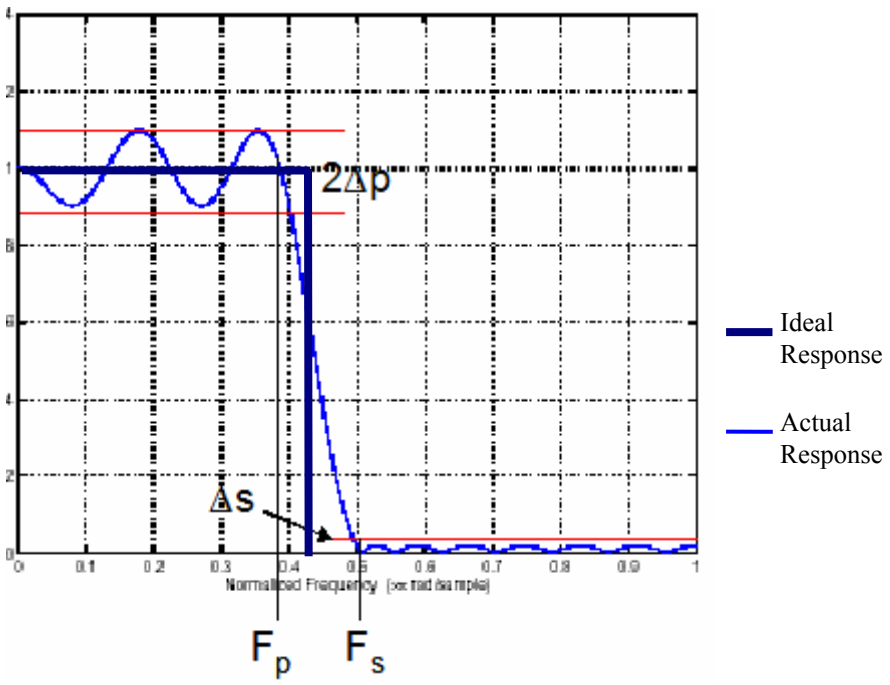


Figure 2.7: Specifications of digital filter

In this case, the pass band is from 0 to F_p and the stop band from F_s to the 1. The band of frequencies from F_p to F_s is known as the transition band and controls the sharpness of fall(or rise) of the magnitude response. In the digital domain, frequencies are normally measured in normalized frequencies. The relation between normalized frequency and frequency in hertz is given by,

$$\omega = \frac{2\pi f}{f_s}$$ Where f_s is the sampling rate.

In the pass band, we require that the magnitude should approximate unity with a maximum error of $\pm\Delta_p$. Mathematically,

$$1-\Delta_p \le |G(j\omega)| \le 1+\Delta_p \text{ for } |\omega| \le F_p$$

$$\text{and, } |G(j\omega)| \le \Delta_s \text{ for } F_s \le |\omega| \le 1$$

2.3.6 FIR filter design

Given the specifications of the filter, the most important choice is the order of the FIR filter. The higher the order, the better the filter, but the computational time increases. To get an sharp cut-off with transition band less than 100Hz and ripples less than .1, the required orders are more than hundred. The difference that order makes on the response is evident from figure 2.8.

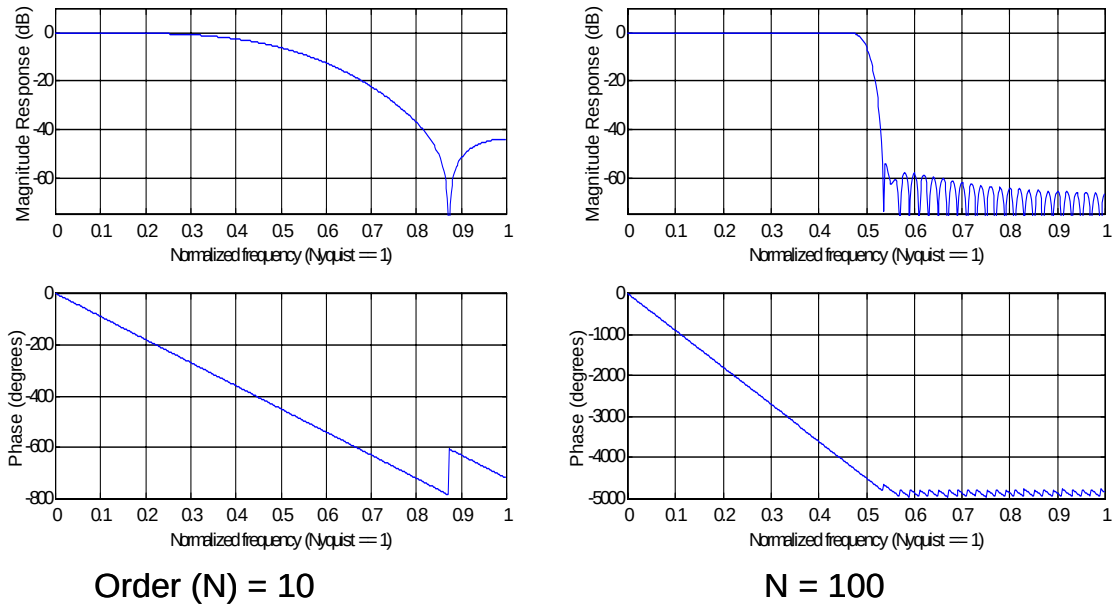


Figure 2.8: FIR filters with cutoff at ω=0.5

To estimate the order, a simple formula has been provide by Kaiser, given by,

$$N \cong \frac{-20\log_{10}\left(\sqrt{\partial_p \partial_s}\right) - 13}{14.6(\omega_s - \omega_p)/2\pi}$$

This gives an approximation to the filter order for moderate passband filters. Order is seen to be inversely proportional to the transition width (ω_s - ω_p) and also increases on decrease of the allowable ripples.

In MATLAB, two functions – *remezord* and *kaiseord* are provided which provide us with the approximate order of the filter on specification of the ripples and the edge frequencies. Remezord is Parks-McClellan optimal FIR filter order estimation and is given as: [N, Fo, Ao, W] = remezord (F, A, DEV, Fs)

Where normalized frequency band edges are given as Fo, frequency band magnitudes are given by Ao and weights W.

2.3.6.1 Windowing method of filter design

To design a FIR filter, we start with the ideal filter response -

$$H(e^{j\omega}) = \begin{cases} 1, & |\omega| < \omega_c \\ 0, & \omega_c < |\omega| < \pi \end{cases}$$

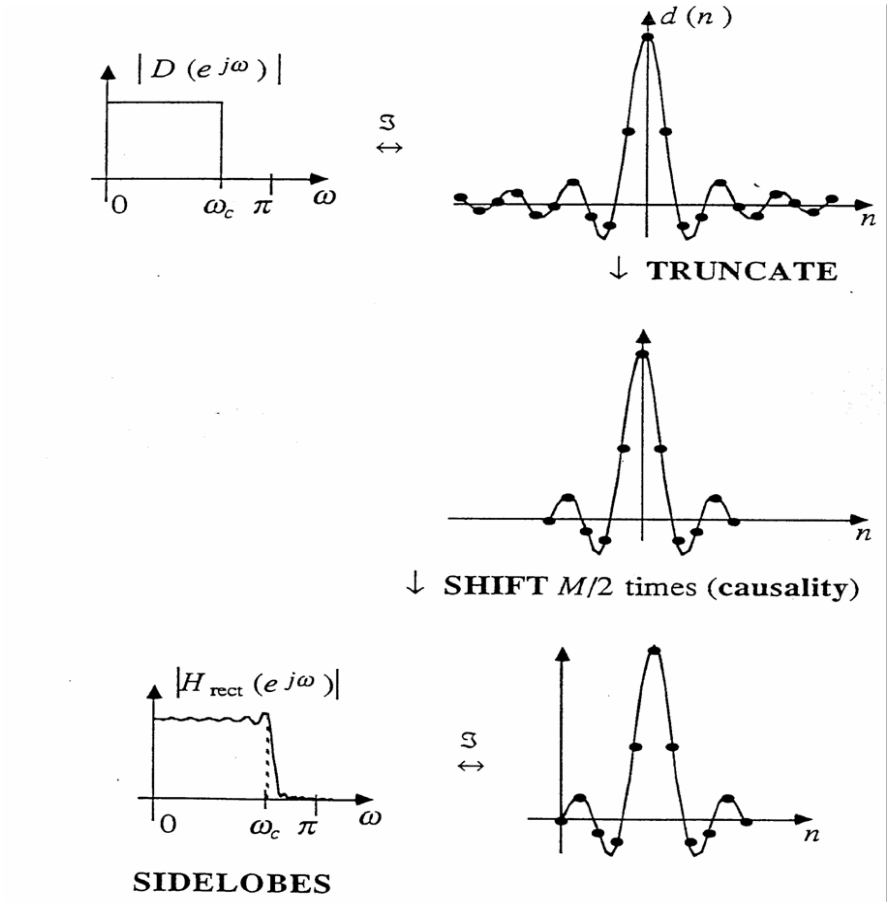


Figure 2.9: Effect of rectangular window

Low Pass is taken as an example and the following discussion is true for any type of filter. The impulse response of this filter is doubly infinite and given by

$$h[n] = \frac{\sin \omega_c n}{\pi n}, -\infty < n < \infty$$

This is impossible to take, as we need to store a finite amount of coefficients. Thus, we truncate the coefficients using a window function. The simplest window function is a rectangular window, given by

$$w_R[n] = \begin{cases} 1, & 0 \le |n| \le M \\ 0, & otherwise \end{cases}$$

This has the effect of leaving out some of the values of the series. The rectangular window has sharp edges and gives rise to Gibbs phenomenon, i.e. an appreciable ripple is introduced into the output as seen in the figure 2.9.

To remedy this, we can use the Bartlett or triangular window, which does not have any edges.

$$w_T[n] = \begin{cases} 2n/M & 0 \le n < M/2 \\ 2 - 2n/M & M/2 < n \le M \\ 0 & otherwise \end{cases}$$

This results in no ripple. However, the transition band is widened appreciably. Many trade-offs between these two are available.

Hanning window - $$w_n = \frac{1}{2}\left[1 + \cos\left(\frac{2\pi n}{2M+1}\right)\right]$$

Hamming window - $$w_n = 0.54 + 0.46\cos\left(\frac{2\pi n}{2M+1}\right)$$

Blackman window - $$w_n = 0.42 + 0.5\cos\left(\frac{2\pi n}{2M+1}\right) + 0.08\cos\left(\frac{4\pi n}{2M+1}\right)$$

Hamming window has been used in this case.

In MATLAB the filter is be designed by fir1 function with the command B = FIR1 (N, Wn) which designs an N'th order low pass FIR digital filter and returns the filter coefficients in length N+1 vector B. The cut-off frequency Wn must be between 0 < Wn

< 1.0, with 1.0 corresponding to half the sample rate. The filter B is real and has linear phase, i.e., even symmetric coefficients obeying B(k) = B(N+2-k), k = 1,2,...,N+1.

B = FIR1 (N, Wn,'high') designs a highpass filter.

B = FIR1 (N, Wn,'stop') is a bandstop filter if Wn = [W1 W2].

B = FIR1 (N, Wn) is a bandpass filter if Wn = [W1 W2].

Where in both the above cases passband is W1 < W < W2.

By default FIR1 uses a Hamming window. Other available windows, including Boxcar, Hanning, Bartlett, Blackman, Kaiser and Chebwin.

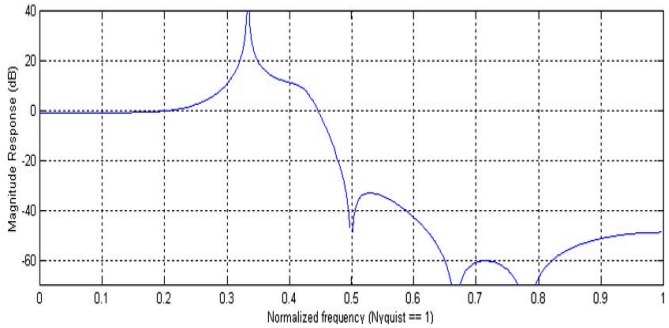
2.3.7 IIR filter design

IIR filters are usually designed by using the Analog filter design. There are broadly two ways to design an IIR filter, IIT and BLT.

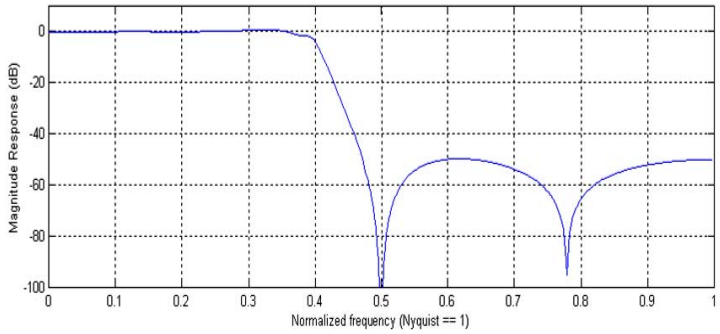
2.3.8 Practical implementation concerns

The functions realized in practice by MATLAB give coefficients to a very high precision but as actual implementation is done in digital domain, a constraint of limited number of bits degrades the performance of the filters and to some extent makes them unstable. The sources are discussed below:

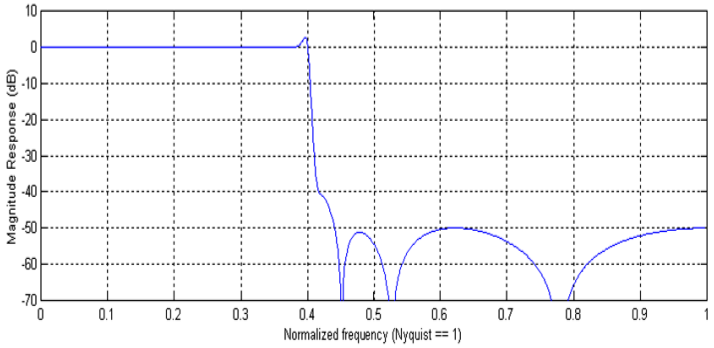
- ***Overflow:*** ADC, DAC and filter design constraints the number of bits in a wordlength. But if due to addition permissible word length is exceeded wrong output samples are generated which leads to instability in IIR filters.
- ***ADC quantization:*** The analog to digital conversion process quantizes each signal into $2^{\text{bits of ADC}}$ and hence this introduces an error seen as ADC noise.
- ***Coefficient quantization:*** The coefficients obtained from Matlab are correct up to many places of decimals, but to implement the digital filter, the coefficients must be represented by a fixed number of bits determined by the software inbuilt filter wordlength. This introduces considerable amount of noise with large transients, changed frequency response, sharp transition widths and brings the poles close to the unit circle. This effect is more severe in IIR filters than in FIR filters and might push them towards instability. It also changes the stop band and pass band attenuation.



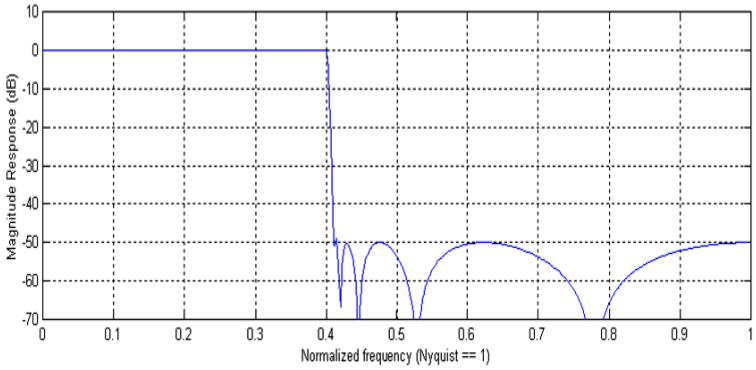
Scaling factor = 2^5 - Spikes present



Scaling factor = 2^10 - Slow Roll-off



Scaling factor = 2^15 - Small amount of pass ripple



Scaling factor = 2^20 - Ideal

Figure 2.10: Illustration of Truncation effect

- **Coefficient Scaling and truncation effect:** The coefficient quantization problem as mentioned above leads to a very important practical consideration in filter design, i.e. coefficient scaling. The FPGA is fixed point device, which implies it cannot accept floating-point numbers. Essentially, we can only provide the coefficients as integers, and we have to remember the decimal point.

Most of the coefficients in the design are around 1 or much smaller. This implies we have to multiply them by a certain constant to represent them as a constant. Apparently the easiest solution is to shift the decimal point by an adequate number of places – which translates as a multiply by 10. However, while providing output the FPGA has to scale down the value by 10. Moreover, many scale-ups and scale-downs may be necessary inside the FPGA. All these would require Multipliers and Dividers. Thus, we scale up and down by factors of 2. This means, scaling up can be done by the FPGA by placing adequate number of zeros after the number. Also, scaling down would involve simply neglecting the required number of LSBs. Thus, a great deal of hardware is saved.

As already mentioned, it is impossible to take numbers with the precision of MATLAB. Thus, after performing the scale-up operation, we have to truncate the trailing decimal values. This results in highly distorted response if the scaling factor is small. The effect of truncation is shown by figure 2.10. The response of an IIR elliptic filter is taken as the example and the truncation effects are simulated on MATLAB.

2.3.9 Application of digital filters:

1. **Digital Audio Technology:**

- Digital filters find considerable use in digital graphic equalizers, CD players and digital audio system. IIR filters are very much in use in Audio frequency splitting of the whole range into bands.

2. **Instrumentation:**

- The digital filters are exclusively used in digital control system to built digital controllers.

- Frequency Generation: IIR filters with poles at the unit circle is unstable and this fact is exploited in designing sine wave frequency oscillators with considerably high accuracy.

3. **Telecommunication:**

- Digital Telephony: Digital communications use PCM data communication. IIR filters provide the necessary low pass, band pass filters at the transmitting and receiving end.

3 Digital Filter Design Using Discrete Time Convolution Equation

3.1 Cost vs. Speed

Traditionally, digital signal processing (DSP) algorithms are most commonly implemented using general-purpose (programmable) DSP chips for low rate applications, or special-purpose (fixed function) DSP chip-sets and application-specific integrated circuits (ASICs) for higher rates. But with the technological advancements of FPGA, they are now efficiently used in the design of custom DSP devices. The main advantage of FPGA is speed. The speed requirement is worth giving attention when the filter order increases. For higher order filters FPGA based designs are much faster compared to DSP chips. Also the FPGA advantage grows for multiple filter channels.

The fully parallel model leads to highest speed where the data rate matches the clock rate (which can be greater than 100MS/S in today's FPGAs). But in this case the number of DALUTs is maximum. In serial implementation, only one DALUT is needed. So hardware requirement is reduced. But the price to be paid for this is the reduction in speed. If the input data is B bits wide, the computation of output takes at least B clock cycles. So the data rate = (Clock rate of the FPGA)/ (No. of Bits in Input Data)

3.2 FIR Filter

A causal FIR filter of order N is characterized by a transfer function *H(z)*.

$$H(z) = \sum_{k=0}^{N} h[k] z^{-k} \qquad \text{---(1)}$$

which is a polynomial in z^{-1}. In the time domain, the input output relation of the above FIR filter is given by

$$y[n] = \sum_{k=0}^{N} h[k] x[n-k] \qquad \text{---(2)}$$

where y[n] and x[n] are the output and input sequences, respectively.

There are several realization methods for FIR filters. These are direct form realization, cascade form realization, polyphase realization etc. Direct form realization is discussed here.

3.2.1 Direct Form FIR filter

An FIR filter of order N is characterized by N+1 coefficients and, in general, requires N+1 multipliers and N two input adders for implementation. Structures in which the multiplier coefficients are precisely the coefficients of the transfer function are called **direct form** structures.

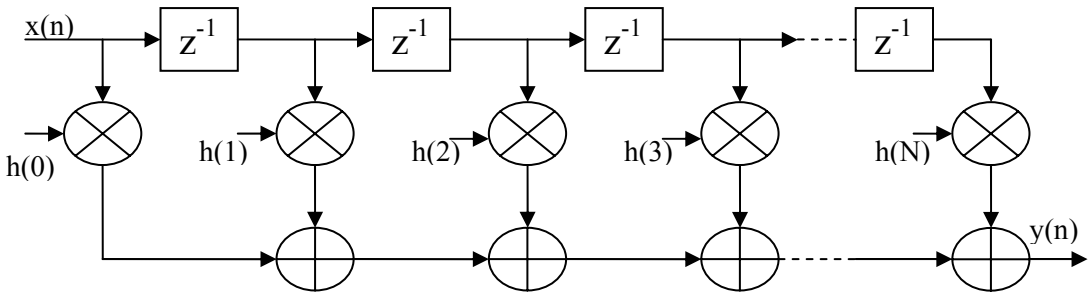


Figure 3.1: Direct form FIR structure

The structure of Figure 3.1 is also called a **tapped delay line** or **transversal filter**. Its transpose gives a second direct form structure. Both direct form structures are canonic with respect to delays.

Even though the figure is a useful conceptualization of the computation performed by the core, the actual FPGA realization is quite different. A discrete time convolution equation (DA) realization is employed. With this approach there are no explicit multipliers employed in the design, only lookup tables (LUTs), shift registers and a scaling accumulator.

3.2.2 Using Filter Symmetry

The impulse response of many filters posses significant symmetry. This symmetry can be exploited to minimize arithmetic requirements and produce area efficient filter realizations. The following figure 3.2 shows the impulse response of a 7 tap symmetric FIR filter.

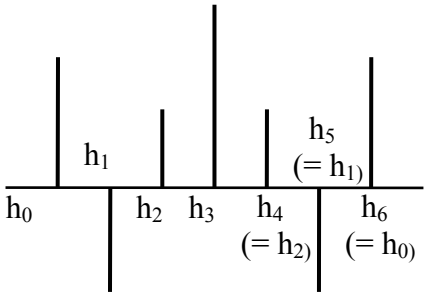


Figure 3.2: Symmetric FIR- odd number of terms

Instead of implementing this filter using the architecture shown in Figure 3.1, the more efficient signal flow-graph in Figure 3.2 can be used. In general the former approach requires N+1multiplications and N additions. In contrast, the architecture in Figure 3.2 requires only (N+1)/2 multiplications and approximately N additions. This significant reduction in the computation workload can be exploited to generate efficient filter hardware implementations.

Figure 3.3. shows the impulse response of an 8 tap negative symmetric or odd symmetric FIR filter.

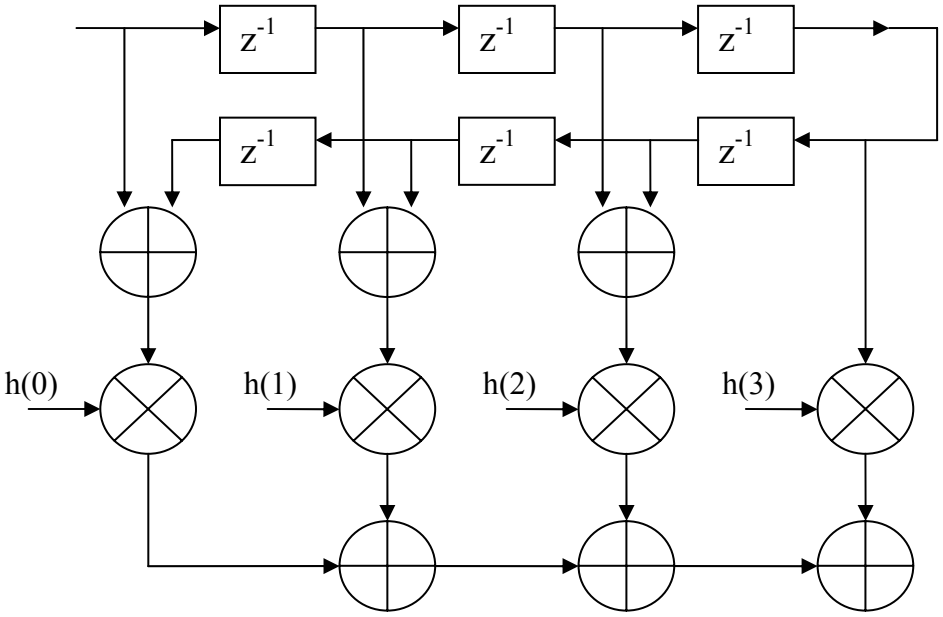


Figure 3.3: Exploiting coeff. symmetry- odd number of filter taps

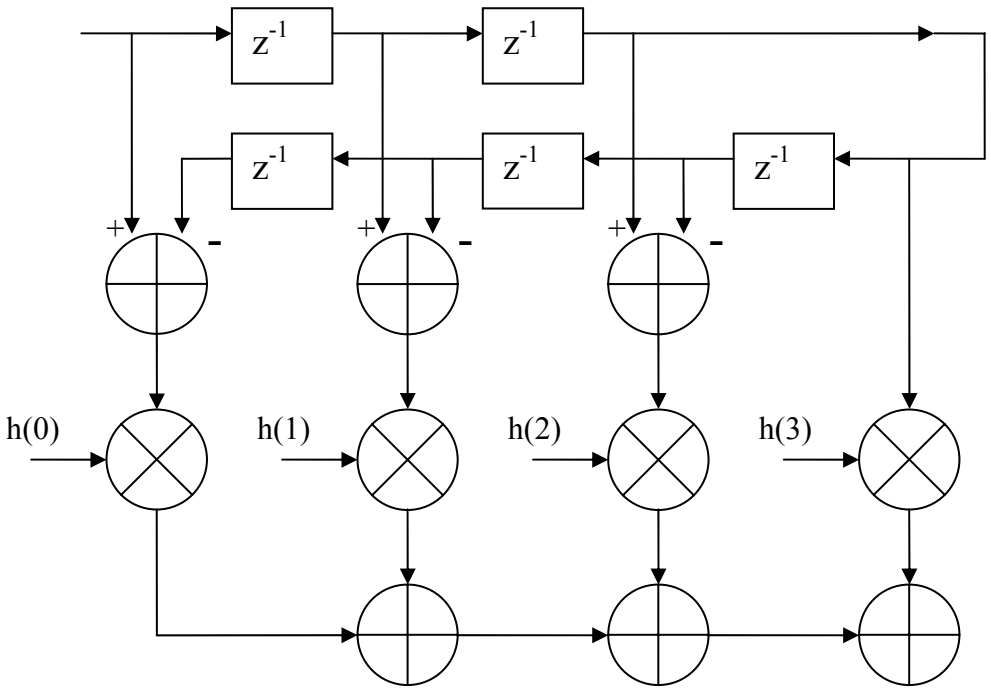


Figure 3.4: Exploiting coeff. Symmetry-even number of filter taps

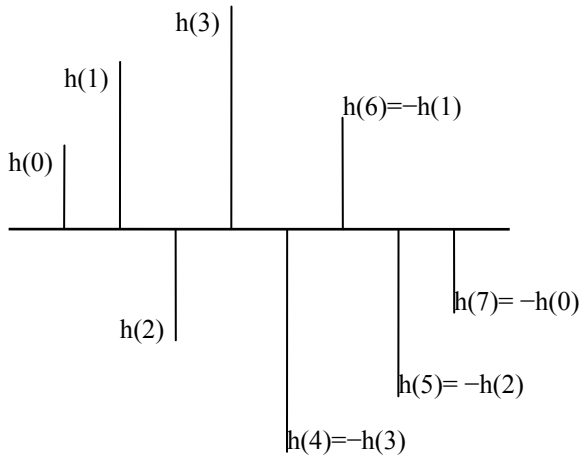


Figure 3.5: Negative Symmetric Impulse Response

This symmetry is easily exploited in a manner similar to that shown in Figure 3.2 and Figure 3.3. In this case the middle layer of adders is replaced by subtractors as illustrated in Figure 3.4.

The example considered here illustrates a filter with an even number of terms, the filter structure for an odd number of terms is a simple extension of the same principle.

The core generator filter module allows the filter symmetry to be specified. When the impulse response does exhibit symmetry, the filter logic requirements can be significantly reduced in comparison to an implementation that does not exploit the impulse response structure. For example a 100 tap non-symmetric filter with 12-bit data samples and 12-bit coefficients consumes 519 Virtex logic slices [Source: *Xilinx Product Guide,* Xilinx Inc. 1999]. In contrast, a 100 tap symmetric filter is realized with 354 slices. This represents approximately a 30% savings in area.

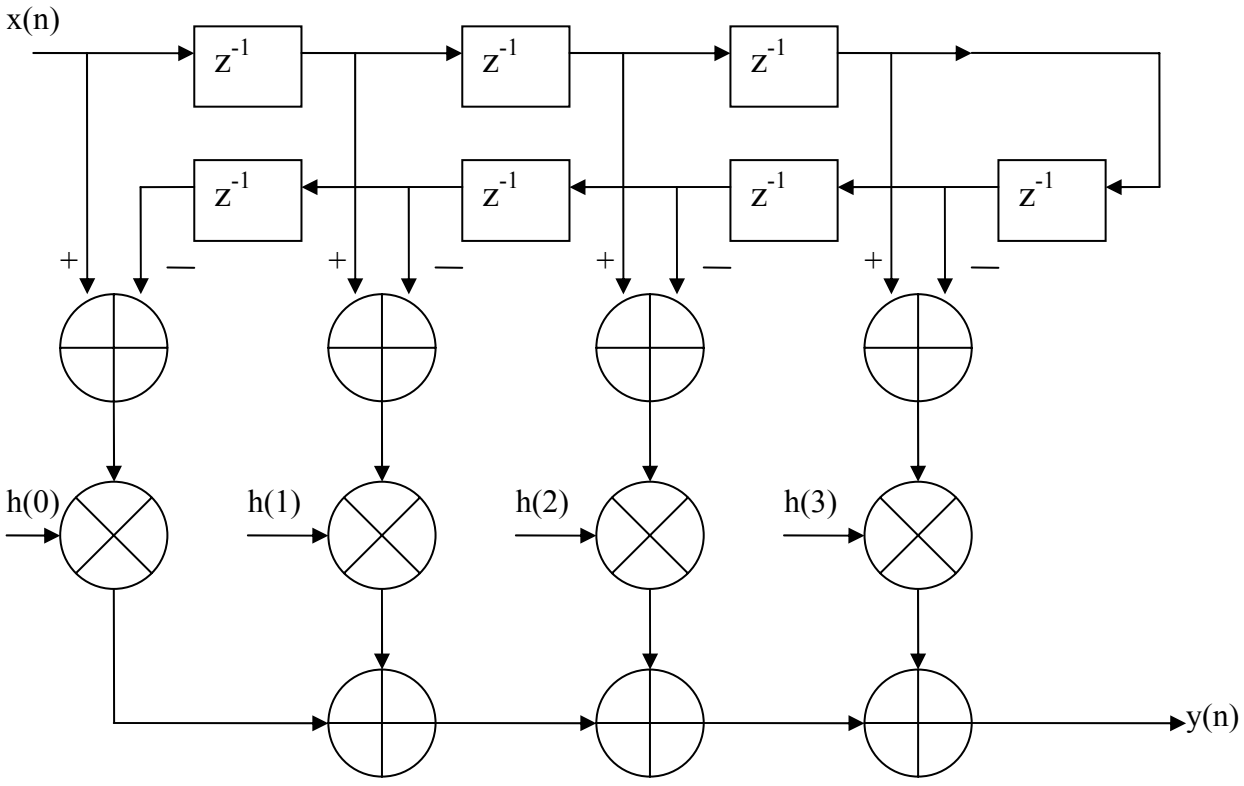


Figure 3.6: FIR Architecture- Exploiting Negative Symmetry

4 Hardware Architecture of Digital Filters

4.1 Introduction

The aim of the project is to design and implement various digital filters on Field Programmable Gate Array. This chapter has two subsections, Hardware architecture and Interfacing Hardware. In the Hardware Architecture section is Modular description of 4-Tap FIR Digital filter and functional units. In the interfacing circuits has been explained.

4.2 Hardware Architecture of 4-Tap FIR Digital Filter

In order to implement the top-level design specifications, we break down our design into simple modules, each of which is a well-defined functional unit. Our modular diagram is shown in Figure 4.1.

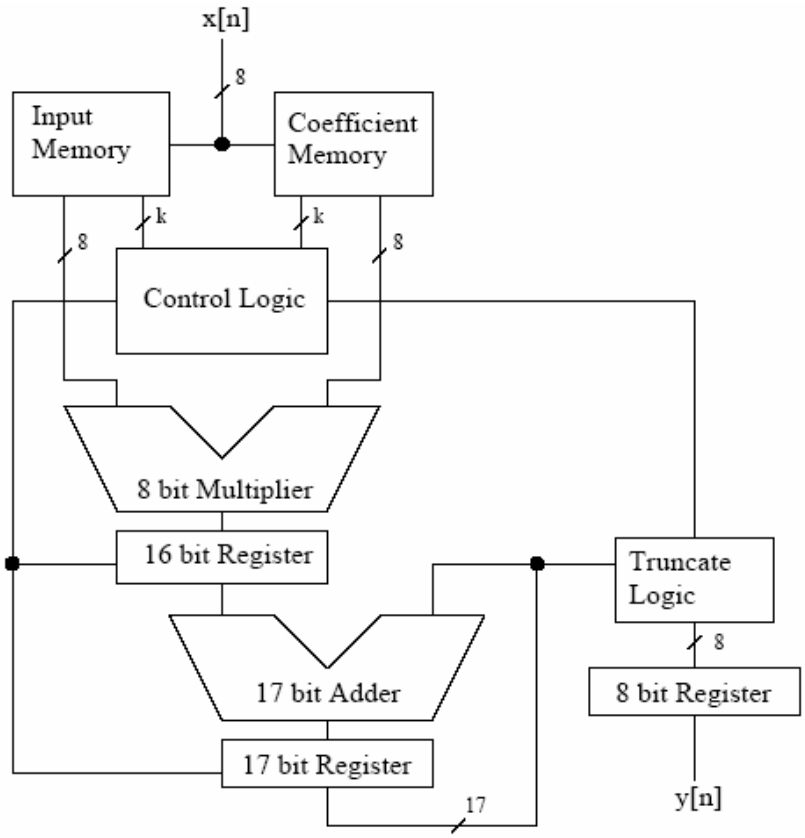


Figure 4.1: Modular Architecture of a 4-Tap FIR Digital Filter

Now we can Estimate the number of transistors necessary in each module.

- Input Memory: 4 8-bit registers = 32 D Flip-Flops 576 devices
- Coefficient Memory: 4 8-bit registers = 32 D Flip-Flops 576 devices
- Control Logic: Mod-4 Counter + Decoder + 20 Logic Gates 400 devices
- 8-bit Multiplier: 56 1-bit Adders + 64 AND gates 1,768 devices
- 16-bit Register: 16*(D flip-flop + 2 transmission gates) 330 devices
- 17-bit Adder: 17 1-bit Adders 510 devices
- 17-bit Register: 17*(D flip-flop + 2 transmission gates) 350 devices
- Truncation Logic: 3 Transmit ion Gates + 10-input OR + 10-input NAND

  70 devices

This list yields an estimate of approximately 4,600 devices total.

The input memory holds the four latest input coefficients x[n] for purposes of multiplication. The coefficient memory holds the four impulse response coefficients h[n]. The control logic functions as the driver for the filtering process: it contains a counter, a decoder, and appropriate logic to direct traffic over the buses, through the adder and multiplier, and into appropriate registers. The 8-bit multiplier is of the array type, and is made up primarily of 1-bit adders and AND gates. Since the output of the multiplier is a 16-bit quantity, we need a 16-bit register to store the intermediate result. The largest 8-bit quantities we can multiply are 0x7f. The largest product of these two multiplicands is 0x3f01. In the worst case, we would add 4 of these products together for our accumulator result, yielding a worst-case answer of 0xfc04, which is a 16-bit unsigned number. Adding a sign bit for two's complement numbers yields a minimum result-storage mechanism of 17 bits. Therefore, we have a 17-bit ripple-carry adder linked to a 17-bit accumulator register. To input the 16-bit multiplier register into the 17-bit adder, we simply sign-extend the 16-bit quantity. Finally, we need to convert our signed 2's complement 17-bit quantity into a signed 2's complement 8-bit quantity. The logic for this operation is contained in the “Truncation Logic" block, which will essentially test for two cases, numbers higher than 0x7f and lower than 0x80.

4.2.1Coefficent and Input Memory Units

The first functional blocks to be implemented are the two memory blocks, which store the four previous input values and the four coefficients for computation purposes. Each memory is essentially a collection of four 8-bit registers with appropriate input enable and output select lines to control the input and output of the 8-bit values into and out of the registers.

A block-level diagram of the input memory is shown in Figure 4.2 The input memory is designed to get a new input value at each computational clock cycle and to shift each previous input down one level, discarding the oldest inputs as necessary. As shown, the output to the multiplier is selected by the two select lines c0 and c1.

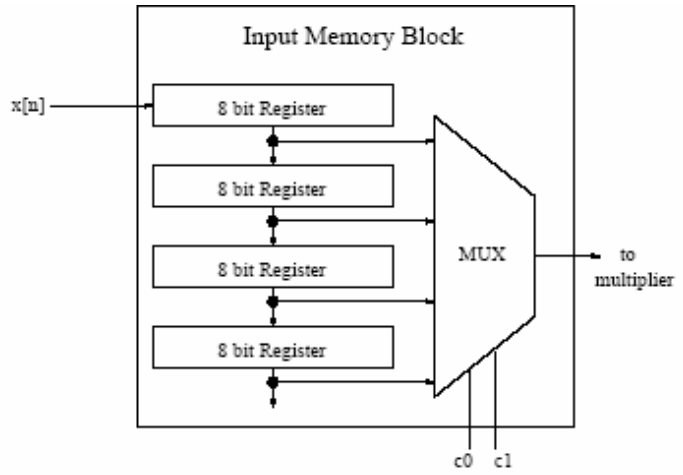


Figure 4.2: Input Memory Block Diagram

The coefficient memory, on the other hand, expects input values only when the COEFIN select pin is HIGH on the chip. A block diagram of the coefficient memory is shown in Figure 4.3. Since, unlike in the input memory case, any arbitrary coefficient can be input directly without changing the other coefficients, a second bus select line (depicted in the figure by the DEMUX) is necessary for the inputs, controlled by the input select lines c0 and c1.

The basic building block of the memory units is the register. In each 8-bit register, there are 8 D flip-flops, 10 transmission gates, 2 inverters, and an AND gate for a total of 174

devices. Our registers have both output and input enables, which leaves them quite flexible for use throughout our project for things other than the memory units.

The Coefficient Memory has 800 devices total, and the Input Memory has 840 devices total. The input memory is larger because of the more complicated internal register transfer capability.

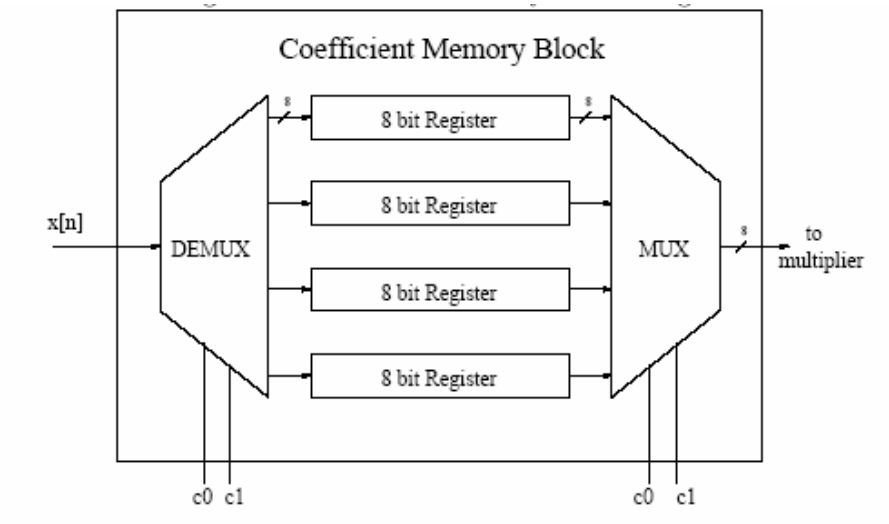


Figure 4.3: Coefficient Memory Block Diagram

4.2.2 Control Logic

The control logic essentially implements the algorithm of the FIR filter. The basic functionality of the control logic involves the maintenance of two counters, one mod-8 and one mod-4, counting the clock pulses coming in to the chip. One “clock cycle" occurs when the mod-4 counter has cycled once, since it is the “tap counter." Since we have four taps, each increment of the mod-4 counter corresponds to a tap of our FIR filters. The mod-8 counter cycles once for each tap, for a total of 24 clock pulses per clock cycle.

A block diagram of the controller is shown in Figure 4.4. The output of the mod-8 counter is fed through a 3-8 decoder, and the resulting lines function as “phase switches" for the clock cycle.

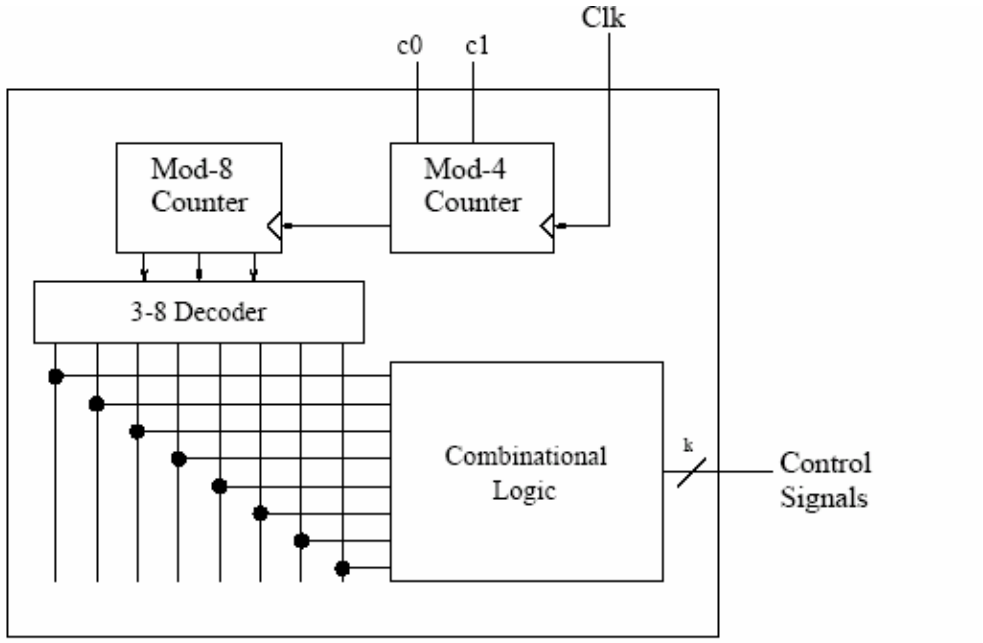


Figure 4.4: Controller Block Diagram

Clock Phase	Output Signals for Control Logic
T0	If (CoefIn) then CoefInEna and CounterReset else, InputInEna and if c1c0==00 then AddRegClear
T1	Multiplier Propagation Delay
T2	Multiplier Propagation Delay
T3	Multiplier Propagation Delay
T4	MultRegInEna
T5	AddRegInEna (Adder Propagation Delay)
T6	If c1c0==11 then OutRegInEna
T7	TapCounterIncrement

Table 1 enumerates the eight-phase algorithm iterated once for each filter tap.

4.2.3 Array Multiplier

The largest single block of our filter is the array multiplier. We decided upon an array multiplier design because of the sheer amount of multiplication present in the operation of a finite impulse response filter (four multiplications per output). Our 8-bit array multiplier takes two 8-bit signed two's complement numbers and produces a 16-bit answer. The actual array of 1-bit adders actually cannot accommodate signed two's Complement numbers, a “feature" we originally discovered during our simulation! As a result, we added appropriate logic before and after the multiplier to accommodate signed arithmetic. This logic analyzes the sign of the multiplier and multiplicand to determine the expected sign of the result, and takes the two's complement of each value as necessary to ensure the proper form of output. The actual multiplier core contains 56 1-bit adders and 64 AND gates but has a propagation delay of only an 8-bit ripple-carry adder! The additional sign logic adds two more adder propagation delays because each Conversion to or from two's complement form requires an addition. The array multiplier core consists of 2176 total transistors. With the additional sign logic included inside our implementation, our multiplier block contains 3424 transistors total.

4.2.4 17-Bit Adder (Critical path)

The 17-bit adder is comprised of 17 1-bit adders. Because the multiplier is given three clock phases for its propagation delay, the 17-bit adder becomes the speed bottleneck for the speed at which our clock may be run. For a conservative estimate of clock rate, we quadruple this propagation delay (allowing a 4.5 ns level hold time) for our clock period. This 6ns clock corresponds to a 166 MHz clock rate! Theoretically, then, our FIR filter should be able to run at approximately 166 MHz, neglecting heat dissipation and load capacitance effects. Because there are 32 clock rises per output, we calculate an output Frequency of roughly 5.2 MHz, or 5.2 million outputs per second. This sampling frequency is more than enough to handle most 8-bit audio and video signal processing demands.

4.2.5 Truncate Logic

The last step taken before the value is sent to the output register is the truncation of the 17-bit Accumulator register value into an 8-bit two's complement quantity. To accomplish this feat, we examine bits b7 through b15 and the sign bit b16. If the sign bit indicates a negative answer, then if there are any zeroes present in b7 to b15 we have negative overflow. On the other hand, if the sign bit b16 indicates a positive answer, then any ones present in b7 to b15 would signify positive overflow. Any other case represents a normal value within our 8-bit range, so we need do nothing. The flowchart is shown in Figure 4.5. Our logic implementation uses transmission gates to select the appropriate output depending upon the status of overflow. For positive overflow, we set the output to (0b01111111), which is the largest 8-bit positive quantity we can represent. For negative overflow, we set the output to (0b10000000), which is the most negative quantity possible for signed two's complement 8-bit numbers. In the case of no overflow, we pass the first eight bits of our answer to the output register unchanged.

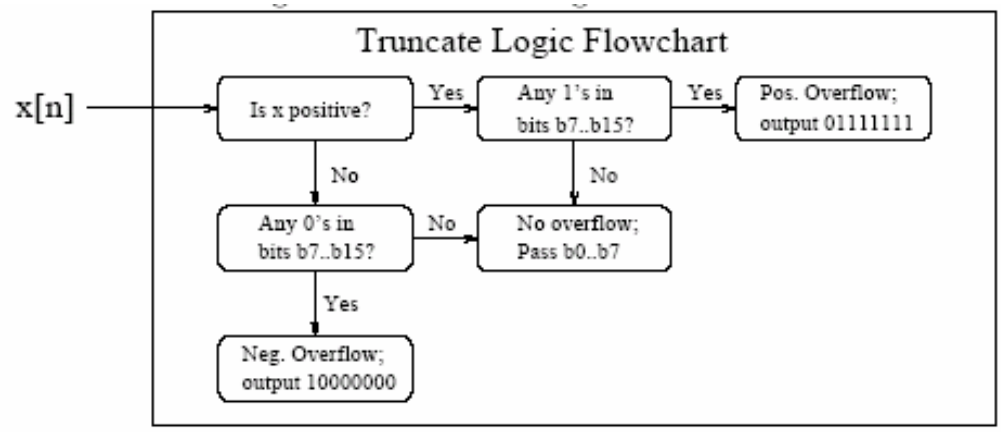


Figure 4.5: Truncate Logic Flow Diagram

4.3 Interfacing Methodology

The basic block diagram used is shown below. An analog signal is input into the system. This analog signal may be obtained from any source, but here a function generator has been used as a signal source.

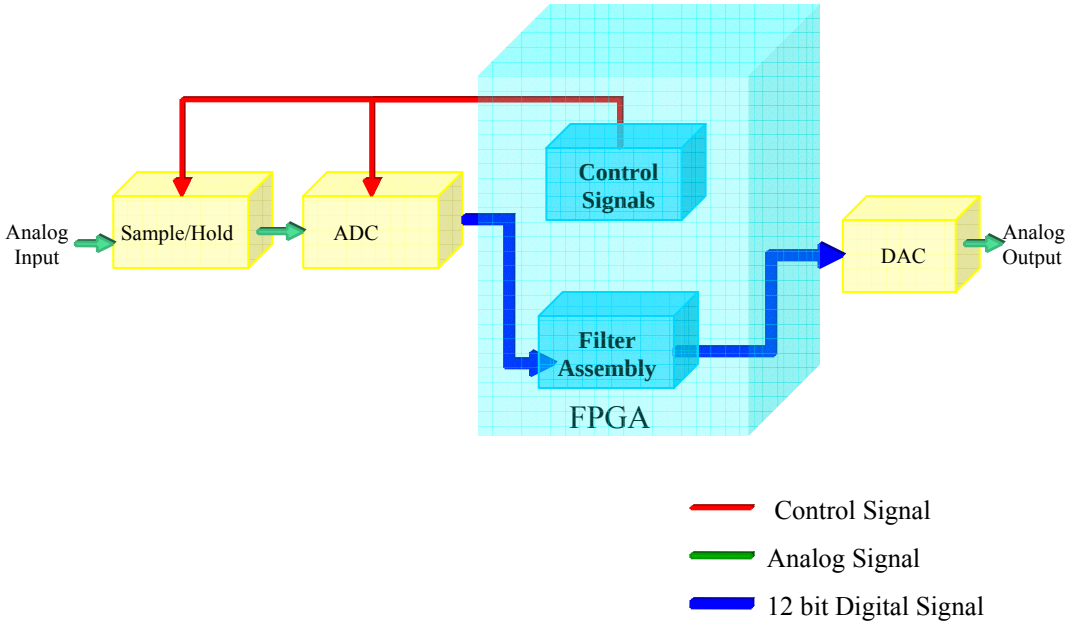


Figure 4.6

The analog input is fed into a sample and hold circuit, so that it can be digitized by the Analog to Digital Converter (ADC). Both the Sample and hold and ADC require control signals for their operation. These are generated by the FPGA itself.

Thus the function of the FPGA is two-fold. It is used to generate control signals for the peripherals as well as to condition the input digital signal by the use of filters. The "filter assembly" shown in the figure has been implemented in various ways. FIR filters and IIR filters have been implemented by using discrete time convolution equation method.

The output of the FPGA, which is the processed signal, is then passed through a Digital to Analog converter (DAC) and the required analog output is obtained.

Any real world signal is an analog one. To use a digital system to process an analog signal, we need a Analog to Digital converter (ADC). The output of a digital system is also usually a real-world analog signal. To obtain this signal, a Digital to Analog converter (DAC) is required. Most ADCs require a Sample and Hold (S/H) circuit to sample the signal before conversion.

On the output side, the DAC receives new data in every sampling period. Thus the output of the DAC is a staircase output. The required analog output signal is obtained from the DAC by using an analog low pass filter, which gets rid of the unwanted high frequency components. This filter is known as the reconstruction or a smoothing filter.

It can be shown, that if the sampling frequency is f_s, the maximum bandwidth of the digital system is $f_s/2$, ie, only frequencies upto $f_s/2$ are interpreted correctly by the system. It can be shown, that if a discrete sequence has a frequency of

$$\left(\frac{f_s}{2}+k\right) \text{ Where } 0 < k < \frac{f_s}{2},$$

the sequence will be exactly like another sequence with frequency $\left(\frac{f_s}{2}-k\right)$.

This phenomenon is called folding. Similarly, it can also be shown, that a discrete signal with a frequency of f, ($0 < f < f_s$), is same as a signal with frequency $f + kf_s$, (k is an integer). This phenomenon is known as aliasing.

While designing a digital system, we must choose the sampling frequency such that it is twice that of the highest relevant frequency of the signal. If in the analog signal, there are high frequency components, which are greater than $\frac{f_s}{2}$, it would appear to be a lower frequency signal and would create a distortion in the relevant signal information. This is prevented by using an anti-aliasing filter, which is an analog lowpass filter, which eliminates the higher frequency components of the input signal, and aliasing is reduced.

4.3.1 Sample and Hold (S/H) Circuit

If an analog signal is connected directly to the input of most ADCs, like the successive approximations (SA) type ADC, the conversion process can be adversely affected if the analog signal changes during the conversion time. Thus a S/H circuit is needed which procures a value of the analog signal at a time instant (called sampling) and retains that value for the time required by the ADC to convert it in the digital form (called holding).

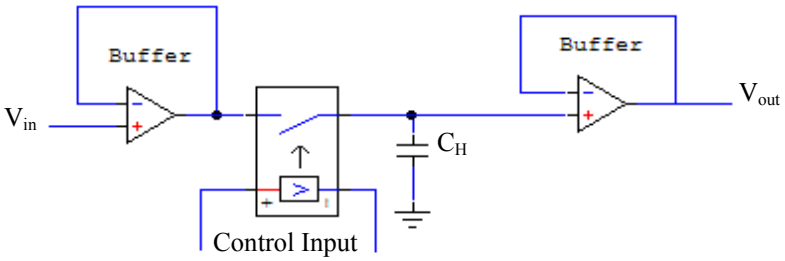


Figure 4.7: Sample and Hold circuit.

The IC used for S/H operation in the implementation is LF398N. It utilizes high-voltage, ion-implant JFET technology to obtain high DC accuracy with fast acquisition time low droop rate. The datasheet of the IC is given in appendix.

4.2.1.1 Design Considerations

The main consideration in design of the circuit is the choice of the hold capacitor C_h. The main tradeoffs are the acquisition time, hold step and droop rate. If the value of the capacitance is low, the acquisition time is lower, i.e. the sampling is faster. However, the hold step will increase due to the stray capacitive coupling between the input logic signals and the hold capacitor. The magnitude of hold step is inversely proportional to the hold capacitor value.

The logic signals are generated form the XILINX FPGA which uses LVTTL (Low Voltage TTL) and thus logic high is 3.3V and logic low is 0V. The logic pin is high during sampling and low during the hold operation.

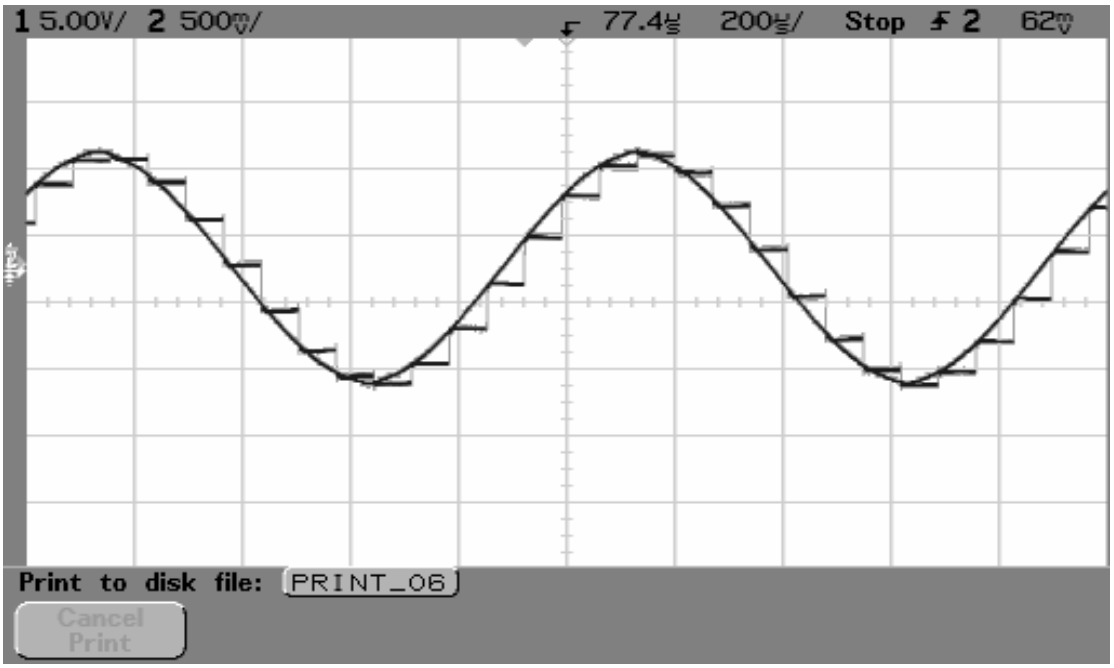


Figure 4.8: Output of sample and hold

The input of the SHA and the corresponding staircase output is shown in the figure. The figure 4.3 has been taken from a digital oscilloscope.

4.3.2 Analog to Digital Converter

The next step in the digital processing of an analog signal is the conversion of the output of the S/H circuit in its hold mode to a digital form. This is done by the *analog to digital converter*. The digital output consists of a number of bits, which is usually in binary code. The numbers of bits limit the achievable dynamic range as well as the accuracy of the converter.

Different methods have been developed for A/D conversion. The fastest type of A/D is the *Flash Type A/D*. It requires (2^n-1) comparators for n bit output. More than 10 bit flash type A/D is difficult to implement and is very expensive. Most IC flash type A/D is available in 2-8 bits output. For higher resolution, other types of ADCs are used. **Integration type ADCs** The c*ounting type* of ADC may also be used, in which a digital counter is employed which increases the value of a digital word at each clock pulse. A DAC calculates the analog equivalent of the digital word and compares it to the input

analog voltage. The counting stops as soon as the two voltages are equal. The conversion time depends on the value of the analog input. A relatively new type of high speed ADC is the *oversampling sigma-delta* ADC in which the sampling rate is much higher than the nyquist rate which enables the difference of the consequent samples to be expressed as 1 bit.

4.3.2.1 Practical Considerations for an ADC

All ADCs have an inherent limitation of resolution. A digital system cannot take any arbitrary value as an analog system can. A digital system recognizes a finite number of steps, and an analog value falling in-between two steps takes the closest one. The error arising due to this is known as the *quantization error* and is equal to ±1/2 of the value of the LSB. The only way to reduce this error is to increase the word length.

ADCs also have *linearity error* which occurs if the differences between two consecutive transition values are not same for the whole range of input. In AD574A this error is limited to ±1 LSB.

Offset error occurs if all transitions are shifted from their ideal locations by a equal amount. If this shift is not equal for all transitions *gain error* occurs.

In the implementation, successive approximation type ADC AD574A has been used. The datasheet is given in Appendix. The AD574A is a 12-bit analog to digital with tri-state output buffers, containing an on-chip high precision voltage reference and clock.

In systems where interfacing is done using a bus, the ADC is operated in "full control mode". However, in our FPGA system, we use a 12 dedicated input pins, and thus, the ADC is operated in the standalone mode, which do not support $\overline{\text{full}}$ bus interface. In this mode the CE and $12/\overline{8}$ are wired high and the CS and A_O are wired low. The conversion is controlled using $R/\overline{C}$.

There are two types of control in this mode. Conversion can be initiated with either a low pulse or a high pulse. In this implementaion the low pulse form is used. After initiation of the conversion with a low pulse, the data remains valid upto a time t_{HDR}. The width of the low pulse is t_{HDL} which must be at least 200ns. A time t_{DS} after $R/\overline{C}$ goes low, the STS signal goes high, indicating that conversion has begun. At this time the output is driven into a high impedance state and the data is invalid. A time t_C afterwards,

STS goes low. This is actually a period t_{HS} after the data becomes valid. The timing diagram is show in the figure 4.4.

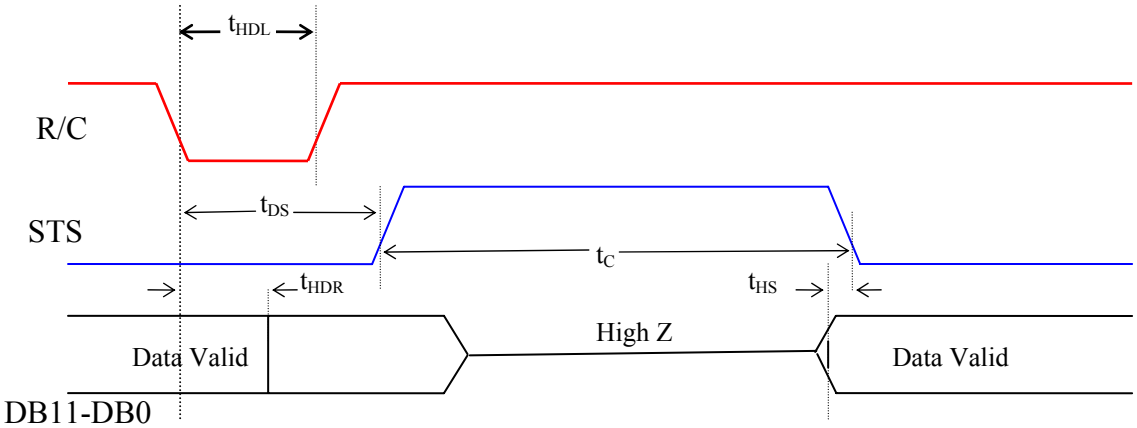


Figure 4.9: Timing Diagram of the ADC in stand-alone mode.

The minimum, typical and/or maximum values of each of the times are shown in the **table**. It is evident from the table that after the data is valid and the conversion is initiated, a maximum of 600 ns is required for the STS to assert and a maximum 35µs for the conversion to take place. Thus, output digital form is obtained 35.6µs after valid data is input.

Table 4.2 : Timing values in stand alone mode

Symbol	Parameter	Min	Typ	Max	Units
t_{HRL}	Low R/$\overline{C}$ Pulse Width	250			ns
t_{DS}	STS Delay from R/$\overline{C}$			600	ns
t_{HDR}	Data Valid After R/$\overline{C}$ Low	25			ns
t_{HS}	STS Delay After Data Valid	300		1000	ns
t_C	Conversion Time				
	8-Bit Cycle	10		24	µs
	12-Bit Cycle	15		35	µs

4.3.2.2 Determination of the Sampling rate

The sampling rate is limited mainly by the speed constraints of the the SHA and the ADC. It is obvious that for digital data to be obtained from an analog input, the acquisition time of the SHA and the total conversion time of the ADC is to be taken into

account. As mentioned before the SHA has a typical acquisition time of 4μs but is guaranteed only at 10μs. Thus the Sample and Hold Logic signal $S/\overline{H}$ must be kept high for 10μs. The sample has to be held for a minimum of 35.6μs for the ADC to finish conversion. The minimum total time that must be granted is thus 45.6 μs.

It may be noticed that we have used the maximum time or the worst-case values for each chip. A faster data acquisition could have been obtained if the sampling pulse (set $S/\overline{H}$ to high) have initiated at the falling edge of the STS signal. Hence potentially we could have a total conversion time of 25μs (including the acquisition time). This corresponds to a frequency of 40kHz.

However, this cannot be applied in this implementation. This is because in this case the digital data samples would not be acquired at regular intervals. However, for filtering applications, the requirement is to have an uniform sampling frequency. Thus to ensure valid data at every sample we consider the worst case values for both SHA and ADC and the maximum sampling rate becomes 21.93KHz.

In this implementation, a little more leeway, the sampling rate has been fixed at 14.237 KHz.

4.3.2.3 Output of the ADC

The output of the ADC is a 12-bit bipolar left justified data with range of -10V to 10V. This means that all zero output means -10V, and all ones signify 10V. The values of the digital words are tabulated below.

Digital Word	Analog Equivalent
000000000000	-10V
100000000000	0V
100000000001	4.88 mV
111111111111	9.997 V

Table 4.2: Digital Word vs. Analog Voltage

4.3.3 The Digital to Analog Converter (DAC)

The output of the XILINX FPGA is converted to analog form using DAC. The analog output of the D/A is proportional to the binary number fed to its input.

Analog output =K* digital input

Where K is a proportionality factor and is a constant for a given DAC.

There are a number of types of DACs. The most widely used DAC is the *R-2R ladder type D/A converter* that will be briefly discussed here. The circuit diagram is shown in the Figure:

This D/A employs only two values of resistors. So it is easy to fabricate. Here absolute values of the resistors are not important, but their ratio is important which is maintained at a value of 2.If the temperature changes, both the resistor values change in the same proportion. So their ratio remains more or less the same.

The current I_{OUT} depends upon the positions of the SPDT switches. The switches are controlled by the binary inputs of the D/A. The voltage V_{OUT} at the output of the Op-Amp is given by,

$$V_{OUT} = (-V_{REF}) \frac{\left(2^{11} B_{11} + 2^{10} B_{10} + 2^{91} B_9 + ...2^1 B_1 + 2^0 B_0\right)}{4096}$$

Which is proportional to the digital input. The same logic can be extended for any number of input binary bits. The D/A converter used in the project is DAC7541A. The datasheet of the IC DAC7541A is given in the appendix. It is a 12-bit multiplying D/A consisting of a highly stable thin film R-2R ladder network and 12 pairs of current steering switches on a monolithic chip.

4.3.3.1 Connections of the DAC

There are different circuit connections of the DAC that have been recommended; such as unipolar two-quadrant operation, unipolar two-quadrant operation with gain adjustment, bipolar four-quadrature operation etc. Among these, the unipolar two-quadrant operation circuit is used mainly, since it is the most simple circuit configuration. The input/output relationship is shown in the following table.

5 BINARY INPUT	6 ANALOG OUTPUT
MSB LSB 1111 1111 1111	$-\frac{4095}{4096} V_{REF}()$
1000 0000 0000	$-\frac{1}{2} V_{REF}$
0000 0000 0001	$-\frac{1}{4096} V_{REF}$
0000 0000 0000	0V

Table 4.4: Binary Input vs. Analog Output

The circuit diagram for the unipolar two-quadrant configuration of the D/A converter is shown in figure 4.5. The D/A converter has no input control signal, nor does it provide any signal to indicate end of conversion. Conversion time of D/A is much less than that of A/D and hence handshaking signals are not necessary.

The following constants have been used in the circuit diagrams

- V_+ = +15 V
- V- = -15 V
- L = Logic Low – connected to Digital Ground
- H = Logic High – connected to +5V.

It is seen that all input from the ADC to the FPGA has been stepped down using a voltage divider circuit. This is because the output of the ADC is TTL compatible and thus logic high is of the order of 5V. The FPGA is LVTTL compatible and logic high is 3.3V. Application of 5V to the input pins of the FPGA is harmful and must be avoided. The voltage from ADC is thus reduced by a factor of $\frac{33}{51}$ to transform it form TTL to LVTTL.

In the implementation, analog ground and digital ground have been separated and connected at one common point. This is because, digital systems produce high frequency switching noise. Digital systems being robust are not affected by such noise. However,

analog systems are affected by this type of noise. If the grounds of both digital and analog parts were connected to the same wire, due to the non-ideal nature of the ground connection, high frequency interference from the digital part would creep into the analog part. However, both the digital and analog needs the same reference. Thus analog and digital grounds are separated, but they are connected at one point.

5 Simulation and Experimental Results on Performance of the Filters

5.1 Introduction

In this Chapter, the frequency responses of each of the filters implemented have given. The responses of the filters are also compared with the ideal responses and their analog equivalent, both taken from MATLAB. The ideal responses have been generated using a modified version of the *freqz* function named *myfreqz.* In this the magnitude response is plotted on a semilog graph instead of the normal linear graph. The frequency in Hertz is plotted instead of the normalized frequency. No change has been made to the Phase response part of the graph. This is plotted on a linear graph paper to show the linearity of phase in FIR filters.

5.2 FIR Filters

The fir filters that have been implemented are of very high order, near about 100. In each of the filters, the order has been estimated using the *remezord* function. The order is estimated for a transition band between of 100 Hz and a pass and stop band deviation of 0.1. The *fir1* function is used to generate the coefficients and they are written using a MATLAB program in the specified format.

5.2.1 Low Pass FIR Filter

In this example the cut-off frequency is 550 Hz, the transition band is from 500Hz to 600Hz, and the allowed deviations are .1 in both stop and pass bands.

The impulse response of the system is shown in figure 5.1. The impulse response actually gives the coefficients, so we have an idea about the magnitudes of the coefficients from the figure. The impulse response is generated by using the *filter* function, using a unit impulse as input and the scaled up coefficients. The impulse response, i.e. the coefficients, are symmetric and resembles a truncated sinc function. The impulse response shows that the filter is stable, as the output comes back 0.

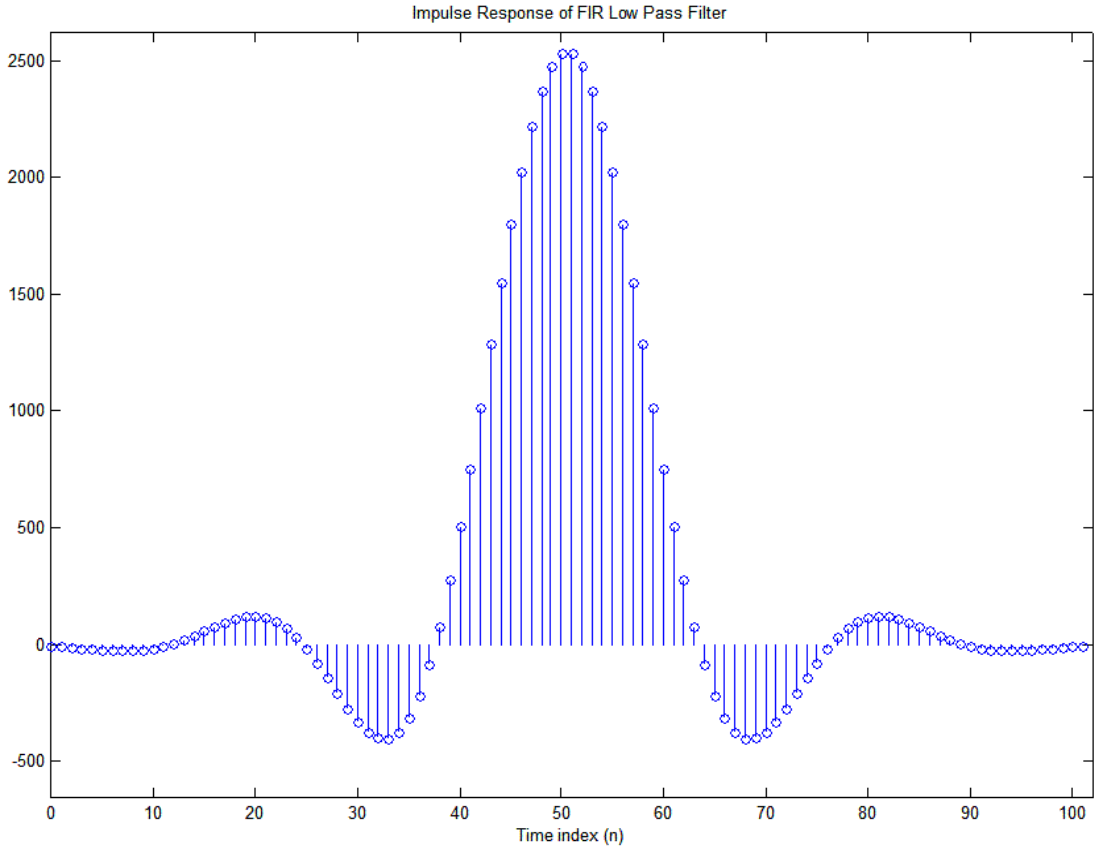
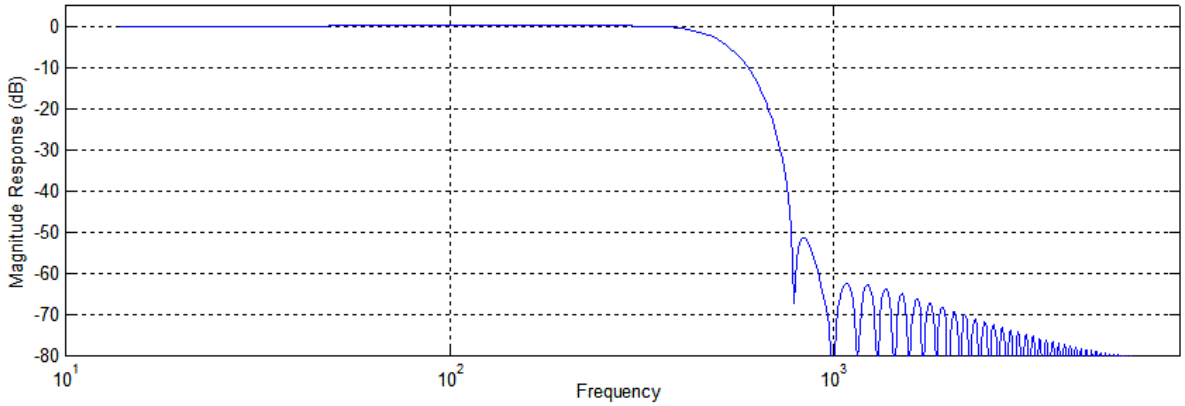


Figure 5.1: Impulse response of FIR Low Pass Filter.



Ideal response from MATLAB

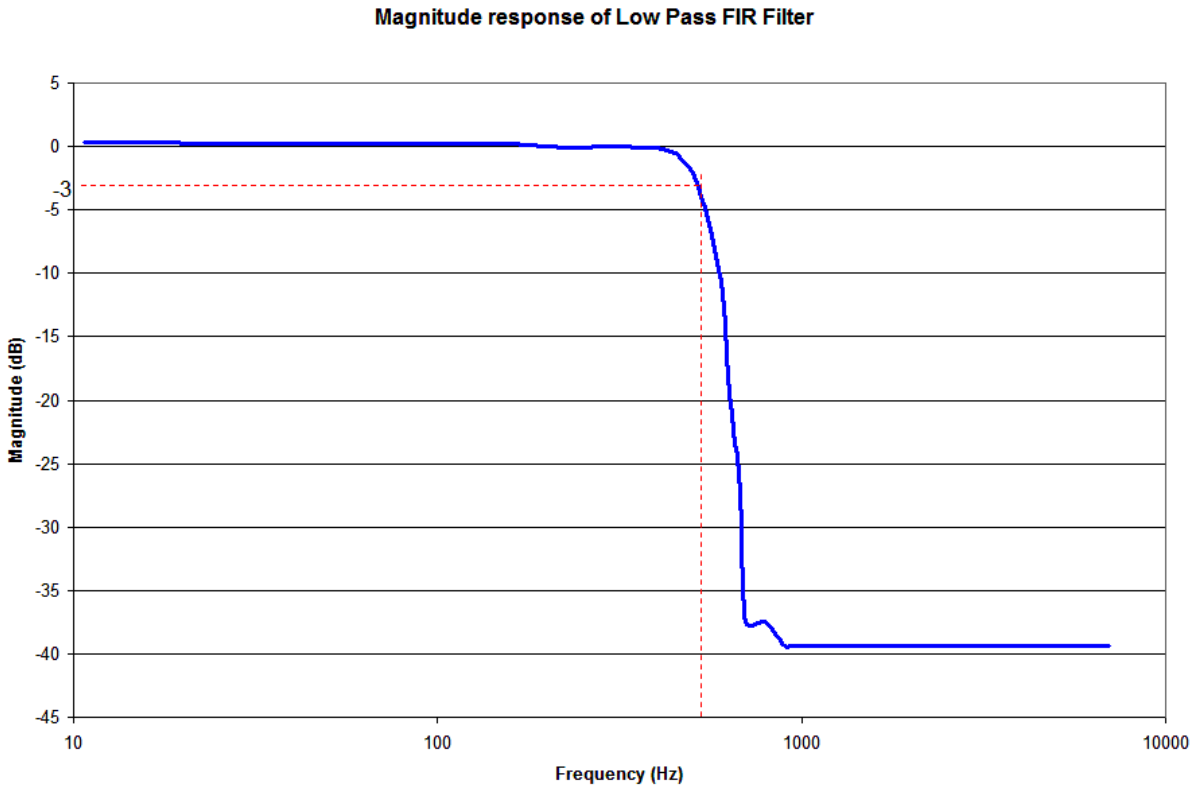
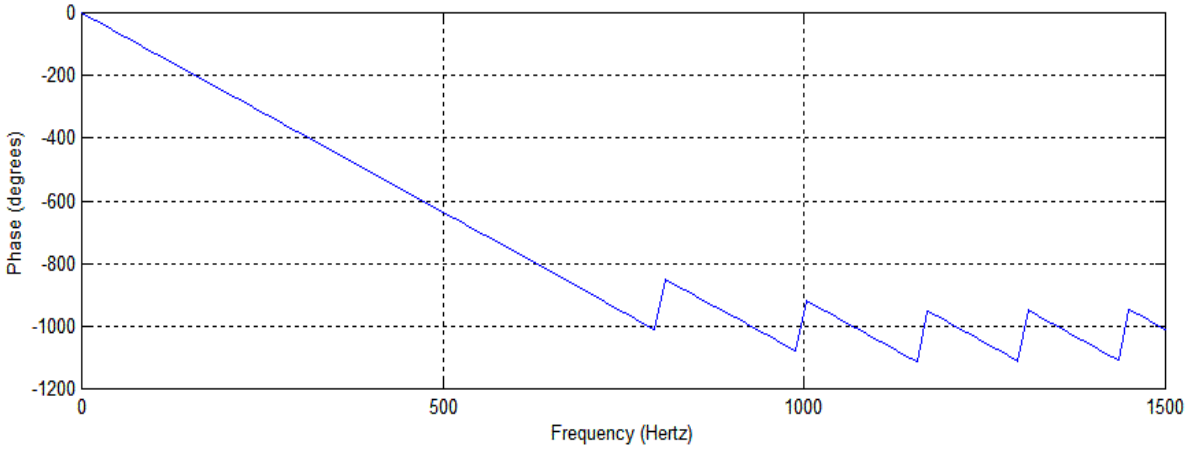


Figure 5.2: Experimental Magnitude response of FIR Lowpass filter

The "ideal" response shown above uses the truncated coefficients. This is very nearly equal to the response shown by MATLAB using high precision floating point values.

Only the stop band characteristics are a little distorted. The -3dB point is approximately 525Hz as opposed to the ideal 550Hz.



Ideal response from MATLAB

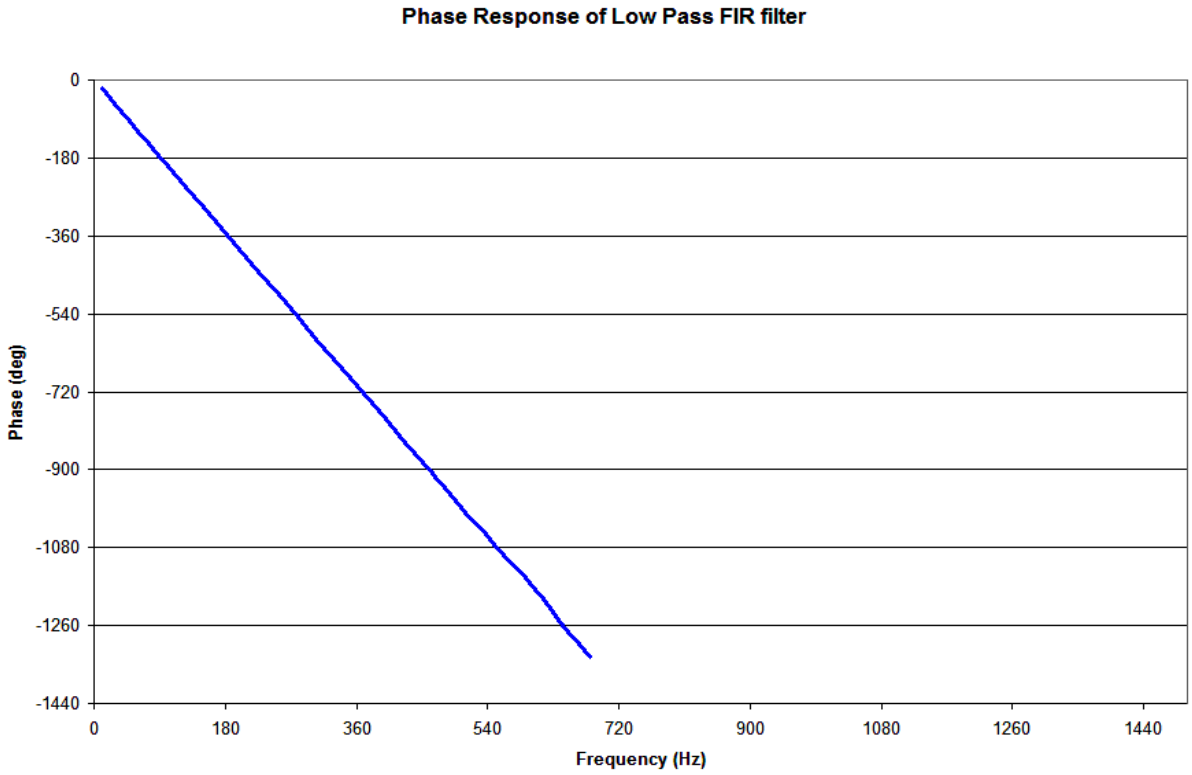


Figure 5.3: Experimental Phase response of FIR Low pass filter

The phase response could not be calculated for the stop band frequencies. This is because of the noise at these levels. Only the pass band response is shown. All readings were based on phase angles between -180° and +180°. However, as MATLAB represents the data in unwrapped phase angles, it can be also shown it as such by adding appropriated multiples of 360° at the required points. This shows the linear phase response of the filter in the pass band.

5.2.2 High Pass FIR Filter

In this example the cut-off frequency is 550 Hz, the transition band is from 500Hz to 600Hz, and the allowed deviations are .1 in both stop and pass bands.

The impulse response is shown in figure 5.4. The impulse response is obviously symmetric. The middle value is very high (945) and is shown in full, so that the shape of the impulse response can be properly seen.

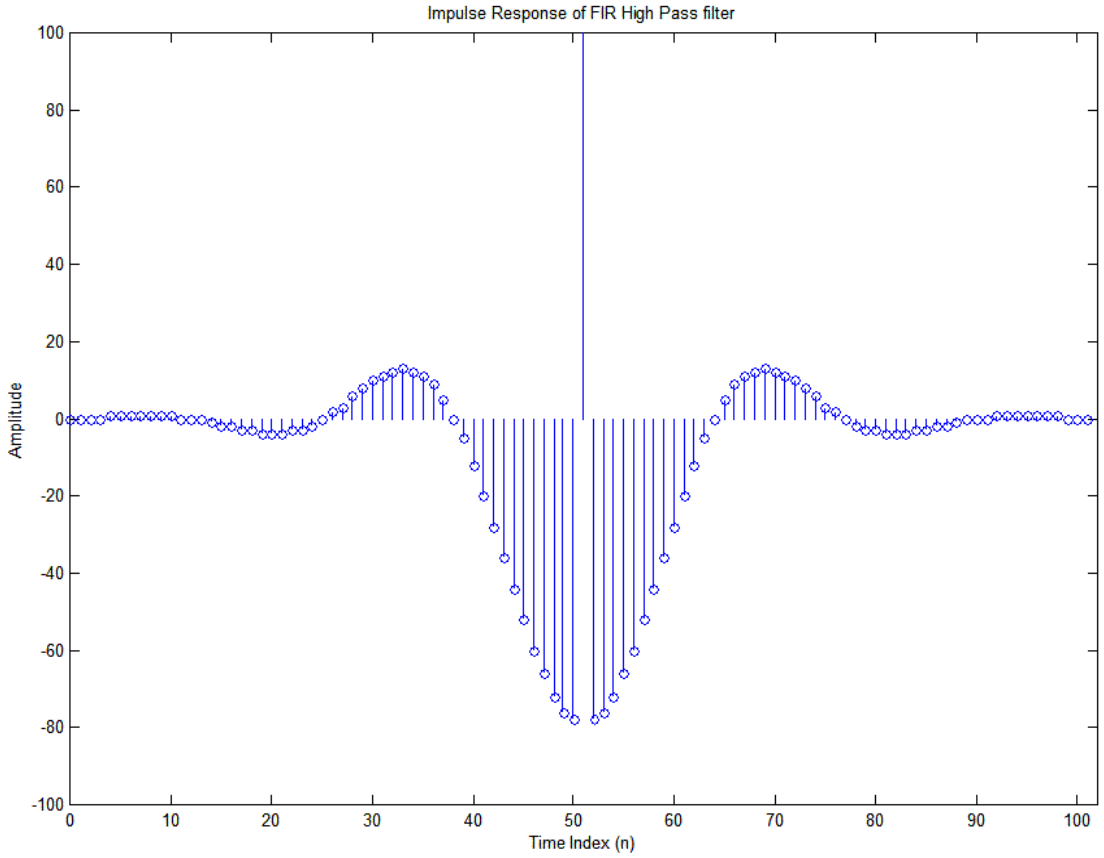
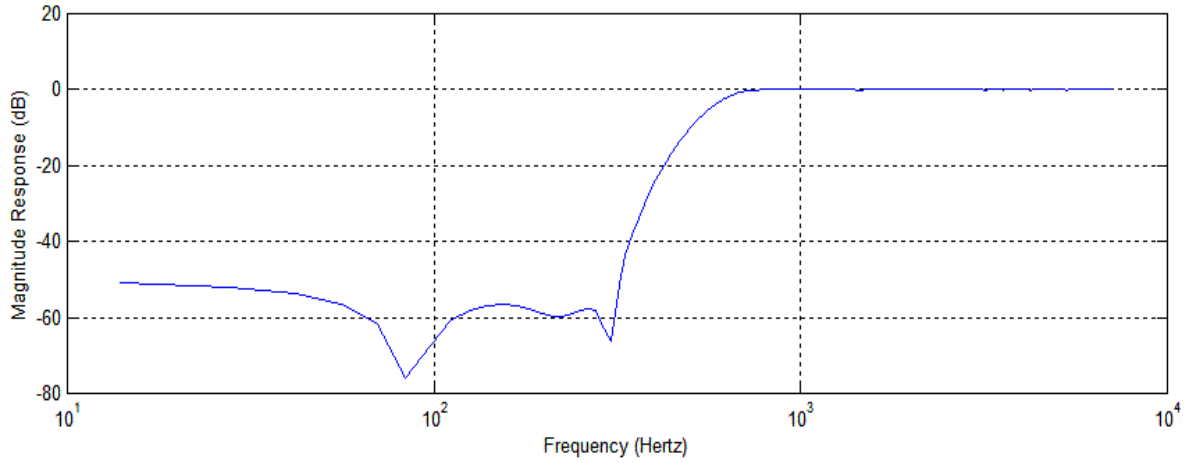


Figure 5.4: Impulse response of FIR High Pass Filter.



Ideal response from MATLAB

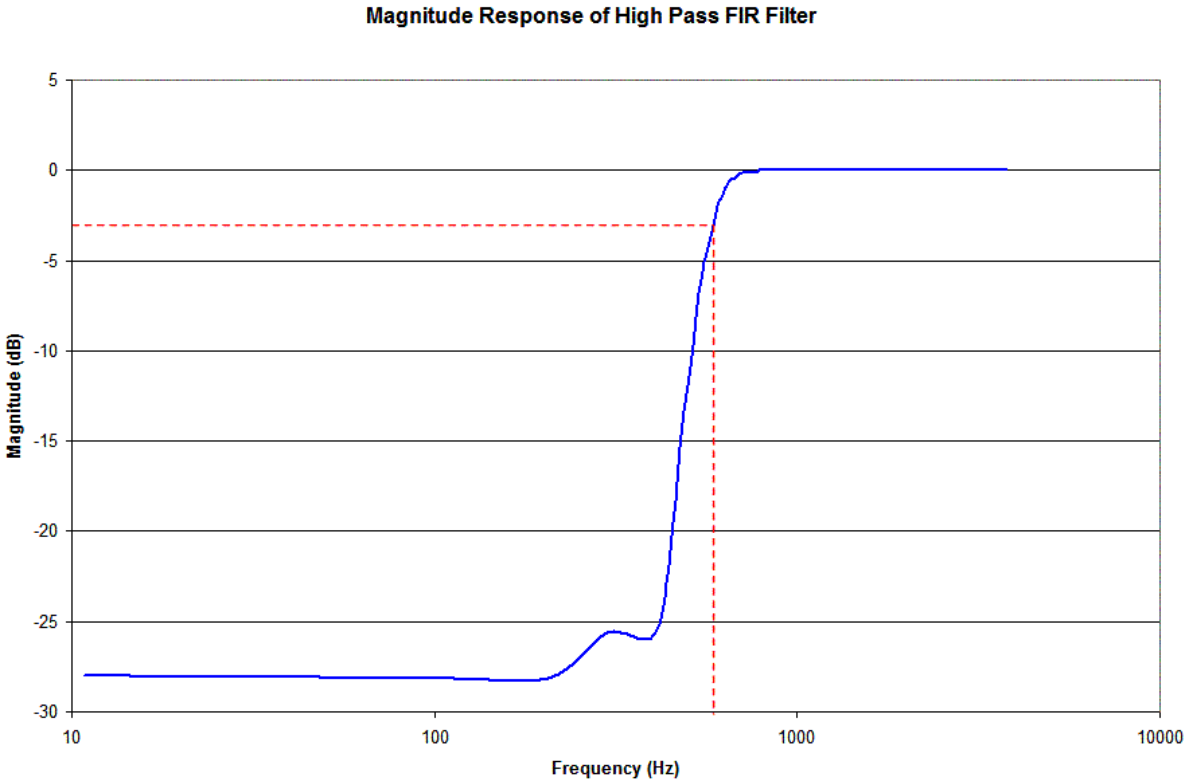
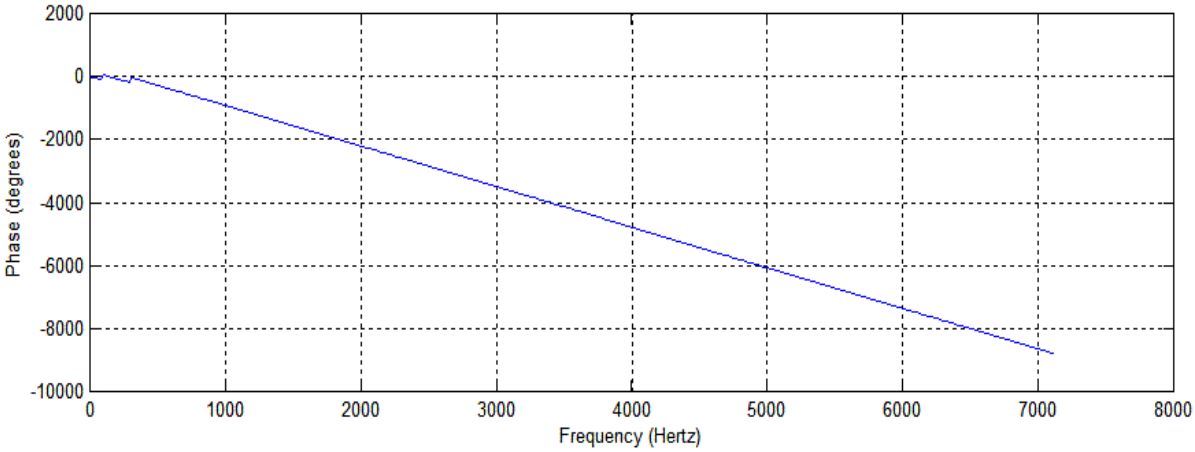


Figure 5.5: Experimental Magnitude response of FIR High pass filter

In figure, the ideal and the experimental magnitude responses are shown. The dotted lines show the -3dB magnitude. The -3dB point is measured to be 580Hz rather than the

550Hz of the design. In the readings, it is difficult to measure values below -30dB. Actually, readings shown are scaled to make the pass band attenuation to be 0dB.



Ideal response from MATLAB

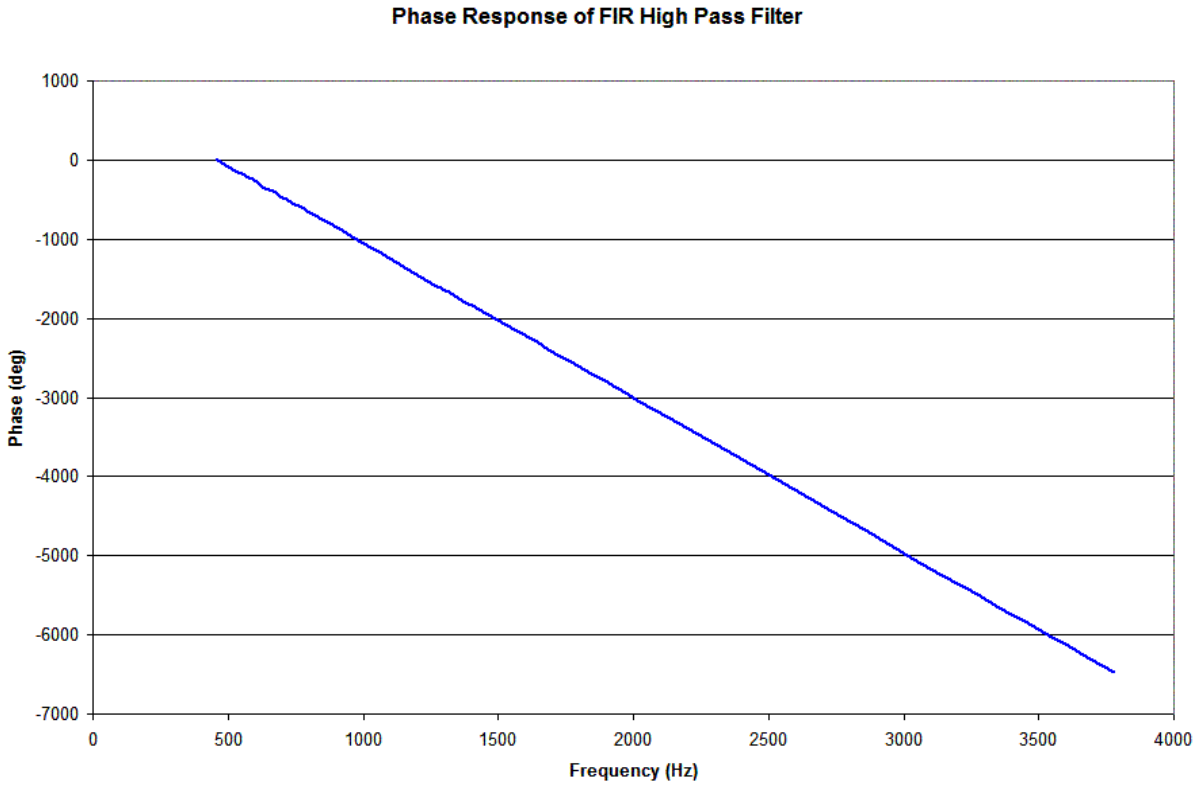


Figure 5.6: Experimental Phase response of FIR High pass filter

However, in reality, the output is attenuated due to the properties of the DAC, and the

readings lower than -30dB cannot be distinguished from noise.

5.2.3 Band Pass FIR filter

In this design, we have the cutoff frequencies of 300Hz and 3000Hz. Thus the and the transition width of 100Hz around both the cut-offs. The pass band is from 250Hz to 2950Hz.

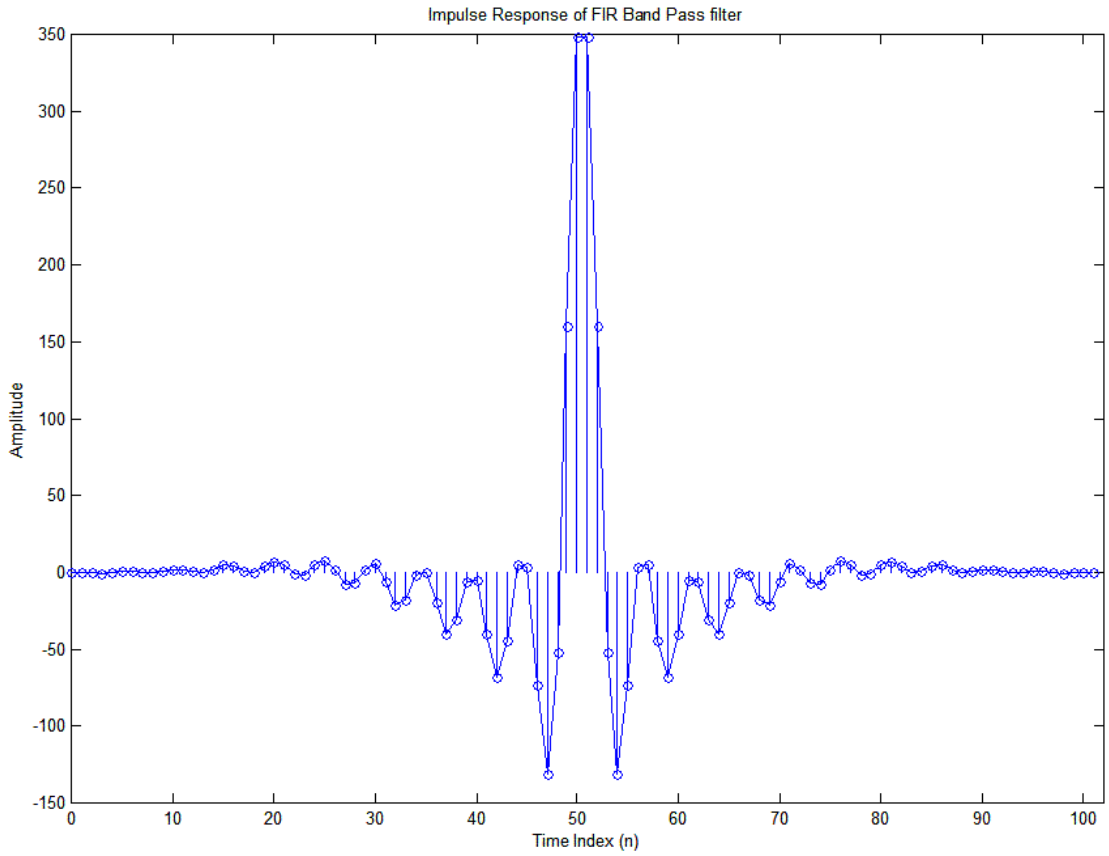
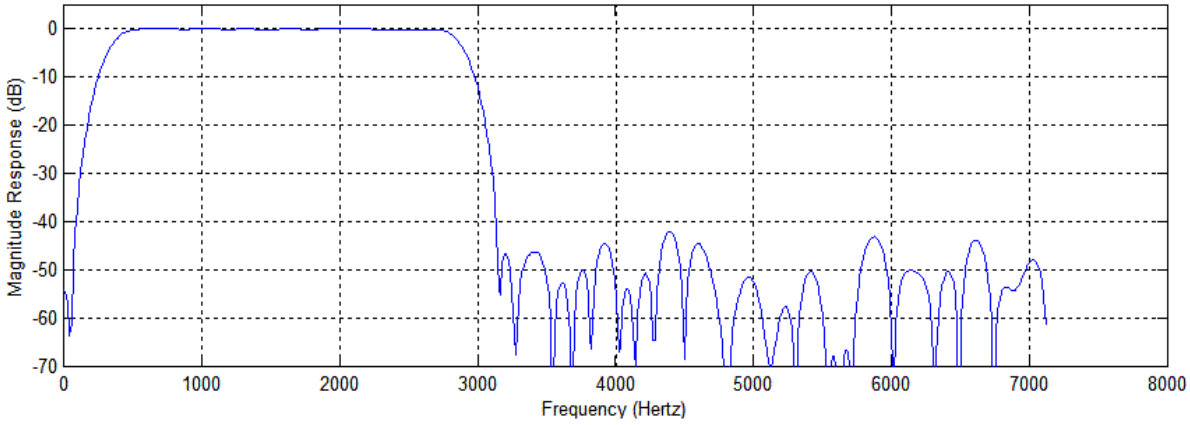


Figure 5.7: Impulse response of FIR Band Pass filter

Figure 5.7 shows the impulse response of the system. The Magnitude response of the system is shown along with the ideal. The comb-like responses present in the ideal are not present our response because it is below the noise level. The magnitude response is shown on a normal graph instead of a semi-log plot.



Ideal response from MATLAB

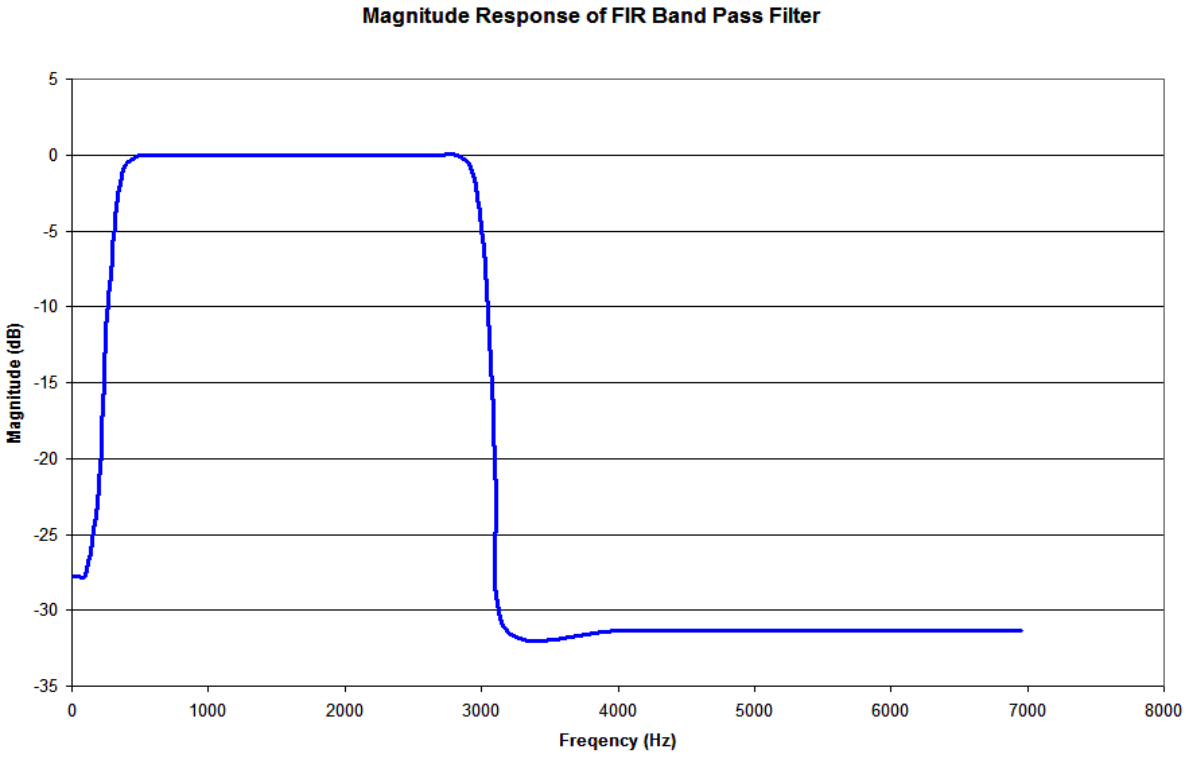
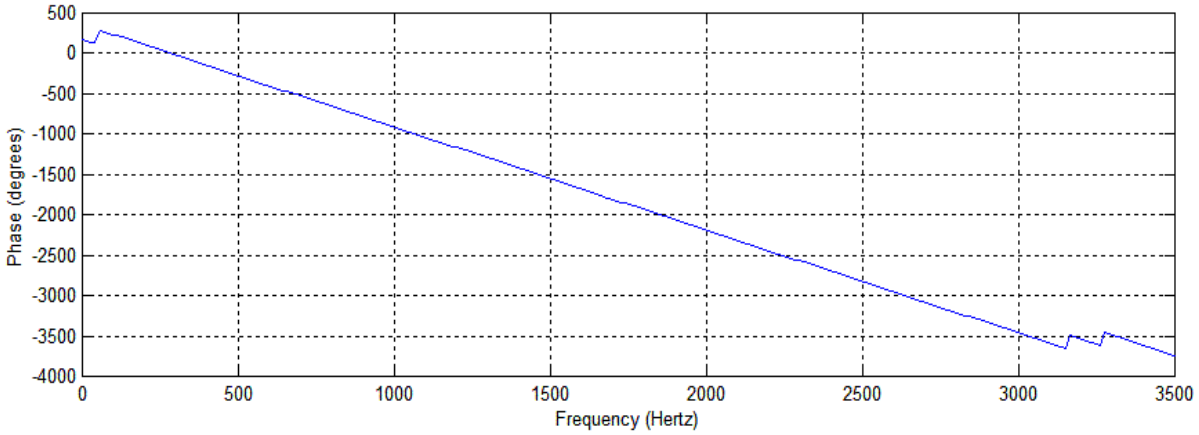


Figure 5.8: Experimental Magnitude response of FIR Bandpass filter

The -3dB points are found to be 324Hz and 2990 Hz for the two transitions.



Ideal response from MATLAB

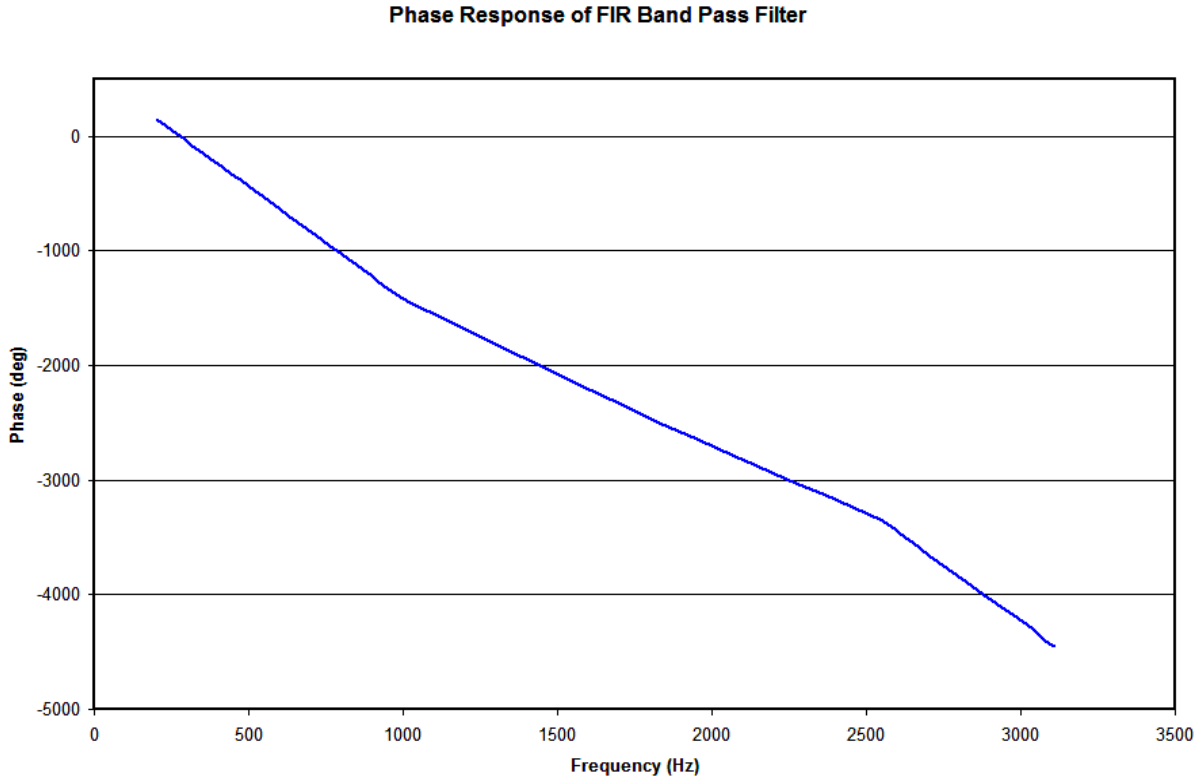


Figure 5.9: Experimental Phase response of FIR Bandpass filter

The phase response shown in figure 5.9 is approximately linear.

5.2.4 Band Stop FIR filter

In this design, we have the cutoff frequencies of 500Hz and 3000Hz. Thus the and the transition width of 100Hz around both the cut-offs. The stop band is from 550Hz to 2950Hz.

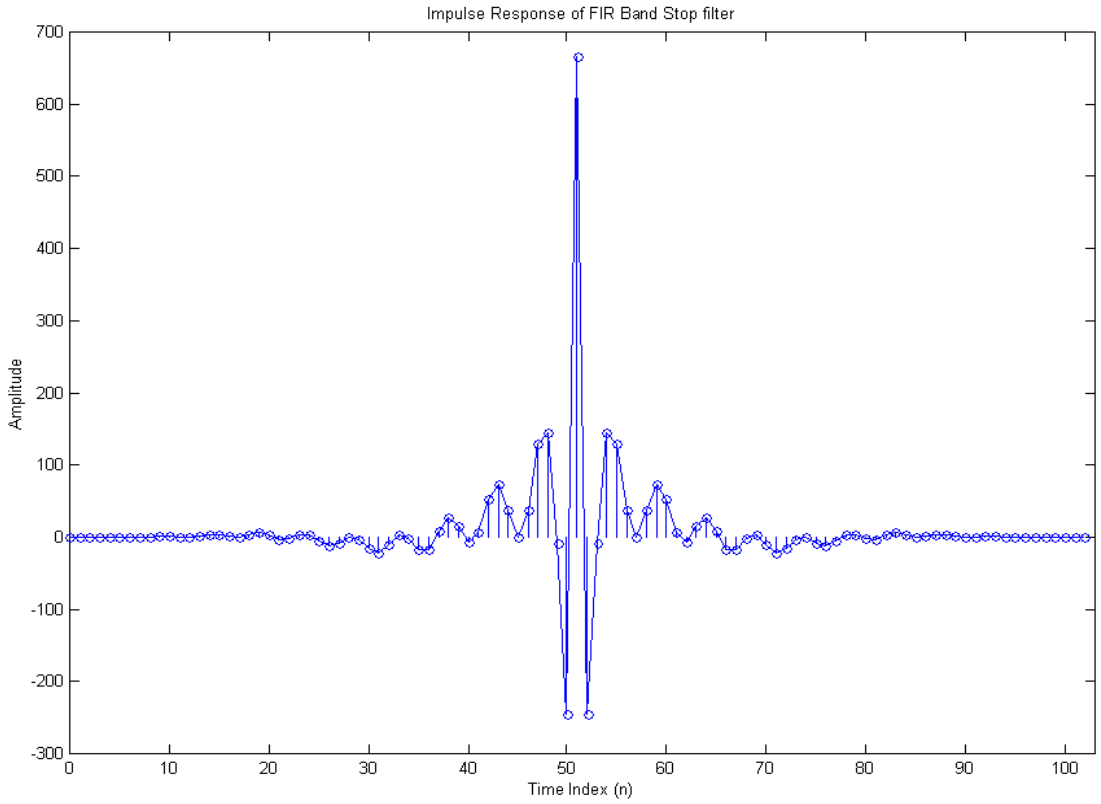
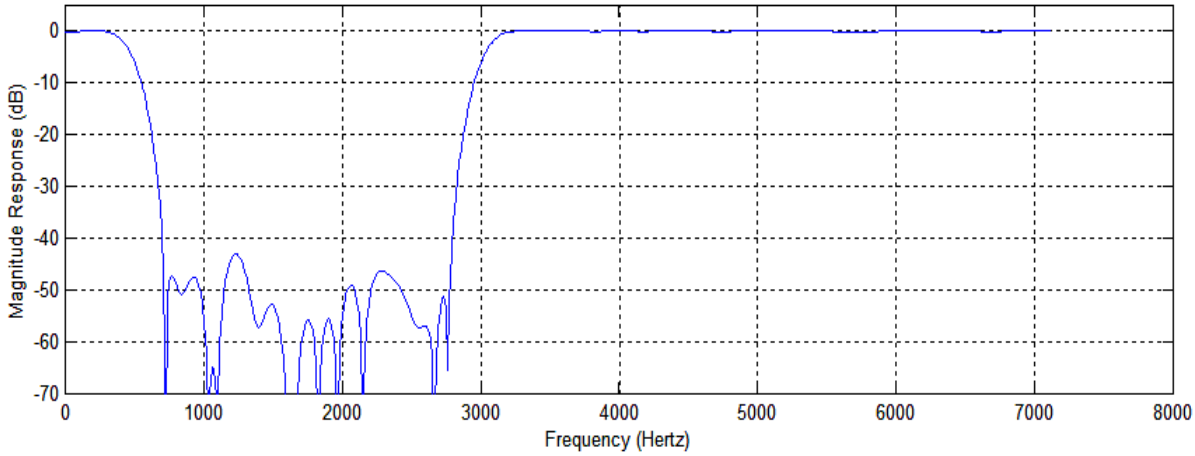


Figure 5.10: Impulse response of FIR Band Stop filter

Figure shows the impulse response of the system. The Magnitude response of the system is shown along with the ideal on a normal graph, like the band pass filter.



Ideal response from MATLAB

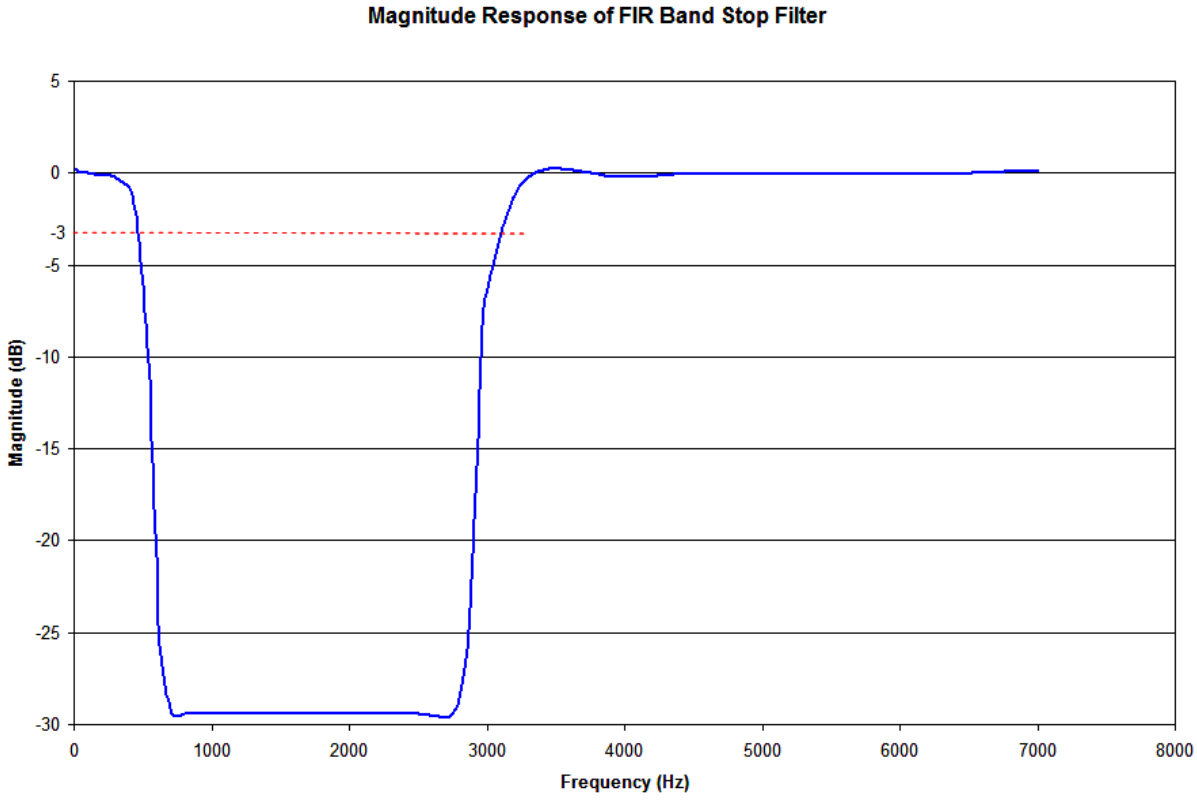
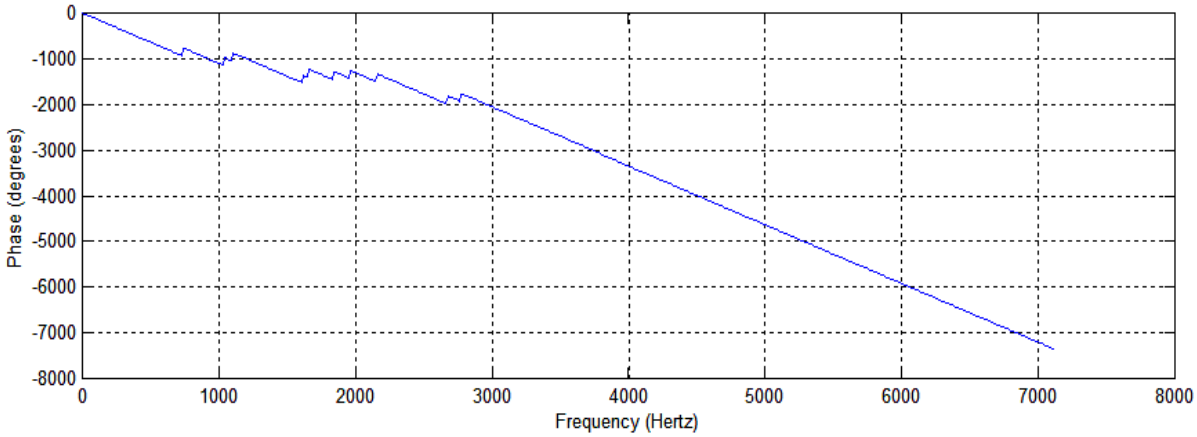


Figure 5.11: Experimental Magnitude response of FIR Bandpass filter

The -3dB points are found to be 475Hz and 3100 Hz for the two transitions.



Ideal response from MATLAB

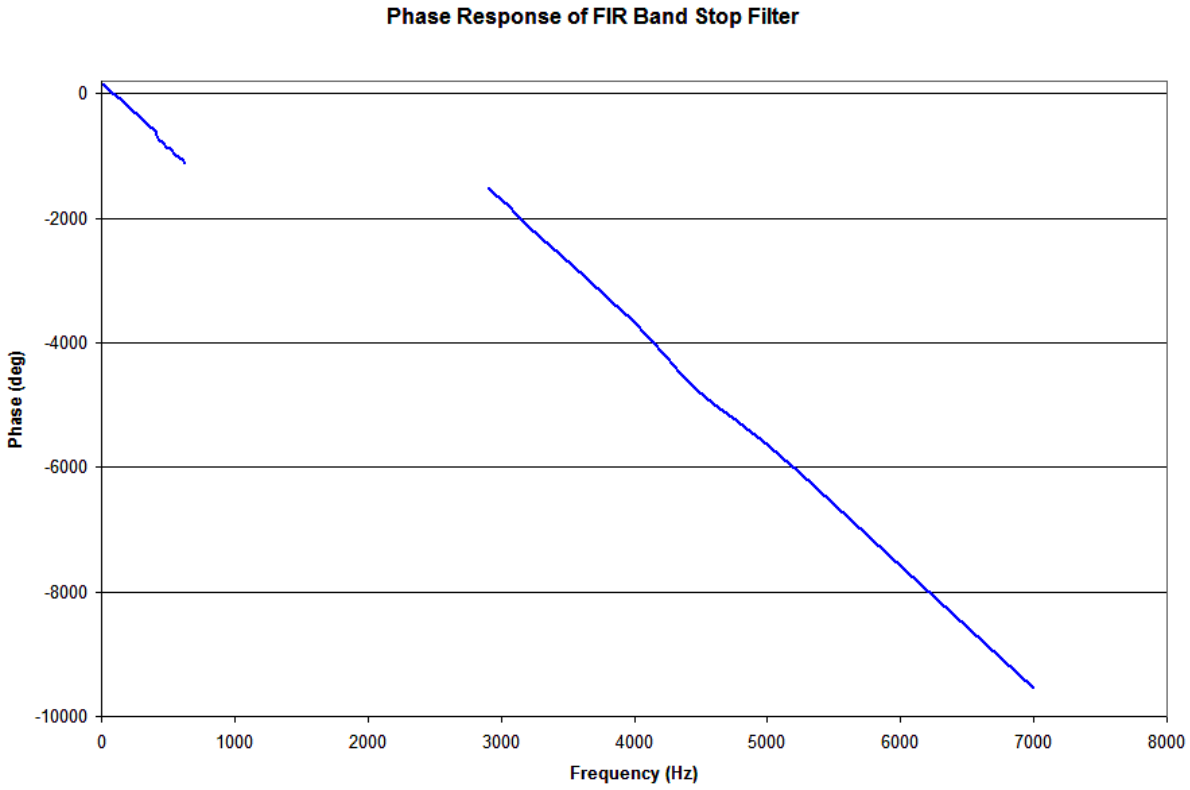


Figure 5.12: Experimental Phase response of FIR Band stop filter

The phase response shown in figure 5.12 is approximately linear.

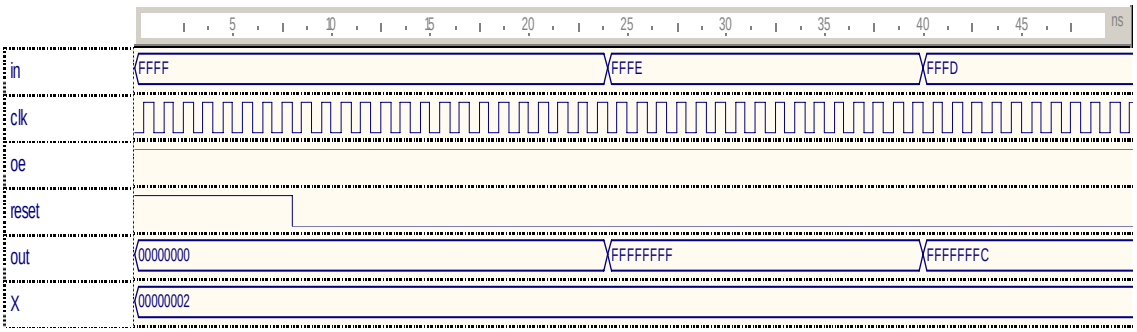


Figure 5.13: Experimental HDL Simulated response of FIR filter

5.3 Relative performance

In this section, the performance of the filters has been presented in both theoretical and experimental way. The filter throughput has been determined by theoretical approach. The quality or performance of the filters has been determined through implementation. Also from the MAP report the CLB count has been given in tabular format.

5.3.1 Filter Throughput

Specifications	DSP's performance	FPGA's performances
16-Tap, 8bit FIR	3.125MSPS	129MSPS
256 Taps, 16bit FIR	15.5MSPS @ fclk = 1 GHZ	300MSPS @ fclk = 300MHz

5.3.2 Q Factor

Specifications	Q Factor
102-Tap, 12bit FIR, Low Pass	14.2
106- Tap, 12 bit FIR, High Pass	12.5
204- Tap, 12 bit FIR, Band Pass	10
254-Tap, 12 bit FIR, Notch	10

5.3.3 CLB count

Specifications	CLB Count
16-Tap, 8-bit FIR, Serial Arch	400
16 Tap, 8 bit FIR, Optimized Arch	320
16 Tap, 8 bit FIR Full Parallel arch	270

6 Conclusions

In this project work, we have implemented various basic filters viz. Low Pass, High Pass, Band Pass, Stop Band Filters using Discrete time convolution equation algorithm on Field Programmable Gate Array. Conclusions of our thesis are:

- We are able to implement Digital Filters on FPGA with comparable performance as on conventional DSPs. The comparison between the ideal and the experimental results given in Chapter 5 shows the satisfactory performance of the filters.

- We have shown the Discrete time convolution equation algorithm described in Chapter 3 suits the architecture of Field Programmable Gate Array especially for the design of Digital Filters and it can efficiently use the resources available in the device.

- We have also shown how to interface the real analog signals with the FPGA after proper conditioning and again how it can be converted back to analog form after processing it digitally in the FPGA. Chapter 4 describes the approach of interfacing the analog signals with the FPGA and implementation of the Discrete time convolution equation based Filters on FPGA in detail.

7 Discussions and Future Work

7.1 Introduction

As a designer of Digital Signal Processing systems, we have a large number of choices to implement our solution. Each solution has its strengths and weaknesses. The purpose of this chapter is to bring out where FPGAs can be used for DSP based applications. In the following sections, a comparison of the implementation of a simple DSP function in both Programmable DSP (pDSP) and Field Programmable Gate Array has been described.

7.1.1 Implementation through Programmable DSPs

The most common vehicle for implementation of a DSP design is *the programmable DSP* or pDSP. The pDSP is an off-the-shelf part that is essentially a microprocessor tuned to DSP applications. PDSPs are highly flexible because you can program them again and again using a familiar high-level language like C. They allow fast design iterations and reduce time to market. Typically a pDSP contains several functional units to process the signal stream. The designer encodes the algorithm into a program, which is executed by the pDSP and is limited to a theoretical maximum data rate based on the speed and the number of multiplier/ accumulators in the device. Applications, which require several computations, must be broken up into a sequential stream of computations. For example, an 8-tap FIR filter requires 8 multiplications and one 8-way addition per data sample.

The implementation of this FIR filter might require 8 or more cycles on a pDSP as shown in Figure 6.1. At each data sample, all eight taps require multiplication by their coefficients. If the number of taps for this filter were increased, then the number of cycles would also be increased, hereby reducing the data rate. Programmable DSP chips are intrinsically limited in performance. The more you want to do to a data sample, the more cycles you need and the slower your data is processed. One way to overcome this limitation is to employ more pDSP parts to implement the algorithm. Another method is to use Gate Array technology to implement the algorithm in hardware.

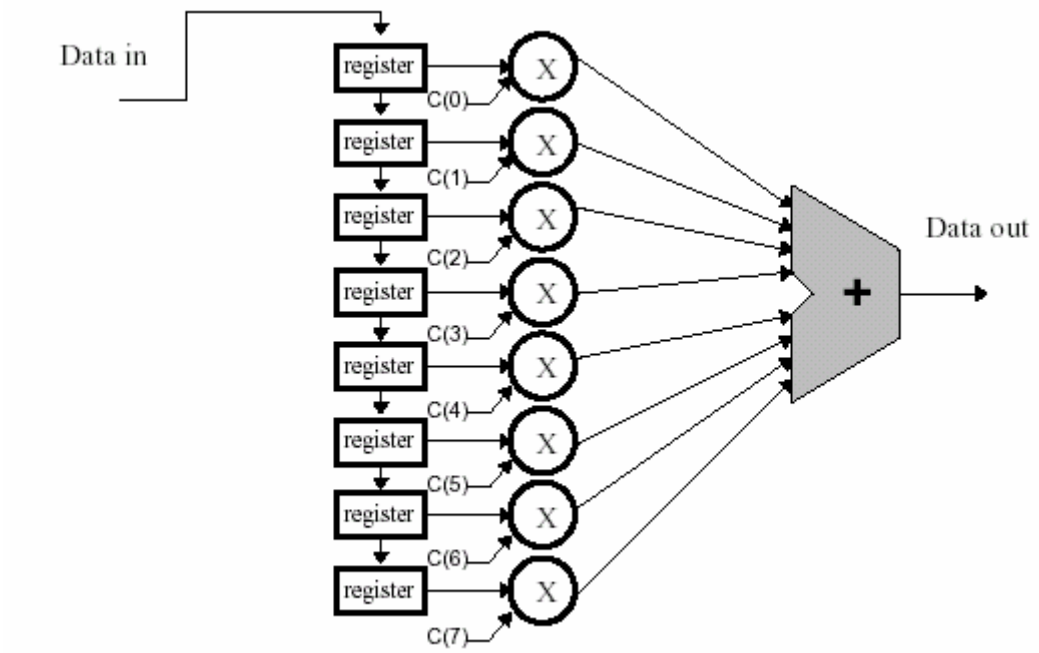


Figure 7.1: Filter structure

MultAcc Reg(0), C(0)
MultAcc Reg(1), C(1)
: : :
MultAcc Reg(7), C(7)

The downside of a Gate Array is that it is a custom part. Once designed, it must be custom or semi-custom fabricated at a silicon foundry. Due to the unique nature of the device a custom Gate Array typically requires several weeks to be fabricated from the prototype plans, and due to the expense of the overall process the design must be carefully verified prior to the manufacture of even small quantities. Design flaws found after fabrication require costly and time-consuming "spins" of the design.

7.1.2 Implementation through FPGAs

Field Programmable Gate Arrays are a technology, which gives the designer a combination of the benefits of a gate array solution and the ease of pDSP design. An FPGA design starts with the same input as a Gate Array design - i.e. a circuit schematic or high-level design description. Automatic synthesis, place, and route tools are used to translate the designer's original circuit into an FPGA specific *configuration*. The big

difference between the Gate Array and FPGA design process is that the user specific custom manufacturing process is eliminated. FPGAs are generic commodity parts and are customized by downloading a user defined configuration in the form of a binary *bitstream*, much the same as you would load a pDSP with its program with a process that typically takes only a few milliseconds How much of an advantage does the FPGA's ability for direct implementation buy you? The answer really depends on how many tries you think you will need to converge on a correct implementation. Most DSP designs are part of a complex system. Experience shows that it is very common for complex systems to go through several design iterations before product completion. Figure below compares the development cycle of anew design as implemented on Gate Array versus FPGA technology. This development cycle includes extra iterations to fix bugs. By virtue of immediate implementation, an FPGA based design solution is ready for delivery much earlier than a Gate Array design. Why, if FPGAs provide such benefits, would anyone use Gate Arrays? The answer is that for very high production volumes, FPGAs do not exhibit as high of a performance/cost ratio as Gate Array or full custom Application Specific Integrated Circuit (ASIC) designs. The lower performance/cost of FPGAs comes in part because FPGAs must sacrifice some silicon area in order to be highly flexible. However, this should not be a concern to the designer expecting to ramp up to high volume. FPGA offers a design migration product called *Hardwire, which* retains much of the performance/cost advantage of Gate Arrays. Once the designer has converged on a.

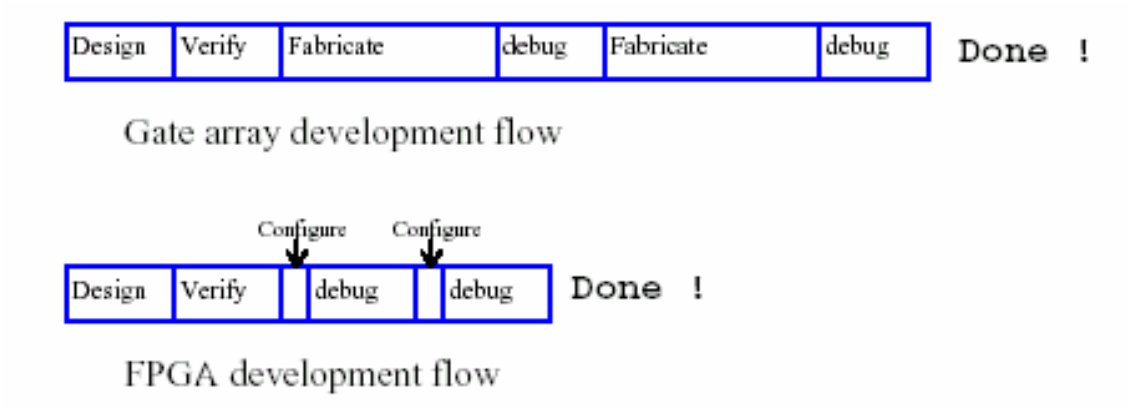


Figure 7.2: Comparison between Gate Array and GPGA based development cycle

working FPGA design, he can then translate the design to an equivalent Hardwire device without the need to redesign or debug the system as shown in Figure 6.2. In summary, FPGAs offer the rapid design cycle of programmable DSPs with the flexibility and raw performance of Gate Array products

A. Other advantages of FPGAs include:

1. Parts may be reprogrammed over and over. If you want to upgrade your design, you do not need to replace FPGAs, just reprogram them.
2. FPGAs are pre-tested. Traditional Gate Array design methodology requires that you also develop costly manufacturing test suites. This task is not required with FPGAs.
3. FPGAs are a commodity part. Xilinx or Altera sells millions of FPGAs annually. This high production volume results in a lower per part cost and those savings are passed on to the customer.
4. FPGAs can be dynamically reconfigured within the system. Sophisticated designers can build systems, which adapt to changing conditions by altering the circuit configured within the FPGA. This re-configurable design approach is becoming more and more popular since many systems need to perform several different functions, but never all of them at the same time.
5. In one FPGA you can build several filters or several systems. Then the SOC design will be more compact. Because a whole system can be made on a single chip.

B. FPGAs possess the following features, which enable high performance DSP design:

1. Flexible logic blocks with bit level arithmetic features - allows Discrete time convolution equation implementations of DSP algorithms.
2. Fine grained distributed RAM and ROM - Increases operand bandwidth.
3. A register rich architecture - enables a high degree of pipelining leading to increased performance.

7.1.3 Cost Comparison

Performance gains of more than one order of magnitude are available with FPGA based DSP at a small fraction of the cost compared to multiple processor solutions. The FPGA based DSP design methodology also has fewer and less complicated steps.

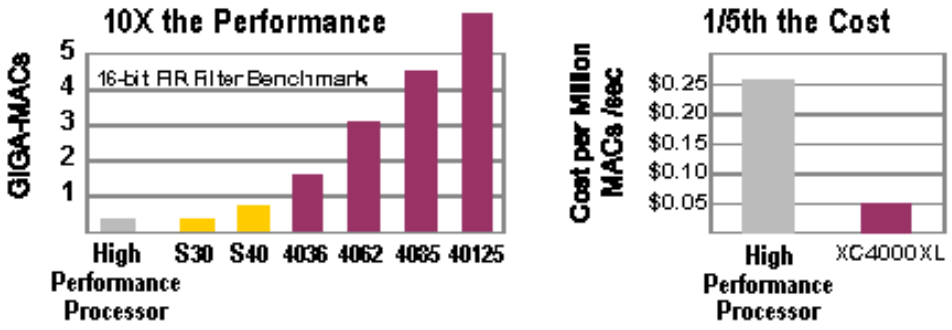


Figure 6.4: Cost and Performance (Source Xilinx Inc.) [DS2003]

In addition to the high capacity XC4000 family, Xilinx DSP now also supports the low cost Spartan family of FPGAs that competes directly with custom gate arrays. Further reducing the cost per MEGA-MAC in DSP applications. Any Spartan FPGA device is available for less than $20.00 in gate array volumes, and in many applications the S40 can do the work of two high-end DSP processors

7.2 Suggestions for Future Work

In this thesis all implementations have been done on Xilinx FPGA. But with the growing industry, architecture of the FPGA also has been changing. Different manufacturers are using different architecture to make their product versatile. Here comes the question of universality. The algorithm we have used is it independent of the device architecture. So, further work can be done to apply the same algorithm to different architecture and compare the relative performances.

Our main concentration in this work was FIR and IIR filters. The same algorithm can also be applied to other related areas in the field of Digital Signal Processing like FFT, Adaptive Filtering, Kalman Filtering, and Fuzzy Filtering. A future enhancement of this project can be the application of the discrete time convolution equation to other related applications.

We have used easily available ADCs and DACs for our implementations and the Filter throughput we have calculated theoretically. This can also be tested practically if we can use much faster ADCs and DACs.

Here, in our implementation we implemented different filters on the FPGA, but never tested in an assembly of filters. Here is a broad domain comes into picture how much area we can reduce to accommodate several filters or systems on a single chip without sacrificing the desired characteristics.

References

[Cart86]

W. Carter, K. Duong, R. H. Freeman, H. Hsieh, J. Y. Ja, J. E. Mahoney, L. T. Ngo and S. L. Sze, "A User Programmable Reconfigurable Gate Array," Proc. 1986 Custom Integrated Circuits Conference, May 1986, pp. 233-235.

[DB2000, Xilinx]

The Programmable Logic Data Book 2000, Xilinx Inc.

[DS2003]

Xilinx Data Source CDROM, 2003

[ElGa88]

A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, "An Architecture for Electrically Configurable Gate Arrays," Proc. 1988 Custom Integrated Circuits Conference, May 1988, pp. 15.4.1 - 15.4.4.

[ElGa89]

A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat and A. Mohsen, "An Architecture for Electrically Configurable Gate Arrays," IEEE Journal of Solid State Circuits Vol. 24, No. 2, April 1988, pp. 394-398.

[Fran90]

R.J. Francis, J. Rose and K. Chung, "Chortle: A Technology Mapping Program for Lookup Table-Based Field-Programmable Gate Arrays," Proc. 27th Design Automation

Conference, June 1990, pp. 613-619.

[Goslin95]

Gregory Ray Goslin, “A Guide to using Field Programmable Gate Arrays for Application Specific Digital Signal Processing Performances”, 2100 Logic Dr., San Jose, CA 95124, 1995

[Plus90]

Plus Logic FPGA2020 Preliminary Data Sheet, 1990.

[Xili89]

The Programmable Gate Array Data Book, Xilinx Co., 1989.