VLSI IMPLEMENTATION OF PIPELINED FIR FILTER

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

MASTER OF TECHNOLOGY

(VLSI DESIGN & CAD)

Submitted by:

AARTI SHARMA

Roll No: 601161010

Under the guidance of:

Mr. SANJAY KUMAR

Assistant Professor

ELECTRONICS AND COMMUNICATION ENGINEERING DEPARTMENT

THAPAR UNIVERSITY
(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June, 2013

DECLARATION

I hereby declare that the work which is being presented here in the dissertation entitled **"VLSI IMPLEMENTATION OF PIPELINED FIR FILTER"** in partial fulfilment of the requirement for the award of degree of M.Tech (VLSI Design & CAD) at Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of Mr. Sanjay Kumar, Assistant Professor, ECED and refers other researcher's work which are duly listed in the reference section.

The matter presented in this dissertation has not been submitted in any other University/Institute for the award of my degree.

Dated: 14/06/2013

Aarti Sharma

Roll No. 601161010

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

Dated: 14/June/2013.

Mr. Sanjay Kumar

Assistant Professor

ECED, Thapar University

Counter Signed By:

Head

ECED, Thapar University

Patiala-147004

Dean of Academic Affairs

Thapar University

Patiala-147004

ACKNOWLEDGEMENT

I would like to take the opportunity to express my gratitude to some people who were involved in this dissertation work. First, I owe my gratitude to my mentor Mr. Sanjay Kumar for doing everything from the inception of the project idea to giving invaluable suggestions at every step.

I am also thankful to Dr. R. K. Khanna, Head of the Department and Dr. Kulbir Singh PG Coordinator as well as all the faculty members and staff of the Department of Electronics and Communication Engineering for being very supportive to me. I would also like to thank all my batchmates for motivating me all the time whenever I needed them and giving me useful tips. I thank all those who have contributed directly or indirectly to this work.

Lastly, I would also like to thank my parents for their years of unyielding love and encourage. They have always wanted the best for me and I admire their determination and sacrifice.

Aarti Sharma
601161010

ABSTRACT

The Finite Impulse Response (FIR) filter are a class of digital filter that have finite impulse response and are extensively used in signal processing and communication system in applications like noise reduction, echo cancellation, image enchancement, speech and waveform synthesis etc. As the complexity of implementation grows with the filter order and the precision of computation, real-time realization of these filters with desired level of accuracy becomes a challenging task. So, the implementation of FIR filters on FPGAs is the need of the day because FPGAs can give enhanced speed and allows reconfigurable architectures for realization of FIR filter.

In this dissertation, digital filter has been designed using Kaiser window. This technique is simple conceptually and computationally & it has the adjustable parameter β, which is used to optimize the mainlobe width. The design complexity is much less than that in non-linear optimizations. On the other hand, because in Kaiser window design the stop band attenuation is determined by the window, direct control over the stop-band attenuation can be achieved. The advantages of the Kaiser window compared to the compared to the fixed windows are their near optimality and flexibility.

The direct form structure has been used in designing of proposed filter as this approach gives a better performance than common structures in terms of speed of operation, cost and power consumption. The concept of pipelining has been incorporated that results in reducing the delay of the FIR filter, thereby enhancing the speed and reducing the power dissipation as compared to the non-pipelined techniques.

The design of non-pipelined and pipelined FIR filter using both the encoding schemes – Radix-4 and Radix-8 has been carried out via Hardware Description Language. Simulation and synthesis for FPGAs are accomplished on XILINX ISE Software (Xilinx ISE 9.2i version) for Spartan 3E series FPGA (Field Programmable Gate Array), target device (XC3S1600E) (Speed Grade-5).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ADC	Analog-to-Digital Converter
ASIC	Application Specific Integrated Circuits
ARMA	Auto-Regressive-Moving-Average
ALU	Arithmetic Logical Unit
BIBO	Bounded Input-Bounded Output
CSLA	Carry Select Adder
CLA	Carry Look Ahead Adder
CAD	Computer-Aided Design
CPLD	Complex Programmable Logic Device
DSP	Digital Signal Processor
DAC	Digital-to-Analog Converter
DA	Distributed Arithmetic
DTFT	Discrete Time Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
FDA	Filter Design Analysis
HDL	Hardware Description Languages
IIR	Infinite Impulse Response
IEEE	Institute of Electrical and Electronic Engineers
ISE	Integrated Software Environment
LUT	Look Up Table
LTI	Linear Time Invariant
MBE	Modified Booth Encoding
MA	Moving Average
MXE	ModelSim Xilinx Edition
RCA	Ripple Carry Adder
PLD	Programmable Logic Device
VHSICHDL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration

CHAPTER 1
INTRODUCTION

1.1 GENERAL

A filter is a device or process that removes some unwanted component or feature from a signal. Filtering is a class of signal processing, the defining feature of filter being the complete or partial suppression of some aspect of the signal.

In signal processing, the function of a filter is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range [1]. A filter is an electrical network that alters the amplitude and/or phase characteristics of a signal with respect to frequency. Ideally, a filter will not add new frequencies to the input signal, nor will it change the component frequencies of that signal, but it will change the relative amplitudes of the various frequency components and/or their phase relationships. Filters are often used in electronic systems to emphasize signals in certain frequency ranges and reject signals in other frequency ranges.

Filters are widely employed in signal processing and communication systems in applications such as channel equalization, noise reduction, radar, audio processing, video processing, biomedical signal processing, and analysis of economic and financial data. For example in a radio receiver band-pass filters, or tuners, filters are used to extract the signals from a radio channel. In an audio graphic equalizer the input signal is filtered into a number of sub-band signals and the gain for each sub-band can be varied manually with a set of controls to change the perceived audio sensation. In a Dolby system pre-filtering and post filtering are used to minimize the effect of noise. In hi-fi audio a compensating filter may be included in the preamplifier to compensate for the non-ideal frequency response characteristics of the speakers. Filters are also used to create perceptual audio-visual effects for music, films and in broadcast studios [2].

The primary functions of filters are one of the followings [2]:

a) To confine a signal into a prescribed frequency band as in low-pass, high-pass, and band-pass filters.

b) To decompose a signal into two or more sub-bands as in filter-banks, graphic equalizers, sub-band coders, frequency multiplexers.
c) To modify the frequency spectrum of a signal as in telephone channel equalization and audio graphic equalizers.
d) To model the input-output relationship of a system such as telecommunication channels, human vocal tract, and music synthesizers.

Depending on the form of the filter equation and the structure of implementation, filters may be broadly classified into the following classes:

a) Linear filters versus nonlinear filters.
b) Time-invariant filters versus time-varying filters.
c) Adaptive filters versus non-adaptive filters.
d) Recursive versus non-recursive filters.
e) Direct form, cascade form, parallel form and lattice structures.

1.2 ALTERNATIVE METHODS FOR DESCRIPTION OF FILTERS

Filters can be described using the following time or frequency domain methods [2]:

a) **Time Domain Input-Output Relationship:** A difference equation is used to describe the output of a discrete-time filter in terms of a weighted combination of the input and previous output samples. For example a first-order filter may have the following difference equation

$$y(m) = a\, y(m-1) + x(m) \tag{1.1}$$

where, $x(m)$ is the filter input, $y(m)$ is the filter output and a is the filter coefficient.

b) **Impulse Response:** A filter can be described in terms of its response to an impulse input. Impulse response is useful because:

(i) Any signal can be viewed as the sum of a number of shifted and scaled impulses, hence the response a linear filter to a signal is the sum of the responses to all the impulses that constitute the signal.
(ii) An impulse input contains all frequencies with equal energy, and hence it excites a filter at all frequencies
(iii) Impulse response and frequency response are Fourier transform pairs.

c) **Transfer Function, Poles and Zeros:** The transfer function of a digital filter $H(z)$ is the ratio of the z-transforms of the filter output and input given by

$$H(z) = \frac{Y(z)}{X(z)} \tag{1.2}$$

A useful method of gaining insight into the behaviour of a filter is the pole zero description of a filter. The poles and zeros are the roots of the denominator and numerator of the transfer function respectively.

d) **Frequency Response**: The frequency response of a filter describes how the filter alters the magnitude and phase of the input signal frequencies. The frequency response of a filter can be obtained by taking the Fourier transform of the impulse response of the filter, or by simple substitution of the frequency variable $e^{j\omega}$ for the z variable, $z = e^{j\omega}$ in the z-transfer function as:

$$H\left(z = e^{j\omega}\right) = \frac{Y(e^{j\omega})}{X(e^{j\omega})} \tag{1.3}$$

The frequency response of a filter is a complex variable and can be described in terms of the filter magnitude response and the phase response of the filter.

1.3 TYPES OF FILTERS

There are two main kinds of filter, analog and digital.

a) Analog filters
b) Digital filters

1.3.1 ANALOG FILTER

An analog filter has an analog signal at both its input $x(t)$ and its output $y(t)$. Both $x(t)$ and $y(t)$ are functions of a continuous variable time (t) and can have an infinite number of values. An analog filter uses analog electronic circuits made up from components such as resistors, capacitors and op amps to produce the required filtering effect. Such filter circuits are widely used in applications such as noise reduction, video signal enhancement, graphic equalizers in hi-fi systems, and many other areas. At all stages, the signal being filtered is an electrical

voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

Advantages:

a) Simple and consolidated methodologies of plan
b) Fast and simple realization

Disadvantages:

a) Little stable and sensitive to temperature variations
b) Expensive to realize in large amount

1.3.2 DIGITAL FILTER

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general purpose computer such as a PC, or a specialised DSP (digital signal processor) chip. Digital filters are used in a wide variety of signal processing applications, such as spectrum analysis, digital image processing, and pattern recognition. Digital filters eliminate a number of problems associated with their classical analog counterparts and thus are preferably used in place of analog filters.

The analog input signal must first be sampled and digitized using an ADC (analog to digital converter). The resulting binary numbers, representing successive sampled values of the input signal, are transferred to the processor, which carries out numerical calculations on them. Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively simple and compact in comparison with the equivalent analog circuitry [3].

1.4 BASICS OF DIGITAL FILTER

A digital filter is a system that performs mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of that signal. This is in contrast to the other major type of electronic filter, the analog filter, which is an electronics circuit operating on continuous time analog signals. An analog signal may be processed by digital filter by first being digitized and represented as a sequence of numbers, then manipulated mathematically, and then reconstructed as a new analog signal. In an analog filter, the input signal is directly

manipulated by the circuit [4].

Digital filters are very important part of DSP. In fact, their extraordinary performance is one of the key reasons that DSP has become so popular. The filters have two uses: signal *separation* and signal *restoration*. Signal separation is needed when a signal has been contaminated with interference, noise, or other signals. For example, imagine a device for measuring the electrical activity of a baby's heart while still in the womb. The raw signal will likely be corrupted by the breathing and heartbeat of the mother. A filter might be used to separate these signals so that they can be individually analyzed. Signal restoration is used when a signal has been distorted in some way. For example, an audio recording made with poor equipment may be filtered to better represent the sound as it actually occurred. Another example is the deblurring of an image acquired with an improperly focused lens, or a shaky camera [2].

A digital system system usually consists of an analog to digital converter to sample the input signal followed by a microprocessor and some peripheral components such as memory to store data and filter coefficients etc. Finally a digital to analog converter to complete the output stage. Program instructions (software) running on the microprocessors implement the digital filter by performing the necessary mathematical operations on the numbers receives from the ADC. In some high performance application, an ASIC or FPGA is used instead of a general purpose microprocessor, or a specialized DSP with specific parallel architecture for expediting operations such as filtering [4].

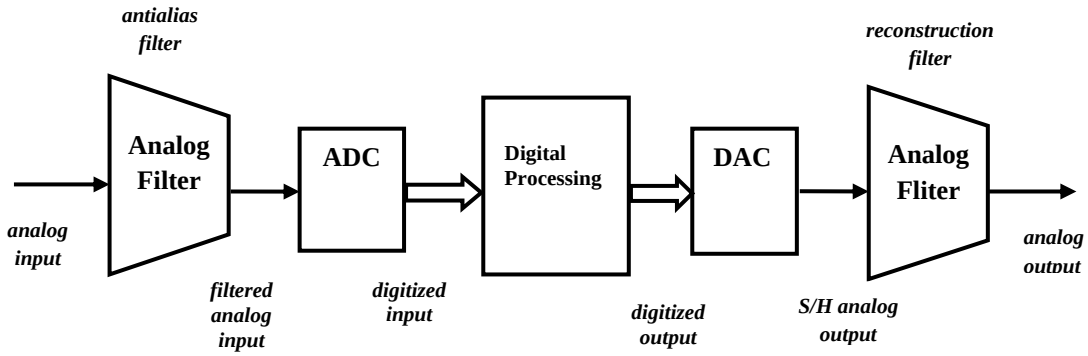


Figure 1.1: Block Diagram of Digital Filter.

Digital filters may be more expensive than an equivalent analog filter due to their increased complexity, but they make practical many designs that are impractical or impossible as analog filters. Since digital filters use a sampling process and discrete-time

processing, they experience latency (the difference in time between the input and the response), which is almost irrelevant in analog filters. The cut-off frequency of the pass-band is a frequency at which the transition of the pass-band to the transition region occurs. The cut-off frequency of the stop-band is a frequency at which the transition of the transition region to the stop-band occurs.

Advantages

The following list gives some of the main advantages of digital over analog filters [3].

a) A digital filter is programmable, i.e. its operation is determined by a program stored in the processor's memory. This means the digital filter can easily be changed without affecting the circuitry (hardware). An analog filter can only be changed by redesigning the filter circuit.
b) Digital filters are easily designed, tested and implemented on a general purpose computer or workstation.
c) The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems, and so are extremely stable with respect to both time and temperature.
d) Unlike their analog counterparts, digital filters can handle low frequency signals accurately. As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF (radio frequency) domain, which in the past was the exclusive preserve of analog technology.
e) Digital filters are very much more versatile in their ability to process signals in a variety of ways including the ability of some types of digital filter to adapt to changes in the characteristics of the signal.
f) Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively simple and compact in comparison with the equivalent analog circuitry.

1.5 COMPARISON OF DIGITAL FILTER AND ANALOG FILTER

Table 1.1: Comparison of Digital and Analog Filter.

Digital filters	Analog filters
Linear phase (FIR) filters	Non-linear phase
High accuracy	Less accuracy component tolerances
No drift due to component variations	Drift due to component variation
Flexible, adaptive filtering possible	Adaptive filtering difficult
Easy to stimulate and design	Difficult to stimulate and design
Computation must be compiled in sampling period-limits real time operation	Analog filters required at higher frequencies and for anti aliasing filters
Requires high performances ADC, DAC and DSP	No ADC, DAC or DSP required

1.6 LINEAR TIME-INVARIANT DIGITAL FILTERS

Linear time-invariant (LTI) [2] filters are a class of filters whose output is a linear combination of the input signal samples and whose coefficients do not vary with time. The *linear* property entails that the filter response to a weighted sum of a number of signals, is the weighted sum of the filter responses to the individual signals. This is the *principle of superposition*. The term *time invariant* implies that the filter coefficients and hence its frequency response is fixed and does not vary with time.

In the time domain the input-output relationship of a discrete-time linear filter is given by the following linear difference equation:

$$y(m) = \sum_{k=1}^{N} a_k \, y(m-k) + \sum_{k=0}^{M} b_k \, x(m-k) \qquad (1.4)$$

where, $\{a_k , b_k\}$ are the filter coefficients, and the output $y(m)$ is a linear combination of the previous *N* output samples $[y(m-1), \ldots\ldots, y(m-N)]$, the present input $x(m)$ sample and the previous *M* input samples $[x(m-1), \ldots\ldots, x(m-M)]$. The characteristic of a filter is completely determined by its coefficients $\{a_k , b_k\}$.

For a time-invariant filter the coefficients $\{a_k , b_k\}$ are constants calculated to obtain

a specified frequency response. The filter transfer function, obtained by taking the *z*-transform of the difference equation (1.4), is given by

$$H(z) = \frac{\sum_{k=0}^{M} b_k\, z^{-k}}{1 - \sum_{k=1}^{N} a_k z^{-k}} \tag{1.5}$$

The frequency response of this filter can be obtained from equation (1.5) by substituting the frequency variable $e^{j\omega}$ for the *z* variable, $z = e^{j\omega}$, as

$$H\left(z = e^{j\omega}\right) = \frac{\sum_{k=0}^{M} b_k\, e^{-j\omega k}}{1 - \sum_{k=1}^{N} a_k\, e^{-j\omega k}} \tag{1.6}$$

Since from Fourier transform a signal is a weighted combination of a number of sine waves, it follows from superposition principle, that in frequency domain linear filtering can be viewed as linear combination of the frequency constituents of the input multiplied by the frequency response of the signal.

1.7 TYPES OF DIGITAL FILTER

Filters can be classified in several different groups, depending on what criteria are used for classification. The two major types of digital filters are finite impulse response digital filters (FIR filters) and infinite impulse response digital filters (IIR).

Both types have some advantages and disadvantages that should be carefully considered when designing a filter. Besides, it is necessary to take into account all fundamental characteristics of a signal to be filtered as these are very important when deciding which filter to use. In most cases, it is only one characteristic that really matters and it is whether it is necessary that filter has linear phase characteristic or not.

1.7.1 NON-RECURSIVE OR FINITE IMPULSE RESPONSE (FIR) FILTERS:

FIR, filters are one of the primary types of filters used in Digital Signal Processing. FIR filters are said to be finite because they do not have any feedback. Therefore, if we send an impulse through the system (a single spike) then the output will invariably become zero as soon as the impulse runs through the filter.

A non-recursive filter [2] has no feedback and its input-output relation is given by

$$y(n) = \sum_{k=0}^{M} p(k)\, x(n-k) \tag{1.7}$$

The output $y(n)$ of a non-recursive filter is a function only of the input signal $x(n)$. The response of such a filter to an impulse consists of a finite sequence of M+1 samples, where M is the filter order. Hence, the filter is known as a *Finite-Duration Impulse Response* (FIR) filter. Other names for a non-recursive filter include all-zero filter, feed-forward filter or moving average (MA) filter a term usually used in statistical signal processing literature.

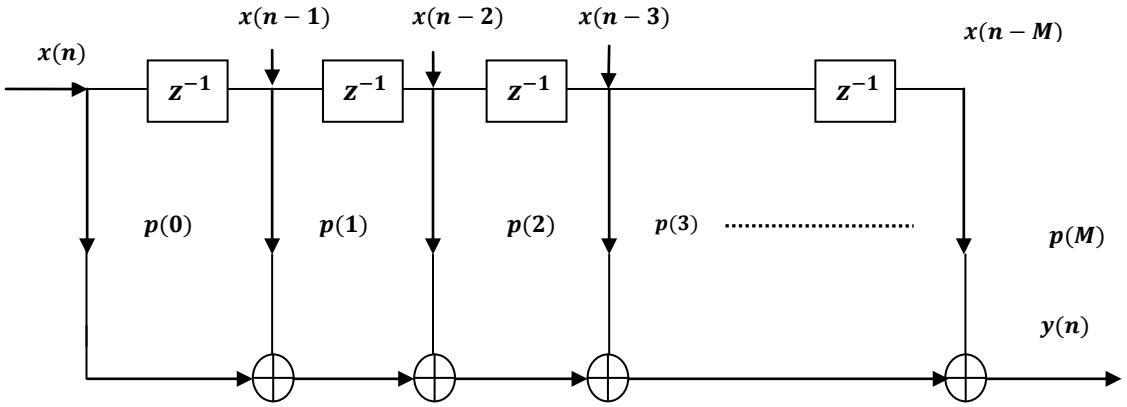


Figure 1.2: Finite Impulse Response Filter Realization.

The basic characteristics of Finite Impulse Response (FIR) filters are:

a) Linear phase characteristic
b) High filter order (more complex circuits)
c) Stability

1.7.2 RECURSIVE OR INFINITE IMPULSE RESPONSE (IIR) FILTERS:

A recursive filter [2] has feedback from output to input, and in general its output is a function of the previous output samples and the present and past input samples as described by the following equation

$$y(n) = \sum_{k=1}^{N} q_k\, y(n-k) + \sum_{k=0}^{M} p_k\, x(n-k) \qquad (1.8)$$

IIR filters have one or more non-zero feedback coefficients. That is, as a result of the feedback term, if the filter has one or more poles, once the filter has been excited with an impulse there is always an output. FIR filters have no non-zero feedback coefficient. That is, the filter has only zeros, and once it has been excited with an impulse, the output is present for only a finite (N) number of computational cycles. Because an IIR filter uses both a feed-forward polynomial (zeros as the roots) and a feedback polynomial (poles as the roots), it has a much sharper transition characteristic for a given filter order.

Like analog filters with poles, an IIR filter usually has non-linear phase characteristics. Also, the feedback loop makes IIR filters difficult to use in adaptive filter applications. Due to its all zero structure, the FIR filter has a linear phase response when the filters coefficients are symmetric, as is the case in most standard filtering applications.

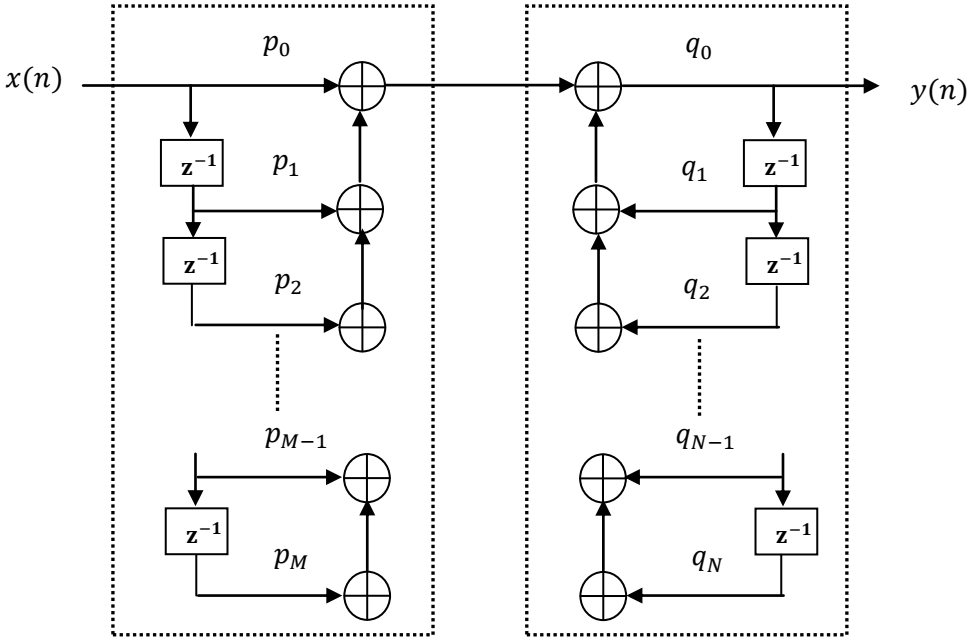


Figure 1.3: Infinite Impulse Response Filter Realization.

A recursive filter is also known as an *Infinite Duration Impulse Response* (IIR) filter. Other names for an IIR filter include feedback filters, pole-zero filters and auto-regressive-moving-average (ARMA) filter a term usually used in statistical signal processing literature. The basic characteristics of Infinite Impulse Response (IIR) are:

a) Non-linear phase characteristic
b) Low filter order (less complex circuits)
c) Resulting digital filter has the potential to become unstable

The main difference between IIR filters and FIR filters is that an IIR filter is more compact in that it can usually achieve a prescribed frequency response with a smaller number of coefficients than an FIR filter. A smaller number of filter coefficients imply less storage requirements and faster calculation and a higher throughput. Therefore, generally IIR filters are more efficient in memory and computational requirements than FIR filters. However, it must be noted that an FIR filter is always stable, whereas an IIR filter can become unstable

(for example if the poles of the IIR filter are outside the unit circle) and care must be taken in design of IIR filters to ensure stability.

1.8 ADVANTAGES OF FIR FILTER OVER IIR FILTER

a) FIR filters are simple to design.
b) They are guaranteed to be bounded input-bounded output (BIBO) stable.
c) FIR filter can be guaranteed to have linear phase. This is a desirable property for many applications such as music and video processing.
d) FIR filters also have a low sensitivity to filter coefficient quantization errors. This is an important property to have when implementing a filter on a DSP processor or on an integrated circuit.

1.9 REAL-WORLD APPLICATIONS OF FIR FILTERS

A few popular applications for FIR filters are listed below:

a) Echo cancellation
    (i) Telecommunications
    (ii) Data communications
    (iii)Wireless communications
b) Multi-path delay compensation
c) Ghosting cancellation in
    (i) HDTV
    (ii) DTV
    (iii)Video processing
d) Speech synthesis
e) Waveform synthesis
f) Filtering
    (i) High-speed modems
g) Image enhancement in
    (i) HDTV
    (ii) DTV

1.10 ORGANIZATION OF DISSERTATION

The dissertation embodies following objectives:

a) To study the window method of calculating filter coefficients for FIR filter design.
b) To study various FIR filter structures used for implementing the filters.
c) To study various synthesis and simulation tools used to implement FIR filter.
d) To study the concept of pipelining technique in designing of FIR filter.
e) To design and implement FIR low-pass on FPGA.

Chapter 2 contains the literature review of papers related to the work. It contains the literature review of papers used in the designing of filters and FPGA implementation of FIR filters. Work related to the designing of low power multipliers and pipelining technique to achieve high speed and low power are also discussed.

Chapter 3 discusses the design stage for digital filter, which includes specification of filter, calculation of filter coefficients, realization of filter structure, finite world length effect and hardware or software implementation of filter. Also discusses window method for calculation of coefficients of FIR filter. The different FIR filter structures are explained in detail and at the end a brief introduction to pipelining technique for low power and high speed are also discussed.

Chapter 4 discusses the different type of adders- Ripple Carry Adder, Carry Select Adder & Carry Look Ahead Adder. Introduction to multiplier and need to design low power multiplier are discussed in brief. Booth Multipliers- Radix-2, Radix-4 and Radix-8 are also discussed in detail.

In Chapter 5, the Simulation and Synthesis tools which are used in implementation of FIR filter are discussed. This chapter also discusses the FPGAs architecture, FPGA Design Flow, SPARTAN 3E FPGA kit specifications and introduction to MATLAB.

Chapter 6 contains simulation and synthesis results of non-pipelined and pipelined FIR filter of given specifications.

Chapter 7 contains conclusion and future scope of the work.

CHAPTER 2
LITERATURE SURVEY

Vaidyanathan [5] addressed the use of architectural transformations techniques for the low power realization of FIR filters on dedicated architectures. They experiment a new encoding for the operators, called Hybrid encoding, which is a compromise between the minimal input dependency offered by the binary encoding and the low switching characteristic of the gray encoding. The results shows that with the use of Hybrid operators in FIR architectures power savings of up to 25% are possible, together with 14% delay improvement, and an area penalty of 28%. They implemented dedicated architectures for FIR filters. Arithmetic operators that operates with a different code, the Hybrid encoding, were experimented in the FIR filter architectures. Performance comparisons for pipelined architectures using Binary and Hybrid operators were investigated and the results showed that despite higher area shown by the architectures with Hybrid operators, these architectures can present less minimum clock period and energy per sample. Thus they explored different values for the size of the groups that work as gray codes in the Hybrid encoding scheme reduction by application of these operators in the FIR architectures.

Lu [7] proposed a method for the design of FIR digital filters with low power consumption. In this method, the digital filter was implemented as a cascade arrangement of low order sections. The first section was designed through optimization, then fixed and a second section was added, which was designed so that the first two sections a cascade satisfy again as far as possible the overall required specifications. The process was repeated until as multi-section filter is obtained that would satisfy the required specifications under the most critical circumstances imposed by the application at hand. In multisection filters of this type, the minimum number of sections required to process the current input signal can be switched in through the use of a simple adaptation mechanism and in this way, the power consumption can be minimized.

Lin et al. [8] limits the search of the prototype filters to the class of filters obtained using Kaiser Window. The design process was reduced to the optimization of a single parameter

i.e. the cut-off frequency ω_c in Kaiser window designs. They further reduce the design complexity by introducing a new objective function $\emptyset_{new}$. It has been shown that the design complexity of the Kaiser window approach is much less than that in non-linear optimizations. On the other hand, because in Kaiser design the stop-band attenuation is determined by the window, direct control can be given over the stop-band attenuation.

Kuncheva et al. [10] discussed the efficient implementation methodologies used in DSP application targeted modern FPGA. Digital filters are very important part of DSP and can be efficiently implemented on FPGA. The modern programmable technology provides possibility to implement digital system with use of dedicated embedded DSP blocks. They synthesised and implemented digital decimation FIR filter in modern FPGA for DSP. The simple, traditional adder-tree approach limited the performance and extensibility of a given filter implementation. By using adder-chain-style implementations, these limitation were lifted and the huge benefits Virtex-4 FPGAs offer are possible. Using MATLAB FDAtool filter order, the required quantization level for the coefficients and their values. Finally by analyzing the design, an efficient way to implement the filter in hardware can be easily found. The use of top-down design methodology decreased the total design time. The high level hardware description language VHDL fully supports arithmetic and binary manipulation that are specific for this design.

Meher et al. [11] presented the design optimization of one and two dimensional fully pipelined computing structures for area-delay-power-efficient implementation of finite-impulse-response (FIR) filter by systolic decomposition of distributed arithmetic (DA)-based inner-product computation. The systolic decomposition scheme is found to offer a flexible choice of the address length of the lookup tables (LUT) for DA-based computation to decide on suitable area time trade-off. It is observed that by using smaller address lengths for DA-based computing units, it is possible to reduce the memory size, but on the other hand that leads to increase of adder complexity and the latency. For efficient DA-based realization of FIR filters of different orders, the flexible linear systolic design is implemented on a Xilinx Virtex-E XCV2000E FPGA using a hybrid combination of Handel-C and parameterizable VHDL cores. Various key performance metrics such as number of slices, maximum usable frequency, dynamic power consumption, energy density, and energy throughput are estimated for different filter orders and address lengths. The proposed FPGA implementation is found

to involve significantly less area-delay complexity compared with the existing DA-based implementations of FIR filter.

Phuong [13] discussed FIR Filter Design with the Window Design method. The design of a FIR Filter starts with its specifications in either discrete-time domain or DTFT frequency domain, or both. In the time domain, the design objective is the impulse response. In the frequency domain, the requirement is on various parameters of the magnitude response. Analyzing all the fixed windows, the only adjustable length (M+1) was of Kaiser window. The Kaiser window has an additional ripple parameter β, enabling the designer to trade-off the transition and ripple.

Kamaraj et al. [14] discussed FIR filters design using HDL languages to enhance the speed of the system. In the whole system if the speed of the individual block is enhanced, the overall speed of the system is enhanced. In order to attain effective utilization hardware is done by applying the pipelining technique. Pipelining is an implementation technique in which multiple instructions are overlapped in execution. The proposed design of this paper is an attempt to optimize the system speed with minimal cost and hardware. The central design concept is to build filters with higher operating frequency without sacrificing the performance of original filters. To enhance system speed and reducing implementation complexity, a lot of work has been done in the process of achieving digital signal processing by use of the FPGA. The paper describes the design of Third order low pass FIR filter with pipelined architecture. The design synthesis is done using Xilinx ISE 12.1 and implemented in Spartan-3E FPGA. By pipelining the delay of FIR filters can be reduced. Pipelined technique may reduce delay and enhances speed as compared to non-pipelined technique.

Jieshan [15] analyzed the basic structure and hardware characteristics of the FIR digital filter, and then a designed method of the FIR filter on the basis of the FIR filter structure. It is a method that is based on FPGA, draws the coefficient by MATLAB and adopts the pipeline to implete the FIR digital filter. The paper focused on the introduction of the overall framework of the FIR digital filter adopting the finite state machine as well as the principle of each module of the design. This paper mainly introduced the design and simulation of FIR filter, which is mainly based on FPGA, Quartus II and MATLAB. The use of these softwares significantly shortens the R&D periods. It is able to greatly improve the speed of the filter by use of pipelining structure. In the practical application, it is easier to achieve other types of

filter design as long as one modify the parameters of the filter, including the width of the input data, the coefficients and so on thereby realizing that the project use less system sources. It can give some reference for the design in the industry.

Kaur et al. [16] proposed the design of FIR & IIR Filters using HDL languages in order enhance the speed of the system. In the whole system if the speed of the individual block is enhanced the overall speed of the system is enhanced. Digital computer arithmetic is an aspect of logic design with the objective of developing appropriate algorithms in order to attain an effective utilization of the available hardware. Since ultimately, speed, power and chip area are the most often used measures of the efficiency of an algorithm, there has a strong link between the algorithms and technology applied for its implementation. Here it is done by applying the technique pipelining. The comparative analysis of pipelined & non-pipelined FIR and IIR filters was performed by using different FPGA's. The results reveal that the implemented filters turn in a consistent quality of output.

Haibatpue et al. [18] designed 4 * 4 bit multipliers, Braun array multiplier, CSA multiplier, and proposed, Vedic multiplier. The multiplier circuits were designed using DSCH2 VLSI CAD tools and their layouts are generated by Microwind 3 VLSI CAD tools. The output parameters such as propagation delay, total chip area, throughput, latency and power dissipation were calculated by using BSIM4 model in Microwind. The simulated results of the three multipliers were compared and it was found that the proposed Vedic multiplier circuit gives better performance.

Aparna et al. [19] presented an area efficient implementation of a high performance parallel multiplier using Radix-4 Booth multiplier with 3:2 compressors and Radix-8 Booth multiplier with 4:2 compressors. The design is structured for m × n multiplication where m and n can reach up to 126 bits and Carry Look Ahead adder has been used as the final adder to enhance the speed of operation. The performance improvement of the proposed multipliers is validated by implementing a higher order FIR filter. Both the designs were implemented on Spartan 3 FPGA. The use of Radix- 8 Booth multiplier with 4:2 compressors for a higher order FIR filter showed a dramatic speed improvement than that using Radix-4 Booth multiplier with 3:2 compressors.

Joshi et al. [20] introduced a brief overview of the basic structure and hardware characteristics of the Finite Impulse Response (FIR) digital filter. FIR filter has been

designed efficiently using MATLAB and implemented on Field Programmable Gate Array (FPGA) platform. MATLAB FDATool has been used to determine filter coefficients and 4th order 32 bit filter has been prototyped. By using these tools time required to get desired results has become less.

Nekoei [21] presented realization of digital FIR filters on field programmable gate array devices Two common architectures called direct and transposed architectures were employed for implementing FIR filters on a Xilinx SPARTAN2-XC2S50-5I-tq144 FPGA using Verilog hardware description language codes.

Jiang et al. [22] proposed a structure characteristics and the basic principles of the finite impulse response (FIR) digital filter, and gives an efficient FIR filter design based on FPGA. They used MATLAB FDATool to determine filter coefficients, and designed a 16-order constant coefficient FIR filter by VHDL language. The simulation was carried out on Quartus-2 and the results meet performance requirements.

Rashidi et al. [23] addressed the use of low power serial multiplier and serial adder, combinational booth multiplier, shift/add multipliers and folding transformation in linear phase architecture for reducing the dynamic power consumption of a digital Finite Impulse Response (FIR). The proposed FIR filters were synthesized implemented using Xilinx ISE Virtex IV FPGA and power is analyzed using Xilinx XPower analyzer. The minimum power achieved is 110 mw in FIR filter based on shift/add multiplier in 100MHZ to 8 taps and 8 bits inputs and 8bits coefficients.

Pal et al. [24] describes the implementation of highly efficient multiplierless serial and parallel distributed arithmetic algorithm for FIR filters. Distributed Arithmetic (DA) had been used to implement a bit-serial scheme of a general symmetric version of an FIR filter due to its high stability and linearity by taking optimal advantage of the look-up table (LUT) based structure of FPGAs. The performance of the bit-serial and bit-parallel DA technique for FIR filter design is analyzed and the results are compared to the conventional FIR filter design techniques. The proposed algorithm has been synthesized with Xilinx ISE 10.1i and implemented as a target device of Spartan3E FPGA.

Soni et al. [25] proposed that the Exponential window provides better side-lobe roll-off ratio than Kaiser Window which is very useful for some applications such as beam forming, filter

design, and speech processing. Besides this a design of digital non-recursive Finite Impulse Response (FIR) filter by using Exponential window was proposed. The far-end stop-band attenuation is most significant parameter when the signal to be filtered has great concentration of spectral energy. In a sub-band coding, the filter is intended to separate out various frequency bands for independent processing. In case of speech, e.g. the far-end rejection of the energy in the stop-band should be more so that the energy leakage from one band to another is minimum. Therefore, the filter should be designed in such a way so that it can provide better far-end stop-band attenuation (amplitude of last ripple in stop-band). Digital FIR filter designed by Kaiser window has a better far-end stop-band attenuation than filter designed by the other previously well known adjustable windows such as Dolph-Chebyshev and Saramaki, which are special cases of Ultraspherical windows, but obtaining a digital filter which performs higher far-end stop-band attenuation than Kaiser window will be useful. In this paper, the design of non-recursive digital FIR filter has been proposed by using Exponential window. It provides better far-end stop-band attenuation than filter designed by well known Kaiser window, which is the advantage of filter designed by Exponential window over filter designed by Kaiser window.

Yunlong et al. [26] They proposed a simple method for design an FIR filter as compare to other existed methods. This method is the simplest except rectangular window method. Filter transition bandwidth is smaller than $4.65/N$ for filter order N. For the same filter specifications filter order obtained by using the new method is much smaller than by using Kaiser window if minimum stop-band attenuation is in the range of 39.5 dB to 48.5dB and corresponding maximum pass-band ripple is from 0.35 dB to 0.18 dB.

Li [27] They used distributed algorithm and its several structures, an implementation method of 60-order FIR filter based on FPGA is presented, which converts multiplication to look-up table structure, and implement multiplication operation. They used FPGA as the hardware platform. This filter system has a good performance, the filter speed is higher and the resource occupation is fewer.

Sudhakar et al. [28] presented the realization of an area efficient architectures using Distributed Arithmetic (DA) for implementation of Finite Impulse Response (FIR) filter. The performance of the bit-serial and bit parallel DA along with pipelining architecture with different quantized versions are analyzed for FIR filter design. Pipelined DA architecture has

achieved double the maximum frequency of operation when compared to their non-pipelined implementations with an increase in hardware. They also analysed that the filters generated using 8 Bit fixed point implementation requires smaller area usage compared to 16 fixed point implementation at the cost of imprecision. The proposed implementations are synthesized with Xilinx ISE 13.2i.

Rajput et al. [29] presented the design and implementation of signed unsigned Modified Booth Encoding (SUMBE) multiplier. The Modified Booth Encoding (MBE) multiplier and the Baugh-Woolley multiplier perform multiplication operation on signed numbers only. The array multiplier and Braun array multipliers perform multiplication operation on unsigned numbers only. Thus, the requirement of the modern computer system is a dedicated and very high speed unique multiplier unit for signed and unsigned numbers. The modified Booth Encoder circuit generates half the partial products in parallel. By extending sign bit of the operands and generating an additional partial product the SUMBE multiplier is obtained. The Carry Save Adder (CSA) tree and the final Carry Look Ahead (CLA) adder used to speed up the multiplier operation. Since signed and unsigned multiplication operation is performed by the same multiplier unit the required hardware and the chip area reduces and this in turn reduces power dissipation and cost of a system.

CHAPATER 3
FIR FILTER DESIGN

3.1 INTRODUCTION

The design of a digital filter involves following five steps [6].

a) **Filter Specification:** This may include stating the type of filter, for example low-pass filter, the desired amplitude and/or phase responses and the tolerances, the sampling frequency, the word length of the input data.

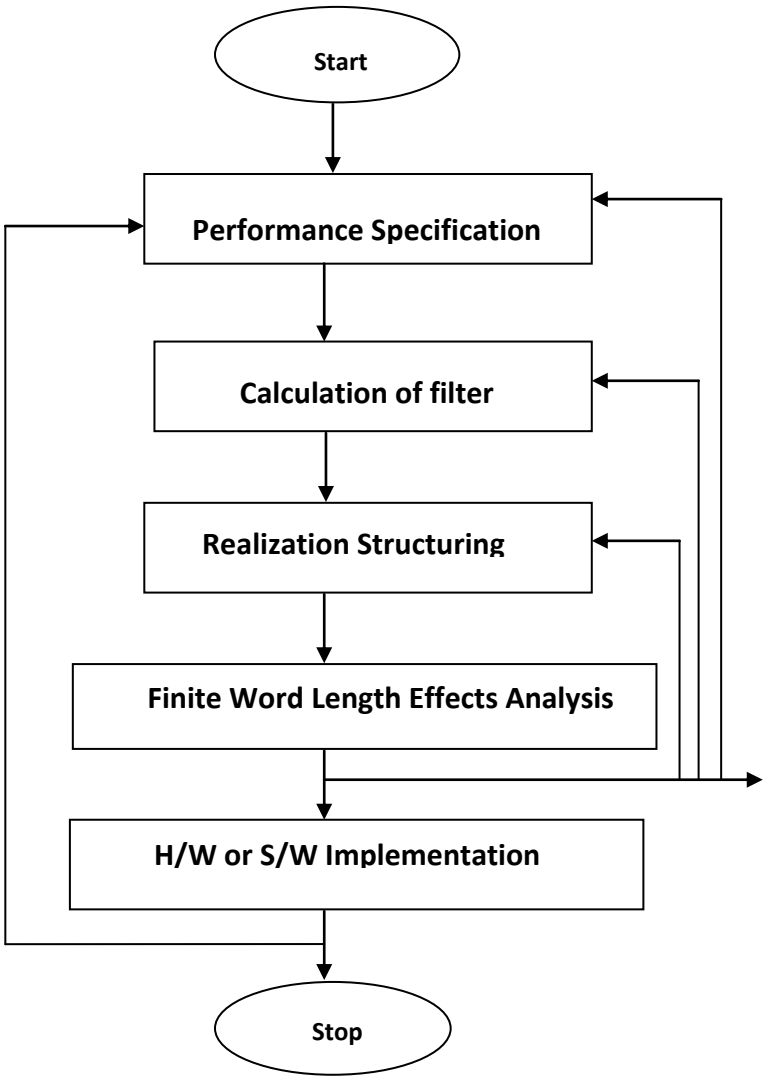


Figure 3.1: Flow Diagram of Digital Filter Design.

b) **Filter Coefficient Calculation:** The coefficient of a transfer function $h(z)$ is determined in is this step, which will satisfy the given specifications. The choice of coefficient calculation method will be influenced by several factors and the most important are the critical requirements i.e. specification. The window, optimal and frequency sampling method are the most commonly used.

c) **Realization:** This involves converting the transfer function into a suitable filter network or structure.

d) **Analysis of Finite Word Length Effects:** The analysis of the effect of quantizing the filter coefficients and input data as well as the effect of carrying out the filtering operation using fixed wordlengths on the filter performance is carried out in this step.

e) **Implementation:** This involves producing the software code and/or hardware and performing the actual filtering.

3.2 FIR FILTER SPECIFICATIONS

The requirement specifications includes

a) Signal characteristics.

b) The characteristics of the filter.

c) The manner of implementation.

d) Other design constraints (cost).

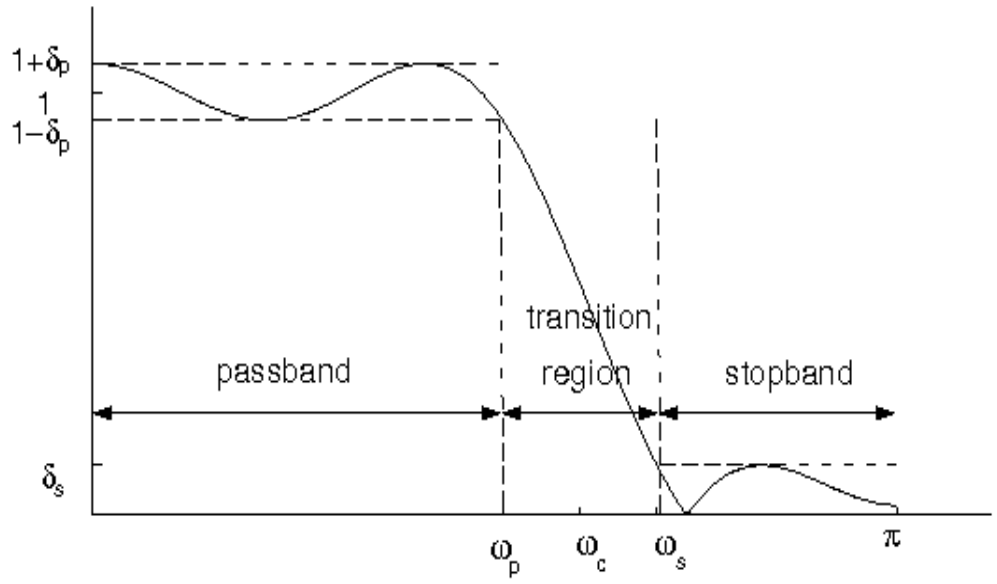


Figure 3.2: Magnitude Frequency Response Specifications for a Low-pass Filter.

All the above requirements are application dependent. The characteristics of digital filters are often in specified in the frequency domain. For frequency selective filters, such as low-pass and band-pass filters, the specifications are often in the form of tolerance [4].

In the pass-band, the magnitude response has a peak deviation of δ_p and in the stop-band, it as a maximum deviation of δ_s. The difference between ω_s and ω_p gives the transition width of the filter and transition band determines how sharp the filter is. The magnitude response decreases monotonically from the pass-band to stop-band in this region. The following are the key parameters of interest:

(i) δ_p Peak pass-band deviation (or ripples).

(ii) δ_s Stop-band deviation.

(iii)ω_s Stop-band edge frequency.

(iv)ω_p Pass-band edge frequency.

(v) F_s Sampling frequency.

Thus the minimum stop-band attenuation, A_s And the peak pass-band ripple, A_p, in decibels are given as

$$A_s \text{ (stop-band attenuation)} = -20 \log_{10} \delta_s$$

$$A_p \text{ (pass-band ripple)} = -20 \log_{10}(1 + \delta_p)$$

Another important parameter is the filter length, *n*, which defines the number of filter.

3.3 FIR COEFFICIENT CALCULATION METHODS

The objective of most FIR coefficient calculation methods is to obtain values of $h(n)$ such that the resulting filter meets the design specifications, such as amplitude frequency response and throughput requirements. The most commonly method to obtain $h(n)$ are the window, optimal and frequency sampling method [9]. All three can lead to linear phase FIR filters.

3.3.1 WINDOW METHOD

In this method, the frequency response of a filter $H_D(\omega)$, the corresponding impulse, $h_D(n)$, are related by the inverse Fourier transform [9] :

$$h_D(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_D(\omega)\, e^{j\omega n}\, d\omega \qquad (3.1)$$

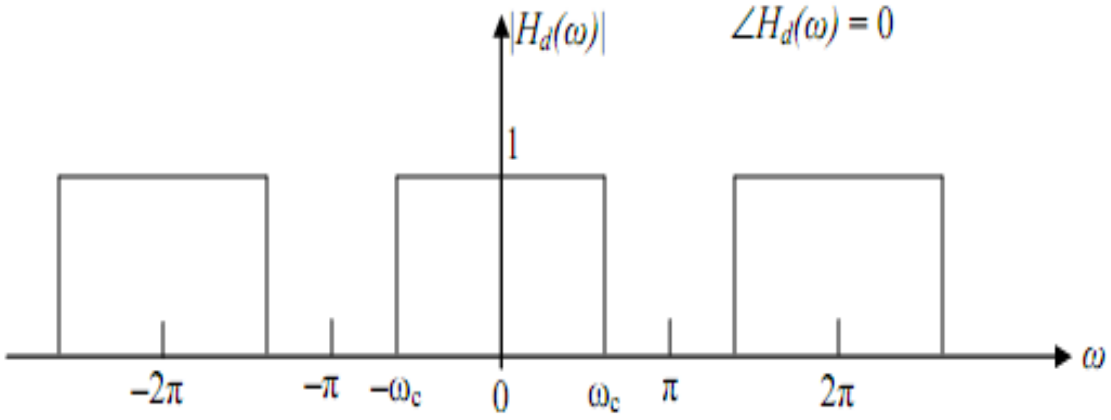


Figure 3.3: Ideal Frequency Response of a Low-pass Filter.

If we know $H_D(\omega)$, we can obtain $h_D(n)$ by evaluating the inverse Fourier Transform by equation. The Figure 3.3 shows the ideal frequency response of low-pass filter where ω_c is the cut-off frequency and the frequency scale is normalised: T = 1. By letting the response go from -ω_c to ω_c The impulse response is given by:

$$h_D(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} 1 * e^{j\omega n}\, d\omega \qquad (3.2)$$

$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} e^{j\omega n}\, d\omega$$

$$= \frac{2f_c\,\mathrm{Sin}(n\omega_c)}{n\omega_c}, \qquad n \neq 0, \qquad -\infty \leq n \leq \infty$$

$$= 2f_c \qquad n = 0$$

The ideal infinite impulse response is truncated by using various windows. When this window is multiplied by the ideal transfer function then all the coefficients within the window are retained and all that are outside the window are discarded.

Table 3.1: Summary of Ideal Impulse Responses [6].

Filter Type	$\boldsymbol{h_D(n), n \neq 0}$	$\boldsymbol{h_D(0)}$
Low Pass	$\frac{2f_c \operatorname{Sin}(n\omega_c)}{n\omega_c}$	$2f_c$
High Pass	$-\frac{2f_c \operatorname{Sin}(n\omega_c)}{n\omega_c}$	$1 - 2f_c$
Band Pass	$\frac{2f_2 \operatorname{Sin}(n\omega_2)}{n\omega_2} - \frac{2f_1 \operatorname{sin}(n\omega_1)}{n\omega_1}$	$2(f_2 - f_1)$

Table3.2: Summary of Important Features of Common Window Function.

Window	**Transition Width**	**Minimum Stop Band Attenuation**
Rectangular	$4\pi/N$	-21 dB
Barlett	$8\pi/N$	-25 dB
Hanning	$8\pi/N$	-44 dB
Hamming	$8\pi/N$	-53 dB
Blackmann	$8\pi/N$	-74 dB
Kaiser	Variable	Variable

a) Rectangular

$$W_R(n) = \begin{cases} 1, & 0 \leq n \leq N-1 \\ 0 & otherwise \end{cases} \quad (3.3)$$

b) Barlett (or triangle)

$$W_B(n) = \begin{cases} \frac{2n}{N-1} & 0 \le n \le (N-1)/2 \\ 2 - \frac{2n}{N-1} & (N-1)/2 \le n \le N-1 \\ 0 & otherwise \end{cases} \quad (3.4)$$

c) Hanning

$$W_{hann}(n) = \begin{cases} \frac{1-\cos\left(\frac{2\pi n}{N-1}\right)}{2} & 0 \le n \le N-1 \\ 0, & otherwise \end{cases} \quad (3.5)$$

d) Hamming

$$W_{hamm}(n) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) & 0 \le n \le N-1 \\ 0, & otherwise \end{cases} \quad (3.6)$$

e) Blackmann

$$W_{black}(n) = \begin{cases} 0.42 - 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right) & 0 \le n \le N-1 \\ 0, & otherwise \end{cases} \quad (3.7)$$

3.3.1.2 KAISER WINDOW

Kaiser window [6] is a well known flexible window and widely used for the spectrum analysis and FIR filter design applications since it achieves a close approximation with the discrete prolate-spheroidal functions that have the maximum energy concentration in the main-lobe.

Because of the difficulty of computing the prolate function, a much simpler approximation using the zeroth-order Modified Bessel function of the first kind was used which resulted in Kaiser window. The Kaiser window function incorporates a ripple parameter, β, which allows the designer to trade-off the transition width against ripple. The Kaiser window is given by

$$w(n) = \begin{cases} I_0\left\{\beta\left[1 - \left(\frac{2n}{N-1}\right)^2\right]^{1/2}\right\} \Big/ I_0(\beta) & -(N-1)/2 \le n \le (N-1)/2 \\ 0 & elsewhere \end{cases} \quad (3.8)$$

where, $I_0(x)$ is the zero-order modified bessel function of the first kind. β controls the way the window function tapers all the edges in the time domain. $I_0(x)$ is normally evaluated using the following power series expansion

$$I_0(x) = 1 + \sum_{k=1}^{L} \left[\frac{(x/2)^k}{k!}\right]^2 \tag{3.9}$$

where, $L < 25$. When $\beta = 0$, the kaiser window corresponds to the rectangular window, and when it is 5.44, the resulting window is very similar, though not identical, to the hamming window. The value of β is determined by the stop-band attenuation requirements and may be estimated from one of the following empirical relationship:

$$\begin{aligned} \beta &= 0 && \text{if } A \leq 21\ dB \\ \beta &= 0.5842(A-21)^{0.4} + 0.078(A-21) && \text{if } 21dB \leq A \leq 50dB \\ \beta &= 0.1102(A-8.7) && \text{if } A \geq 50dB \end{aligned} \tag{3.10}$$

where, $A = -20\ \log_{10}(\delta)$ is the stop-band attenuation, $\delta = \min(\delta_p, \delta_s)$, since the pass-band and stop-band are nearly equal, δ_p is the desired pass-band ripple and δ_s is the desired stop-band ripple. The number of filter coefficients, *n*, is given by

$$N \geq \frac{A-7.95}{14.36\Delta f} \tag{3.11}$$

where, Δf is the normalized transition width. The values of β and *n* are used to compute the coefficients for the Kaiser window *w(n)*.

3.4 FILTER STRUCTURES

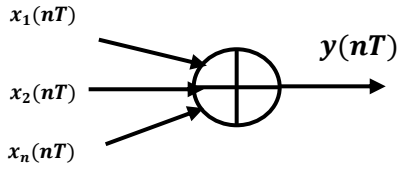
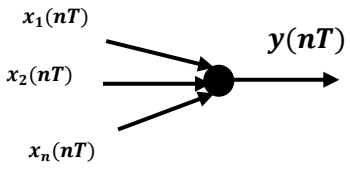
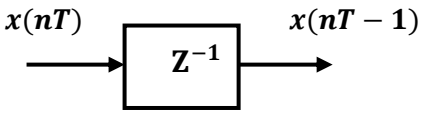
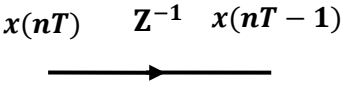
In order to realize a digital filter [12], the given differential equation is broken into small equations. Then for each small equation, a structure using elementary blocks is drawn. Finally all these blocks are interconnected. The filter structures characterizing the difference equations are represented using basic elements such as multipliers, adders and delay elements. These basic block elements and their equivalent signal flow diagrams are as shown in Table 3.3.

This way of presenting the difference equations in the form of block diagram and signal flow diagram makes us easy to write an algorithm, which can be implemented in the

digital computer. A digital filter structure is said to be canonic if the number of delays in the block diagram representation is equal to the order of the transfer function otherwise, it is a non-canonic structure. The different types of structures used to realize the digital filter are as follows:

a) Direct form structure
b) Cascade form structure
c) Parallel form structure

Table 3.3: Block Representation & Signal Flow of Basic Elements [12]**.**

Basic Elements	**Block Representation**	**Signal Flow**
Adder	 $$y(nT) = \sum_{i=0}^{N} x_i(nT)$$	
Time Delay		

The difference equation representing the FIR filter is described as

$$y(n) = \sum_{k=0}^{M-1} b_k\, x(n-k) \tag{3.12}$$

Equation (3.12) shows that the system has length *M* as the limits of summation are from 0 to *M* -1. These limits also indicates that system is causal. Taking z transform of equation (3.12)

$$Y(z) = \sum_{k=0}^{M-1} b_k\, z^{-k} X(z) \tag{3.13}$$

System transfer function is given by

$$H(z) = \frac{Y(z)}{X(z)} \tag{3.14}$$

From equation (3.13) we get

$$H(z) = \frac{Y(z)}{X(z)} = \sum_{k=0}^{M-1} b_k \, z^{-k} \tag{3.15}$$

Equation (3.15) is called as system transfer function of FIR filter.

3.4.1 DIRECT FORM STRUCTURE

The direct-form structure provides a convenient method for the implementation of FIR filters. However, for IIR filters the direct-form implementation is not normally used due to problems with the design and operational stability of direct-form IIR filters. The direct form realization of FIR filter can be obtained by using the equation of linear convolution given by:

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)x(n-k) \tag{3.16}$$

If we consider that there are '*M*' samples the equation becomes

$$y(n) = \sum_{k=0}^{M-1} h(k)x(n-k) \tag{3.17}$$

Expanding equation (3.17) we get

$$y(n) = h(0)x(n) + h(1)x(n-1) + h(2)x(n-2) + \cdots \ldots + h(M-1)x(n-M+1)$$

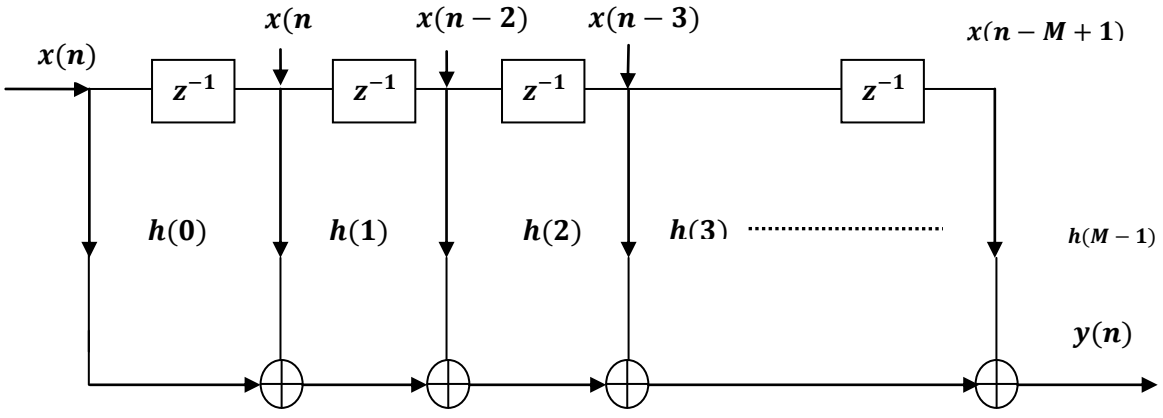


Figure 3.4: Direct Form Realization of FIR Filter.

a) There are *M*-1 delay blocks.
b) Input signal is delayed *M*-1 times so to store this delayed input signal *M*-1 memory locations are required.

c) Equation shows that present input $x(n)$ and past input are multiplied by corresponding sample of $h(n)$ hence output $y(n)$ is weighted linear combination of present input and past inputs.

3.4.2 CASCADE FORM STRUCTURE

Cascade means the number of stages are connected in series. Generally the higher order FIR filter is realised by using a series of different FIR sections. Each section is characterized by the second order transfer function. Then due to cascade connection the total transfer function $h(z)$ will be multiplication of all second order transfer functions. Therefore

$$H(z) = H_1(z)H_2(z)H_3(z)H_4(z) \dots \dots H_k(z) \tag{3.18}$$

Here $H_k(z)$ is the second order transfer and it is given by,

$$H_k(z) = b_{k0} + b_{k1}z^{-1} + b_{k2}z^{-2} \tag{3.19}$$

$$H_k(z) = \frac{Y_k(z)}{X_k(z)}$$

$$\frac{Y_k(z)}{X_k(z)} = b_{k0} + b_{k1}z^{-1} + b_{k2}z^{-2} \tag{3.20}$$

$$Y_k(z) = b_{k0}X_k(z) + b_{k1}z^{-1}X_k(z) + b_{k2}z^{-2}X_k(z) \tag{3.21}$$

Taking inverse z-transform of both sides of equation (3.21) we get

$$Y_k(n) = b_{k0}x_k(n) + b_{k1}x_k(n-1) + b_{k2}x_k(n-2) \tag{3.22}$$

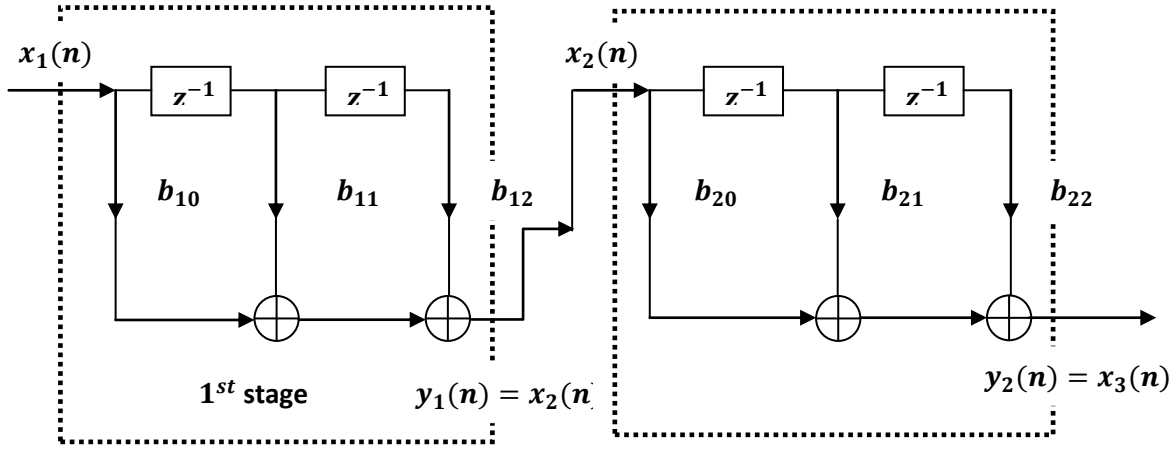


Figure 3.5: Cascade Realization of FIR Filter.

The total structure is obtained by cascading all second order sections.

a) This cascade form is canonic with respect to delay.
b) It has *M*-1 adders and *m* multipliers for $(M-1)^{th}$ order FIR transfer function.

3.4.3 PARALLEL FILTER STRUCTURE

An alternative to the cascade implementation described in the previous section is to express the filter transfer function *H*(*z*), using the partial fraction method, in a parallel form as parallel sum of a number of second order and first order terms as

$$H(z) = K + \sum_{k=1}^{N_1} \frac{e_{0k}+e_{1k}z^{-1}}{1-a_{1k}\,z^{-1}-a_{2k}\,z^{-2}} \tag{3.23}$$

where, in general the filter is assumed to have N_1 complex conjugate poles, N_1 real zeros, and N_2 real poles and *K* is a constant. Figure 3.6 shows a parallel filter structure

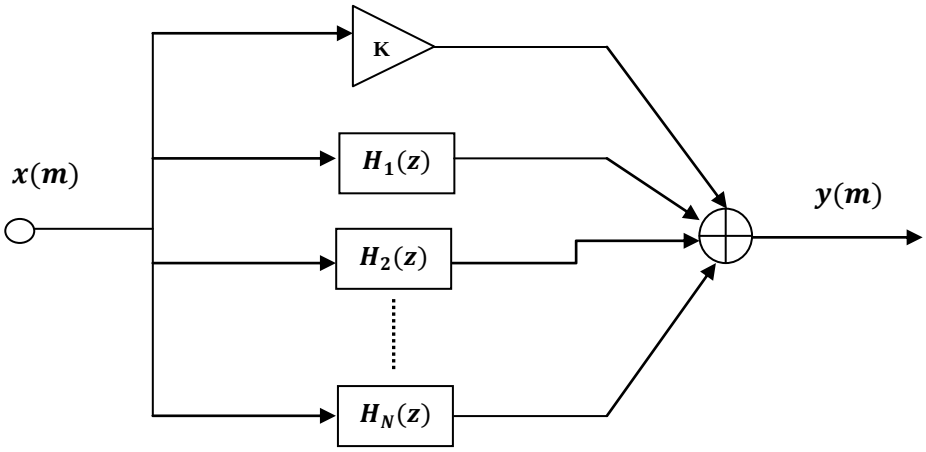


Figure 3.6: Parallel Realisation of FIR Filter.

3.5 CHOICE BETWEEN DIFFERENT STRUCTURES

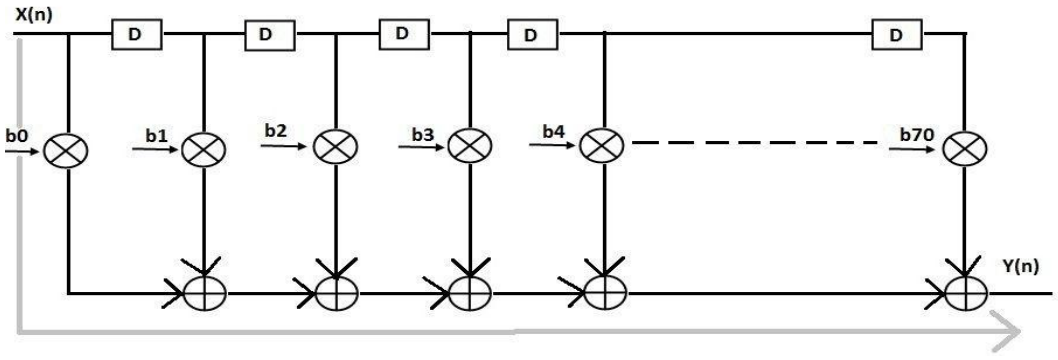
The choice between structures depends on a number of factors and trade-offs which include ease of implementation that is the implied hardware or software complexity, how difficult it is to obtain the impulse response or transfer coefficients and their relative sensitivity to coefficient quantization.

The direct structure is very easy to program and is efficiently implemented by most DSP chips as these have instructions tailored to transversal FIR filtering. It is the most common structure used to realize non-recursive filters and its main attraction is its simplicity,

requiring only a minimum of components and uncomplicated memory accesses for data. The cascade is less sensitive to coefficients errors and quantization noise, but the coefficients require more effort to obtain and the programming is not suited to DSP chips architectures [6].

3.6 PIPELINING

Pipelining is an implementation technique in which multiple instructions are overlapped in execution and results in speed enhancement for the critical path in most of the DSP system, microprocessors etc. It can either increase the clock speed or reduce the power consumption at the same speed in a DSP system. Moreover some real time application requires faster input rates where direct structures cannot be used. In such cases we can use pipelining by introducing latches along the critical data path.



$$T_M + 70T_A$$

Figure 3.7: Seventy-One Tap Non-pipelined FIR Structure.

In pipelining any operation along the critical path is broken into smaller and quicker operation with registers between the levels in order to get a smaller critical path or increase in the operating frequency which leads to higher throughput [14]. The total execution time for each individual instruction is not altered by pipelining. It does not accelerate instruction execution time but it does accelerate program execution time by increasing the number of instruction finished per cycle [16].

The critical path or the minimum time required for processing a new sample is limited by 1 multiply time and 2 add times i.e. If T_M is the time required for multiplication operation and T_A is the time required for addition operation then sample period is given by

$$t_{sample} \geq T_M + 2\,T_A \tag{3.24}$$

Sampling frequency or, throughput is given by**:**

$$f_{sampling} = \frac{1}{T_M + 2T_A} \tag{3.25}$$

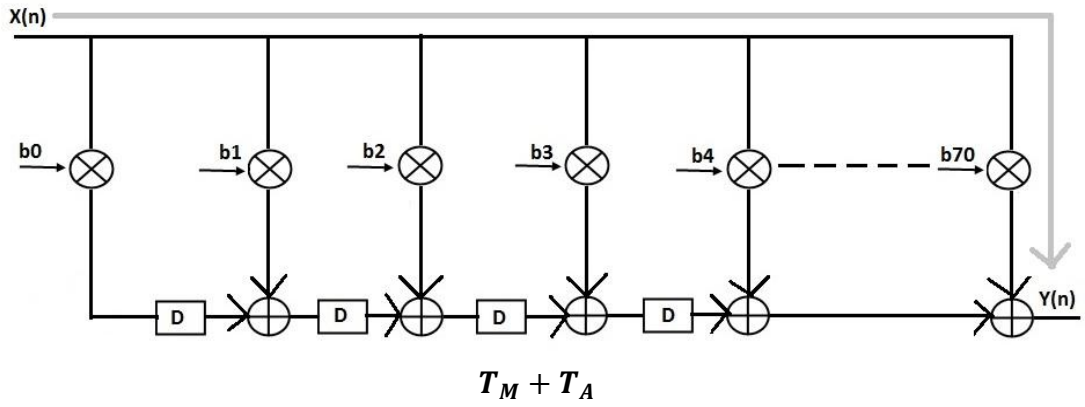


Figure 3.8: Seventy-One Tap Pipelined FIR Structure.

Due to pipelining the critical path has been reduced from $T_M + 70T_A$ to $T_M + T_A$ where, T_M is the time required for multiplication operation and T_A is the time required for addition operation. The following points give the significances of pipelining;

a) The speed of architecture (or the clock period) is limited by the longest path between any two latches or between an input and a latch or between a latch and an output or between the input and the output.

b) This longest path or the "critical path" can be reduced by suitably placing the pipelining latches in the architecture.

CHAPTER 4
ADDERS AND MULTIPLIERS

4.1 INTRODUCTION

Addition [17] is the most common and often used arithmetic operation on microprocessor, digital signal processor, especially digital computers. Also, it serves as a building block for synthesis all other arithmetic operations. Therefore, regarding the efficient implementation of an arithmetic unit, the binary adder structures become a very critical hardware unit.

In electronics, an adder is a digital circuit that performs addition of numbers. In modern computers adders reside in the arithmetic logic unit (ALU) where other operations are performed. Although adders can be constructed for many numerical representations, such as binary-coded decimal or excess-3, the most common adders operate on binary numbers. In cases where two's complement is being used to represent negative numbers it is trivial to modify an adder into an adder / subtractor.

Although many researches dealing with the binary adder structures have been done, the studies based on their comparative performance analysis are only a few. In our dissertation, qualitative evaluations of the classified binary adder architectures are given. Among the huge member of the adders we wrote VHDL (Hardware Description Language) code for Ripple Carry, Carry Select and Carry Look Ahead to emphasize the common performance properties belong to their classes. In the following section, we give a brief description of the studied adder architectures.

The first class consists of the very slow Ripple Carry adder with the smallest area. In the second class, the Carry Skip, Carry Select adders with multiple levels have small area requirements and shortened computation times. From the third class, the Carry Look Ahead adder and from the fourth class, the parallel prefix adder represents the fastest addition schemes with the largest area complexities. For single bit adders, there are two general types.

4.1.1 HALF ADDER

A half adder is a logical circuit that performs an addition operation on two binary digits. The half adder produces a sum and a carry value which are both binary digits.

$$S = A\ xor\ \ B \tag{4.1}$$

$$C = A.B \tag{4.2}$$

Table 4.1: Truth Table of Half Adder.

A	***B***	***S***	***C***
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

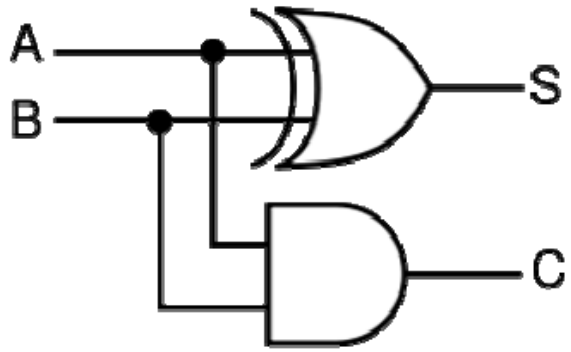


Figure 4.1: Half Adder Circuit Diagram.

4.1.2 FULL ADDER

A full adder is a logical circuit that performs an addition operation on three binary digits. The full adder produces a sum and carries value, which are both binary digits. It can be combined with other full adders or work on its own.

$$S = A\ xor\ B\ xor\ C_{in} \tag{4.3}$$

$$C_{out} = A\ B + \ C_{in}(A\ xor\ B\ ) = \ A\ B + BC_{in} + \ C_{in}\ A \tag{4.4}$$

Table 4.2: Truth Table of Full Adder.

A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

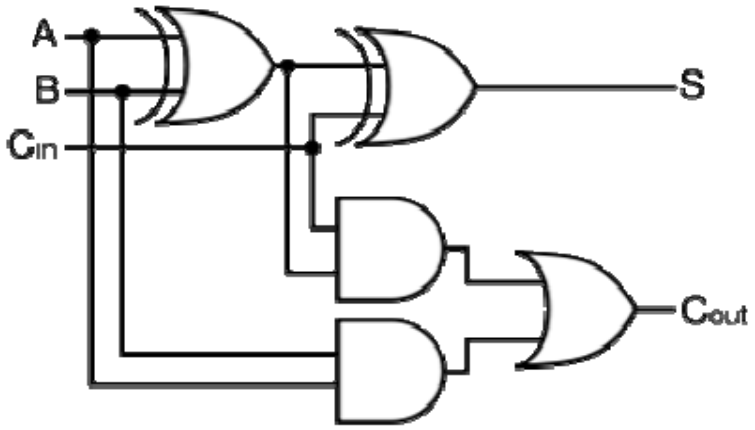


Figure 4.2: Full Adder Circuit Diagram.

4.2 TYPES OF ADDERS

4.2.1 RIPPLE CARRY ADDERS (RCA)

The well known adder architecture, Ripple Carry Adder is composed of cascaded full adders for *n*-bit adder, as shown in Figure 4.3. It is constructed by cascading full adder blocks in series. The carry out of one stage is fed directly to the carry-in of the next stage. For an *n*-bit parallel adder it requires *n* full adders.

a) Not very efficient when large number bit numbers are used.

b) Delay increases linearly with bit length.

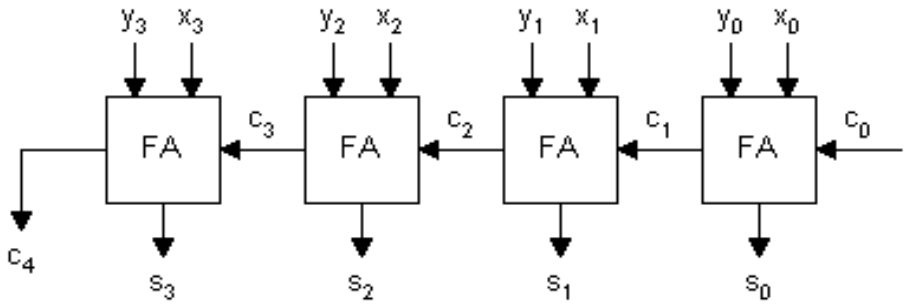


Figure 4.3: Four-Bit Ripple Carry Adder

4.2.2 CARRY SELECT ADDER (CSLA)

In Carry Select Adder scheme, blocks of bits are added in two ways: one assuming a carry-in of 0 and the other with a carry-in of 1. This results in two precomputed sum and carry-out signal pairs**,** later as the block's true carry-in becomes known, the correct signal pairs are selected. Generally multiplexers are used to propagate carries.

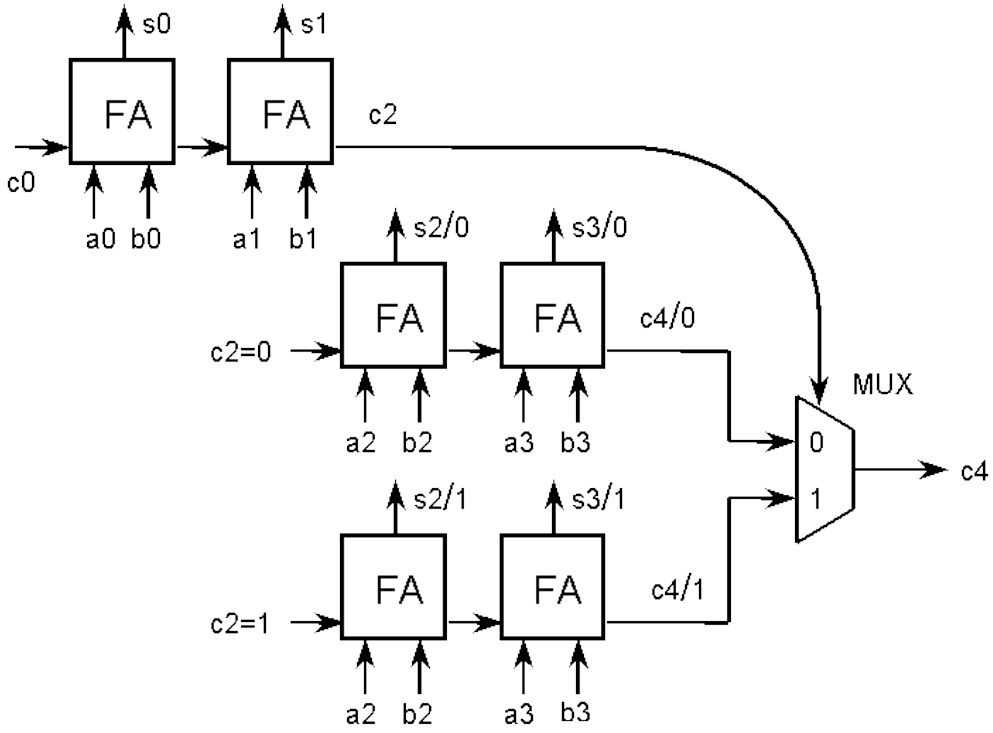


Figure 4.4: Four-Bit Carry Select Adder.

a) Because of multiplexers larger area is required.
b) Have a lesser delay than ripple carry adders (half delay of RCA).
c) Hence we always go for carry select adder while working with smaller number of bits.

4.2.3 CARRY LOOK AHEAD ADDERS (CLA)

Carry Look Ahead Adder produce carries faster due to carry bits generated in parallel by an additional circuitry whenever inputs change. This technique uses carry bypass logic to speed up the carry propagation. Let a_i and b_i be the augends and addend inputs, c_i The carry input, s_i and c_{i+1} , the sum and carry-out to the i_{th} bit position. If the auxiliary functions, p_i and g_i called the propagate and generate signals, the sum output respectively are defined as follows

$$p_i = a_i + b_i \tag{4.5}$$

$$g_i = a_i\, b_i \tag{4.6}$$

$$s_i = a_i\, xor\, b_i\, xor\, c_i \tag{4.7}$$

$$c_{i+1} = g_i + p_i\, c_i \tag{4.8}$$

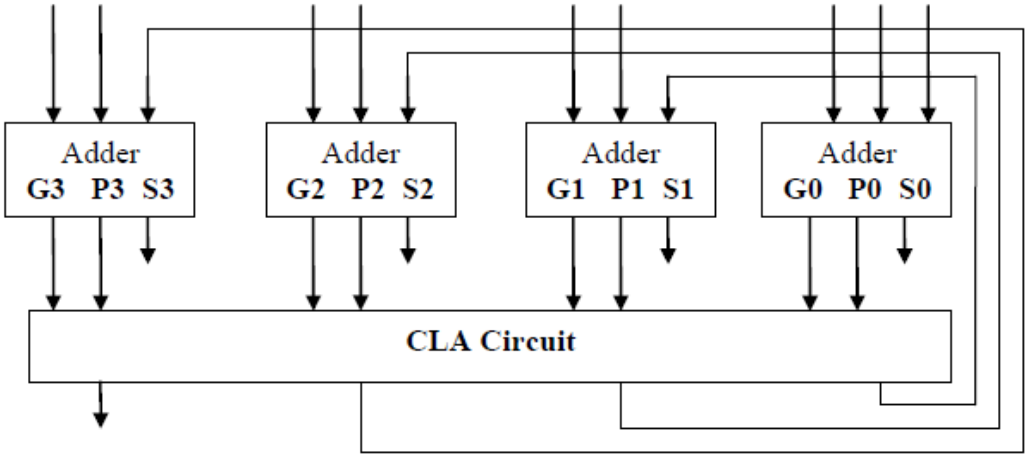


Figure 4.5: Four-Bit Carry Look Ahead Adder.

As we increase the no of bits in the Carry Look Ahead adders, the complexity increases because the no. of gates in the expression c_{i+1} increases. So practically its not desirable to use the traditional CLA shown above because it increase the space required and the power too.

4.3 MULTIPLIERS

Multiplication [18] is a fundamental operation in most signal processing algorithms. Multipliers have large area, long latency and consume considerable power. Therefore low-power multiplier design has been an important part in low power VLSI system design. There has been extensive work on low power multipliers at technology, physical, circuit and logic levels. A system's performance is generally determined by the performance of the multiplier because the multiplier is generally the slowest element in the system. Furthermore, it is generally the most area consuming. Hence, optimizing the speed and area of the multiplier is a major design issue. However, area and speed are usually conflicting constraints so that improving speed results mostly in larger areas. As a result, a whole spectrum of multipliers with different area- speed constraints has been designed with fully parallel.

Multiplier modules are common to many DSP applications. The fastest types of multipliers are parallel multipliers. Among these, the Wallace multiplier is among the fastest. However, they suffer from a bad regularity. Hence, when regularity, high performance and low power are primary concerns, Booth multipliers tend to be the primary choice. Booth multipliers allow the operation on signed operands in two's complement. They derive from array multipliers where, for each bit in a partial product line, an encoding scheme is used to determine if this bit is positive, negative or zero. Radix 2^n multipliers which operate on digits in a parallel fashion instead of bits bring the pipelining to the digit level. These structures are iterative and modular. The pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of the multiplier. The clock speed is only determined by the digit size which is already fixed before the design is implemented. The Modified Booth algorithm achieves a major performance improvement through Radix-4 and Radix-8 encoding scheme.

This chapter discusses an efficient implementation of a high speed parallel multiplier using booth algorithm. Multipliers are designed using both the encoding schemes – Radix–4 and Radix-8 and finally a regular Carry Select Adder has been used to add all the partial products.

4.4 BOOTH MULTIPLIER

Booth multipliers [18] are the parallel multipliers that operate on signed operands in two's complement form and have high performance, low power consumption and does not suffer from bad regularity. Booth multiplier can be used in different modes such as Radix-2, Radix-4, Radix-8 etc. In our dissertation an efficient implementation of high speed parallel multipliers using both the encoding schemes Radix–4 and Radix-8 has been carried out which can be further used in the designing of FIR filter. Radix-4 and Radix-8 reduces the number of partial products to *n*/2 and *n*/3 respectively where *n* is the length of the multiplier. The number of partial products can be further reduced by using higher radices but the disadvantage is that we need to generate more multiples of multiplicand.

4.4.1 BOOTH MULTIPLICATION ALGORITHM (RADIX – 4)

Booth algorithm is a powerful algorithm [19] for signed number multiplication, which treats both positive and negative numbers uniformly. Since a *n*-bit binary number can be interpreted as *n*/2-digit Radix-4 number, a *n*/3-digit Radix-8 number and so on, it can deal with more than one bit of the multiplier in each cycle by using high radix multiplication. One of the solutions realizing high speed multipliers is to enhance parallelism which helps in decreasing the number of subsequent calculation stages. The original version of booth's multiplier (Radix–2) had two drawbacks

a) The number of add / subtract operations became variable and hence became inconvenient while designing parallel multipliers.
b) The algorithm becomes inefficient when there are isolated 1s.

The Radix-4 Modified Booth Algorithm overcomes all these limitations of Radix-2 algorithm. For operands equal to or greater than 16 bits, the modified Radix-4 booth algorithm has been widely used. It is based on encoding the two's complement multiplier in order to reduce the number of partial products to be added to *n*/2. The multiplier, Y in two's complement form can be written as

$$Y = -Y_{n-1}\,2^{n-1} + \sum_i Y_i\,2^i; \qquad 0 \le i \le n-2 \qquad (4.9)$$

The Radix–4 algorithm is as follows:

a) Extend the sign bit 1 position if necessary to ensure that *n* is even.
b) Append a 0 to the right of the LSB of the multiplier.
c) Multiplier bits are grouped into the block of three such that each block overlaps the previous block by one bit. The overlapping is necessary so as to know what was happened in the last block as the MSB of the block act as sign bit.
d) For each group of three bits a partial product is generated according to the encoding table as shown in Table 4.3 which encodes the multiplier bits into [-2, 2].

Table 4.3: Radix-4 Recoding Scheme [19].

Multiplier Bits			**Recoding Operation on Multiplicand, X**
Y_{i+1}	Y_i	Y_{i-1}	
0	0	0	0
0	0	1	+1X
0	1	0	+1X
0	1	1	+2X
1	0	0	-2X
1	0	1	-1X
1	1	0	-1X
1	1	1	0

4.4.2 BOOTH MULTIPLICATION ALGORITHM (RADIX – 8)

Radix-8 booth recoding [19] applies the same algorithm as that of Radix-4, but the difference is here the multipliers bits are grouped into block of four bits and encodes the multiplier bits into [-4, 4] according to the encoding table as shown in Table 4.4. Finally a regular carry select adder has been used to add all the partial products. Radix-8 algorithm reduces the number of partial products to *n*/3, where *n* is the number of multiplier bits. Thus it allows a time gain in the partial products summation.

Table 4.4: Radix-8 Recoding Scheme

Multiplier Bits				**Recoding Operation on Multiplicand, X**
Y_{i+2}	Y_{i+1}	Y_i	Y_{i-1}	
0	0	0	0	0
0	0	0	1	+1X
0	0	1	0	+1X
0	0	1	1	+2X
0	1	0	0	+2X
0	1	0	1	+3X
0	1	1	0	+3X
0	1	1	1	+4X
1	0	0	0	-4X
1	0	0	1	-3X
1	0	1	0	-3X
1	0	1	1	-2X
1	1	0	0	-2X
1	1	0	1	-1X
1	1	1	0	-1X
1	1	1	1	0

Multiply by zero means the multiplicand is multiplied by “0”. Multiply by “1” means the product still remains the same as the multiplicand value. Multiply by “-1” means that the product is the two’s complement form of the number. Multiply by “-2” is to shift left one bit the two’s complement of the multiplicand value and multiply by “2” means just shift left the multiplicand by one place. Multiplying the multiplicand by “3” is equivalent to addition of multiplicand and left shifted multiplicand by one digit. Multiply by “-3” means addition of two’s complement of multiplicand and shift left one bit the two’s compliment of the multiplicand value. Multiply by “4” means shift left the multiplicand by two places. Multiply by “-4” means shift left the two’s complement of multiplicand by two places.

4.5 FLOW CHART

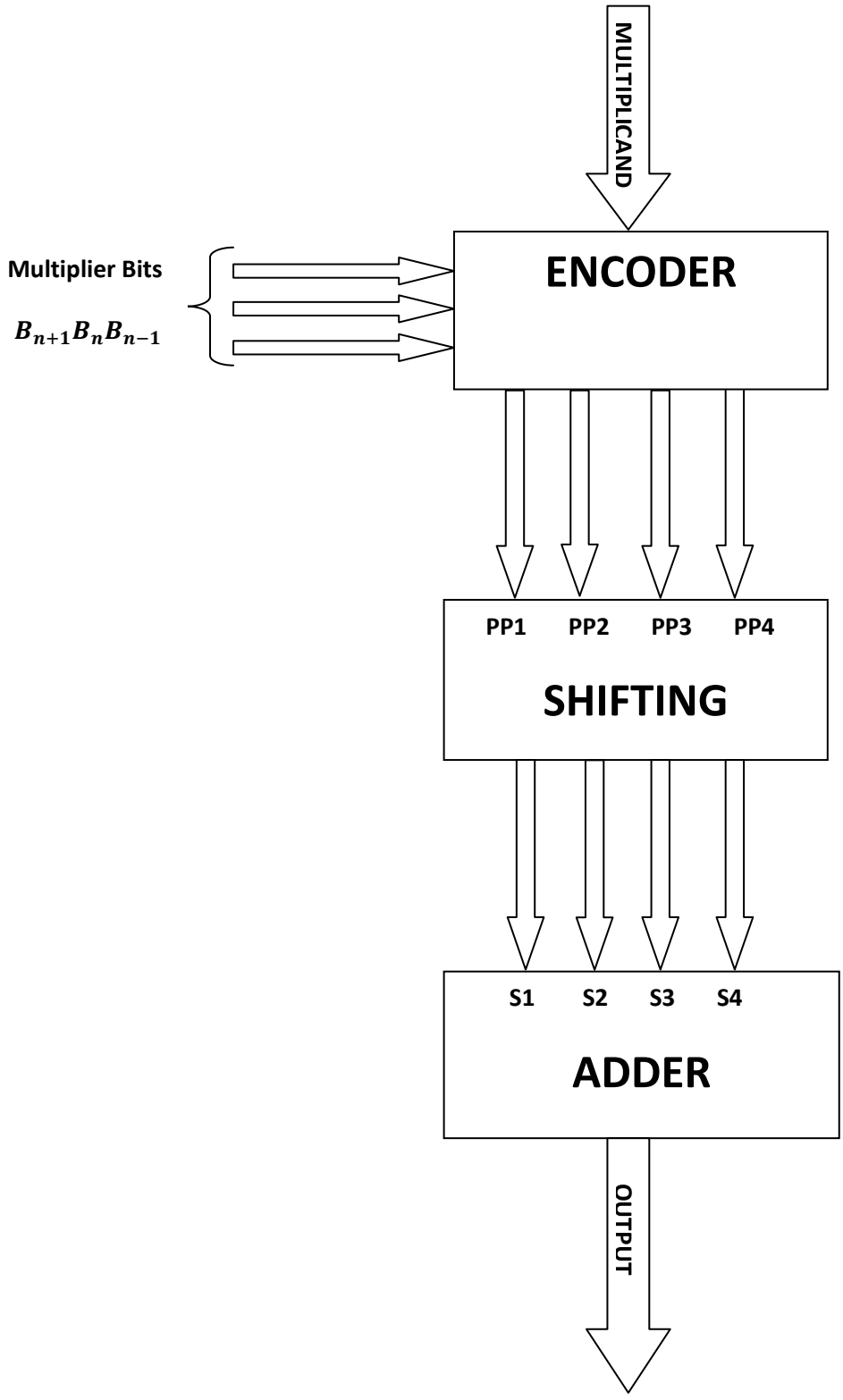


Figure 4.6: Flow Chart of Multiplier Design.

CHAPTER 5
SIMULATION AND SYNTHESIS TOOLS

5.1 INTRODUCTION

The following tools are used for the designing of FIR filter:

a) XILINX ISE web pack 9.2i for design, synthesis and implementation.
b) MODELSIM 6.3f for simulation.
c) MATLAB for filter design.
d) Synopsys for power calculation

5.2 SIMULATION TOOLS

5.2.1 VHDL BASICS

VHDL [30] stands for very high speed integrated circuits (VHSIC) hardware description language (HDL). It is a language for describing digital electronic systems. It was born out of United States government's VHSIC program in 1980 and was adopted as a standard for describing the structure and function of integrated circuits (ICS). Soon after, it was developed and adopted as a standard by the institute of electrical and electronic engineers (IEEE) in the US (IEEE-1076-1987) and in other countries. VHDL continues to evolve. Although new standards have been prepared (VHDL-93) most commercial VHDL tool use 1076-1987 version of VHDL, thus making it most compatible when using different compilation tools. VHDL can be used to design the lowest level (gate level) of a digital system to the highest level (VLSI module). VHDL though being a rigid language with a standard set of rules allows the designer to use different methods of design giving different perspectives to the digital system.

Other than VHDL there are many hardware description languages available in the market for the digital designers such as Verilog, ABEL, PALASM, CUPL, and etc but VHDL and Verilog are the most widely used HDLs. The major difference between hardware description programming languages and others is the integration of time. Timing specifications are used to incorporate propagation delays present in the system. VHDL enables the designer to:

a) Describe the design in its structure, to specify how it is decomposed into sub-designs, and how these sub-designs are interconnected.
b) Specify the function of designs using a familiar, C-like programming language form.
c) Simulate the design before sending it off for fabrication, so that the designer has a chance to rapidly compare alternative approach and test for correctness without the delay and expense of multiple prototyping.

5.2.1.1 TYPES OF REPRESENTATION:

VHDL [30] representation can be seen as text file describing a digital system. The digital system can be represented in different forms such as a behavioural model or a structural model. Most commonly known as levels of abstraction, these levels help the designer to develop complex systems efficiently.

a) Behavioural Model: Behavioural level describes the system the way it behaves instead of a lower abstraction of its connections. Behavioural model describes the relationship between the input and output signals. The description can be a register transfer level (RTL) or algorithmic (set of instruction) or simple boolean equations.

(i) **Register Transfer Level:** RTL typically represents data flow within the systems like data flow between registers. RTL is mostly used for design of combinational logics.

(ii) **Algorithmic Level:** In this method, specific instruction set of statements define the sequence of operations in the system. Algorithmic level is mostly used for design of sequential logics.

b) Structural Model: Structural level describes the systems as gates or component block interconnected to perform the desired operations. Structural level is primarily the graphical representation of the digital system and so it is closer to the actual physical representation of the system.

5.2.1.2 VHDL PROGRAMMING STRUCTURE:

Entity and architecture are the two main basic programming structures in VHDL.

a) **Entity:** The basic building block of a VHDL model is the entity. A digital system in VHDL is modelled as an entity which itself can be composed of other entities. An entity is described as a set of design units:

(i) Entity declaration
(ii) Architecture body
(iii) Package declaration
(iv) Package body
(v) Configuration declaration

b) **Architecture:** Architecture defines what is in our black box that we described using entity. We can use either behavioural or structural models to describe our system in the architecture. In architecture we will have interconnections, processes, components, etc.

5.2.1.3 ADVANTAGES:

VHDL is a powerful and versatile language and offers numerous

a) **Design Methodology:** VHDL supports many different design methodologies (top-down, bottom-up, delay of detail) and is very flexible in its approach to describing hardware.
b) **Technology Independence:** VHDL is independent of any specific technology or process. However, VHDL code can be written and then targeted for many different technologies.
c) **Wide Range of Descriptions:** VHDL can model hardware at various levels of design abstraction. VHDL can describe hardware from the standpoint of a "black box" to the gate level. VHDL also allows for different abstraction-level descriptions of the same component and allows the designer to mix behavioural descriptions with gate level descriptions.
d) **Standard Language:** The use of a standard language allows for easier documentation and the ability to run the same code in a variety of environments. Additionally, communication among designers and among design tools is enhanced by a standard language.
e) **Design Management:** Use of VHDL constructs, such as packages and libraries, allows common elements to be shared among members of a design group.
f) **Flexible Design:** VHDL can be used to model digital hardware as well as many other types of systems, including analog devices.

5.2.2 SYNOPSYS

Synopsys tools can be used to perform power analysis for all the VHDL designs. Generally, the better design has smaller power consumption. On the other hand, improve the power always means sacrificing other design metrics such as performance, area size or NRE cost. Therefore, a designer need to balance these metrics to find the best implementation for the given application and constraints. FIR digital filter has the biggest power consumption because it has a more complex circuit doing DSP computation.

5.3 FILTER DESIGN TOOL

5.3.1 MATLAB

MATLAB [30] is a technical computing environment for high performance numeric computation and visualization. MATLAB integrates numerical analysis, matrix computation, signal processing (via the signal processing toolbox), and graphics into an easy-to-use environment where problems and solutions are expressed just as they are written mathematically, without development much traditional programming. The name MATLAB stands for matrix laboratory.

Key Features

a) High-level language for numerical computation, visualization, and application development.
b) Interactive environment for iterative exploration, design, and problem solving.
c) Mathematical functions for linear algebra, statistics, fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations.
d) Built-in graphics for visualizing data and tools for creating custom plots.
e) Tools for improving code quality and maintainability and maximizing performance.
f) Tools for building applications with custom graphical interfaces.
g) Functions for integrating MATLAB based algorithms with external applications and languages such as C, JAVA, .NET, and Microsoft Excel.

5.3.2 FDA TOOL

The filter design and analysis tool (FDA tool) [31] is a powerful user interface for designing and analyzing filters. FDAtool enables you to quickly design digital FIR or IIR filters by setting filter performance specifications, by importing filters from your MATLAB workspace, or by directly specifying filter coefficients. FDAtool also provides tools for analyzing filters, such as magnitude and phase response plots and pole-zero plots. FVtool (Filter Visualisation Tool), which can be launched from FDAtool, provides a separate window for analyzing filters. The FDAtool can be used as a convenient alternative to the command line filter design functions. FDAtool opens the filter design and analysis tool (FDAtool). It can be used as follows:

a) Design filters
b) Quantize filters
c) Analyze filters
d) Modify existing filter designs
e) Realize simulink models of quantized, direct form, fir filters
f) Perform digital frequency transformations of filters

FDAtool also integrates additional functionality from these other Mathworks products:

a) **DSP System Toolbox:** Adds advanced FIR and IIR design techniques (i.e. Filter transformations, multirate filters) and generates equivalent block for the filter
b) **Embedded Coder:** Generates builds and deploys code for Texas Instruments c6000 processors.
c) **Filter Design HDL Coder:** Generates synthesizable VHDL or Verilog code for fixed-point filters
d) **Simulink:** Generates filters from atomic simulink blocks.

5.3.3 HDL CODER

The filter design HDL coder [32] product adds hardware implementation capability to MATLAB. It enables user to generate efficient, synthesizable, and portable VHDL and Verilog code for fixed point filters that are designed with DSP system toolbox software, for implementation in ASICs or FPGAs. It also automatically creates VHDL and Verilog test benches for quickly simulating, testing, and verifying the generated code.

Key features

a) Generates synthesizable IEEE 1076 compliant VHDL code and IEEE 1364-2001 compliant Verilog code for implementing fixed-point filters in ASICs and FPGAs.
b) Controls the content, optimization, and style of generated code.
c) Provides options for speed verses area trade-offs and architecture exploration, including distributed arithmetic.
d) Generates VHDL and Verilog test benches for quick verification and validation of generated HDL filter code.
e) Generates simulation and synthesis scripts.

The generated VHDL and Verilog code adheres to a clean HDL coding style that enables architects and designers to quickly customize the code if needed. The test bench feature increases confidence in the correctness of the generated code and saves time spent on test bench implementation.

5.4 SYNTHESIS TOOLS

5.4.1 XILINX ISE 9.2I OVERVIEW

WEBPACK ISE [34] design software offers a complete design suite based on the XILINX ISE series software. Individual WEBPACK ISE modules give us the ability to the design environment to our chosen PLDs as well as the preferred design flow. In general, the design flow for FPGAs and CPLDs is identical [22]. The design can also comprise of a mixture of schematic diagrams and embedded HDL symbols. There is also a facility to create state machines.

WEBPACK ISE software offers an easy-to-use GUI to visually create a test pattern. A test bench is then generated and compiled into MXE, along with the design under test. This XILINX has been used for synthesis and implementation of our design. The flow diagram below shows the similarities and differences between CPLD and FPGA software flows. WEBPACK ISE software incorporates a XILINX version of the Modelsim simulator from model technology (a mentor graphics company) referred to as MXE (Modelsim Xilinx Edition). This powerful simulator is capable of simulating functional VHDL before synthesis, or simulating after the implementation process for timing verification.

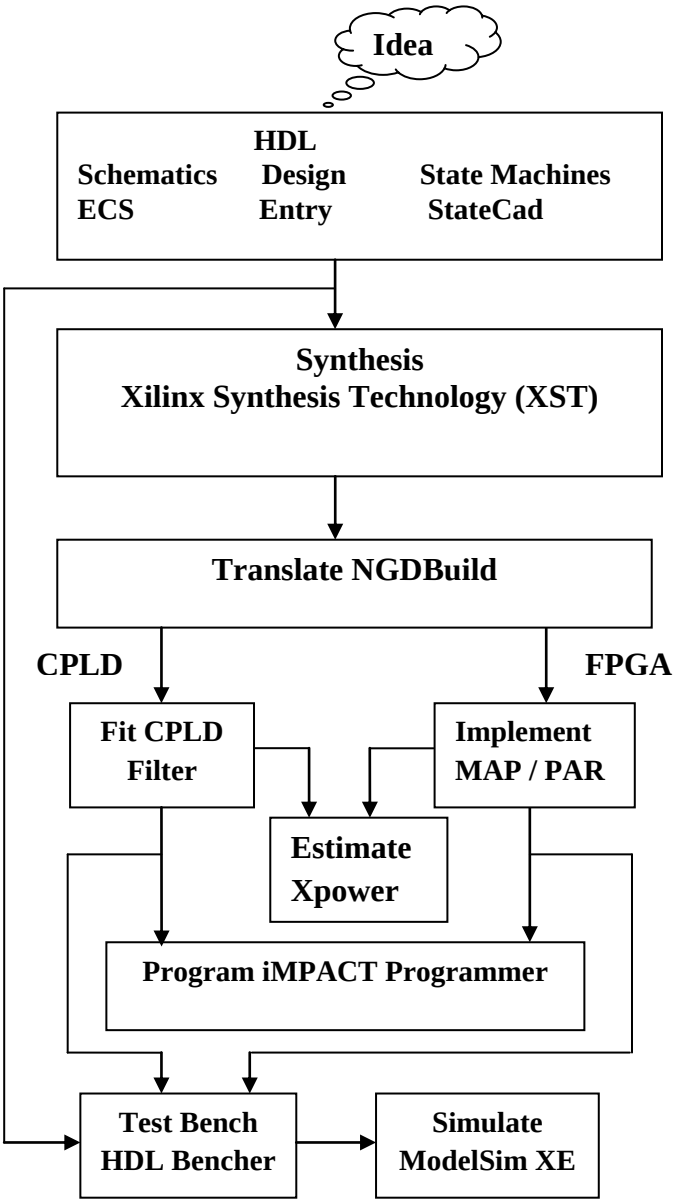


Figure 5.1: Webpack Software Design Flow [20].

The various steps involved are as follows:

a) **Synthesis:** Synthesis is the general term that describes the process of transformation of the model of a design in HDL, from one level of behavioural abstraction to a lower, more detailed level. With reference to VHDL, synthesis is an automatic method of converting a higher level of abstraction to a lower level of abstraction. The synthesis tools convert RTL descriptions to gate level net-lists. The preparation of a synthesizable model requires the knowledge about features. It is important here to note that not all features of VHDL

can be synthesized; therefore, one must consult XILINX simulation and synthesis guide for a list of synthesizable features.

b) **Implementation:** It is divided into three major operations:

(i) **Translation:** Merges all of the input net lists.

(ii) **Mapping:** Map optimizes the gates and removes unused logic. This step also maps the designs logic resources.

(iii) **Place and Route:** The place and route process places each macro from the synthesis net list into an available on the target silicon and connects the macros using routing resources available on the target silicon. The job of the place and route tool is to create the programming files that will be used to specify the logic function of the logic macros in the logic areas and the switch programming of the wires used to connect the macros together.

Each switch adds capacitance and resistance to the routed signal. After a few connections, signals start to slow significantly because of capacitance and resistance of the line. The place and route tools can make trade-offs if speed critical signals are known ahead of time and is implemented using the highest speed interconnecting signals. The placement algorithm also tries to place logical gates on the critical path close to each other so that local interconnect can used to connect the gates.

c) **Generation of Programming File:** this feature generates the bit file to be downloaded on to the target device (FPGA/PROM) using the downloading cable. Thereafter, the following tools are used to program the device:

(i) Impact

(ii) Prom file formatter

The impact programmer module allows you to program a device in-system for all devices available in the WEBPACK software. For FPGAs, the programmer module allows you to configure a device via the JTAG. Impact, a command line and GUI based tool, allows one to:

a) Configure FPGA designs using boundary- scan, master serial

b) Download

c) Read-back and verify design configuration data

d) Perform functional test on any device.

5.5 FPGA

A field programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing hence "field programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application specific integrated circuit (ASIC). Contemporary FPGAs have large resources of logic gates and RAM blocks to implement complex digital computations. As FPGA designs employ very fast IOs and bidirectional data buses it becomes a challenge to verify correct timing of valid data within setup time and hold time. Floor planning enables resources allocation within FPGA to meet these time constraints. FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial reconfiguration of a portion of the design and the low non-recurring engineering costs relative to an ASIC design, offer advantages for many applications [35].

5.5.1 BASIC STRUCTURE

The basic FPGA architecture consists of a two-dimensional array of logic blocks and flip-flops with means for the user to configure [36]

a) The function of each logic blocks,
b) The inputs/outputs
c) The interconnection between blocks. Families of FPGAs differ from each other by the physical means for implementing user programmability, arrangement of interconnection wires, and basic functionality of the logic blocks.

Currently there are four technologies in use. They are static RAM cells, anti-fuse, EPROM transistors, and EEPROM transistors. Depending upon the application, one FPGA technology may have features desirable for that application.

a) **Static RAM Technology:** In the Static RAM FPGA programmable connections are made using pass transistors, transmission gates, or multiplexers that are controlled by SRAM cells. The advantage of this technology is that it allows fast in-circuit reconfiguration. The major disadvantage is the size of the chip required by the RAM technology.

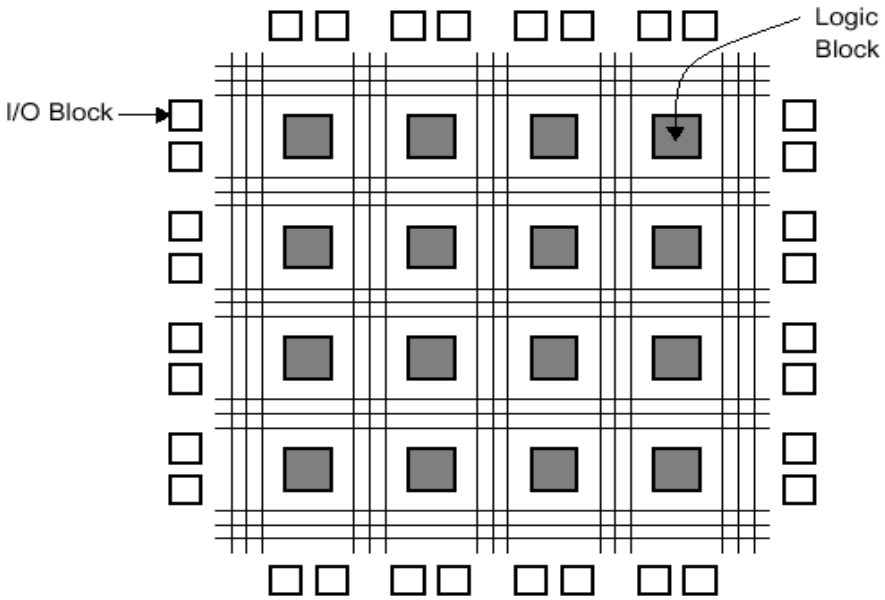


Figure 5.2: FPGA Structure.

b) **Anti-Fuse Technology:** An anti-fuse resides in a high-impedance state and can be programmed into low impedance or "fused" state. A less expensive than the RAM technology, this device is a program-one device.

c) **EPROM / EEPROM Technology:** This method is the same as used in the EPROM memories. One advantage of this technology is that it can be reprogrammed without external storage of configuration; though the EPROM transistors cannot be reprogrammed in-circuit.

5.5.2 DESIGN FLOW

The designer [37] facing a design problem must go through a series of steps between initial ideas and final hardware. This series of steps is commonly referred to as the ‘design flow’. First, after all the requirements have been spelled out, a proper digital design phase must be carried out. It should be stressed that the tools supplied by the different FPGA vendors to target their chips do not help the designer in this phase. They only enter the scene once the designer is ready to translate a given design into working hardware.

The most common flow nowadays used in the design of FPGAs involves the following subsequent phases:

a) **Design Entry:** This step consists in transforming the design ideas into some form of computerized representation. This is most commonly accomplished using Hardware Description Languages (HDLs). The two most popular HDLs are Verilog and the Very High Speed Integrated Circuit HDL (VHDL).

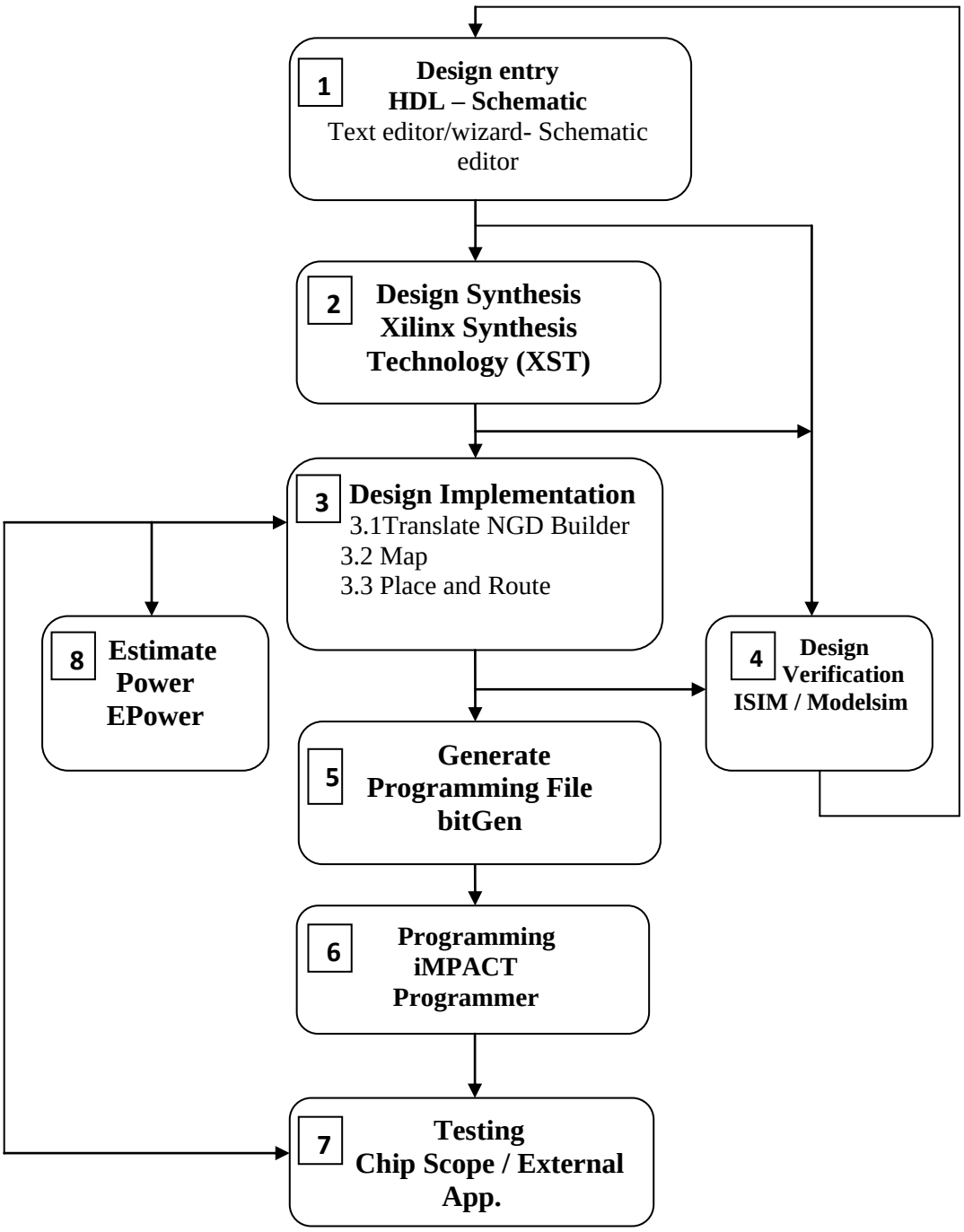


Figure 5.3: Design Flow of FPGA.

b) **Synthesis:** The synthesis tool receives HDL and a choice of FPGA vendor and model. From these two pieces of information, it generates a net-list which uses the primitives proposed by the vendor in order to satisfy the logic behaviour specified in the HDL files. Most synthesis tools go through additional steps such as logic optimization, register load balancing, and other techniques to enhance timing performance, so the resulting net-list can be regarded as a very efficient implementation of the HDL design.

c) **Place and Route:** The placer takes the synthesized net-list and chooses a place for each of the primitives inside the chip. The router's task is then to interconnect all these primitives together satisfying the timing constraints. The most obvious constraint for a design is the frequency of the system clock, but there are more involved constraints one can impose on a design using the software packages supported by the vendors.

d) **Bit Stream Generation:** FPGAs are typically configured at power-up time from some sort of external permanent storage device, typically a flash memory. Once the place and route process is finished, the resulting choices for the configuration of each programmable element in the FPGA chip, be it logic or interconnect, must be stored in a file to program the flash.

5.6 SPARTAN 3E KIT

The Spartan 3E [38] starter kit provides us the basic features as provided by the Spartan 3E FPGA. It also provides easy way to test the various programs in the FPGA itself, by dumping the 'bit' file into the FPGA and then observing the output.

5.6.1 THE VARIOUS PERIPHERALS AVAILABLE IN THE KIT

The Spartan 3E FPGA board comes built in with many peripherals that help in the proper working of the board and also in interfacing the various signals to the board itself. Some of the peripherals included in the Spartan 3E FPGA board include:

a) 2-line, 16-character LCD screen: This LCD screen can be interfaced with the various on-board signals of the FPGA to display various texts as desired by the programmer.

b) PS/2 mouse or keyboard port: A PS/2 keyboard can be connected to the FPGA board and then depending on the key pressed the FPGA would do a variety of things, as programmed.

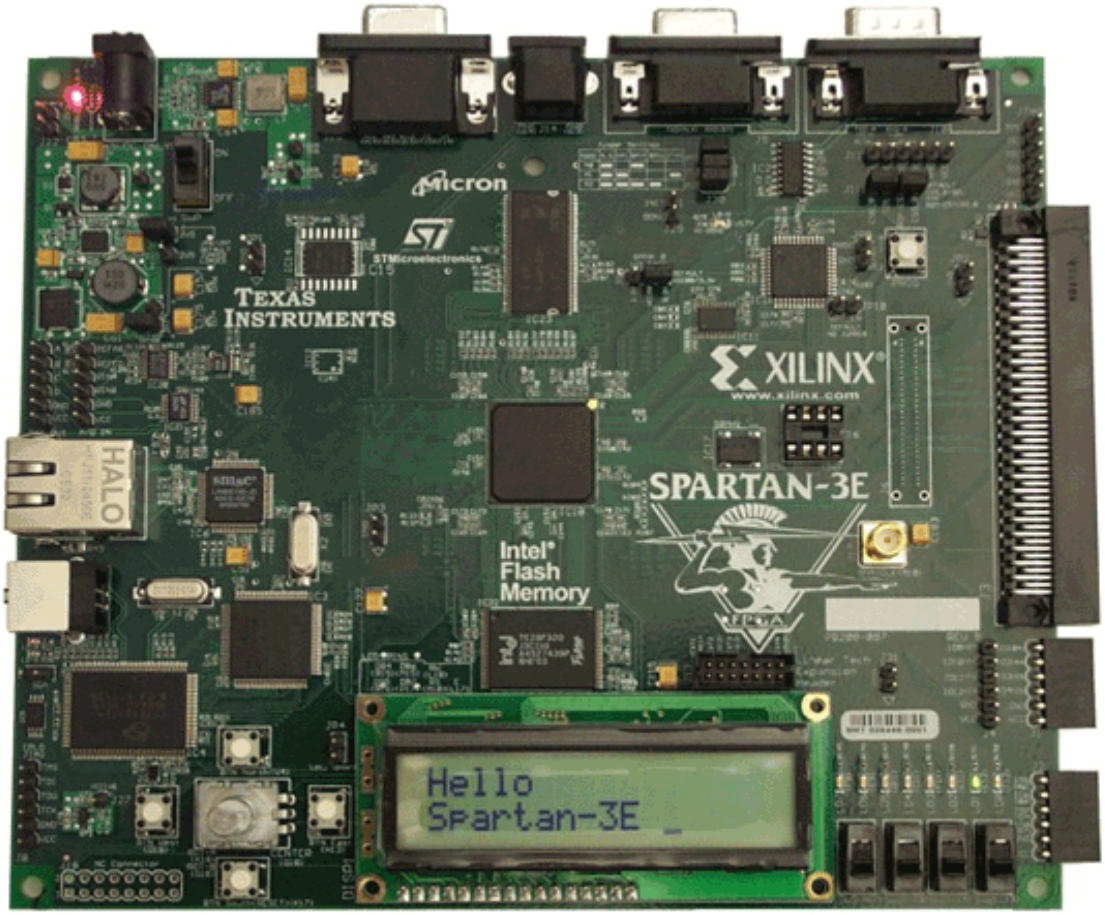


Figure 5.4: SPARTAN 3E KIT.

c) VGA display port: This port can be used to display various encoded images via a screen. The image encoding would be done by the FPGA via the aid of the program and then the encoded image would be displayed on the screen.

d) Two 9-pin RS-232 ports: This ports help in the transmission of serial data to and from the FPGA board.

e) 50 MHz clock oscillator: This is the system clock which helps in giving the clock signal to the various events taking place within the FPGA and the various programs that require clock for their working, A Digital clock manager can also be used to reduce the frequency of the system clock so that is useful for various other purposes which need smaller clock frequency.

f) On-board USB-based FPGA download and debug interface: The programmable file is dumped into the FPGA via the USB based download cable. Hence it is v very much helpful in the testing of the programs whether they are working correctly or not.
g) Eight discrete LEDs: The LEDs can be interfaced to glow when a particular output becomes high. Hence the LEDs can be interfaced to show the output of a single bit.
h) Four slide switches and four push-button switches: These switches are used to give the inputs to the FPGA board. They can also act as the reset switches for the various programs.
i) Four-output, SPI-based Digital-to-Analog Converter (DAC): It is the on-board DAC which is to be interfaced to give the analog output to the digital data values.
j) Two-input, SPI-based Analog-to-Digital Converter (ADC) with programmable gain preamplifier: It is the on-board ADC which converts the real world analog signals into digital values.

CHAPTER 6
RESULTS AND DISCUSSIONS

In this chapter, the low-pass filter has been designed using Kaiser window. The filter specifications are real world and MATLAB FDATOOL is used to find out the filter coefficients.The design of non-pipelined and pipelined FIR filter using both the encoding schemes – Radix-4 and Radix-8 has been accomplished via Hardware Description Language and synthesized on XILINX ISE Software (Xilinx ISE 9.2i version) for Spartan 3E FPGA (Field Programmable Gate Array) family (XC3S1600E).

Although any filter specifications can be taken but for the sake of implementation the following specifications are considered for low-pass filter:

a) Stop-band attenuation = 40 dB
b) Pass-band ripple = 0.01 dB
c) Transition width = 500 Hz
d) Sampling frequency = 10 KHz
e) Ideal cut-off frequency = 1200 Hz

The filter order of given specification is calculated using Kaiser window. From specification,

$$20\,log(\,1+\,\delta_p\,) = 0.01 \text{ dB, giving } \delta_p\, = 0.00115$$

$$-20\,log(\delta_s) = 40 \text{ dB, giving } \delta_s = 0.01$$

Since both the pass-band and stop-band ripples are equal (as they cannot be specified independently) in the window method, we use smaller of the ripples

$$\delta = \delta_s = \delta_p = 0.0015$$

This means stop-band attenuation is more than actually required, in this case

$$-\,20\,log(0.00115) = 58.8 \text{ dB}$$

The number of coefficients required is

$$N = \frac{A - 7.95}{14.36\,\Delta f} = \frac{58.8 - 7.95}{14.36\,(500/10000)} \approx 71$$

The ripple parameter is obtained by

$$\beta = 0.5842\,(A - 21)^{0.4} + 0.07886\,(A - 21)$$

$$\beta = 0.5842\,(58.8 - 21)^{0.4} + 0.07886\,(58.8 - 21) = 5.48$$

The FIR coefficients are obtained from $h(n) = h_D(n)\,w(n)$ where

$$h_D(n) = 2\,f_c \frac{sin(n\omega_c)}{n\omega_c} \qquad n \neq 0$$

$$h_D(n) = 2\,f_c \qquad n = 0$$

$$w(n) = \begin{cases} I_0\left\{\beta\left[1 - \left(\frac{2n}{N-1}\right)^2\right]^{1/2}\right\} \Big/ I_0(\beta) & -(N-1)/2 \leq n \leq (N-1)/2 \\ 0 & elsewere \end{cases}$$

The cut-off frequency used in calculating $h(n)$ is different from that given in specification to account for the smearing effect of the window function and is given by

$$f_c{}' = 1200 + \Delta f/2 = 1450\ Hz$$

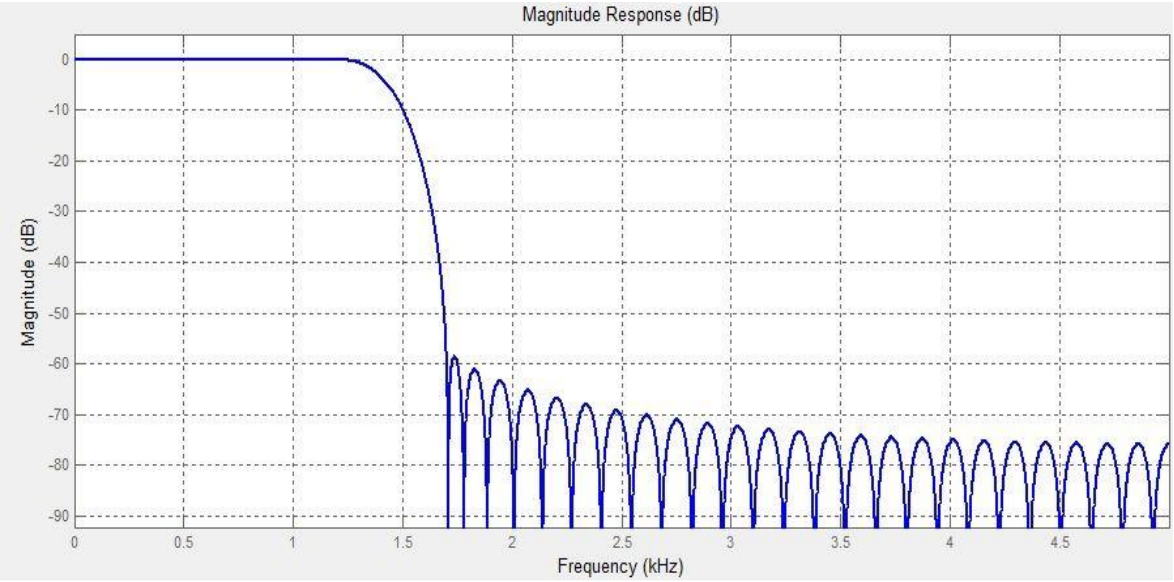


Figure 6.1: Magnitude Response of Low-pass Filter in dB.

The coefficients are calculated using FDA Tool as shown in Figure 6.2 and are given in Appendix I. These coefficients are directly used in VHDL code to generate the low-pass filter.

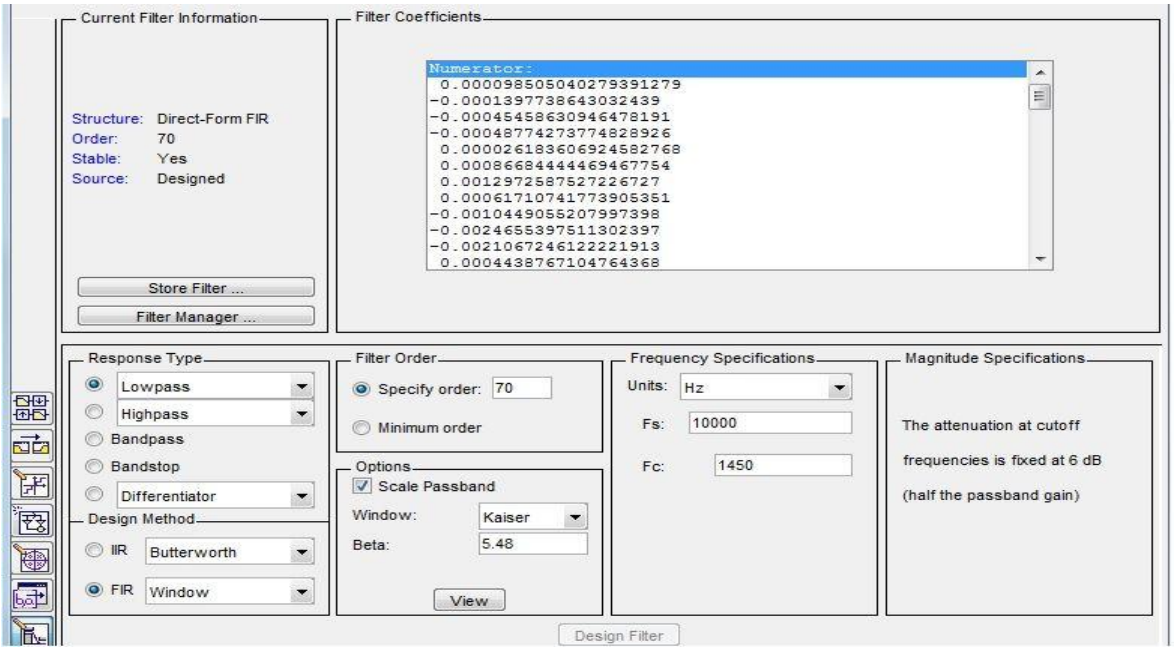


Figure 6.2: FIR Digital Low-pass Filter Parameters.

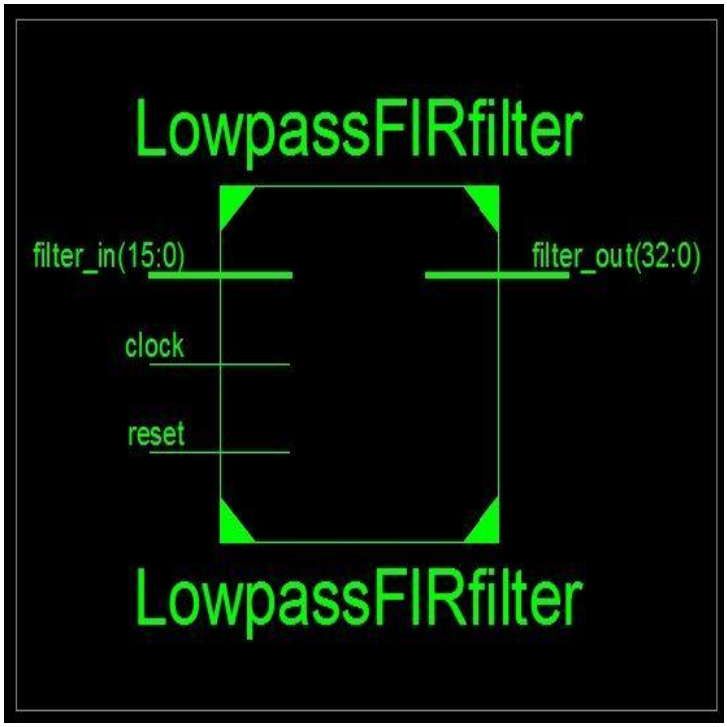


Figure 6.3: Top Level Circuit Diagram of Low-pass Filter.

6.1 SYNTHESIS RESULTS OF FIR FILTER USING RADIX-4 MULTIPLIER

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	1,186	29,504	4%	
Number of 4 input LUTs	19,687	29,504	66%	
Number of occupied Slices	11,304	14,752	76%	
Number of Slices containing only related logic	11,304	11,304	100%	
Number of Slices containing unrelated logic	0	11,304	0%	
Total Number of 4 input LUTs	20,037	29,504	67%	
Number used as logic	19,687			
Number used as a route-thru	350			
Number of bonded IOBs	51	250	20%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.65			

Figure 6.4: Synthesis Report of Non-Pipelined FIR Filter Using Radix-4 Multiplier.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,299	29,504	7%	
Number of 4 input LUTs	11,412	29,504	38%	
Number of occupied Slices	6,714	14,752	45%	
Number of Slices containing only related logic	6,714	6,714	100%	
Number of Slices containing unrelated logic	0	6,714	0%	
Total Number of 4 input LUTs	11,716	29,504	39%	
Number used as logic	11,412			
Number used as a route-thru	304			
Number of bonded IOBs	51	250	20%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.78			

Figure 6.5: Synthesis Report of Pipelined FIR Filter Using Radix-4 Multiplier.

6.2 ADVANCED HDL SYNTHESIS REPORT FOR FIR FILTER USING RADIX-4 MULTIPLIER

```
=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1206
 17-bit adder                                          : 1136
 33-bit adder                                          : 70
# Registers                                            : 1120
 Flip-Flops                                            : 1120
# Comparators                                          : 568
 16-bit comparator less                                : 568
# Xors                                                 : 23856
 1-bit xor3                                            : 23856

=========================================================================
```

Figure 6.6: Advanced HDL Synthesis Report of Non-pipelined FIR Filter Using Radix-4 Multiplier.

```
=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1206
 17-bit adder                                          : 1136
 33-bit adder                                          : 70
# Registers                                            : 2310
 Flip-Flops                                            : 2310
# Comparators                                          : 568
 16-bit comparator less                                : 568
# Xors                                                 : 23856
 1-bit xor3                                            : 23856

=========================================================================
```

Figure 6.7: Advanced HDL Synthesis Report of Pipelined FIR Filter Using Radix-4 Multiplier.

6.3 SIMULATION RESULTS FOR FIR FILTERS USING RADIX-4 MULTIPLIER

Figure 6.8: Simulation of Non-Pipelined FIR Filter Using Radix-4 Multiplier.

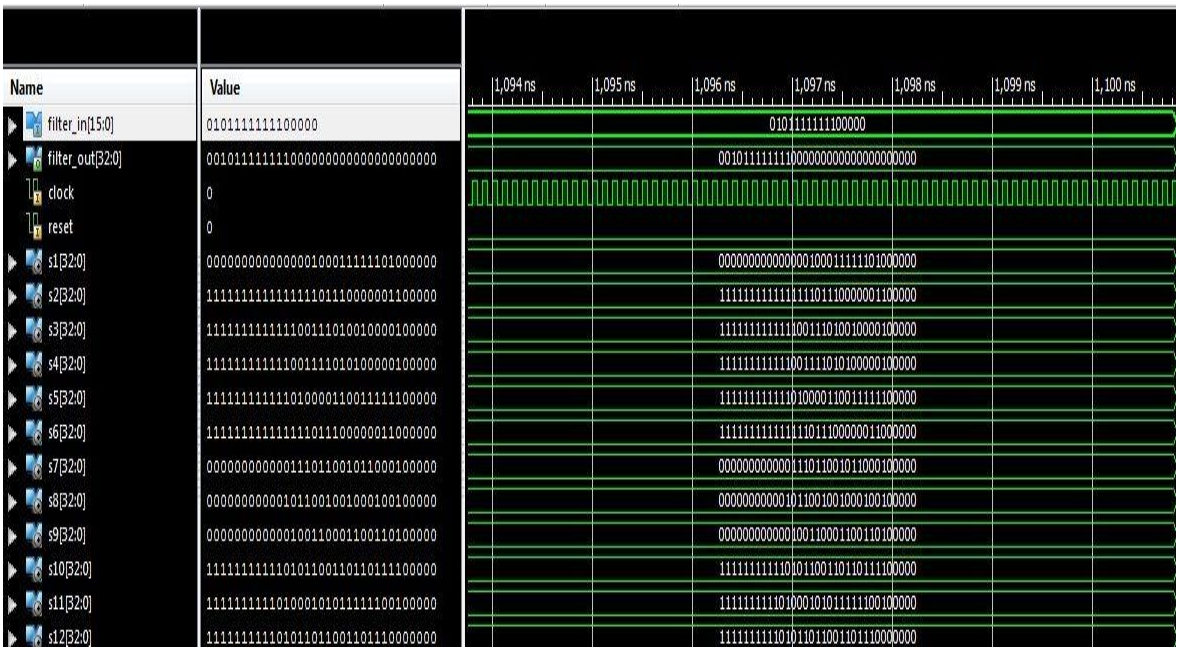


Figure 6.9: Simulation of Pipelined FIR Filter Using Radix-4 Multiplier.

6.4 FPGA IMPLEMENTATION OF FIR FILTER USING RADIX-4 MULTIPLIER

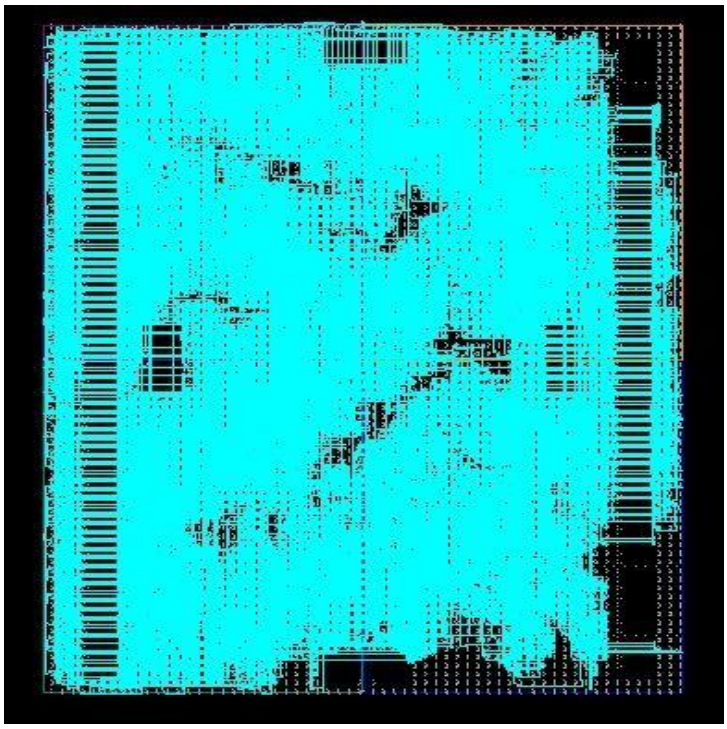


Figure 6.10: FPGA Realization of Non-Pipelined FIR Filter Using Radix-4 Multiplier.

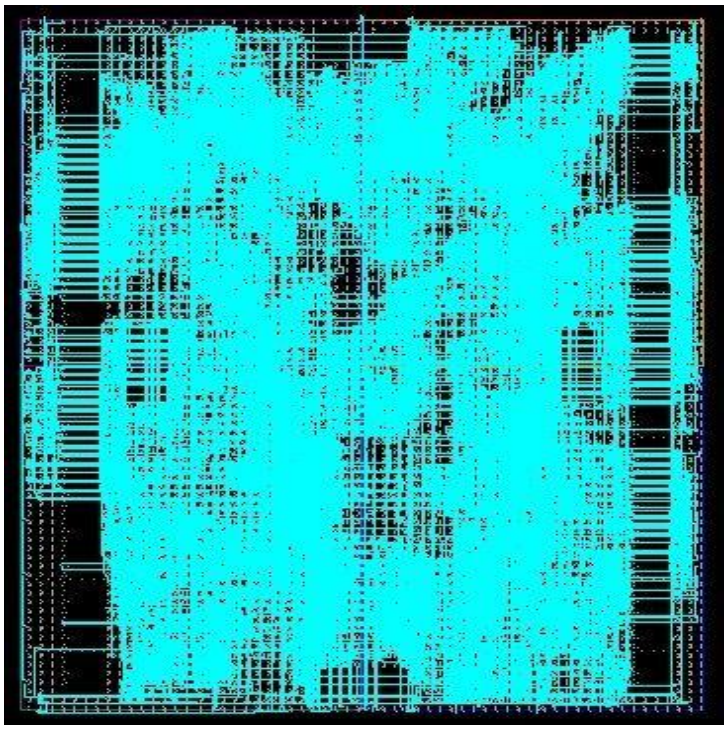


Figure 6.11: FPGA Realization of Non-Pipelined FIR Filter Using Radix-4 Multiplier.

6.5 SYNTHESIS RESULTS OF FIR FILTER USING RADIX-8 MULTIPLIER

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	1,186	29,504	4%	
Number of 4 input LUTs	14,665	29,504	49%	
Number of occupied Slices	8,393	14,752	56%	
Number of Slices containing only related logic	8,393	8,393	100%	
Number of Slices containing unrelated logic	0	8,393	0%	
Total Number of 4 input LUTs	14,917	29,504	50%	
Number used as logic	14,665			
Number used as a route-thru	252			
Number of bonded IOBs	51	250	20%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.55			

Figure 6.12: Synthesis Report of Non-Pipelined FIR Filter Using Radix-8 Multiplier.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,298	29,504	7%	
Number of 4 input LUTs	8,698	29,504	29%	
Number of occupied Slices	5,004	14,752	33%	
Number of Slices containing only related logic	5,004	5,004	100%	
Number of Slices containing unrelated logic	0	5,004	0%	
Total Number of 4 input LUTs	8,906	29,504	30%	
Number used as logic	8,698			
Number used as a route-thru	208			
Number of bonded IOBs	51	250	20%	
Number of BUFGMUXs	1	24	4%	
Average Fanout of Non-Clock Nets	2.75			

Figure 6.13: Synthesis Report of Pipelined FIR Filter Using Radix-8 Multiplier.

6.6 ADVANCED HDL SYNTHESIS REPORT FOR FIR FILTER USING RADIX-8 MULTIPLIER

```
=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1348
 18-bit adder                                          : 1278
 33-bit adder                                          : 70
# Registers                                            : 1120
 Flip-Flops                                            : 1120
# Comparators                                          : 426
 16-bit comparator less                                : 426
# Xors                                                 : 17040
 1-bit xor3                                            : 17040

=========================================================================
```

Figure 6.14: Advanced HDL Synthesis Report of Non-pipelined FIR Filter Using Radix-8 Multiplier.

```
=========================================================================
*                      Advanced HDL Synthesis                           *
=========================================================================

=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1348
 18-bit adder                                          : 1278
 33-bit adder                                          : 70
# Registers                                            : 2310
 Flip-Flops                                            : 2310
# Comparators                                          : 426
 16-bit comparator less                                : 426
# Xors                                                 : 17040
 1-bit xor3                                            : 17040

=========================================================================
```

Figure 6.15: Advanced HDL Synthesis Report of Pipelined FIR Filter Using Radix-8 Multiplier.

6.7 SIMULATION RESULTS FOR FIR FILTER USING RADIX-8 MULTIPLIER

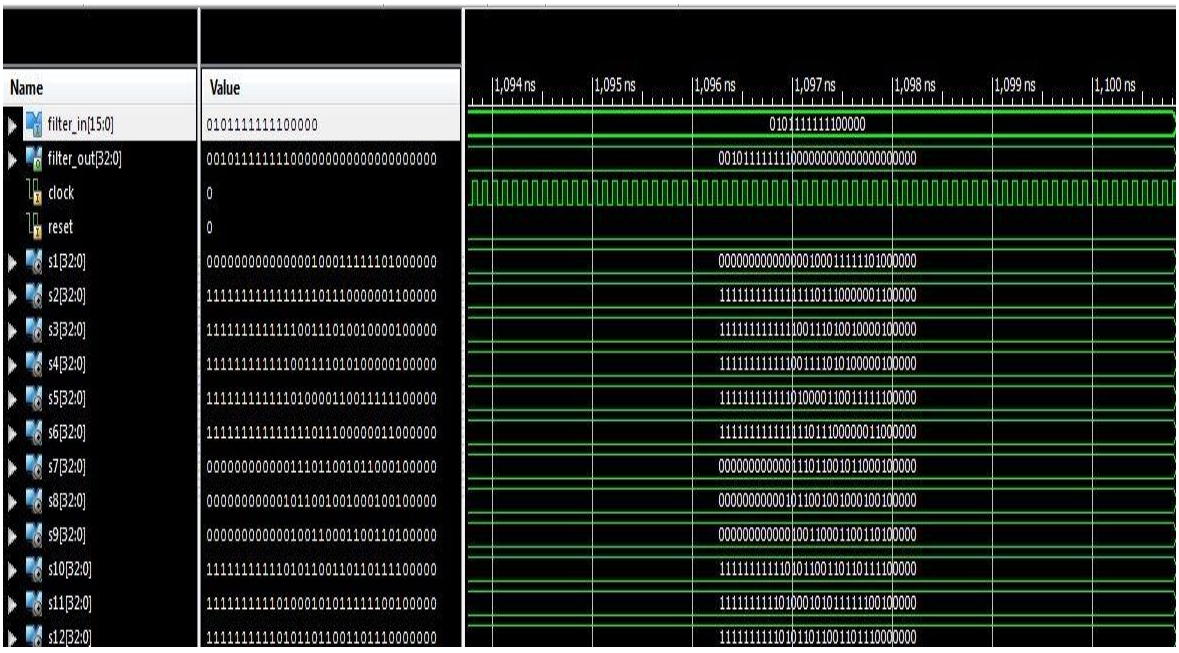


Figure 6.16: Simulation Result of Non-pipelined FIR Filter Using Radix-8 Multiplier.

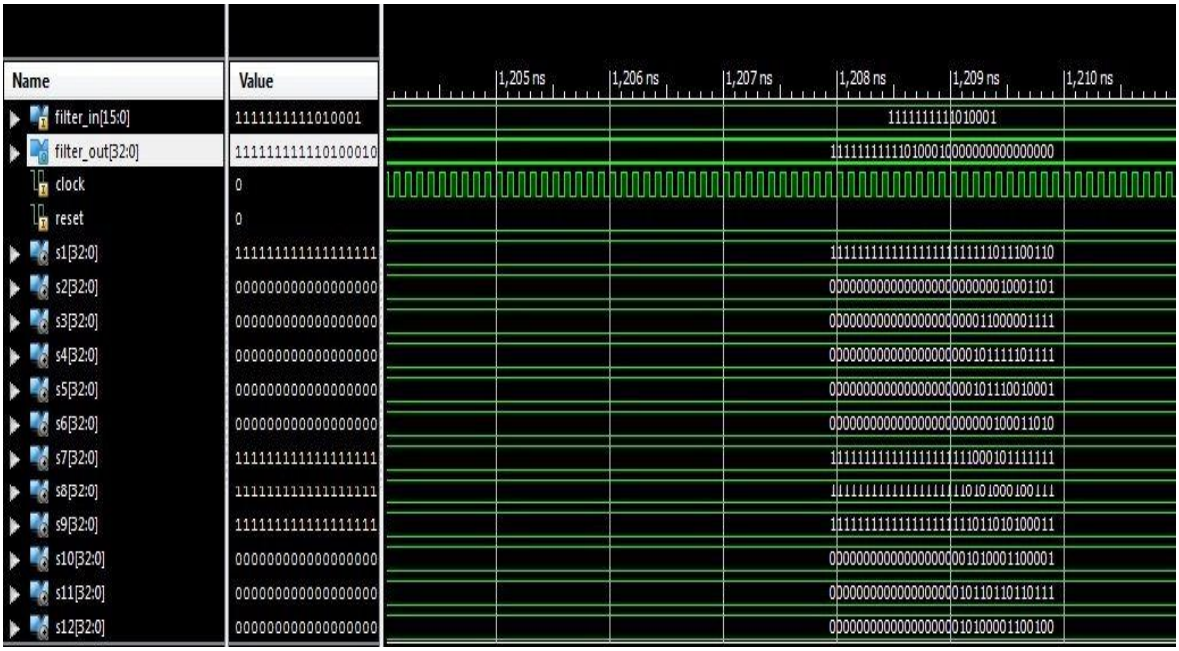


Figure 6.17 Simulation Result of Pipelined FIR Filter Using Radix-8 Multiplier.

6.8 FPGA IMPLEMENTATION OF FIR FILTER USING RADIX-8 MULTIPLIER

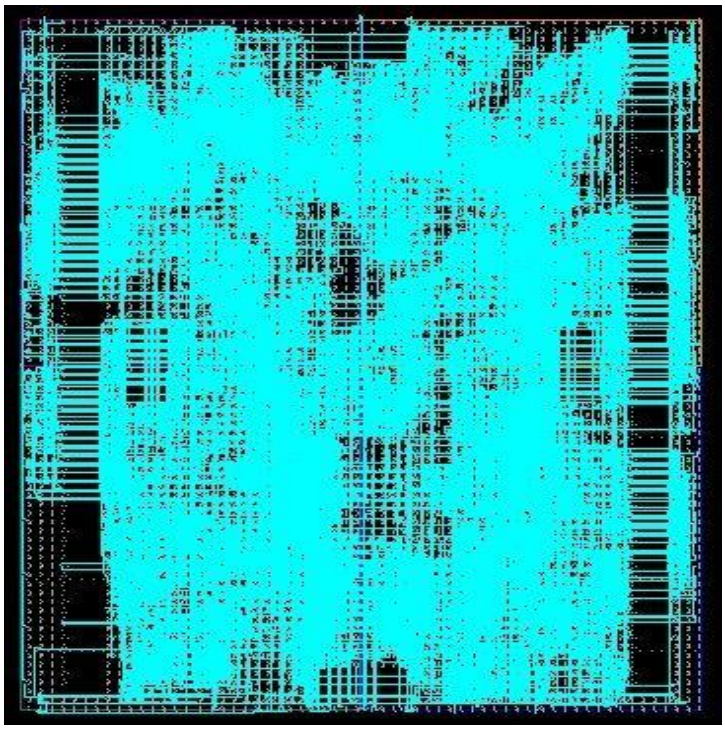


Figure 6.18: FPGA Realization of Non-Pipelined FIR Filter Using Radix-8 Multiplier.

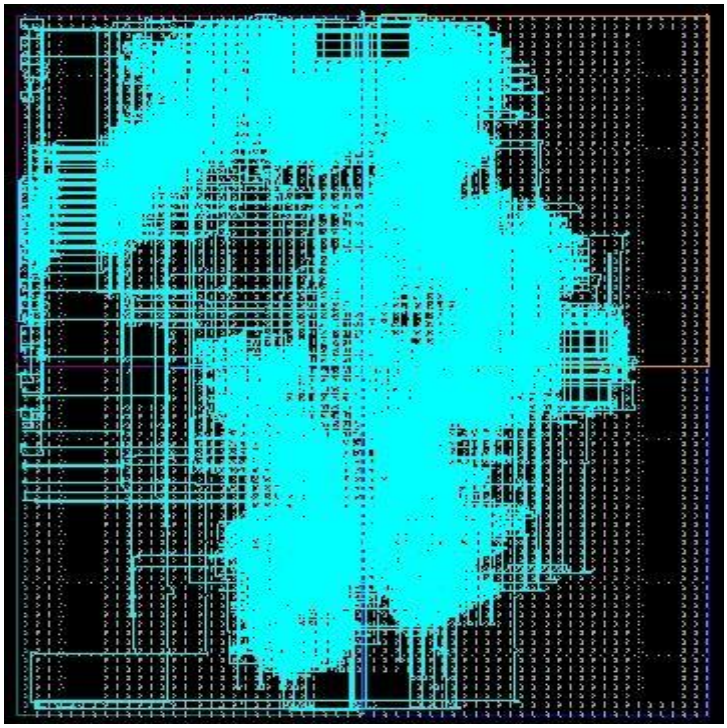


Figure 6.19: FPGA Realization of Pipelined FIR Filter Using Radix-8 Multiplier.

6.9 DELAY AND POWER ANALYSIS FOR FIR FILTER

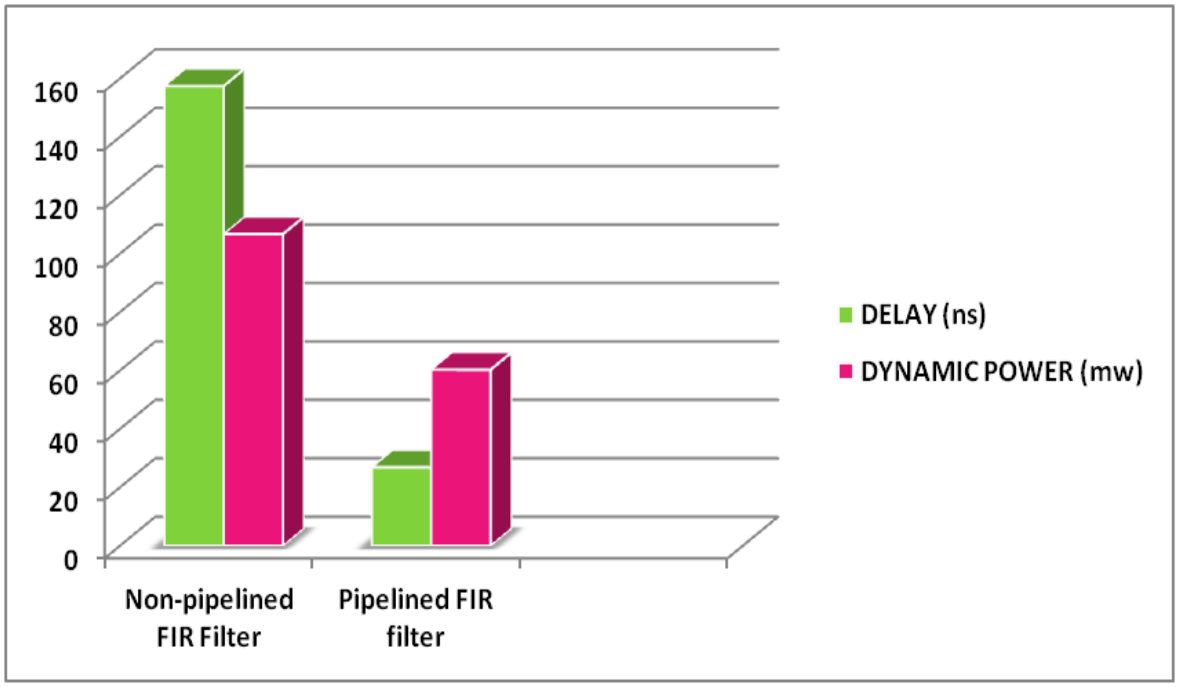


Figure 6.20: Delay and Power Analysis of FIR Filter Using Radix-4 Multiplier.

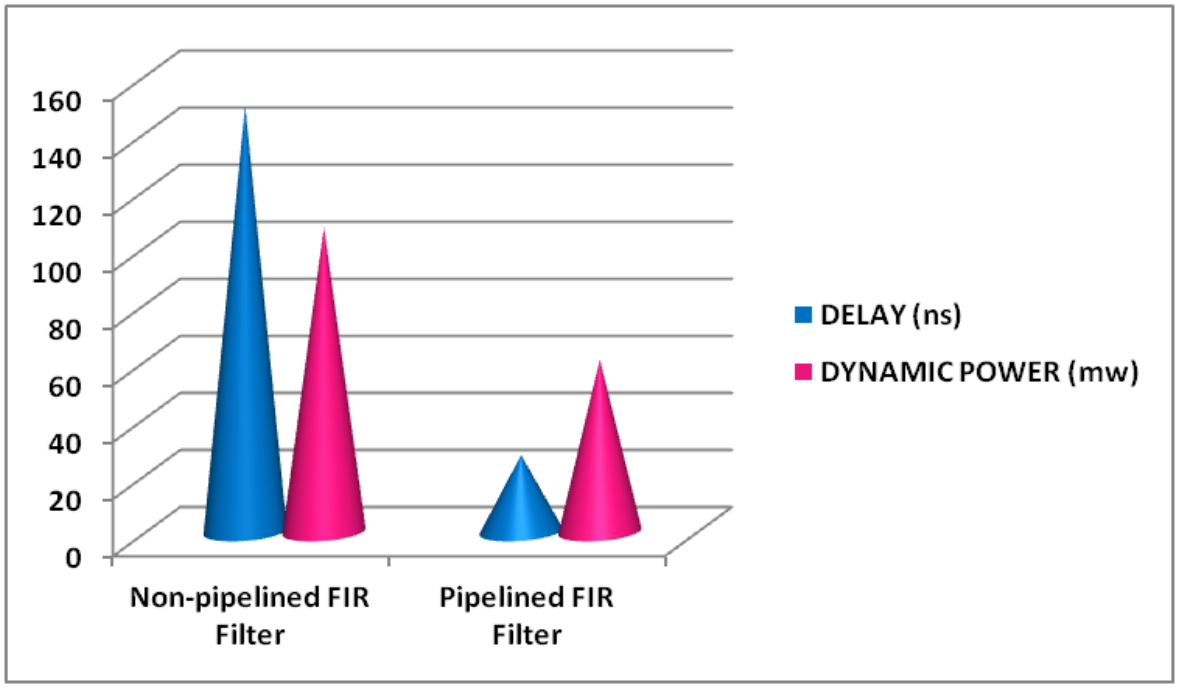


Figure 6.21: Delay and Power Analysis of FIR filter Using Radix-8 Multiplier.

6.10 DISCUSSIONS

In this dissertation, digital filter has been designed using Kaiser window. This technique is simple conceptually and computionally. The design of the filter is formulated as a problem of optimizing $I_0(x)$, which is the modified zeroth-order Bessel function of the first kind. Besides this the adjustable parameter β, has been selected so as to optimize the mainlobe width. The design complexity is much less than that in non-linear optimizations.

On the other hand, because in Kaiser window design the stop-band attenuation is determined by the window, direct control over the stop-band attenuation can be achieved. The advantages of the Kaiser window compared to the compared to the fixed windows are their near optimality and flexibility. Given ω_p, ω_s, and the minimum stop-band attenuation A_p of the filter, the adjustable parameter (β for the Kaiser) can be determined to give the desired value for stop-band attenuation.

The concept of pipelining has been incorporated that results in reducing the delay of the FIR filter, thereby enhancing the speed and reducing the power dissipation as compared to the non-pipelined techniques.

The design of non-pipelined and pipelined FIR filter using both the encoding schemes – Radix-4 and Radix-8 has been accomplished via Hardware Description Language and synthesized on XILINX ISE Software (Xilinx ISE 9.2i version) for Spartan 3E FPGA (Field Programmable Gate Array) family (XC3S1600E). Synthesis reports are shown in Figure 6.4 & Figure 6.5 for FIR filter designed using Radix-4 multiplier and Figure 6.12 & Figure 6.13 for FIR filter designed using Radix-8 multiplier. The Figure 6.20 demonstrates that the pipelined technique enhances the speed and reduces delay from 157.413 ns to 26.892 ns i.e., 82.9 % in pipelined FIR filter designed using Radix-4 multiplier as compared to non-pipelined technique. On the other hand, delay has been reduced from 149.386 ns to 26.934 ns i.e. 81.97 % in pipelined FIR filter designed using Radix-8 multiplier as compared to non-pipelined technique as shown in Figure 6.21. Power analysis shows that the dynamic power has been reduced by the same amount from 106.7415 mW to 60.3062 mW i.e., 43.50 % due to pipelining technique in both the designing of FIR filters as shown in Figure 6.20 and Figure 6.21. As all the structures are implemented using VHDL language, they can be ported to any FPGA family.

CHAPTER 7
Conclusion and Future Scope of Work

7.1 CONCLUSION

The FIR filters are extensively used in digital signal processing and can be implemented using programmable digital processors. With the advancement in Very Large Scale Integration (VLSI) technology as the DSP has become increasingly popular over the years, the high speed realization of FIR filters with less power consumption has become much more demanding. Since the complexity of implementation grows with the filter order and the precision of computation, real-time realization of these filters with desired level of accuracy is now becoming a challenging task. So, the implementation of FIR filters on FPGAs is the need of the day because FPGAs can give enhanced speed and allows reconfigurable architectures for realization of FIR filter. This is due to the fact that the hardware implementation of a lot of multipliers can be done on FPGA which are limited in case of programmable digital processors.

In this dissertation, a seventy order low pass FIR filter has been designed using Kaiser window. The design of the filter is formulated as a problem of optimizing $I_0(x)$ and adjustable parameter β, has been selected so as to optimize the main lobe width. Moreover, we can achieve direct control over stop-band attenuation thereby sustaining optimality and flexibility.

The direct form structure is used in designing of this filter as this approach gives a better performance than common structures in terms of speed of operation, cost and power consumption. In direct form structure, *N* shift registers, *N* adder and *N* +1 multipliers are used to realize the *N* order low pass filter. The concept of pipelining has been incorporated that results in reducing the delay of the FIR filter, thereby enhancing the speed and reducing the power dissipation as compared to the non-pipelined techniques.

The design of non-pipelined and pipelined FIR filter using both the encoding schemes – Radix-4 and Radix-8 are carried out via Hardware Description Language. Simulation and synthesis for FPGAs are accomplished on XILINX ISE Software (Xilinx ISE 9.2i version)

for Spartan 3E series FPGA (Field Programmable Gate Array), target device (XC3S1600E) (Speed Grade -5).

7.2 FUTURE SCOPE OF WORK

The future scope of this work includes the following:

1. The implemented FIR structures at code level can be modified to make full benefit from the FPGA, such as using fast carry chains, Embedded Array Blocks etc.
2. To achieve the peak performance fully parallel pipelined version can be implemented.
3. The A/D and D/A converter can be interfaced within the FPGA.
4. The optimization of the design can be done in terms of area occupied on chip.
5. The combination of parallel and pipelining techniques can be used in designing of FIR filter for further optimization and lowering power consumption.

REFERENCES

[1] S. K. Mitra, “Digital Signal Processing”, New York: Tata McGraw Hill, 2005.

[2] S. W. Smith, “The Scientist and Engineer's Guide to Digital Signal Processing”, San Diego: California Technical Publications, 1997.

[3] L. Tan, and J. Jiang, “Digital Signal Processing: Fundamentals and Applications”, Amsterdam: Academic Press, 2007.

[4] http://en.wikipedia.org/wiki/Digital_filter

[5] P. P. Vaidyanathan, “Optimal Design of Linear-Phase FIR Digital Filters with Very Flat Passbands and Equiripple Stopbands”, *IEEE Transactions on Circuits and Systems*, vol. 32, no. 9, pp. 904-917, Sep. 1985

[6] E. C. Ifeachor, and B. W. Jervis, “Finite impulse response (FIR) filter design” in *Digital Signal Processing: A Practical Approach*, South Asia: Prentice hall, 2002, pp. 342-440.

[7] W. S. Lu, A. Antoniou, and S. Saab, “Sequential design of FIR digital filters for low-power DSP applications”, *Conference Record of the Thirty-First Asilomar Conference on Signals, Systems &amp*, IEEE, pp. 701-704, 1997.

[8] Y.P. Lin, and P. P. Vaidyanathan, “A Kaiser Window Approach for the Design of Prototype Filters of Cosine Modulated Filterbanks”, *Signal Processing Letters, IEEE*, vol. 5, no. 6, pp. 132-134, Jun. 1998.

[9] J. G. Proakis, and D. G. Manolakis, “Digital Signal Processing”, Prentice Hall India publication, 2004.

[10] A. Kuncheva, and G. Yanchev, “Synthesis and Implementation of DSP Algorithms in Advanced Programmable Architectures”, *International Scientific Conference Computer Science*, pp. 153-157, 2008.

[11] P. K. Meher, S. Chandrasekaran, and A. Amira, “FPGA Realization of FIR Filters by Efficient and Flexible Systolization Using Distributed Arithmetic”, *IEEE Transactions on Signal Processing*, vol. 56, no. 7, pp. 3009-3017, Jul. 2008.

[12] R. A. Barapate, “Digital Signal Processing”, Pune: Tech-Max Publication, 2007.

[13] N. H. Phuong ,“ *The FIR Filter Design : The Window Design Method”*, 2009.

[14] Kamaraj, C. K. Sundaram, and J. Senthilkumar, “Pipelined FIR Filter Implementation using FPGA”, *International Journal of Scientific Engineering and Technology*, vol. 1, no. 4, pp. 55-60, Oct. 2012

[15] L. Jieshan, and H. Shizhen, "An Design of the 16-order FIR Digital Filter Based on FPGA", *International Conference on Information Science and Engineering*, pp. 489-492, 2009.

[16] R. Kaur, A. Raman, M. H. Singh, and J. Malhotra, "Design and Implementation of High Speed IIR and FIR Filter using Pipelining", *International Journal of Computer Theory and Engineering*, vol. 3, no. 2, pp. 292-295, Apr. 2011.

[17] V. A. Pedroni, "Circuit Design and Simulation with VHDL", Mass.: The MIT press, 2004, pp. 285-293.

[18] V. V. Haibatpure, P. S. Kasliwal, and B.P. Patil, "Performance Evaluation of Proposed Vedic Multiplier in Microwind", *International Journal of Communication Engineering Applications*, vol. 3, no. 3, pp. 498-502, Jul. 2012.

[19] P. R. Aparna, and N. Thomas, "Design and Implementation of a High Performance Multiplier using HDL", *International Conference on Computing, Communication and Appilcation*, pp. 1-5, 2012.

[20] S. Joshi, and B. Ainapure, "FPGA Based FIR Filter", *International Journal of Engineering Science and Technology*, vol. 2, no. 12, pp. 7320-7323, 2010.

[21] F. Nekoei, Y. S. Kavian, and O. Strobel, "Some Schemes of Realization Digital FIR Filter on FPGA for Communication Application", 20th International Crimean Conference on Microwave & Telecommunication Technology, pp. 616-619, 2010.

[22] X. Jiang, and Y. Bao, "FIR filter design based on FPGA", *International Conference on Computer Application and System Modeling*, pp. 621-624, 2010.

[23] B. Rashidi, B. Rashidi, and M. Pourormazd, "Design and Implementation of Low Power Digital FIR Filter based on low power multipliers and adders on Xilinx FPGA", *International Conference on Electronics Computer Technology*, pp. 18-22, 2011.

[24] N. S. Pal, H. P. Singh, R. K. Sarin, and S. Singh, "Implementation of High Speed FIR Filter using Serial and Parallel Distributed Arithmetic Algorithm", *International Journal of Computer Applications*, vol. 25, no. 7, pp. 26-32, Jul. 2011.

[25] V. Soni, P. Shukla, and M. Kumar, "Application of Exponential Window to Design a Digital Nonrecursive FIR Filter", *International Conference on Communications and Signal Processing*, pp. 1015-1019, 2011.

[26] W. Yunlong, W. Shihu, and J. Rendong, "An Extreme Simple Method for Digital FIR Filter Design" in proc. *Third International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), IEEE*, pp. 410-413, 2011.

[27] J. Li, M. Zhao, and X. Wang, "Design and Simulation of 60-order Filter Based on FPGA" in proc. *International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), IEEE*, pp. 113 – 115, 2011.

[28] V.Sudhakar, N.S.Murthy, and L.Anjaneyulu, "Area Efficient Pipelined Architecture For Realization of FIR Filter Using Distributed Arithmetic", *International Conference on Industrial and Intelligent Information,* pp. 169-173, 2012.

[29] R. P. Rajput, and M. N .S Swamy, "High speed Modified Booth Encoder multiplier for signed and unsigned numbers", *International Conference on Modelling and Simulation*, pp. 649-654, 2012.

[30] https://wiki.ittc.ku.edu/ittc/images/3/37/EECS_140_VHDL_Tutorial.pdf

[31] http://www.mathcom.com/corpdir/techinfo.mdir/q125.html

[32]http://www.mathworks.in/help/signal/examples/introduction-to-the-filter-design-and-analysis-tool-fdatool.html

[33] http://www.mathworks.in/products/filterhdl/

[34] http://www.Xilinx.com/bvdocs/whitepapers/wp116.pdf.

[35] http://en.wikipedia.org/wiki/Field-programmable_gate_array

[36] FPGA Architect - XilinxXC4000/Spartan" by ELANIX Inc.

[37] J. Serrano, "Introduction to FPGA design", pp. 231-247, 2008.

[38] http://www.xilinx.com/support/documentation/boards_and_kits/ug230.pdf

APPENDIX-I

FIR Coefficients Using Kaiser window for Low-pass Filter:

h(0)	9.8470163e-05	h(70)
h(1)	-1.3972411e-04	h(69)
h(2)	-4.5442489e-04	h(68)
h(3)	-4.8756977e-04	h(67)
h(4)	2.6173965e-05	h(66)
h(5)	8.6653647e-04	h(65)
h(6)	1.2967984e-03	h(64)
h(7)	6.1688894e-04	h(63)
h(8)	-1.0445340e-03	h(62)
h(9)	-2.4646644e-03	h(61)
h(10)	-2.1059775e-03	h(60)
h(11)	4.4371801e-04	h(59)
h(12)	3.5954580e-03	h(58)
h(13)	4.5526695e-03	h(57)
h(14)	1.5922295e-03	h(56)
h(15)	-3.8904820e-03	h(55)
h(16)	-7.6398162e-03	h(54)
h(17)	-5.6061945e-03	h(53)
h(18)	2.2010888e-03	h(52)
h(19)	1.0450148e-02	h(51)
h(20)	1.1760002e-02	h(50)
h(21)	2.8239875e-03	h(49)
h(22)	-1.1380549e-02	h(48)
h(23)	-1.9631856e-02	h(47)
h(24)	-1.2665935e-02	h(46)
h(25)	8.0061777e-03	h(45)
h(26)	2.8182781e-02	h(44)
h(27)	2.9474031e-02	h(43)
h(28)	3.8724896e-03	h(42)
h(29)	-3.5942288e-02	h(41)
h(30)	-5.9766794e-02	h(40)
h(31)	-3.7113570e-02	h(39)
h(32)	4.1378026e-02	h(38)
h(33)	1.5291289e-01	h(37)
h(34)	2.5100632e-01	h(36)
h(35)	2.9000000e-01	h(35)