Implementation of FIR Filters on FPGA

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

MASTER of TECHNOLOGY (M.Tech.)
in
VLSI Design & CAD

Submitted By

PRABHAT RANJAN
Roll No.-60661018

Under the guidance of

Dr. KULBIR SINGH
Assistant Professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING
THAPAR UNIVERSITY
(Formerly Thapar Institute of Engineering and Technology)
Patiala-147004, Punjab, India

July 2008

CERTIFICATE

I, PRABHAT RANJAN **(Roll No.-60661018)** hereby declare that the work which is being presented in this thesis entitled **"Implementation of FIR Filter on FPGA"** by me in partial fulfillment of requirements for the award of degree of Master of Technology in VLSI Design & CAD from Thapar University, Patiala is an authentic record of my own work carried under the supervision and guidance of Dr. Kulbir Singh, Assistant Professor, Electronics & Communication Engineering Department, Thapar University, Patiala.

The matter presented in this thesis has not been submitted in any other University or Institute for the award of any other degree.

Prabhat Ranjan

(PRABHAT RANJAN)

This is certified that the above statement made by the candidate is correct to the best of my knowledge.

Date: 24.7.08

Kulbir

(Dr. KULBIR SINGH)
Assistant Professor (ECED)
Thapar University
Patiala-147004

AKChatterji

(Dr. A. K. CHATTERJEE)
Professor & Head (ECED)
Thapar University
Patiala-147004

R K Sharma

(Dr. R.K. SHARMA)
Dean of Academic Affairs
Thapar University
Patiala-147004

ACKNOWLEDGEMENT

With great pleasure, I avail this opportunity to express my deep sense of gratitude and indebtedness to my guide **Dr. Kulbir Singh**, Assistant Professor, Electronics & Communication Engineering Department, Thapar University, Patiala, for his spirited guidance and inspiration in completing this thesis. I consider myself extremely fortunate for getting the opportunity to learn and work under their able supervision. I have deep sense of admiration for their innate goodness and inexhaustible enthusiasm. It helped me to work in right direction to attain desired objectives

I am also thankful to all my friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I do not find enough words with which I can express my feeling of thanks to the entire faculty and staff of ECED, Thapar University, Patiala, for their help, inspiration and moral support which went a long way in successful completion of my work. I thank all those who have contributed directly or indirectly to this work

Lastly, and more importantly, I would like to thank my parents for their years of unyielding love and encouragement. They have always wanted the best for me and I admire my parents' determination and sacrifice to put me through college.

Prabhat Ranjan

Prabhat Ranjan

Roll.No:60661018

M.Tech (VLSI Design & CAD)

ABSTRACT

The Finite Impulse Response (FIR) filter is a digital filter widely used in Digital Signal Processing applications in various fields like imaging, instrumentation, communications, etc. Programmable digital processors signal (PDSPs) can be used in implementing the FIR filter. However, in realizing a large-order filter many complex computations are needed which affects the performance of the common digital signal processors in terms of speed, cost, flexibility, etc.

Field-Programmable gate Array (FPGA) has become an extremely cost-effective means of off-loading computationally intensive digital signal processing algorithms to improve overall system performance. The FIR filter implementation in FPGA, utilizing the dedicated hardware resources can effectively achieve application-specific integrated circuit (ASIC)-like performance while reducing development time cost and risks.

In this thesis, a low-pass, band pass and high pass FIR filter is implemented in FPGA. Direct-form approach in realizing a digital filter is considered. This approach gives a better performance than the common filter structures in terms of speed of operation, cost, and power consumption in real-time. The FIR filter is implemented in Spartan-III-xc3s500c-4fg320 FPGA and simulated with the help of Xilinx ISE (Integrated Software Environment). Software WEBPACK project navigator 9.2i was used for synthesizing and simulation the code.

Codes for direct form fixed point FIR filter have been realized. Modules such as multiplier, adder, ram and two's compliment were used. For an N order filter the number of shift register and adders required is N and the number of multipliers required is N+1. These filters can work in real time.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATION

ADC	Analog-To-Digital Converter
ASIC	Application Specific Integrated Circuits
BIBO	Bounded Input-Bounded Output
BPF	Band Pass Filter
CAD	Computer-Aided Design
CPLD	Complex Programmable Logic Device
DAC	Digital-To-Analog Converter
DCM	Digital Clock Management
DSP	Digital Signal Processor
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Arrays
HDL	Hardware Description Languages
HPF	High Pass Filter
IEEE	Institute of Electrical and Electronic Engineers
IIR	Infinite Impulse Response
ISE	Integrated Software Environment
LPF	Low Pass Filter
MAC	Multiply-Accumulate
MXE	ModelSim Xilinx Edition
PDSP	Programmable Digital Processors Signal
PLD	programmable logic device
SPLD	Simple Programmable Logic Device
UCF	User Constraints File
VHDL	Very High Speed Integrated Circuit Hardware Description Language

CHAPTER 1

Introduction

1.1 General

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].

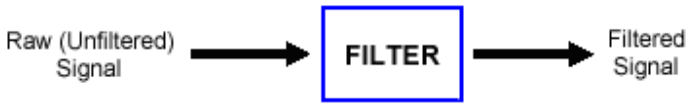


Figure 1.1 A block diagram of a basic filter.

There are two types of filter analog and digital. FIR Filter is the kind of digital filter, which can be used to perform all kinds of filtering i.e. high pass, low pass, band pass and band reject etc.

1.1.1 Analog Filters

An analog filter uses analog electronic circuits made up from components such as resistors and capacitors to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, signal enhancement, and many other areas [2].

- *Advantages:*
  - Simple and consolidated methodologies of plan,
  - Fast and simple realization.
- *Disadvantages:*
  - Little stable and sensitive to temperature variations,
  - Expensive to realize in large amounts.

1.1.2 Digital Filters

A digital filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialized DSP (Digital Signal Processor) chip [2].

- *Advantages:*
  - A digital filter is programmable,
  - Digital filters are easily designed, tested and implemented on computer or workstation,
  - Digital filters are extremely stable with respect both to time and temperature,
  - Digital filters can handle low frequency signals accurately,
  - Digital filters are very much versatile.

1.2 Basics of Digital Filter

In signal processing, there are many instances in which an input signal to a system contains extra unnecessary content or additional noise which can degrade the quality of the desired portion. In such cases we may remove or filter out the useless samples. For example, in the case of the telephone system, there is no reason to transmit very high frequencies since most speech falls within the band of 400 to 3,400 Hz. Therefore, in this case, all frequencies above and below that band are filtered out. The frequency band between 400 and 3,400 Hz, which isn't filtered out, is known as the passband, and the frequency band that is blocked out is known as the stopband.

1.2.1 Digital Filter Types

There are two basic types of digital filters, Finite Impulse Response (FIR) and Infinite Impulse Response (IIR) filters. The general form of the digital filter difference equation is

$$y(n) = \sum_{i=0}^{N} a_i x(n-i) - \sum_{i=1}^{N} b_i y(n-i) \qquad (1.1)$$

where, *y(n)* is the current filter output, the $y(n-i)$'s are previous filter outputs, the $x(n-i)$'s are current or previous filter inputs, the a_i are the filter's feed forward

coefficients corresponding to the zeros of the filter, the b_i are the filter's feedback coefficients corresponding to the poles of the filter, and N is the filter's order.

FIR, Finite Impulse Response, 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.

IIR filters have one or more nonzero 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 [3].

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 nonlinear 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 filter's coefficients are symmetric, as is the case in most standard filtering applications. A FIR's implementation noise characteristics are easy to model, especially if no intermediate truncation is used. In this common implementation, the noise floor is at - 6.02 B + 6.02 $\log_2$N dB, where B is the number of actual bits used in the filter's coefficient quantization and N is again the filter order. That's why most Intersil filter ICs have more coefficient bits than data bits.

An IIR filter's poles may be close to or outside the unit circle in the Z plane. This means an IIR filter may have stability problems, especially after quantization is applied. An FIR filter is always stable [3].

A digital filter is characterized in terms of difference equations .There are two types of digital filters, they are non-recursive, and recursive filters which are characterized based on their responses [1].

The response of a non-recursive filter at any instant depends on the present, past and future values of the input. At any specific instant *nT*. The response is of the form

$$y(nT) = f(...., x(nT - T), x(nT), x(nT + T).....) \tag{1.2}$$

Assuming linearity and time-invariance *y(nT)* can be expressed as

$$y(nT) = \sum_{i=-\infty}^{\infty} a_i x(nT - iT) \tag{1.3}$$

where 'a_i's represents constants.

Now assuming causality for the filter we have

$$a_{-1} = a_{-2} = ... = 0$$

In addition, assuming $a_i = 0$ for i > N the response can be written as Nth-order linear difference equation given as:

$$y(nT) = \sum_{i=0}^{N} a_i x(nT - iT) \tag{1.4}$$

Such a linear, time-invariant, causal, non-recursive filter represented as N^{th}-order linear difference equation is called the Finite Impulse Response (FIR) filter.

When a unit impulse defined as

$$\delta(nT) = \begin{cases} 1 & for\ n = 0 \\ 0 & for\ n \neq 0 \end{cases}$$

is applied to the system described by Equation (1.4), then the response, which is nothing but the impulse response *h(nT)* is given as

$$h(nT) = \sum_{i=0}^{N} a_i \delta(nT - iT) \tag{1.5}$$

From the above equation it can be inferred that the impulse response is finite and from the property of the impulse function we can see that the constants 'a_i's are nothing but the samples of the impulse response. That means

$$h(0) = a_0, h(T) = a_1 .....h(nT) = a_n \tag{1.6}$$

these constants are called the filter coefficients. They determine the type of the filter, whether it is Low-pass, or High-pass, etc. Thus in filter design it is always important to find the filter coefficients which mostly approximates the desired response.

In general, one can view equation 1.3 as a computational procedure (an algorithm) to determine the output sequence *y(nT)* from the input sequence *x(nT).* In addition, in various ways, the computations in equation 1.3 can be arranged into equivalent sets of difference equations. Normally such a kind of re-arrangement of the basic difference equation is done, to gain benefits in terms of memory, time-delays, computational complexity, etc. before implementing the system in the computer. Each set of equations

defines a computational procedure or an algorithm for implementing it in a digital computer system [1].

From these set of difference equations we can construct a block diagram consisting of an interconnection including delay elements, multipliers, and adders. Such a block diagram can be further analyzed in terms of signal flow diagrams. Such a block diagram can be referred as a *realization* of the system or in other words as a *structure* for realizing the system. These structures are nothing but the filter structures.

One of the limitations of the FIR filter is that the order of the filter is generally large in order to meet the desired specifications of the filter. As the filter order is increased, the computational complexity is more which may limit the frequency of operation.

Traditionally, a DSP algorithms are implemented either using general purpose DSP processors (low speed, less expensive, flexible) or using Application Specific Integrated Circuits (ASIC) which offer high speed but are expensive and less flexible.

An alternate approach is to use Field Programmable Gate Arrays (FPGA) as they provide solutions that maintain both the advantages of the approach based on DSP processors and the approach based on ASICs. Since many current FPGA architectures are in-system programmable, the configuration of the device may be changed to implement different functionality if required.

1.2.2 Digital FIR filters Characterrization

The behavior and performance of FIR filter can be characterized using following few terms:

- **Filter Coefficients** - The set of constants, also called tap weights, used to multiply against delayed sample values. For an FIR filter, the filter coefficients are, by definition, the impulse response of the filter.
- **Impulse Response** - A filter's time domain output sequence when the input is an impulse. An impulse is a single unity-valued sample followed and preceded by zero-valued samples. For an FIR filter the impulse response of a FIR filter is the set of filter coefficients.
- **Tap** - The number of FIR taps, typically N, tells us a couple things about the filter. Most importantly it tells us the amount of memory needed, the number of calculations required, and the amount of "filtering" that it can do. Basically, the more taps in a filter

results in better stop band attenuation (less of the part we want filtered out), less rippling (less variations in the pass band), and steeper roll-off (a shorter transition between the pass band and the stop band).

- **Multiply-Accumulate (MAC)** - In the context of FIR Filters, a "MAC" is the operation of multiplying a coefficient by the corresponding delayed data sample and accumulating the result. There is usually one MAC per tap.

1.3 Advantages of FIR filters

- FIR filters are simple to design;
- They are guaranteed to be bounded input-bounded output (BIBO) stable.
- FIR filter can be guaranteed to have linear phase. This is a desirable property for many applications such as music and video processing.
- 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[2].

1.4 Real-World Applications of FIR Filters

A few popular applications for FIR filters are listed below:

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

- ISDN
- Image enhancement in
    - HDTV
    - DTV
    - Video Processing
    - Digital Cameras
    - Digital Video Camcorders
- Special effects
    - Reverberation/echo effect
    - Video, audio
- Wireless/satellite communications security
    - Spread-spectrum jamming compensation
- Biomedical signal processing
    - Compensation of EOG contamination of EEG reading
    - De-emphasis of maternal ECG to easily observe fetal ECG

1.5 Field Programable Gate Array (FPGA)

FPGA arrived in 1984 as an alternative to programable logic devices(PLDs) and ASICs. FPGA offers the significant benifits of being readily programable. FPGA can be programed again and again, giving designers multiple oppertunities to tweak ther circiuts. FPGA consists of an array of logic blocks that are configured using software. Programable input/output blocks surround these logiic blocks. Botha are connected by programable interconnects.

Today , however, FPGA offers millions of gates of logic capacity, operate at 300 MHz, can cost less then $10, and offer integrated functions like processors and memory. FPGA offers all of the features needed to implement most complex designs. Clock management is facilitated by on-chip PLL(phase-locked loop) or DLL(delay-locked loop) circuitry. Dedicated memory blocks can be configured as basic single-port RAMs, ROMs, FIFOs, or CAMs.

Now a days FPGAs are system building resourceas such as high-speed serial input/output, arithmetic modules, embedded processors, and large amount of memory.

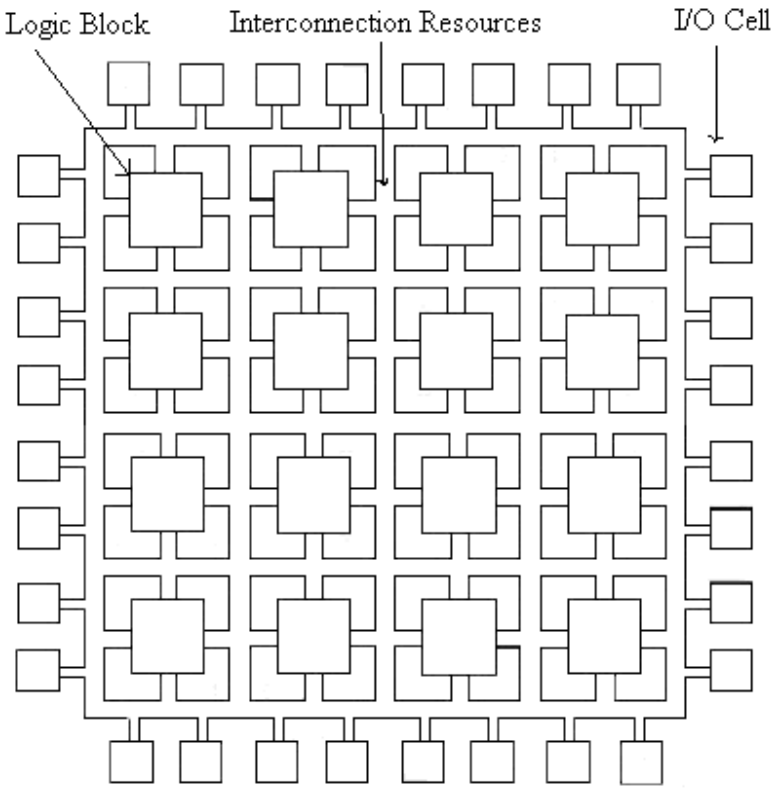


Figure1.2 FPGA Architecture

1.6 Objective of Thesis

The thesis embodics following objectives:

(1) To study the different methods of calculating filter coefficients such as Windowing, Frequency sampling and Optimal method for FIR filter design.

(2) To study various FIR filter structures used for implementing the filters.

(3) To study various synthesis and simulation tools used to implement FIR filter.

(4) To design and implement FIR lowpass, bandpass and highpass filters on FPGA.

1.7 Organization of Thesis

Chapter 2 discusses the design stage for digital filter, which includes specification of filter, calculation of filter coefficients, realization of filter structure, finite worldlegnth effect and hardware or software implementation of filter. Also discusses coefficient calculation method for FIR filter, such as window, frequency sampling and optimal method. And at last comparison between these methods are presented.

Chapter 3 discusses the analysis of linear, time-invariant FIR filter which is generally carried out by using the *Z*-transforms; a brief review of the *Z*-transform is presented. Also the filter structures characterizing the difference equations are represented using basic elements such as multipliers, time-delays, and adders. The characteristics of an ideal FIR filter and the design using windowing techniques are given in this chapter.

In Chapter 4, the Simulation and Synthesis tools which are used in implementation of FIR filter is discussed. This chapter also discusses the FPGAs architecture, FPGA Design Flow, Spartan-III FPGA kit specifications.

Chapter 5 discusses about how FIR filter can be realized and the process of implementation of FIR filter. It also discusses restriction and assumption of filter, choosing the filter structure, because it is often important to choose a particular filter structure for a given transfer function *H (z)*.This chapter also discusses Fixed Point Representation of data.
Chapter 6 contains results of simulation using ModelSim and XILINX ISE 9.2i of lowpass, bandpass and highpass filters of given specifications.

Chapter 7 concludes the thesis and also discusses future scope of work.

CHAPTER 2

FIR Filter Design

2.1 Introduction

The design of a digital filter involves following five steps

- **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 wordlength of the input data.
- **Filter coefficient calculation:** The coefficient of a transfer function H(z) is determined in this step, which will satisfy the given specification. The choice of coefficient calculation method will be influenced by several factors. The most important of which are the critical requirements i.e specification.
- **Realization:** This involves converting the transfer function into a sutable filter network or structure.

These five inter related steps are summarized by flow chart

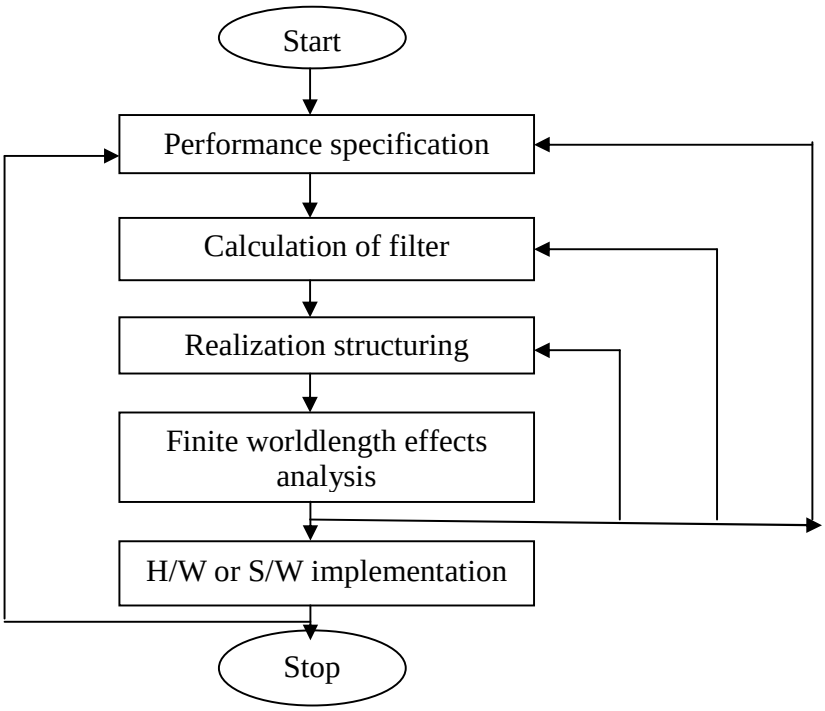


Figure 2.1 Summary of design stage for digital filter [4]

- **Analysis of finite wordlength effects:** The effect of quantizing the filter coefficients and input data as well as the effect of carrying out the filtering

operation using fixed wordlength on the filter performance is analyzed here.

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

2.2 FIR Filter Specifications

The specifications includes

(i) Signal characteristics.

(ii) The characteristics of the filter.

(iii) The manner of implementation.

(iv) Other design constraints (cost).

Although the above requirements are application dependent it will be helpful to devote some time on the characteristics of the filter. The characteristics of digital filters are often in specified in the frequency domain. For frequency selective filters, such as lowpass and bandpass filters, the specifications are often in the form of tolerance.

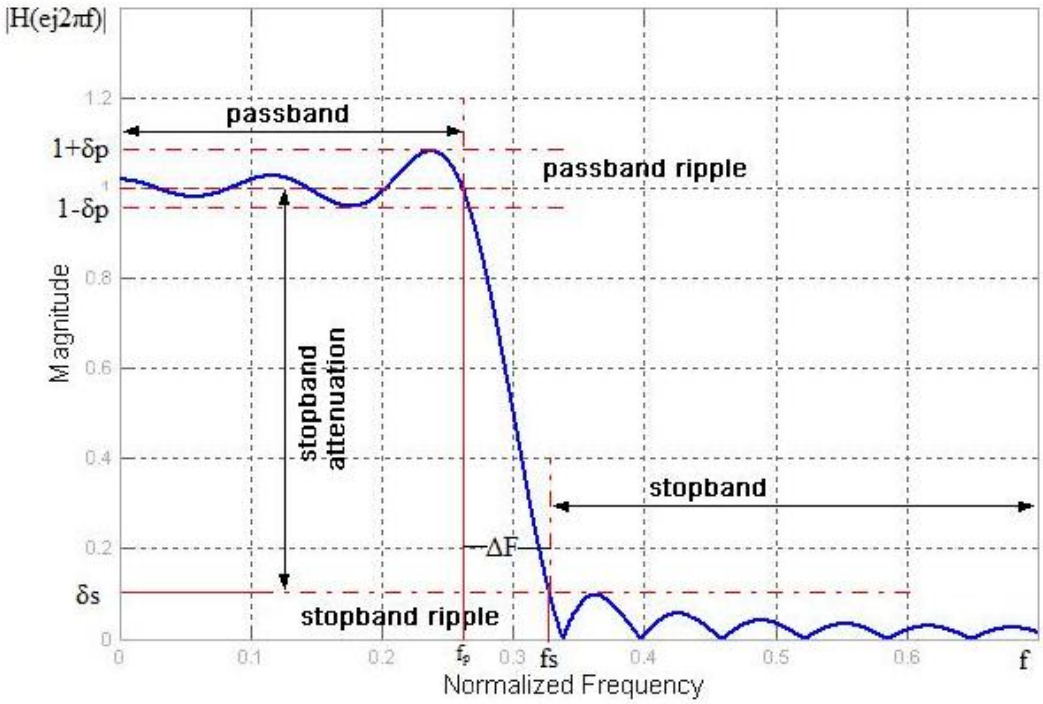


Figure 2.2 Magnitude frequency response specifications for a lowpass filter.

In the passband, the magnitude response has a peak deviation of δ_p and in the stopband, it as a maximum deviation of δ_s. The width of transition band determines haw

sharp the filter is. The magnitude response decreases monotonically from the passband to stopband in this region.

The following are the key parameters of interest:

δ_p peak passband deviation(or ripples)

δ_s stopband deviation.

f_s stopband edge frequency.

f_p passband edge frequency.

F_s sampling frequency.

The edge frequencies are often given n the normalized form, that is as the fraction of the sampling frequency (f/F_s). Passband and stopband deviation may be expressed in decibels. When they specify the passband ripples and minimum stopband attenuation respectively. Thus the minimum stopband attenuation, A_s and the peak passband ripple, $A_{p,}$ in decibels are given as

A_s (stopband attenuation) = $-20\log_{10} \delta s$

A_p (passband ripple) = $20 \log_{10} (1+\delta_p)$

The difference between f_s and f_p gives the transition width of the filter. Another important parameter is the filter length, N, which defines the number of filter.

2.3 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. Several methods are available for obtaining $h(n)$. The window, optimal and frequency sampling method are the most commonly used [4].

2.3.1 Window Method

In this method, use is made of the fact that the frequency response of a filter, $H_D(\omega)$and the corresponding impulse, h_D are related by the Fourier transform:

$$h_D(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_D(\omega)e^{j\omega n}d\omega \qquad (2.1)$$

Now start with the ideal lowpass response shown in figure, where ω_c is the cutoff frequency and the frequency scale is normalised: T=1. By letting the response go from

– ω_c to ω_c we simplify the integration operation. Thus the impulse response is given by:

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

$$= \frac{2f_c}{n\omega_c}\sin(n\omega_c) \quad , n \neq 0, -\infty \leq n \leq \infty$$

$$= 2f \quad , n = 0 \qquad (2.2)$$

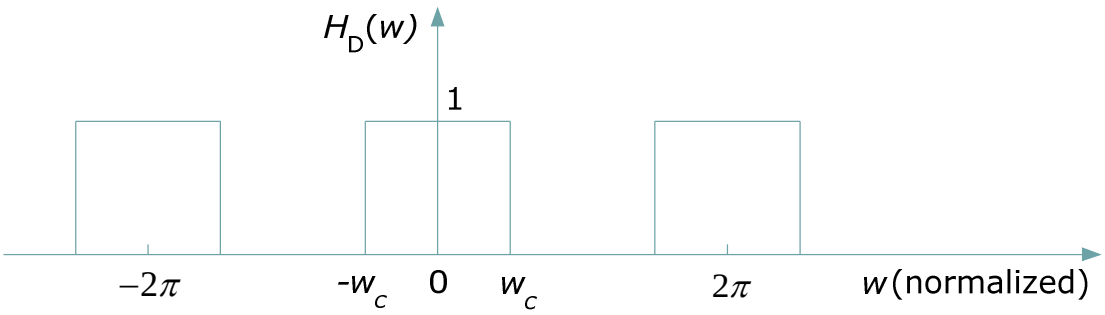


Figure 2.3 Ideal frequency response of a lowpass filter.

Table2.1 Summary of ideal impulse responses

Filter type	$h_D(n), n \neq 0$	$h_D(0)$
Lowpass	$2f_c \frac{\sin(n\omega_c)}{n\omega_c}$	$2f_c$
Highpass	$-2f_c \frac{\sin(n\omega_c)}{n\omega_c}$	$1-2f_c$
Bandpass	$2f_2 \frac{\sin(n\omega_2)}{n\omega_2} - 2f_1 \frac{\sin(n\omega_1)}{n\omega_1}$	$2(f_2 - f_1)$
Bandstop	$2f_1 \frac{\sin(n\omega_1)}{n\omega_1} - 2f_2 \frac{\sin(n\omega_2)}{n\omega_2}$	$1-2(f_2 - f_1)$

The ideal infinite impulse response is truncated by using various windows. Here we multiply the ideal frequency response with a window function. When this window is multiplied by the ideal transfer function then all the coefficients with in the window are retained and all that are outside the window are discarded.

Table2.2 Summary of important features of common window function

Name of window function	Transition width(Hz) (normalized)	Passband Ripple(dB)	Main lobe Relative to side lobe (dB)	Stopband Attenuation (dB)	Window function w(n), |n|<=(N-1)/2
Rectangular	0.9/N	0.7416	13	21	1
Hanning	3.1/N	0.0546	31	44	
Hamming	3.3/N	.0194	41	53	
Kaiser	4.32/N(β=6.76) 5.71/N(β=8.96) 2.93/N(β=4.54)	0.00275 0.000275 0.0274		70 90 50	
Blackman	5.5/N	0.0017	57	75	

2.3.2 Frequency Sampling Method

The frequency sampling method allows us to design nonrecursive FIR filter for both standard frequency filters (lowpass, highpass & bandpass filter) and filter with arbitrary frequency response. A unique attraction of the frequency sampling method is that it also allows recursive implementation of FIR filters.

2.3.2.1 Nonrecursive frequency sampling

To obtain the FIR coefficients of the filter whose frequency response is depicted in Figure 2.3

By taking N samples of the frequency response at intervals of Kf_S/N, k = 0, 1, …, N-1. The filter coefficients $h(n)$ can be obtained as inverse DFT of frequency samples.

$$h(n) = \frac{1}{N}\sum_{k=0}^{N-1} H(k) e^{j(\frac{2\pi}{N})nk} \tag{2.8}$$

where *H(k)*, k = 0, 1, 2,…….., N-1, are sample of the ideal frequency response.

The impulse response coefficients of linear phase FIR filter with positive symmetry, for N even, can be expressed as:

$$h(n) = \frac{1}{N}\left[\sum_{k=1}^{\frac{N}{2}-1} 2\,|H(k)|\cos[2\pi k(n-\alpha)/N] + H(0)\right] \tag{2.9}$$

where α = (N-1)/2, and H(k) are the samples of the frequency response of the filter taken at intervals of kFs/N. For N odd, the upper limit in the summation is (N-1)/2.The resulting filter will have exactly the same frequency response as the original response at the sampling instants. To obtain a good approximation to the desired frequency response, a sufficient number of frequency samples must be taken.

An alternative frequency sampling filter, know as type 2, results if frequency sample taken at intervals of

$$f_k = (k+1/2)F_s / N, \qquad k = 0,1,2,\ldots\ldots\ldots,N-1 \tag{2.10}$$

To improve the amplitude response of frequency samples in the wider transition, introducing frequency samples in the transition band. For a lowpass filter the stopband attenuation increases, approximately, by 20 dB for each transition band frequency sample, with a corresponding increase in the transition width:

Approximate stopband attenuation (25+20M) dB

Approximate transition width $(M+1)F_s/N$

where M is the number of transition band frequency samples and N is the filter length.

2.3.2.2 Recursive frequency sampling

Recursive forms of the frequency sampling offer significant computational advantages over the nonrecursive forms if a large number of frequency samples are zero valued.The transfer function of an FIR filter, $H(z)$, can be expressed in a recursive form:

$$H(z) = \frac{1-Z^{-N}}{N}\sum_{k=0}^{N-1}\frac{H(k)}{1-Z^{-1}e^{j2\pi k/N}} = H_1(z)H_2(z) \tag{2.11}$$

Thus in recursive form, $H(z)$ can be viewed as a cascade of two filters: a comb filter, $H_1(z)$, which has N zeros uniformly distributed around the unit circle, and a sum of N single all-pole filters, $H_2(z)$. The zero of comb filter and the poles of the single pole filters are coincide on the unit circle at points $z_k = e^{2\pi k/N}$.Thus the zero cancel the pole, making $H(z)$ an FIR as it effectively has no poles[4].

In practice, due to finite wordlength effects the poles of $H_2(z)$ not to be located exactly on unit circle so that they are not cancelled by the zeros, making $H(z)$ an IIR and potentially unstable. Stability problems can be avoided by sampling $H(z)$ at a radius, r, slightly less than unity. Thus the transfer function in this case becomes

$$H(Z) = \frac{1 - r^N Z^{-N}}{N} \sum_{k=0}^{N-1} \frac{H(k)}{1 - re^{j2\pi k/N} Z^{-1}} \tag{2.12}$$

In general, the frequency samples, *H(k)*, are complex. Thus direct implementation requires complex arithmetic. To avoid this, the symmetry inherent use in frequency response of any FIR filters with real impulse, $h(n)$. So above equation can expressed as

$$H(z) = \frac{1 - r^N Z^{-N}}{N} \left[\frac{|H(k)| \{2\cos(2\pi k\alpha / N) - 2r\cos[2\pi k(1+\alpha)/N] z^{-1}\}}{1 - 2r\cos(2\pi k / N) Z^{-1} + r^2 z^{-2}} + \frac{H(0)}{1 - Z^{-1}} \right] \tag{2.13}$$

where α= (N-1)/2. For N odd M= (N-1)/2 and for N even M= N/2-1

2.3.3 The optimal method

The optimal method of calculating FIR filter coefficients is very powerful, very flexible and very easy to apply. For this reasons it has become the method of first choice in many FIR applications.

The optimal method is based on the concept of equiripple passband and stopband. Consider the lowpass filter frequency response, in passband the response oscillates between 1- δ_p and 1+ δ_p. In the stopband the filter response lies between 0 and δ_s. The difference between the ideal filter and the practical response can be viewed as an error function:

$$E(\omega) = W(\omega)[H_D(\omega) - H(\omega)]$$

Where $H_D(\omega)$ is the ideal response and $W(\omega)$ is a weighting function that allows the relative error of approximation between different bands to be defined. In optimal method, the objective is to determine the filter coefficients, $h(n)$, such that the value of the weighted error, $|E(\omega)|$, is minimized in the passband and stopband. Mathematically, this may be expressed as: min[max$|E(\omega)|$], over the passbands and stopbands. It has been established that when max$|E(\omega)|$ is minimized the resulting filter response will have equiripple passband and stopband. The minima and maxima are known as extrema. For linear phase lowpass filter, there are either r+1 or r+2 extrema, where r = (N+1)/2 (for type 1 filter) or r =N/2 (for type 2 filter) [4].

For a given set of filter specifications, the location of the extremal frequencies, apart from those at band edges (that is at $f=f_p$ and $f= F_s/2$), are not known a priori. Thus the main problem in the optimal method is to find the locations of the extremal frequencies. A powerful technique which employs the Remez exchange algorithm to find the extremal frequencies has been developed.

By knowing the locations of the extremal frequencies, it is a simple matter to work out the actual frequency response and the impulse response of filter. For given set of specifications the optimal method involves the following key steps:

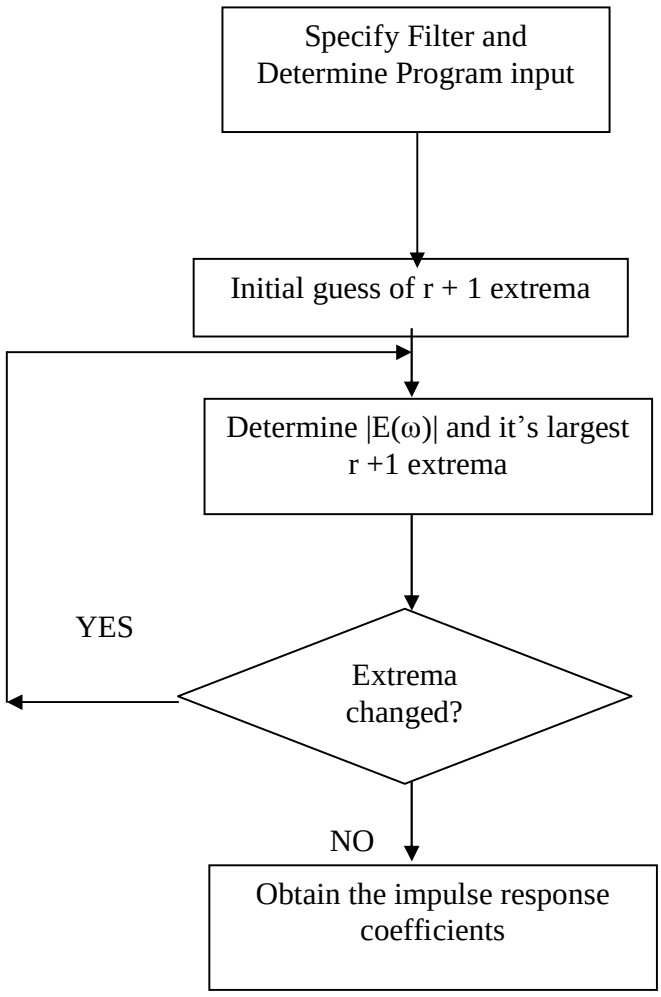


Figure 2.4 Simplified flowchart of the optimal method.

The heart of the optimal method is the first step where an iterative process is used to determine the extremal frequencies of a filter whose amplitude-frequency response satisfies the optimality condition.

2.3.4 Comparison of the window, frequency sampling and optimal methods

The optimum method provides the easy and optimum way of computing FIR filter coefficients. Although the method provides total control of filter specifications, the availability of the optimal filter design software is mandatory. For most applications the optimal method will yield filters with good amplitude response characteristics for reasonable value of N. The method is particularly good for designing Hilbert transformers and differentiators. Other methods will yield larger approximation errors for differentiators and Hilbert transformers than the optimal method.

In the absence of the optimal software or when the passband and stopband ripples are equal, the window method represents a good choice. It is a particularly simple method to apply and conceptually easy to understand. However, the optimal method will often give a more economic solution in terms of the numb of the filter coefficients. The window method does not allow the designer a precise control of the cut off the cutoff frequencies or ripple in the passband and stopband.

The frequency sampling approach is the only method that allows both nonrecursive and recursive implementations of FIR filters, and should be used when such implementations are envisaged as the recursive approach is computationally economical. The special form with integer coefficients should be considered only when primitive arithmetic and programming simplicity are vital, but a check should always be made to see whether its poor amplitude response is acceptable. Filters with arbitrary amplitude-phase response can be readily designed by the frequency sampling method. The frequency sampling method lacks precise control of the location of the band edge frequencies or the passband ripples and relies on the availability of the design[4].

CHAPTER 3

FIR Filter Structures

3.1 Introduction

The analysis of linear, time-invariant FIR filter is generally carried out by using the *Z*-transforms. A brief review of the *Z*-transform is presented. The filter structures characterizing the difference equations are represented using basic elements such as multipliers, time-delays, and adders.

3.1.1 Z Transform

The Z-transform is very useful role in the analysis and characterization of the linear time-invariant systems. This is because the difference equations characterizing the discrete system are transformed into algebraic equations, which are much easier to manipulate.

The two sided Z-transform of discrete-time function ***f****(nT)* is given as

$$F(Z) = \sum_{n=-\infty}^{\infty} f(nT) z^{-n} \tag{3.1}$$

for all *z* for which *F(z)* converges. Here the argument *z* is a complex variable.

Now, evaluating the Z-transform on Equation (1.4) we obtain,

$$z\{y(nT)\} = z\left\{\sum_{i=0}^{N} a_i x(nT - iT)\right\}$$

By using the time translation property and the convolution property of Z-transform, Equation (1.3) can be re-arranged as

$$Y(z) = X(z)\sum_{i=0}^{N} a_i z^{-i}$$

Or, $Y(z) = H(z).X(z)$ where (3.2)

$$H(z) = \sum_{i=0}^{N} a_i z^{-i} \tag{3.3}$$

Where *H (z), X (z), Y (z)* are the Z-transforms of Impulse Response, Input samples and Output samples[1]. *H (z)* is called the transfer function of the filter and the time-domain

samples of this transfer function, which are the filter coefficients are approximated according to the desired response.

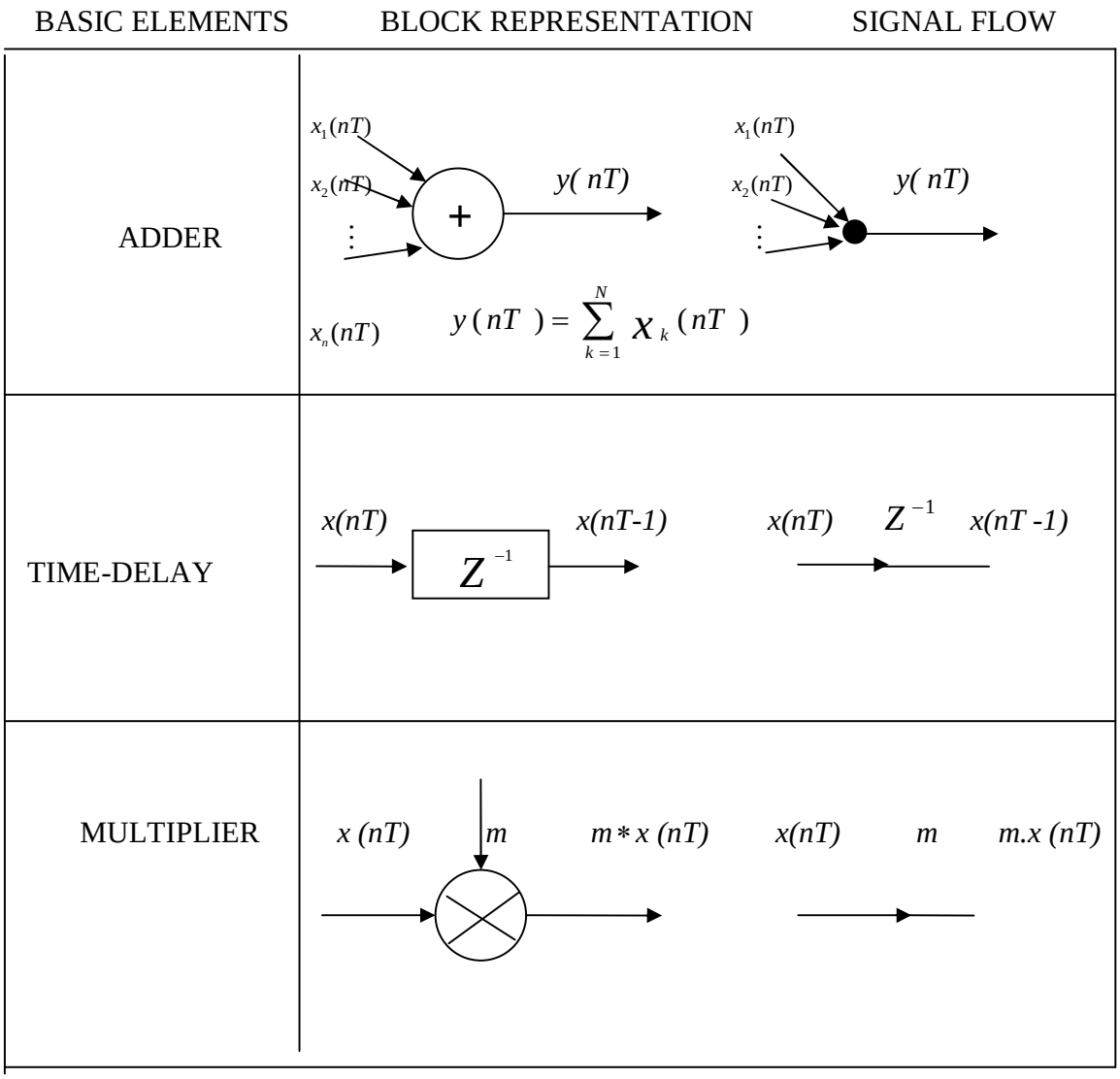


Figure 3.1 Block representation & Signal flow of basic elements.

3.2 Filter Structures

The computational algorithm implementing Equation (1.3) of an FIR filter can be conveniently represented in block diagram. It is done using the basic building blocks elements such as Multipliers, Adders, and Unit Delays. These basic block elements and their equivalent Signal Flow Diagrams are as shown in Figure 3.1.

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

3.2.1 Direct-Form Structure

Direct structures for the Digital filter are those in which the real filter coefficients appear as multipliers in the block diagram representation. If *X(z)* is the filter input and *Y(z)* is the filter output then the transfer function *H(z)* is given as [5]

$$H(z) = \frac{Y(Z)}{X(z)} = \sum_{i=0}^{n} a_i z^{-i} \qquad (3.4)$$

There are four Direct-form structures, which are different realizations of Equation (3.4). The first Direct structure only is presented here and is as shown in Figure 3.2.

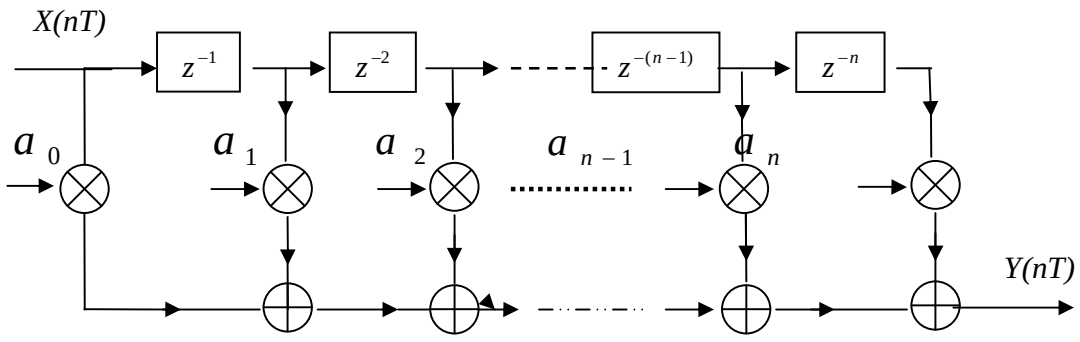


Figure 3.2 Direct-Form of FIR Filter

The 1-D structure is also called canonical because it possesses n-time delay elements. The signal flow diagram of this structure is as shown below in figure 3.3.

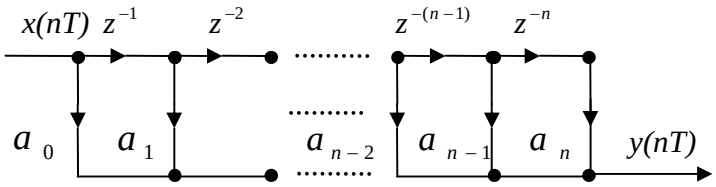


Figure 3.3 Signal flow diagram of Direct-Form

As seen from the Signal Flow Diagram the above representation requires "*n"* Delay elements, *"n + 1"* multipliers and *"n"* adders to implement in the digital computer. The above structure suffers extreme coefficient sensitivity as the value of grows large. That is a small change in a coefficient for large value of *n* causes large changes in the zeroes of *H (z).*

3.2.2 Transpose-form FIR filter structure

The flow-graph-reversal theorem says that if one changes the directions of all the arrows, and inputs at the output and takes the output from the input of a reversed flow-graph, the new system has an identical input-output relationship to the original flow-graph [6].

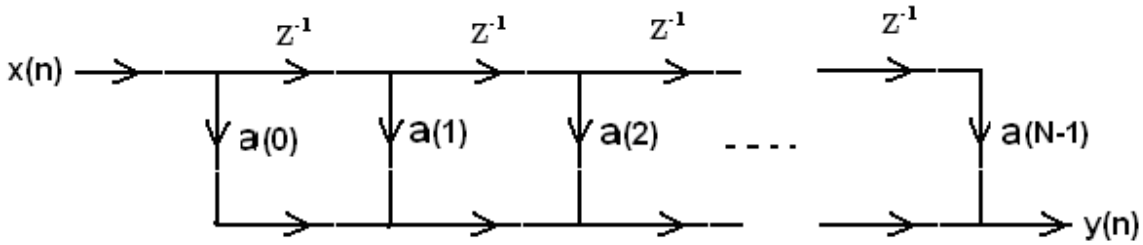


Figure 3.4 Direct form FIR Filter.

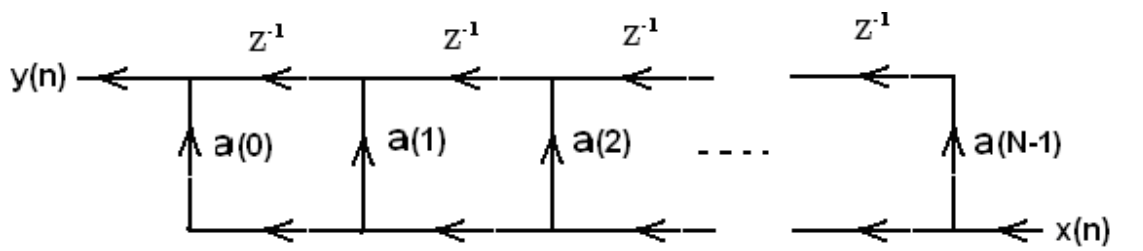


Figure 3.5 Reverse = transpose-form FIR filter structure.

or redrawn

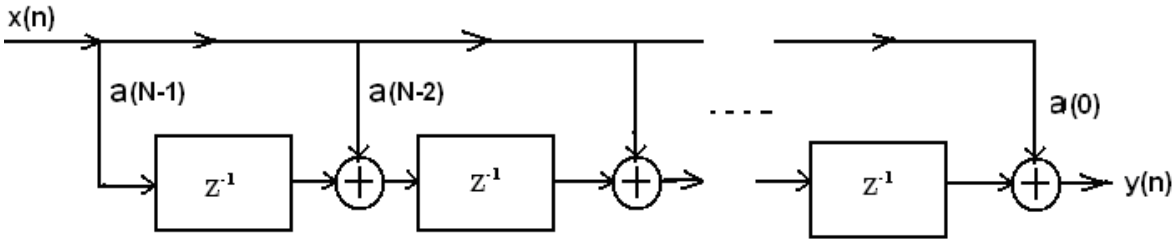


Figure 3.6 Reverse = transpose-form FIR filter structure

3.2.3 Cascade structures

The z-transform of an FIR filter can be factored into a cascade of short-length filters

$$b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_m z^{-m} = b_0 (1 - z_1 z^{-1})(1 - z_1 z^{-1}).....(1 - z_m z^{-1})$$

Where the z_i are the zeros of this polynomial. Since the coefficients of the polynomial are usually real, the roots are usually complex-conjugate pairs, so we generally combine $\left(1 - z_j z^{-1}\right)\left(1 - \bar{z}_j z^{-1}\right)$ into one quadratic (length-2) section with *real* coefficients

$$\left(1 - z_j z^{-1}\right)\left(1 - \bar{z}_j z^{-1}\right) = 1\text{-}2R(Z_j)Z^{-1} + (|Z_j|)^2 Z^{-2} = H_j(Z)$$

The overall filter can then be implemented in a **cascade** structure.

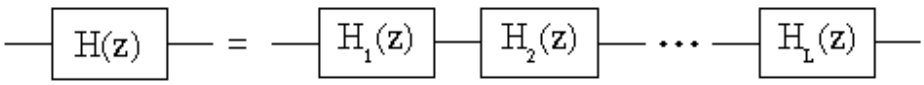


Figure 3.7 Cascaded Structures.

This is occasionally done in FIR filter implementation when one or more of the short-length filters can be implemented efficiently.

3.2.4 Lattice Structure

It is also possible to implement FIR filters in a lattice structure: this is sometimes used in adaptive filtering and digital speech processing.

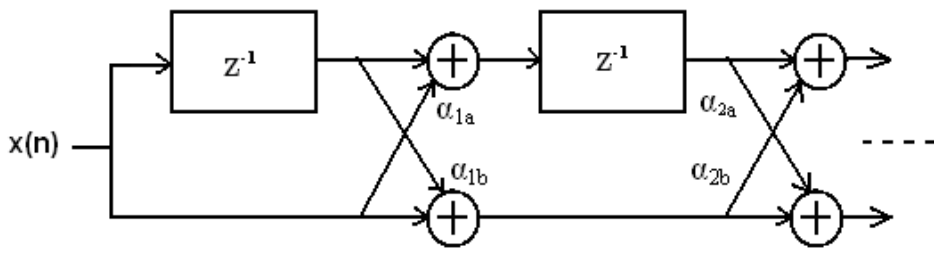


Figure 3.8 Lattice Structure

3.2.5 Comparsion of various structure

The simplest of these structures,namely, the direct-form realizations. However, there are other more practical structures that offer some distinct advantages, especially when quantization effects are taken into consideratoin.

The cascade, parallel, and lattice structures,which exhibit robustness in finite-word-length implementations. The frequency-sampling has the advantage of being computationally efficient when compared with alternative FIR realizations. Other filter structures are obtained by employing a state-space formulation for linear time-invariant system. Due to space limitations,state-space structures are not generally used.

CHAPTER 4
Simulation and Synthesis tools

4.1 Introduction

The following tools for implementation of FIR filter on FPGA:

1. *XILINX ISE web pack 9.2i* for design, synthesis and implementation.
2. MODSIM 5.5c for simulation.

4.2 Simulation Tools

Very High Speed Integrated Circuit Hardware Description Language (VHDL) is used as the designing language. Hardware Description Languages (HDLs) are used to describe the behavior and structure of system and circuit designs. [7]

4.2.1 Advantages of using HDLs to design FPGAs

Top Down Approach – HDLs are used to create complex designs. The top-down approach to system design supported by HDLs is advantageous for large projects that require many designers working together.

Functional Simulation Early in the Design Flow – One can verify the functionality of your design early in the design flow by simulating the HDL description.

Synthesis of HDL Code to Gates – One can synthesize your hardware description to a design implemented with gates. This step decreases design time by eliminating the need to define every gate.

Early Testing of Various Design Implementations – HDLs allows one to test different implementations of your design early in the design flow. One can then use the synthesis tool to perform the logic synthesis and optimization into gates.

Reuse of RTL Code – One can retarget RTL code to new FPGA architectures with a minimum of recoding.

4.2.2 Basics of VHDL

VHDL 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 tools use 1076-1987 version of VHDL, thus making it most compatible when using different compilation tools [8].

VHDL enables the designer to:

- Describe the design in its structure, to specify how it is decomposed into sub-designs, and how these sub-designs are interconnected.
- Specify the function of designs using a familiar, C-like programming language form.
- 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.

VHDL is a C-like, general purpose programming language with extensions to model both concurrent and sequential flows of execution, and allowing delayed assignment of values. To a first approximation, VHDL can be considered to be a combination of two languages: one describing the structure of the integrated circuit and its interconnections (structural description) and the other one describing its behavior using algorithmic constructs (behavioral description).

VHDL allows three styles of programming:

- Structural
- Register Transfer Level (RTL)
- Behavioral

The first one, structural, is the most commonly used as it allows description of the structure of the IC very precisely by the user. This in very many cases gives the best

performance over compiler-optimized structures, especially for high speeds, fixed-point applications like polyphase IIR structures. Its behavioral style permits the designer to quickly test concepts, where the designer can specify the high-level function of the design without taking much care how it will be done structurally. This can be very attractive for quick design of low and medium speed and low-volume applications, where the designer expertise is not available [7].

4.3 Synthesis Tools

4.3.1 XILINX ISE 9.2i Overview

WebPACK ISE 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. We can choose whether to enter the design in schematic form or in HDL, such as VHDL, Verilog.

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 incorporates a Xilinx version of the ModelSim simulator from Model Technology (a Mentor Graphics company), referred to as MXE (ModelSim Xilinx Edition). In a diagrammatic form and let the software tools generate optimized code from a state diagram.

This powerful simulator is capable of simulating functional VHDL before synthesis, or simulating after the implementation process for timing verification. 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 release 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.

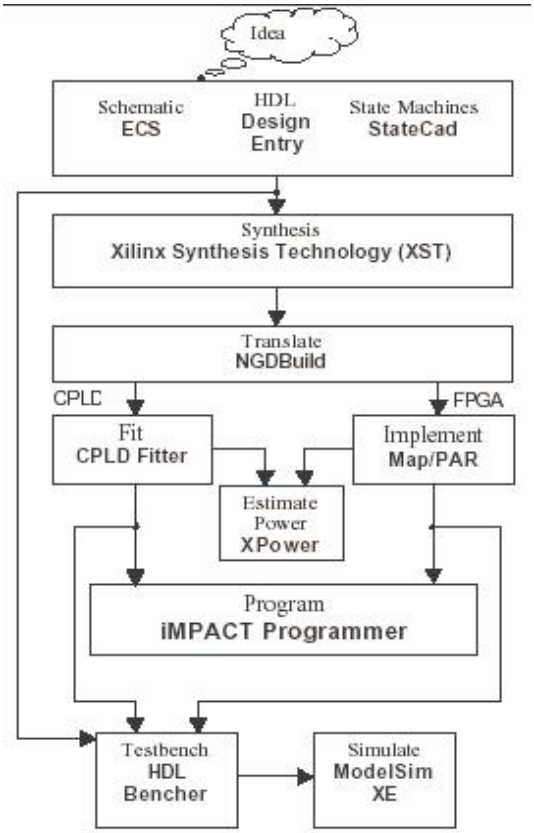


Figure 4.1: Webpack software design flow[11]

The various steps involved are as follows:

- Synthesis: Synthesis is the general term that describes the process of transformation of the model of a design in HDL, from one level of Behavioral 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.

- Implementation: It is divided into three major operations:

  *Translation***:** Merges all of the input net lists

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

*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 tradeoffs 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.

- 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:

    1. iMPACT
    2. PROM File Formatter

The iMPACT programmer module allows you to program a device in-system for all devices available in the WebPACK software. (we must connect a JTAG cable to the PC's parallel port.)

For FPGAs, the programmer module allows you to configure a device via the JTAG cable.

iMPACT, a command line and GUI based tool, allows one to:

I. Configure FPGA designs using Boundary-Scan, Master Serial
II. Download
III. Read-Back and Verify design configuration data
IV. Perform functional tests on any device.

4.4 FPGAs: An Overview

An FPGA is a completely reconfigurable computer logic chip. Like traditional hardwired gate arrays, the chip consists of a series of logic gates. In the traditional array, these gates are specified and hard interconnected at the manufacturing stage. The field programmable gate array differs in that it can be programmed, and re-programmed, in-situ. This has the advantages of allowing fast prototyping for applications it is intended to be implement with hard-wired chips.

4.4.1 Computer Aided Design for VLSI circuits

The design of digital systems with VLSI circuits containing millions of transistors is a formidable task and requires the assistance of computer-aided design (CAD) tools. CAD tools consist of software programs that support computer-based representation and aid in the development of digital hardware by automating the design process. The designer can choose between a full-custom IC, a programmable logic device (PLD), an application specific integrated circuit (ASIC), or a field-programmable gate array (FPGA) [9].

4.4.2 Programmable logic

Programmable logic is loosely defined as a device with configurable logic and flip-flops linked together with programmable interconnect. Memory cells control and define the function that the logic performs and how the various logic functions are interconnected.

What kinds of programmable logic devices are available today? How are they different from one another?

There are a few major programmable logic architectures available today. Each architecture typically has vendor-specific sub-variants. The major types include: [10]

- Simple Programmable Logic Devices (SPLDs)
- Complex Programmable Logic Devices (CPLDs)
- Field Programmable Gate Arrays (FPGAs)

4.4.3 FPGA - Field Programmable Gate Array

The FPGA is advancing rapidly as a highly important element of the future of computing. Already developments have shown that it can massively reduce the price of specialized system development and it can compete on a variety of attributes with the top range commercially available microprocessors.

Its initial role in rapid system prototyping is still important but in more recent times it has grown in importance as a platform for implementing complete solutions. The ability to implement a fully functional system, microcontroller or even full blown computer using FPGAs and the recent advances in development software lead to highly exciting possibilities with regards to the development of complete re-configurable computing systems.

There are four main categories of FPGAs currently commercially available: symmetrical array, row-based, hierarchical PLD, and sea-of-gates (Figure 4.2). In all of these FPGAs the interconnections and how they are programmed vary.

The basic FPGA architecture consists of a two-dimensional array of logic blocks and flip-flops with means for the user to configure (i) the function of each logic blocks, (ii) the inputs/outputs, and (iii) 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.

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.

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.

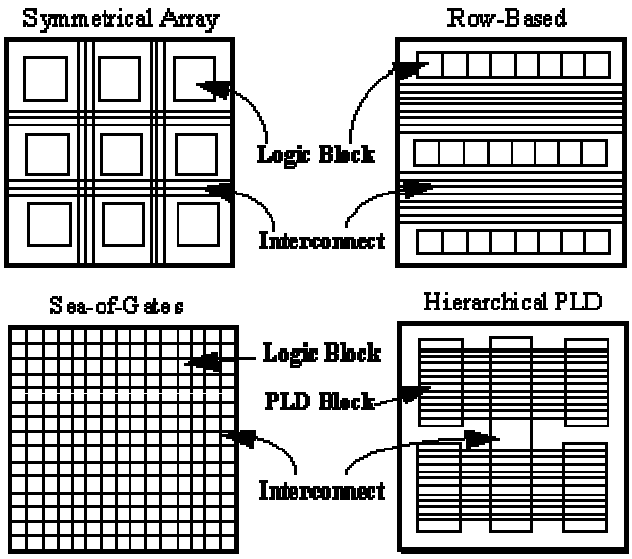


Figure 4.2 Classes of FPGAs [9]

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 re-programmed in-circuit [11].

The following table shows some of the commercially available FPGAs.

Table 4.1 Commercial FPGA Technology

Company Name	**Architecture**	**Logic Block Type**	**Programming Technology**
Actel	Row-based	Multiplexer-Based	anti-fuse
Altera	Hierarchical-PLD	PLD Block	EPROM
QuickLogic	Symmetrical Array	Multiplexer-Based	anti-fuse
Xilinx	Symmetrical Array	Look-up Table	Static RAM

4.5 The Design Flow

Section examines the design flow for any device, whether it is an ASIC, an FPGA, or a CPLD. This is the entire process for designing a device that guarantees that you will not overlook any steps and that you will have the best chance of getting backs a working prototype that functions correctly in your system. The design flow consists of the steps in:

Step1: Writing a Specification

The importance of a specification cannot be overstated. This is an absolute must, especially as a guide for choosing the right technology and for making your needs known to the vendor. As specification allows each engineer to understand the entire design and his or her piece of it. It allows the engineer to design the correct interface to the rest of the pieces of the chip. It also saves time and misunderstanding. There is no excuse for not having a specification.

A specification should include the following information:

- An external block diagram showing how the chip fits into the system.
- An internal block diagram showing each major functional section.
- A description of the I/O pins including
- Output drive capability
- Input threshold level
- Timing estimates including
- Setup and hold times for input pins
- Propagation times for output pins
- Clock cycle time
- Estimated gate count
- Package type
- Target power consumption
- Target price
- Test procedures

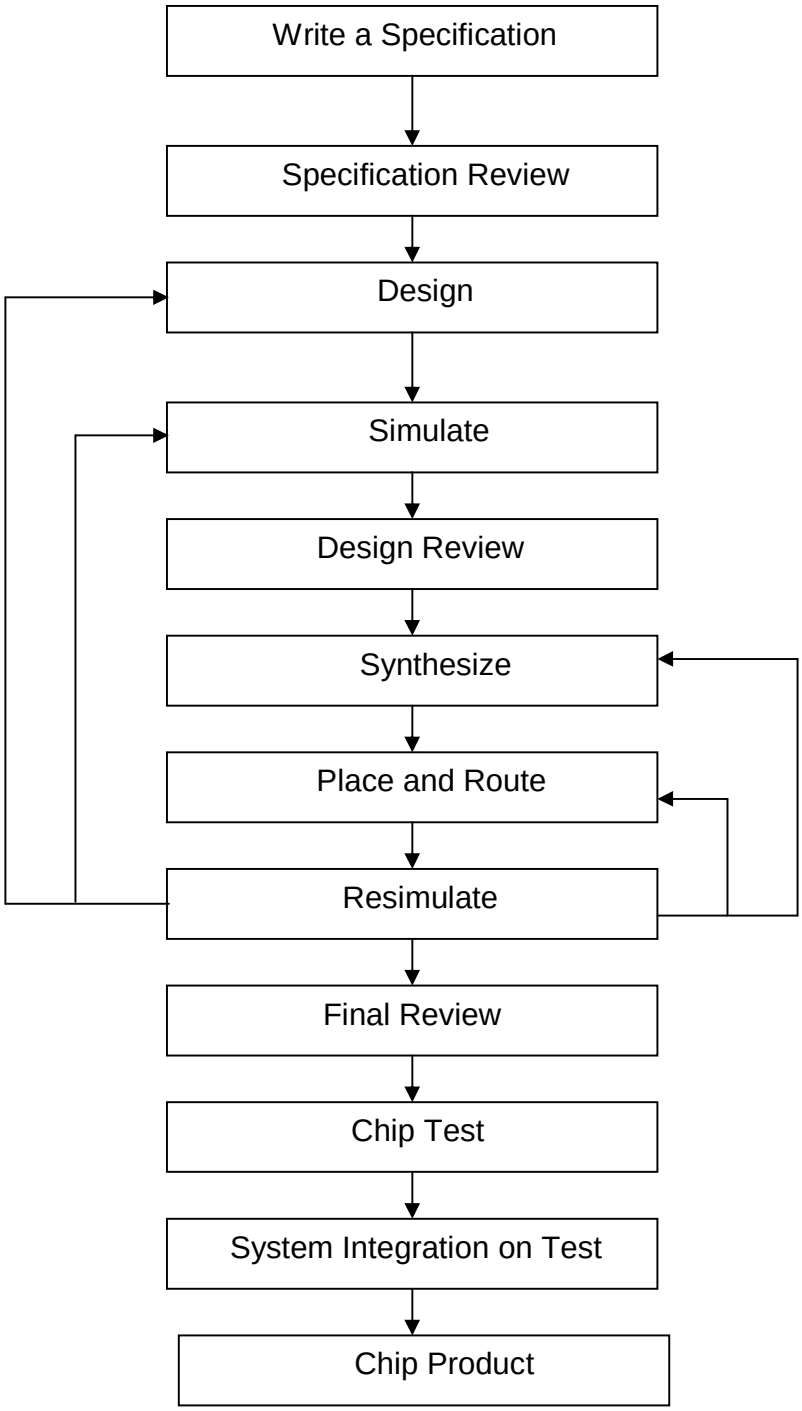


Figure 4.3 FPGA Design Flow

It is also very important to understand that this is a living document. Many sections will have best guesses in them, but these will change as the chip is being designed.

Step2: Choosing a Technology

Once a specification has been written, it can be used to find the best vendor with a technology and price structure that best meets your requirements.

Step3: Choosing a Design Entry Method

You must decide at this point which design entry method you prefer. For smaller chips, schematic entry is often the method of choice, especially if the design engineer is already familiar with the tools. For larger designs, however, a hardware description language (HDL) such as Verilog or VHDL is used because of its portability, flexibility, and readability. When using a high level language, synthesis software will be required to "synthesize" the design. This means that the software creates low level gates from the high level description.

Step4: Choosing a Synthesis Tool

You must decide at this point which synthesis software you will be using if you plan to design the FPGA with an HDL. This is important since each synthesis tool has recommended or mandatory methods of designing hardware so that it can correctly perform synthesis. It will be necessary to know these methods up front so that sections of the chip will not need to be redesigned later on. At the end of this phase it is very important to have a design review. All appropriate personnel should review the decisions to be certain that the specification is correct, and that the correct technology and design entry method have been chosen.

Step5: Designing the chip

It is very important to follow good design practices. This means taking into account the following design issues that we discuss in detail later in this chapter.

- Top-down design
- Use logic that fits well with the architecture of the device you have chosen
- Macros
- Synchronous design
- Protect against metastability
- Avoid floating nodes
- Avoid bus contention

Step6: Simulating - design review: Simulation is an ongoing process while the design is being done. Small sections of the design should be simulated separately before hooking them up to larger sections. There will be much iteration of design and simulation in order to get the correct functionality. Once design and simulation are finished, another design review must take place so that the design can be checked. It is important to get others to look over the simulations and make sure that nothing was missed and that no improper assumption was made. This is one of the most important reviews because it is only with correct and complete simulation that you will know that your chip will work correctly in your system.

Step7: Synthesis

If the design was entered using an HDL, the next step is to synthesize the chip. This involves using synthesis software to optimally translate your register transfer level (RTL) design into a gate level design that can be mapped to logic blocks in the FPGA. This may involve specifying switches and optimization criteria in the HDL code, or playing with parameters of the synthesis software in order to insure good timing and utilization.

Step8: Place and Route

The next step is to lay out the chip, resulting in a real physical design for a real chip. This involves using the vendor's software tools to optimize the programming of the chip to implement the design. Then the design is programmed into the chip.

Step9: Resimulating - final review

After layout, the chip must be resimulated with the new timing numbers produced by the actual layout. If everything has gone well up to this point, the new simulation results will agree with the predicted results. Otherwise, there are three possible paths to go in the design flow. If the problems encountered here are significant, sections of the FPGA may need to be redesigned. If there are simply some marginal timing paths or the design is slightly larger than the FPGA, it may be necessary to perform another synthesis with better constraints or simply another place and route with better constraints. At this point, a final review is necessary to confirm that nothing has been overlooked.

Step10: Testing: For a programmable device, you simply program the device and immediately have your prototypes. You then have the responsibility to place these prototypes in your system and determine that the entire system actually works correctly. If you have followed the procedure up to this point, chances are very good that your

system will perform correctly with only minor problems. These problems can often be worked around by modifying the system or changing the system software. These problems need to be tested and documented so that they can be fixed on the next revision of the chip. System integration and system testing is necessary at this point to insure that all parts of the system work correctly together. When the chips are put into production, it is necessary to have some sort of burn-in test of your system that continually tests your system over some long amount of time. If a chip has been designed correctly, it will only fail because of electrical or mechanical problems that will usually show up with this kind of stress testing.

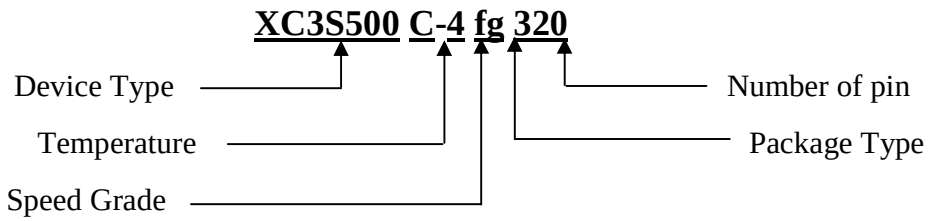
4.6 Spartan-III FPGA kit

The Xilinx Spartan 3E starter board, made by Digilent Inc. uses a XC3S500E FPGA. It has a following feature, such as Flash Memory, DDR SDRAM, LCD display, ADCs, DACs, RS232, VGA, Ethernet Phy and much more. It has a number of 6 pin headers for adding small 4 bit modules, as well as a Hirose 100 pin FX2 connector, which can be used for such things as the VDEC-1 Video digitizer.

The limitation of the Spartan 3E start board is that there is no SRAM, which means we need a DDR-SDRAM controller core to use it unless we are using EDK. There may be a DDR SDRAM controller in ISE somewhere. Also, like the Spartan 3 starter board, the VGA connector only has 3 bits connected to it which means there are only 8 colours which limits it's use in displaying digitized images.

Xilinx Spartan FPGAs are ideal for low-cost, high volume applications. The Spartan-III family is based on IBM and UMC advanced 90 nm, eight layer metal process technology. Xilinx uses 90 nm technology to drive pricing down to under $20 for a one-million-gate FPGA (approximately 17,000 logic cells), which represents a cost savings as high as 80 percent compared to competitive offerings. Our kit was XC3S500C-4fg320 Spartan-III device.

The device, which we are using, has the following specifications:



CHAPTER 5

Implementing of FIR filter on FPGA

5.1 Realization of FIR Filter

The realization of FIR filters can be accomplished by using the following design procedure:

1. Choose filter structure
2. Choose between fixed-point and floating-point arithmetic.
3. Choose number representation, e.g. signed magnitude, two's compliment
4. Choose between serial and parallel processing
5. Implement software code, or hardware circuit, which will perform actual filtering.
6. Verify the simulation that the resulting design meets given performance specifications.

5.2 Process of Implementing FIR filter

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

5.2.1 Restriction and assumption

There are certain assumptions and restrictions in this implementation which are as follows:

1) The input and output are in the digital form.
2) For filter coefficient, used in this thesis are calculated using windowing technique.
3) Filter is considered as symmetric.

5.2.2 Choosing the Filter structure

It is often important to choose a particular filter structure for a given transfer function *H (z)*. In the design of fixed point, digital filters the choice is usually based on minimizing the effects of finite register lengths. These effects include round-off noise, coefficient sensitivity, overflow oscillations, and zero input limit cycles[12].

There are direct-form structures of FIR Filter These Direct structures are effected by coefficient sensitivity problems, which means, for large value of the order of filter the poles (in case of recursive filters) and zeroes locations could be changed. In our project, Direct form of FIR filter has been implemented whose new look is given in Figure 5.1.

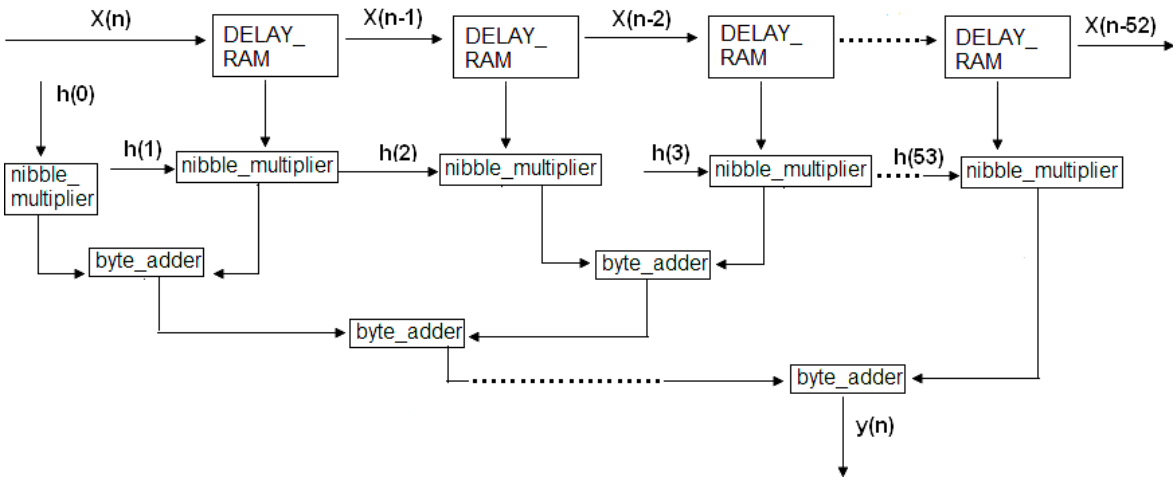


Figure 5.1 Direct-Form Structure

5.2.3 Data Representation

In general, there are two kinds of Data representation, one is fixed-point representation, and the other is IEEE floating-point representation.

In this Thesis, the procedure of representing the filter coefficients and input samples is given as below:

The data is represented in fixed-point notation. In the fixed-point format, the numbers are usually assumed proper fraction. A binary point is usually set between the first and second bit positions of the register as shown in *Figure 5.2* is as given below [13]

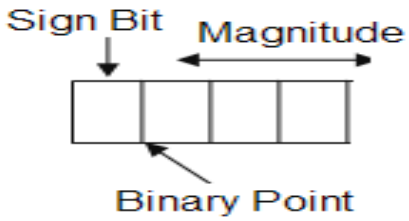


Figure 5.2 Fixed Point Representation

The addition or subtraction of two fixed-point numbers falling in the given range may produce a result outside that range, though. Such a result, called *overflow*, it must be either avoided, or corrected during DSP calculations. We are avoiding here.

5.3 Module for Implementing FIR filter

The modules used for implementing FIR filter are as follows:

5.3.1 Multiplication Module (nibble_multiplier)

It is known that the multiplication operation takes more cycles than an adder or shift register operation. If the number of multiplications in the structure is more, then more time is needed to perform the filtering operation. Thus, the speed of operation will be affected. In our project adders and time-shifters, replace the multipliers, thereby increasing the speed of operation as compared to traditional filter structures in which multipliers were present.

The multiplication is the basic operation in computation of output *y (k)*. Considering the multiplication of two *n-bit* numbers, we have the product of *2n* bits. This way of multiplication is implemented in project while multiplying the coefficients and the input samples.

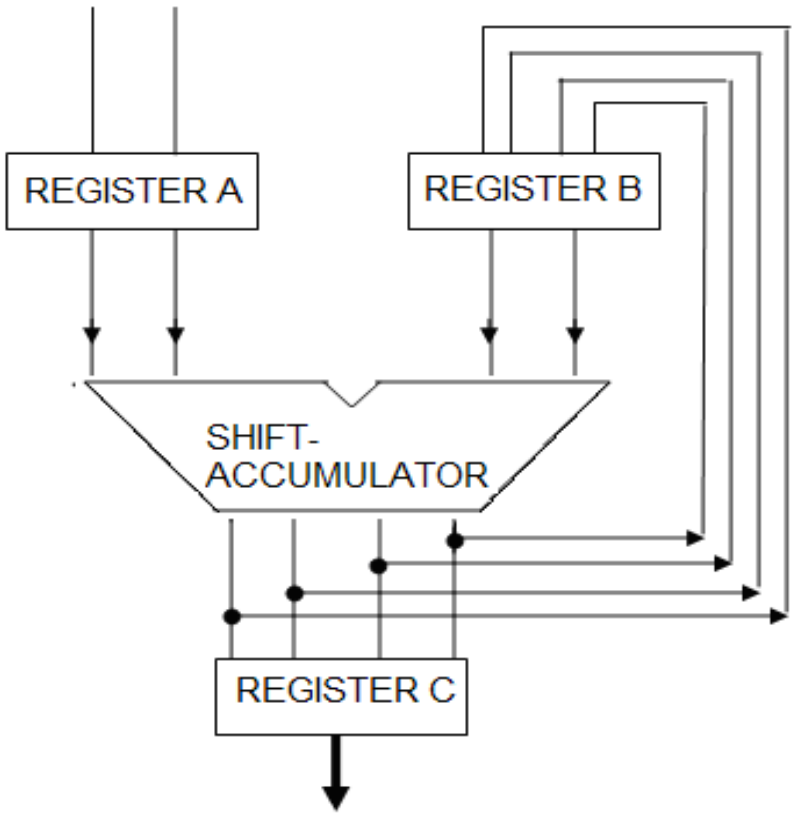


Figure 5.3 Multiplication Modules.

5.3.2 Addition Module (byte_adder)

It will add two numbers with taking care of negative number. If one of them or both are negative, then before addition, negative number will be converted into two's compliment format and then it will be added. It will check whether result is overflowing or not if it is overflowing then it will take two's compliment of result.

5.3.3 Delay and Storing Module (delay_ram)

Input will be shifted to right and result will be storing into temporary register. Shifted input should have array size one more as compare to input but here array size remains constant.

The above modules are used to implement the filter structure and the results are discussed in next chapter.

CHAPTER 6
Results and Discussion

In this chapter, the lowpass, bandpass and highpass filters are implemented on FPGA using real world examples. The filter specifications are real world and windowing method is used to design the filter coefficients. These coefficients are used to implement filter on Xilinx FPGA Spartan 3E kit using Xilinx ISE 9.2i.

6.1 Lowpass Filters

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

Passband edge frequency =1.5 kHz

Transition width =0.5 kHz

Stopband attenuation > 50 dB

Sampling frequency =8 kHz

The filter order of given specification is calculated using Hamming window because stopband attenuation > 50 dB.

$$\Delta f = \frac{0.5}{8} = 0.0625 \qquad (6.1)$$

$$N = \frac{3.3}{\Delta f} = 52.80 \approx 53 \qquad (6.2)$$

The designed coefficients are given in Appendix-I. These coefficient are directly use in VHDL Code. Now this VHDL code is used to generate the circuit using Xilinx synthesis tool for low pass filter design and main circuit block is shown in figure 6.1.

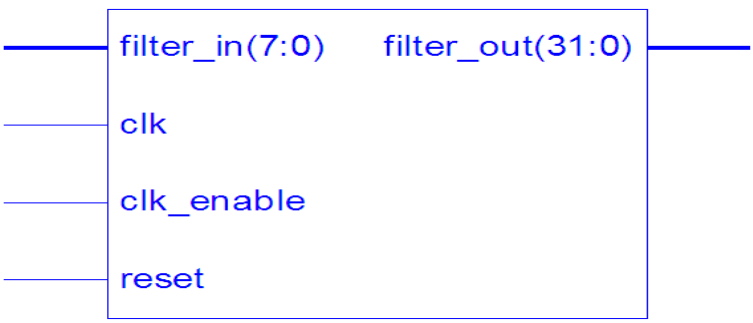


Figure 6.1 Block diagram of the Low Pass Filter (LPF).

Now the generated circuit is simulated using Xilinx simulator with certain inputs (8, 9, and 10) and the corresponding input and output waveform are shown in Fig 6.2 and Fig 6.3.

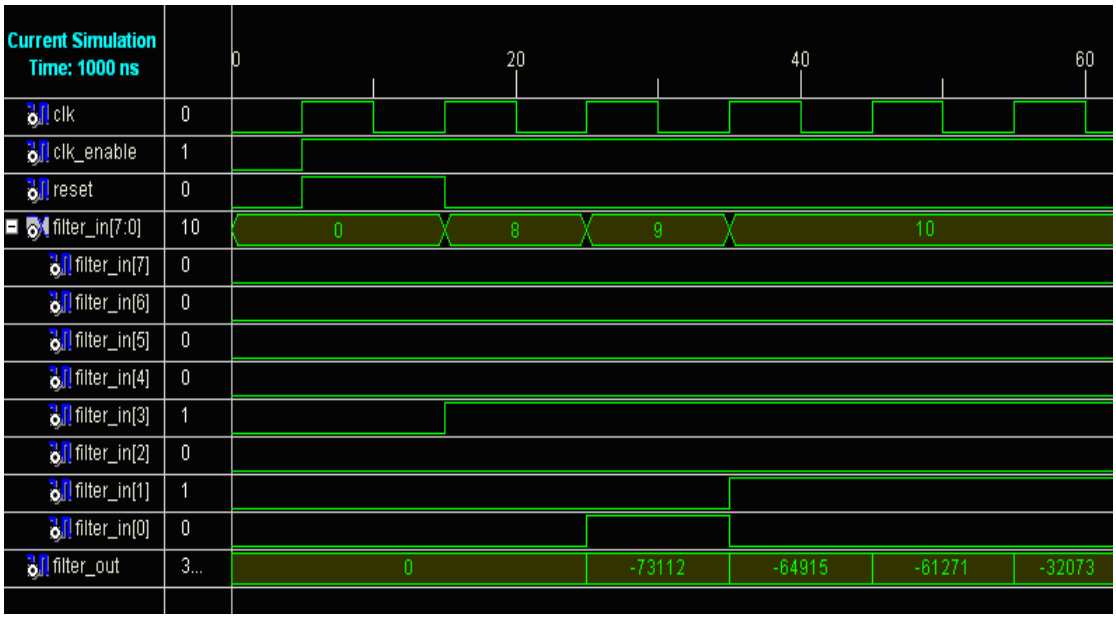


Figure 6.2 Input waveform of LPF.

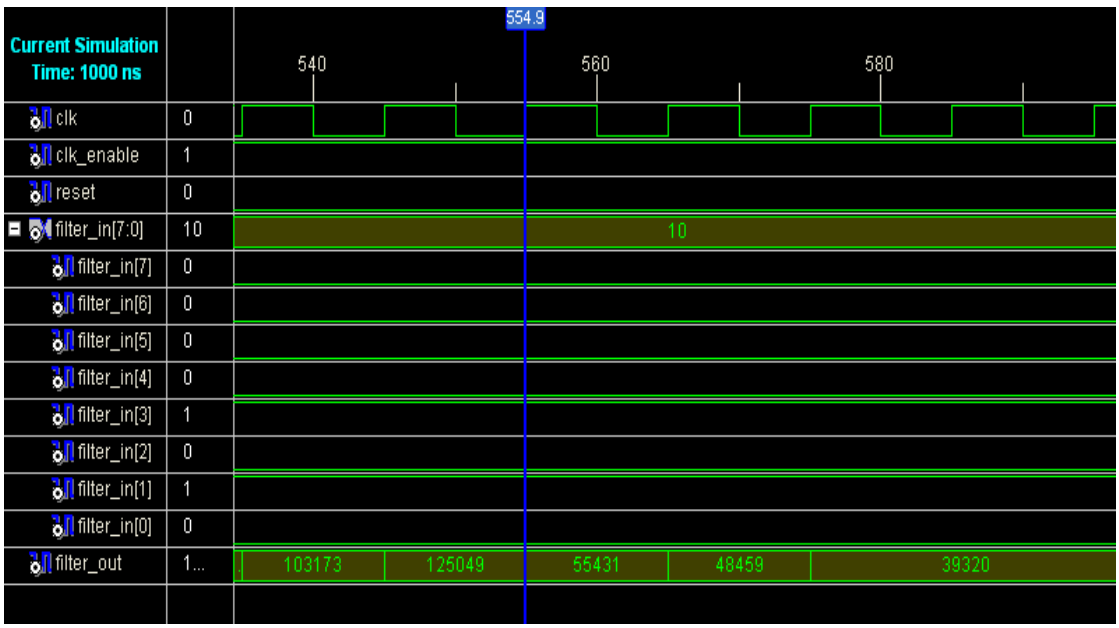


Figure 6.3 Corresponding output waveform of LPF.

Table 6.1 Advanced HDL Synthesis Report of LPF

==========

Macro Statistics:

32x12-bit multiplier	: 20
32x13-bit multiplier	: 14
32x14-bit multiplier	: 14
32x15-bit multiplier	: 6
Total Multipliers	**: 54**
2-bit adder	: 1
3-bit adder	: 1
32-bit adder	: 53
4-bit adder	: 1
5-bit adder	: 1
6-bit adder	: 1
7-bit adder	: 1
8-bit adder	: 1
Total Adders/Subtractors	**: 60**
Flip-Flops	: 464
Total Registers (Flip-Flops)	**: 464**

==========

Table 6.2 Timing Summary of LPF

==========

Speed Grade: -4

Minimum period	: 75.513ns (Maximum Frequency: 13.243MHz)
Minimum input arrival time before clock	: 3.172ns
Maximum output required time after clock	: 4.283ns
Maximum combinational path delay	: No path found

Table 6.3 Thermal summary of LPF

Estimated junction temperature:	27^0C
Ambient temp:	25^0C
Case temp:	26^0C
Theta J-A range:	26 – 26^0C/W

Table 6.4 Power summary of LPF

	I(mA)	P(mW)
Total estimated power consumption		81
Vccint 1.20V	26	31
Vccaux 2.50V	18	45
Vcco25 2.50V	2	5
Clocks	0	0
Inputs	0	0
Logic	0	0
Outputs		
Vcco25	0	0
Signals	0	0
Quiescent Vccint 1.20V	26	31
Quiescent Vccaux 2.50V	18	45
Quiescent Vcco25 2.50V	2	5

Table 6.5 Design summary of Low Pass Filter.

Project File:	prabhat.ise	**Current State:**	Programming File Generated
Module Name:	filter_integer	**• Errors:**	No Errors
Target Device:	xc3s500e-4fg320	**• Warnings:**	59 Warnings
Product Version:	ISE 9.2i	**• Updated:**	Tue Jun 24 01:21:36 2008

PRABHAT Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	424	9,312	4%	
Number of 4 input LUTs	3,806	9,312	40%	
Logic Distribution				
Number of occupied Slices	2,790	4,656	59%	
Number of Slices containing only related logic	2,790	2,790	100%	
Number of Slices containing unrelated logic	0	2,790	0%	
Total Number of 4 input LUTs	4,561	9,312	48%	
Number used as logic	3,806			
Number used as a route-thru	755			
Number of bonded IOBs	43	232	18%	
IOB Flip Flops	40			
Number of GCLKs	1	24	4%	
Number of MULT18X18SIOs	10	20	50%	

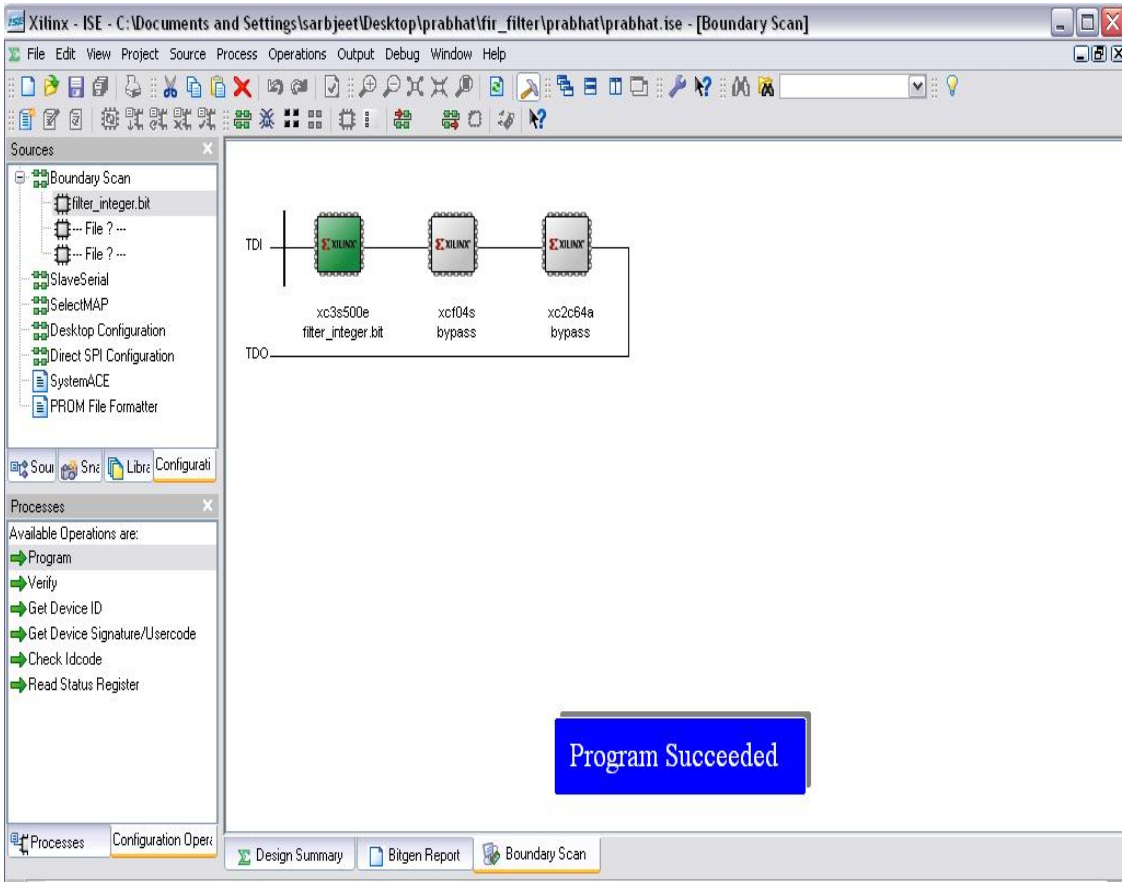


Figure 6.4 Low pass filter burn on FPGA

6.2 Bandpass Filter

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

Passband edge frequency =150 - 250 Hz

Transition width =50 Hz

Passband ripple = 0.1 dB

Stopband attenuation = 60 dB

Sampling frequency =1 kHz

From the specification, the passband and stopband ripples are

$$20\log(1+\delta_p) = 0.1dB\,, \quad \text{giving } \delta_p = 0.0115 \text{ and}$$

$$-20\log(\delta_s) = 60dB, \quad \text{giving } \delta_s = 0.001$$

Thus the attenuation requirements can be met by the Kaiser or the Blackman window. For the Kaisar window, the number of filter coefficients is

$$N \geq \frac{A-7.95}{14.36\Delta F} = \frac{60-7.95}{14.36(50/1000)} = 72.49 \approx 73$$

The designed coefficients are given in Appendix II. These coefficient are directly use in VHDL Code. Now this VHDL code is used to generate the circuit using Xilinx synthesis tool for bandpass filter design and main circuit block is shown in Figure 6.5

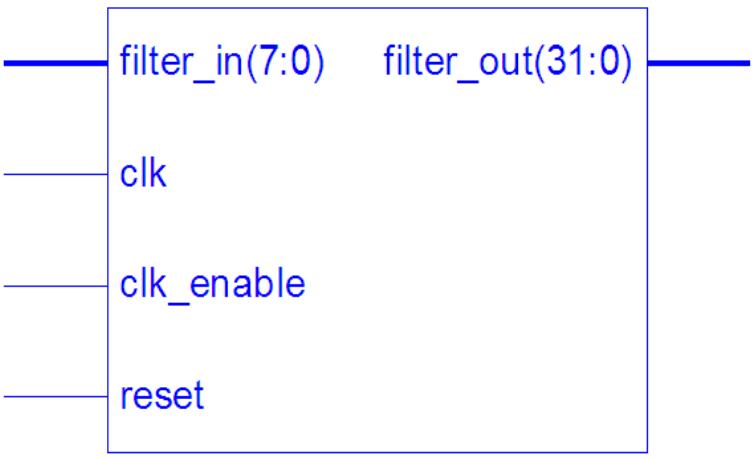


Figure 6.5 Block diagram of the Band Pass Filter (BPF).

Now the generated circuit is simulated using Xilinx simulator with certain inputs (8, 61, 9 and 13) and the corresponding input and output waveform are shown in Fig 6.6 and Fig 6.7

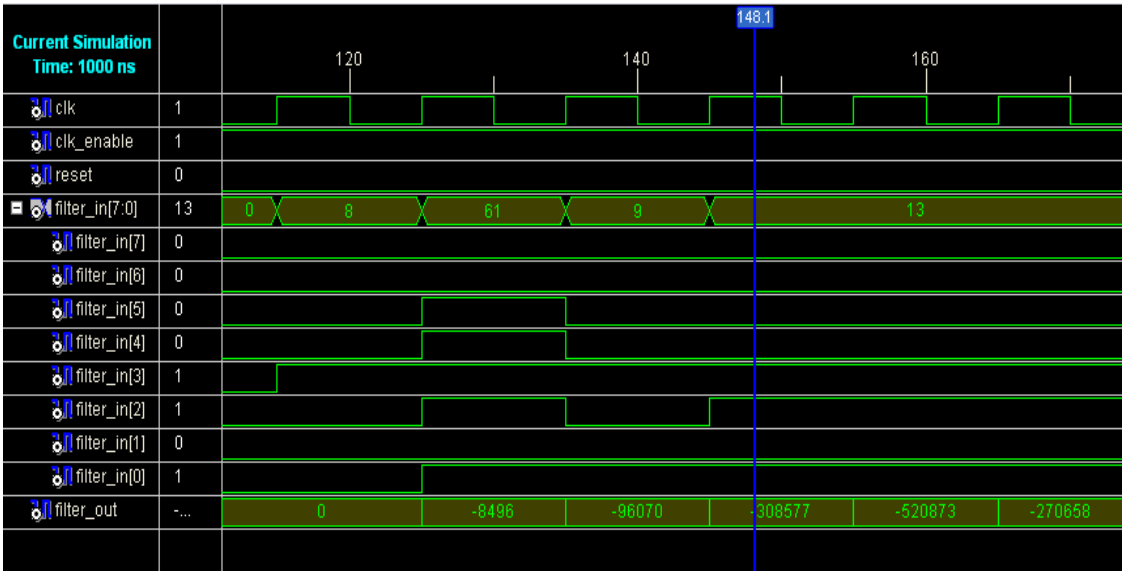


Figure 6.6 Input waveform of BPF

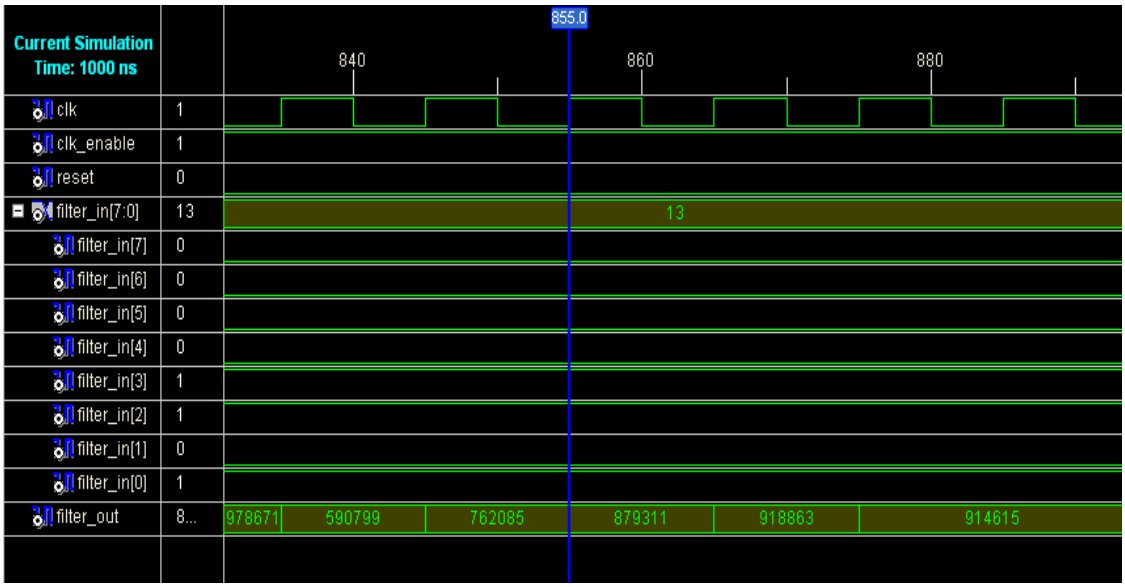


Figure 6.7 Corresponding output waveform of BPF.

Table 6.6 Advanced HDL Synthesis Report 0f BPF

Macro Statistics:

32x12-bit multiplier	: 24
32x13-bit multiplier	: 15
32x14-bit multiplier	: 28
32x15-bit multiplier	: 6
Total Multipliers	**: 73**
2-bit adder	: 1
3-bit adder	: 1
32-bit adder	: 72
4-bit adder	: 1
5-bit adder	: 1
6-bit adder	: 1
7-bit adder	: 1
8-bit adder	: 1
Total Adders/Subtractors	**: 79**
Flip-Flops	: 616
Total Registers(Flip-Flops)	**: 616**

Table 6.7 Timing Summary of BPF

Speed Grade: -4

Minimum period	: 93.191ns (Maximum Frequency: 10.731MHz)
Minimum input arrival time before clock	: 5.176ns
Maximum output required time after clock	: 4.283ns
Maximum combinational path delay	: No path found

Table 6.8 Thermal summary of BPF

Estimated junction temperature:	27^0C
Ambient temp:	25^0C
Case temp:	26^0C
Theta J-A range:	$26 – 26^0$C/W

Table 6.9 Power summary of BPF

	I(mA)	P(mW)
Total estimated power consumption		81
Vccint 1.20V	26	31
Vccaux 2.50V	18	45
Vcco25 2.50V	2	5
Clocks	0	0
Inputs	0	0
Logic	0	0
Outputs		
Vcco25	0	0
Signals	0	0
Quiescent Vccint 1.20V	26	31
Quiescent Vccaux 2.50V	18	45
Quiescent Vcco25 2.50V	2	5

Table 6.10 Design summary of Band Pass Filter

PRABHAT1 Project Status			
Project File:	prabhat1.ise	**Current State:**	Programming File Generated
Module Name:	filter_integer1	**• Errors:**	No Errors
Target Device:	xc3s500e-4fg320	**• Warnings:**	135 Warnings
Product Version:	ISE 9.2i	**• Updated:**	Tue Jun 24 03:22:36 2008

PRABHAT1 Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	576	9,312	6%	
Number of 4 input LUTs	5,481	9,312	58%	
Logic Distribution				
Number of occupied Slices	3,908	4,656	83%	
Number of Slices containing only related logic	3,908	3,908	100%	
Number of Slices containing unrelated logic	0	3,908	0%	
Total Number of 4 input LUTs	6,381	9,312	68%	
Number used as logic	5,481			
Number used as a route-thru	900			
Number of bonded IOBs	43	232	18%	
IOB Flip Flops	40			
Number of GCLKs	1	24	4%	
Number of MULT18X18SIOs	10	20	50%	
Total equivalent gate count for design	65,298			
Additional JTAG gate count for IOBs	2,064			

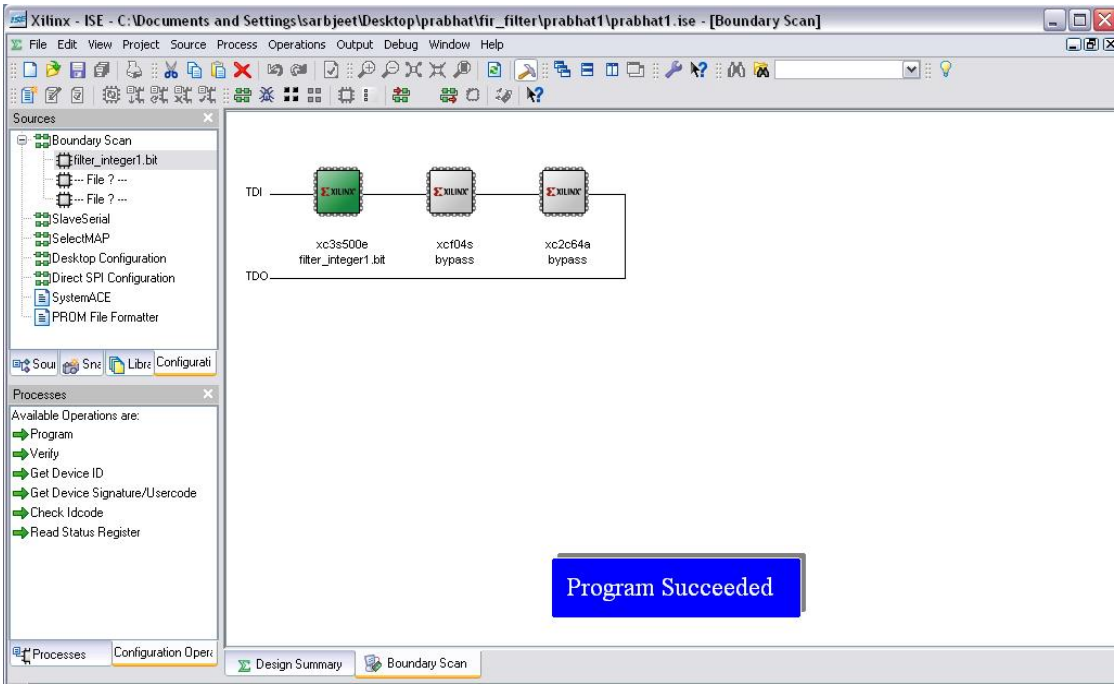


Figure 6.8 Band pass filter burn on FPGA

6.3 Highpass Filter

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

Passband edge frequency = 10 kHz

Transition width =0.5 kHz

Passband ripple = 0.1 dB

Stopband attenuation = 60 dB

Sampling frequency =8 kHz

The filter order of given specification is calculated using Hamming window technique as;

$$\Delta f = \frac{0.5}{8} = 0.0625 \qquad (6.3)$$

$$N = \frac{3.3}{\Delta f} = 52.80 \approx 53 \qquad (6.4)$$

The designed coefficients are given in Appendix III. These coefficient are directly use in VHDL Code. Now this VHDL code is used to generate the circuit using Xilinx synthesis tool for highpass filter design and main circuit block is shown in figure 6.9

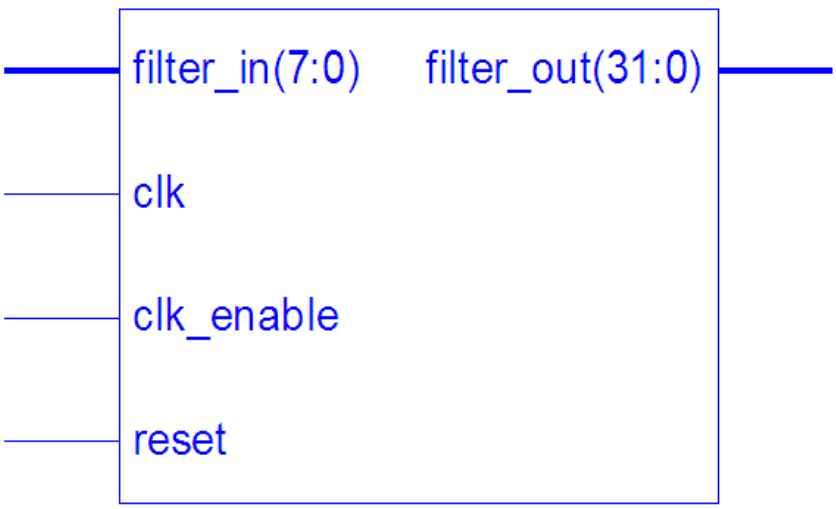


Figure 6.9 Block diagram of the High Pass Filter (HPF).

Now the generated circuit is simulated using Xilinx simulator with certain inputs (8, 9, 10, 11 and 12) and the corresponding input and output waveform are shown in Fig 6.10 and Fig 6.11

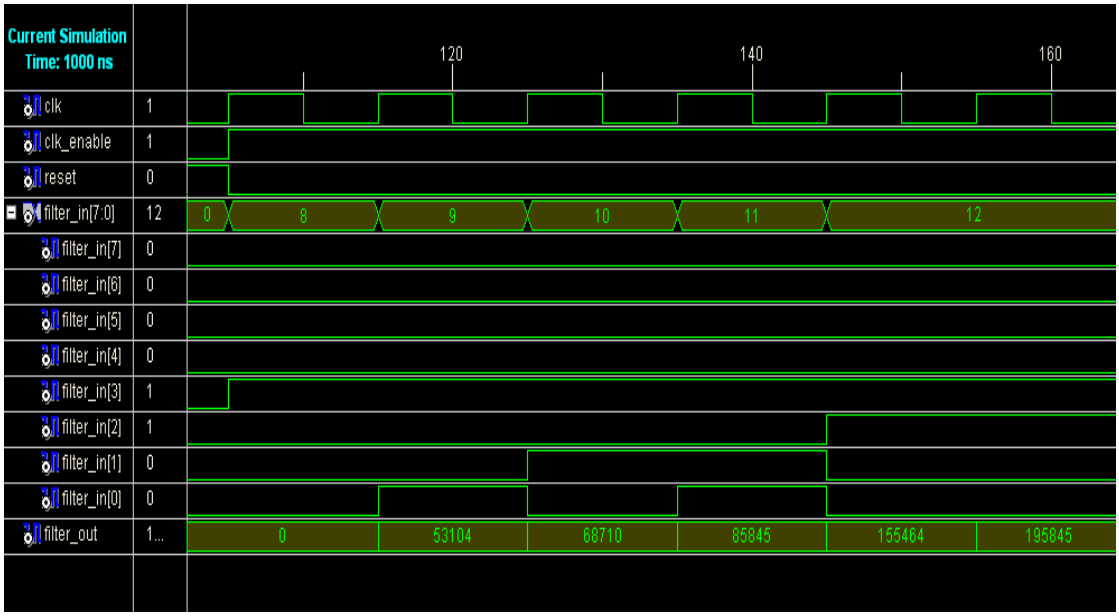


Figure 6.10 Input waveform of HPF

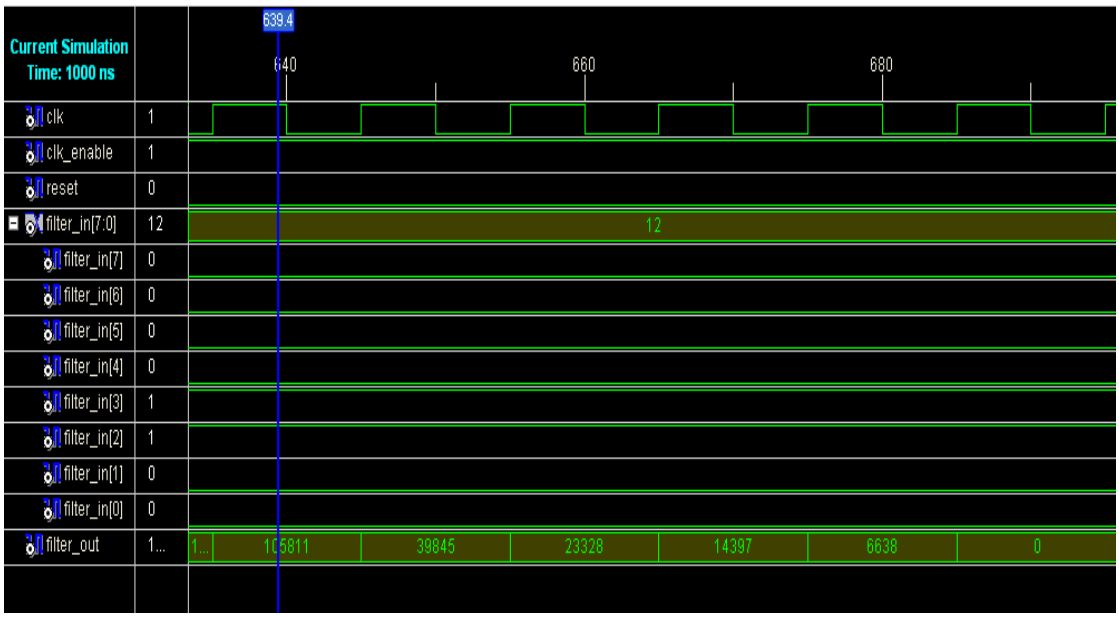


Figure 6.11 Corresponding output waveform of HPF.

Table 6.11 Advanced HDL Synthesis Report of HPF

==========

Macro Statistics:

32x12-bit multiplier	: 20
32x13-bit multiplier	: 14
32x14-bit multiplier	: 14
32x15-bit multiplier	: 6
Total Multipliers	**: 54**
2-bit adder	: 1
3-bit adder	: 1
32-bit adder	: 53
4-bit adder	: 1
5-bit adder	: 1
6-bit adder	: 1
7-bit adder	: 1
8-bit adder	: 1
Total Adders/Subtractors	**: 60**
Flip-Flops	: 464
Total Registers (Flip-Flops)	**: 464**

==========

Table 6.12 Timing Summary of HPF

==========

Speed Grade: -4

Minimum period : 75.513ns (Maximum Frequency: 13.243MHz)
Minimum input arrival time before clock : 3.172ns
Maximum output required time after clock : 4.283ns
Maximum combinational path delay : No path found

Table 6.13 Thermal summary of HPF

Estimated junction temperature:	27^0C
Ambient temp:	25^0C
Case temp:	26^0C
Theta J-A range:	26 – 26^0C/W

Table 6.14 Power summary of HPF

	I(mA)	P(mW)
Total estimated power consumption		81
Vccint 1.20V	26	31
Vccaux 2.50V	18	45
Vcco25 2.50V	2	5
Clocks	0	0
Inputs	0	0
Logic	0	0
Outputs		
Vcco25	0	0
Signals	0	0
Quiescent Vccint 1.20V	26	31
Quiescent Vccaux 2.50V	18	45
Quiescent Vcco25 2.50V	2	5

Table 6.15 Design summary of High Pass Filter

PRABHAT2 Project Status			
Project File:	prabhat2.ise	**Current State:**	Programming File Generated
Module Name:	filter2	**• Errors:**	No Errors
Target Device:	xc3s500e-4fg320	**• Warnings:**	159 Warnings
Product Version:	ISE 9.2i	**• Updated:**	Wed Jun 25 02:03:16 2008

PRABHAT2 Partition Summary
No partition information was found.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	424	9,312	4%	
Number of 4 input LUTs	4,008	9,312	43%	
Logic Distribution				
Number of occupied Slices	2,854	4,656	61%	
Number of Slices containing only related logic	2,854	2,854	100%	
Number of Slices containing unrelated logic	0	2,854	0%	
Total Number of 4 input LUTs	4,688	9,312	50%	
Number used as logic	4,008			
Number used as a route-thru	680			
Number of bonded IOBs	43	232	18%	
IOB Flip Flops	40			
Number of GCLKs	1	24	4%	
Number of MULT18X18SIOs	10	20	50%	
Total equivalent gate count for design	47,523			
Additional JTAG gate count for IOBs	2,064			

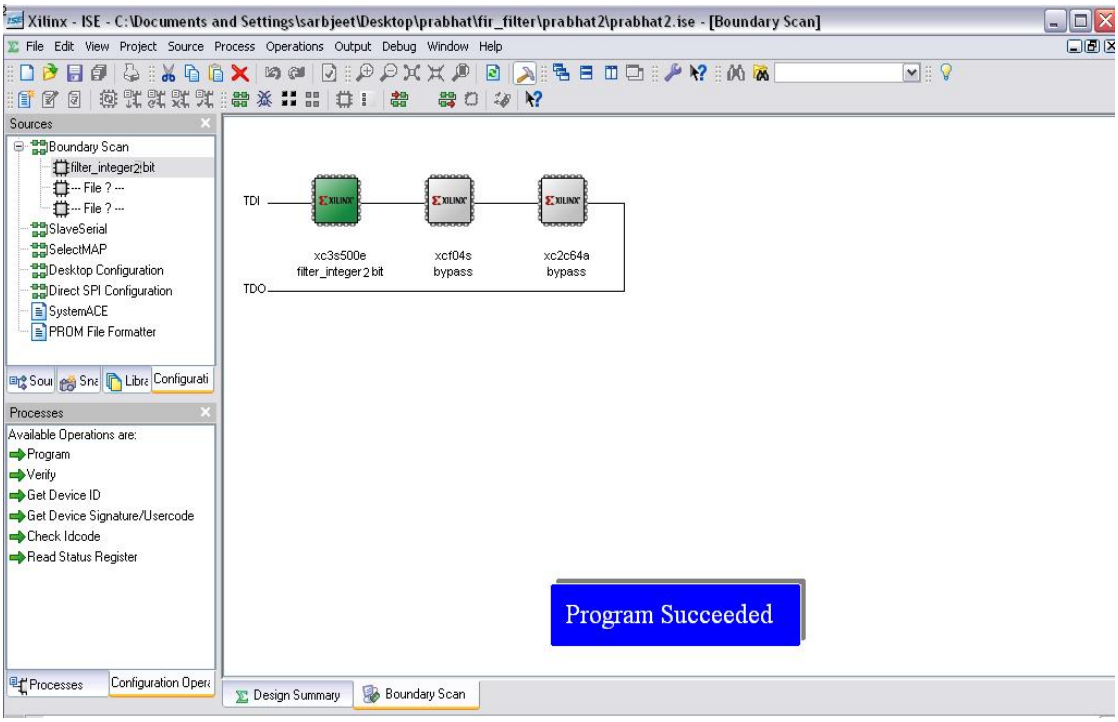


Figure 6.12 High pass filter burn on FPGA

6.4 Discussions

The multipliers used for lowpass and highpass filters are 54 because they have equal number of coefficients and that for highpass filter are 74. Similarly all other components e.g. adders and registers are same for lowpass filter and highpass filter. The adder for lowpass, highpass filter are 60 and for bandpass filter are 79. The registers are 464 for lowpass, highpass and 616 for bandpass filter.

The minimum period for lowpass, highpass filter are 75.51 ns (maximum frequency: 13.243 MHz) and for bandpass filter is 93.191 ns (maximum frequency: 10.731 MHz). power consumed by the all three types of filters are 81 mW. The estimated junction temperature of all three types of filter is 27^0C. The total number of 4 input LUTs for lowpass, bandpass and highpass filters are 4561, 6381 and 4688 respectively. Total gates used in design implementation are 47523, 65298, 47523, for lowpass, bandpass and highpass filters respectively.

CHAPTER 7
Conclusion and Future Scope of Work

7.1 Conclusion

The FIR filters are widely used in digital signal processing and can be implemented using programmable digital processors. But in the realization of large order filters the speed, cost, and flexibility is affected because of complex computations. So,the implementation of FIR filters on FPGAs is the need of the day because FPGAs can give enhanced speed. 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 thesis, a fifty three-order low-pass and highpass and seventy-three order band pass FIR filter is implemented in Spartan-III-xc3s500c-4fg320 FPGA. The Direct-form structure of these filters are implemented. This approach gives a better performance than the common filter structures in terms of speed of operation, cost, and power consumption. In the Direct-form structure, N Shift Registers, N Adder and N+1 multipliers are used to realize the N order lowpass , high pass and bandpass filter. The designed filters can work for real time processing of any digital signal.

7.2 Future Scope of Work

The future scope of this work includes the following:

- The A/D and D/A converter can be interfaced within the FPGA.

- The optimization of the design can be done in terms of area occupied on chip.

References

[1] http://www.dsptutor.freeuk.com/dfilt1.htm

[2] Carmelina Ruggiero, "SEGNALI BIOMEDICI 1" Laboratorio MedInfo

[3] "An Introduction to Digital Filters" by INTERSIL, Application Note, January 1999

[4] Ifeachor E.C., Jervis B.W., "Digital Signal Processing",2nd Edition, Low Price Edition 2007.

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

[6] Jones D. L., "FIR Filter Structures", Version 1.2: Oct 10, 2004.

[7] Perry D., "VHDL", 3rd Edition, Tata Mc. Graw Hill Publications, 2001.

[8] Bhaskar J., "VHDL primer" ,3rd Edition, Pearson Education Asia Publications, 2000.

[9] Chen W. K., " Logic Design",CRC Press, 2000.

[10] Wakerly J. F., "Digital Design & Practice"; Pearson Education Asia 3rd edition

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

[12] Burrus C S, "Digital Filters Structures described by Distributed Arithmetic", IEEE Transactions on Circuits and Systems, vol. CAS-24, page: 12, December 1977.

[13] http://en.wikipedia.org/wiki/floating_point

[14] Parhi K K., "A Systematic Approach for Design of Digit-serial Signal Processing Architectures", Circuits and Systems, 1991.

[15] Prokis J. G., Manolakis D. G., "Digital Signal Processing", 3rd Edition, PHI publication 2004.

[16] Antoniou A.," Digital Filter", 3rd Edition, Tata Mc. Graw Hill publications, 2001.

[17] Mitra S. K., "Digital Signal Processing" 3rd Edition, Tata Mc. Graw Hill Publications.

[18] Chapman S. J., " Matlab Programming for Engineers", 3rd Edition, Thomson learning 2005.

[19] Lee H., Sobelman G E. "Performance Evaluation and Optimal design for FPGA-based Digit-serial DSP Functions". Computers and Electrical Engineering 29 ,2003

[20] Mirzaei S., Hosangadi A. and Kastner R. , "FPGA Implementation of High Speed FIR Filters Using Add and Shift Method", International Conference on Computer Design (ICCD),pp 308-313, 2006

[21] Takahashi Y. and Yokoyama M., "New cost-effective VLSI implementation of multiplierless FIR filter using common subexpression elimination", Proc. IEEE Int. Symp. on Circuits and Systems (ISCAS 2005), pp.845–848, May 2005.

[22] Rocha Ed., "Implementation trade-offs of digital FIR filters," Military Embeded System, open system publishing,2007.

[23] Choi, Seak C. and Lee H., "A Partial Self-Reconfigurable Adaptive FIR Filter System," signal processing systems,pp.204-209,2007.

[24] Takalashi Y., Sekine T. and Yokoyama M., "A 70MHz Multiplierless FIR Hilbert Transformer in 0.35 um standard CMOS Library," IEICE Trans. 1376-1383,2007.

Appendix-I

FIR coefficients for lowpass filter:

h[0]=	-9.1399895e-04	=h[52]
h[1]=	2.1673690e-04	=h[51]
h[2]=	1.3270280e-03	=h[50]
h[3]=	3.2138355e-04	=h[49]
h[4]=	-1.9238177e-03	=h[48]
h[5]=	-1.4683633e-03	=h[47]
h[6]=	2.3627318e-03	=h[46]
h[7]=	3.4846558e-03	=h[45]
h[8]=	-1.9925839e-03	=h[44]
h[9]=	-6.2837282e-03	=h[43]
h[10]=	4.5320247e-09	=h[42]
h[11]=	9.2669460e-03	=h[41]
h[12]=	4.3430586e-03	=h[40]
h[13]=	-1.1271299e-02	=h[39]
h[14]=	-1.1402453e-02	=h[38]
h[15]=	1.0630714e-02	=h[37]
h[16]=	2.0964392e-02	=h[36]
h[17]=	-5.2583216e-03	=h[35]
h[18]=	-3.2156086e-02	=h[34]
h[19]=	-7.5449714e-03	=h[33]
h[20]=	4.3546153e-02	=h[32]
h[21]=	3.2593190e-02	=h[31]
h[22]=	-5.3413653e-02	=h[3o]
h[23]=	-8.5682029e-02	=h[29]
h[24]=	6.0122145e-02	=h[28]
h[25]=	3.1118568e-01	=h[27]
h[26]=	4.3750000e-01	=h[26]

Appendix-II

FIR coefficients for bandpass filter:

h[0]=	-1.0627330e-04	=h[72]
h[1]=	-3.9118142e-04	=h[71]
h[2]=	-7.5561629e-05	=h[70]
h[3]=	-1.3695577e-04	=h[69]
h[4]=	-6.8122013e-04	=h[68]
h[5]=	5.0929290e-04	=h[67]
h[6]=	2.3413494e-03	=h[66]
h[7]=	8.0280013e-04	=h[65]
h[8]=	-1.7031635e-04	=h[64]
h[9]=	-5.5034956e-04	=h[63]
h[10]=	-4.9912488e-04	=h[62]
h[11]=	-4.4036355e-03	=h[61]
h[12]=	-2.1639856e-03	=h[60]
h[13]=	6.9094151e-03	=h[59]
h[14]=	6.6067599e-03	=h[58]
h[15]=	-1.6445200e-03	=h[57]
h[16]=	4.5229777e-09	=h[56]
h[17]=	2.1890066e-03	=h[55]
h[18]=	-1.1720511e-02	=h[54]
h[19]=	-1.6377726e-02	=h[53]
h[20]=	6.8804519e-03	=h[52]
h[21]=	1.8882837e-02	=h[51]
h[22]=	2.9068601e-03	=h[50]
h[23]=	4.3925286e-03	=h[49]
h[24]=	1.8839744e-02	=h[48]
h[25]=	-1.2481155e-02	=h[47]
h[26]=	-5.2063428e-02	=h[46]
h[27]=	-1.6557375e-02	=h[45]
h[28]=	3.3298453e-02	=h[44]
h[29]=	1.0439025e-02	=h[43]
h[30]=	9.4320244e-03	=h[42]
h[31]=	8.5673629e-02	=h[41]
h[32]=	4.5314758e-02	=h[40]
h[33]=	-1.6657147e-01	=h[39]
h[34]=	-2.0669512e-01	=h[38]
h[35]=	8.9135544e-02	=h[37]
h[36]=	3.0000000e-01	=h[36]

Appendix-III

FIR coefficients for highpass filter:

h[0]=	6.6389895e-04	=h[52]
h[1]=	1.1213670e-04	=h[51]
h[2]=	1.1720280e-03	=h[50]
h[3]=	7.5868355e-04	=h[49]
h[4]=	2.9838177e-03	=h[48]
h[5]=	-1.4233633e-03	=h[47]
h[6]=	-1.5017318e-03	=h[46]
h[7]=	-9.8166558e-03	=h[45]
h[8]=	-6.1825839e-03	=h[44]
h[9]=	1.1077282e-03	=h[43]
h[10]=	1.9120247e-09	=h[42]
h[11]=	2.0509460e-03	=h[41]
h[12]=	1.3640586e-03	=h[40]
h[13]=	8.4521299e-02	=h[39]
h[14]=	-1.6082453e-02	=h[38]
h[15]=	-2.8550714e-02	=h[37]
h[16]=	-3.1614392e-02	=h[36]
h[17]=	-2.1873216e-03	=h[35]
h[18]=	-1.0606086e-02	=h[34]
h[19]=	2.8679714e-03	=h[33]
h[20]=	5.4896153e-02	=h[32]
h[21]=	6.7153190e-02	=h[31]
h[22]=	5.3403653e-02	=h[3o]
h[23]=	3.1162029e-02	=h[29]
h[24]=	-1.2472145e-02	=h[28]
h[25]=	-6.0548568e-01	=h[27]
h[26]=	-5.3750000e-01	=h[26]