FPGA IMPLEMENTATION OF DIGITAL FIR FILTER DESIGN

Thesis submitted in partial fulfilment of the requirement for the award of the degree of

MASTER OF TECHNOLOGY
in
VLSI DESIGN & CAD

Submitted By

TANVEET KAUR

Regn. No. 600961022

Under the guidance of

Mr. SANJAY KUMAR

Assistant Professor

Department of Electronics & Communication Engineering
THAPAR UNIVERSITY
PATIALA – 147001
INDIA

July, 2011

CERTIFICATE

I hereby certify that the work which is being presented in this thesis entitled **"FPGA Implementation of Digital FIR Filter Design"** in partial fulfillment of requirements for the award of degree of **Master of Engineering in VLSI Design & CAD** Thapar University, Patiala, is an authentic record of my own work carried under the supervision of Mr. Sanjay Kumar, Assistant Professor, Department of Electronics and Communication Engineering

The matter embodied in this thesis has not been submitted in any other University / Institute for the award of Master of Engineering.

Tanveet

(Tanveet Kaur)
Signature of the Student

Date 11/07/2011
Place Patiala

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

Date
Place

(Mr. Sanjay Kumar)
Thesis Supervisor
Assistant Professor, ECED

Countersigned

(Dr. A.K. Chaterjee)
Head of Department
ECED, T.U. Patiala

(Dr. S. K. Mohapatra)
Dean of Academic Affairs
T.U. Patiala

ACKNOWLEDGEMENTS

Words are often to less to reveal one's deep regards. An understanding of the work like this is never the outcome of the efforts of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis.

First, I would like to thank the Supreme Power, one who has always guided me to work on the right path of the life. Without his grace, this would never come to be today's reality.

I wish to express earnest acknowledgement, indebtness and gratitude to my respected Thesis Supervisor **Mr. Sanjay Kumar, Assistant Professor**, for his patient guidance and support throughout this thesis. I am truly fortunate to have the opportunity to work with him. I found his guidance to be extremely valuable and his feedback and editorial comments quite generous and helpful for the writing this thesis.

I am also thankful to our **Head of the Department, Dr. A. K. Chatterjee** as well as **PG Coordinator, Dr. Alpana Agrawal, Associate Professor,** and the entire faculty & staff of Electronics and Communication Engineering Department along with my friends who devoted their valuable time and helped me in all possible ways towards successful completion of this work. I thank all those who have contributed directly or indirectly to this work

Lastly, 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 their determination and sacrifice

Place: T.U. Patiala, India (Tanveet Kaur)

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, digital FIR filter has been designed by using Kaiser windows. This technique is simple conceptually and computationally. The design of the filter is formulated as a problem of optimizing I_o(x), which is the modified zeroth-order Bessel function of the first kind. Besides this the adjustable parameter α, has been selected so as to optimise the mainlobe width. Design examples have also shown that the design complexity of the Kaiser window approach is much less than that in nonlinear optimizations. On the other hand, because in Kaiser designs the stopband attenuation is determined by the window, direct control over the stopband attenuation can be achieved. The FIR filter is implemented in Spartan-III-xc3s200-4tq144 FPGA and simulated with the help of Xilinx ISE (Integrated Software Environment). Software WEBPACK project navigator 10.1 was used for synthesizing and simulation the code. For an N order filter the number of shift register and adders required is N and the number of multipliers required is M+1. These filters can work in real time.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ASIC	Application Specific Integrated Circuit
BPF	Band Pass Filter
BRF	Band Reject Filter
CLB	Configurable Logic Block
CPLD	Complex Programmable Logic Device
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
FA	Full Adder
FF	Flip Flop
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
HDL	Hardware Description Language
HPF	High Pass Filter
LAB	Logic Array Block
LPF	Low Pass Filter
LUT	Look-up Table
MAC	Multiplier Accumulate
MMI	Monolithic Memories Incorporated
PAL	Programmable Array Logic
PLA	Programmable Logic Array
PLD	Programmable Logic Devices
PROM	Programmable Read-Only Memory
RAM	Random Access Memory
RTL	Register Transfer Level
SoC	System on Chip
VHDL	Very High Speed Integrated Circuits (VHSIC) Hardware Description Language

CHAPTER 1 INTRODUCTION

1.1 GENERAL

A **filter** is a device or process that removes from a signal some unwanted component or feature. Filtering is a class of signal processing, the defining feature of filters being the complete or partial suppression of some aspect of the signal. Most often, this means removing some frequencies and not others in order to suppress interfering signals and reduce background noise. The word "filtering" refers to an attempt to extract the important part of some data while eliminating random contributions called "noise" or other unwanted features which obscure the ones that matter.

There are many different bases of classifying filters and these overlap in many different ways; there is no simple hierarchical classification. Filters may be:

- Analog or digital
- Discrete-time (sampled) or continuous-time
- Linear or non-linear
- Time-invariant or time-variant, also known as shift invariance. If the filter operates in a spatial domain then the characterization is space invariance.
- Passive or active type of continuous-time filter
- Infinite impulse response (IIR) or finite impulse response (FIR) type of discrete-time or digital filter.

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. 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.

Fig. 1.1. Block Diagram of a Basic Filter

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.

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.

Advantages of Digital filters

- are easily designed, tested and implemented on computer or workstation,
- are extremely stable with respect both to time and temperature,
- can handle low frequency signals accurately
- are programmable
- are very much versatile.

The difference between digital filters and analog filters are enumerated in Table 1.1.

Table 1.1. Difference between Digital Filters and Analog Filters

DIGITAL FILTERS	ANALOG FILTERS
High accuracy	Less accuracy component tolerances
Linear Phase(FIR Filters)	Non linear Phase
No drift due to component variations.	Drift due to component variation
Flexible, Adaptive filtering possible.	Adaptive filters difficult.
Easy to simulate and design.	Difficult to simulate and design.
Computation must be computed 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.

1.2 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 electronic circuit operating on continuous-time analog-signals. An analog signal may be processed by a 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.

A digital filter 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 microprocessor implement the digital filter by performing the necessary mathematical operations on the numbers received from the ADC. In some high performance applications, an FPGA or ASIC is used instead of a general purpose microprocessor,

or a specialized DSP with specific paralleled architecture for expediting operations such as filtering.

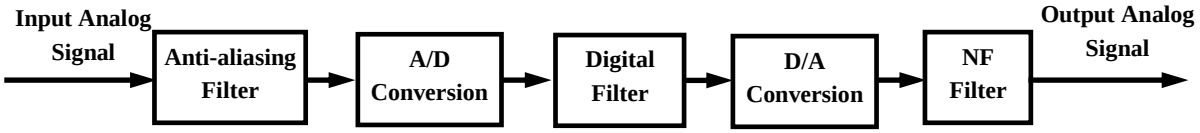


Fig. 1.2. Block Diagram of Digital Filtering Process

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 passband is a frequency at which the transition of the passband to the transition region occurs. The cut-off frequency of the stopband is a frequency at which the transition of the transition region to the stopband occurs.

1.3 TYPES OF DIGITAL FILTERS

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.

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

- linear phase characteristic
- high filter order (more complex circuits)
- stability

The basic characteristics of Infinite Impulse Response (IIR) are:

- non-linear phase characteristic;
- low filter order (less complex circuits); and
- resulting digital filter has the potential to become unstable.

The general form of the digital filter difference equation is

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

where, $y(n)$ is the current filter output, the $y(n-i)$'s are previous filter outputs, the $x(n-i)$'s are current corresponding to the zeros of the filter, or previous filter inputs, the a_i are the filter's feed forward coefficients corresponding to the zeroes 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, 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. 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 $-6.02\,B + 6.02 \log 2N$ 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. 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.

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

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

where a_i represents constants.

Now assuming causality for the filter we have

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

In addition, assuming $a = 0$ for $i > N$ the response can be written as N^{th}-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

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

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 . $i\,a.\,s$ are nothing but the samples of the impulse response. That means these constants are the filter coefficients.

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

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.

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.

1.4 WHY FIR FILTERS?

The main advantages of the FIR filter designs over their IIR equivalents are the following:

1. FIR filters with exactly linear phase can easily be designed.

2. There exist computationally efficient realizations for implementing FIR filters. These include both nonrecursive and recursive realizations.
3. FIR filters realized nonrecursively are inherently stable and free of limit cycle oscillations when implemented on a finite-word length digital system.
4. Excellent design methods are available for various kinds of FIR filters with arbitrary specifications.

The output noise due to multiplication round-off errors in an FIR filter is usually very low and the sensitivity to variations in the filter coefficients is also low.

1.5 DIGITAL FIR FILTERS CHARACTERIZATION

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.

1.6 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.

1.7 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

1.8 FIELD PROGRAMMABLE GATE ARRAY (FPGA)

A field programmable gate array is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex logic functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories.

FPGA arrived in 1984 as an alternative to programmable logic devices (PLDs) and ASICs. FPGA offers the significant benefits of being readily programmable. FPGA can be programmed again and again. FPGA consists of an array of logic blocks that are configured using software. Programmable input/output blocks surround these logic blocks. Both are connected by programmable interconnects. A hierarchy of programmable interconnects allows the logic blocks of FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer to implement any logical function- hence field programmable.

Today, however, FPGA offers millions of gates of logic capacity, operate at 300 MHz, can cost less than $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 resource as such as high-speed serial input/output, arithmetic modules, embedded processors, and large amount of memory.

The development of these designs is made on regular FPGAs and then migrated into a fixed version that more resembles an ASIC. Complex programmable logic devices, or CPLDs, are other alternative. FPGAs are generally slower than their application specific integrated circuit (ASIC) counterparts, as they can't handle as complex a design, and draw more power. However, they have several advantages such as shorter time to market, ability to re-program in field to fix bugs, and lower non recurring engineering costs.

1.9 FEASIBILITY STUDY

FIR filter designing using VHDL is very cheap operation to be performed. The FPGA kit Xilinx3E was provided by the college and the code has been downloaded in the kit. It means FIR filter implementing has been done.

Codes are quite simple as direct from fixed point FIR filter has been realized.

1.9.1 DSP VERSUS FPGA

For a Digital Signal Processor (DSP) to process one frame of data it will have to fetch the data to be processed, perform the required mathematical operation, and then store the result back to memory. For an average sized image it is likely that the whole frame may be too large to be stored in on-chip memory and will therefore need to be fetched from and stored back to external memory. This process will add to the total number of cycles involved in processing each pixel of the image.

In addition to this memory overhead several cycles may be required by the CPU just to perform the mathematical operations that have been specified. Add to this the possibility of processor branches to handle interrupts and servicing of other high priority threads and the overall data rate drops significantly. The end result is that many image-processing functions tend to require several processor clock cycles per pixel to complete.

As the majority of image processing can be broken down into highly repetitive tasks FPGAs present a very interesting alternative to the DSP. What is important when using an FPGA is that, the data rate through the FPGA is better than through a DSP. This can easily be achieved as the individual logic cells of FPGAs map well to the individual mathematical steps involved in image processing. In addition, there are also dedicated resources for multiplication and memory storage that can be used to further improve performance. Particularly, the more complex functions such as

convolution can be mapped very successfully to FPGAs. When convolving an image, a window of pixels are treated with a mask where individual locations in the window are 'weighted' according to a set of previously defined co-efficient. For each position of the window all pixels are multiplied against their respective co-efficient. The final result is then scaled to produce a single output pixel for the centre location of the window.

If a conventional DSP were to be used the overall performance would be limited by the number of multiplications that could be done in parallel. In practice a DSP will require several clock cycles to perform all of the necessary multiplications and additions to calculate a single pixel. The FPGA on other hand can implement as many multipliers as are necessary in order to calculate one pixel at the full input data rate. By mapping convolution to FPGAs that already provide dedicated multipliers among their sea-of-gates, it becomes very easy to build a processing pipeline that can convolve at very high data rates.

1.9.2 A ROLE FOR THE DSP

Although a large proportion of image processing algorithms are simply highly repetitive processes there is still a role for the DSP. In a system that can benefit from the performance advantages of the FPGA there is often a point in the data flow where a decision has to be made, and this decision will often take the form of if, then, else logic rather than a pixel-by-pixel iteration.

For control loops and complex branches in operation, the DSP can still prove to be a highly effective tool. Implementing equivalent logic in an FPGA can quickly eat-up the available gates and reduce the overall data rate, so a compromise may be necessary. The simple solution to this situation is to use both types of resources in a single system. The high data rate FPGA used as the data-reducing engine feeding results downstream to a DSP that can effectively reach to accept or reject, pass fail decision for the overall system.

1.10 OBJECTIVE OF THESIS

The thesis embodies 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, highpass, bandpass and band reject filters on FPGA using the specified window technique

CHAPTER 2 LITERATURE REVIEW

In this chapter the work done by many earlier researchers has been presented and discussed related the thesis topic.

L. Jieshanproc and H. Shizhen (2009) analyzed and discussed 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 was a method that is based on FPGA, draws the coefficient by Matlab and adopts the pipeline to implete the FIR digital filter. The article 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 **[8]**.

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 the 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.

P. Longa and A. Miri (2006) presented a highly area-efficient multiplier-less FIR filter. Distributed Arithmetic (DA) has been used to implement a bit-serial scheme of a general asymmetric version of an FIR filter, taking optimal advantage of the 4-input LUT-based structure of FPGAs. Furthermore, introduced a modification in the accumulator stage to achieve further savings, resulting in reduced area requirements in comparison to previous LUTless DA architectures **[12]**.

A modified version of the accumulator, which is intended for low-precision applications, has also been introduced. Besides this they showed that LUT-based scheme is still more efficient than the proposed LUT-less architecture for all the implemented FIR filter orders. In this sense, it has been determined that the argument to save resources in a LUT-less scheme can be applied to the LUT-

based structure too if one follows the same criteria of reducing the word-length in the LUT unit depending on the used filter coefficients.

C. Li introduced the definition and basic principles of FIR digital filters, and the design methods based on MATLAB. After the description of the process of design and simulation of a FIR band-pass filter by means of window function method, the results prove that various performance of the designed FIR filter reach the appointed requirement, the designed method is simple, the design process is handy and efficient**.** The FIR filter, thus saved a lot of programming time, improving the efficiency of programming, and parameter changes were also very convenient **[6]**.

G. Gaizhi, Z. Pengju, Y. Zongzuo and W. Hailong (2010) discussed the general principles of FIR digital wave filter, describing a design method for FIR digital electric wave filter based on DSP processor with TMS320VC54x fixed point series. The coefficient of wave filters obtained using the MATLAB window function calculator and verified with the DSP measuring system. The digital electric wave filter's all functionalities met design expectations **[5]**.

For Digital wave filter plays a major part in digital signals processing with broad application. This type of filter was easy to transform with flexibility and applicability. The sample low-pass filter designed in this study obtained filter coefficients on the MATLAB platform. This method can be used to design band-pass, band-resistance and high-pass filters.

W. Yunlong, W. Shihu and J. Rendong (2011) an extreme simple method for digital FIR filter design . In comparison with existed methods the 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 stopband attenuation is in the range of 39.5db to 48.5db and corresponding maximum passband ripple is from 0.35db to 0.18db **[1]**.

Using the method for the design of FIR digital filters based on sinc sum function they obtained a series of filter formulas. These formulas consisted of two sub-filters, which are the sum of two weighted impulse response of an ideal lowpass filter with the shift of cut-off frequency. The amplitude frequency response of each sub-filter can be expressed with the sum of two sinc sum functions. Compared with existed filter formulas these formulas are the simplest except the one which is truncated directly from the impulse response of an ideal lowpass filter. The cause of improving filter performances is that the kinds of local extrema of frequency responses of the two sub-filters are always different except in transition band and therefore deviations of the two sub-filters can partly or mostly counteract in frequency domain both in passband and in stopband. Thus the new filter formulas can be used in the design of narrow passband filter, notch filter and filters that permit larger passband ripples.

V. Soni, P. Shukla and M. Kumar (2011) 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 nonrecursive Finite Impulse Response (FIR) filter by using Exponential window was proposed. The far-end stopband attenuation is most significant parameter when the signal to be filtered has great concentration of spectral energy **[2]**.

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 stopband 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 stopband attenuation (amplitude of last ripple in stopband). Digital FIR filter designed by Kaiser window has a better far-end stopband attenuation than filter designed by the other previously well known adjustable windows such as Dolph-Chebyshev and Saramaki, which are special cases of Ultra spherical windows, but obtaining a digital filter which performs higher far-end stopband attenuation than Kaiser window will be useful. The design of nonrecursive digital FIR filter has been proposed by using Exponential window. It provides better far-end stopband 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. **[6]**

Z. Yu, M.L. Yu, K. Azade and A. N. Willson (2010) tried to exploit the sign-extension property of a 2's complement number and proposed a reduced representation of 2's complement numbers to avoid sign-extension. Instead of having the high switching activity at the MSB side of the datapath, the proposed number representation avoids switching of the MSBs altogether, and therefore reduces the power dissipation in digital arithmetic circuits. While arithmetic circuits using 2's complement representation are easy to implement, it is well-known that the sign-extension bits of a 2's complement number cause high switching activity in digital arithmetic circuits. Such switching is undesirable in low power applications. With the proposed technique, the maximum magnitude of a 2's complement number is detected and a reduced representation is dynamically generated to represent the signal. There is a constant error introduced by the reduced representation and such error is compensated accordingly. The proposed signal representation is particularly useful in digital filters where the coefficients are slowly varying and have small magnitudes thereby showing a **38%** power saving using the proposed technique **[4]**.

P. P. Vaidyanathan 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 the 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 show that with the use of Hybrid operators in FIR architectures power savings of up to 25% are possible, together with a 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 **[21]**.

W. S. Lu, A. Antoniou and S. Saab (1998) 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 an cascade satisfy again as far as possible the overall required specifications. This process was repeated until a multisection 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 **[18]**.

R. A. Kennedy Abhayapala introduced a novel sinhc kernel which generates a Kaiser window based TFD. Time-frequency distributions (TFDs) have been used as a tool to locate the time dependency of the spectrum in analysing nonstationary signals. The Kaiser window has been used extensively in designing finite impulse response digital filters, and in other digital signal processing applications. The Kaiser window is an asymptotic approximation to the prolate spheroidal wave functions, which have been shown to provide concentration of energy simultaneously in time and frequency and thus showed that Kaiser distribution is a member of the Cohen class, and the corresponding kernel satisfies all the desirable kernel properties. Besides this they showed that the shape of the Kaiser window can be used to control the trade-off between auto-term and cross-term energy **[7]**.

A. Kuncheva and G. Yanchev (2008) Discussed of efficiency implementation methodologies used in DSP application targeted modern FPGA. Digital filters are very important part of DSP. And there is a present need for digital filters that can efficiently implement in 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 **[10]**.

The simple, traditional adder-tree approach limited the performance and extensibility of a given filter implementation. By using adder-chain-style implementations, these limitations were lifted and the huge benefits Virtex-4 FPGAs offer are possible. Using tool such as Matlab they found the filer 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 manipulations that are specific for this design.

N. H. Phuong 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 tradeoff the transition and ripple **[17]**.

Y. P. Lin and P. P. Vaidyanathan (1998) proposed limiting the search of the prototype filters to the class of filters obtained using Kaiser windows. The design process was reduced to the optimization of a single parameter i.e. the cutoff frequency ω_c in Kaiser window designs. They further reduce the design complexity by introducing a new objective function ϕ_{new}. It have been shown that the design complexity of the Kaiser window approach is much less than that in nonlinear optimizations. On the other hand, because in Kaiser designs the stopband attenuation is determined by the window, direct control can be given over the stopband attenuation **[11]**.

Lee H. and Sobelman G. (2003) discussed efficiency implementation methodologies used in DSP application targeted modern FPGA. The modern programmable technology provides possibility to implement digital system with the use of dedicated embedded DSP blocks. The simple, traditional adder-tree approach limited the performance and extensibility of a given filter implementation. By using adder-chain style implementations, these limitations were lifted and thus found out the filter order **[14].**

CHAPTER 3

FIR FILTER DESIGN

3.1 INTRODUCTION

FIR filters are digital filters with finite impulse response. They are also known as **non-recursive digital filters** as they do not have the feedback (a recursive part of a filter), even though recursive algorithms can be used for FIR filter realization.

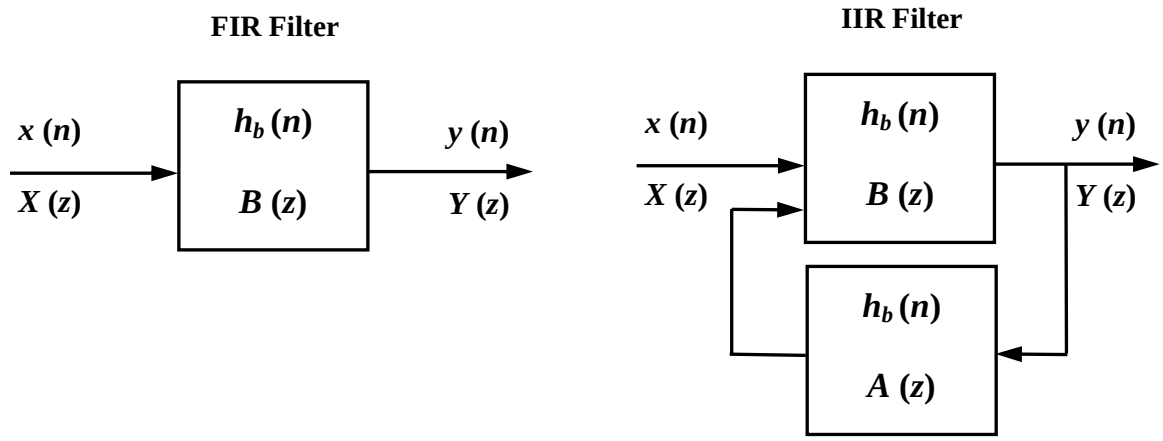


Fig 3.1. Block Diagrams of FIR and IIR Filters

FIR filters can be designed using different methods, but most of them are based on ideal filter approximation. The objective is not to achieve ideal characteristics, as it is impossible anyway, but to achieve sufficiently good characteristics of a filter. The transfer function of FIR filter approaches the ideal as the filter order increases, thus increasing the complexity and amount of time needed for processing input samples of a signal being filtered.

3.2 CONCEPT OF FIR FILTER DESIGN

The fundamental concept of FIR filter design is that the filter frequency response is determined by the impulse response & quantized impulse response & the filter coefficients are identical. The input to FIR filter is an impulse, and as the impulse propagates through the delay elements, the

filter output is identical to the filter coefficients. The FIR filter design process therefore consists of determining the impulse response & then quantizing the impulse response to generate the filter coefficients.

3.3 CHARACTERISTICS OF FIR FILTERS

- Impulse response has a finite duration (N-cycles).
- Linear phase, constant-Group delay (N must be odd).
- No analog equivalent, unconditionally stable and can be adaptive.
- Computational Advantages when decimating output.
    - Easy to understand and Design
    - Windowed-Sinc Method.
    - Fourier-Series Expansion with Windowing.
    - Frequency-sampling using inverse FFT-arbitrary Frequency response.

A rather complete procedure for the design of FIR filters are as follows:

- Specifications of the filter.
- Choosing an appropriate linear phase filter type.
- Choosing the method of design such as window, optimal, frequency sampling…
- Calculating the filter coefficients (impulse response).
- Finding suitable structure.
- Analysis of the finite wordlength effects.
- Implementation of the filter in hardware/software.

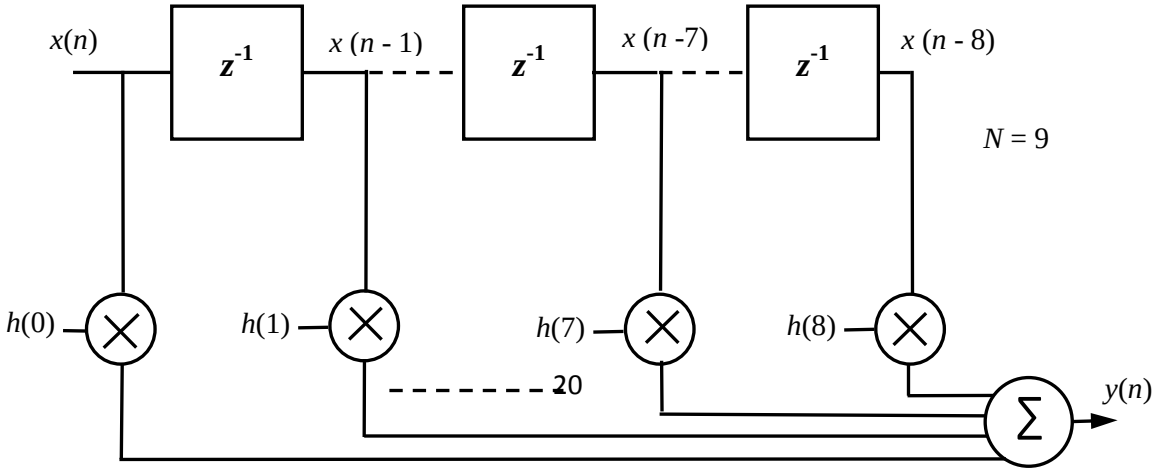


Fig. 3.2. FIR Filter Impulse Response determines the Filter Coefficients

The design of a digital filter involves following five steps:

1. 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.

2. 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.

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

4. Analysis of finite word length effects: The effect of quantizing the filter coefficients and input data as well as the effect of carrying out the filtering operation using fixed word length on the filter performance is analyzed here.

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

These five inter related steps are summarized by flow chart:

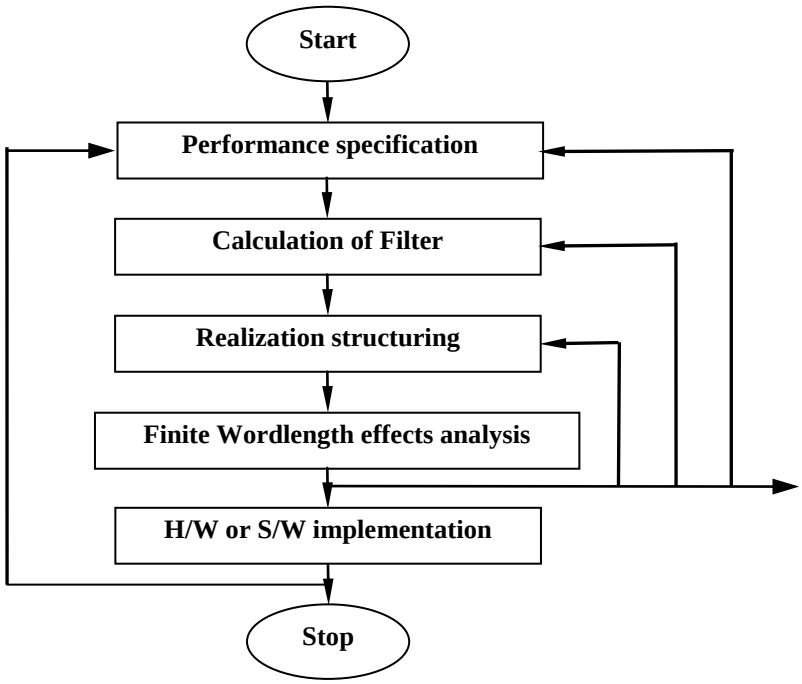


Fig. 3.3. Summary of Design Stage for Digital Filter

3.4 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.

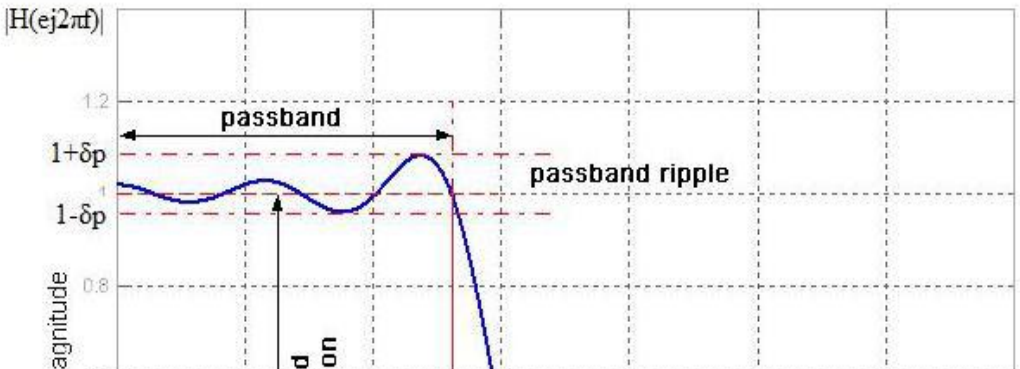


Fig. 3.4. Magnitude Frequency Response Specifications for a Lowpass Filter.

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 how 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 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 δ_s and δ_p is the transition width of the filter. Another important parameter is the filter length, N, which defines the number of filter.

3.5 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.

3.5.1 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:

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

Now start with the ideal lowpass response shown in figure, where ω_c is the cutoff frequency and the frequency scale is normalized: $T = 1$. By letting the response go from $-\omega_c$ to ω_c simplify the integration operation. Thus the impulse response is given by:

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

$$= (2 f_1 c)/(n\omega_1 c)\,[(\sin c)]\,[(n\omega_1 c)], \quad n \neq 0,\ -\infty \le n \le \infty$$

$$= 2f, \ n = 0 \tag{3.2}$$

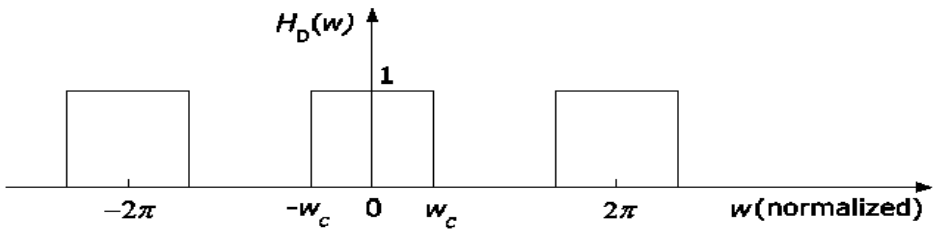


Fig. 3.5. Ideal Frequency Response of a Lowpass Filter

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 within the window are retained and all that are outside the window are discarded.

3.5.2 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.

Table 3.1. Summary of Ideal Impulse Responses

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

3.5.2.1 Nonrecursive frequency sampling

To obtain the FIR coefficients of the filter whose frequency response is depicted in Fig. 2.3. By taking N samples of the frequency response at intervals of kF_S / n, $K = 0, 1, ..., n-1$.

The filter coefficients can be obtained as inverse DFT of frequency samples.

$$h(n) = \frac{1}{N} \sum_{k=0}^{\frac{n}{2}-1} H(k)\, e^{j\left(\frac{2\pi}{N}\right)nk} \tag{3.3}$$

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:

$$f(fn) = \frac{1}{N}\left[\sum_{k=1}^{\frac{N}{2}-1} 2\,|H(k)|\cos 2\pi k\,(n-\alpha)/N\ H(0)\ \right] \tag{3.4}$$

where $\alpha = (N-1)/2$, and $H(k)$ are the samples of the frequency response of the filter taken at intervals of kF_s/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

$$f\ k = \left(k + \frac{1}{2}\right)\frac{fs}{N},\ k = 0, 1, 2, ... N-1 \tag{3.5}$$

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+20 M) 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.

3.5.2.2 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:

$$H(z) = \frac{1-z^{-n}}{N}\sum_{nk=0}^{n-1}\frac{H(k)}{1-z^{-1}\,e^{\frac{j2\pi}{N}}} = H_1(z)\,H_2(z) \tag{3.6}$$

Thus in recursive form, $H(z)$ can be viewed as a cascade of two filters: a comb filter, $H(z)$, which has N zeros uniformly distributed around the unit circle, and a sum of N single all-pole

filters, $2H(z)$. The zero of comb filter and the poles of the single pole filters are coincide on the unit circle at points $zk =$. 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 $2H(z)$ 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

$$H(z) = \frac{1-r^{N_z-N}}{N} \sum_{k=0}^{N-1} \frac{H(k)}{1-re^{j2\pi k/N}Z-1} \tag{3.7}$$

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. So above equation can expressed as

$$H(z) = \frac{1-r^{N_Z-N}}{N} \left[\frac{|H(k)| \left\{ 2\cos\left(\frac{2\pi k\alpha}{N}\right) - 2\cos[2\pi k(1+\alpha)] \right\}}{1-2r\cos\left(\frac{2\pi k}{N}\right)Z^{-1} + r^2z^2} \right] + \frac{H(0)}{1-z^{-1}} \tag{3.8}$$

where $\alpha = (N-1)/2$. For N odd $M = (N-1)/2$ and for N even $M = \frac{N}{2} - 1$.

3.5.3 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 $1-\delta_p$ and $1+\delta_p$. 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:

$$E(u) = W(u)[H\,D(u).H(u)]$$

where $H\ D(u)$ is the ideal response and $W(u)$ 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, such that the value of the weighted error, $|E(u)|$, is minimized in the passband and stopband. Mathematically, this may be expressed as: min[max$|E(u)|$], over the passbands and stopbands. It has been established that when max$|E(u)|$ 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).

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:

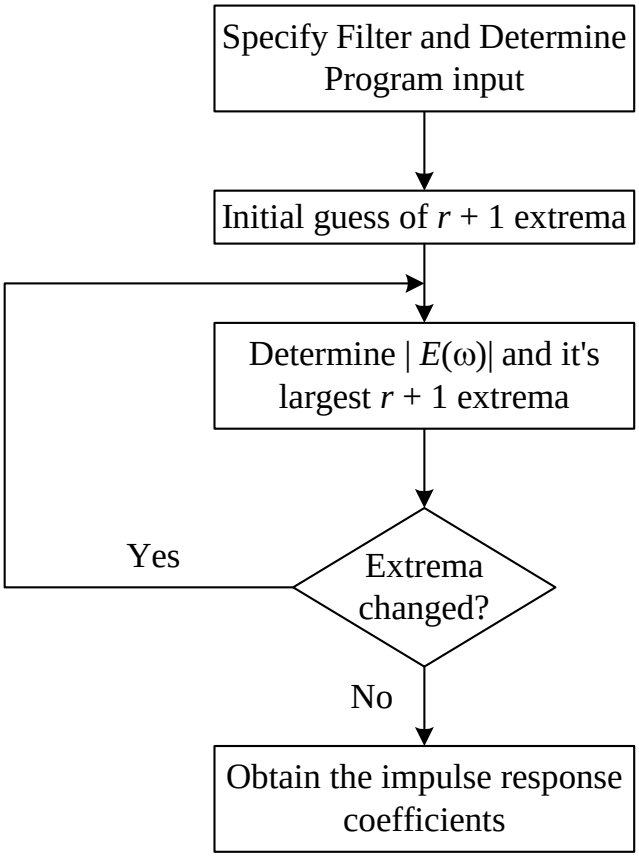


Fig 3.6 Simplified Flowchart of the Optimal Method

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.

3.5.4 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.

3.6 FIXED WINDOWS

The impulse response of ideal filters is infinite duration (IIR). We cannot evaluate the corresponding frequency response and, especially, implement the filter by hardware and/or software. Thus we must truncate the impulse response at both ends with respect to the central. Even we truncate the impulse response when it is small enough, but such a sudden cutoff will cause some undesired effects. The window method will reduce them. In the time domain, windowing means that we multiply the desired (usually ideal) infinite duration impulse response $h_d(n)$ by a finite duration window (or window function) $\omega(n)$ to get a soft truncation.

The resulted impulse response $h(n)$ of the designed filter is the product

$$h(n) - h_d(n)\, w(n); \qquad 0 \le n \le M$$

Here we assume that all desired impulse responses and windows are causal, from $n = 0$ to $n = M$, i.e. window length is $M + 1$ samples (time indices). Windowing of a simple waveform, like cosωt causes its Fourier transform to develop non-zero values (commonly called spectral leakage) at frequencies other than ω. The leakage tends to be worst (highest) near ω and least at frequencies farthest from ω.

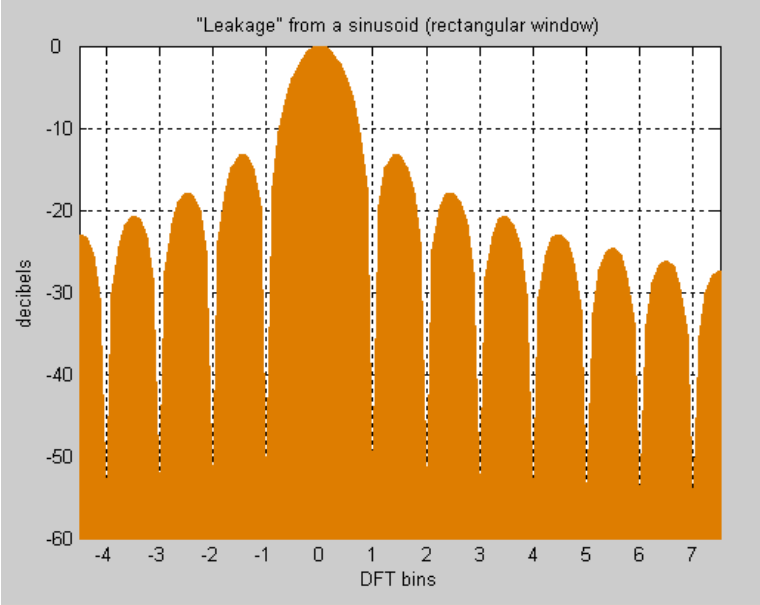


Fig. 3.7 View of Spectral Leakage

If the signal under analysis is composed of two sinusoids of different frequencies, leakage can interfere with the ability to distinguish them spectrally. If their frequencies are dissimilar and one component is weaker, then leakage from the larger component can obscure the weaker's presence. But if the frequencies are similar, leakage can render them unresolvable even when the sinusoids are of equal strength.

The rectangular window has excellent resolution characteristics for signals of comparable strength, but it is a poor choice for signals of disparate amplitudes. This characteristic is sometimes described as low-dynamic-range. At the other extreme of dynamic range are the windows with the poorest resolution. These high-dynamic-range low-resolution windows are also poorest in terms of **sensitivity**; this is, if the input waveform contains random noise close to the signal frequency, the response to noise, compared to the sinusoid, will be higher than with a higher-resolution

window. In other words, the ability to find weak sinusoids amidst the noise is diminished by a high-dynamic-range window. High-dynamic-range windows are probably most often justified in wideband applications, where the spectrum being analyzed is expected to contain many different signals of various strengths.

In between the extremes are moderate windows, such as Hamming and Hann. They are commonly used in narrowband applications, such as the spectrum of a telephone channel. In summary, spectral analysis involves a tradeoff between resolving comparable strength signals with similar frequencies and resolving disparate strength signals with dissimilar frequencies. That tradeoff occurs when the window function is chosen.

3.7 WINDOW EXAMPLES

Terminology:

- M represents the width, in samples, of a discrete-time window function. Typically it is an integer power-of-2, such as $2^{10} = 1024$.
- n is an integer, with values $0 \le n \le M-1$. So these are the time-shifted forms of the windows**:**

  $$w(n) = w_0\left(n - \frac{N-1}{2}\right), \text{ where } w_0(n) \text{ is maximum at } n = 0.$$

  Some of these forms have an overall width of $N-1$, which makes them zero-valued at $n = 0$ and $n\ N-1$. That sacrifices two data samples for no apparent gain, if the DFT size is N. When that happens, an alternative approach is to replace $N-1$ with N in the formula.
- Each figure label includes the corresponding noise equivalent bandwidth metric (B), in units of DFT bins. As a guideline, windows are divided into two groups on the basis of B. One group comprises $1 \le B \le 1.8$ and the other group comprises $B \ge 1.98$ The Gauss and Kaiser windows are families that span both groups.

3.8 COMMON WINDOWS

Rectangular: $$w[n] = \begin{cases} 1, & 0 \le n \le M \\ 0, & \text{else} \end{cases}$$

Barlett (triangular): $$w[n] = \begin{cases} 2n/M, & 0 \le n \le M/2 \\ 2-2n/M, & M/2 \prec n \le M \\ 0, & \text{else} \end{cases}$$

Hanning: $$w[n] = \begin{cases} 0.5-0.5\cos(2\pi n/M), & 0 \le n \le M \\ 0, & \text{else} \end{cases}$$

Hamming: $$w[n] = \begin{cases} 0.54-0.46\cos(2\pi n/M), & 0 \le n \le M \\ 0, & \text{else} \end{cases}$$

Blackman: $$w[n] = \begin{cases} 0.42-0.5\cos(2\pi n/M)+0.08\cos(4\pi n/M), & 0 \le n \le M \\ 0, & \text{else} \end{cases}$$

Note the above windows are all linear phase, i.e., shifted version of a symmetric window. There exist many of these window functions (lots of research papers on this stuff in the 1960's and 1970's).

3.9 KAISER WINDOWS

A simple approximation of the window using Bessel functions, as discovered by Jim Kaiser can be given by the formula:

$$w(n) = \frac{I_0\left(\pi\alpha\sqrt{1-\left(\frac{2n}{N-1}-1\right)^2}\right)}{I_0(\pi\alpha)}$$

where I_0 is the zero-th order modified Bessel function of the first kind, and usually α = 3.

Note that:

$$w_0(n) = \frac{I_0\left(\pi\alpha\sqrt{1-\left(\frac{2n}{N-1}\right)^2}\right)}{I_0(\pi\alpha)}$$

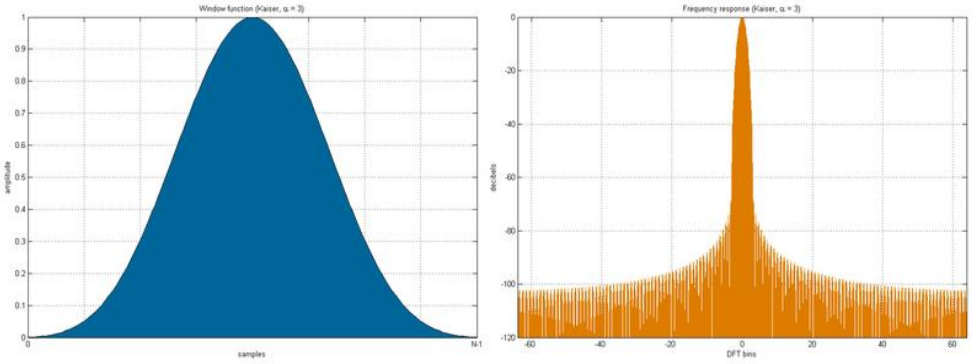


Fig. 3.8. Kaiser Window, α = 2; *B* =1.5

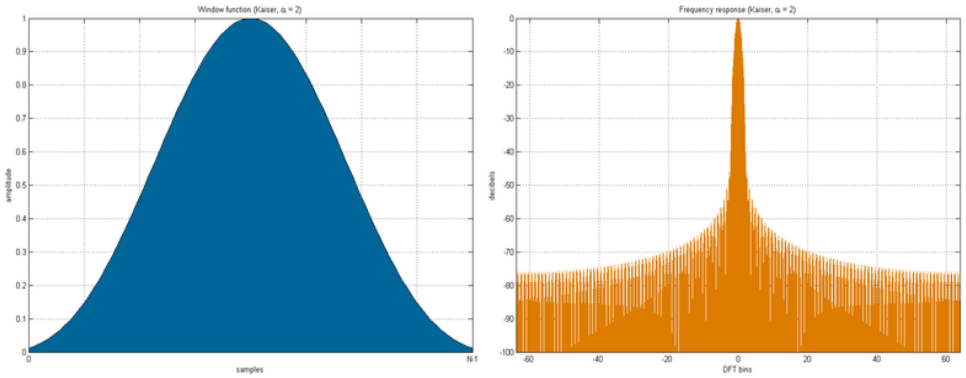


Fig. 3.9. Kaiser Window, α = 3; *B* = 1.8

The Kaiser window is a one-parameter family of window functions used for digital signal processing, and is defined by the formula:

$$w_n = \begin{cases} w(n) = \dfrac{I_0\left(\pi\alpha\sqrt{1-\left(\dfrac{2n}{M}-1\right)^2}\right)}{I_0(\pi\alpha)}, & 0 \le n \le M \\ 0 & \text{otherwise} \end{cases}$$

- I_0 is the zeroth order Modified Bessel Function of the first kind.
- α is an arbitrary real number that determines the shape of the window. In the frequency domain, it determines the trade-off between main-lobe width and side lobe level, which is a central decision in window design.
- M is an integer, and the length of the sequence is $N = M + 1$.
- When N is an odd number, the peak value of the window is $w_{M/2} = 1$. And when N is even, the peak values are $w_{\frac{N}{2}-1} = w_{\frac{N}{2}} < 1$.

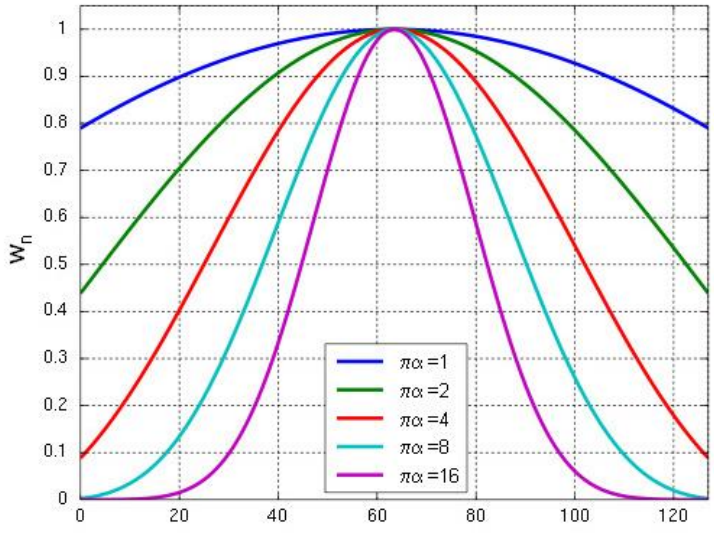


Fig. 3.10. Kaiser Window Function for M = 128 and $\pi\alpha$ = 1, 2, 4, 8, 16

3.10 FREQUENCY RESPONSE

Underlying the discrete sequence is this continuous-time function and its Fourier transform**:**

$$\underbrace{\frac{I_0\left(\pi\alpha\sqrt{1-\left(\frac{2t}{M}\right)^2}\right)}{I_0(\pi\alpha)}}_{w(t)} \overset{F}{\Leftrightarrow} \underbrace{\frac{M\,.\sinh\left(\pi\sqrt{\alpha^2-(M\,.\,f)^2}\right)}{I_0(\pi\alpha)\,.\,\pi\sqrt{\alpha^2-(M.f)^2}}}_{w(f)}$$

The maximum value of $w(t)$ is $w(0) = 1$. The w_n sequence defined above are the samples of**:**

$$w\left(t-\frac{M}{2}\right)\,.\,\text{rect}\left(\frac{t-M/2}{M+1}\right)$$

for all integer values of t, and where rect() is the rectangle function.

The larger the value of $|\alpha|$, the narrower the window becomes; α = 0 corresponds to a rectangular window. Conversely, for larger $|\alpha|$ the main lobe of $W(f)$ increases in width, while the side lobes decrease in amplitude. Thus, this parameter controls the tradeoff between main-lobe width and side-lobe area, as is illustrated in the plot of the frequency spectra below. For large α, the shape of the Kaiser window (in both time and frequency domain) tends to a Gaussian curve. The Kaiser window is nearly optimal in the sense of its peak's concentration around ω = 0.

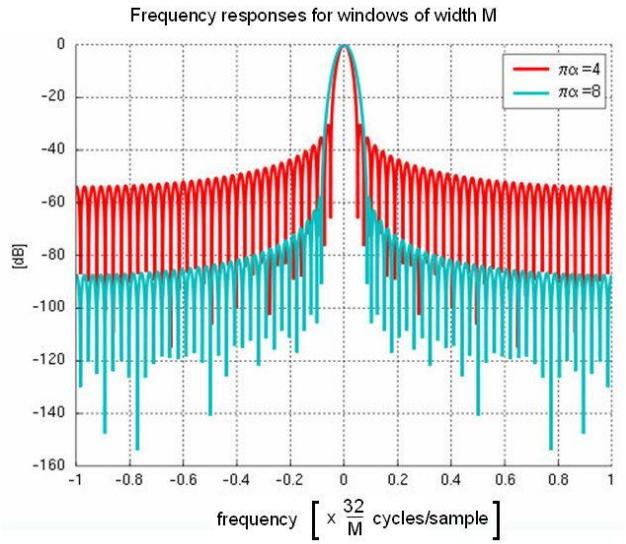


Fig. 3.10. Frequency Spectra of Kaiser Windows for $\pi\alpha = 4$ and 8.

3.11 WINDOW METHOD OF FIR FILTER DESIGN

The basic idea behind the Window method of filter design is that the ideal frequency response of the desired filter is equal to 1 for all the pass band frequencies, and equal to 0 for all the stop band frequencies, then the filter impulse response is obtained by taking the Discrete Fourier Transform (DFT) of the ideal frequency response. Unfortunately, the filter response would be infinitely long since it has to reproduce the infinitely steep discontinuities in the ideal frequency response at the band edges. To create a Finite Impulse Response (FIR) filter, the time domain filter coefficients must be restricted in number by multiplying by a window function of a finite width. The simplest window function is the rectangular window which corresponds to truncating the sequence after a certain number of terms.

Rectangular windowing the time domain will result in a frequency spectrum with the width of the pass band close to the desired value but with side lobes appearing at the band edges. To suppress the side lobes and make the filter frequency response approximate more closely to the ideal, the width of the window must be increased and the window function tapered down to zero at the ends. This will increase the width of the transition region between the pass and stop bands, but will lower the side lobe levels outside the pass band.

3.12 KAISER-BESSEL FILTER GENERATOR

The Kaiser-Bessel window provides a simple to calculate set of window coefficients, the parameters of which can be adjusted to produce the desired maximum side lobe level for a near minimal filter length. To demonstrate the power and simplicity of this technique the Kaiser-Bessel filter generator is shown below. Simply set the sample rate and the type of filter desired, low pass, band pass or high pass, then set the frequency values of the edges of the filter and the minimum attenuation that is required in the stop band. Then press the "CALC FILTER" button, the filter coefficients are calculated and plotted along with a graph of the frequency response of the filter.

3.13 KAISER-BESSEL FILTER DESIGN FORMULAE

The methods used in this FIR generator were taken from the paper by J. F. Kaiser, "Nonrecursive digital filter design using I_0-sinh window function". In this paper Kaiser presented empirical formulae for calculating the shape parameter of the Kaiser-Bessel window required to achieve a desired stop band side lode attenuation.

The values of window shape parameter α needed to achieve the attenuation *Att* may be calculated from:

$$\alpha = \begin{cases} 0.1102\ (Att - 8.7) & Att \geq 50 \\ 0.5842\ (Att - 21)^{0.4} + 0.07886(Att - 21) & 21 \leq Att \leq 50 \\ 0 & Att \leq 21 \end{cases}$$

$$M \geq \frac{Att - 8}{14.36\ dF / Fs}$$

where:
$$w|j| = \frac{I_0\left(\alpha\sqrt{1-\left(\frac{j-Np}{Np}\right)^2}\right)}{I_0(\alpha)} \quad for \quad 0 \leq j < M\ ; \quad Np = (M-1)/2$$

CHAPTER 4

SIMULATION AND SYNTHESIS TOOLS

4.1 INTRODUCTION

The following tools required for implementation of FIR filter on FPGA are:

- XILINX ISE web pack 10.1 for design, synthesis and implementation.
- MODSIM SE 6.3f 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.

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.

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.

4.3 SYNTHESIS TOOLS

4.3.1 XILINX ISE 10.1 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.

4.3.2 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.

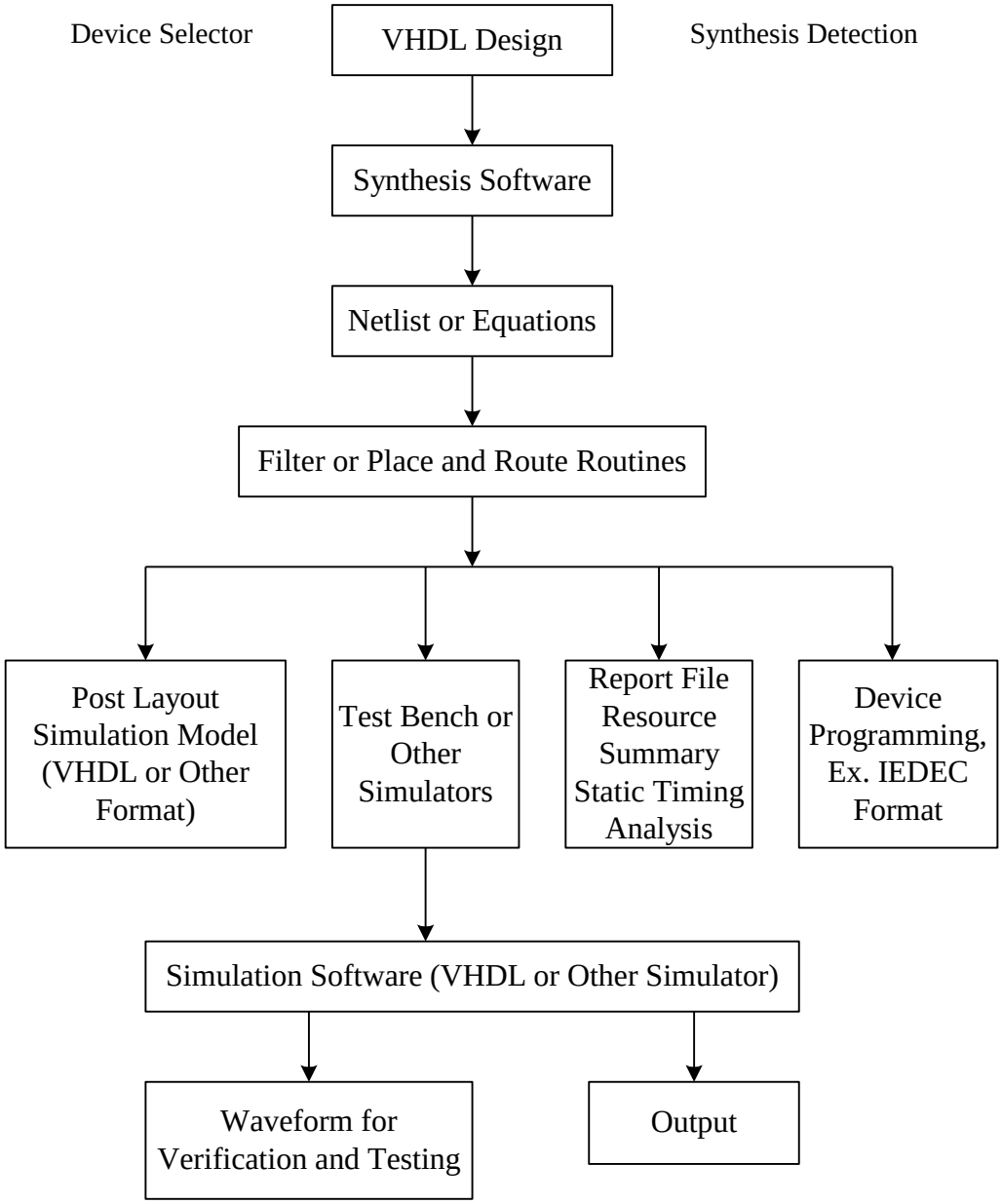


Fig 4.1 Tool Flow Diagram

- **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 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

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.

- 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.

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

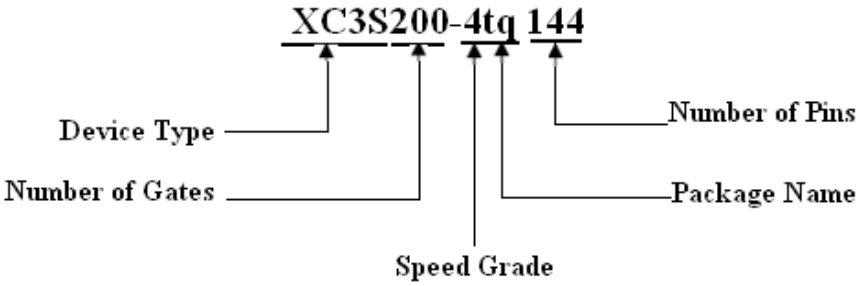
For a programmable device, you simply program the device and immediately have your prototypes. You then have the responsibility to place these prototypes in your system and determine that the entire system actually works correctly. If you have followed the procedure up to this point, chances are very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

4.5 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 XC3S200-4tq144 Spartan-III device.

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



CHAPTER 5

IMPLEMENTATION OF FILTER ON FGPA

5.1 INTRODUCTION

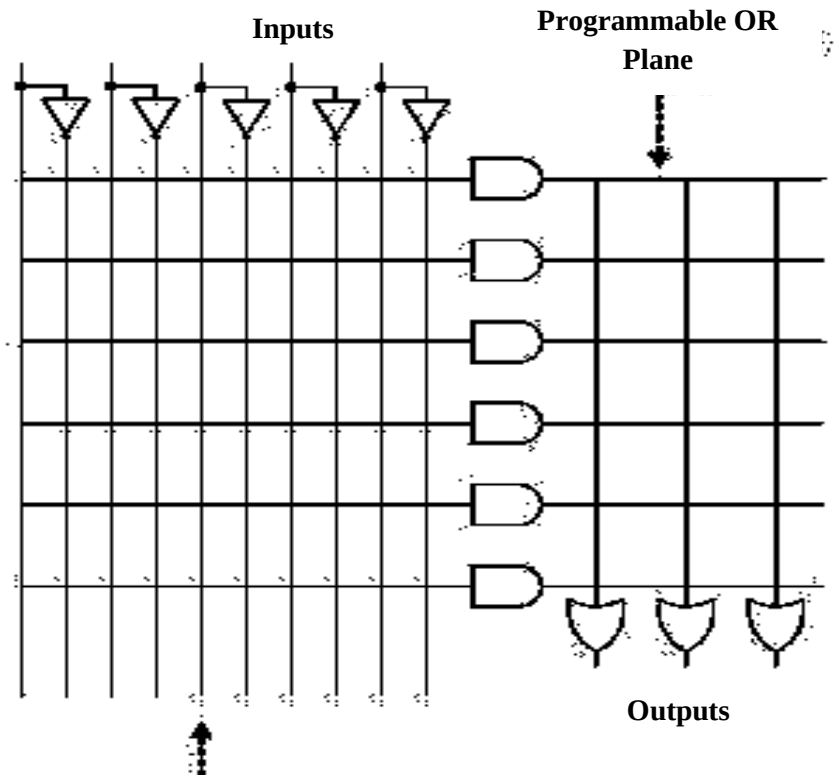
A **field-programmable gate array** (**FPGA**) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). FPGAs can be used to implement any logical function that an ASIC could perform. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

The FPGA is an integrated circuit that contains many (64 to over 10,000) identical logic cells that can be viewed as standard components. Each logic cell can independently take on any one of a limited set of personalities. The individual cells are interconnected by a matrix of wires and programmable switches. A user's design is implemented by specifying the simple logic function for each cell and selectively closing the switches in the interconnect matrix. The array of logic cells and interconnects form a fabric of basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to create the desired circuit.

A typical integrated circuit performs a particular function defined at the time of manufacture. In contrast, the FPGA's function is defined by a program written by someone other than the device manufacturer. Depending on the particular device, the program is either ’burned’ in permanently or semi-permanently as part of a board assembly process, or is loaded from an external memory each time the device is powered up. This user programmability gives the user access to complex integrated designs without the high engineering costs associated with application specific integrated circuits.

5.2 FPGA BACKGROUND

In the 1970s, a series of read-only memory (ROM)-based programmable devices were introduced and provided a new way to implement logic functions. Although mask-programmable ROMs and fuse-programmable ROMs (PROMs) with *N* address inputs can implement any *N*-input logic function, area efficiency quickly becomes an issue for all but small values of *N* due to the exponential dependence of area on *N*. The first programmable logic arrays (PLAs) improved on this with two-level AND–OR logic planes (each plane in a wired-AND or wired-OR structure along with inverters can build any AND or OR logic term) that closely match the structure of common logic functions and are significantly more area-efficient. An example PLA is shown in Fig.5.1.



These architectures evolved further with the realization that sufficient flexibility was provided by a programmable AND plane followed by a fixed OR plane, in the programmable array logic (PAL) devices that were introduced in 1977 by Monolithic Memories Incorporated (MMI). As shown in Fig. 5.2, these devices contained programmable combinational logic which fed fixed sequential logic in the form of D-type flip–flop macro cells. With these devices, logic functions must be implemented using one or more levels of two-level logic structures. Device inputs and intermediate combinational sums are fed into the array via a programmable interconnect that is typically a full cross-bar, leading to significant interconnect costs for this programmable architecture. For data path and multi-level circuits, the area costs of two-level implementation quickly become prohibitive.

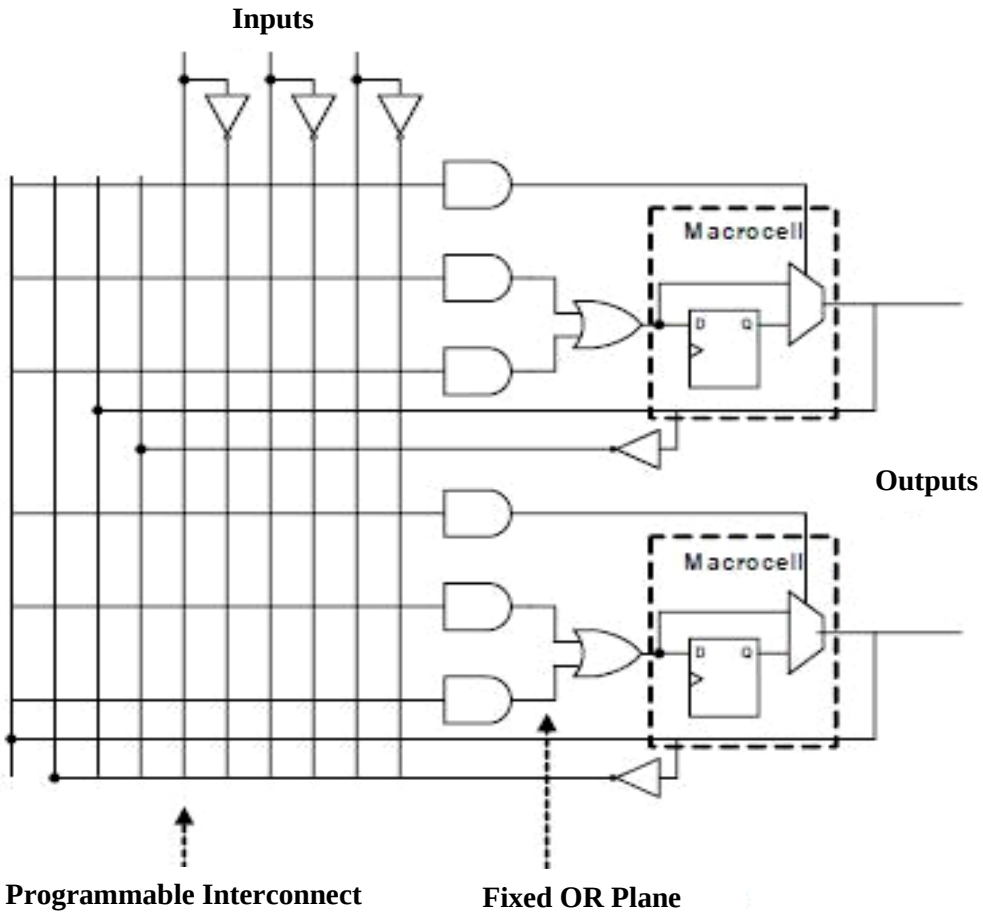


Fig. 5.2. PAL Architecture

In the late 1980s, the Naval Surface Warfare Department funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gates. Casselman was successful and a patent related to the system was issued in 1992. Xilinx Co-

Founders Ross Freeman and Bernard Vonderschmitt invented the first commercially viable field programmable gate array in 1985- the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 boasted a mere 64 configurable logic blocks (CLB), with the two 3-input look-up tables (LUTs). Xilinx continued unchallenged and quickly growing from 1985 to the mid-1990s, when competitors sprouted up, eroding significant market-share. By 1993, Actel was serving about 18 percent of the market. FPGAs got a glimpse of fame in 1997, when Adrian Thompson, a researcher working at the University of Sussex, merged genetic algorithm technology and FPGAs to create a sound recognition device. Thomson's algorithm configured an array of 10 x 10 cells in a Xilinx FPGA chip to discriminate between two tones, utilizing analogue features of the digital chip. The application of genetic algorithms to the configuration of devices like FPGA's is now referred to as Evoluble hardware.

FPGA devices are one of the modern technologies that are changing the electronic industry. FPGAs are riding the same integrated circuit process curves as processors and memories and keep getting larger, faster, and cheaper. FPGAs are now common in low and mid volume embedded products where they offer the following advantages:

- Fast time to market
- Better integration
- In system programmability
- FPGAs tend to have long life cycles and are usually replaced with pin compatible parts.

A recent trend has been to take the coarse-grained architectural approach a step further by combining the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals to form a complete "system on a programmable chip. That work was done in 1982. Examples of such hybrid technologies can be found in the Xilinx Virtex-II PRO and Virtex-4 devices, which include one or more PowerPC processors embedded within the FPGA's logic fabric.

An alternate approach to using hard-macro processors is to make use of soft processor cores that are implemented within the FPGA logic. As previously mentioned, many modern FPGAs have the ability to be reprogrammed at "run time," and this is leading to the idea of reconfigurable computing or reconfigurable systems — CPUs that reconfigure themselves to suit the task at hand. Additionally, new, non-FPGA architectures are beginning to emerge. Software-configurable

microprocessors such as the Stretch S5000 adopt a hybrid approach by providing an array of processor cores and FPGA-like programmable cores on the same chip.The first static memory-based FPGA (commonly called an SRAM based FPGA) was proposed by Wahlstrom in 1967.

This architecture allowed for both logic and interconnection configuration using a stream of configuration bits. Unlike its contemporary cellular array counterparts, both wide-input logic functions and storage elements could be implemented in each logic cell. Additionally, the programmable inter-cell connections could be easily changed (through memory-configurability) to enable the implementation of a variety of circuit topologies. Although static memory offers the most flexible approach to device programmability, it requires a significant increase in area per programmable switch compared to ROM implementations. It is likely this issue delayed the introduction of commercial static memory-based programmable devices until the mid-1980's, when the cost per transistor was sufficiently lowered.

The first modern-era FPGA was introduced by Xilinx in 1984. It contained the now classic array of Configurable Logic Blocks. From that first FPGA which contained 64 logic blocks and 58 inputs and outputs, FPGAs have grown enormously in complexity. Modern FPGAs now can contain approximately 330,000 equivalent logic blocks and around 1100 inputs and outputs in addition to a large number of more specialized blocks that have greatly expanded the capabilities of FPGAs.

5.3 FPGA COMPARISONS

FPGA has the ability to re-program in the field to fix bugs, and may include a shorter time to market and lower non-recurring engineering costs. Vendors can also take a middle road by developing their hardware on ordinary FPGAs, but manufacture their final version so it can no longer be modified after the design has been committed.

Historically, FPGAs have been slower, less energy efficient and generally achieved less functionality than their fixed ASIC counterparts. A study has shown that designs implemented on FPGAs need on average 18 times as much area, draw 7 times as much dynamic power, and are 3 times slower than the corresponding ASIC implementations.

Xilinx claims that several market and technology dynamics are changing the ASIC/FPGA paradigm. Integrated Circuits costs are rising aggressively

- ASIC complexity has lengthened development time
- R&D resources and headcount are decreasing
- Revenue losses for slow time-to-market are increasing
- Financial constraints in a poor economy are driving low-cost technologies

These trends make FPGAs a better alternative than ASICs for a larger number of higher-volume applications than they have been historically used for, to which the company attributes the growing number of FPGA design starts.

5.3.1 FPGA VERSUS COMPLEX PROGRAMMABLE LOGIC DEVICES

The primary differences between CPLDs and FPGAs are architectural. A CPLD has a somewhat restrictive structure consisting of one or more programmable sum-of-products logic arrays feeding a relatively small number of clocked registers. The result of this is less flexibility, with the advantage of more predictable timing delays and a higher logic-to-interconnect ratio. The FPGA architectures, on the other hand, are dominated by interconnect. This makes them far more flexible (in terms of the range of designs that are practical for implementation within them) but also far more complex to design for.Another notable difference between CPLDs and FPGAs is the presence in most FPGAs of higher-level embedded functions (such as adders and multipliers) and embedded memories, as well as to have logic blocks implements decoders or mathematical functions.

- FPGAs are "fine-grain" devices - that means that they contain a lot (up to 100000) of tiny blocks of logic with flip-flops. CPLDs are "coarse-grain" devices - they contain relatively few (a few 100's max) large blocks of logic with flip-flops.
- FPGAs are RAM based - they need to be "downloaded" (configured) at each power-up. CPLDs are EEPROM based - they are active at power-up (i.e. as long as they've been programmed at least once...).
- CPLDs have faster input - to - output timings than FPGAs (because of their coarse-grain architecture), so are better suited for microprocessor decoding logic for example than FPGAs.
- FPGAs have special routing resources to implement efficiently arithmetic functions (binary counters, adders, comparators...). CPLDs do not.

In general, FPGAs can contain large digital designs, while CPLDs can contain small designs only.

5.3.2 FPGAS VS. MICROCONTROLLERS

- FPGAs implement programmable logic elements running in a parallel fashion.
- Microcontrollers are based on a CPU architecture (executes a set of instructions in a sequential manner).

Microcontrollers have on-chip peripherals that also execute in parallel with their CPU. But they are still much less configurable than FPGAs.

5.4 APPLICATIONS

Applications of FPGAs include digital signal processing, software defined radio, aerospace and defence systems, ASIC prototyping, medical imaging, computer vision, speech reorganization, cryptography, bioinformatics, radio astronomy, metal detection and a growing range of other areas.

FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SoC). Particularly with the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications which had traditionally been the sole reserve of DSPs began to incorporate FPGAs instead. FPGAs especially find applications in any area or algorithm that can make use of

the massive parallelism offered by their architecture. FPGAs are increasingly used in conventional high performance computing applications where computational kernels such as FFT or Convolution are performed on the FPGA instead of a microprocessor.

The inherent parallelism of the logic resources on an FPGA allows for considerable computational throughput even at a low MHz clock rates. The flexibility of the FPGA allows for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs. The adoption of FPGAs in high performance computing is currently limited by the complexity of FPGA design compared to conventional software and the turn-around times of current design tools.

5.5 FPGA PROGRAMMABLE LOGIC

A field programmable gate array is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex logic functions such as decoders or simple mathematical functions. In most FPGAs, these programmable logic components (or logic blocks, in FPGA parlance) also include memory elements, which may be simple flip-flops or more complete blocks of memories. A hierarchy of programmable interconnects allows the logic blocks of FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. These logic blocks and interconnects can be programmed after the manufacturing process by the customer/designer to implement any logical function- hence field programmable.

FPGAs are generally slower than their application specific integrated circuit (ASIC) counterparts, as they can't handle as complex a design, and draw more power. However, they have several advantages such as shorter time to market, ability to re-program in field to fix.

5.6 FPGA BASICS

FPGA: Field Programmable Gate Array. Conceptually it can be considered as an array of Configurable Logic Blocks (CLBs) that can be connected together through a vast interconnection matrix to form complex digital circuits. FPGAs have traditionally found use in high-speed custom digital applications where designs tend to be more constrained by performance rather than cost.

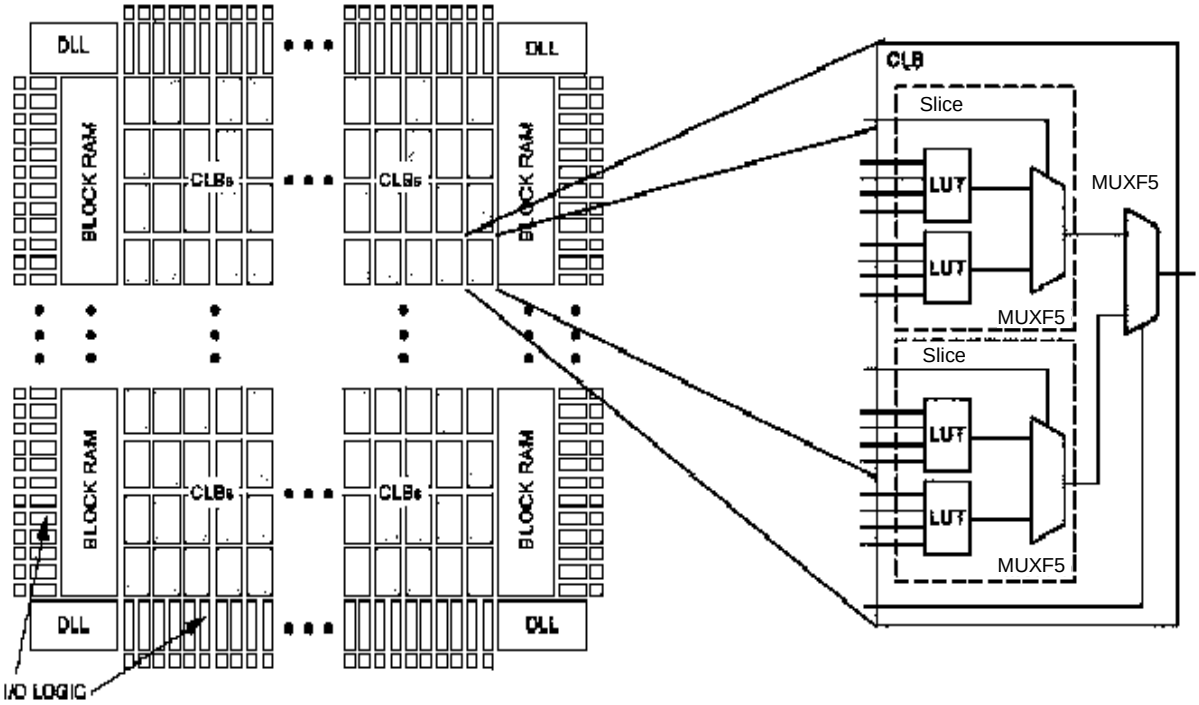
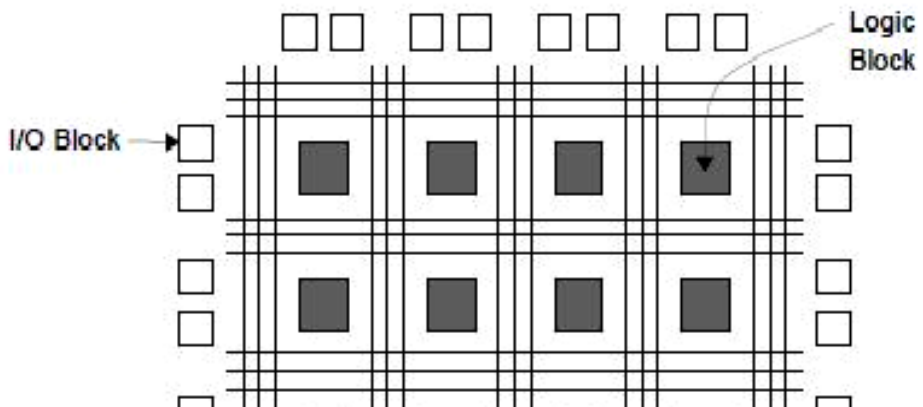


Fig. 5.3. View of Typical FPGA

FPGAs, along with their non-volatile versions CPLDs (Complex Programmable Logic Devices), are emerging as the next digital revolution that will bring about change in much the same way that microprocessors did.

With current high-end devices exceeding 2000 pins and topping billions of transistors, the complexity of these devices is such that it would be impossible to program them without the assistance of high-level design tools. Xilinx, Altera, Actel, and Lattice all offer high-end EDA tool suites designed specifically to support their own devices however they also offer free versions aimed at supporting the bulk of FPGA development. These vendors understand the importance of tool availability to increased silicon sales and they all seem committed to supporting a free version of their tools for the foreseeable future.

Field- Programmable Gate Arrays (FPGAs) comprise an array of uncommitted circuit elements, called *logic blocks*, and interconnect resources, but FPGA configuration is performed through programming by the end user. An illustration of a typical FPGA architecture appears in Figure 5.3. As the only type of FPD that supports very high logic capacity, FPGAs have been responsible for a major shift in the way digital circuits are designed.



5.7 ARCHITECTURE

The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array. An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most designs that will fit in terms of LUTs and IOs can be routed..

In general, a logic block (CLB or LAB) consists of a few logical cells (called ALM, LE, Slice etc). A typical cell consists of a 4-input Look-Up (LUT), a Full Adder(FA) and a D-type flip-flop, as shown below. The LUT are in this figure split into two 3-input LUTs. In *normal mode* those are combined into a 4-input LUT through the left mux. In *arithmetic* mode, their outputs are fed to the FA. The selection of mode is programmed into the middle mux. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right, in the figure example. In practice, entire or parts of the FA are put as functions into the LUTs in order to save space.

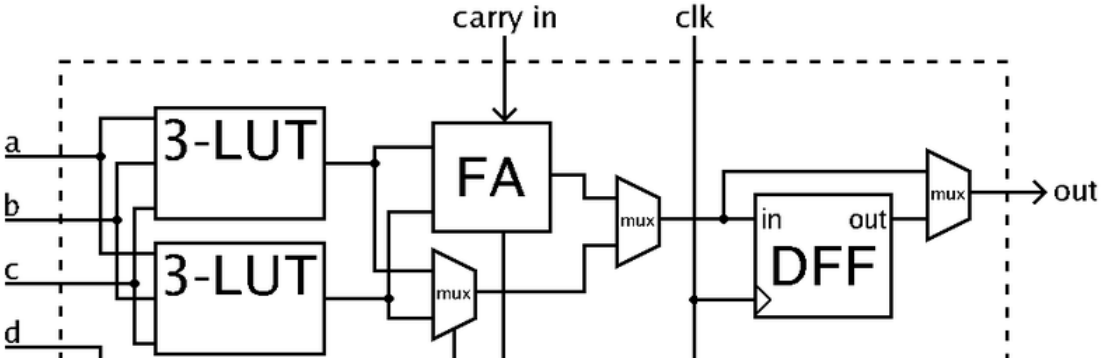


Fig. 5.5. Simplified Example Illustration of a Logic Cell

In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance. Since clock signals (and often other high fan-out signals) are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed. A completely different architecture was introduced in the mid-1980's that uses RAM-based lookup tables instead of AND-OR gates to implement combinational logic.

These devices are called field programmable gate arrays (FPGAs). The device consists of an array of configurable logic blocks (CLBs) surrounded by an array of I/O blocks. The Spartan-3E from Xilinx also contains some blocks of RAM, 18 x 18 multipliers, as well as Digital Clock Manager (DCM) blocks. These DCMs are used to eliminate clock distribution delay and can also increase or decrease the frequency of the clock. Each CLB in the Spartan-3E FPGA contains four slices, each of which contains two 16 x 1 RAM look-up tables (LUTs), which can implement any combinational logic function of four variables. In addition to two look-up tables, each slice contains two D flip-flops which act as storage devices for bits. The basic architecture of a Spartan-3E FPGA is shown in Fig. 5.6.

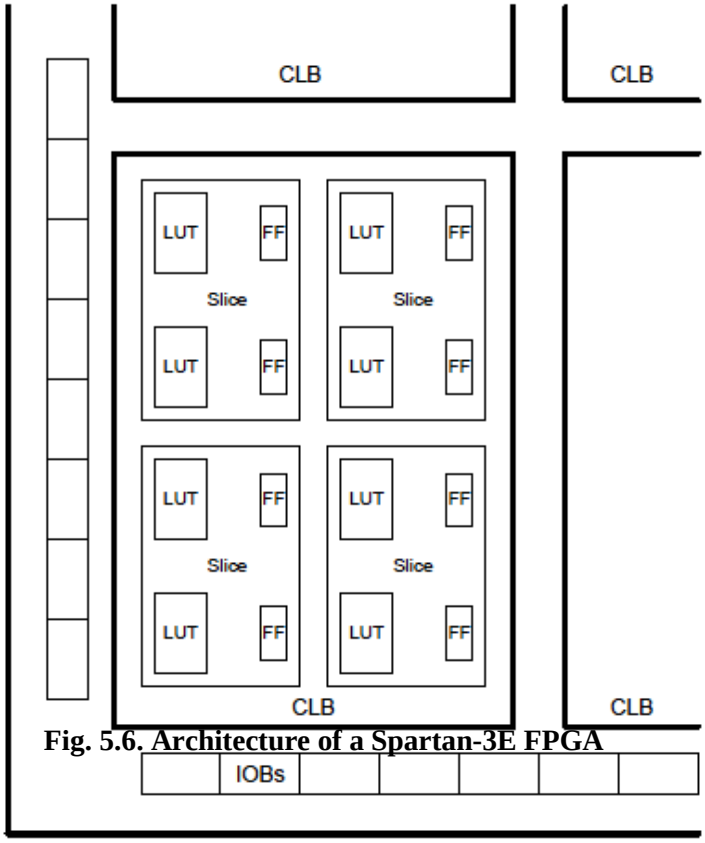


Fig. 5.6. Architecture of a Spartan-3E FPGA

Modern FPGA far [illegible] functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives.

Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time-to-market.

5.8 FPGA CONFIGURATION

An FPGA can be into 2 states: "configuration mode" or "user mode". When the FPGA wakes up after power-up, it is in configuration mode, sitting idle with all its outputs inactive. User need to configure it. Configuring an FPGA means downloading a stream of 0's and 1's into it through some special pins. Once the FPGA is configured, it goes into "user-mode" and becomes active, performing accordingly to your programmed "logic function".

There are 3 classical ways to configure your FPGA:

- Use of a cable from the PC to the FPGA, and run software on your PC to send data through the cable.
- Use of a microcontroller on the board, with an adequate firmware to send data to the FPGA.
- Use of a "boot-PROM" on the board, connected to the FPGA, that configures the FPGA automatically at power-up (FPGA vendors have such special boot-PROMs in their catalogs).

During development, the first method is the easiest and quickest. Once your FPGA design works, we probably don't need the PC anymore, so the other 2 methods come in use. FPGA configuration can quickly become a complex subject. Development boards usually come with a special cable that we can use to configure the FPGA from PC with no knowledge of the underlying interface.

5.9 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.

5.10 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.

5.10.1 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. 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 this work Direct form of FIR filter has been implemented whose new look is given in Figure 5.7.

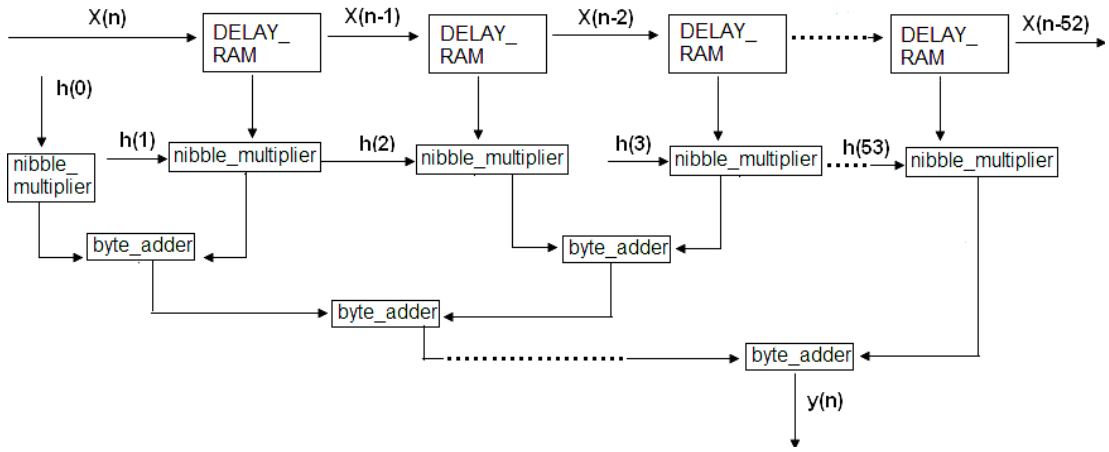


Fig. 5.7. Direct-Form Structure

5.10.2 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 Kaiser windowing techniques.
3. Filter is considered as symmetric.

CHAPTER 6 RESULTS AND DISCUSSION

In this chapter, the lowpass, highpass, bandpass and bandreject filters are implemented on FPGA. The filter specifications are real world and Kaiser 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 10.1.

6.1 DESIGN OF FIR FILTER

Filtering is a technique to modify the frequency property of the input signal x (n) to meet certain design requirements. Broadly, it can be said as a numerical procedure, or algorithm that transfers a given sequence of No‘s into a second sequence that has some more desirable properties, such as less noise or distortion. A digital filter consist of the inter connection of three simple elements, adders, multipliers and delays.

6.2 DIGITAL FIR FILTER DESIGN PROBLEM

Digital Filter Design Process : Digital filter design involves usually the following steps:

1. Determine a desired response or a set of desired responses (e.g., a desired magnitude response and/or a desired phase response).
2. Select a class of filters for approximating the desired response(s) (e.g., linear-phase FIR filters or IIR filters being implementable as a parallel connection of two allpass filters).
3. Establish a criterion of “goodness” for the response(s) of a filter in the selected class compared to the desired response(s).
4. Develop a method for finding the best number in the filter class.
5. Synthesize the best filter using a proper structure and a proper implementation form, for example using a computer program, a signal processor, or a VLSI chip.
6. Analyze the filter performance.

Filter Specifications : The requirements for a filter are normally specified in the frequency domain in terms of the desired magnitude response and/or the desired phase (delay) response. It is desired to preserve signal components in the region called *passband* of the filter, and to reject signal components in the region called the *stopband* of the filter.

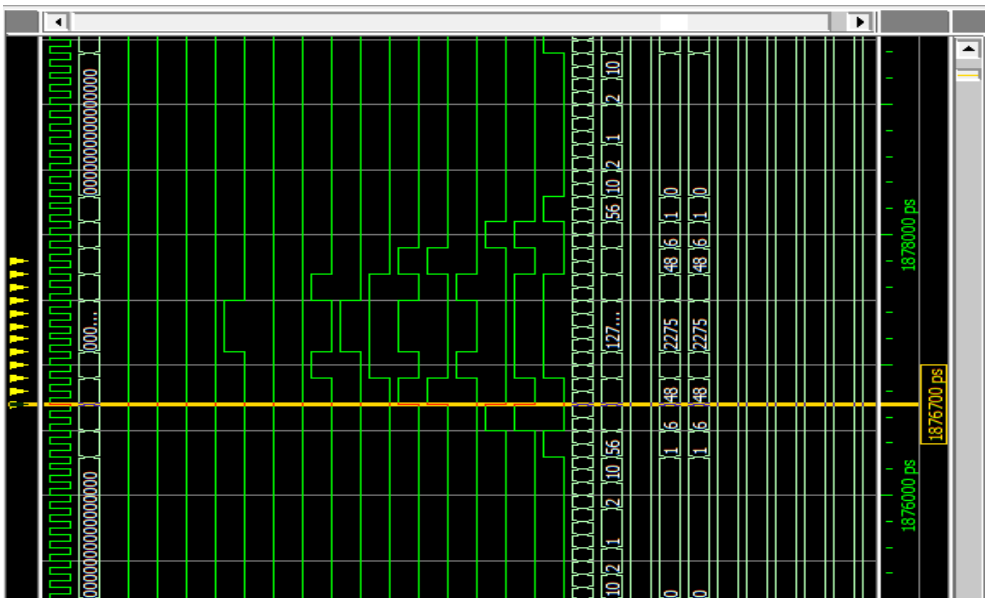
Approximation Criteria: Three different error measures are normally used in designing digital filters:

1. Minimax Error Designs
2. Least-Squared Error Designs
3. Maximally Flat Approximations

6.3 FIR FILTER DESIGN USING WINDOW TECHNIQUE

Fixed Window Functions: The Gibbs phenomenon can be reduced by using a less abrupt truncation of the Fourier series. This is achieved by using a window function that tapers smoothly towards zero at both ends. There are some of the well-known fixed window functions $\omega[n]$ depending on the type of the window like *Rectangular*, *Bartlett*, *Hann*, *Hamming* or *Blackman*. These linear-phase filters can also be characterized by the property that only M+1 multipliers are needed in the actual implementation because of the symmetry in the filter coefficients. For these fixed window functions, the only adjustable parameter is M, half the filter order. These window functions suffer from the drawback that stopband attenuation cannot be varied. Only passband and stopband edge can be adjusted by properly selecting ωc and M.

Adjustable Window Functions: The above problem can be overcome by using window functions having additional parameter with which stopband attenuation can be varied. There exist three approaches to obtaining good adjustable windows. The first alternative is to minimize the energy in the sidelobes of the frequency response of the window function, whereas the second one is to minimize the peak sidelobe ripple. Third alternative is to properly combine the first two approaches. Kaiser window provides optimum solution to the problem.



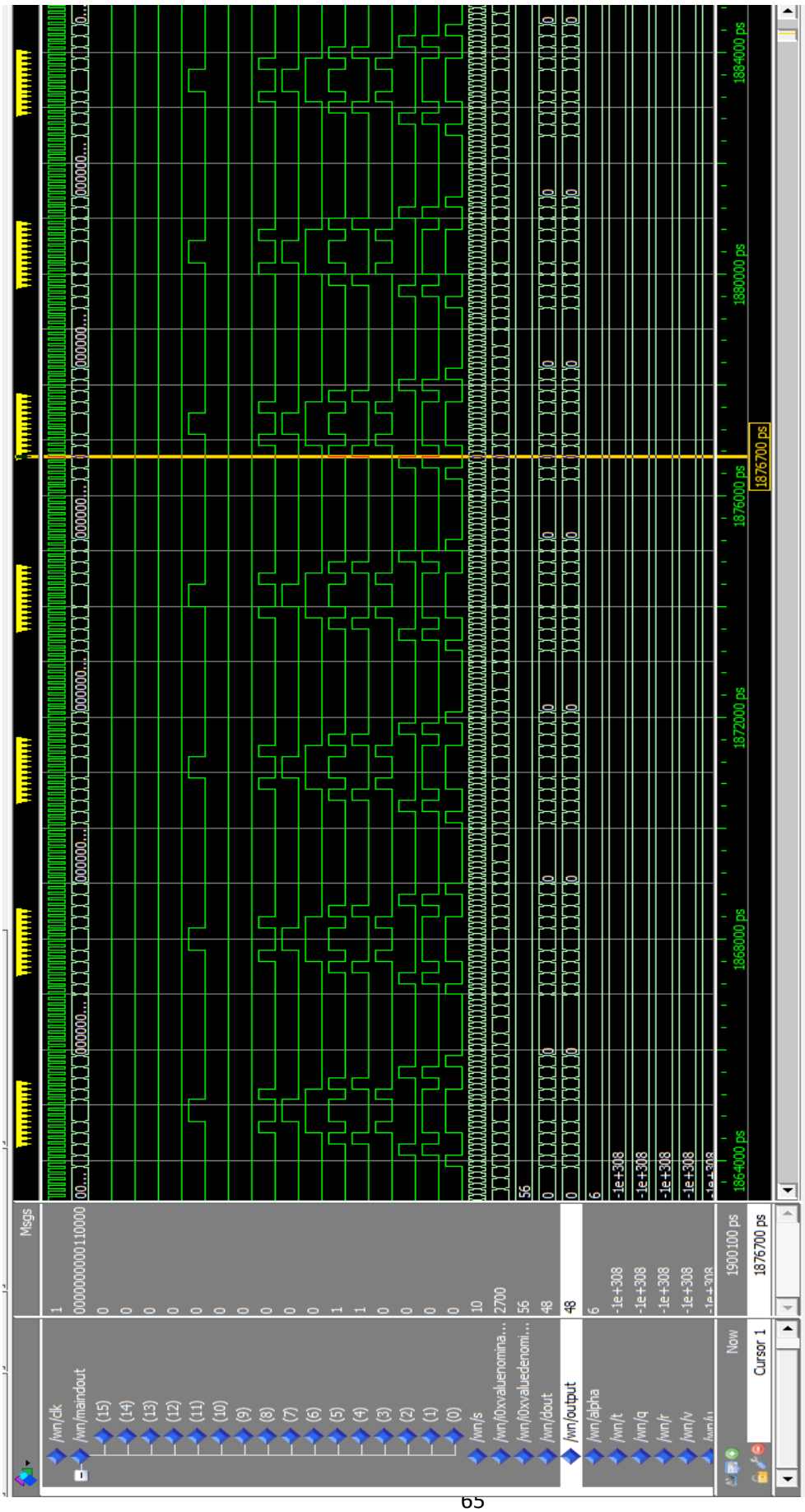


Fig. 6.2. Output of Kaiser Window Technique

6.4 LOW PASS FILTER

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

Maximum frequency of the system 200 MHz

Minimum frequency for the low band frequency 50 MHz

Signal frequency < 50MHz pass; otherwise block

The filter order of given specification is calculated using Kaiser window:

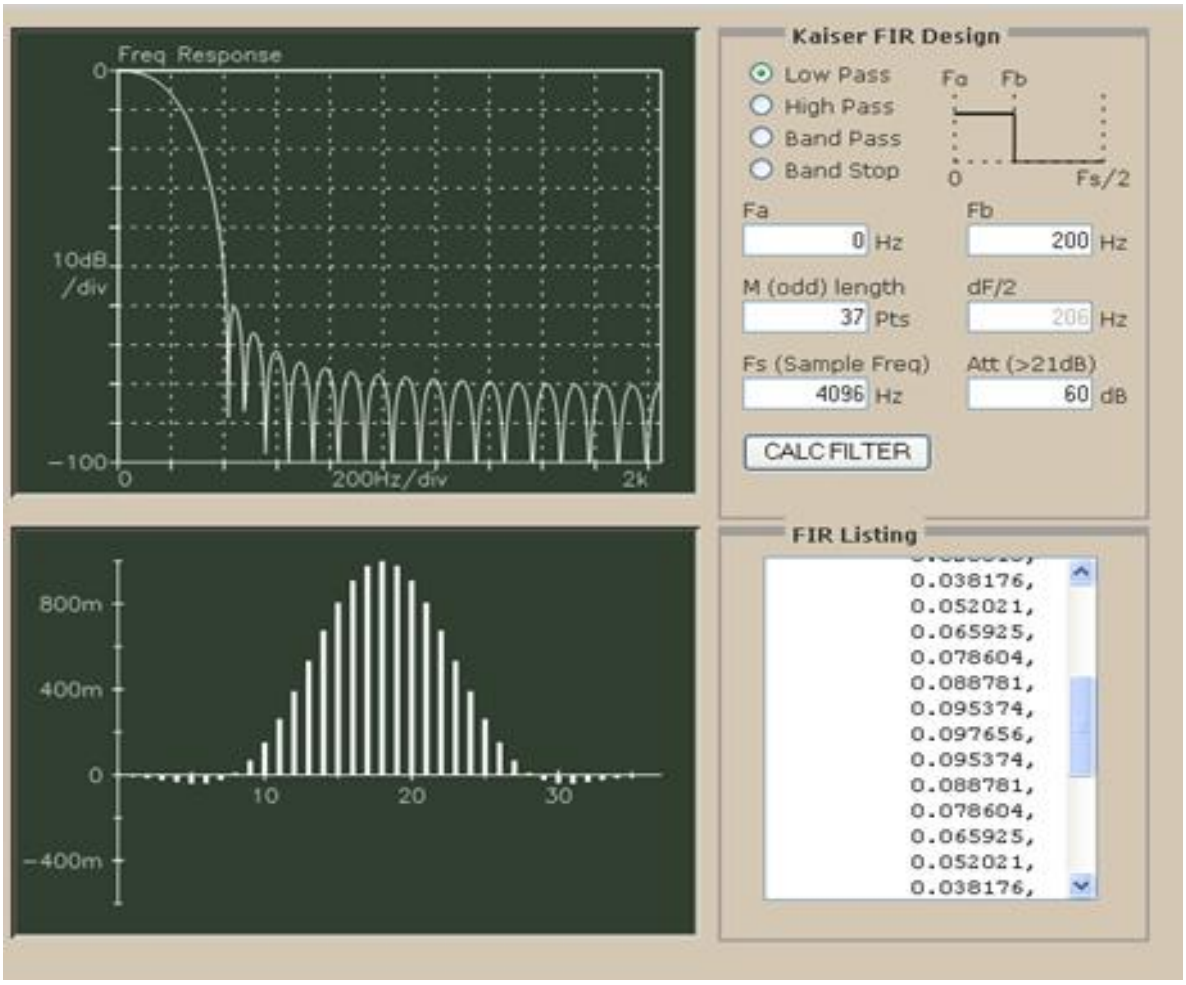


Fig. 6.3. Kaiser FIR Design for LPF

These coefficients are directly used 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 6.4.

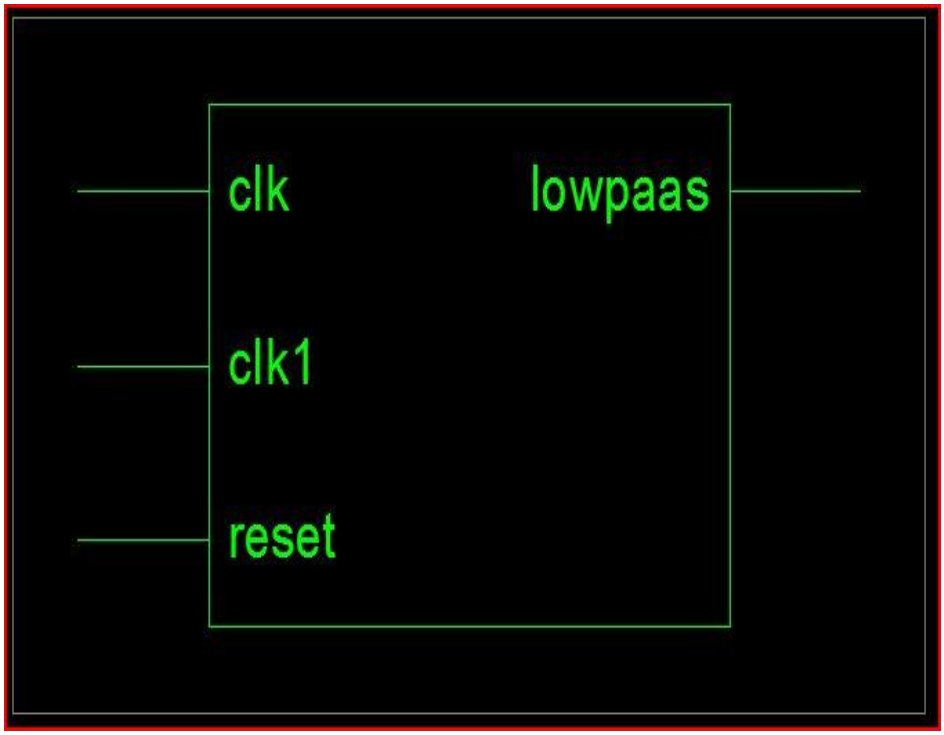


Fig. 6.4. Block Diagram of the Low Pass Filter (LPF).

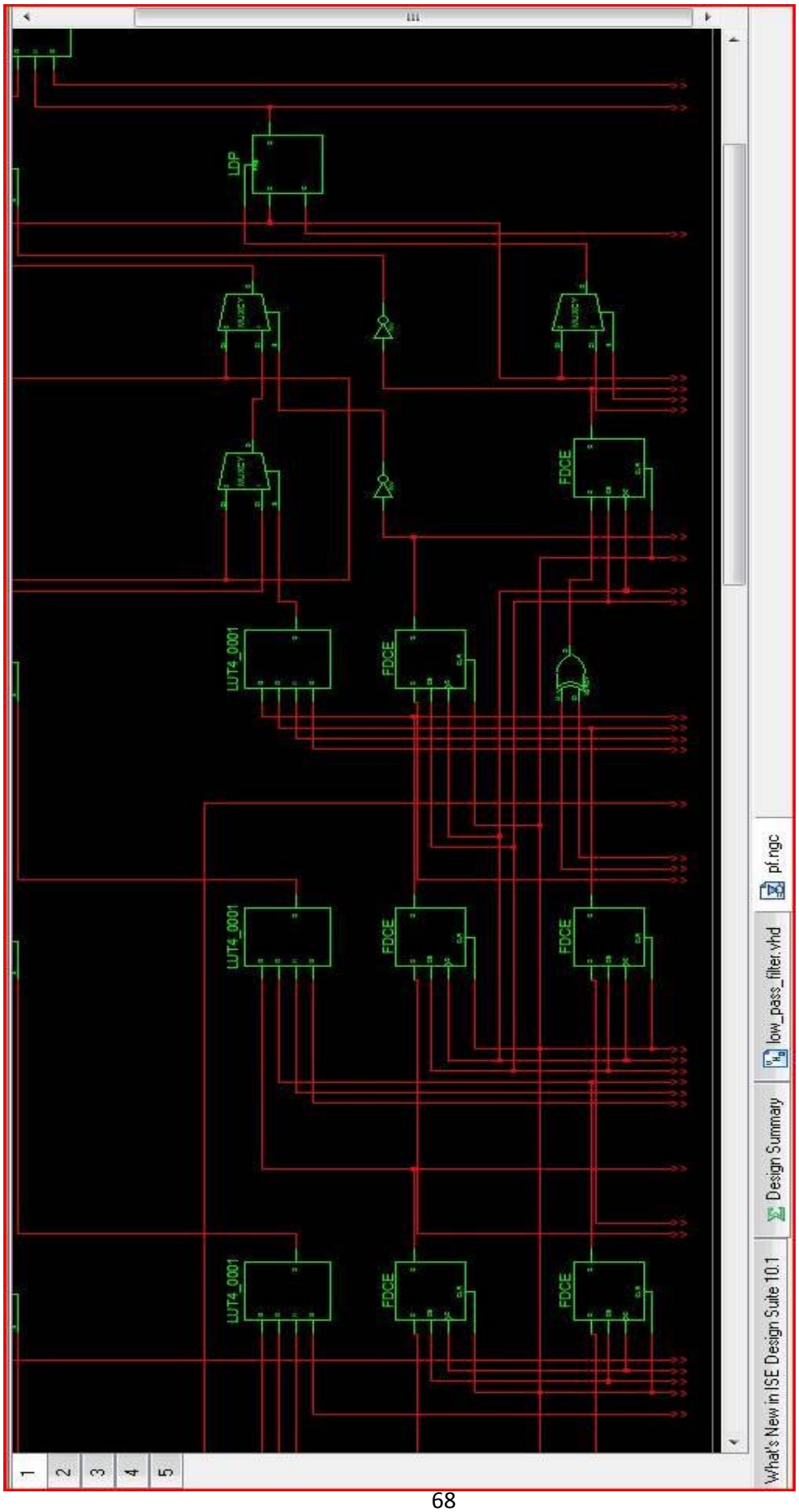


Fig. 6.5. Circuit Diagram of Low Pass Filter (Part 1)

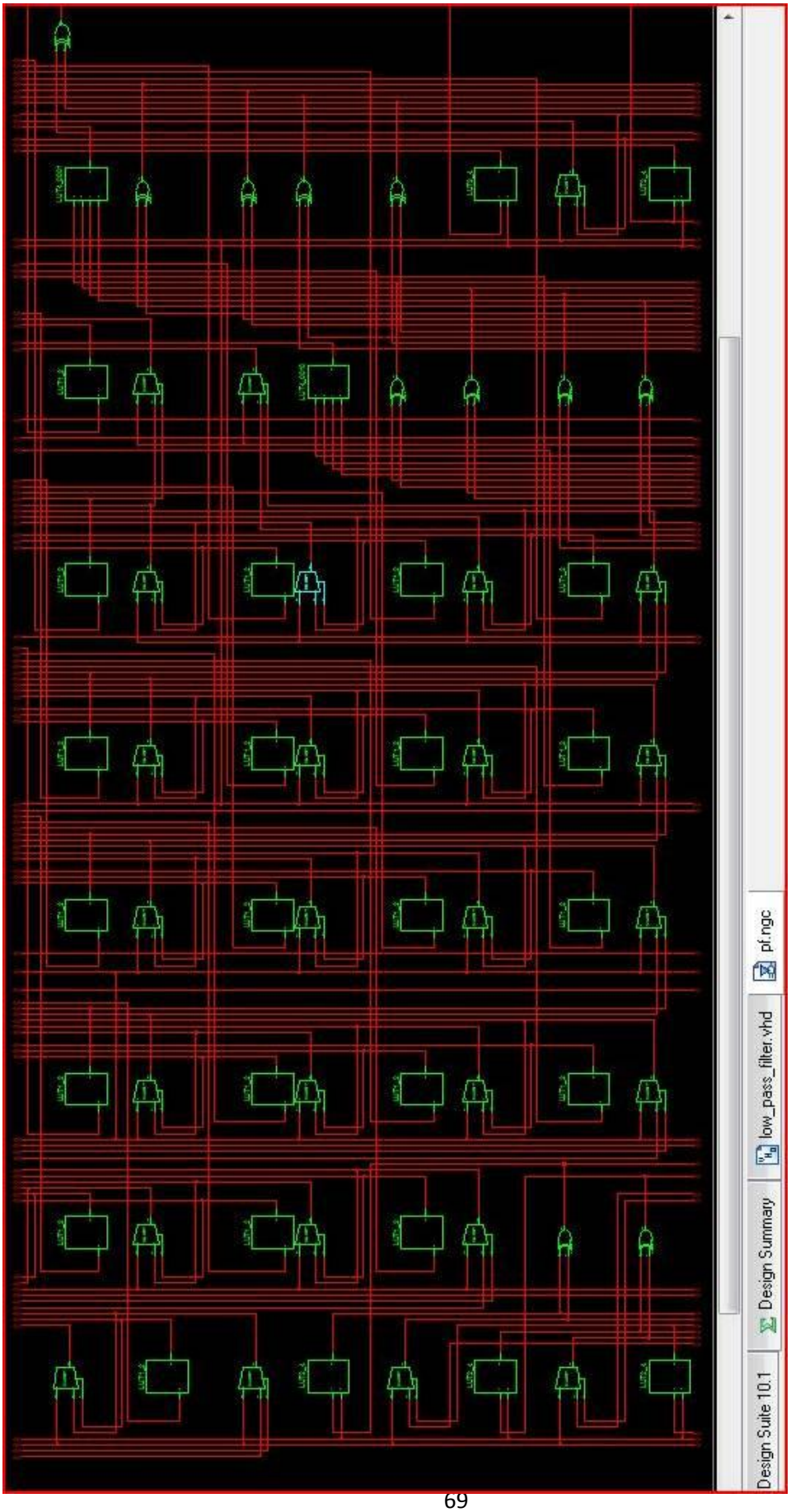


Fig. 6.6. Circuit Diagram of Low Pass Filter (Part 2)

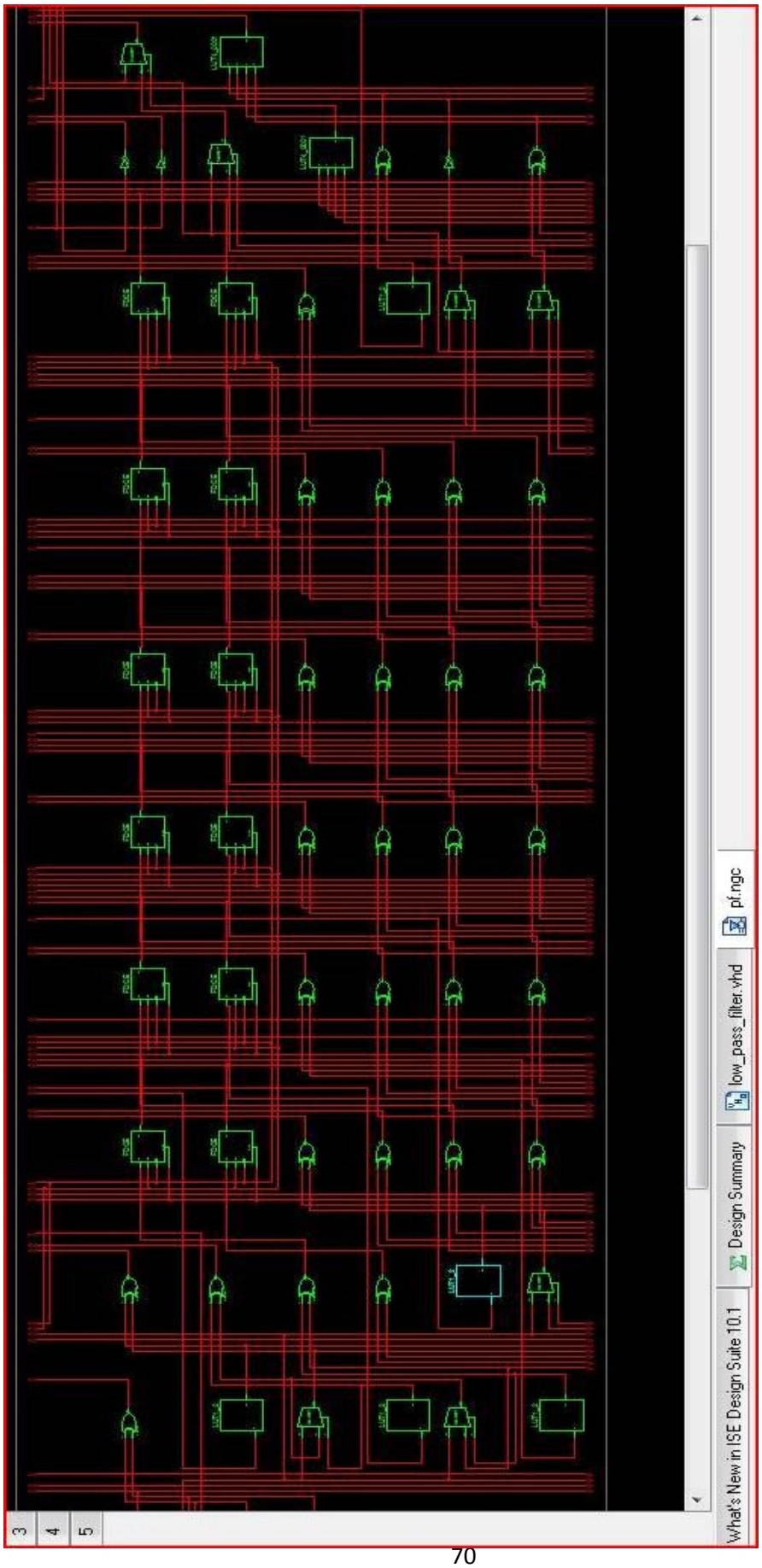


Fig. 6.7. Circuit Diagram of Low Pass Filter (Part 3)

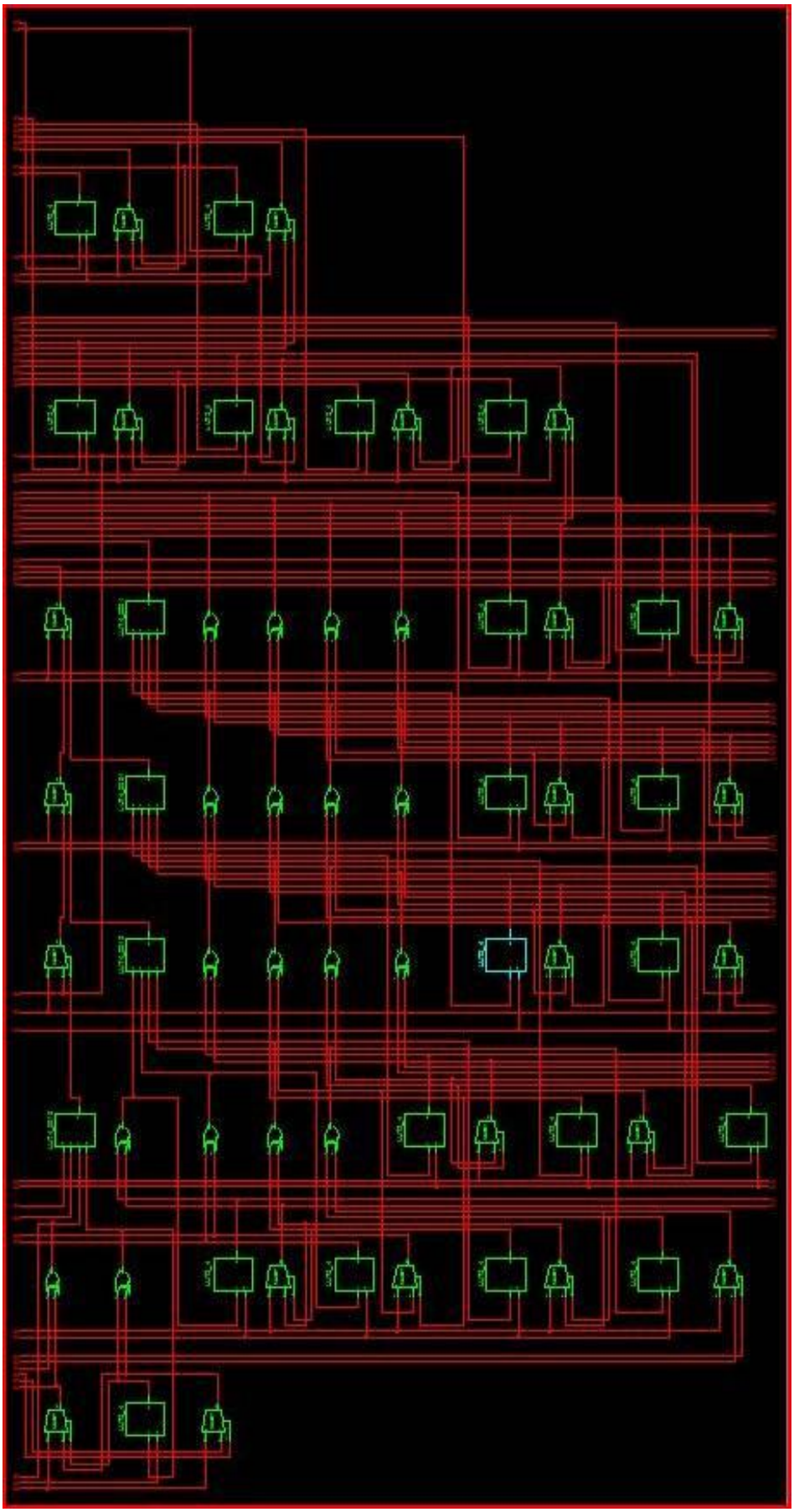


Fig. 6.8. Circuit Diagram of Low Pass Filter (Part 4)

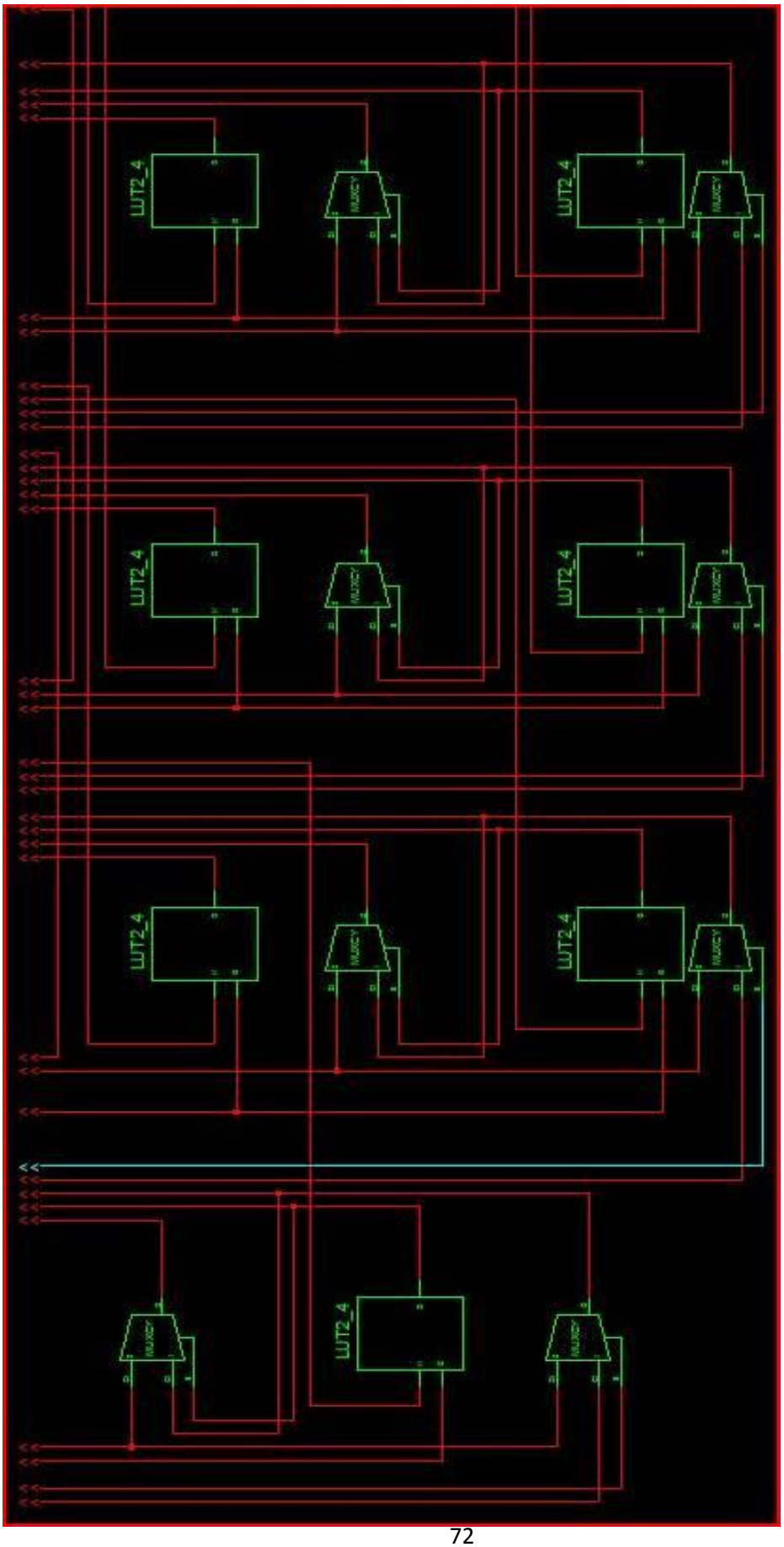


Fig. 6.9. Circuit Diagram of Low Pass Filter (Part 5)

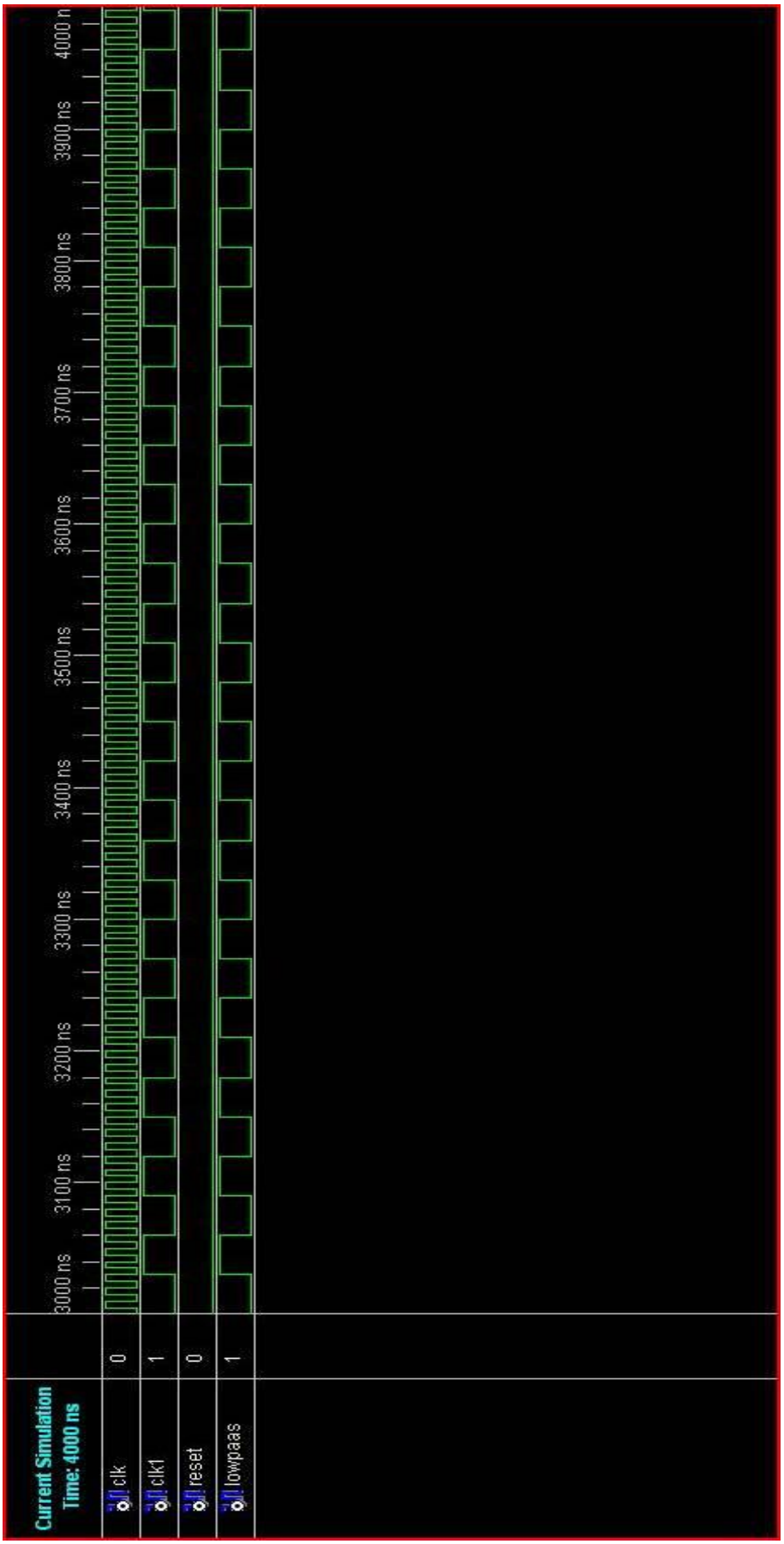


Fig. 6.10. Input Waveform of LPF

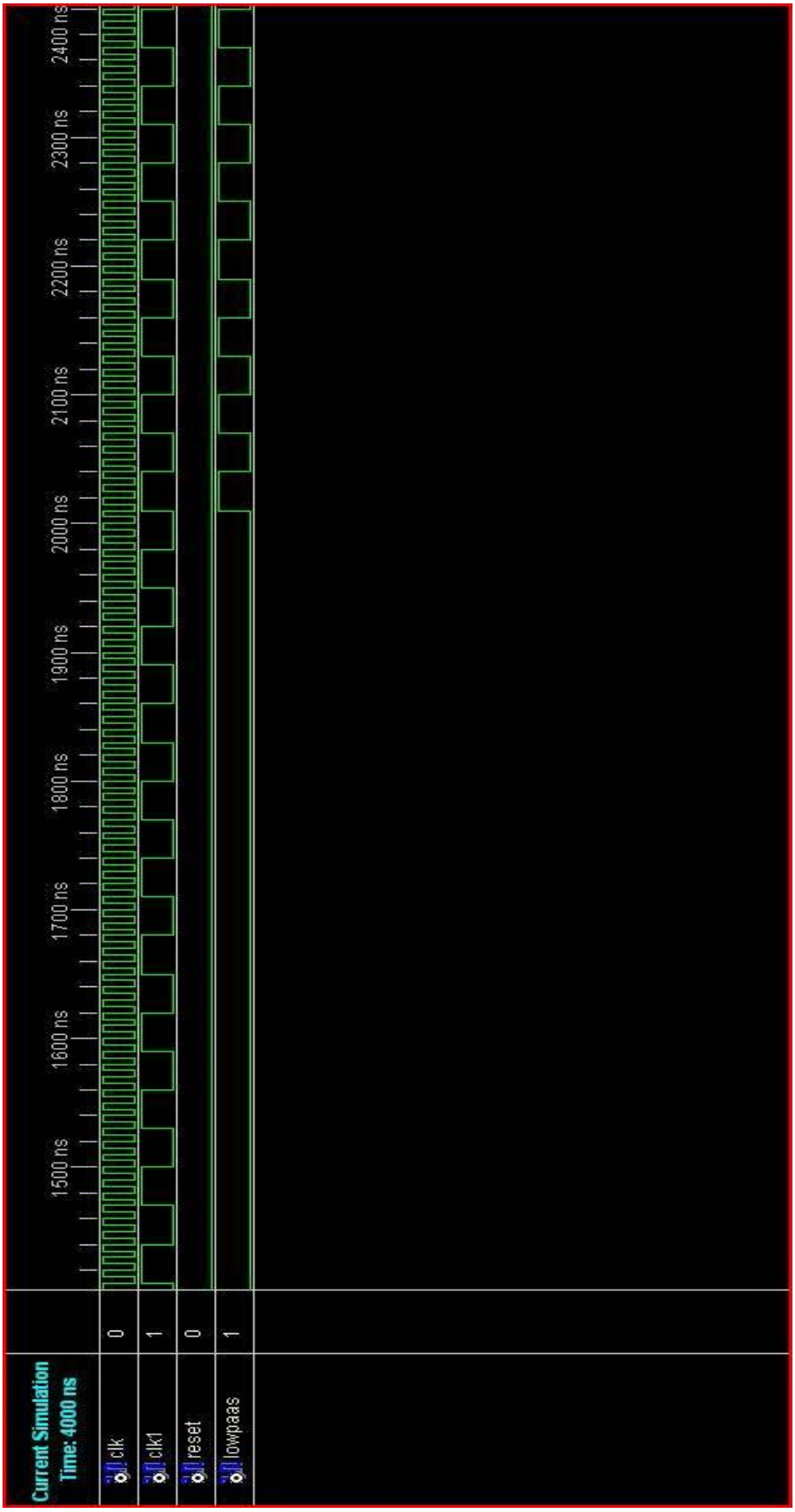
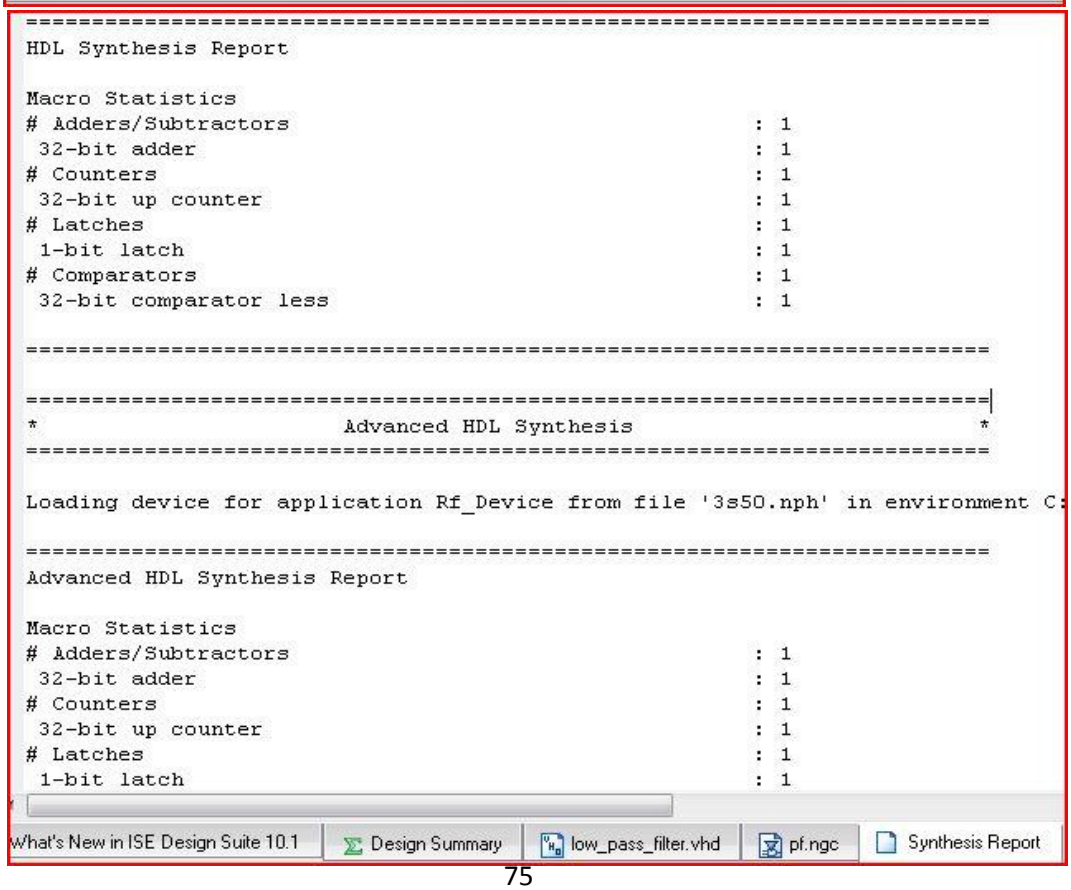


Fig. 6.11. Corresponding Output waveform of LPF

Table 6.1 Synthesis Report of Low Pass Filter

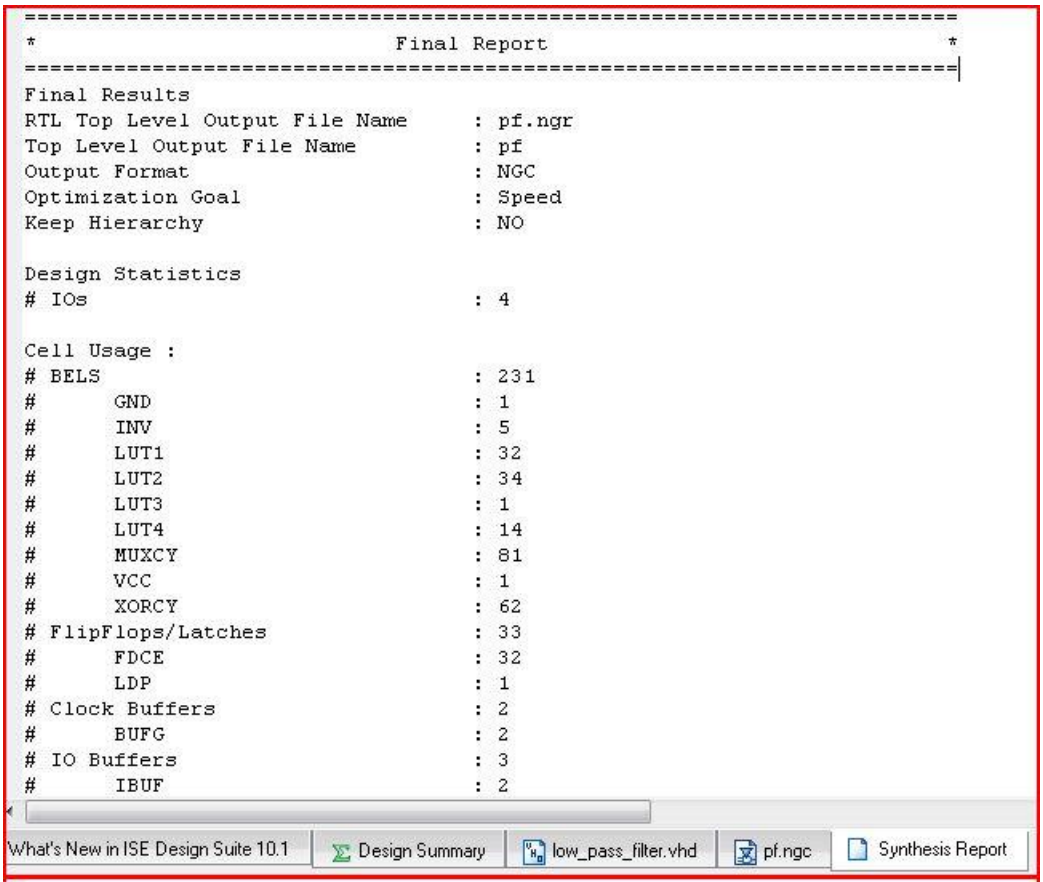
```
Total REAL time to Xst completion: 1.00 secs
Total CPU time to Xst completion: 0.17 secs

--> Parameter xsthdpdir set to C:/Users/tanveetkaur/Desktop/allpaasfilters/xst


Total REAL time to Xst completion: 1.00 secs
Total CPU time to Xst completion: 0.19 secs

--> Reading design: pf.prj

TABLE OF CONTENTS
  1) Synthesis Options Summary
  2) HDL Compilation
  3) Design Hierarchy Analysis
  4) HDL Analysis
  5) HDL Synthesis
     5.1) HDL Synthesis Report
  6) Advanced HDL Synthesis
     6.1) Advanced HDL Synthesis Report
  7) Low Level Synthesis
  8) Partition Report
  9) Final Report
     9.1) Device utilization summary
     9.2) Partition Resource Summary
     9.3) TIMING REPORT


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

What's New in ISE Design Suite 10.1 | Design Summary | low_pass_filter.vhd | pf.ngc | Synthesis Report



```
=========================================================================
HDL Synthesis Report

Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit adder                                         : 1
# Counters                                             : 1
 32-bit up counter                                    : 1
# Latches                                              : 1
 1-bit latch                                          : 1
# Comparators                                          : 1
 32-bit comparator less                               : 1

=========================================================================

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

Loading device for application Rf_Device from file '3s50.nph' in environment C:

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

Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit adder                                         : 1
# Counters                                             : 1
 32-bit up counter                                    : 1
# Latches                                              : 1
 1-bit latch                                          : 1
```

What's New in ISE Design Suite 10.1 | Design Summary | low_pass_filter.vhd | pf.ngc | Synthesis Report

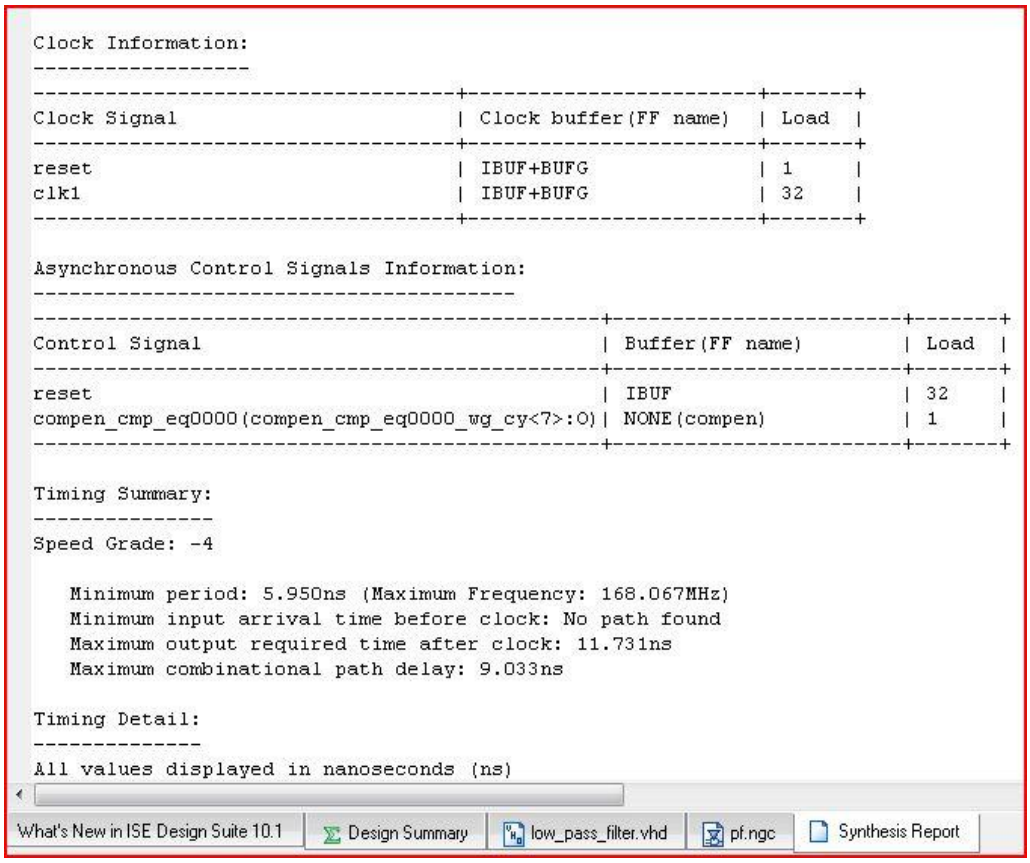


```
=========================================================================
*                          Final Report                                 *
=========================================================================
Final Results
RTL Top Level Output File Name     : pf.ngr
Top Level Output File Name         : pf
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 4

Cell Usage :
# BELS                             : 231
#      GND                         : 1
#      INV                         : 5
#      LUT1                        : 32
#      LUT2                        : 34
#      LUT3                        : 1
#      LUT4                        : 14
#      MUXCY                       : 81
#      VCC                         : 1
#      XORCY                       : 62
# FlipFlops/Latches                : 33
#      FDCE                        : 32
#      LDP                         : 1
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 3
#      IBUF                        : 2
```

What's New in ISE Design Suite 10.1 | Design Summary | low_pass_filter.vhd | pf.ngc | Synthesis Report

```
Device utilization summary:
---------------------------

Selected Device : 3s50tq144-4

 Number of Slices:                       44  out of     768     5%
 Number of Slice Flip Flops:             33  out of    1536     2%
 Number of 4 input LUTs:                 86  out of    1536     5%
 Number of IOs:                           4
 Number of bonded IOBs:                   3  out of      97     3%
 Number of GCLKs:                         2  out of       8    25%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
```

What's New in ISE Design Suite 10.1 | Design Summary | low_pass_filter.vhd | pf.ngc | Synthesis Report



```
Clock Information:
------------------
-----------------------------------+------------------------+-------+
Clock Signal                       | Clock buffer(FF name)  | Load  |
-----------------------------------+------------------------+-------+
reset                              | IBUF+BUFG              | 1     |
clk1                               | IBUF+BUFG              | 32    |
-----------------------------------+------------------------+-------+

Asynchronous Control Signals Information:
-----------------------------------------
------------------------------------------------+------------------------+-------+
Control Signal                                  | Buffer(FF name)        | Load  |
------------------------------------------------+------------------------+-------+
reset                                           | IBUF                   | 32    |
compen_cmp_eq0000(compen_cmp_eq0000_wg_cy<7>:O)| NONE(compen)           | 1     |
------------------------------------------------+------------------------+-------+

Timing Summary:
---------------
Speed Grade: -4

   Minimum period: 5.950ns (Maximum Frequency: 168.067MHz)
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: 11.731ns
   Maximum combinational path delay: 9.033ns

Timing Detail:
--------------
All values displayed in nanoseconds (ns)
```

What's New in ISE Design Suite 10.1 | Design Summary | low_pass_filter.vhd | pf.ngc | Synthesis Report

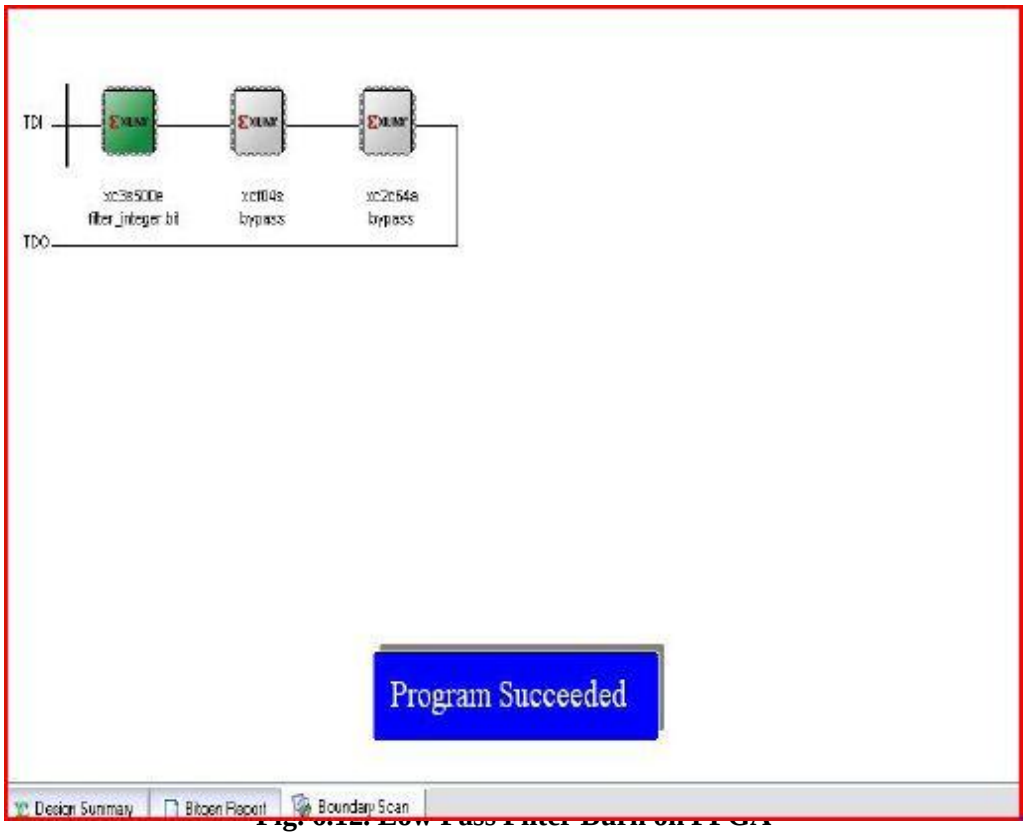


Fig. 6.12: Low Pass Filter Burnt on FPGA

6.5 HIGH PASS FILTER

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

Maximum frequency of the system	2048 Hz
Min frequency for the low band frequency	410 Hz

The filter order of given specification is calculated using Kaiser window:

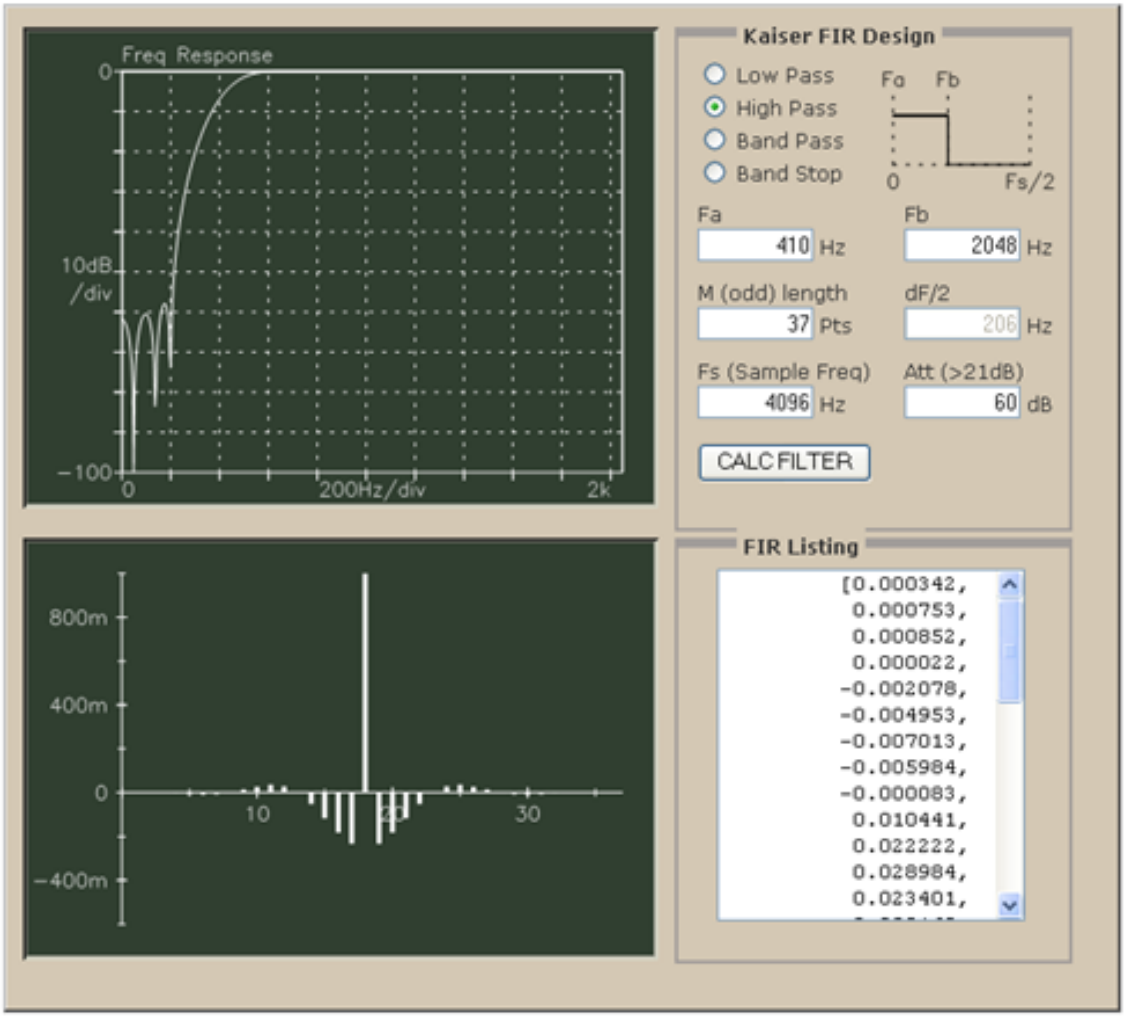


Fig. 6.13. Kaiser FIR Design for HPF

The coefficients are directly used 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 6.14.

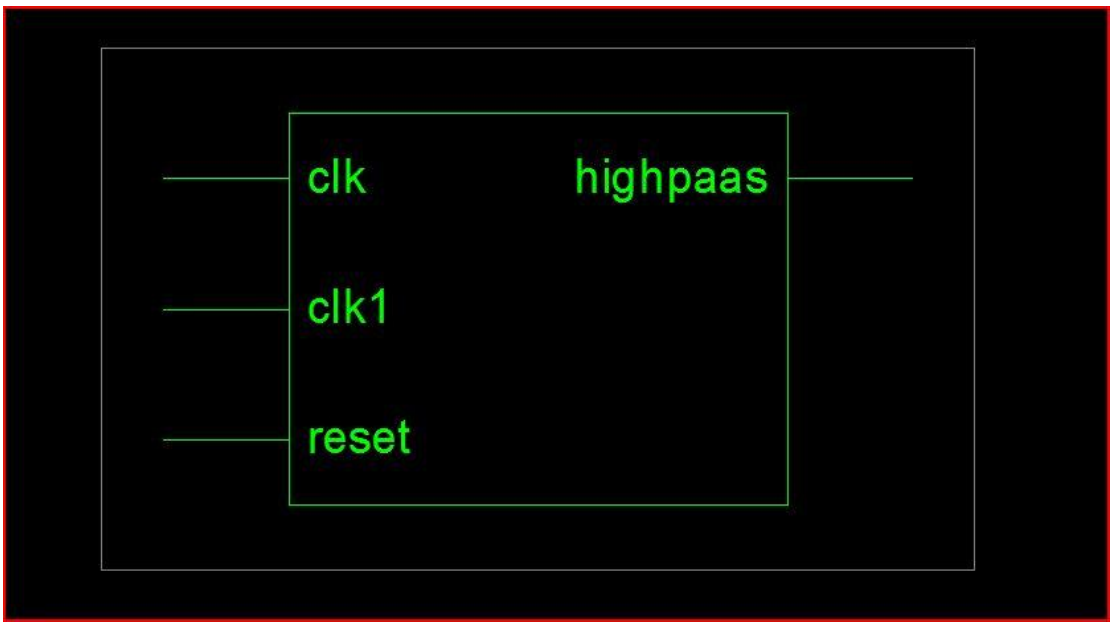


Fig. 6.14. Block Diagram of the High Pass Filter (HPF).

Maximum frequency of the system 200 MHz

High pass signal frequency > 100MHz ; pass otherwise block.

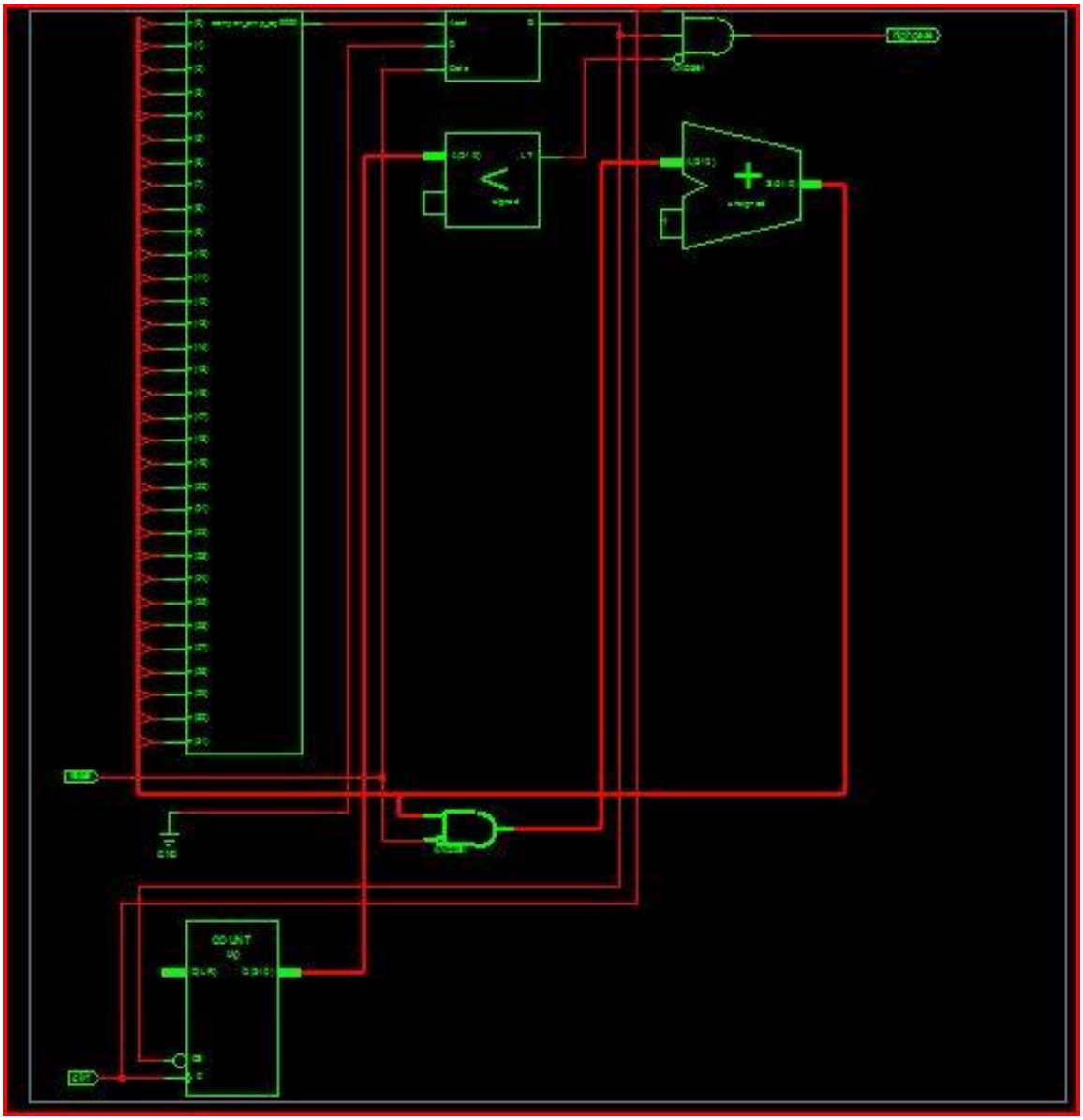


Fig. 6.15. Circuit Diagram of High Pass Filter (Part 1)

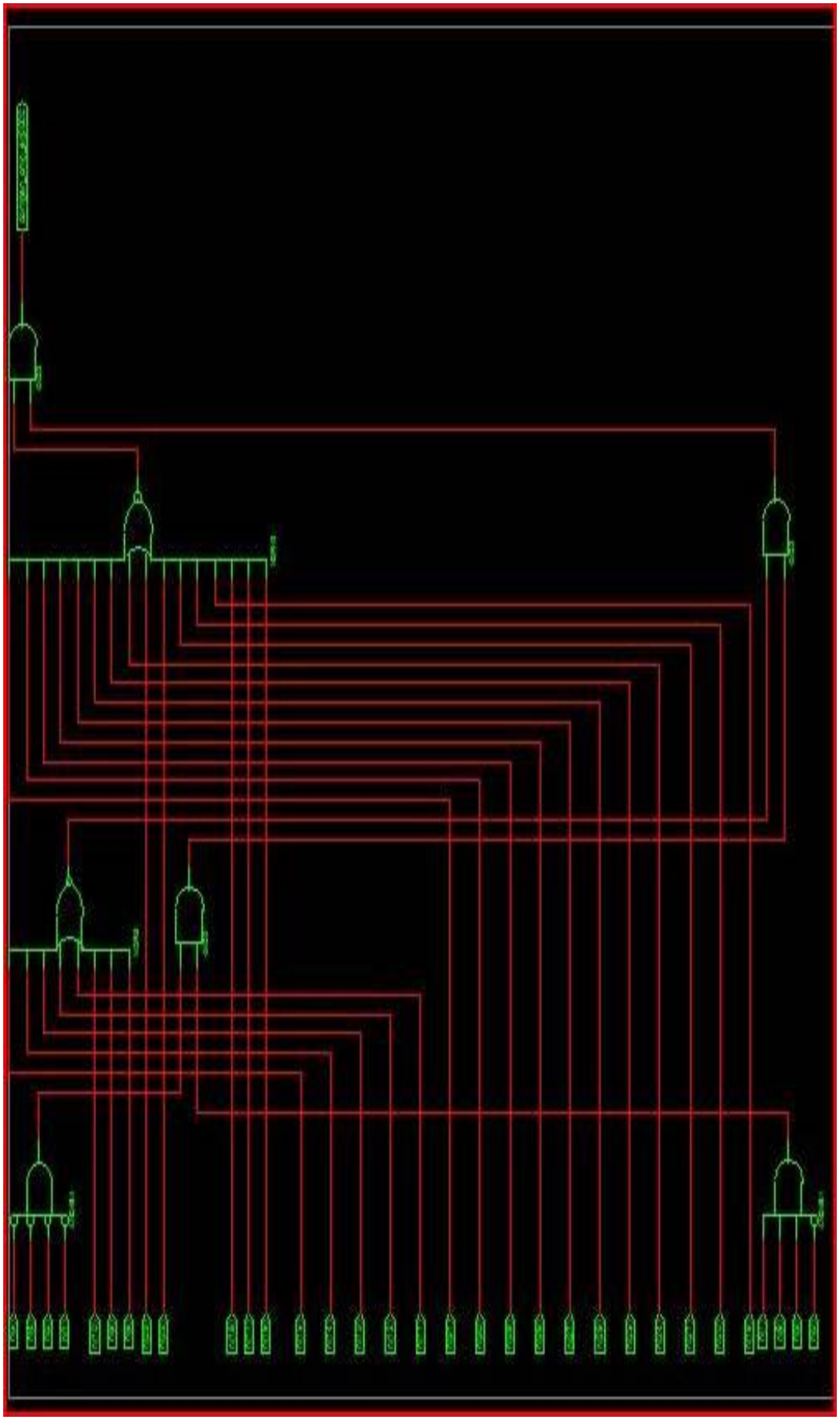


Fig. 6.16. Circuit Diagram of High Pass Filter (Part 2)

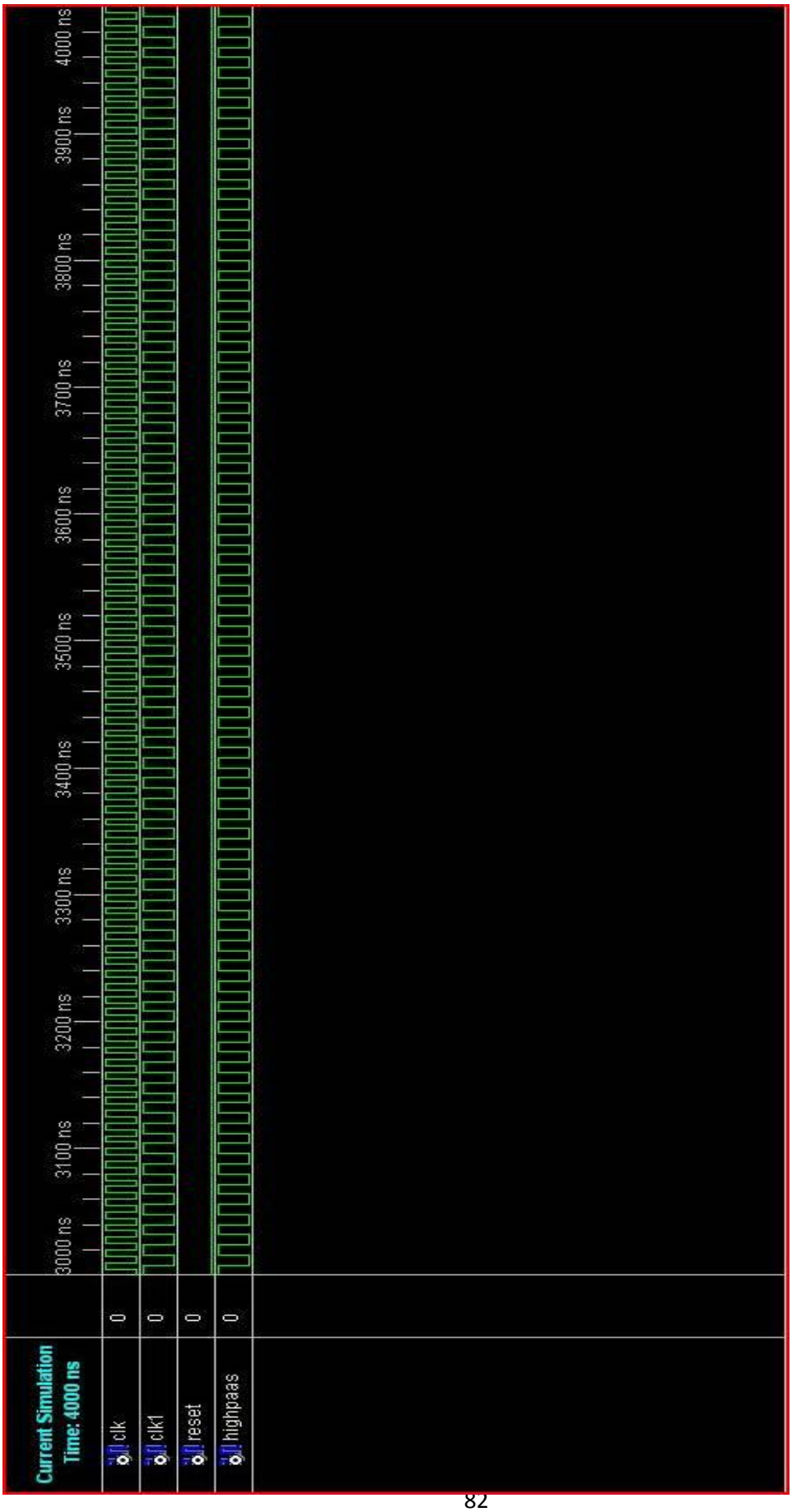


Fig. 6.17. Input Waveform of HPF

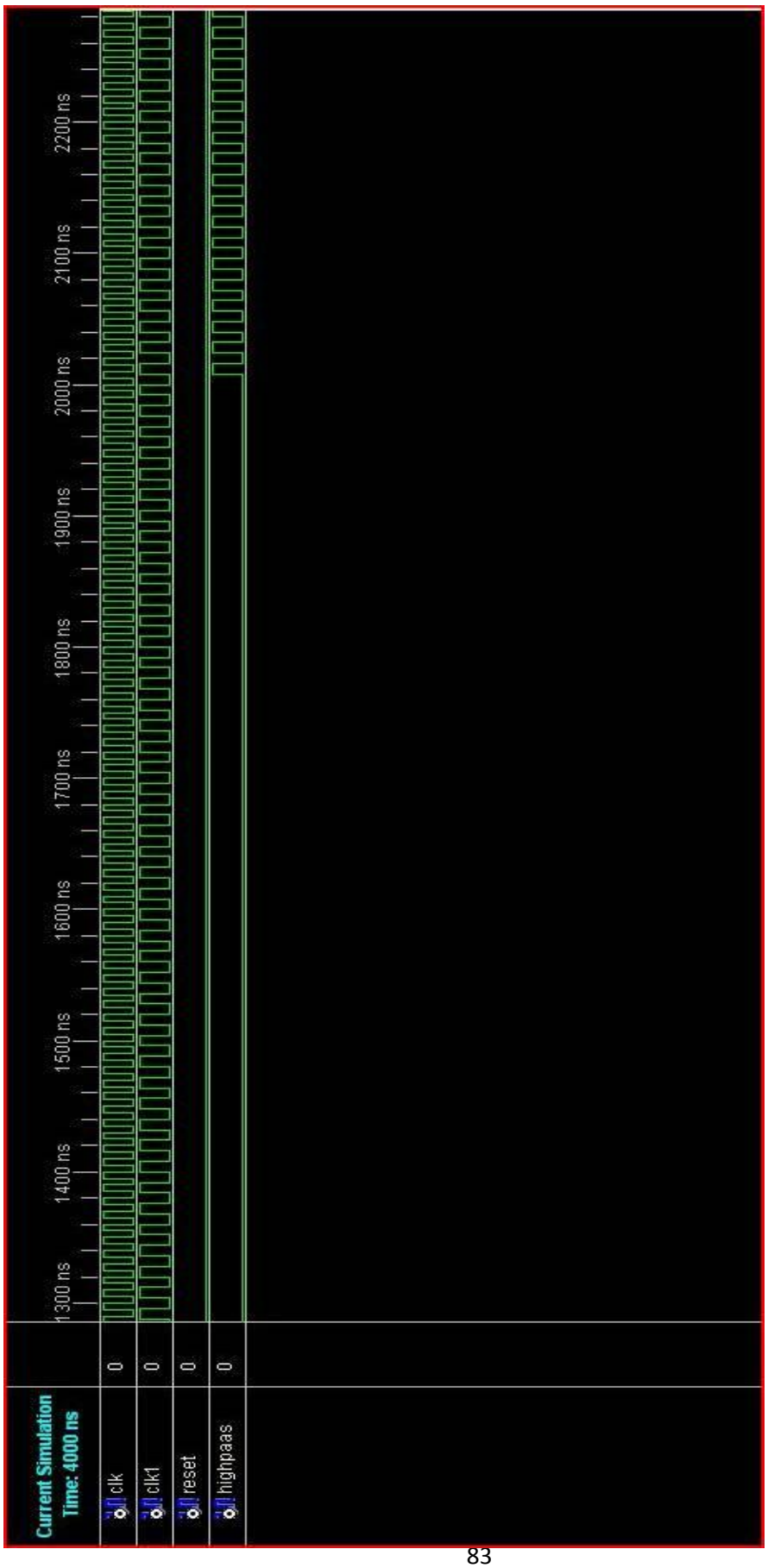


Fig. 6.18. Output Waveform of HPF.

Table 6.2 Synthesis Report of High Pass Filter

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : pf.ngr
Top Level Output File Name         : pf
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 4

Cell Usage :
# BELS                             : 251
#      GND                         : 1
#      INV                         : 6
#      LUT1                        : 33
#      LUT2                        : 36
#      LUT4                        : 21
#      MUXCY                       : 91
#      VCC                         : 1
#      XORCY                       : 62
# FlipFlops/Latches                : 33
#      FDCE                        : 32
#      LDP                         : 1
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 3
#      IBUF                        : 2
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan

```
Device utilization summary:
---------------------------

Selected Device : 3s50tq144-4

 Number of Slices:                       49  out of    768     6%
 Number of Slice Flip Flops:             33  out of   1536     2%
 Number of 4 input LUTs:                 96  out of   1536     6%
 Number of IOs:                           4
 Number of bonded IOBs:                   3  out of     97     3%
 Number of GCLKs:                         2  out of      8    25%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan

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

Loading device for application Rf_Device from file '3s50.nph' in environment C:\Xilinx\

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

Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit adder                                          : 1
# Counters                                             : 1
 32-bit up counter                                     : 1
# Latches                                              : 1
 1-bit latch                                           : 1
# Comparators                                          : 2
 32-bit comparator less                                : 2

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

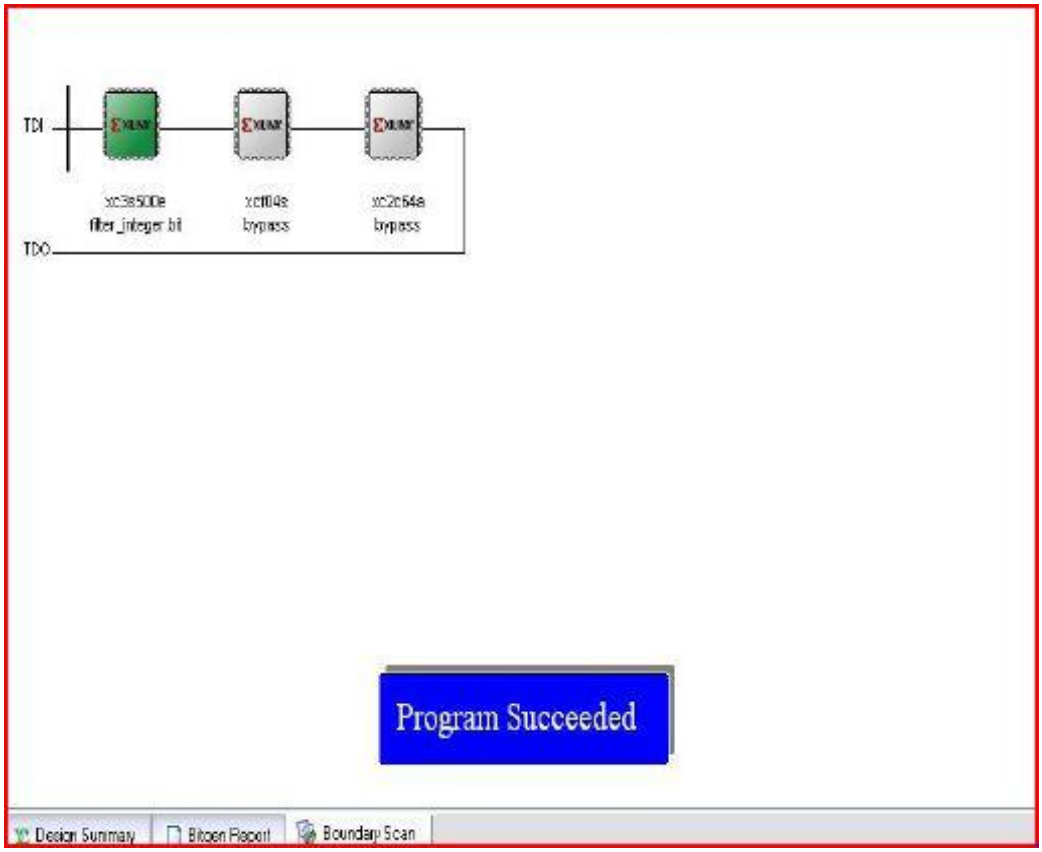


Fig. 6.19. High Pass Filter Burn on FPGA

6.6 BAND PASS FILTER

Following specifications are considered for lowpass filter

Band pass 50MHz < signal frequency < 100MHz pass; otherwise block.

The filter order of given specification is calculated using Kaiser window:

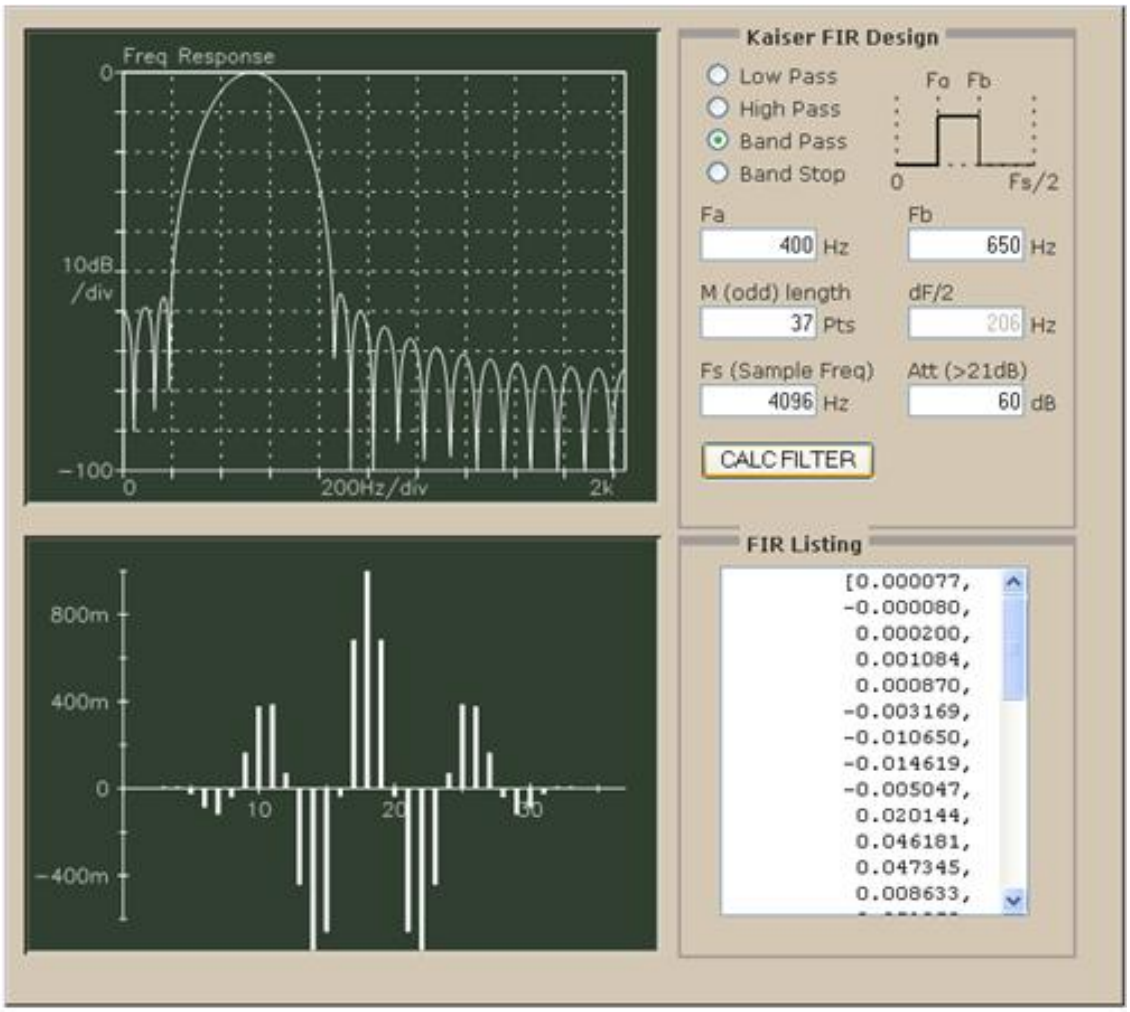


Fig. 6.20. Kaiser FIR Design for BPF

The coefficients are directly used 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 6.21.

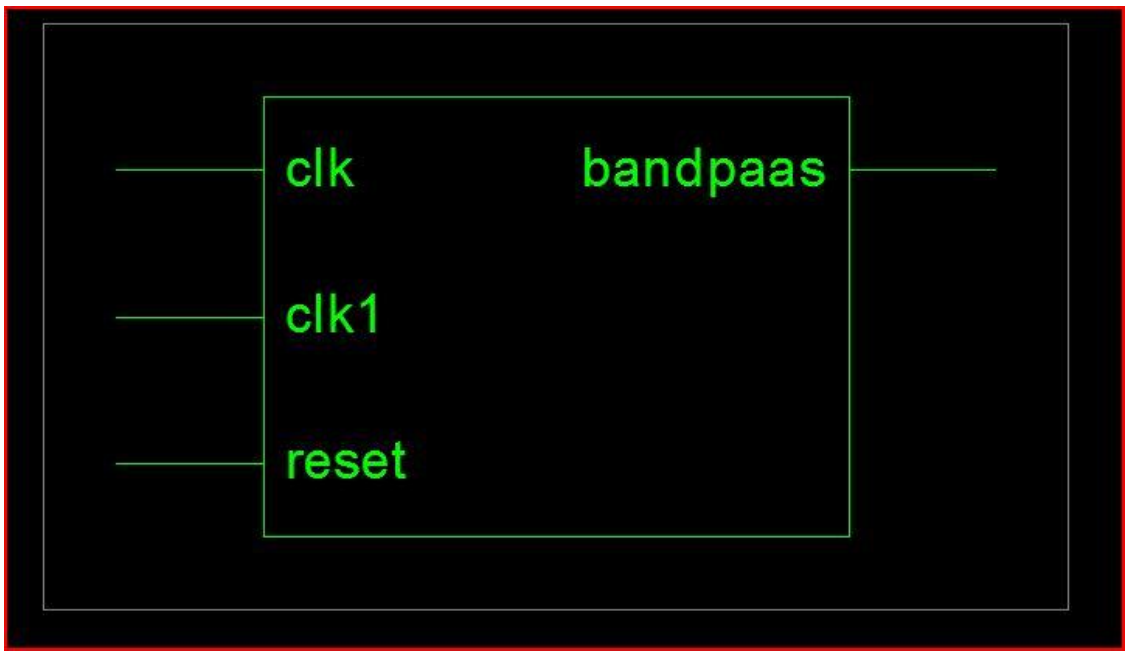


Fig. 6.21. Block Diagram of the Band Pass Filter (BPF).

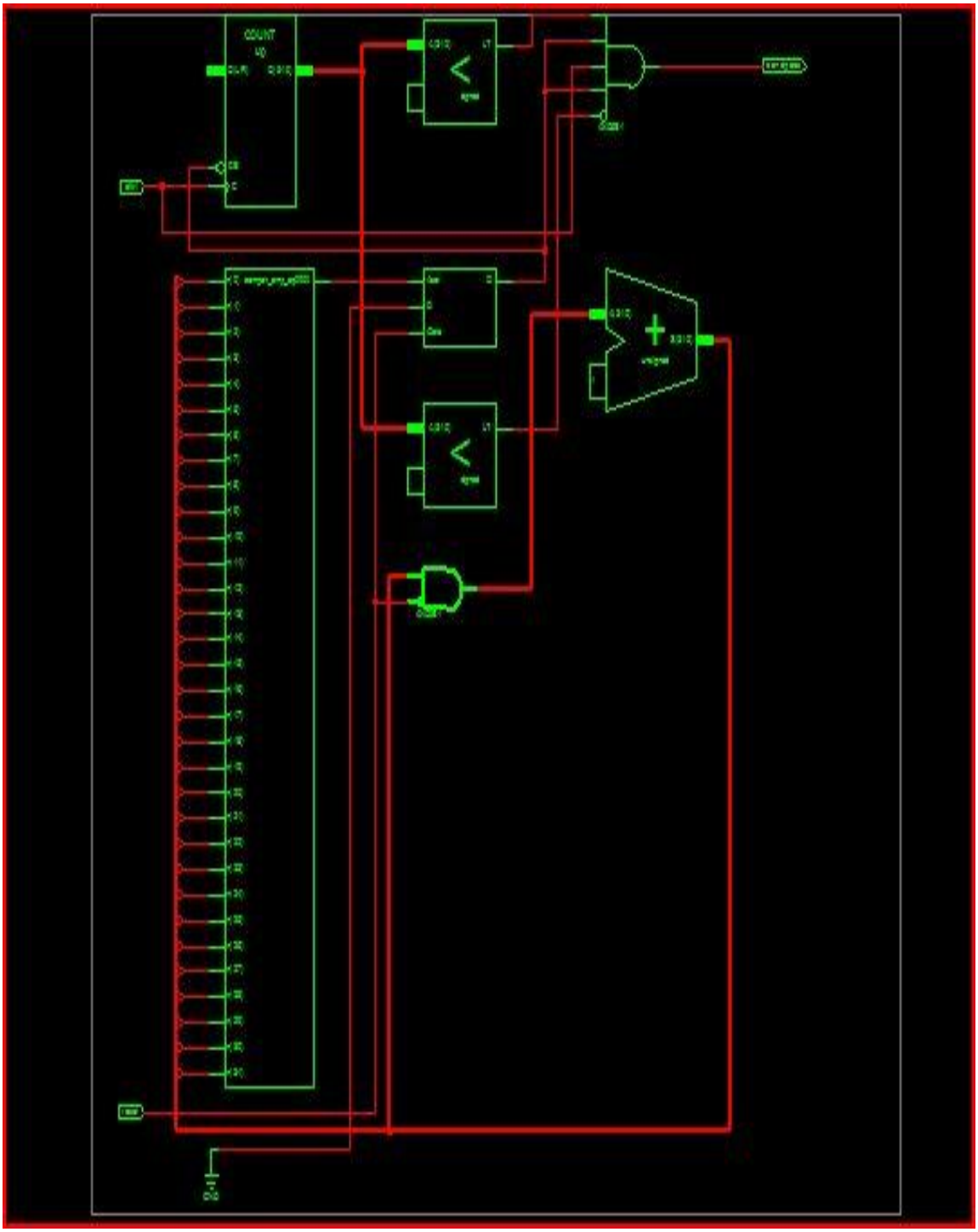


Fig. 6.22. Circuit Diagram of Band Pass Filter (Part 1)

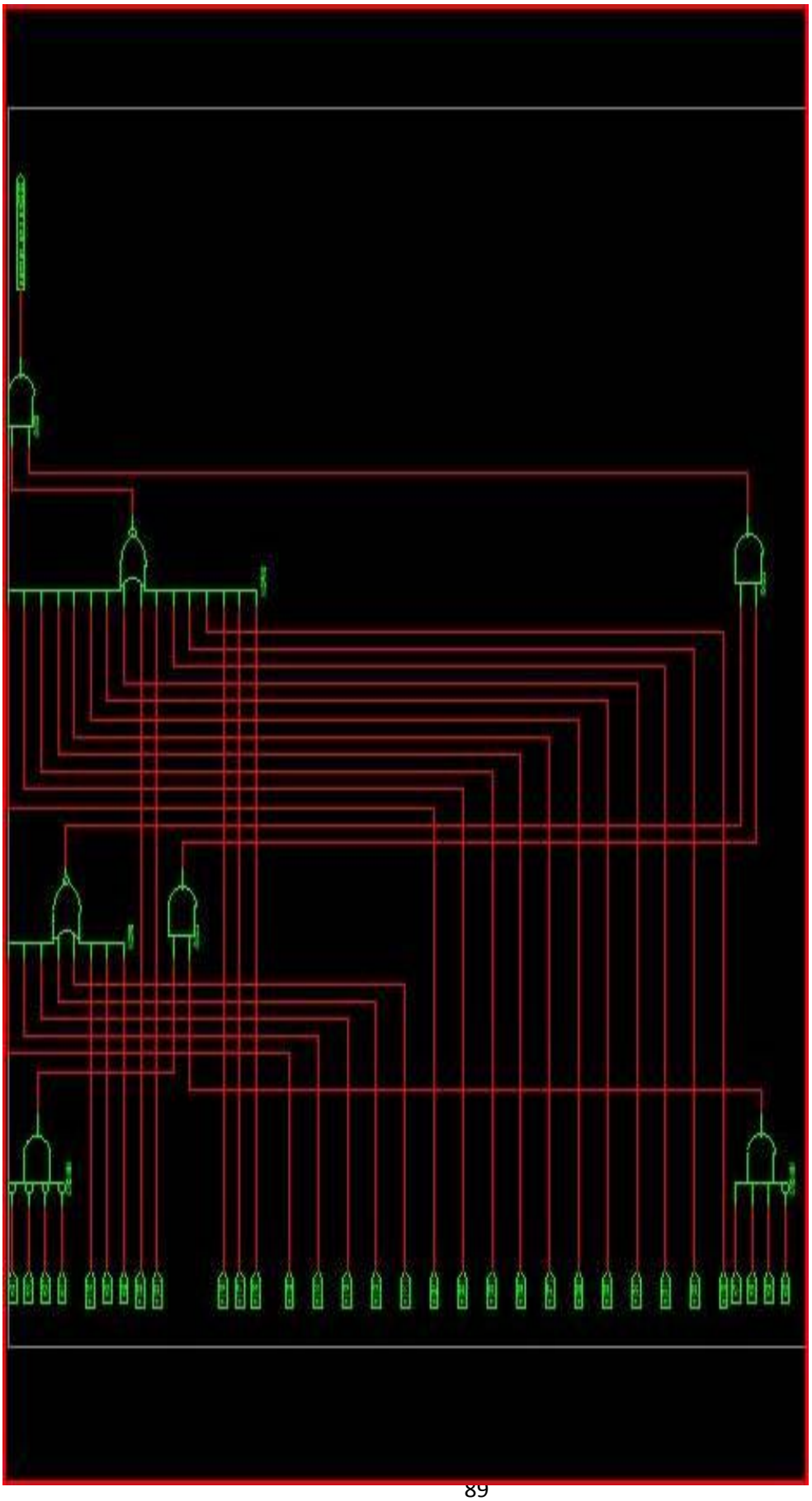


Fig. 6.23. Circuit Diagram of Band Pass Filter (Part 2)

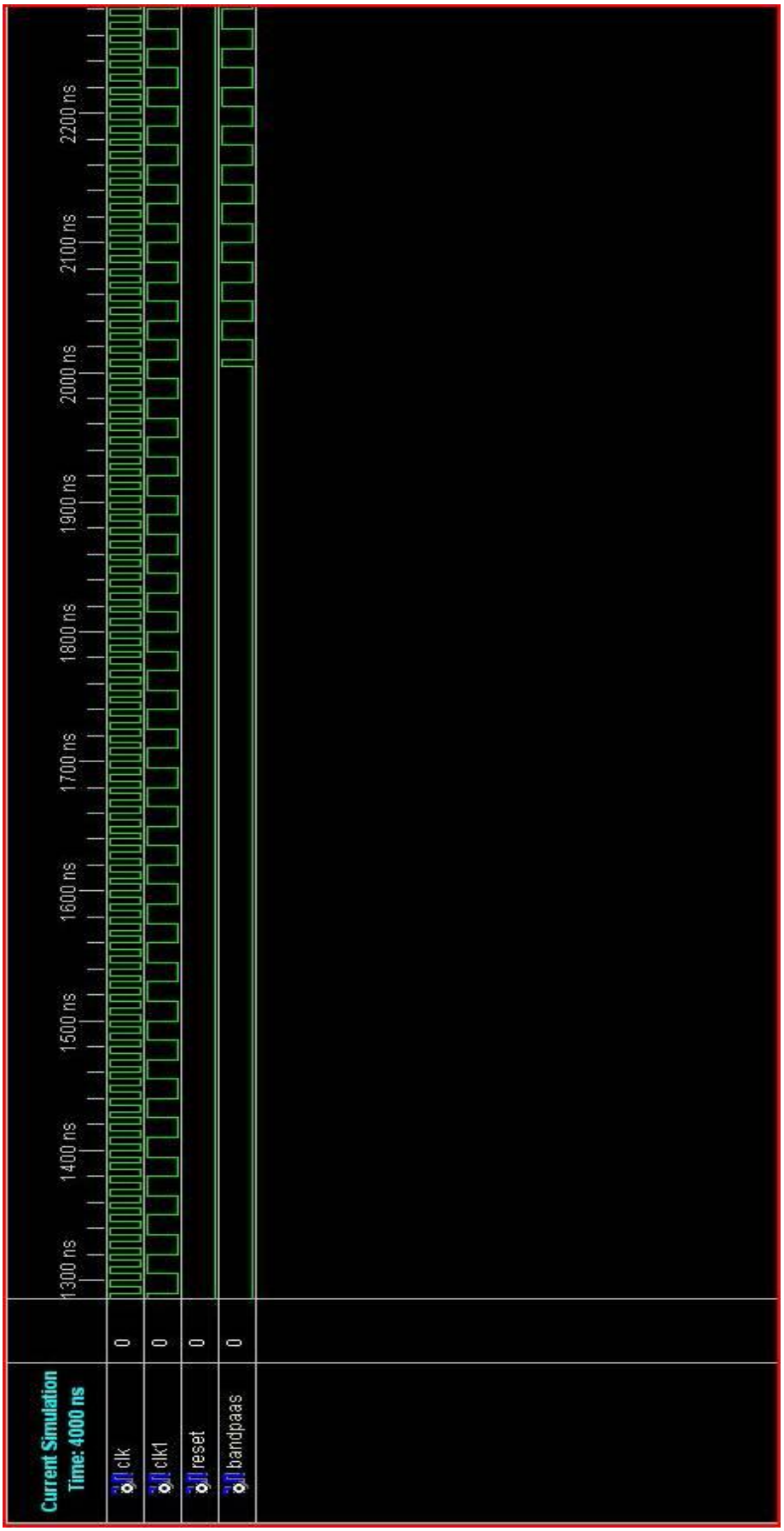
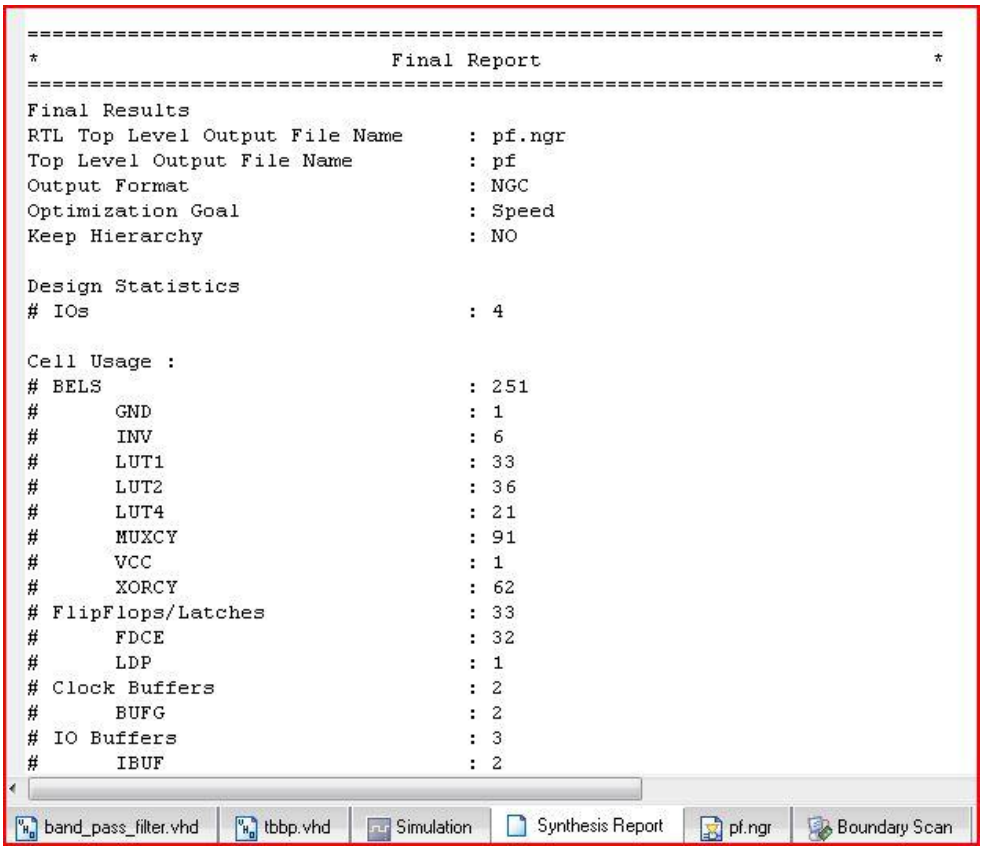


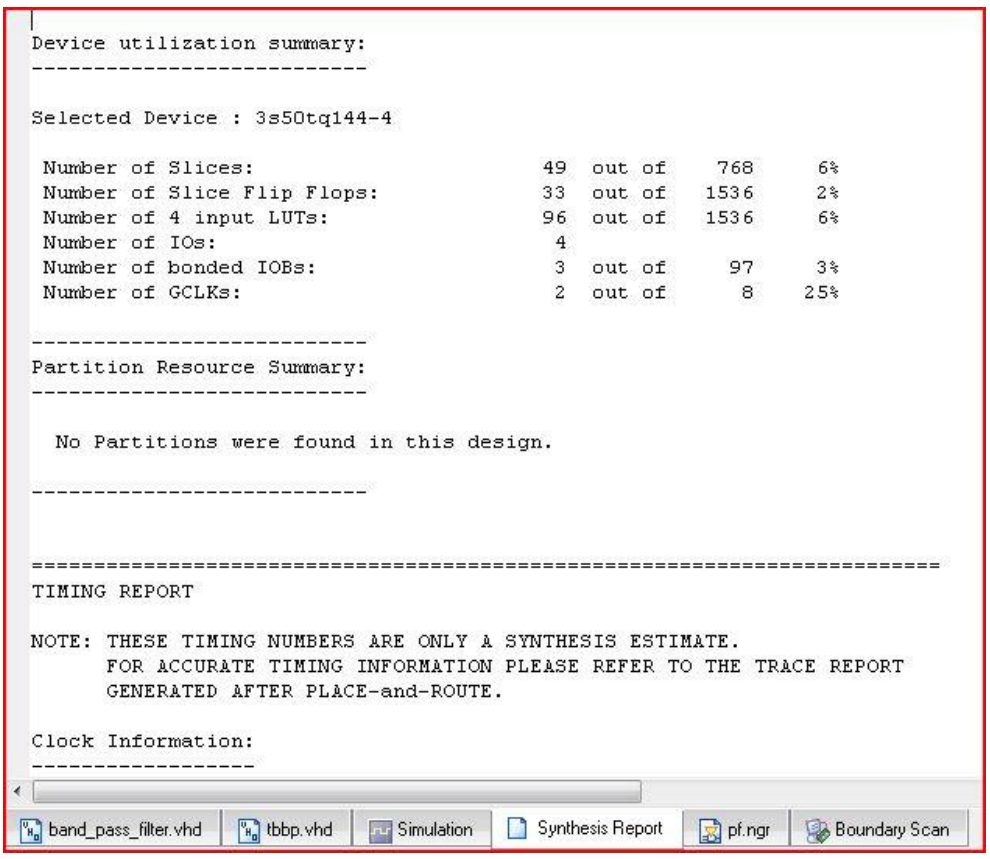
Fig. 6.24. Output Waveform of BPF.

Table 6.3 Synthesis Report of Band Pass Filter



```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : pf.ngr
Top Level Output File Name         : pf
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 4

Cell Usage :
# BELS                             : 251
#      GND                         : 1
#      INV                         : 6
#      LUT1                        : 33
#      LUT2                        : 36
#      LUT4                        : 21
#      MUXCY                       : 91
#      VCC                         : 1
#      XORCY                       : 62
# FlipFlops/Latches                : 33
#      FDCE                        : 32
#      LDP                         : 1
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 3
#      IBUF                        : 2
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan



```
Device utilization summary:
---------------------------

Selected Device : 3s50tq144-4

 Number of Slices:                       49  out of    768     6%
 Number of Slice Flip Flops:             33  out of   1536     2%
 Number of 4 input LUTs:                 96  out of   1536     6%
 Number of IOs:                           4
 Number of bonded IOBs:                   3  out of     97     3%
 Number of GCLKs:                         2  out of      8    25%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan

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

Loading device for application Rf_Device from file '3s50.nph' in environment C:\Xilinx\

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

Macro Statistics
# Adders/Subtractors                                   : 1
 32-bit adder                                          : 1
# Counters                                             : 1
 32-bit up counter                                     : 1
# Latches                                              : 1
 1-bit latch                                           : 1
# Comparators                                          : 2
 32-bit comparator less                                : 2

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

```
=========================================================================
*                    Design Hierarchy Analysis                          *
=========================================================================
Analyzing hierarchy for entity <pf> in library <work> (architecture <pf_arc>) with generics.
        highfreq = 100
        lowfreq = 50
        maxfreq = 200


=========================================================================
*                          HDL Analysis                                 *
=========================================================================
Analyzing generic Entity <pf> in library <work> (Architecture <pf_arc>).
        highfreq = 100
        lowfreq = 50
        maxfreq = 200
```

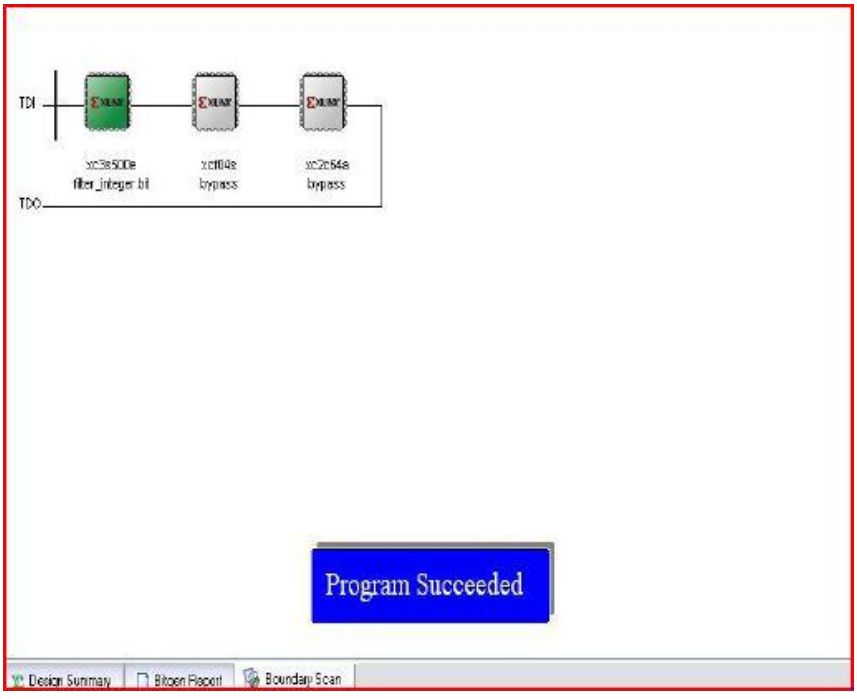


Fig. 6.25. Band Pass Filter Burn on FPGA

6.7 BAND REJECT FILTER

Following specifications are considered for lowpass filter

Band reject 50MHz < signal frequency < 100MHz block otherwise pass.

The filter order of given specification is calculated using Kaiser window:

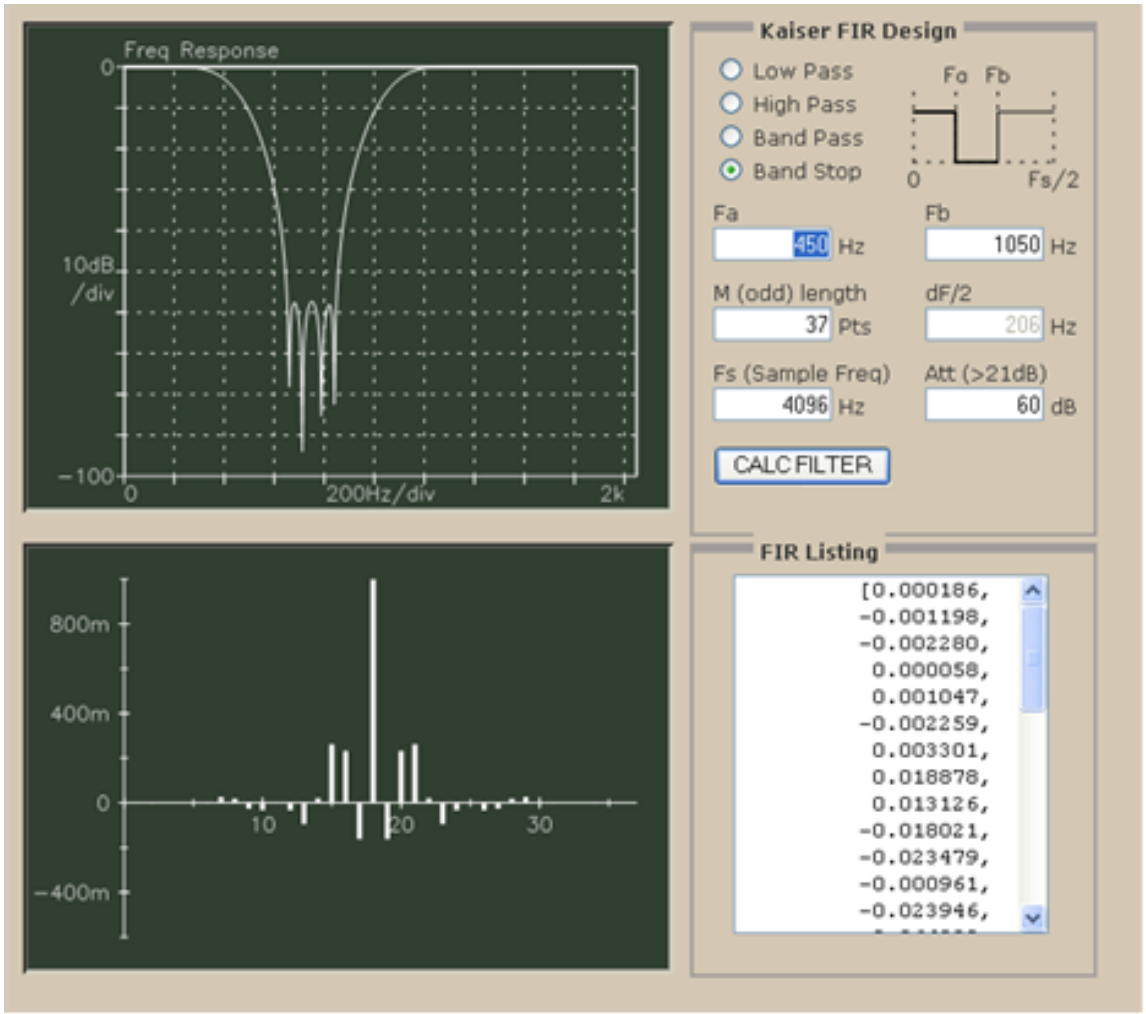


Fig. 6.26. Kaiser FIR Design for Band Reject Filter

The coefficients are directly used 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 6.27

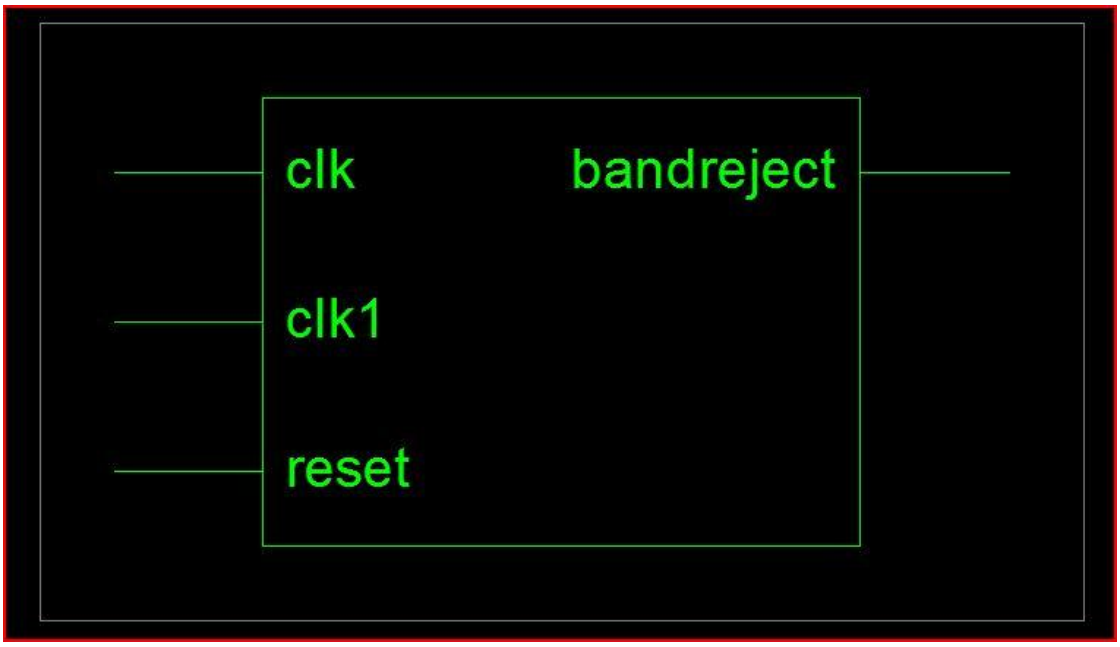


Fig. 6.27. Block Diagram of the Band Reject Filter (BRF)

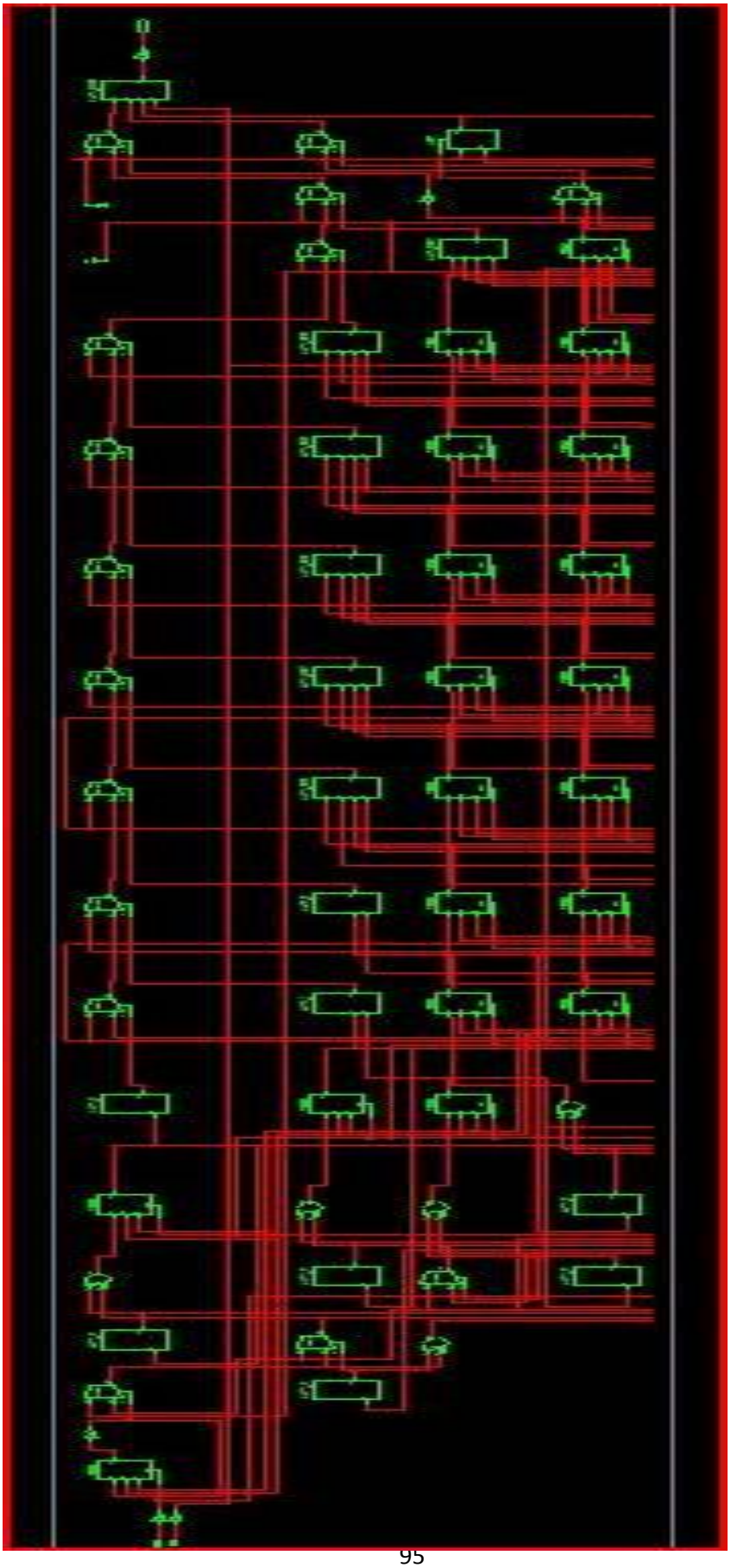


Fig. 6.28. Circuit Diagram of Band Reject Filter (Part 1).

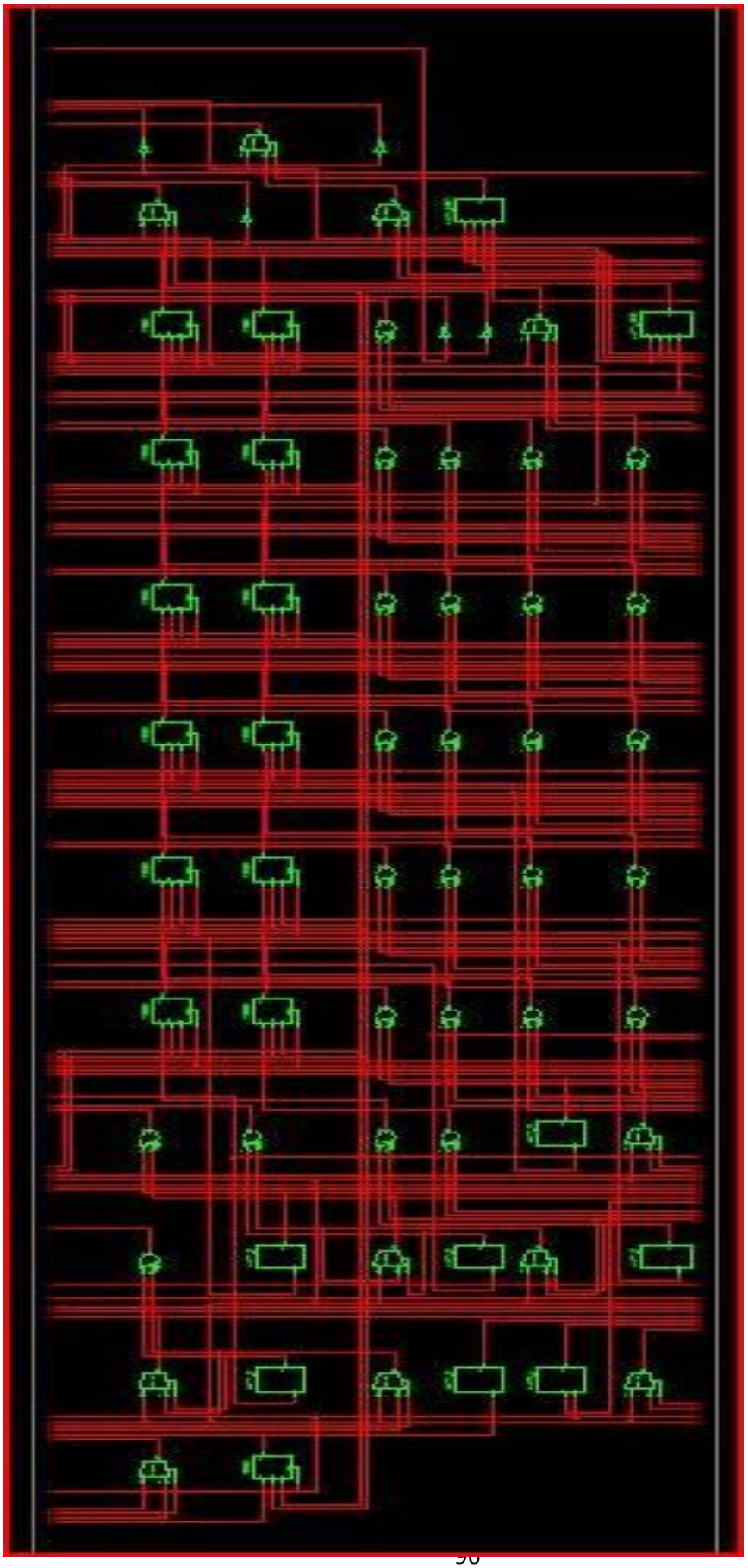


Fig. 6.29. Circuit Diagram of Band Reject Filter (Part 2).

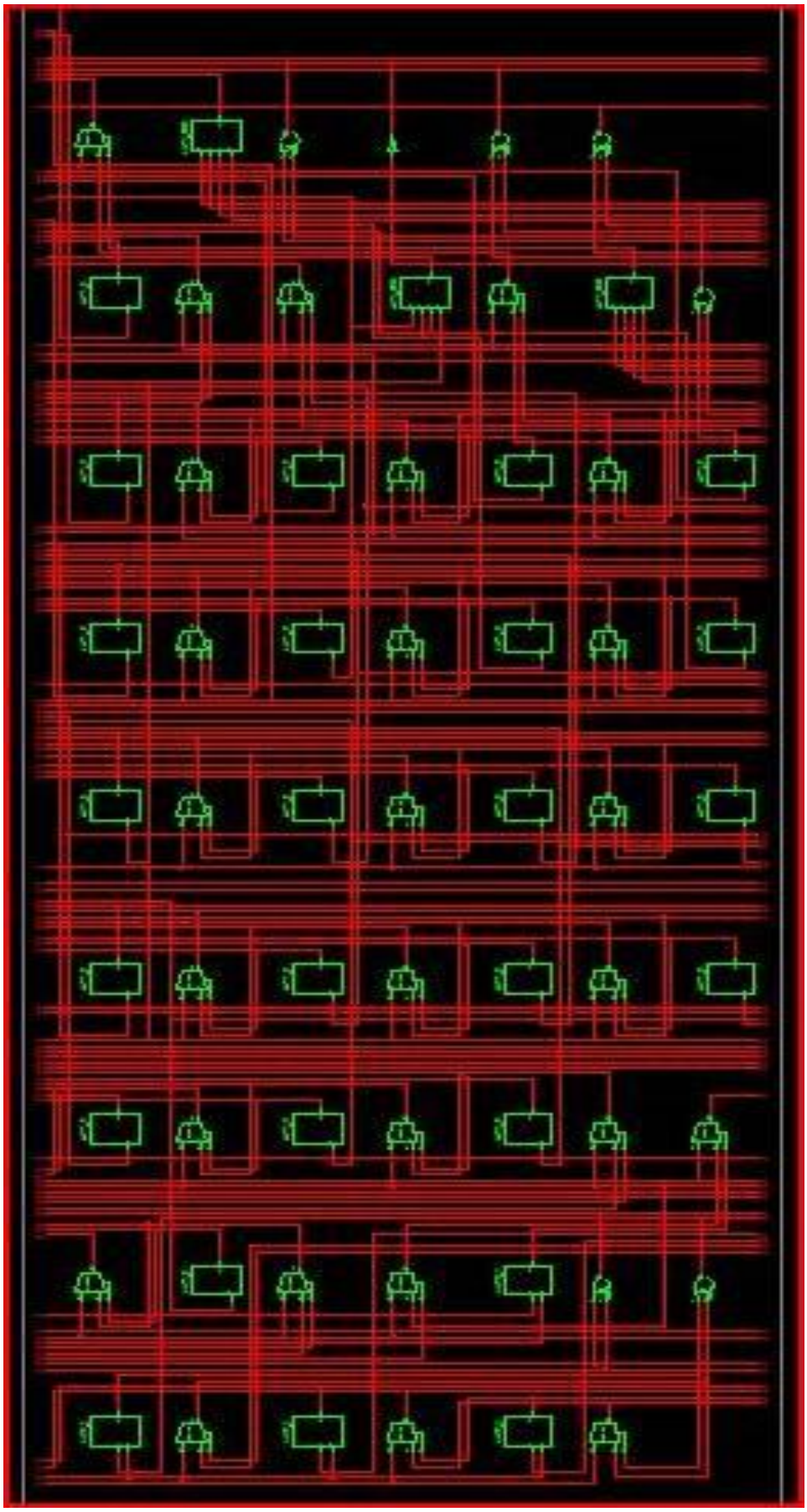


Fig. 6.30. Circuit Diagram of Band Reject Filter (Part 3).

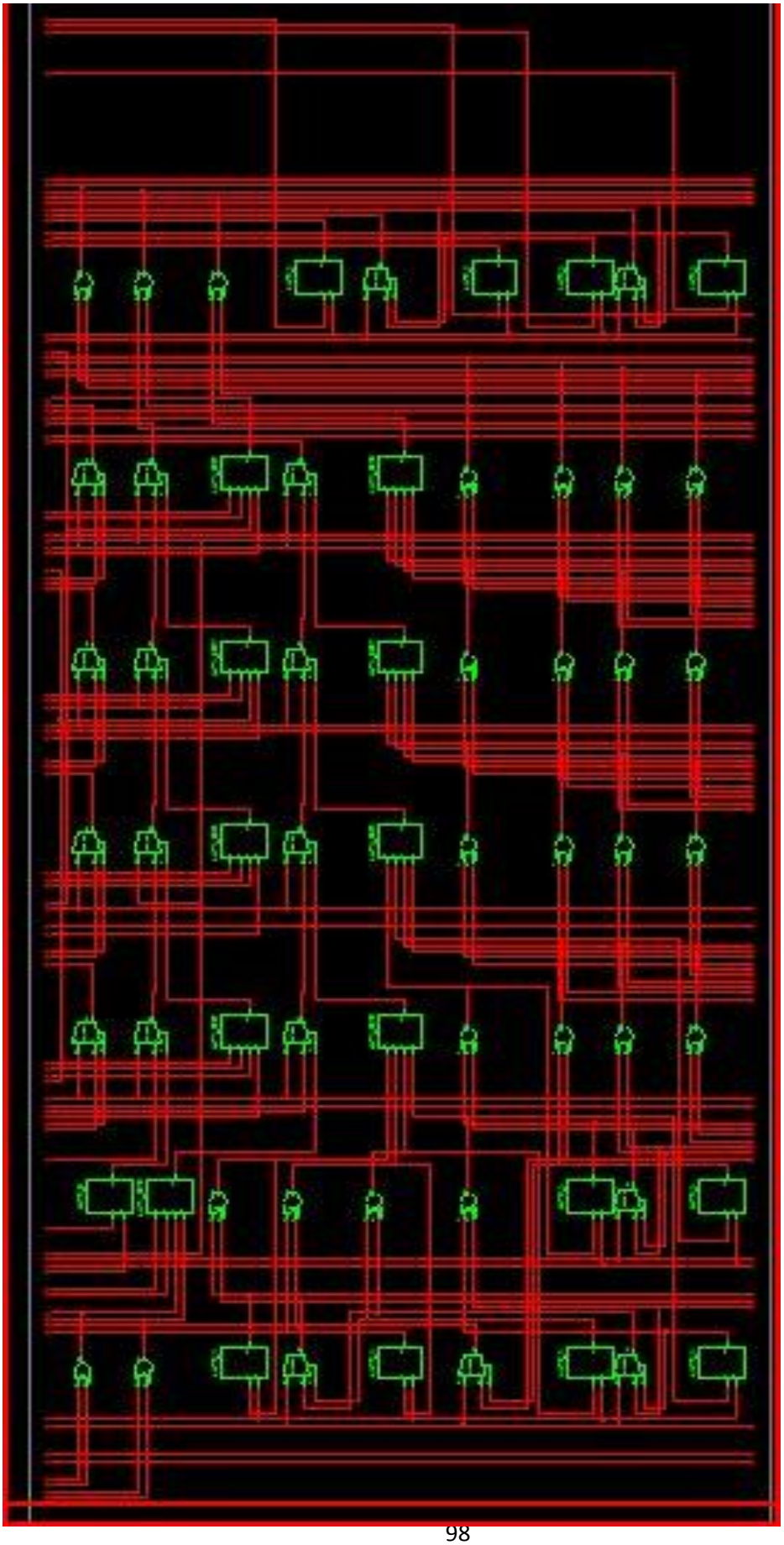


Fig. 6.31. Circuit Diagram of Band Reject Filter (Part 4).

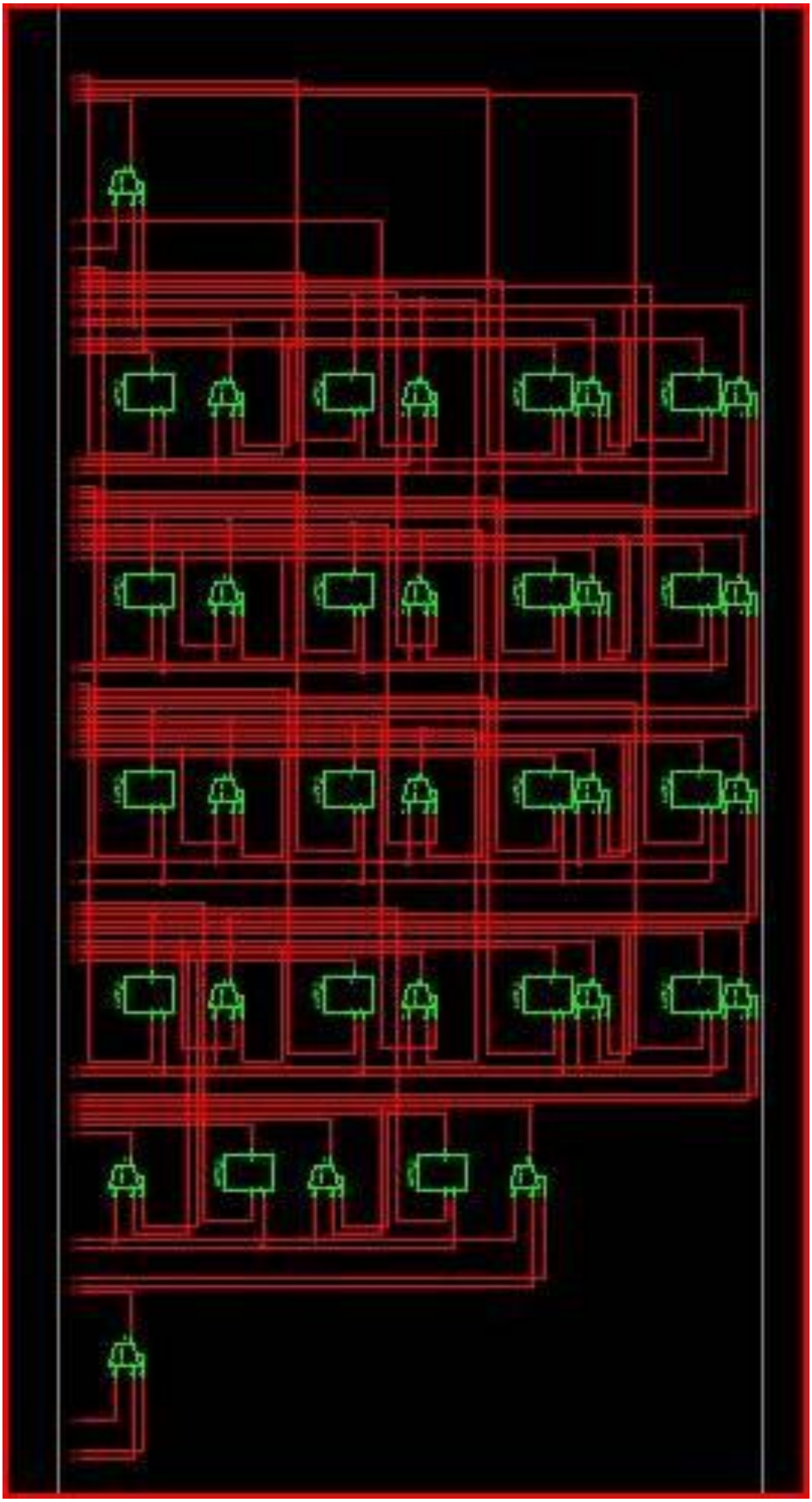


Fig. 6.32. Circuit Diagram of Band Reject Filter (Part 5).

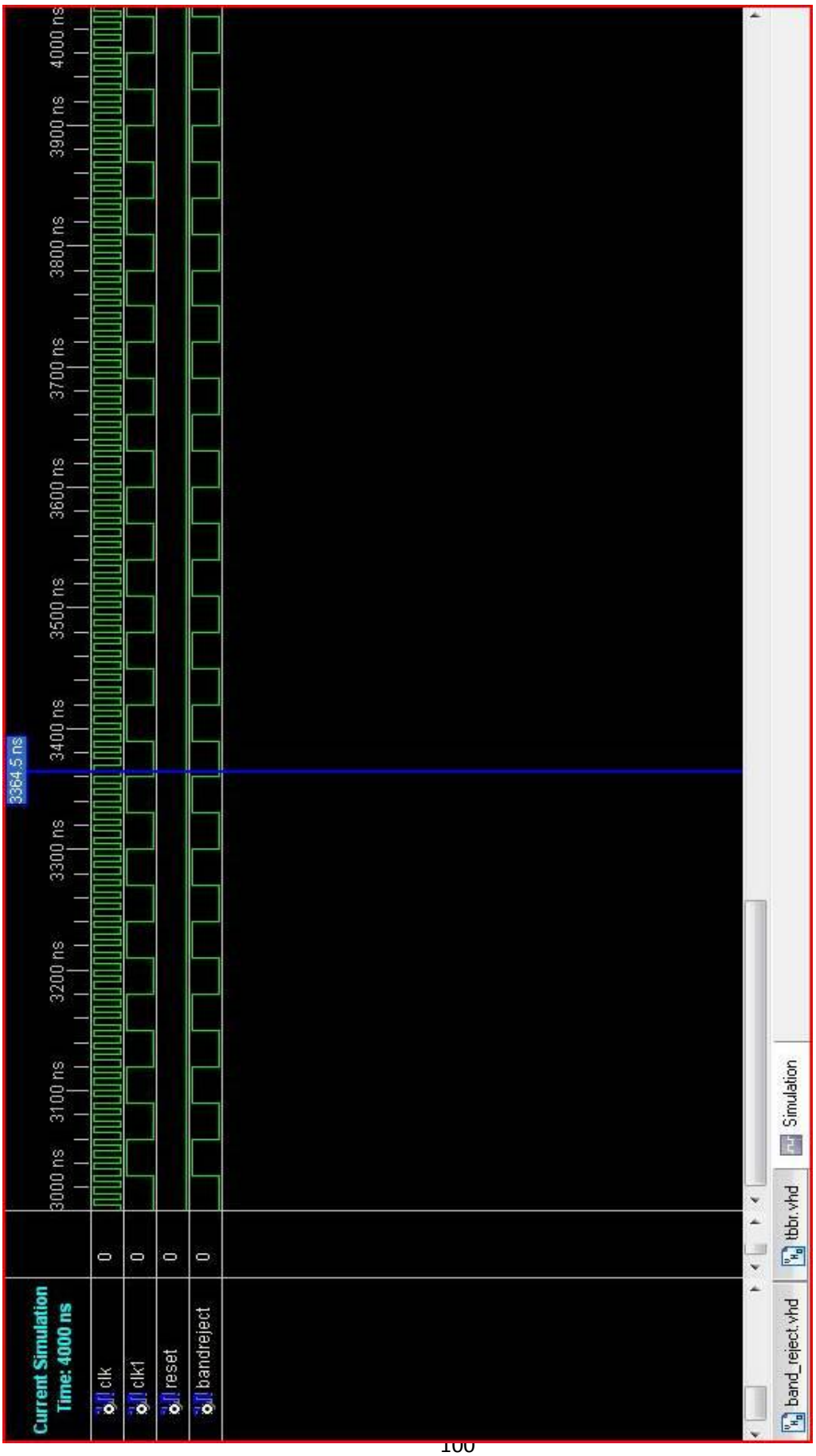


Fig. 6.33. Input Waveform of BRF.

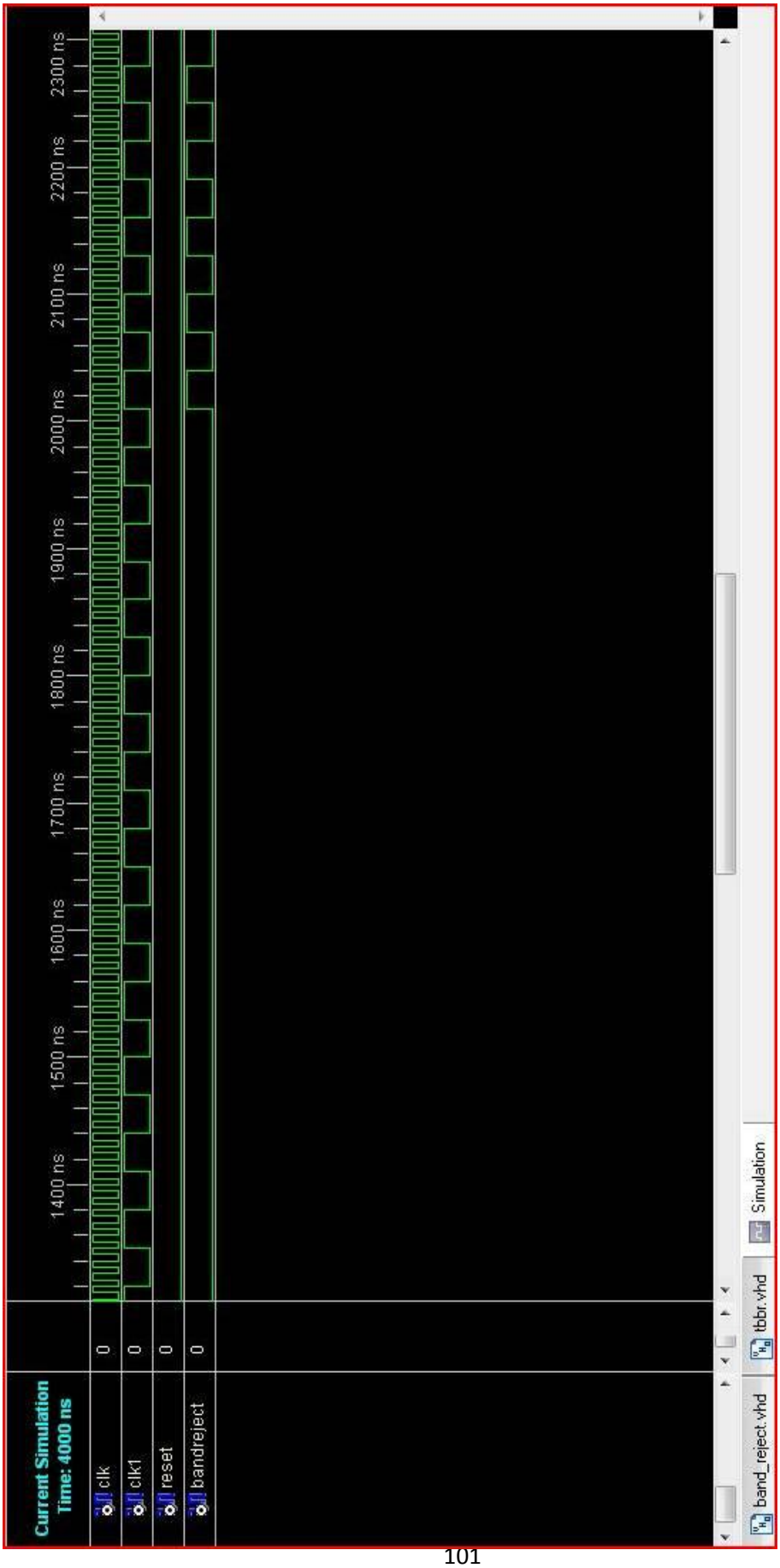


Fig. 6.34. Output Waveform of BRF.

Table 6.4 Synthesis Report of Band Reject Filter

```
=========================================================================
*                            Final Report                               *
=========================================================================
Final Results
RTL Top Level Output File Name     : pf.ngr
Top Level Output File Name         : pf
Output Format                      : NGC
Optimization Goal                  : Speed
Keep Hierarchy                     : NO

Design Statistics
# IOs                              : 4

Cell Usage :
# BELS                             : 251
#      GND                         : 1
#      INV                         : 6
#      LUT1                        : 33
#      LUT2                        : 36
#      LUT4                        : 21
#      MUXCY                       : 91
#      VCC                         : 1
#      XORCY                       : 62
# FlipFlops/Latches                : 33
#      FDCE                        : 32
#      LDP                         : 1
# Clock Buffers                    : 2
#      BUFG                        : 2
# IO Buffers                       : 3
#      IBUF                        : 2
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan

```
=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for entity <pf> in library <work> (architecture <pf_arc>) with generics.
        highfreq = 100
        lowfreq = 50
        maxfreq = 200


=========================================================================
*                             HDL Analysis                              *
=========================================================================
Analyzing generic Entity <pf> in library <work> (Architecture <pf_arc>).
        highfreq = 100
        lowfreq = 50
        maxfreq = 200
```

```
=========================================================================
*                     Design Hierarchy Analysis                         *
=========================================================================
Analyzing hierarchy for entity <pf> in library <work> (architecture <pf_arc>) with generics.
        highfreq = 100
        lowfreq = 50
        maxfreq = 200


=========================================================================
*                             HDL Analysis                              *
=========================================================================
Analyzing generic Entity <pf> in library <work> (Architecture <pf_arc>).
        highfreq = 100
        lowfreq = 50
        maxfreq = 200
```

```
Device utilization summary:
---------------------------

Selected Device : 3s50tq144-4

 Number of Slices:                       49  out of    768     6%
 Number of Slice Flip Flops:             33  out of   1536     2%
 Number of 4 input LUTs:                 96  out of   1536     6%
 Number of IOs:                           4
 Number of bonded IOBs:                   3  out of     97     3%
 Number of GCLKs:                         2  out of      8    25%

---------------------------
Partition Resource Summary:
---------------------------

  No Partitions were found in this design.

---------------------------


=========================================================================
TIMING REPORT

NOTE: THESE TIMING NUMBERS ARE ONLY A SYNTHESIS ESTIMATE.
      FOR ACCURATE TIMING INFORMATION PLEASE REFER TO THE TRACE REPORT
      GENERATED AFTER PLACE-and-ROUTE.

Clock Information:
------------------
```

band_pass_filter.vhd | tbbp.vhd | Simulation | Synthesis Report | pf.ngr | Boundary Scan

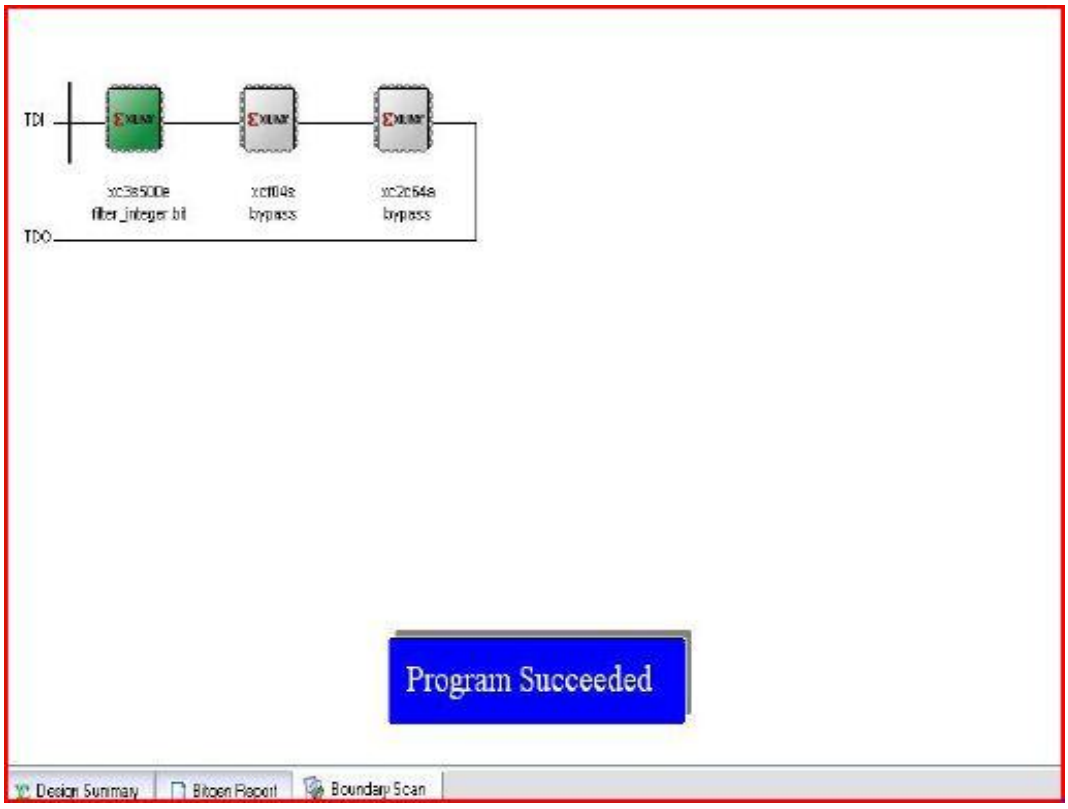


Fig. 6.35. Band Reject Filter Burn on FPGA

6.8 DISCUSSION

In this thesis, digital FIR filter has been designed by using Kaiser windows. This technique is simple conceptually and computationally. 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 α, has been selected so as to optimise the mainlobe width. Design examples have also shown that the design complexity of the Kaiser window approach is much less than that in nonlinear optimizations.

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

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 low-pass, bandpass, bandreject and highpass FIR filter are implemented in Spartan-III-xc3s200c-4tq144 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 M+1 multipliers are used to realize the N order lowpass , high pass, bandpass and bandreject filter. The designed filters can work for real time processing of any digital signal.

In this thesis, digital FIR filter has been designed by using Kaiser windows. The design of the filter is formulated as a problem of optimizing $I_0(x)$ and the adjustable parameter α , has been selected so as to optimise the mainlobe width. We can achieve direct control over the stopband attenuation thereby sustaining optimality and flexibility.

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] W. Yunlong, W. Shihu and J. Rendong "*An Extreme Simple Method for Digital FIR Filter Design*" 2011 Third International Conference on Measuring Technology and Mechatronics Automation, IEEE 2011.

[2] V. Soni, P. Shukla and M. Kumar "*Application of Exponential Window to Design a Digital Nonrecursive FIR Filter*", ICACT, 2011

[3] Sanjay Sharma, A Book on "*Digital Signal Processing*", S K Kataria & Sons, New Delhi, 2011.

[4] Z. Yu,, M.L. Yu, K. Azade and A. N. Willson "*A Low Power FIR Filter Design Technique Using Dynamic Reduced Signal Representation*", Proceedings of IEEE, 2010.

[5] G. Gaizhi, Z. Pengju, Y. Zongzuo and W. Hailong, "*Design and Implementation of FIR Digital Wave Filter Based on DSP*" in proceedings of IEEE, 2010.

[6] C. Li "*Design and Realization of FIR Digital Filters Based on MATLAB*", Proceedings of IEEE, 2010.

[7] S. Kodituwakku, R. A. Kennedy and T. D. Abhayapala "*Kaiser Window based kernel for time-frequency distributions*", IEEE, 2009.

[8] L. Jieshanproc and H. Shizhen "*An Design of the 16-order FIR Digital Filter Based on FPGA*" in The 1st International Conference on Information Science and Engineering (ICISE), 2009.

[9] R. A. Barapate, A Book on "*Digital Signal Processing*", Tech-Max Publications, Pune, 2008.

[10] A. Kuncheva and G. Yanchev "*Synthesis and Implementation of DSP Algorithms in Advanced Programmable Architectures*", International Scientific Conference Computer Science, 2008.

[11] Y. P. Lin and P. P. Vaidyanathan "*A Kaiser Window Approach for the Design of Prototype Filters of Cosine modulated Filterbanks*", IEEE Signal Processing Letters, 1998.

[12] P. Longa and A. Miri "*Area-Efficient FIR Filter Design on FPGAs using Distributed Arithmetic*", IEEE International Symposium on Signal Processing and Information Technology, 2006

[13] Prokis J. G., Manolakis D. G., A Book on "*Digital Signal Processing*", 3rd Edition, PHI publication 2004.

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

[15] Perry D., *VHDL*, 3rd Edition, Tata McGraw Hill Publications, 2001.

[16] Bhaskar J., A Book on "*VHDL primer*", 3rd Edition, Pearson Education Asia Publications, 2000.

[17] N. H. Phuong "*FIR Filter Design: The Window Design Method*".

[18] W. S. Lu, A. Antoniou and S. Saab "*Sequential Design of FIR Digital Filters for Low-Power DSP Applications*", IEEE, 1998.

[19] Sanjit K. Mitra and James F. Kaiser, "*Handbook for Digital Signal Processing*", John Wiley & Sons, Inc, 1993.

[20] B. A. Bowen and W. R. Brown, A Book on "*System Design, Volume II of VLSI System Design for Digital Signal Processing*", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.

[21] P. P. Vaidyanathan, "*Optimal Design of Linear-Phase FIR Digital Filters with Very Flat Passbands and Equiripple Stopbands*", Proceedings of IEEE, 1985.