IMPLEMENTATION OF FIR FILTER IN FPGA

A DISSERTATION

Submitted in partial fulfillment of the requirements for the award of the degree

of

MASTER OF TECHNOLOGY

in

ELECTRICAL ENGINEERING

(With Specialization in System Engineering & Operations Research)

By

DEVARA DILIP KUMAR

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE
ROORKEE -247 667 (INDIA)
JUNE, 2006

CONTENTS

INDIAN INSTITUTE OF TECHNOLOGY
ROORKEE-247667

CANDIDATE'S DECLARATION

I hereby declare that the work, which is being presented in this dissertation entitled **"Implementation of FIR filter in FPGA"** in the partial fulfillment of the requirement for the award of the degree of **Master of Technology in Electrical Engineering** with specialization in **System Engineering and Operation Research,** submitted in the **Department of Electrical Engineering, Indian Institute of Technology Roorkee,** Roorkee, is an authentic record of my own work carried out during July 2005 to June 2006 under the supervision of **Dr. Indra Gupta,** Assistant Professor and **Prof. M. K. Vasantha,** Professor, Department of Electrical Engineering, IIT Roorkee, Roorkee.

I have not submitted the matter embodied in this dissertation for award of any other degree.

D. Dilip kumar

Date: 30/06/2006 **(Devara Dilip Kumar)**

Place: Roorkee

CERTIFICATE

This is to certify that the above statement made by the candidate is true to the best of my knowledge and belief.

[illegible]

Prof. M. K. Vasantha
Professor,
Department of Electrical Engineering,
Indian Institute of Technology Roorkee.
Roorkee – 247667.

[illegible]

Dr. Indra Gupta
Asstt. Professor,
Department of Electrical Engineering,
Indian Institute of Technology Roorkee.
Roorkee – 247667.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere gratitude to **Prof. M.K. Vasantha,** Professor, and Dr. Indra Gupta, Asstt. Professor, Department of Electrical Engineering, IIT Roorkee, for encouraging me to undertake this dissertation as well as providing me their valuable guidance and inspirational support.

I consider myself fortunate for getting the opportunity to learn and work under their able supervision. They provided me the latest technology to work on and state of the art facilities and equipments at the lab. Their able guided ensured me that I should not have any lack of resources while carrying out my thesis work. I have deep sense of admiration for their innate kindness and enthusiasm. Working under their guidance will always remain a cherished experience. I once again thank Prof. M.K. Vasantha for allowing me to attend the classes of course, "Advance Computer Control Systems" which helped me a lot in the dissertation work.

I also thank all the teachers of SEOR group who, with their encouraging and caring words, constructive criticism and suggestions, have contributed directly or indirectly in a significant way towards the completion of this report.

I am especially thankful to Mr, Solomon Raju, Research Scholar, Dept of Electronics and Computer Engg, IIT Roorkee, for his valuable guidance in working with the Virtex 2 Pro kit.

I also thank Mr. Vishal Saxena and Mr. Vijender Singh, Research Scholar, for extending their kindness in every aspect during my time spent in Lab.

I am thankful to Mr. Kalyan Singh and Mr. C.M. Joshi, Laboratory staff of Microprocessor & Computer Lab for providing me their co-operation in the time of need.

I also thank all my friends who have encouraged me and gave valuable suggestions in my thesis work.

Most of all, I thank the God Almighty for providing me His grace and help in all my work.

D. Dilip Kumar

Devara Dilip Kumar

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 signal processors (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, an Eight-order low-pass FIR filter is implemented in FPGA. Two different known approaches in the filter theory are used in this implementation. Firstly, Cascade Decomposition is considered which overcomes the coefficient-sensitivity problem prevalent in FIR Direct Structures. However, this approach requires more number of complex multiplications than the FIR Direct structures that limits the speed of operation in real-time. Secondly, Distributed Arithmetic approach in realizing a digital filter is considered. This approach gives a better performance than the common filter structures in terms of speed of operation, cost and power consumption in real-time. It replaces the uses of complex multiplications by using Adders, Shift Registers and Look-Up Tables. The FIR filter is implemented in Virtex-2-Pro FPGA and simulated with the help of Xilinx ISE (Integrated Software Environment).

List of Figures

Chapter 1: INTRODUCTION TO FIR FILTER

Digital Filter as a system can be represented in the form of a block diagram as shown in *Figure 1.1* The input *x(nT)* to the Digital Filter is the sampled input coming from the Analog-Digital converter. The output *y(nT)* which is the response of the system is again the digital data going as input to the Digital–Analog converter.

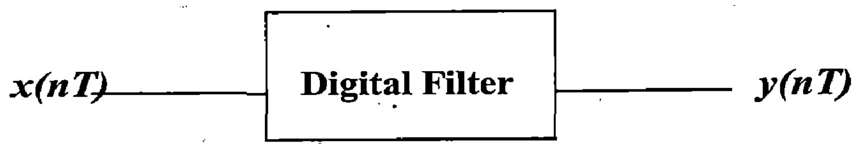


Figure 1.1: Digital Filter as a system

The output or response of such a system is related to the input by some function in accordance with the required specifications. The response can be given as

$$y(nT) = Rx(nT) \qquad ..(1.1)$$

where ***R*** is an operator performing desired operation.

Some of the important properties in analyzing any system are Time-Invariance, Causality and Linearity which are mentioned below:

A Digital Filter is said to be time-invariant if the internal parameters do not change with time, which means for a specific input or excitation the response will be the same independent of the time of application of the input.
A Digital Filter is said to be Causal if its response at a specific instant is independent of subsequent values of the excitation, which means the response is dependent on the current and past values of the input samples.
A Digital Filter is said to be Linear if it possesses the property of Superposition which says that if an input consists of weighted sum of several signals, then the

output is the superposition or the weighted sum of the responses of the system to each of the signals.

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

$$y(nT) = f(\ldots, x(nT-T), x(nT), x(nT+T), \ldots) \qquad ..(1.2)$$

Assuming Linearity and Time-invariance *y(nT)* can be expressed as

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

where 'a_i's represents constants.

Now assuming causality for the filter we have

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

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

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

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

In contrast to this filter, the response of a causal recursive filter is a function of elements consisting of past, present inputs and past outputs. It is expressed in Nth-order linear difference equation as

$$y(nT)=\sum_{i=0}^{N}a_i x(nT-iT)-\sum_{i=1}^{N}b_i y(nT-iT) \qquad ..(1.5)$$

When a unit impulse defined as

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

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

$$h(nT)=\sum_{i=0}^{N}a_i\delta(nT-iT) \qquad ..(1.6)$$

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

$$h(0)=a_0\ ,\ h(T)=a_1\ \ldots\ldots\ldots\ldots\ h(nT)=a_n \qquad ..(1.7)$$

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

In general, one can view Equation (1.4) as a computational procedure (an algorithm) to determine the output sequence *y(nT)* from the input sequence *x(nT)*. Also, in various ways, the computations in Equation (1.4) can be arranged into equivalent sets of difference equations. Normally such a kind of re-arrangement of the basic difference equation is done, so as 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, Digital Signal Processing algorithms (DSP) are implemented either using general purpose DSP processors 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.

1.1 Scope of the Report

The report is mainly concentrated on implementing a Low-pass FIR filter. An Eight-order filter is taken as an example for the implementation. There are many realizations of FIR filters. Two such realizations, Cascade decomposition and Distributed Arithmetic approach are simulated in the thesis. The first part of the report describes the theory behind the FIR filters. In the first part, a brief explanation of few of the filter structures and the design of the filter is discussed. The second part mainly concentrates on implementing filter in FPGA. The data storage and finite-word length effect is also described in this section.

1.2 Organization of the Thesis

Chapter 2 discusses the some of the filter structures like Direct structures, cascaded and parallel decomposition and an alternative approach Distributed Arithmetic approach for the implementation of the filter in FPGA.

Chapter 3 discusses, which filter structure to choose, the data storage of input samples and the coefficients, how the truncation operation is done. Two filter structures, cascade decomposition and using Distributed Arithmetic approach are considered in the thesis, and how they are implemented in FPGA is discussed.

Chapter 4 shows the simulation results
Chapter 5 discusses about Virtex 2 Pro FPGA and the kit used.

Chapter 2: FIR Filter Structures and Design

Introduction

The analysis of linear, time-invariant digital filter is generally carried out by using the *Z*-transforms. A brief review of the *Z*-transform is presented. The filter structures characterizing the difference equations are represented using basic elements such as multipliers, time-delays and adders. The characteristics of an ideal digital filter and the design using windowing techniques are given. Finally, the four different cases where an FIR filter presents linear-phase is included in this chapter.

2.1 Z Transform

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

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

$$F(Z) = \sum_{n=-\infty}^{\infty} f(nT)\, z^{-n} \qquad \text{.... Equation (2.1)}$$

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

Some important properties of the Z-transform such as Linearity, Translation and Convolution are given as below:

If *a* and *b* are arbitrary constants and *f(nT)* and *g(nT)* are arbitrary functions such that

$$Z\, f(nT) = F(z) \quad \text{and} \quad Z\, g(nT) = G(z) \text{ then for}$$

Linearity:

$$Z\left[af(nT)+bg(nT)\right]=aF(z)+bG(z)$$ ..... Equation (2.2

Translation:

$$Zf\left(nT+mT\right)=z^{m}F(z)$$ ....Equation (2.3)

Convolution:

$$Z\sum_{k=-\infty}^{\infty}f(kT)g(nT-kT)=F(z)G(z)$$ ... Equation (2.4)

The above properties are useful in deriving the transfer function of the filter. Now, evaluating the Z-transform on Equation (1.4) we obtain,

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

By using the time translation property and the convolution property of Z-transform, Equation (1.4) can be re-arranged as

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

Or, $Y(z)=H(z).X(z)$ where ...Equation (2.5)

$$H(z)=\sum_{i=0}^{N}a_{i}z^{-i}$$ ...Equation (2.6)

where *H (z), X (z), Y (z)* are the Z-transforms of Impulse Response, Input samples and Output samples.

H(z) is called the transfer function of the filter and the time-domain samples of this transfer function, which are the filter coefficients are approximated according to the desired response.

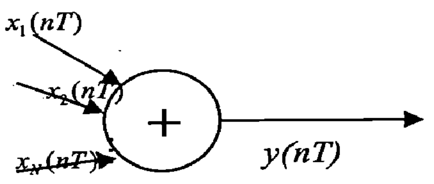
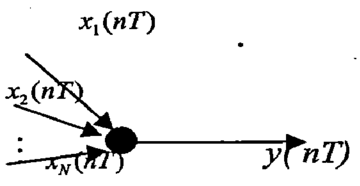
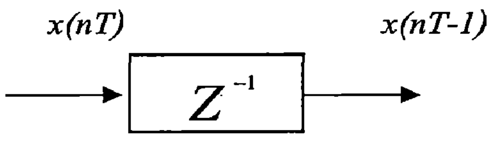
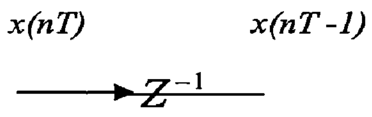
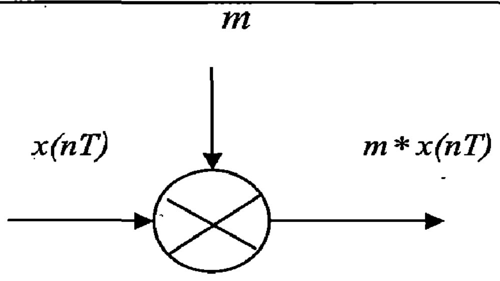
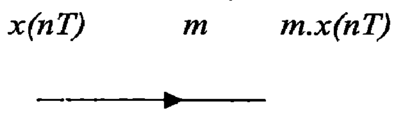
BASIC ELEMENTS	BLOCK REPRESESENTATION	SIGNAL FLOW DIAGRAM
ADDER	 $y(nT) = \sum_{k=1}^{N} x_k(nT)$	
TIME-DELAY		
MULTIPLIER		

Figure (2.1): Basic Block elements

2.2 Filter Structures

The computational algorithm implementing Equation (1.4) of an FIR filter can be conveniently represented in block diagram. It is done using the basic building blocks elements such as Multipliers, Adders and Unit Delays. These basic block elements and their equivalent Signal Flow Diagrams are as shown in **Figure (2.1).**

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

2.2.1 Direct Structures

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

$$H(z)=\frac{Y(z)}{X(z)}=\sum_{i=0}^{n}a_i z^{-i} \qquad \text{...Equation (2.7)}$$

There are four Direct structures which are different realizations of Equation (2.7). The first Direct structure only is presented here and is as shown in Figure (2.2)

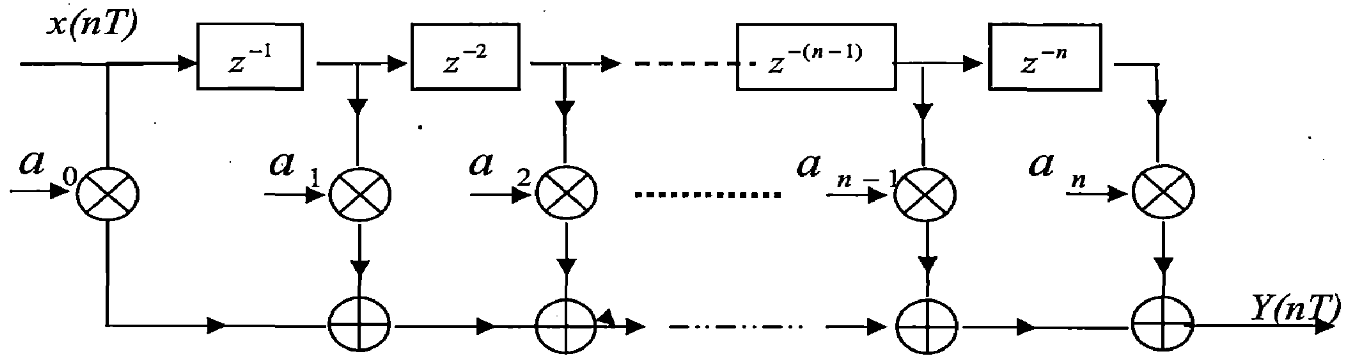


Figure 2.2: Direct Structures

The 1-D structure is also called canonical because it possesses n-time delay elements. The signal flow diagram of this structure is as shown below in *Figure (2.3).*

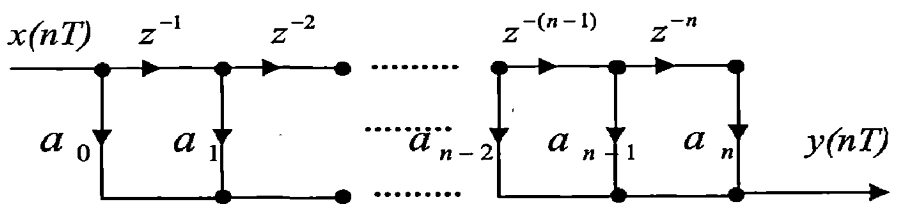


Figure 2.3: Signal Flow Diagram of Direct Structures

As seen from the Signal Flow Diagram the above representation requires "*n*" Delay elements, "*n + 1*" multipliers and "*n*" adders to implement in the digital computer.

The above structure suffers extreme coefficient sensitivity as the value of grows large. That is a small change in a coefficient for large value of *n* causes large changes in the zeroes of *H(z)*.

2.2.2 Cascade Decomposition:

A Cascade realization is done by cascading Second order modules. A second-order module is given as:

$$H_0(z)=a_{01}+a_{01}z^{-1}+a_{01}z^{-2}$$

For realization of this structure Equation (2.7) is factorized and the obtained factorized terms which are nothing but the second order modules are cascaded, such decomposition is called the cascade decomposition.

Representing Equation (2.7) in terms of Second-order modules an *n*-order filter can represented as:

$$H(z)=\prod_{i=1}^{m}\left(a_{i0}+a_{i1}z^{-1}+a_{i2}z^{-2}\right)=\prod_{i=1}^{m}\left(A_i(z)\right) \quad \text{... Equation (2.8)}$$

Here, *m* is the least integer greater than or equal to *n*/ 2.

The cascaded block structure, block representation using basic elements and the signal-flow diagram is as shown in *Figure (2.4).*

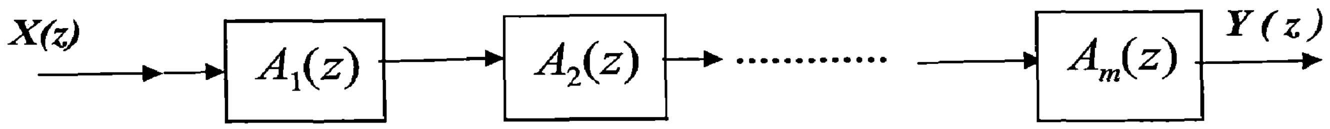


Figure (2.4.1): Block Representation of Cascade Decomposition

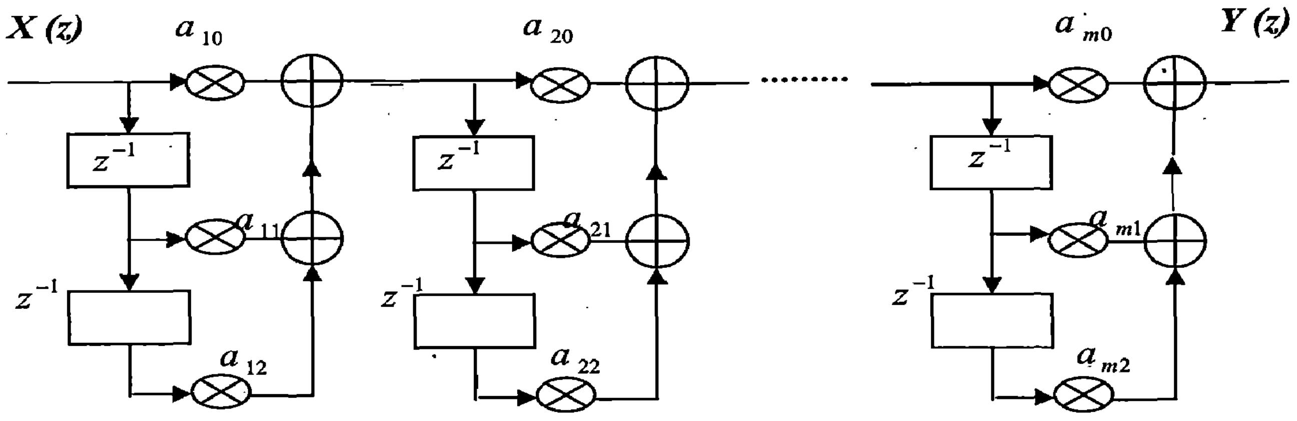


Figure (2.4.2): Block Representation of Cascade Decomposition

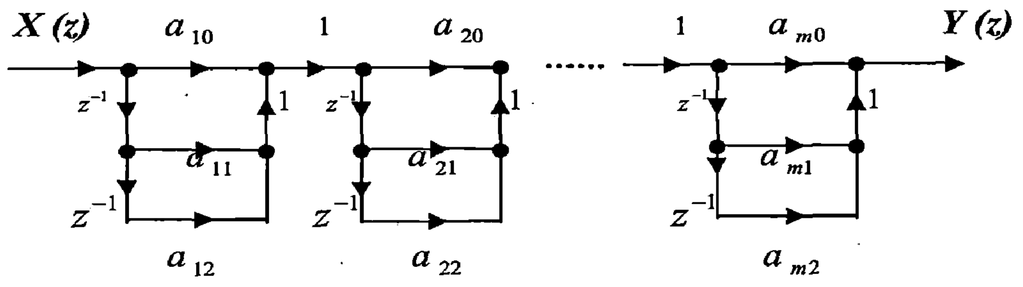


Figure (2.4.3) Signal Flow Representation of Cascade Decomposition

As seen from the signal flow diagram, an n-order system requires *3m* multipliers, *2m* delay elements, and *2m* adders to implement in the digital system. We shall see such implementation in the next chapter by considering an Eighth-order system and there by writing an algorithm or a computational procedure for implementing it in FPGA.

2.2.3 Parallel Decomposition

In this method the transfer function for an m-order digital filter is represented of the form:

$$H(z)=\beta_0+\sum_{i=0}^{m}B_i(z) \qquad \text{... Equation (2.9)}$$

Where $B_i(z)=\beta_{i1}z^{-1}+\beta_{i2}z^{-2}$

Such representation has an advantage of avoiding coefficient – sensitivity problems. The parallel block structure, block representation using basic elements and the signal-flow diagram is as shown in *Figure (2.5).*

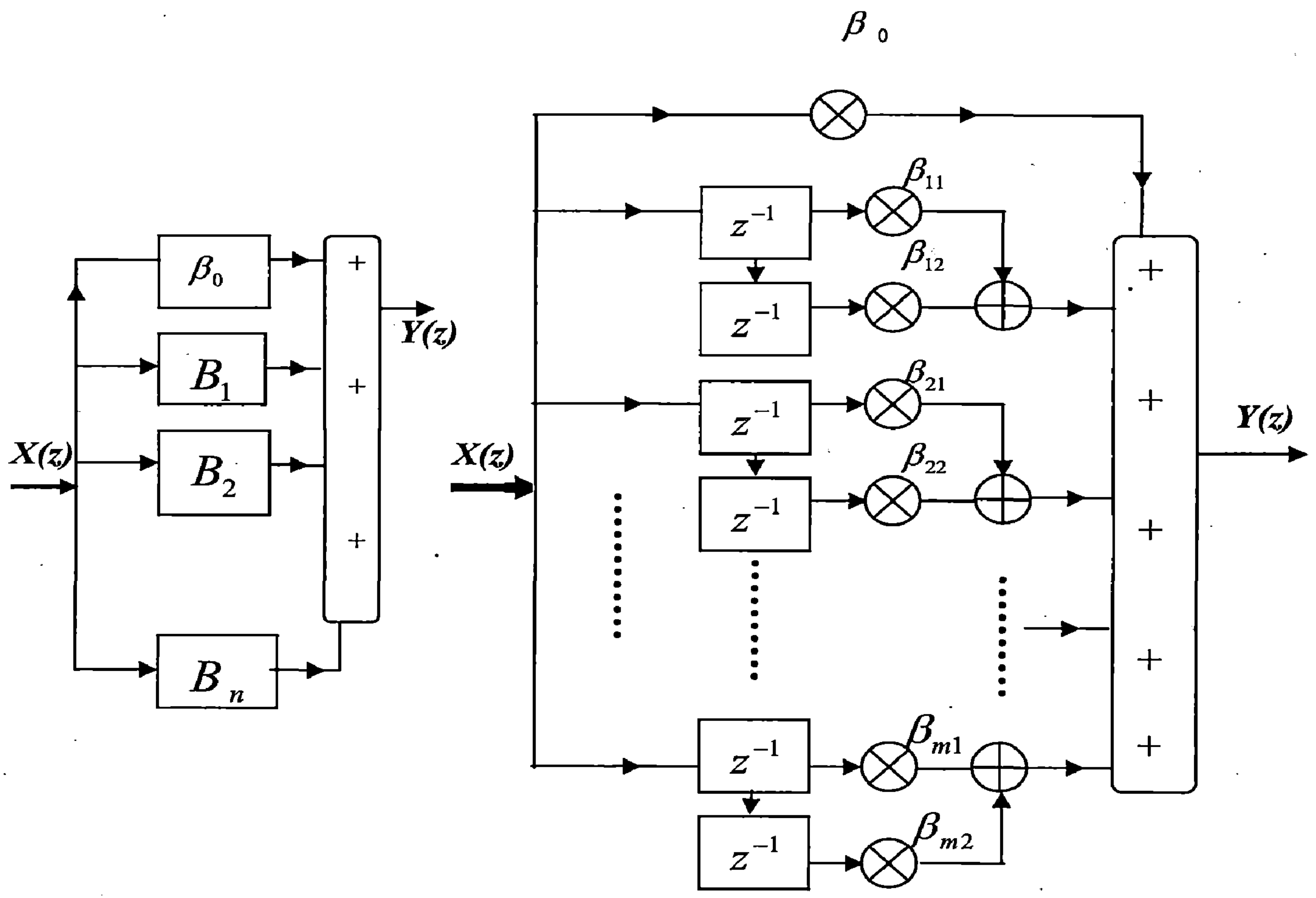


Figure 2.5.1 Block Representation of Parallel Decomposition

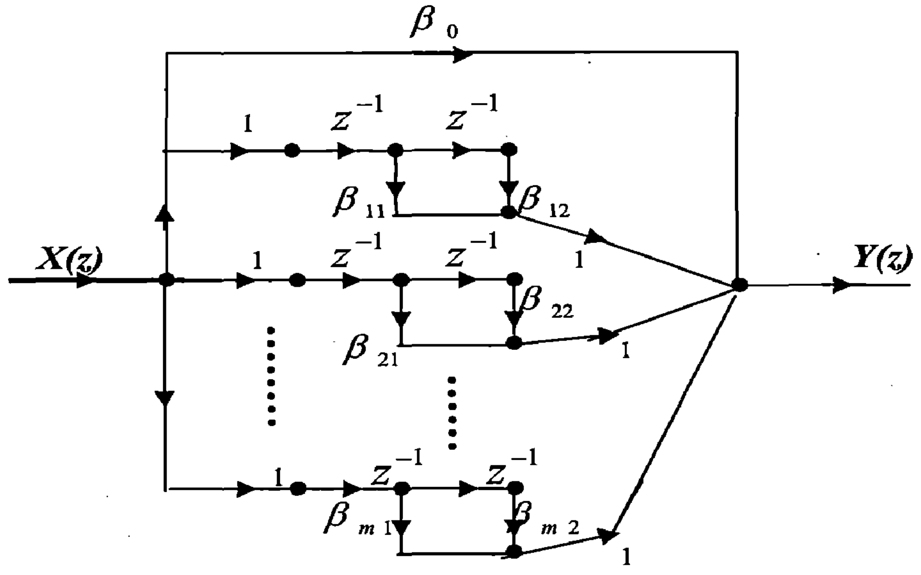


Figure 2.5.2: Signal Flow Diagram for parallel Decomposition

As seen from the Signal flow diagram for an *n*-order filter, *2m* adders, *2m* time-delays and *(2m + 1)* multipliers are needed to implement in digital computer. Here *m* is the least integer greater than *n/2,* where *n i*s the order of the filter.

2.2.4 Distributed Arithmetic Approach

In the above filter structures, multipliers are used. As the multiplier operation takes more time to compute as compared to adders and time-shifters, more number of multiplications often increases the complexity and affects the performance of the system.

The basic idea in Distributed Arithmetic approach is to remove the multipliers, i.e., to perform the summation of the products between filter coefficients and internal signals without using multipliers.

Rewriting Equation (1.4), we have

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

Let $x(nT)=x(k)$ *and* $y(nT)=y(k)$ *then*

$$x(nT-T)=x(k-1),\ x(nT-2T)=x(k-2)......x(nT-NT)=x(k-N) \quad \text{...Equation (2.10)}$$

Thus on substituting above equation, Equation (1.4) can be written as

$$y(k) = \sum_{i=0}^{N} a_i x(k-i) \qquad \text{... Equation (2.11)}$$

Assuming that the Input samples are represented in 2's complement format with *(b+1)* bits, also if the input samples are properly scaled such that their magnitude is less than 1.

Then the input sample *x(k)* is given as,

$$x(k) = -x(k)_0 + \sum_{j=1}^{b} x(k)_j 2^{-j} \qquad \text{... Equation (2.12)}$$

where $x(k)_j$ is the j^{th} bit of *x (k)* and $x(k)_0$ is the most-significant bit of *x(k)*

Substituting Equation (2.12) in Equation (2.11) we obtain,

$$y(k) = -\sum_{i=0}^{N} a_i x(k)_0 + \sum_{j=1}^{b} 2^{-j} \sum_{i=0}^{N} a_i x(k)_j \qquad \text{... Equation (2.13)}$$

Now defining a binary function F_i such that

$$F_i = \sum_{i=0}^{N} a_i x(k)_j \ \ where\, j{=}0\ for\ \ i{=}0$$

$$and\ 0<j\leq b\ for\ 0<i\leq N \qquad \text{...Equation (2. 14)}$$

Then Equation (2.14) can be written as

$$y(k) = -F_0 + \sum_{j=1}^{b} 2^{-j} F_i \qquad \text{...Equation (2. 15)}$$

If the filter coefficients are pre-known then the values of F_i in Equation (2.14) can be evaluated and these values can be stored in a Read-only-Memory (ROM). This ROM is nothing but a 2^N Look-up-Table (LUT) containing all the possible combinations of the filter coefficients evaluated in Equation (2.14).

The distributed arithmetic implementation of Equation (2.15) is as shown in *Figure (2.6)*. In this implementation the Shift Registers (SR) are used to store the previous samples of input *x(k)*, i.e. storing *x(k-1), x(k-2)* and so on in SR_1, SR_2 and so on. Each Shift Register has *(b+1)* bits. The N-outputs which are binary digits of the Shift Registers are used to address the ROM unit. Thus, after the *j-th* right shift the ROM address will be $x(k)_j, x(k-1)_j\ x(k-2)_j$ and so on. The corresponding evaluated output from the Look-up-Table of the ROM is loaded into register A and is added with a partial register B, where B is acting liking an accumulator having the pre-computed accumulated value.

The result after each addition is divided by 2 till all the *b* bits are shifted from the shift registers. The final result is then subtracted with F_0 and this result is stored in partial register C from which the output *y(k)* is taken. After every *y(k)* the partial registers A, B and C are set to zero.

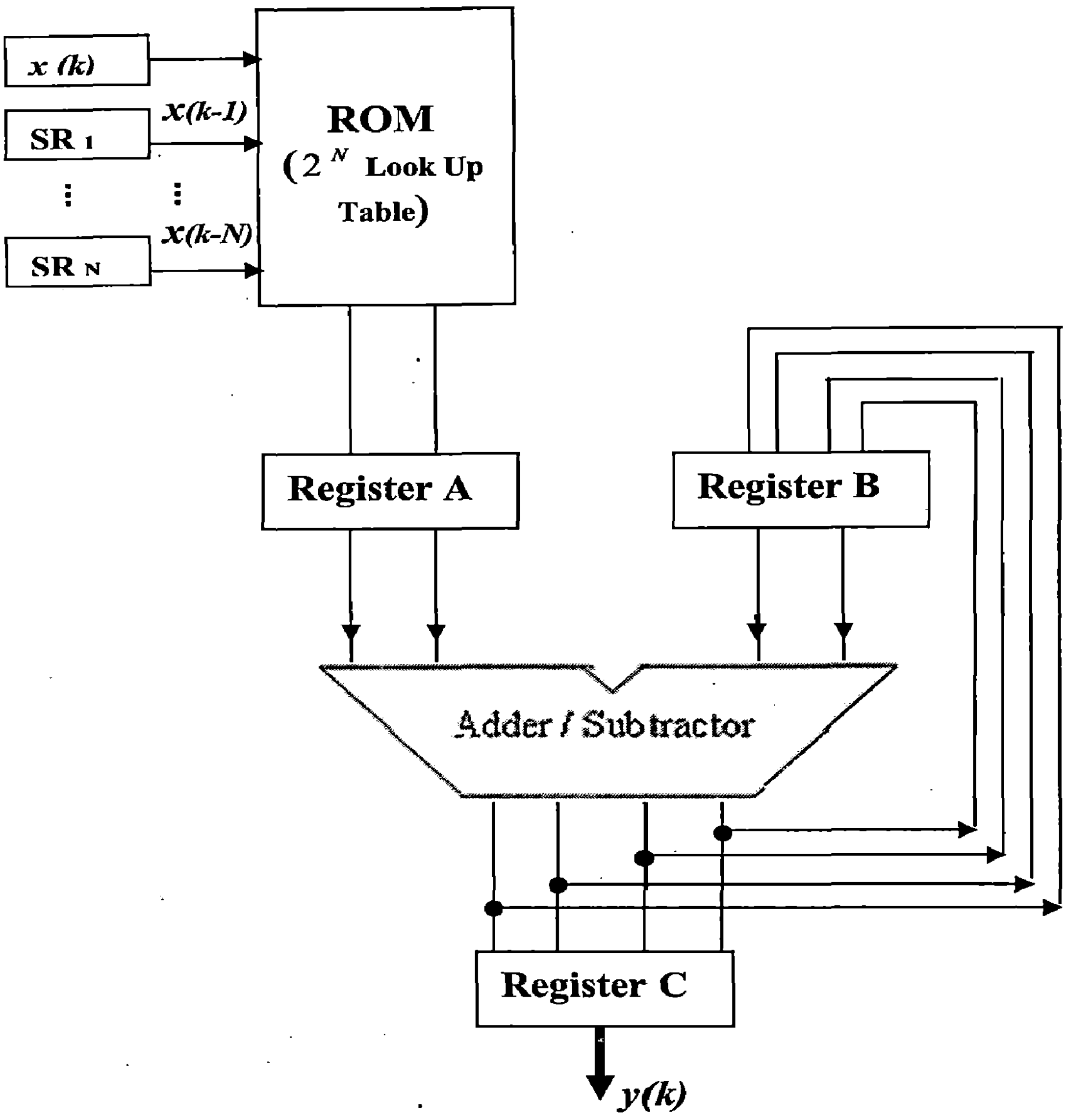


Figure 2.6: Basic Block diagram of Distributed Arithmetic approach

This approach is implemented by considering an Eight-order filter in the next chapter.

2.3 Approximating the Filter Coefficients:

In designing a filter, the filter coefficients are determined by using various techniques like frequency sampling, using window functions, by optimization methods, etc. To explain all these techniques is beyond the scope of this chapter, however, we shall consider the ideal characteristics standard filters and then discuss some of the standard window techniques used in approximating the FIR filter.

Firstly, considering the Z-transform of Equation (1.3), we can write it as,

$$y(z) = x(z)\sum_{i=-\infty}^{\infty} a_i z^{-i} \qquad \text{... Equation (2.16)}$$

Or,
$$y(z) = x(z).H(z) \qquad \text{... Equation (2.17)}$$

where
$$H(z) = \sum_{i=-\infty}^{\infty} a_i z^{-i} \qquad \text{... Equation (2.18)}$$

is the impulse response of the filter.

The frequency response of the impulse response is obtained by substituting $z = e^{j\omega}$. Then, the frequency response and its time-domain representation are given as follows:

$$H(e^{j\omega}) = \sum_{i=-\infty}^{\infty} a_i e^{-j\omega i} \qquad \text{... Equation (2.18)}$$

Taking the inverse Fourier-transform, and letting *h(nT)* as *h(n)* we have,

$$h(n) = \frac{1}{2\Pi}\int_{-\Pi}^{\Pi} H(e^{j\omega}) e^{-j\omega i}\, d\omega \qquad \text{.. Equation (2.19)}$$

The ideal magnitude response of standard Low-Pass filter is as shown in *Figure* 2.7.

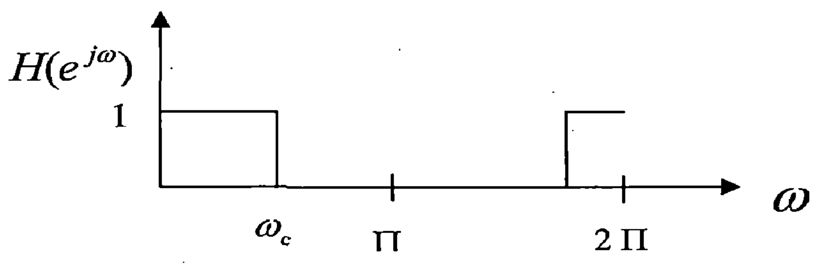


Figure 2.7: Ideal magnitude response of Low-Pass filter

The ideal characteristics are given by,

$$|H(e^{j\omega})| = \begin{cases} 1, for\ |\omega| \le \omega_c \\ 0, for\ \omega_c < |\omega| < \Pi \end{cases}$$ ...Equation (2. 20)

Substituting Equation (2.20) in Equation (2.19) we obtain

$$h(n) = \frac{1}{2\Pi} \int_{-\omega_c}^{\omega_c} e^{j\omega n} d\omega = \begin{cases} \frac{\omega_c}{\Pi} & for \quad n = 0 \\ \frac{\sin(\omega_c n)}{\Pi n} & for \quad n \ne 0 \end{cases}$$

... Equation (2.21)

As can be seen from Equation (2.21) the impulse response corresponding to the ideal low-pass filter is not realizable, as it has infinite duration and also it is not a causal. The above problem is dealt by realizing a digital filter with finite-duration impulse response. Hence, in general FIR filter coefficients are determined by considering a finite-length impulse response *h(n)* whose frequency response approximates the desired frequency response.

2.4 Windowing Techniques:

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

The truncated filter has coefficients $h'(n)$, given as

$$h'(n) = \omega_i h(n) \qquad \text{... Equation (2.22)}$$

where ω_i are the window coefficients.

In the frequency domain, such a multiplication corresponds to a periodic convolution operation between the frequency response of the ideal filter, $H(e^{j\omega})$, and of the window function, $W(e^{j\omega})$, that is

$$H'(e^{j\omega}) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H\left(e^{j\omega'}\right) W\left(e^{j(\omega-\omega')}\right) d\omega' \qquad \text{... Equation (2.23)}$$

For a *Rectangular* window the window coefficients are defined as

$$\omega_i = \begin{cases} 1 & if\,|\,i\,| \le m \\ 0 & if\,|\,i\,| > m \end{cases} \qquad \text{... Equation (2.24)}$$

For *von Hann Window* the window coefficients is generated by a raised cosine(sometimes it is called the "raised cosine" window) and is given as,.

$$\omega_i = \begin{cases} \frac{1}{2} + \frac{1}{2}\cos\left(\frac{\pi i}{m+1}\right) & if\,|\,i\,| \le m \\ 0 & otherwise \end{cases} \qquad \text{... Equation (2.25)}$$

For *Hamming Window*: the window is a mixture of the uniform and the von Hann windows. It is a function of parameters a and b.

$$\omega_i = \begin{cases} 2a\cos\left(\frac{\pi i}{m}\right) + b & if\,|\,i\,|\le m \\ 0 & otherwise \end{cases}$$ ... Equation (2.26)

(3.7)

where 2a+b = 1.

In *Kaiser Window*: J.F. Kaiser used prolate spheroidal function, Io(x), for a window by making the function's argument depend on the window coefficient, *i*.

$$\omega_i = \begin{cases} \dfrac{I_0\left[\alpha\sqrt{1-\left(\frac{i}{m}\right)^2}\right]}{I_0} & if\,|\,i\,|\le m \\ 0 & otherwise \end{cases}$$ ... Equation (2.26)

(3.8)

Of the four windows discussed above, the uniform window generates the narrowest transition regions, the Hamming is the most widely used window and the Kaiser is the most versatile.

Chapter 3: Implementing the filter in FPGA

3.1 Choosing the Filter structure

Different filter structures are discussed in Chapter 2. 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 four Direct structures of Equation2.7,we have considered only 1-D structure. 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. However, in cascade decomposition, these coefficient sensitivity problems are minimized as we have large number of poles (in case of recursive filter) and zeroes.

In the thesis, an Eighth-order low pass filter is implemented using a cascade decomposition of second-order modules in FPGA.

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

An Eighth-order low-pass filter is implemented using the Distributed Arithmetic approach in the thesis.

3.2 Data Representation

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

In general, there are two kinds of Data representation, one is fixed-point representation and the other is IEEE floating-point representation. In the thesis the data is represented in fixed-point notation. In the fixed-point format, the numbers are usually assumed to be proper fraction. A binary point is usually set between the first and second bit positions of the register as shown in *Figure 3.1* is as given below

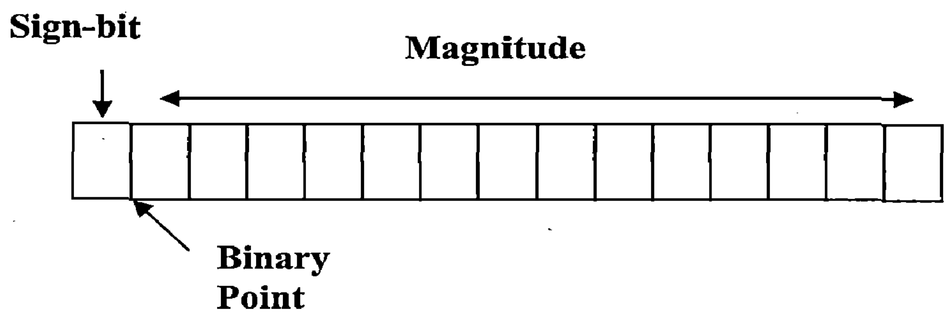


Figure 3.1: Fixed-Point Representation

The numbers are represented in two's-complement format, as this notation is convenient in Digital Signal Processing (DSP) algorithms, because numbers can be added, subtracted, multiplied or divided in straight binary fashion while preserving the sign of the result. The addition or subtraction of two fixed-point numbers falling in the given range may produce a result outside that range, though. Such a result, called *overflow*, it must be either avoided or corrected during DSP calculations.

In the implementation, the two's-complement numbers are represented in the same way as they are represented in Intel 8086. Such implementation is described as follows:

The two's-complement of a 16 bit number N is represented by

$$N = (SM_{14}\, M_{13} \,\ldots\ldots M_1\, M_0) \qquad \ldots \text{ Equation 3.1}$$

Where $-2^{-15} \leq N \leq 2^{15} - 1$

If we consider all numbers to be scaled then, we have

$$-1 \leq N \leq 1 - 2^{-15} \qquad \ldots \text{ Equation 3.2}$$

In Intel 8086 machine, the coefficients are stored as half of their actual values. That is, the VALUE_STORED = [Value * 2^{14} + 0.5] and a left shift operation (multiply by 2) is performed in each routine to compensate for this change.
Note that, here the symbol [x] means largest integer less than x.
However the scaled input samples are multiplied by 2^{15} and then stored in the register removing the fractional part.

The multiplication is the basic operation in computation of output *y(k)*. Considering the multiplication of two *n-bit* numbers, we have the product of *2n* bits. This product is often used as another multiplicand in a later multiplication. As the width of product will increases after each such multiplication, it is impractical to represent such large products in the computer using fixed point arithmetic.

Hence we quantize or truncate the product back to *n-bits* before multiplying it with the other number. This way of multiplication is implemented in thesis work while multiplying the coefficients and the input samples. The following steps are considered while multiplication:

1) Load the coefficient a_i and the sample $x\ (k)$ into registers say AX and DX, where AX and DX are given by

$$AX = a_i \times 2^{n-2} \text{ and } DX = x(k) \times 2^{n-1}$$ ... Equation 3.3

2) Multiply the sample *x(k)* and the coefficient, stored result in DX, AX register

$$DX,AX = a_i \times 2^{n-2} * x(k) \times 2^{n-1}$$ ... Equation 3.4

$$= a_i x(k)/4 \times 2^{2n-1}$$ ... Equation 3.5

3) In order to quantize the product back to *n-bits* perform double left shift operation of the combined double register DX, AX register then truncate the result back to 2^n bits. That is,

$$(DX, AX)_T = \text{Single Register (R)} = a_i x(k) \times 2^{2n-1} / 2^n$$

$$= a_i x(k) \times 2^{n-1}$$

... Equation 3.6

The above mentioned steps are repeated for every multiplication operation used in the cascaded decomposition structure.

3.3 Implementing Cascade Decomposition

An Eight-order system can be represented from Equation (2.8) by substituting *m* as *4,* then it can be written as,

$$H(z) = \prod_{i=1}^{4}\left(a_{i0} + a_{i1}z^{-1} + a_{i2}z^{-2}\right) = \prod_{i=1}^{4}\left(A_i(z)\right)$$ ... Equation (3.7)

The cascaded block structure and the signal-flow diagram for an Eight-order filter is as shown in *Figure (3.2).*

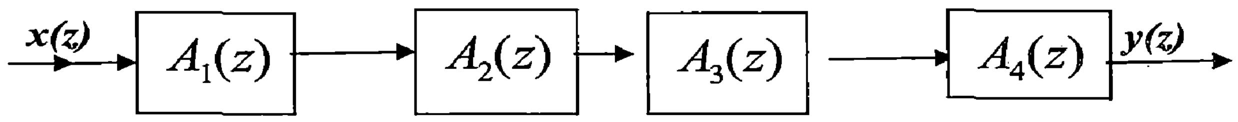


Figure 3.2.1 Block structure of Eight-order Cascade Decomposition

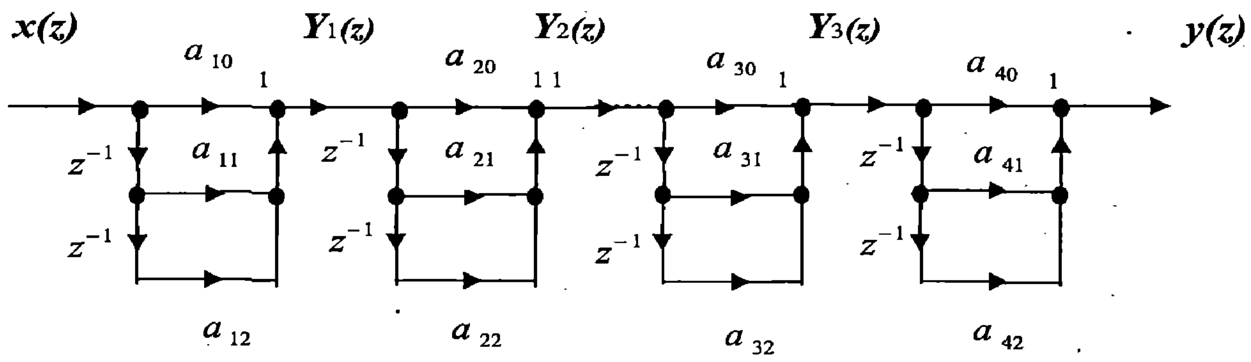


Figure 3.2.2) Signal Flow Diagram of Eight-order Cascade Decomposition

3.3.1 *Algorithm for cascaded decomposition structure:*

1. Finding the necessary variables from the Signal flow diagram:

x(z) → x(k) = Xk

y(z) → y(k) = Yk

$Y_1(z)$ → $Y_1(k)$ = Y_{1k}

$Y_2(z)$ → $Y_2(k)$ = Y_{2k}

$Y_3(z)$ → $Y_3(k)$ = Y_{3k}

z^{-1} **$Y_1(z)$ → $Y_1(k-1)$ = Y_{1KM1}**

z^{-2} **$Y_1(z)$ → $Y_1(k-2)$ = Y_{1KM2}**

z^{-1} **$Y_2(z)$ → $Y_2(k-1)$ = Y_{2KM1}**

z^{-2} **$Y_2(z)$ → $Y_2(k-2)$ = Y_{2KM2}**

z^{-1} **$Y_3(z)$ → $Y_3(k-1)$ = Y_{3KM1}**

z^{-2} **$Y_3(z)$ → $Y_3(k-2)$ = Y_{3KM2}**

z^{-1} **X(z) → X(k-1) = XKM1**

z^{-2} **X(z) → X(k-2) = XKM2**

z^{-1} **Y(z) → Y(k -1) = Y_{KM1}**

z^{-2} **Y(z) → Y(k -2) = Y_{KM2}**

In the above equations, "→" operation indicates Inverse Z-transform and "=" operation indicates that the Left-Hand-Side term (LHS) is assigned to the variable at the Right-Hand-Side term (RHS).

2. Initialize all the necessary variables to Zero.

3. Evaluation

a) Read the current input from Analog-Digital Converter and assign it to $\mathbf{X_k}$

b) Determine the following variables:

$Y_{1k} = a_{10} . X_k + a_{11} X_{KM1} + a_{12} . X_{KM2}$ ... Equation (3.8)

$Y_{2k} = a_{20} . Y_{1k} + a_{21} . Y_{1KM1} + a_{22} . Y_{1KM2}$ ... Equation (3.9)

$Y_{3k} = a_{30} . Y_{2k} + a_{31} . Y_{2KM1} + a_{32} . Y_{2KM2}$ ... Equation (3.10)

$Y_k = a_{40} . Y_{3k} +. a_{41} . Y_{3KM1} + a_{42} . Y_{3KM2}$ ... Equation (3.11)

All the above calculated results in each step should be truncated back to input sample bit width before using the result in the next step.

c) Output $\mathbf{Y_k}$ to Digital-Analog Converter.

4. Updating the necessary variables sequentially.

$X_{KM2} = X_{KM1}$ ... Equation (3.12)

$X_{KM1} = X_k$ ... Equation (3.13)

$Y_{1KM2} = Y_{1KM1}$ ... Equation (3.14)

$Y_{1KM1} = Y_{1k}$ ... Equation (3.15)

$Y_{2KM2} = Y_{2KM1}$ ... Equation (3.16)

$Y_{2KM1} = Y_{2k}$ ... Equation (3.17)

$Y_{3KM2} = Y_{3KM1}$ ... Equation (3.18)

$Y_{3k} = Y_{3KM1}$ ... Equation (3.19)

$$Y_{KM2} = Y_{KM1} \quad \text{... Equation (3.20)}$$

$$Y_{KM1} = Y_k \quad \text{... Equation (3.21)}$$

5. **Repeat the steps 3 to 5 sequentially for the filtering action.**

The above algorithm for cascade decomposition is implemented using VHDL language in FPGA.

3.4 Implementing the Distributed Arithmetic approach in FPGA

Using the approach an Eight-order filter is implemented in FPGA. The basic idea of this approach was discussed in Chapter 2. There can be different ways of implementing this approach. One such way is implemented and is discussed as follows:

Rewriting Equation (2.11) as follows:

$$y(k) = \sum_{i=0}^{N} a_i x(k-i)$$

For an Eight-order filter, substituting $N = 8$ we have,

$$y(k) = \sum_{i=0}^{8} a_i x(k-i) \quad \text{... Equation (3.22)}$$

Consider the input sample width of *(b+1)* bits where *(b+1)th* bit is the sign bit and the other b. bits represent the magnitude of the sample.

In order to implement the above inner product in Equation (3.22) one can consider an Look-Up-Table N address bits, where N is the order of the filter. Then we can have 2^N different combinations of the input coefficients. Thus we need to have 2^N locations in order to store the coefficients in the memory. As N is *8* here, we need to have *256* locations allocated for storing the coefficients in the memory.

One alternative way to implement the Equation (2.11) is to break down the summation of $(N+1)$ terms into several smaller sums. Lets say that there are k sums of M_p terms each. That is,

$$N+1=\sum_{p=1}^{k} M_p \qquad \text{where } p = 1, 2, \ldots\ldots k$$

... Equation (3.23)

In this case, we need to have k Look-Up-Tables and in each Look-Up-Table requires 2^{M_p} terms. Thus the memory is reduced it requires $k\times 2^{M_p}$ locations.

In the thesis, the above mentioned alternative approach is implemented. Here as N is *8,* the value of k is taken as 3 and M_p is taken as 3.

Figure 3.3 shows the way the Eighth-order filter is implemented in FPGA. Here *x(k)* is the current input stored in Shift Register (SR0), all the past inputs are stored in the Shift Registers SR1, SR2, ..., SR8. That is, *x(k-1)* is stored in SR1, *x(k-2)* in SR2 and so on.

ROM1, ROM2, ROM3 are the three Look-Up-Tables storing the filter coefficients. These Look-Up-Tables are as shown in *Figure 3.4.*

R1, R2 and R3 are the partial registers storing the data coming from the Look-Up-Tables.

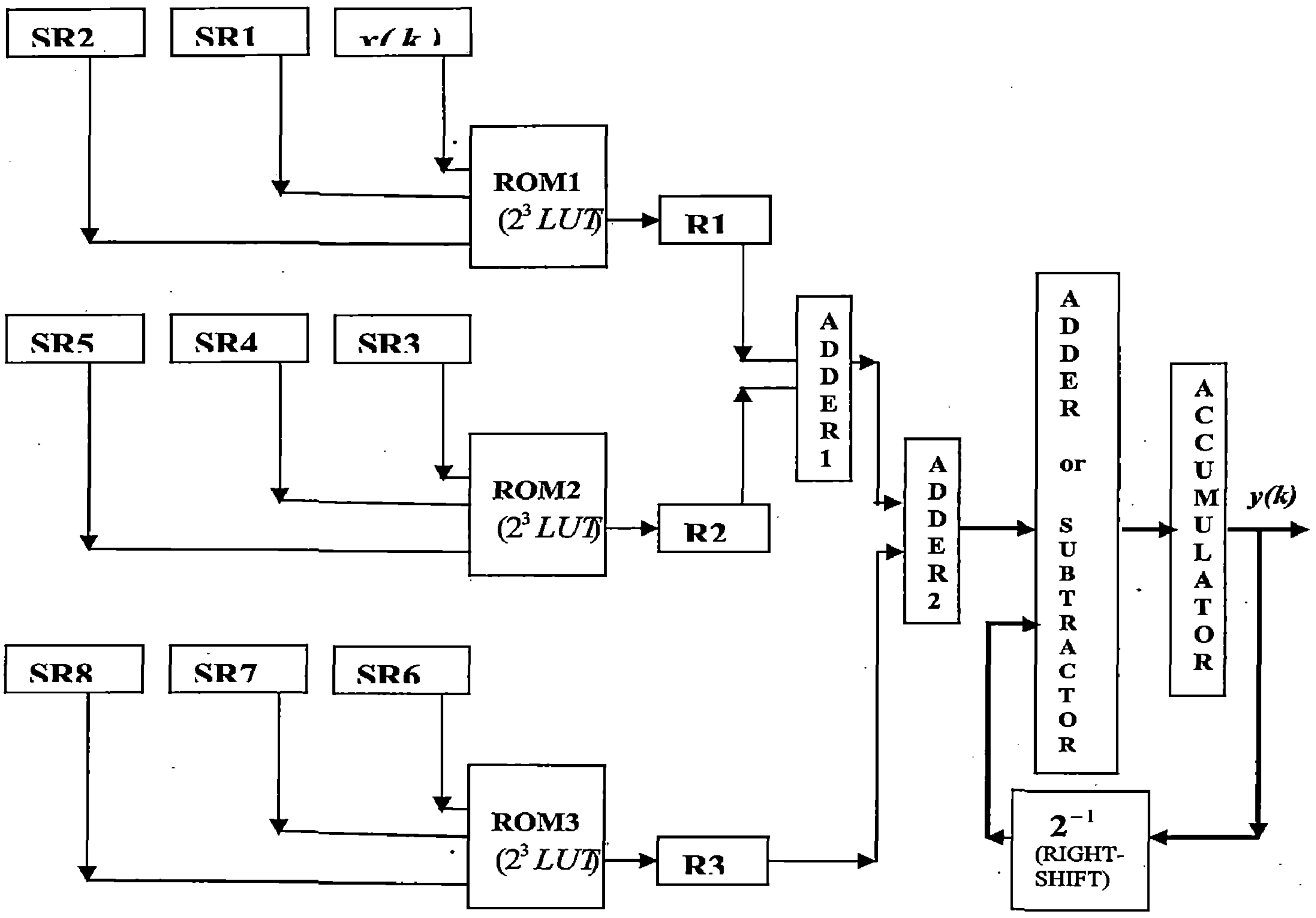


Figure 3.3 Block Diagram for an Eight-order filter using DA approach

ADDRESS	DATA
0 0 0	0
0 0 1	a_0
0 1 0	a_1
0 1 1	$a_1 + a_0$
1 0 0	a_2
1 0 1	$a_2 + a_0$
1 1 0	$a_2 + a_1$
1 1 1	$a_2 + a_1 + a_0$

ROM1

ADDRESS	DATA
0 0 0	0
0 0 1	a_3
0 1 0	a_4
0 1 1	$a_4 + a_3$
1 0 0	a_5
1 0 1	$a_5 + a_3$
1 1 0	$a_5 + a_4$
1 1 1	$a_5 + a_4 + a_3$

ROM2

ADDRESS	DATA
0 0 0	0
0 0 1	a_6
0 1 0	a_7
0 1 1	$a_7 + a_6$
1 0 0	a_8
1 0 1	$a_8 + a_6$
1 1 0	$a_8 + a_7$
1 1 1	$a_8 + a_7 + a_6$

ROM3

Figure 3.4: Contents of three Look-Up-Tables

3.4.1 Algorithm implementing the Distributed Arithmetic approach:

1) Initialize all the Shift registers, Partial registers and the Accumulator to zero.
2) Read the current input sample *x(k)* from A/D converter and store it in SR0.
3) Generate the Address to ROM1, ROM2, ROM3 Look-Up-Tables from the Shift Registers by shifting the register contents one bit at a time.
4) Store the data from the Look-Up-Tables into the partial registers.
5) Add the contents of the partial registers ADDER1, ADDER2 adders by taking care of the overflow.
6) Shift the contents of Accumulator to Right and then add it with the result evaluated in step 5.
7) Repeat steps 3 to 6 till the Most-Significant Bit (sign-bit) is reached.
8) When the sign bit is reached, subtract the result from obtained from step 5 with the contents of the accumulator.
9) Quantize the final accumulated value in the accumulator back to the input sample width and then output this value which is *y(k)* to D/A converter
10) Update the Shift Registers such that the contents of SR7 go to SR8, SR6 to SR7, SR5 to SR6 and so on.
11) Clear the accumulator content and the partial registers contents to zero.
12) Repeat steps 2 to 10 for the filtering action.

The above algorithm is implemented in FPGA.

CHAPTER 4: Simulation Results AND Discussions

Considering an Eight-order low-pass filter defined by the following transfer function:

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

Where N = 8, and the values of the filter coefficients corresponding to cut-off frequency ω_c=0.5 *rads* are given by:

$a_0 = -0.0202779$	$a_5 = -0.29463011$
$a_1 = 0.0542924$	$a_6 = 0.05275543$
$a_2 = 0.05275543$	$a_7 = 0.05429246$
$a_3 = -0.29463011$	$a_8 = -0.0202779$
$a_4 = -0.564509$	

Factorizing H(z) and writing in the form of

$$H(z) = \prod_{i=1}^{m} \left(a_{i0} + a_{i1} z^{-1} + a_{i2} z^{-2}\right),$$

Where

m = 4

$a_{10} = 0.0711508$	$a_{30} = 0.39611047$
$a_{11} = -0.4002231$	$a_{31} = 0.7633587$
$a_{12} = 0.76335878$	$a_{32} = 0.39611047$
$a_{20} = 0.4941662$	$a_{40} = -1.4559792$
$a_{21} = 0.76335878$	$a_{41} = 0.76335878$
$a_{22} = 0.4941662$	$a_{42} = -0.13570834$

Taking the input samples as follows

$x(0) = 0.0$	$x(6) = 0.8$	$x(12) = -0.4$	$x(18) = -0.4$
$x(1) = 0.2$	$x(7) = 0.6$	$x(13) = -0.6$	$x(19) = -0.2$
$x(2) = 0.4$	$x(8) = 0.4$	$x(14) = -0.8$	$x(20) = 0.0$
$x(3) = 0.6$	$x(9) = 0.2$	$x(15) = -0.98$	
$x(4) = 0.8$	$x(10) = 0$	$x(16) = -0.8$	
$x(5) = 0.98$	$x(11) = -0.2$	$x(17) = -0.6$	

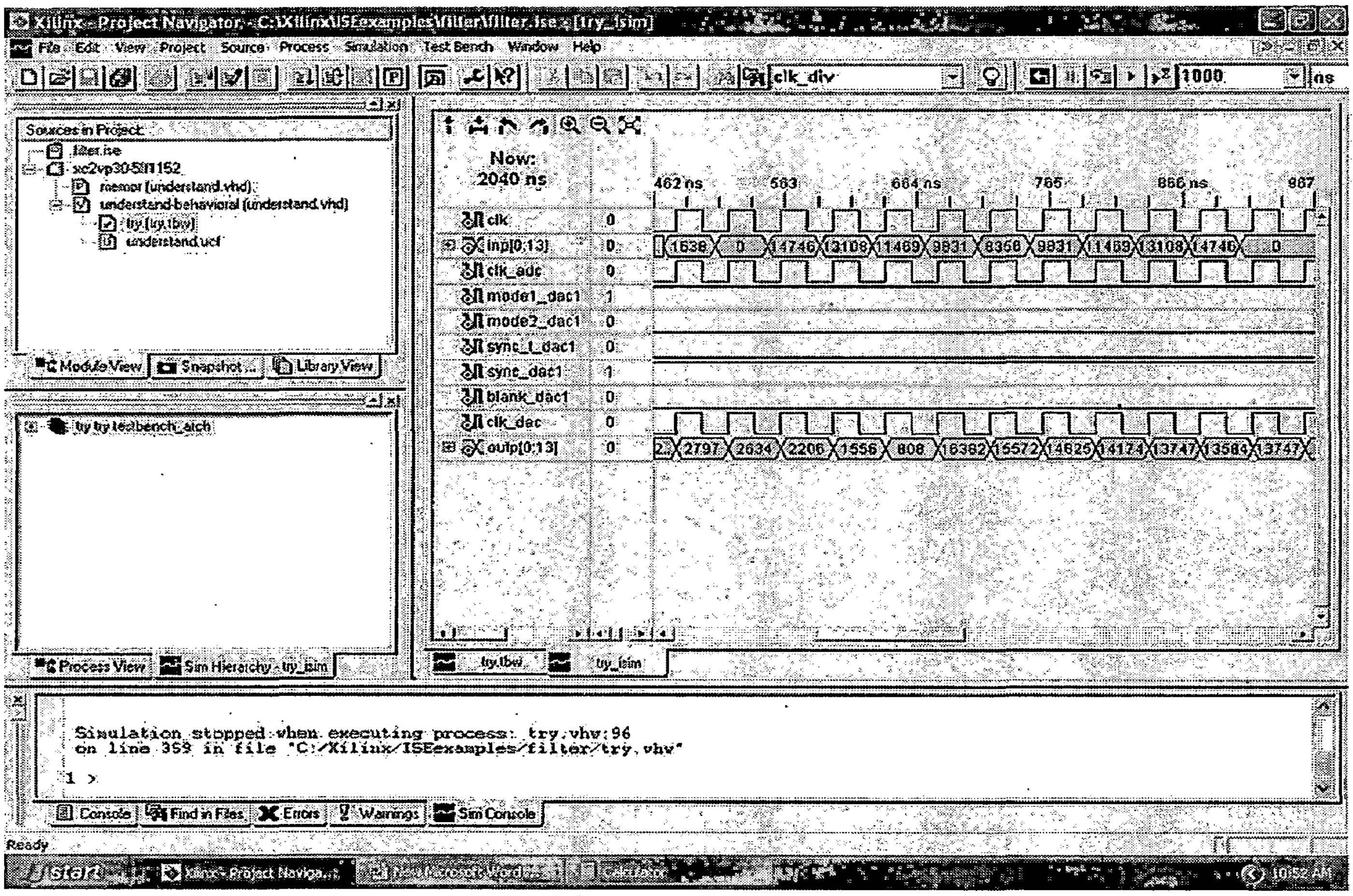
The above input samples are given to the filter implemented in FPGA,
These samples are stored as

$x(0) = 0$	$x(6) = 6553$	$x(12) = 13108$	$x(18) = 131080$
$x(1) = 1638$	$x(7) = 4915$	$x(13) = 11469$	$x(19) = 14746$
$x(2) = 3276$	$x(8) = 3276$	$x(14) = 9831$	$x(20) = 0$
$x(3) = 4915$	$x(9) = 1638$	$x(15) = 8356$	
$x(4) = 6553$	$x(10) = 0$	$x(16) = 9831$	
$x(5) = 8028$	$x(11) = 14746$	$x(17) = 11469$	

The following sampled outputs are obtained for the given above input samples after simulation:

$y(0) = 0$	$y(6) = 1665$	$y(12) = 1556$	$y(18) = 13747$
$y(1) = 31$	$y(7) = 2238$	$y(13) = 808$	$y(19) = 13584$
$y(2) = 108$	$y(8) = 2634$	$y(14) = 16382$	$y(20) = 13477$
$y(3) = 300$	$y(9) = 2797$	$y(15) = 15572$	
$y(4) = 623$	$y(10) = 2634$	$y(16) = 14825$	
$y(5) = 1109$	$y(11) = 2206$	$y(17) = 14174$	

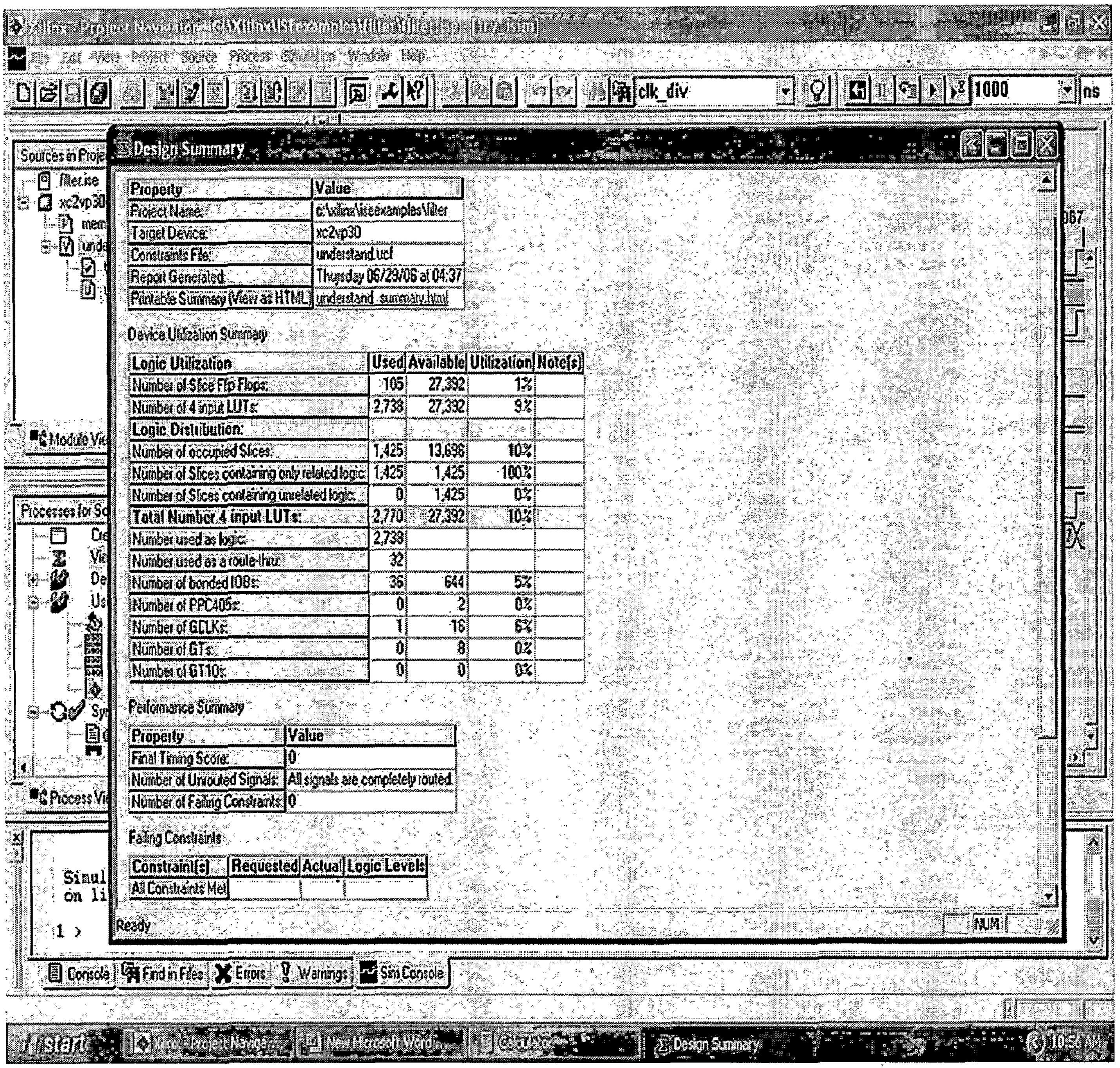
The simulation results in Xilinx ISE test-bench waveform as follows:



```
Simulation stopped when executing process: try.vhw:96
on line 359 in file "C:/Xilinx/ISEexamples/filter/try.vhw"

1 >
```

The Device utilization information for cascade decomposition is given as follows:



Design Summary

Property	Value
Project Name:	c:\xilinx\iseexamples\filter
Target Device:	xc2vp30
Constraints File:	understand.ucf
Report Generated:	Thursday 06/29/06 at 04:37
Printable Summary (View as HTML)	understand_summary.html

Device Utilization Summary

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops:	105	27,392	1%	
Number of 4 input LUTs:	2,738	27,392	9%	
Logic Distribution:				
Number of occupied Slices:	1,425	13,696	10%	
Number of Slices containing only related logic:	1,425	1,425	100%	
Number of Slices containing unrelated logic:	0	1,425	0%	
Total Number 4 input LUTs:	2,770	27,392	10%	
Number used as logic:	2,738			
Number used as a route-thru:	32			
Number of bonded IOBs:	36	644	5%	
Number of PPC405s:	0	2	0%	
Number of GCLKs:	1	16	6%	
Number of GTs:	0	8	0%	
Number of GT10s:	0	0	0%	

Performance Summary

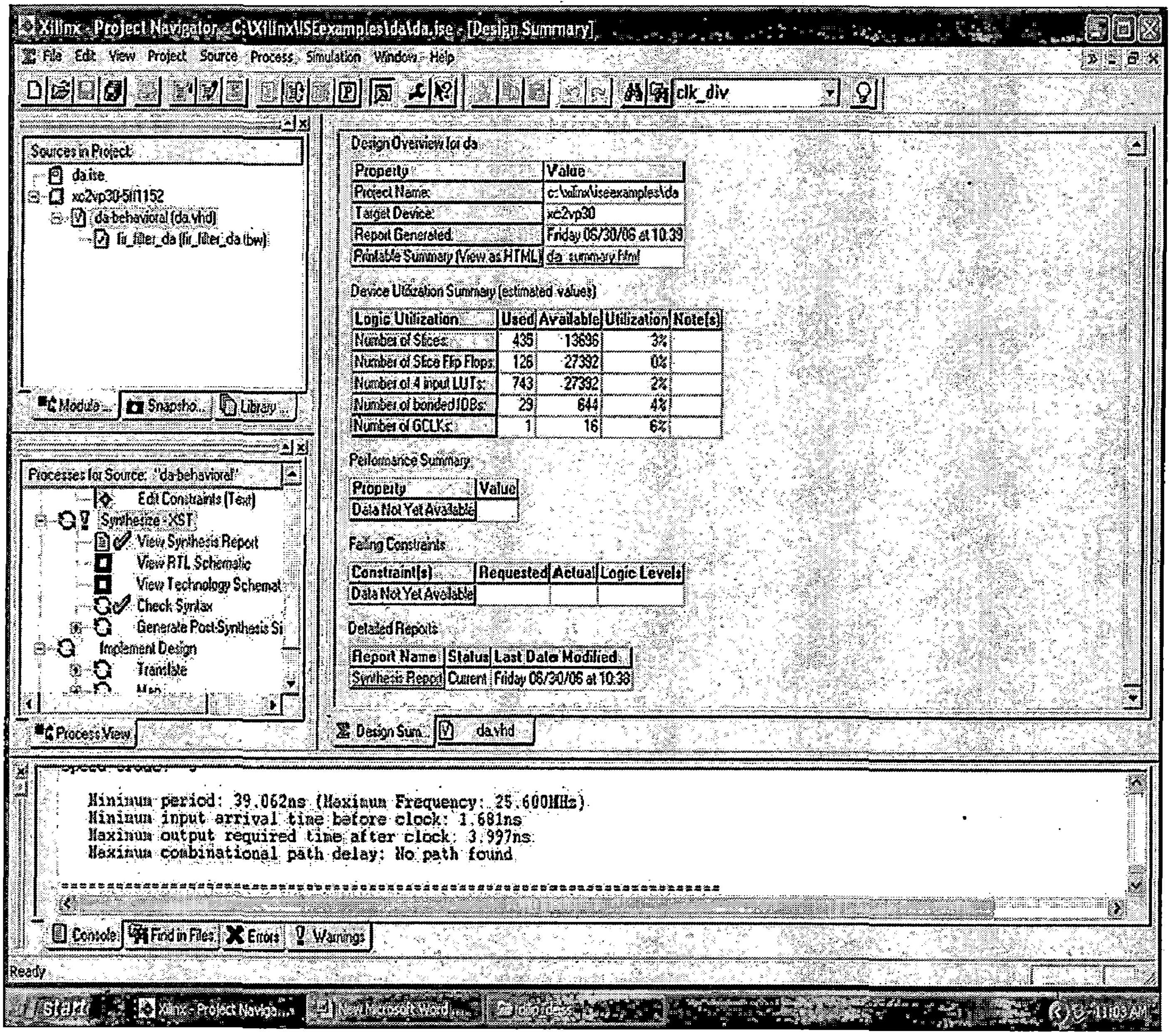
Property	Value
Final Timing Score:	0
Number of Unrouted Signals:	All signals are completely routed.
Number of Failing Constraints:	0

Failing Constraints

Constraint(s)	Requested	Actual	Logic Levels
All Constraints Met			

Ready

The Device utilization information for Distributed Arithmetic approach is given as follows:


Xilinx - Project Navigator - C:\Xilinx\ISEexamples\da\da.ise - [Design Summary]
File Edit View Project Source Process Simulation Window Help
clk_div
Sources in Project
da.ise
xc2vp30-5ff1152
da-behavioral (da.vhd)
fir_filter_da (fir_filter_da.bw)
Module... Snapsho... Library...
Processes for Source: "da-behavioral"
Edit Constraints (Text)
Synthesize - XST
View Synthesis Report
View RTL Schematic
View Technology Schemat
Check Syntax
Generate Post-Synthesis Si
Implement Design
Translate
Process View
Design Sum...
da.vhd
Console
Find in Files
Errors
Warnings
Ready
Start
Xilinx - Project Naviga...

Design Overview for da

Property	Value
Project Name:	c:\xilinx\iseexamples\da
Target Device:	xc2vp30
Report Generated:	Friday 06/30/06 at 10:39
Printable Summary (View as HTML)	da_summary.html

Device Utilization Summary (estimated values)

Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slices:	435	13696	3%	
Number of Slice Flip Flops:	126	27392	0%	
Number of 4 input LUTs:	743	27392	2%	
Number of bonded IOBs:	29	644	4%	
Number of GCLKs:	1	16	6%	

Performance Summary

Property	Value
Data Not Yet Available	

Failing Constraints

Constraint(s)	Requested	Actual	Logic Levels
Data Not Yet Available			

Detailed Reports

Report Name	Status	Last Date Modified
Synthesis Report	Current	Friday 06/30/06 at 10:38

```
   Minimum period: 39.062ns (Maximum Frequency: 25.600MHz)
   Minimum input arrival time before clock: 1.681ns
   Maximum output required time after clock: 3.997ns
   Maximum combinational path delay: No path found

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

The user constraints file is given as:

```
NET "clk" TNM_NET = "clk";
TIMESPEC "TS_clk" = PERIOD "clk" 100 ns HIGH 50 %;
OFFSET = IN 95 ns BEFORE "clk"  ;
#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments
NET "BLANK_DAC1"  LOC = "AH33"  ;
NET "clk"  LOC = "D18"  ;
NET "clk_adc"  LOC = "AA30"  ;
NET "clk_dac"  LOC = "AK34"  ;
NET "inp<0>"  LOC = "AA27"  ;
NET "inp<10>"  LOC = "W31"  ;
NET "inp<11>"  LOC = "W32"  ;
NET "inp<12>"  LOC = "W33"  ;
NET "inp<13>"  LOC = "Y33"  ;
NET "inp<1>"  LOC = "AA28"  ;
NET "inp<2>"  LOC = "AA25"  ;
NET "inp<3>"  LOC = "AA26"  ;
NET "inp<4>"  LOC = "Y31"  ;
NET "inp<5>"  LOC = "Y32"  ;
NET "inp<6>"  LOC = "Y29"  ;
NET "inp<7>"  LOC = "Y28"  ;
NET "inp<8>"  LOC = "Y25"  ;
NET "inp<9>"  LOC = "Y26"  ;
NET "MODE1_DAC1"  LOC = "AK31"  ;
NET "MODE2_DAC1"  LOC = "AK32"  ;
NET "outp<0>"  LOC = "AJ30"  ;
NET "outp<1>"  LOC = "AJ31"  ;
NET "outp<2>"  LOC = "AH29"  ;
NET "outp<3>"  LOC = "AH30"  ;
NET "outp<4>"  LOC = "AG29"  ;
NET "outp<5>"  LOC = "AG30"  ;
NET "outp<6>"  LOC = "AG28"  ;
NET "outp<7>"  LOC = "AL33"  ;
NET "outp<8>"  LOC = "AL34"  ;
NET "outp<9>"  LOC = "AK33"  ;
NET "SYNC_DAC1"  LOC = "AJ34"  ;
NET "SYNC_T_DAC1"  LOC = "AJ33"
```

Chapter 5: Overview of Virtex-2Pro and kit

5.1 Virtex 2 Pro FPGA

The Virtex-II Pro Platform FPGA solution is the most technically sophisticated silicon and software product development in the history of the programmable logic industry. The Virtex-II Pro family marks the first paradigm change from programmable logic to programmable systems, with profound implications for leading-edge system architectures in networking applications, deeply embedded systems, and digital signal processing systems. It allows custom user-defined system architectures to be synthesized, next-generation connectivity standards to be seamlessly bridged, and complex hardware and software systems to be co-developed rapidly with in-system debug at system speeds.

The Virtex-2 Pro ordering information is given in *Figure 5.1* as follows

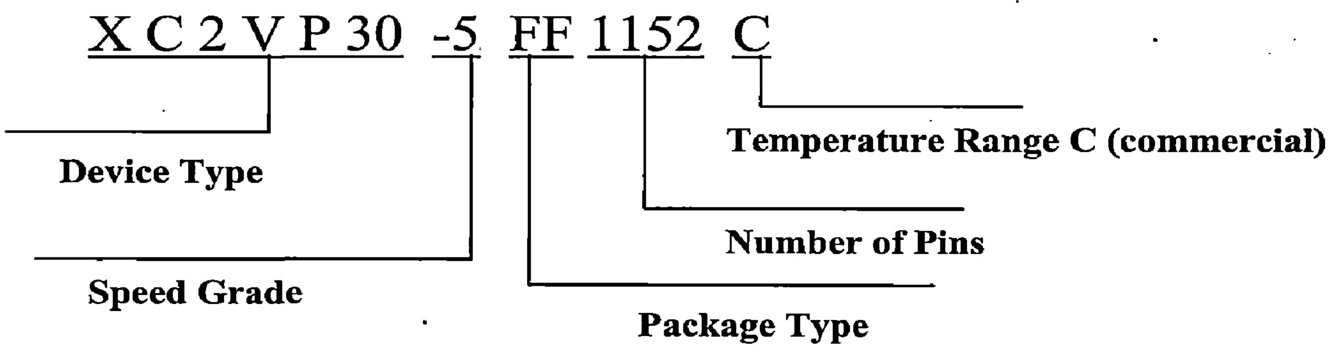


Figure 5.1: Virtex-2Pro ordering information

Virtex-2 Pro are user programmable gate arrays with configurable elements and embedded blocks optimized for high density and high performance system designs. A brief overview of the components Virtex-2 Pro is given as follows:

1. Embedded High-Speed Serial Transceiver

These devices have Rocket IO Multi-Giga bits. The RocketIO Multi-Gigabit Transceiver, based on Mindspeed's SkyRail technology, is a flexible parallel-to-serial and serial-to-parallel embedded transceiver used for high-bandwidth interconnection between buses, backplanes, or other subsystems. Multiple user instantiations in an FPGA are possible, providing up to 120 Gb/s of full-duplex raw data transfer. Each channel can be operated at a maximum data transfer rate of 3.125 Gb/s.

2. Power PC

The Power PC is the hard processor core that is embedded into FPGA fabric. The PPC405 RISC CPU can execute instructions at a sustained rate of one instruction per cycle. On-chip instruction and data cache reduce design complexity and improve system throughput.

3. Input / Output Blocks

I / O blocks provide the interface between package pins and the internal configuration logic. Most popular and leading-edge I/O standards are supported by the programmable IOBs.

4. Configuration Logic Blocks (CLBs)

Configurable Logic Blocks (CLBs) provide functional elements for combinatorial and synchronous logic, including basic storage elements. CLB resources include four slices and two 3- state buffers.
Each slice is equivalent and contains:

- Two function generators (F & G)
- Two storage elements
- Arithmetic logic gates
- Large multiplexers
- Wide function capability
- Fast carry look-ahead chain
- Horizontal cascade chain (OR gate)

The function generators F & G are configurable as 4-input look-up tables (LUTs), as 16-bit shift registers, or as 16-bit distributed SelectRAM+ memory. In addition, the two storage elements are either edge-triggered D-type flip-flops or level-sensitive latches. Each CLB has internal fast interconnect and connects to a switch matrix to access general routing resources.

5. Block Select RAM + Memory

The block SelectRAM+ memory resources are 18 Kb of True Dual-Port RAM, programmable from 16K x 1 bit to 512 x 36 bit, in various depth and width configurations. Each port is totally synchronous and independent, offering three "read-during-write" modes. Block SelectRAM+ memory is cascadable to implement large embedded storage blocks.

6. Embedded 18-bit x 18-bit dedicated multiplier blocks:

These are 18x18 multipliers. A multiplier block is associated with each Select RAM+ memory block. The multiplier block is a dedicated 18 x 18-bit 2s complement signed multiplier, and is optimized for operations based on the block Select RAM+ content on one port. The 18 x 18 multiplier can be used independently of the block Select RAM+ resource. These make

Read/multiply/accumulate operations and DSP filter structures are extremely efficient.

7. Digital Clock Manager

The DCM and global clock multiplexer buffers provide a complete solution for designing high-speed clock schemes. Up to eight DCM blocks are available. To generate de-skewed internal or external clocks, each DCM can be used to eliminate clock distribution delay. The DCM also provides 90-, 180-, and 270-degree phase-shifted versions of its output clocks. Fine-grained phase shifting offers high-resolution phase adjustments in increments of 1/256 of the clock period. Very flexible frequency synthesis provides a clock output frequency equal to a fractional or integer multiple of the input clock frequency. Virtex-II Pro devices have 16 global clock MUX buffers, with up to eight clock nets per quadrant. Each clock MUX buffer can select one of the two clock inputs and switch glitch-free from one clock to the other. Each DCM can send up to four of its clock outputs to global clock buffers on the same edge. Any global clock pin can drive any DCM on the same edge.

8. Routing Resources

The IOB, CLB, block SelectRAM+, multiplier, and DCM elements all use the same interconnect scheme and the same access to the global routing matrix. Timing models are shared, greatly improving the predictability of the performance of high-speed designs. There are a total of 16 global clock lines, with eight available per quadrant. In addition, 24 vertical and horizontal long lines per row or column, as well as massive secondary and local routing resources, provide fast interconnect. Virtex-II Pro buffered interconnects are relatively unaffected by net fanout, and the interconnect layout is designed to minimize crosstalk.

9. Configuration

Virtex-II Pro devices are configured by loading the bitstream into internal configuration memory using one of the following modes:

- Slave-serial mode
- Master-serial mode
- Slave Select-MAP mode
- Master Select-MAP mode
- Boundary-Scan mode (IEEE 1532)

Virtex-2Pro has 10 members. In the dissertation XC2VP30 device is used which has the following resources:

Number of Rocket IO Transceivers Blocks:	10
Power PC Processor Blocks:	2
Logic Cells:	30816
Slices:	13696
18 x 18 Multipliers Blocks:	136
Block Select RAM + (18 Kb blocks):	136
Block RAM (kb):	2448
DCM:	8
Maximum User I/O pads:	644

5.2 Virtex-2 Pro Video Processing Board

This board is used in implementing the filter. A brief overview about this board is as given below:

Virtex-II Pro based video processing card with PCI interface offers a cost-effective platform for developing video and multimedia based applications. With on board high speed video ADC, DAC the board supports real time video processing of component video signals of NTSC or PAL standards. The on board SRAM and Flash memories may be used as data/code store for PowerPC or as video coefficient/data buffer(s).

The board supports three different modes of FPGA configuration. Configuration through PROM, JTAG Port and PCI. Flash PROM memory can be programmed with code for PowerPC through PCI interface.

This board can also be used as standalone video processing board. In that case RS232 port provided on board will be used to program flash PROM. Further this platform is optimised for experimentation with 32-bit IBM PowerPC™ RISC processor core integrated into the FPGA fabric.

This board contains the following:

FPGA

Xilinx Virtex-II Pro XC2VP30 device in FF1152 package.

These are platform FPGA's that are based on IP cores and customized modules, optimized for high density and high performance system design. They empower complete solutions for telecommunication, wireless, networking, and Video and DSP applications.

PowerPC 405 Processor

The PPC405 RISC CPU can execute instructions at a sustained rate of one instruction per cycle. On-chip instruction and data cache reduce design complexity and improve system throughput.

Clocks

User has option of using 2 different clock sources on board which provide all necessary clocks for User logic and PowerPC The clock sources provided on board are as follows

Clock Source1: 32 MHz clock oscillator – supplied as standard and can be used as system clock for PPC.

Clock Source2: Socket for user clock source (foot print compatible with clock sources from 40 to 300 MHz. the clock sources are connected to the global clock inputs. Thus user can use these external clocks as input to the on-chip DCM's.

Flash PROM

1.5 MB of Flash PROM is provided as standard, using three 512K X 16 PROMs. Can be upgraded to 3 MB, as these proms are footprint compatible with 1M X 16 PROMs.

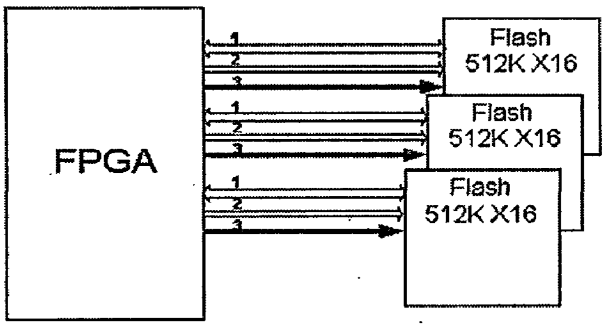


Figure 5.2 Virtex2Pro Flash PROM

SRAM

Five 1M X 16 SRAM devices are independently interfaced to the on board FPGA.

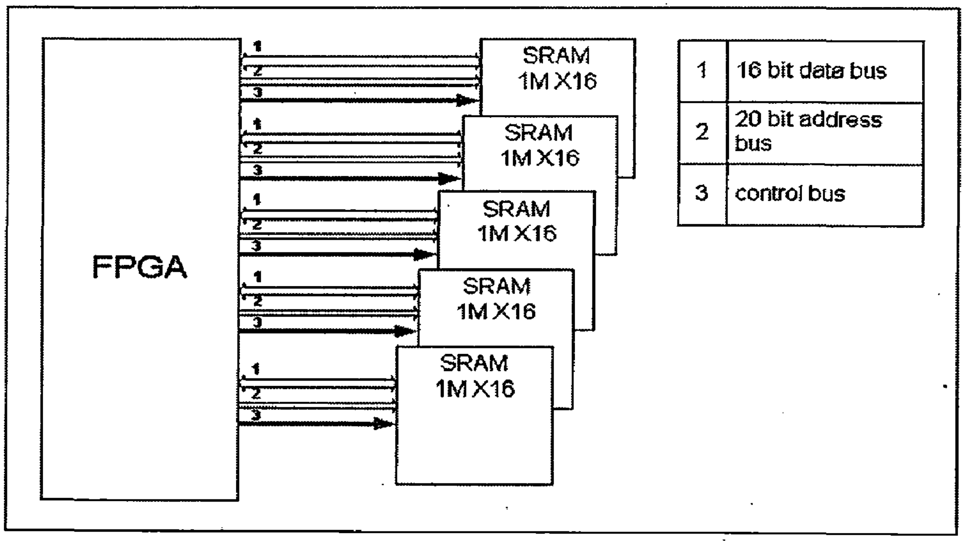


Figure 5.3: Virtex-2 Pro SRAM

RS 232 Port

RS232C compatible connectivity is provided using device MAX3223. Signals provided are Rx, Tx, RTS and CTS. These signals are terminated on a 10 pin FRC connector on board and a 10 pin FRC to 9-pin D connector interface cable is provided as standard accessory with the board. Is compatible with the UART core provided by the Xilinx EDK.

PCI Interface

A 32-bit, 33 MHz PCI interface, using PLX-9054 master interface with DMA capability.

Analog Input : AD9240.

AD9240 is the Analog – Digital Converter. On the board one analog input channel is available with the following specifications.

- Resolution – 14 bits
- Max Sampling rate – 10 MSPS
- Input range - 0 to 5 Volts, single ended
- Input buffer – using AD8052 op-amp.
- Connector type – SMA.

AD9240 is useful in various applications such as imaging, communications, and medical and data acquisition systems.
In the Dissertation this A./ D converter is used to supply the input.

Video Input

One video input channel is available with the following specifications.

- Video ADC - TLV5734
- Video signal format – NTSC / PAL compliant RGB /YUV component video signal.
- Resolution – 8 bit

- Sampling rate –30 MSPS maximum.
- Input range - 1 Vp-p.
- Input buffer – using AD8052 op-amp.
- Connector type – SMA
- Selectable clamping function for RGB/YUV applications.
- Selectable Output Data Format for 4:4:4 (RGB, YUV), 4:2:2 and 4:1:1 (YUV) Format.

Video Output

Two video output channels are provided on board with the following specifications

- Video DAC - THS8133
- Video signal format – NTSC / PAL compliant RGB /YUV component video signal.
- Resolution – 10 bit
- Sampling rate –80 MSPS maximum.
- Connector type – SMA

THS8133 provides current output; these current outputs can be converted into NTSC/PAL standard voltage levels by connecting a double terminated 75 ohms load.

Sync Generation

Using sync, sync_t control signals, video sync signals can be added either on AGY (G/Y) channel or on all three channels with 7:3 video/sync ratios. Depending on the timing control of these signals both horizontal and vertical sync signals can be generated as well as either bi-level negative going or tri level pulses can be generated.

Blanking Generation

An additional control input BLANK is provided that will fix the output amplitude on all channels to the blanking level. The absolute amplitude of the blanking level with respect to active video is determined by the GBR or YPbPr operation mode of the device.

Histogram Equalizer

The on board histogrammer LF48410 is capable of generating histograms and cumulative distribution functions of video images. It provides following features:

- 40 MHz data input and computation rate.
- 1024 X 24 bit memory array
- Histograms of images up to 4k X 4k with 10-bit pixel resolution.
- User programmable modes – Histogram, histogram accumulate mode, look-up table mode, Delay memory, single port memory.

Sync Separator

Sync separator EL4583 used on board extracts timing from video sync in NTSC, PAL, and SECAM systems.

- Sync Separator – EL4583
- Input voltage range – 0.5 V to 2 Vp-p.
- Output signals –composite sync signal, vertical sync signal, horizontal sync signal, burst signal, odd/even signal, no signal detect output.

- Connector type – SMA

Digital I/Os

Maximum of 16 true bi-directional IOs are available when using XC2VP30 device. (32 with XC2VP40 and XC2VP50).

Power Supply

When Board is to be used as PCI add on card then Board can be powered from PC's SMPS. While using the board in standalone mode an external power supply will power the board.

User LEDs

8 LEDs are provided on board, which can be used by user to monitor signals from his design.

Reset Switch

Can be used by user as a manual Reset input while verifying his designs.

The above board is as shown in *Figure 5.1*

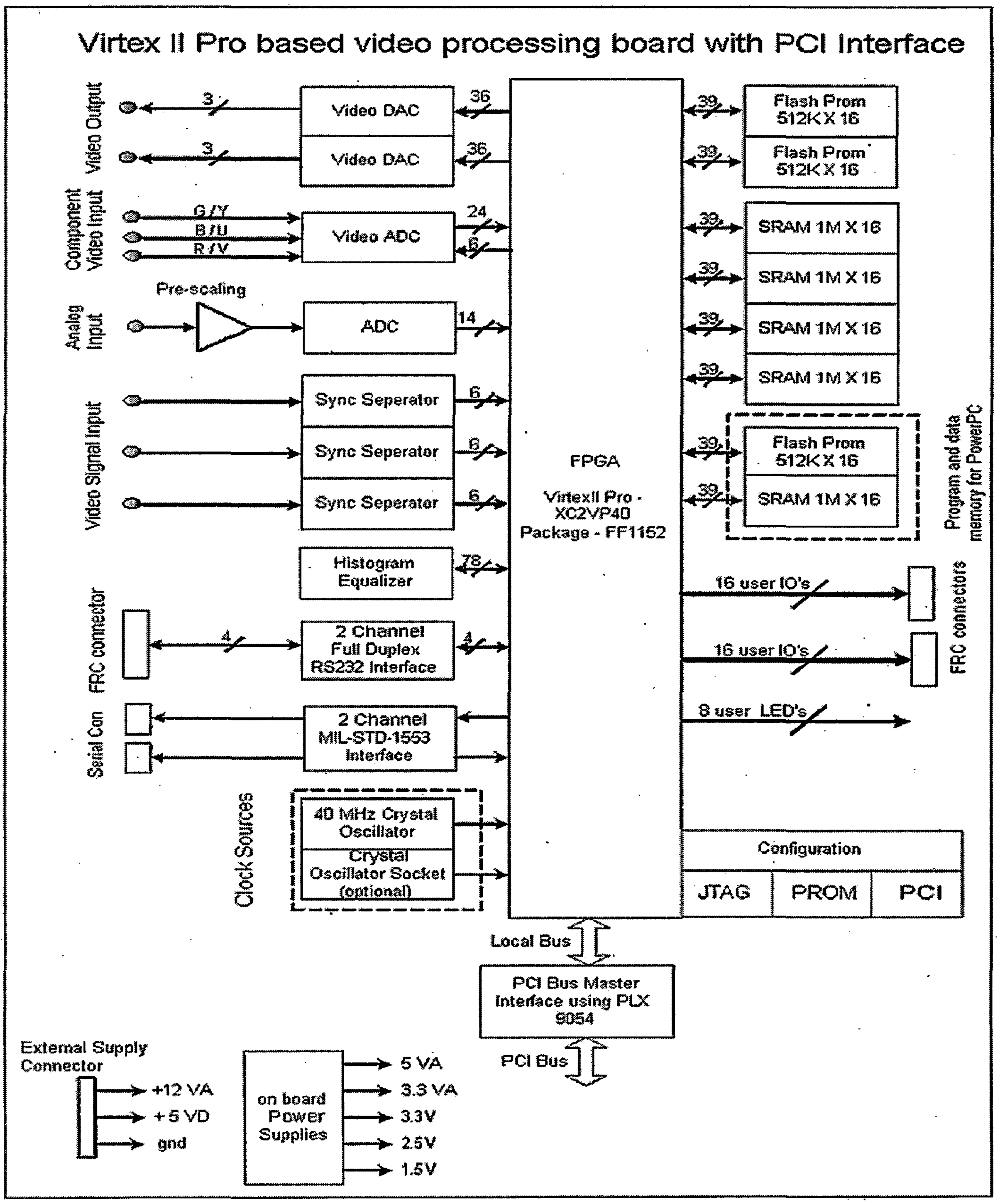


Figure 5.4: Virtex2Pro Board

Conclusion and Future Scope of Work

7.1 Conclusion

An Eight- order low-pass filter Finite Impulse Response (FIR) Filter has been implemented in Virtex-2Pro FPGA. Some of the common filter structures were discussed. And two realizations Cascaded Decomposition and Distributed Arithmetic approach were implemented. A cascade decomposition consisting of four second order modules is realized. In the Distributed Arithmetic Approach three Look-Up-Tables, Eight Shift Registers and Adder/ Subtractor are used to realize the Eight-order filter. Attempts have been made to implement the above filter in real-time.

7.2 Future Scope of Work

- The low-pass FIR filter implemented is of order Eight. A higher order filter can be realized and can be extended to band-pass, band-stop and differentiators

- On the board there is no Digital – Analog converter, one can interface an external D/A converter to the board to view the output signal in Cathode Ray Oscilloscope (CRO).

- The implemented filter uses more logic blocks, one can optimize the design in terms of area occupied.

- The Look-Up-Tables were stored in FPGA internal memory. However these Look-Up-Tables can be implemented in SRAMs and the coefficients can be directly accessed from the external memory. of the board.

References

1. Kaiser J.F. Digital Filters, System Analysis by Digital Computer, F.F. Kuo and J.F.Kaiser, eds. John Wiley & Sons, Inc., 1966, p.p. 218-285
2. Peled, A., and B. Liu: A new approach to the realization of non-recursive Digital filters, IEEE Trans. Audio Electrostatics, AU-21(6), 477-484 (1973).
3. C Sidney Burrus: Digital Filters Structures described by Distributed Arithmetic, IEEE Trans. On Circuits and Systems, vol. CAS-24, N0. 12, December, 1977.
4. Abraham Peled, and Bede Liu: A new hardware realization of Digital Fitlers, IEEE Transactions on Accoustics, Speech, and Signal Processing, vol. ASSP-22, No.6, December 1974.
5. Charles L.. Philips, H.Troy Nagle: Digital Control System Analysis and Design, 3rd Edition, Prentice Hall Publications.
6. Andreas Antoniou: Digital Filters Analysis and Design, TMH Edition 1980, Tata McGraw-Hill Publications.
7. Paulo S.R. Diniz, Eduardo A.B. da Silva, and Sergio L. Netto: Digital Signal Processing, System Analysis and Design, 1st Edition,2002, Cambridge University Press.
8. Uwe Meyer-Baese: Digital Signal Processing with Field-Programmable Gate Arrays, 1st Edition, 2001, Springer-Verlag.
9. Douglas Perry: VHDL, 3rd Edition. Tata Mc. Graw Hill Publications.
10. Virtex-2Pro Handbook.
11. Les Thede: Practical Analog Filter and Design, 1st Edition, 2005, Artech House Inc.
12. Chi-Jui Chou, Satish , Joseph B. Evans: FPGA Implementation of Digital Filters, Telecommunications & Information Sciences Lab, Department of Electrical& Computer Engg, University of Kanvas.