VLSI IMPLEMENTATION OF AN M-BAND WAVELET FILTER

by

K.Yavuz Atabek

B.S. in Electrical and Electronics Option, Naval Academy, 1988

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfilment of
the requirements for the degree of

Master of Science

in

Defence Technology Program, Electronics Option

Boğaziçi University

1996

ACKNOWLEDGMENTS

I would like to express my deepest appreciation to Turkish Naval Forces and Prof. Dr. Yorgo İstefanopulos for starting such a program, thus allowing me to have an invaluable educational and social opportunity in Boğaziçi University. I would also like to express my gratitude to Prof. Dr. Emin Anarım for his continuous support.

Also, I would like to thank to my thesis supervisor Assoc. Prof. Dr. Sina Balkır and Assist. Prof. Dr. Günhan Dündar for their always being more than a teacher.

I am obliged for Assist. Prof. Dr. Cem Ersoy for his kind interest, great understanding and invaluable guiding.

I am also gratified for the friendship among the department people; especially for their help during my first year. In addition, I am grateful to the people in the ECAD laboratory, particularly, to our leader Prof. Dr. Ömer Cerid and dear friends Hakan Binici, İsmet Bayraktaroğlu and Alper Altınordu, for their patience and support .

Special thanks to Assoc. Prof. Dr. Hakan Çağlar and Prof. Dr. Gülen Aktaş for their kind supports.

I would like to thank my friends; Biltor Uluçay for encouraging me to attend this program and Gürcan Elbek for being a part of me like a family and to all my friends, generating an "EKİP spirit" in our lives.

Finally, I would like to appreciate my family for their invaluable, persistent support for all my life and my wife Zeynep İmre Atabek for being a color of my life and great support especially during this thesis period.

ABSTRACT

In this thesis, an integrated circuit realizing M-Band wavelet transform and its perfect reconstruction (PR) counter part is designed. Although *M* is taken as 4 in the design, the M-Band Wavelet Transform has all four filters implemented with only one filter by the help of the shuffling property. With this method, it is possible to realize a VLSI design for the M-Band Wavelet Transform.

The circuit is designed to process 8-bit input signals. So, the filter has 8x8 signed multipliers and 16 bit full adders. After synthesis and optimization of the circuit, which was designed in a fully hierarchical manner in VHDL, its schematic diagram is obtained. From the schematic diagram, the chip layout of the circuit is created.

The circuit was designed, synthesized, optimized, simulated and its IC layout was created and prepared for IC processing by using the Mentor Graphics v.8.4.1 tools running on a Sun Sparc2 workstation. In the design, EuroPractice ES2 1 µm CMOS target technology is used.

ÖZET

Bu tez çalışmasında, M-Bantlı dalgacık dönüşümünü ve tam geri çatmasını gerçekleyecek bir tümleşik devre tasarlanmıştır. Tasarımda kullanılan süzgeç sayısı, M=4 olarak alındığından dolayı, 4 adettir. Buna karşın, M-Bantlı Dalgacık Dönüşümünün devrişime uygunluk özelliği ile 4 süzgeç yerine tek bir süzgeç kullanılarak, tümleşik devre olmaya uygun bir yapı elde edilebilmiştir.

Devre 8 bitlik giriş işaretleri için tasarlanmış olup, süzgeç içinde 8X8 işaretli çarpma yapan çarpıcılar ile 16 bitlik toplayıcılar kullanılmıştır. Tamamı hiyerarşik olarak, en basit kapı elemanına kadar VHDL'de tasarlanan devrenin, sentezleme ve eniyileme işlemlerinden sonra şematik görüntüsü ortaya çıkmıştır. Bu şemadan da tümleşik devre serimi (layout) yaratılmıştır.

Tasarımın yazılması, sentezlenmesi, eniyilenmesi, benzetiminin yapılması ve tümleşik devre seriminin çıkarılması için Sun Sparc2 iş istasyonu üzerinde çalışan Mentor Graphics yazılım paketi, sürüm 8.4.1'den yararlanılmıştır. Ayrıca tasarımda EuroPractice ES2 1 µm. CMOS gerçekleme teknolojisi kullanılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

PR	Perfect Reconstruction
PR-QMF	Perfect Reconstruction Quadrature Mirror Filter
VHDL	Very High Speed Integrated Circuit Description Language
fs	Sampling Frequency
fc	Center Frequency
b	Number of Quantization Bits
$coef_{max}$	Maximum Value of the Coefficients
$qntzr_{new}$	New Quantizer Value used for $\mathrm{coef}_{\mathrm{max}}$
C	Multiplication Constant of the Filters
clk	Clock inputs of the circuits
clr	Clear signal for Synthesis part, generated in Analysis part
clr2	Clear signal for Analysis part

<u>IN ANALYSIS PART (for Figure 4.1a-b)</u>

MBAND_IN	Data input
S13BUS(7:0)	Coefficient input to the 1st tap of FIR filter
S11BUS(7:0)	Coefficient input to the 2nd tap of FIR filter
S9BUS(7:0)	Coefficient input to the 3rd tap of FIR filter
S7BUS(7:0)	Coefficient input to the 4th tap of FIR filter
S5BUS(7:0)	Coefficient input to the 5th tap of FIR filter
S2BUS(7:0)	Coefficient input to the 6th tap of FIR filter
S1BUS(7:0)	Coefficient input to the 7th tap of FIR filter
S14BUS(7:0)	Coefficient input to the 8th tap of FIR filter
S8BUS(7:0)	Data input to the 1st tap of FIR filter
S15BUS(7:0)	Data input to the 2nd tap of FIR filter
S16BUS(7:0)	Data input to the 3rd tap of FIR filter
S10BUS(7:0)	Data input to the 4th tap of FIR filter
S12BUS(7:0)	Data input to the 5th tap of FIR filter
S3BUS(7:0)	Data input to the 6th tap of FIR filter
S6BUS(7:0)	Data input to the 7th tap of FIR filter
S0BUS(7:0)	Data input to the 8th tap of FIR filter
N$72(15:0)	FIR filter output (16 bit)
ING0(7:0)	Output of the 1st band
ING1(7:0)	Output of the 2nd band

ING2(7:0)	Output of the 3rd band
ING3(7:0)	Output of the 4th band

<u>IN SYNTHESIS PART (for Figure 4.2a-c)</u>

ING0(7:0)	Input of the 1st band
ING1(7:0)	Input of the 2nd band
ING2(7:0)	Input of the 3rd band
ING3(7:0)	Input of the 4th band
X(7:0)	Mixed inputs, fed to the FIR filter
S12BUS(7:0)	Coefficient input to the 1st tap of FIR filter
S10BUS(7:0)	Coefficient input to the 2nd tap of FIR filter
S8BUS(7:0)	Coefficient input to the 3rd tap of FIR filter
S6BUS(7:0)	Coefficient input to the 4th tap of FIR filter
S4BUS(7:0)	Coefficient input to the 5th tap of FIR filter
S2BUS(7:0)	Coefficient input to the 6th tap of FIR filter
S1BUS(7:0)	Coefficient input to the 7th tap of FIR filter
S14BUS(7:0)	Coefficient input to the 8th tap of FIR filter
S7BUS(7:0)	Data input to the 1st tap of FIR filter
S15BUS(7:0)	Data input to the 2nd tap of FIR filter
S16BUS(7:0)	Data input to the 3rd tap of FIR filter
S9BUS(7:0)	Data input to the 4th tap of FIR filter
S11BUS(7:0)	Data input to the 5th tap of FIR filter
S3BUS(7:0)	Data input to the 6th tap of FIR filter
S5BUS(7:0)	Data input to the 7th tap of FIR filter
S0BUS(7:0)	Data input to the 8th tap of FIR filter
sdout(15:0)	FIR filter output (16 bit)
RCSTR_OUT	Reconstructed output

1. INTRODUCTION

Wavelets are functions, like sines and cosines, that satisfy certain requirements. These functions may be used as basis functions in the representation of an arbitrary function, in a manner similar to Fourier analysis which uses sines and cosines as basis functions. We use this "wavelet transform" to represent certain functions which do not respond well to traditional Fourier analysis. For example, functions having sharp spikes or discontinuities will require a large number of Fourier basis functions if they are to be accurately represented. In contrast, if a wavelet basis is utilized, significantly fewer terms are required. Wavelets, like sinusoids, integrate to zero, but wavelets are localized in both frequency and time, whereas Fourier basis functions are localized only in frequency. This additional localization property of wavelets may be used to the advantage of those who are faced with problems involving large and noisy data sets. Such data sets can be transformed using wavelets so that the data is preserved in the form of the wavelet basis coefficients. Processing may then be carried out on the encoded data sets. The advantage of this approach is that it is much less computationally expensive than Fourier analysis.

In addition to the advantages of wavelets, recently, Perfect Reconstruction Quadrature Mirror Filter's (PR-QMF) have gained popularity in applications that involve multiresolution signal decomposition. Most significant ones have been in subband coding of digitized speech, still images and video signals for a variety of purposes such as encoding, efficient transmission, compression, and detection .

Especially in the last few years, the relationship between subband coders and the wavelet transform has been discovered. Mallat has shown the multiresolution form of the orthonormal wavelet transform is functionally equivalent to the analysis section of a subband coder with PR property [1].

For some applications it is desirable to see the wavelet transform as a signal decomposition onto a set of basis functions. In fact, basis functions called wavelets always underlie the wavelet analysis. They are obtained from a single prototype wavelet by dilations and contractions (scalings) as well as shifts. The prototype wavelet can be thought of as a bandpass filter and constant-Q property of the other bandpass filters (wavelets) follows because they are scaled versions of the prototype [2].

In traditional block coders, lengths of filters are taken to be the same as the length of each block (N=M). From multiresolution point of view, this limitation makes it difficult to achieve good time-frequency localization. An overlapping block transform termed the Lapped Orthogonal Transform (LOT) has been proposed by Cassereau [3]. Blocking effects at low bit rates are reduced through the use of basis functions that overlap in adjacent blocks with $N=2M$. The existence of PR-QMF solutions for two-band coders was shown by Smith and Barnwell [4]. In two band architectures, there are two types of filters; one is low-pass, the other is high-pass. However, in M-Band, the first filter is low-pass, the last one is high-pass and the others are band-pass filters.

Orthonormal M-channel wavelet bases can be obtained by iterating on the impulse response coefficients of the subband filters. Regularity of the basis functions depends on the locations of zeros in the low-pass branch transfer function and thus affects the design process [1].

Most implementations of M-band coders use Finite Impulse Response (FIR) filters. Although structures utilizing Infinite Impulse Response (IIR) filters have also been proposed for this purpose, stability of these proved to be difficult to achieve.

In this design, only one single FIR filter is used as a replacement for four filters representing the M=4 case by using the shuffling operation. Shuffling operation is performed by a group of switching circuits, which send one of four coefficients to the taps of the FIR filter at each clock, representing each filter for one clock cycle. This kind of shuffling operation results in downsampling by 4 automatically. A similar shuffling operation can make it possible for us to realize the synthesis part with PR property with only one FIR filter instead of four.

These single filter operations are also easy to be represented in C code. After shuffling operations being successful in C with real number filter coefficients, quantization is done for coefficients to be used in digital integer FIR filter.

The design is made via VHDL codes, with a hierarchical bottom-up methodology. After synthesis and optimization of the design, the whole circuit is mapped onto the ES2 (an IC vendor affiliate of the EuroPractice consortium) 1μm CMOS target technology. Therefore, standard cells used in the subcircuits of the design are from the ECPD10, 1μm. CMOS standard cell library of the ES2 foundry.

2. BACKGROUND

This thesis is on the VLSI design of a wavelet transformer. Wavelet transformation is a kind of subband coding. The main goal in subband coding is the compression and is widely used for image or speech data transfer. General information about some special headlines, concerning these subjects will be given in this chapter.

2.1. Wavelets

Fourier series, or expansion of periodic functions in terms of harmonic sines and cosines, date back to the early part of the 19th century when Fourier proposed harmonic trigonometric series [5]. The first wavelet (the only example for a long time!) was found by Haar early in this century. However the construction of more general wavelets to form bases for square-integrable functions was investigated in the 1980's, along with efficient algorithms to compute the expansion.

While linear expansions of functions are a classic subject, the recent constructions contain interesting new features. For example, wavelets allow good resolution in time and frequency, and should thus allow one to see "the forest and the trees". This feature is important for non-stationary signal analysis. While Fourier basis functions are given in closed form, many wavelets can only be obtained through a computational procedure (and even then, only at specific rational points). While this might seem to be a drawback, it turns out that if one is interested in implementing a signal expansion on real data, then a computational procedure is better than a closed-form expression!

If we look deeper to this non-stationary signal analysis subject [2]; the aim of the signal analysis is to extract relevant information from a signal by transforming it. Some methods make a priori assumptions on the signal to be analyzed; this may yield sharp results if these assumptions are valid, but is obviously not of general applicability. In this paper, we focus on methods that are applicable to any general signal. In addition, we consider invertible transformations. The analysis thus unambiguously represents the signal, and more involved operations such as parameter estimation, coding and pattern recognition can be performed on the "transform side," where relevant properties may be more evident.

Such transforms have been applied to stationary signals; that is, signals whose properties do not evolve in time. For such signals x(t), the natural "stationary transform" is the well-known Fourier transform.

$$X(f) = {}^{\cdot}\; x(t)\; e^{-2j\pi f t}\; dt \qquad (2.1)$$

The analysis coefficients X(f) define the notion of global frequency f in a signal. As shown in Formula 2.1, they are computed as inner products of the signal with sinewave basis functions of infinite duration. As a result, Fourier analysis works well if $x(t)$ is composed of a few stationary components (e.g. sinewaves). However, any abrupt change in time in a non-stationary signal $x(t)$ is spread out over the whole frequency axis in $X(f)$. Therefore, an analysis adapted to *non-stationary* signals require more than the Fourier Transform.

The usual approach is to introduce time dependency in the Fourier analysis while preserving linearity. The idea is to introduce a "local frequency" parameter (local in time) so that the "local" Fourier Transform looks at the signal through a window over which the signal is approximately stationary. Another equivalent way is to modify the sinewave basis functions which are more concentrated in time (but less concentrated in frequency).

The recent surge of interest in the types of expansions discussed here is due to the convergence of ideas from several different fields, and the recognition that techniques developed independently in these fields could be cast into a common framework.

The name "wavelet" had been used with its current meaning first by J. Goupillaud, J. Morlet and A. Grossman [6]. In the context of geophysical signal processing they investigated an alternative to local Fourier analysis based on a single prototype function, and its scales and shifts. The modulation by complex exponential in the Fourier transform is replaced by a scaling operation, and the notion of scale replaces that of frequency. The simplicity and elegance of the wavelet scheme was appealing and mathematicians started studying wavelet analysis as an alternative to Fourier analysis. This led to the discovery of wavelets which form orthonormal bases for square-integrable and other function spaces by Meyer [7], Daubechies [8], Battle [9], Lemarié [10], and others. A formalization of such constructions by Mallat and Meyer created a framework for wavelet expansions called multiresolution analysis, and established links with methods used in other fields. Also, the wavelet construction by Daubechies [11] is closely connected to filter bank methods used in digital signal processing as we shall see.

Of course, these achievements were preceded by a long-term evolution from the 1910 Haar [12] wavelet (which, of course, was not called a wavelet back then) to work using octave division of the Fourier spectrum (Littlewood-Paley [13]) and results in harmonic analysis (Calderon-Zygmund operators [14]). Other constructions were not recognized as leading to wavelets initially.

Paralleling the advances in pure and applied mathematics were those in signal processing, but in the context of discrete-time signals. Driven by applications such as speed and image compression, a method called subband coding was proposed by Croisier, Esteban, and Galand [15] using a special class of filters called quadrature mirror filters (QMF) in mid 1970's, and by Crochiere, Webber and Flanagan [16]. This led to the study of perfect construction filter banks, a problem solved in the 1980's by several people, including Smith and Barnwell [17], Mintzer [18], Vetterli[19], and Vaidyanathan[20].

In a particular configuration, namely when the filter bank has octave bands, one obtains a discrete-time wavelet series. Such a configuration has been popular in signal processing less for its mathematical properties than because an octave band or logarithmic spectrum is more natural for certain applications such as audio compression since it emulates the hearing process. Such an octave-band filter bank can be used, under certain conditions, to generate wavelet bases, as shown by Daubechies [8].

In computer vision, multiresolution techniques have been used for various problems, ranging from motion estimation to object recognition. Images are successively approximated starting from a coarse version and going to a fine-resolution version. In particular, Burt and Adelson proposed such a scheme for image coding in the early 1980's, calling it pyramid coding [21]. The importance of the pyramid algorithm was not immediately recognized. One of the reviewers of the original Burt and Adelson paper said, "I suspect that no one will ever use this algorithm again" [5]. This method turns out to be similar to subband coding. Moreover, the successive approximation view is similar to the multiresolution framework used in the analysis of wavelet schemes.

In computer graphics, a method called successive refinement iteratively interpolated curves or surfaces, and the study of such interpolators is related to wavelet constructions from filter banks.

Finally, many computational procedures use the concept of successive approximation, sometimes alternating between fine and coarse resolutions. The multigrid methods used for the solution of partial differential equations are an example.

2.2. Image Compression And Encoding Techniques

Image compression techniques basically involve the processing of the image prior to transmission or archiving [22]. The image data is then transmitted (or stored, in the archival application) and decoded, decompressed, or reconstructed prior to use. The heart of any of the image compression techniques centers on two entities:

1. The development of an image representation that removes a significant amount of the inherent redundancy in the image data. From a statistical viewpoint, we seek a transformation of the image data such that the transformed image consists, ideally, of uncorrelated data.

2. The achievement of a reconstruction scheme that 'undoes' the compression or encoding scheme. Most important, this reconstruction scheme, together with the chosen compression technique, is chosen to minimize subjective distortion in the resulting image.

2.2.1. Image Data Requirements And Algorithm Performance Measures

As a preliminary example, consider an image with 512 x 512 pixel spatial resolution and 8 bits (256 levels) intensity resolution. This represents 0.25 Mbyte (1M byte = (1 K byte)2; 1 K = 1024) of image data. At what is sometimes referred to as "real-time" rates (i.e., the R-170 frame rate) this amount of data over time represents a data rate of almost 63 million bits per second. Compression techniques seek to reduce this data rate.

Measures of compression algorithm performance are basically composed of three entities:

1. A quantitative measure of the amount of data reduction expressed in terms of memory bits per image or bits per pixel (e.g., reduced to 1 bit/pixel, etc.).

2. A quantitative or qualitative assessment of the degradation (if any) of the image data.

3. A measure of the algorithm complexity, particularly with respect to compression/expansion processing speed.

2.2.2. Elementary Compression Approaches

An obvious approach for the reduction of image data is to undersample the image. The implicit assumption in subsampling is that adjacent lines or adjacent pixels contain information that is so highly correlated it may be assumed to be replicated and that the replication of this data during the reconstruction phase results in an image that is, subjectively speaking, close to original. Subsampling methods exist in several forms; the most popular is to sample one of the image frame fields and discard the other. Most home videocassette recorders (VCRs) employ this technique.

2.2.3. Transform Approaches

2.2.3.1. Block Transform

If the majority of the image content could be represented using relatively few of (some) transform basis functions, the image could be transmitted or archived in transformed form, with a significant data reduction. This type of approach was considered in Chapter 3. Other approaches partition the image into smaller regions, or blocks, and encode these blocks of local data. Block coding of data usually results in greater success, since the likelihood of a small block containing highly correlated data is probably greater in a local region than over the entire image. [22]

2.2.3.2. Karhunen-Loeve Transform

Another popular transform is known as the Karhunen-Loeve (KL) principal component, or Hotelling transform. In contrast with the deterministic approaches treated previously, the KL transform is based on the statistical characterization of the image data. The KL transformation is based on representation of a sampled image function as a vector and statistical characterization of his vector to determine ("principal") components which represent most of the image intensity variation. [22]

2.2.3.3. Wavelet Transform

Perhaps the biggest potential of wavelets has been claimed for signal compression [2]. Since discrete wavelet transforms are essentially subband coding systems, and since subband coders have been successful in speech and image compression, it is clear that wavelets will find immediate application in compression problems. The only difference with traditional subband coders is the fact that filters are designed to be regular (that is, they have many zeroes at $z=0$ or $z=$). Note that although classical subband filters are not regular , they have been designed to have good stopbands and thus are close to being "regular", at least for the first few octaves of subband decomposition.

It is therefore clear that drastic improvements of compression will not be achieved so easily simply because wavelets are used. However, wavelets bring new ideas and insights. In this respect, the use of wavelet decompositions in connection with other techniques (like multiscale edges or vector quantization [23]) are promising compression techniques which make use of the elegant theory of wavelets.

3. ARCHITECTURE

Single filter architecture is the only way to make a VLSI design for a wavelet filter. Simple ideas make it possible to design the circuit with one filter, instead of four. Otherwise silicon area will be too large and this will increase the cost considerably. In this chapter, the architecture of the design will be explained in detail.

We can separate the architecture into two parts; analysis (coder) part; m-band subband coder structure and synthesis (decoder) part; perfect reconstruction structure. Architectures are basically, the realization of the block diagrams proposed in [24]. As in the Figure 3.1. , the structures consist of 4 filters. H_0; which stands for $\Psi 1$ in Table 3.1. is a low pass filter, H_1; which stands for $\Psi 2$, is a band pass filter with small f_c, H_2; which stands for $\Psi 3$, is also a band pass filter but with a higher f_c and H_3; which stands for $\Psi 4$, is high pass filter. The frequency responses of the filters are presented in Figure 3.2..

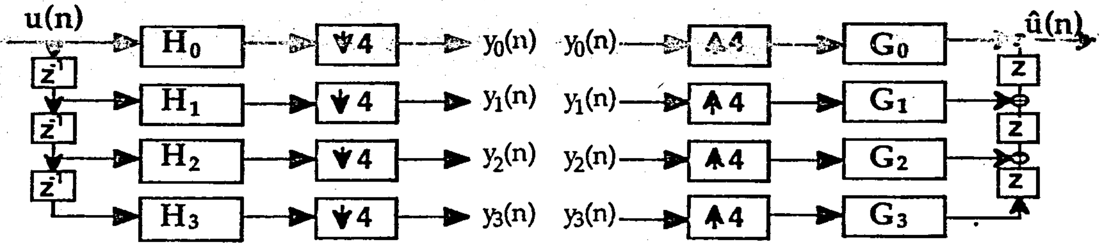


Figure 3.1. M-Band coder and decoder structures

Table 3.1. Original Analysis Filter Coefficients

n	Ψ1(v)	Ψ2(v)	Ψ3(v)	Ψ4(v)
0	-0,067371764	-0,094195111	-0,094195111	-0,067371764
1	0,094195111	0,067371764	-0,067371764	-0,094195111
2	0,40580489	0,56737176	0,56737176	0,40580489
3	0,56737176	0,40580489	-0,40580489	-0,56737176
4	0,56737176	-0,40580489	-0,40580489	0,56737176
5	0,40580489	-0,56737176	0,56737176	-0,40580489
6	0,094195111	-0,067371764	-0,067371764	0,094195111
7	-0,067371764	0,094195111	-0,094195111	0,067371764

In such a sub-band filtering application, filters acting as M-band coders must have certain properties. The analysis filter generates overlapping terms during the coding operation and the synthesis filter must be able to decode these terms. Only by this way, input of the analysis (coder) part u(n) can be obtained at the output of the synthesis (decoder) part as û(n) (Figure 3.1.); PR property. Because of that, the coefficients of the synthesis filter can be derived from the coefficients of the analysis filter [24]. Therefore the structures of the filters used either in the analysis or the synthesis are identical.

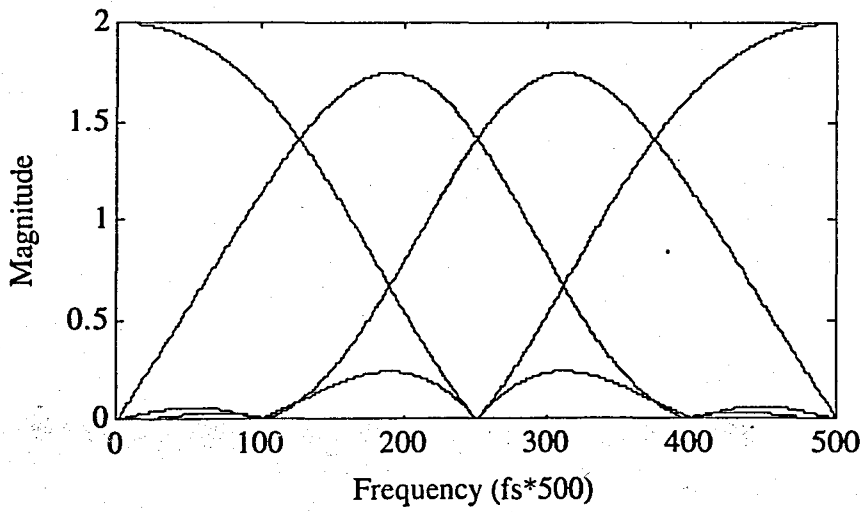


Figure 3.2. Frequency Responses of the Filters

Architecture consists of 37 modules; written in VHDL, synthesized and optimized by Mentor Graphics AutoLogic software package. These modules have a hierarchical order among themselves and hierarchy listing is presented in Figure 3.3.. In design process, we synthesized and optimized modules from very bottom to topmost level one by one. If you happen to optimize the topmost module directly, AutoLogic will optimize a lower level module as much as the number of times it's used in the circuit. Since AutoLogic does not close the optimization file of the lower level module before the optimization of all design is finished, the program needs to open another optimization file for that module during its next usage. For example; after synthesizing an FIR filter, you can face 368 synthesis versions of an AND gate. Certainly this takes a lot more time and a lot more harddisk space. VHDL listings for all of the modules used in the design is represented in Appendix A directory of floppy disk.

```
rcstr(/) / opt_s
  eight2_tap_signed_fir(I$286) / schematic
    eight_bit_signed_mul(I$31) / schematic
      dff_positive(G108) / opt_s
      eight_bit_pipelined_mul(G4) / opt_s
        dff_positive(G25) / opt_s
        four_bit_cla_adder(G5) / opt_s
          and2(G8) / opt_s
          and3(G11) / opt_s
          and4(G10) / opt_s
          and5(G14) / opt_s
          full_adder_special(GS_4) / opt_s
            and2(G3) / opt_s
            xor2(G1) / opt_s
          or2(G6) / opt_s
          or3(G9) / opt_s
          or4(G13) / opt_s
          or5(G18) / opt_s
        four_bit_positive_latch(G20) / opt_s
        radix16_cell(G1) / opt_s
          four_bit_cla_adder(G1) / opt_s
            and2(G8) / opt_s
            and3(G11) / opt_s
            and4(G10) / opt_s
            and5(G14) / opt_s
            full_adder_special(GS_4) / opt_s
              and2(G3) / opt_s
              xor2(G1) / opt_s
            or2(G6) / opt_s
            or3(G9) / opt_s
            or4(G13) / opt_s
            or5(G18) / opt_s
          four_bit_wallace_mul(G2) / opt_s
            and2(G0) / opt_s
            full_adder(G20) / opt_s
              and2(G3) / opt_s
              or2(G5) / opt_s
              xor2(G1) / opt_s
      eight_bit_twoscomp(G177) / opt_s
        dff_positive(G2) / opt_s
        four_bit_cla_adder(G10) / opt_s
          and2(G8) / opt_s
          and3(G11) / opt_s
          and4(G10) / opt_s
          and5(G14) / opt_s
          full_adder_special(GS_4) / opt_s
            and2(G3) / opt_s
            xor2(G1) / opt_s
          or2(G6) / opt_s
          or3(G9) / opt_s
          or4(G13) / opt_s
          or5(G18) / opt_s
        four_bit_positive_latch(G1) / opt_s
        twobyone_mux(G35) / opt_2_s
        xor2(G6) / opt_s
      sixteen_bit_twoscomp(G14) / opt_s
        sixteen_bit_cla_adder(G18) / opt_s
          and2(G8) / opt_s
          four_bit_cla_adder(G1) / opt_s
            and2(G8) / opt_s
            and3(G11) / opt_s
            and4(G10) / opt_s
            and5(G14) / opt_s
            full_adder_special(GS_4) / opt_s
              and2(G3) / opt_s
              xor2(G1) / opt_s
            or2(G6) / opt_s
            or3(G9) / opt_s
            or4(G13) / opt_s
            or5(G18) / opt_s
          four_bit_mux(G15) / opt_s
          or2(G11) / opt_s
        sixteen_bit_positive_latch(G19) / opt_s
        xor2(G11) / opt_s
      xor2(I$34) / opt_s
    four_bit_positive_latch(G33) / opt_s
    sixteen_bit_cla_adder(G26) / opt_s
      and2(G8) / opt_s
      four_bit_cla_adder(G1) / opt_s
        and2(G8) / opt_s
        and3(G11) / opt_s
        and4(G10) / opt_s
        and5(G14) / opt_s
        full_adder_special(GS_4) / opt_s
          and2(G3) / opt_s
          xor2(G1) / opt_s
        or2(G6) / opt_s
        or3(G9) / opt_s
        or4(G13) / opt_s
        or5(G18) / opt_s
      four_bit_mux(G15) / opt_s
      or2(G11) / opt_s
    sixteen_bit_positive_latch(G12) / opt_s
  eight_bit_positive_latch(G14) / opt_s
  m_band_switch_1(I$1912) / opt_s
    eight_bit_4in_or(G7) / opt_s
    eight_bit_tgate_2sel(G3) / opt_s
    m_band_dff_positive(G1) / opt_s
```

Figure 3.3. Hierarchy Listing

3.1. Coefficients and Quantization

As represented in Table 3.1., original filter coefficients consist of the combination of 8 real numbers. To use them in this design, we have to quantize them. Quantization for integer FIR filters is usually done by rounding the number [25] which is calculated by Formula 3.1 where $coef_{max}$ is the maximum

$$coef_{quant} = ((2^{b-1})-1) * (coef_{orig}) / (coef_{max}) \qquad (3.1)$$

value among the coefficients, $coef_{orig}$ is the original coefficient to be quantized, b is the number of bits to quantize and $coef_{quant}$ is the resulting quantized coefficient. We wrote a C code for deciding the value of b (Appendix B). In this program, first we quantize both the synthesis and analysis part coefficients and the channel between these parts by the same number of bits. Then analyze the difference between the original input to the analysis part and the output of the synthesis part and calculate the RMS and percentage errors [26]. We repeat this procedure for numbers between 5 and 20. Calculated values are analyzed and represented in Figure 3.4..

In Figure 3.4., the lines represent the percentage errors for the whole design (from analysis part to synthesis part) for different bit numbers of quantization; the solid line, close to the bottom, represents the errors for no compression case. In this case, all four channels are used in the reconstruction. The dashed line, in the middle, represents the percentage error generated when 25% compression is applied (high-pass filter is omitted). And as the last case, 50% compression is applied (high-pass filter and band-pass filter with higher frequency response are omitted) and the error for this case is represented in small dashed line, at the top.

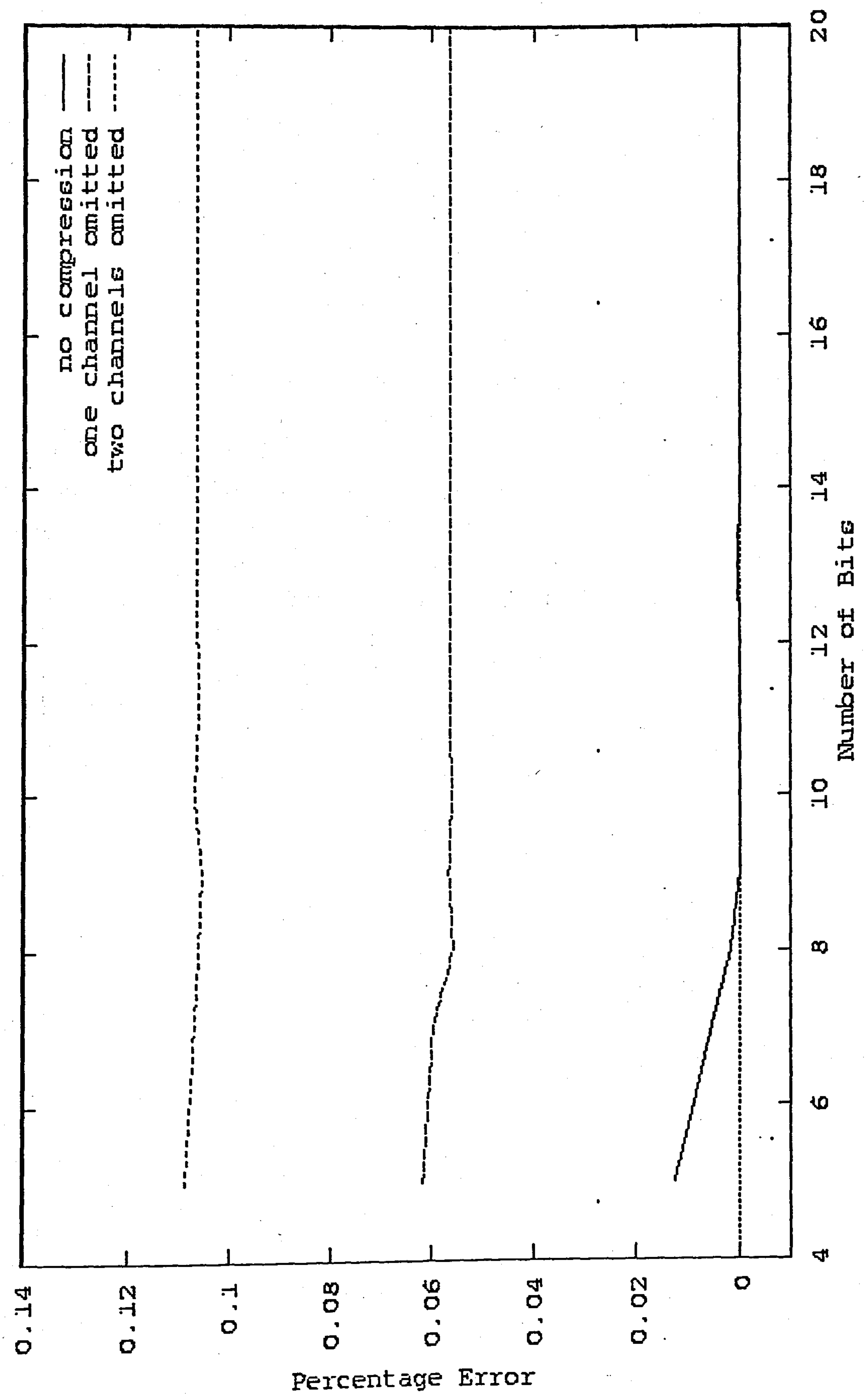


Figure 3.4. Quantization Errors for Different Bit Numbers

The aim of proper operation with less count of transistors drove us to implement the design with the minimum number of bits. So that, we decided to take *b* as 8. As it can be seen from the graph in Figure 3.4., at 50% compression in 8 bit case, the percentage error does not exceed the value of 10.5%. This error value is really comparable for such a case.

$$C = (2^{b-1})-1 \,/\, (coef_{max}) \tag{3.2a}$$

$$C = (2^{7})-1/\,0.56737176 \tag{3.2b}$$

$$C = 223.839 \tag{3.2c}$$

After quantization, each coefficient is multiplied by a constant number (3.2c). However, the output has to be scaled by this number at the end. If this number is not a power of two, this scaling can be problematic and we cannot get the same input, but a scaled one, after the PR stage. Therefore, the number is chosen as the closest power of two by using the Formula 3.3a-d, so that scaling can be done only by a shift right by 7 operation. The coefficients quantized by this method are represented in Table 3.2. and Table 3.3. and are used in the design.

$$C = (2^{b-1})-1 \,/\, (qntzr_{new}) \tag{3.3a}$$

$$qntzr_{new} \quad > \quad coef_{max} \tag{3.3b}$$

$$for\ C=256 \quad => \quad qntzr_{new} \;=\; 0.49609\ (not\ acceptable) \tag{3.3c}$$

$$for\ C=128 \quad => \quad qntzr_{new} \;=\; 0.99218\ (acceptable) \tag{3.3d}$$

Table 3.2. Analysis Filter

n	Ψ1(n)	Ψ2(v)	Ψ3(n)	Ψ4(v)
0	F7	F4	F4	F7
1	0C	09	F7	F4
2	34	49	49	34
3	49	34	CC	B7
4	49	CC	CC	49
5	34	B7	49	CC
6	0C	F7	F7	0C
7	F7	0C	F4	09

Table 3.3. Synthesis Filter

n	Ψ1(n)	Ψ2(v)	Ψ3(n)	Ψ4(v)
0	F7	0C	F4	09
1	0C	F7	F7	0C
2	34	B7	49	CC
3	49	CC	CC	49
4	49	34	CC	B7
5	34	49	49	34
6	0C	09	F7	F4
7	F7	F4	F4	F7

3.2. Filter Structure

With an FIR filter structure like the one in Figure 3.5., inputs should wait for the output belongs to the previous input. This structure slows down the operation speed of all the system and so FIR filter becomes the bottleneck of the system. In this design, the structure shown in Figure 3.6.; which depicts the pipelining-scheme, is used. In this structure inputs do not need to wait output of the previous input. All operations are being done while the inputs are flowing through the pipeline. The pipeline depth of the filter should be so that it runs properly with clock frequency of the circuit. Therefore, the FIR filter is not the system speed defining element anymore, but the multipliers are.

Since the circuit is designed for 8 bit operation, Radix-2^n multipliers [27] performing 8x8 signed multiplication are used. Although single clock is used throughout the design, an inverted (with 180° phase shift) clock is used in the multipliers to avoid setup and hold violations at data read times. These multipliers have a pipeline depth of 8 and they are capable of running in excess of 33 MHz.

The adders are 16 bit carry look ahead adders. Finally, eight and sixteen bit D-flip flops with reset are used as delay elements. Both of these elements introduce an additional pipeline depth of one clock pulse to the circuit.

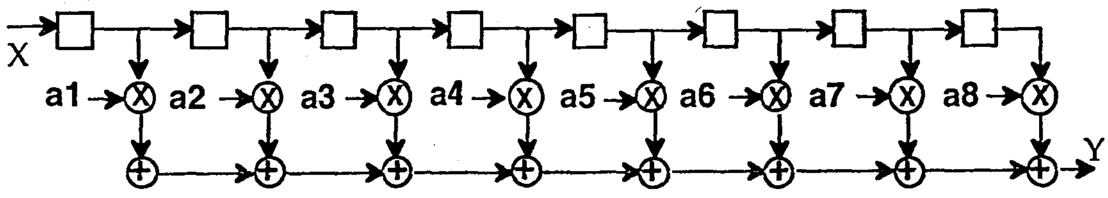


Figure 3.5. Commonly Used FIR Filter Structure

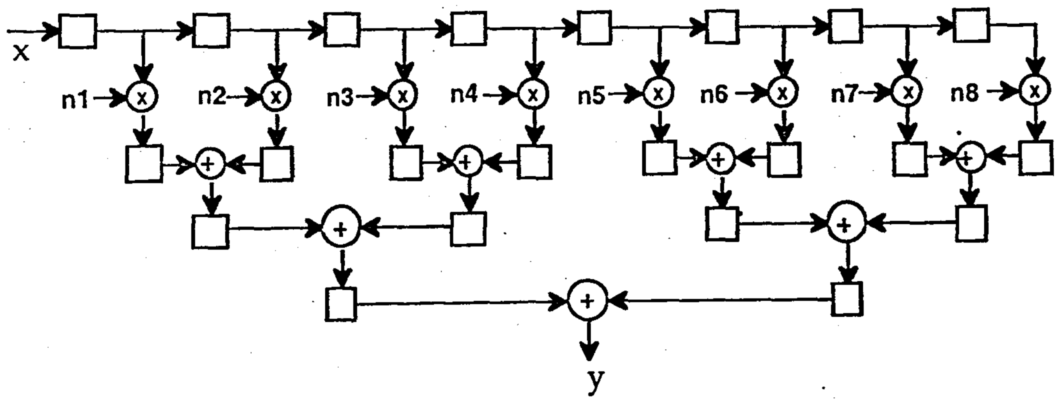


Figure 3.6. FIR Filter Used in this Design

3.2.1. Analysis Filter

According to the actual algorithm, inputs are fed to the filters with a delay sequentially (multiplying by Z^{-1}) (Figure 3.1.). After filtering, downsampling by 4 operation is performed for the outputs. This makes the filtering operations useless by 3 in 4 time. Actually, filter rests for 3 cycles in a 4 cycle operation and this situation is valid for all filters (again) with a cycle delay sequentially. This means, at one cycle only one of the filters can have a valid output. Therefore, operation of only that filter can be performed while the others are resting (performing downsampling). This situation is led us to accomplish the whole operation with a single filter, instead of four.

The analysis part is designed with a single FIR filter as in the architecture in Figure 3.7., representing the coder structure in Figure 3.1. [28]. The input signals are fed to the filter, through a D-flip flop which controls the flow of inputs.

The switching circuit in Figure 3.7., designed with four transmission gates and two D-flip flops as shown in Figure 3.8., sends a different set of coefficients of filters on each clock cycle to make the filter acts as four filters. Consequently, with this technique, the chip area is reduced by a factor of 4. All of these shuffling operations, being done by the switching circuit in Figure 3.7. will be described in detail in the next subsection.

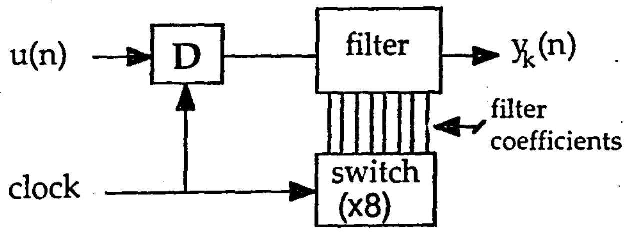


Figure 3.7. Architecture with single filter

The outputs can be separated by knowing that the first output belongs to the first FIR filter and others follow it sequentially. Therefore, we can apply compression by omitting some of the channels before transmission; 25% by omitting the last high pass filter outputs, 50% by omitting the last two channels (high pass and band pass).

3.2.1.1. Switching Circuit

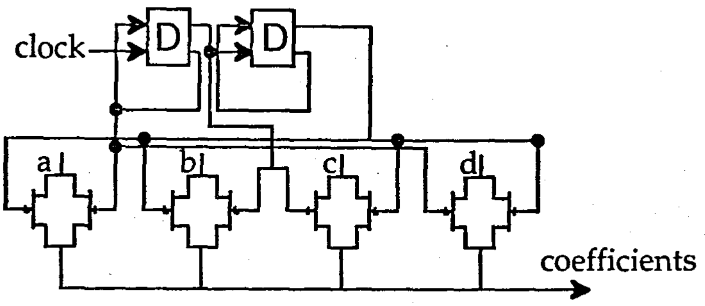


Figure 3.8. Switching Circuit

As it has been introduced before, switching circuit consists of four 8-bit transmission gates and two 8-bit D-flip flops. Actually, the function of this circuit has the greatest importance among all submodules. Switching circuit perfoms the synchronization of inputs and coefficients to meet each other properly.

The design has 8 switching circuits (one for each filter tap). Two D-flip flops runs as counter for transmission gates, such that at each clock pulse, one of four coefficients (a, b, c, d) for each band are sent to the FIR filter. Logic table for transmission gate selection is on the following Table 3.4..

Table 3.4. Switching Circuit Logic Table

clk	out1	$\overline{\text{out1}}$	out2	Xgate
1	1	0	1	a
0	1	0	1	a
1	0	1	1	b
0	0	1	1	b
1	1	0	0	c
0	1	0	0	c
1	0	1	0	d
0	0	1	0	d

3.2.2. Synthesis Filter

An FIR filter working as the inverse of analysis filter, like in Table 3.3., acts as the synthesis filter. As shown in the decoder part of Figure 3.1., four channel inputs coming from the analysis part are upsampled by 4 since M=4 and fed through the corresponding filters. Then the outputs of the filters are to be added to each other with a time shift backward (multiplying by Z). A similar chip area saving, like in the analysis part can be accomplished here as well; the operation procedure mentioned above, is equivalent to the case where inputs are delayed properly (for 1st filter 3 clock cycles, for 2nd 2, for 3rd 1 and for 4th 0) and filter outputs added simultaneously as shown in step 1 of the Figure 3.9.. In this representation, there are four filters.

The inputs will be processed in the filters only at 1st, 5th,...,$(4n+1)$th clock cycles where n is the number of the input. During three clock cycle periods, filters will become idle. To overcome this drawback, the delay elements in front of the filters can be neglected and coefficients can be shifted right by that amount (Step 2 of the Figure 3.9.). Now there are four filters with shifted coefficients, but at most two of the coefficients for each filter will be functional while only two of the filter taps are with data. Therefore, those functional ones at one time can be put together and form four different filters (Table 3.5.).

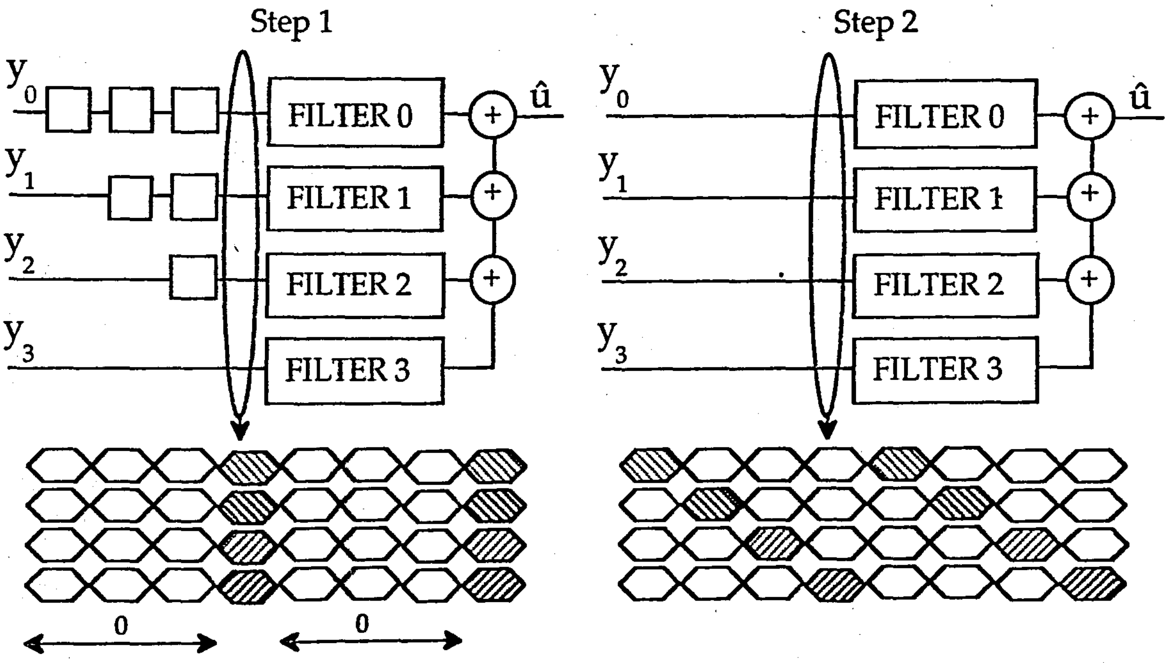


Figure - 9 Inputs to the synthesis part

Table - 5 Synthesis Filter (2)

n	$\Psi_1(n)$	$\Psi_2(n)$	$\Psi_3(n)$	$\Psi_4(n)$
0	09	0C	CC	49
1	F4	F7	49	CC
2	0C	F7	B7	CC
3	F7	0C	34	49
4	B7	34	F4	F7
5	CC	49	F7	F4
6	34	49	09	F4
7	49	34	0C	F7

1st Filter Coeff. *3 rd Filter Coeff.*

2nd Filter Coeff. 4th. Filter. Coeff.

Outputs of the analysis part can be fed through the synthesis part without separating to 4 channels. Initially, synthesis part will not respond for 4 clock cycles, till the first input of the 4th filter arrives. After that, the inputs corresponding to the taps of the filter will be held for 4 clock cycles and operated on by all the corresponding filters at each clock cycle sequentially. Here, different from other FIR filter operations, inputs are not flowing through the filter at each clock cycle, but they stay at the corresponding taps for four clock cycles while the coefficients for four different filters are shuffling at the taps of the filter. This coefficient shuffling operation is like inputs have been upsampled by 4 and flowing through the filters related to the ones they had been generated by in the analysis filter. While this process is being executed, the inputs coming from the analysis filter are also fed through the filter without touching the taps. After 4 clock cycles, corresponding inputs will be held at the taps and the similar operation will continue. As a result, upsampling by 4 is being done automatically and all reconstruction operation is being realized with a single filter.

These operations are done by the help of two switching circuits as it's been mentioned in analysis part. One of the switching circuit is in front of the input buses; to keep inputs for 4 clock cycles at the taps of the filter while actual inputs flowing through the circuit. The second one acts as the same of the analysis part switching circuits and like in the analysis part we have 8 this type of switches here as well.

4. SIMULATION

First, a C program (Appendix B and Appendix B directory of floppy disk) is written to simulate the operation of the filters in Figure 3.1. (mb.c and rc.c) with coefficients represented by real numbers. After achieving Perfect Reconstruction with real coefficients, quantization process begins. Again a C program is written to quantize the coefficients (qnt.c) and the channel between analysis and synthesis parts (channel.c) and another to calculate the value of error (calc.c). A couple of batch codes are written to run all these programs automatically for different conditions (comp10, comp11, comp15). Results for these steps are represented in section 3.1.

After C language steps, similar operations representing the originals, as mentioned earlier, are written in VHDL (Appendix A) and simulated to observe that these operations are performed properly. Logic synthesis and optimization are done using the 1μm ES2 target technology from bottom to top. After each resulting schematic circuit a simulation is done to check the functions of the module. And finally two resulting top-level schematics (analysis and synthesis parts) are simulated to verify the proper operation. In all these steps, Mentor Graphics tools (Design Architect, Autologic, Quicksim,...) and a C compiler running on a Sun Sparc2 work station are used. If we overall the steps;

- Writing and running C program
- Writing VHDL codes
- Simulating VHDL codes
- Synthysizing and optimizing VHDL codes
- Simulating resulting schematic circuit
- Creating the chip layout

In simulation of the analysis filter, first, a small part of an image is fed through the filter. A clock with 33 MHz frequency is used during the simulations. The analysis circuit gives out four channel separated outputs and a clear signal (Appendix C), just before the first output comes, to activate the synthesis filter (Figure 4.1a-b). This output is in suitable format for the input of the synthesis filter (Appendix C). The inputs and outputs of the synthesis filters are shown in Figure 4.2.a-b-c.

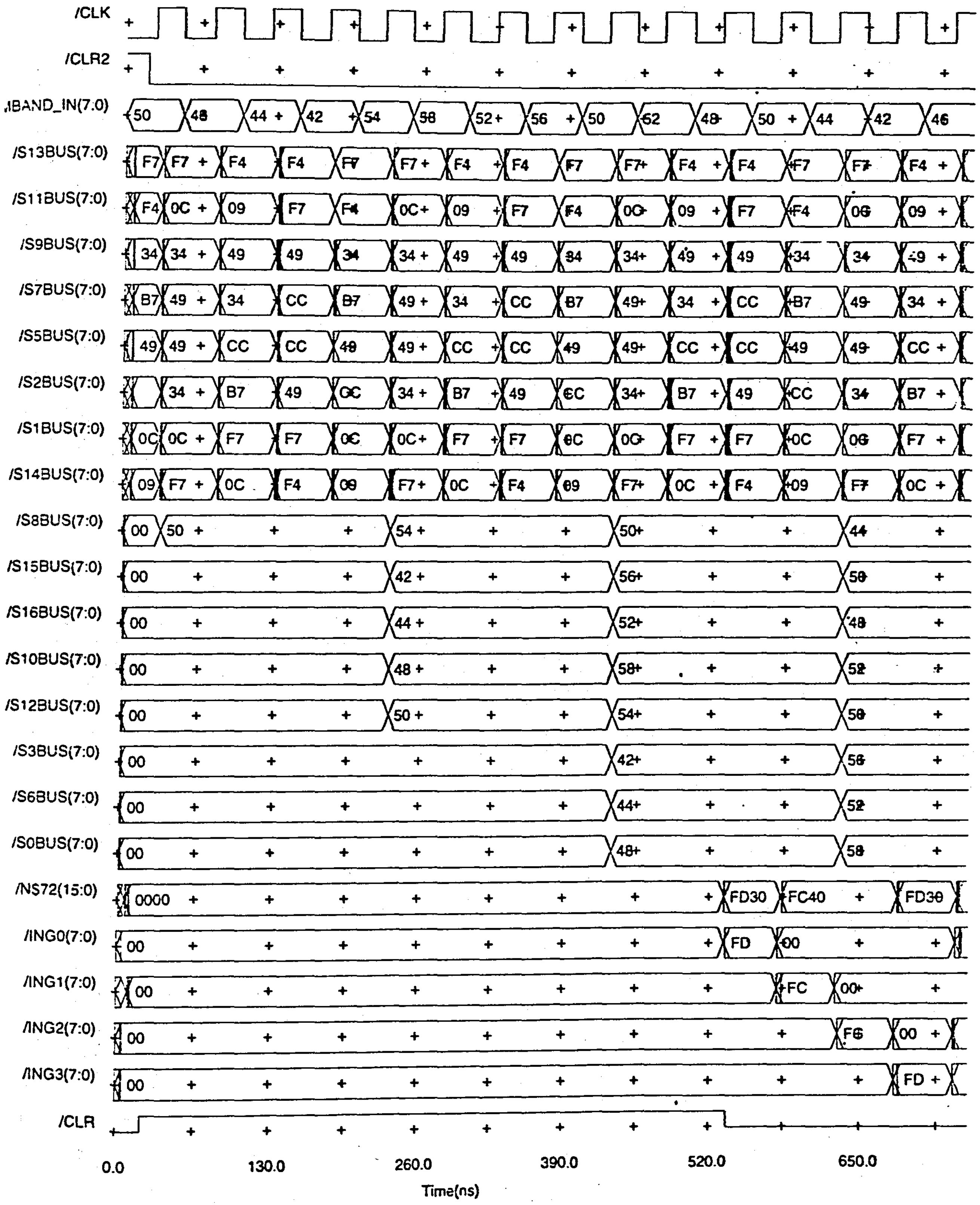


FIGURE 4.1a. Timing of theAnalysis Filter (Part 1)

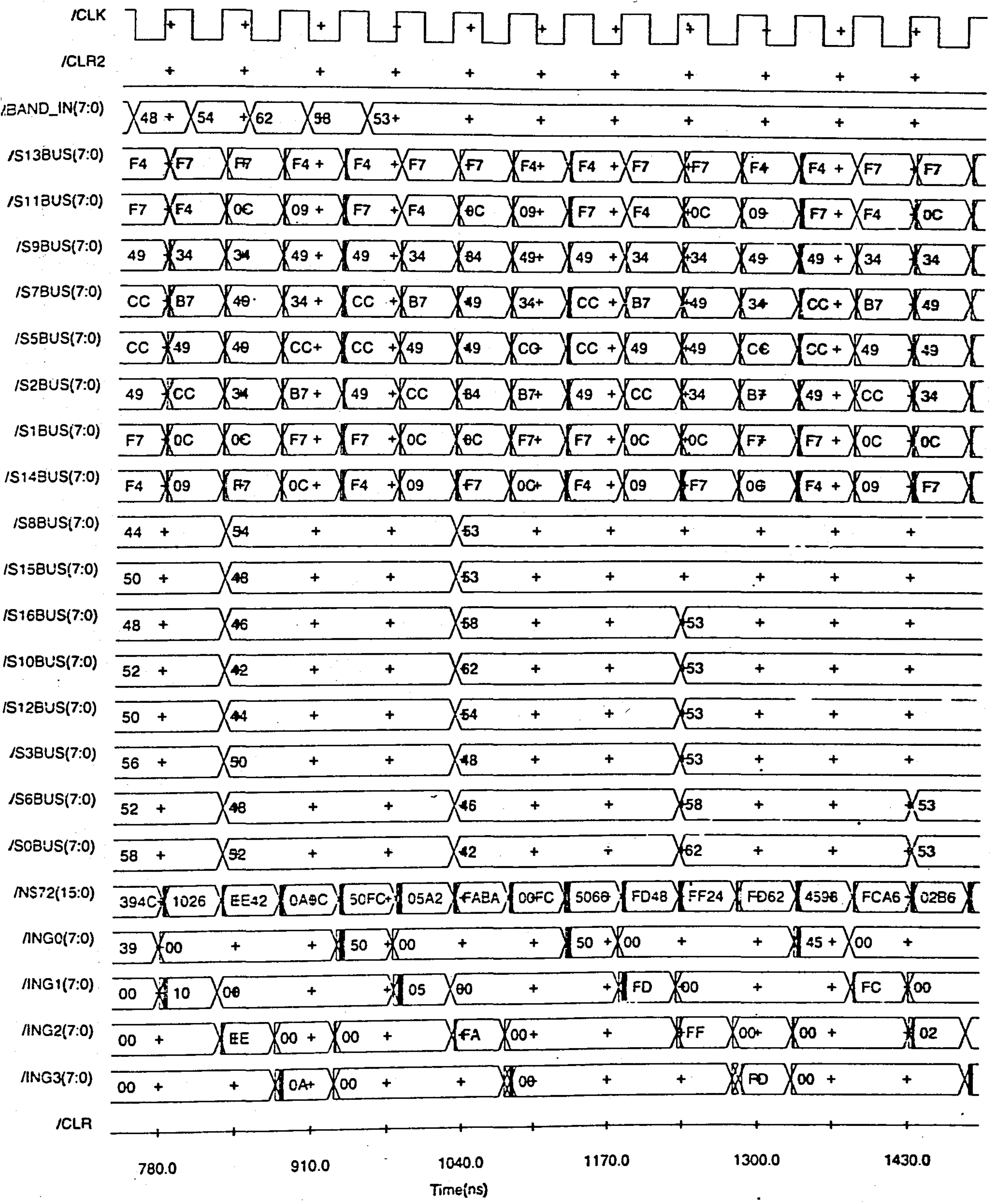


FIGURE 4.1b. Timing of the Analysis Filter (Part 2)

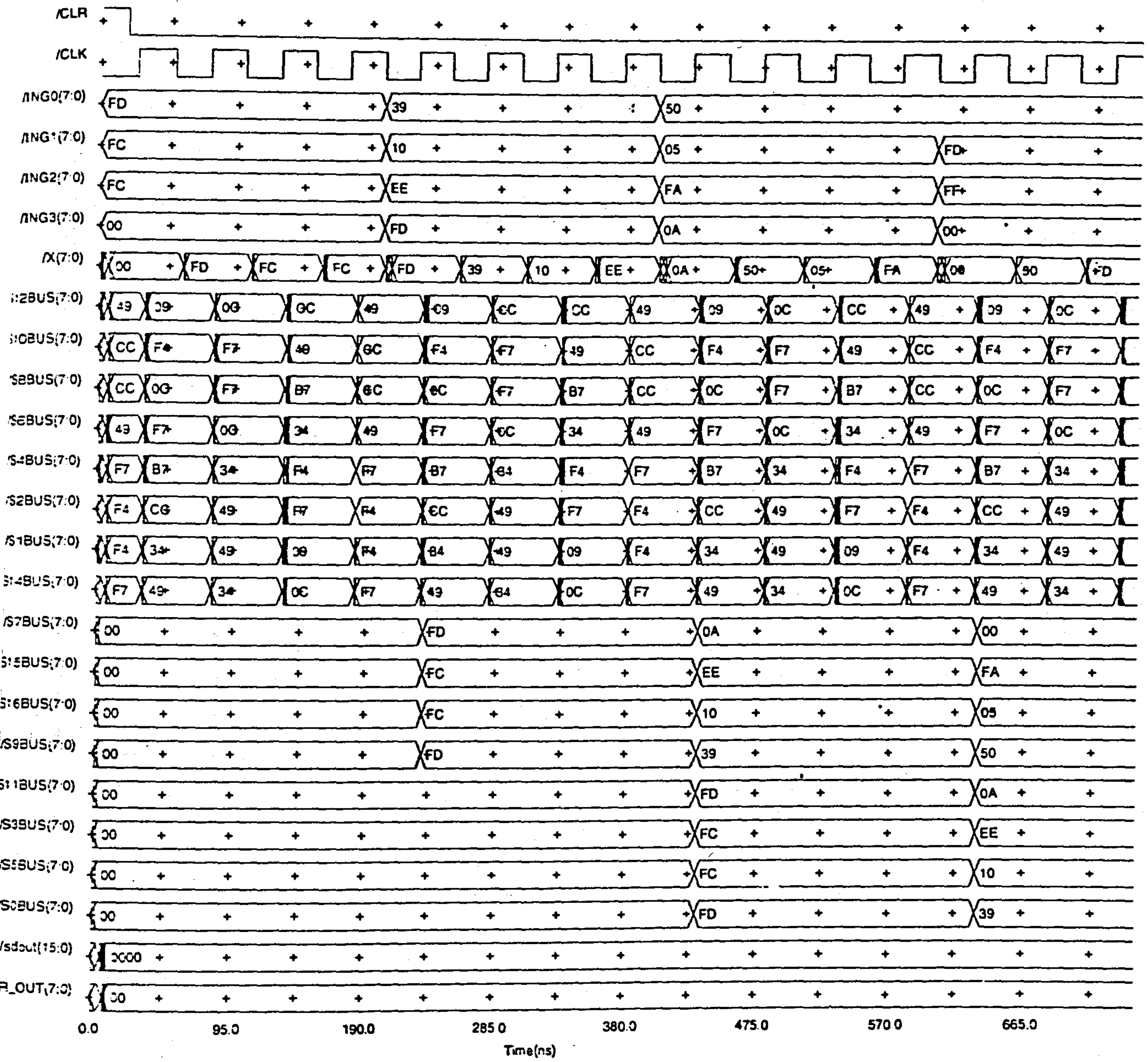


FIGURE 4.2a. Timing of the Synthesis Filter (Part 1)

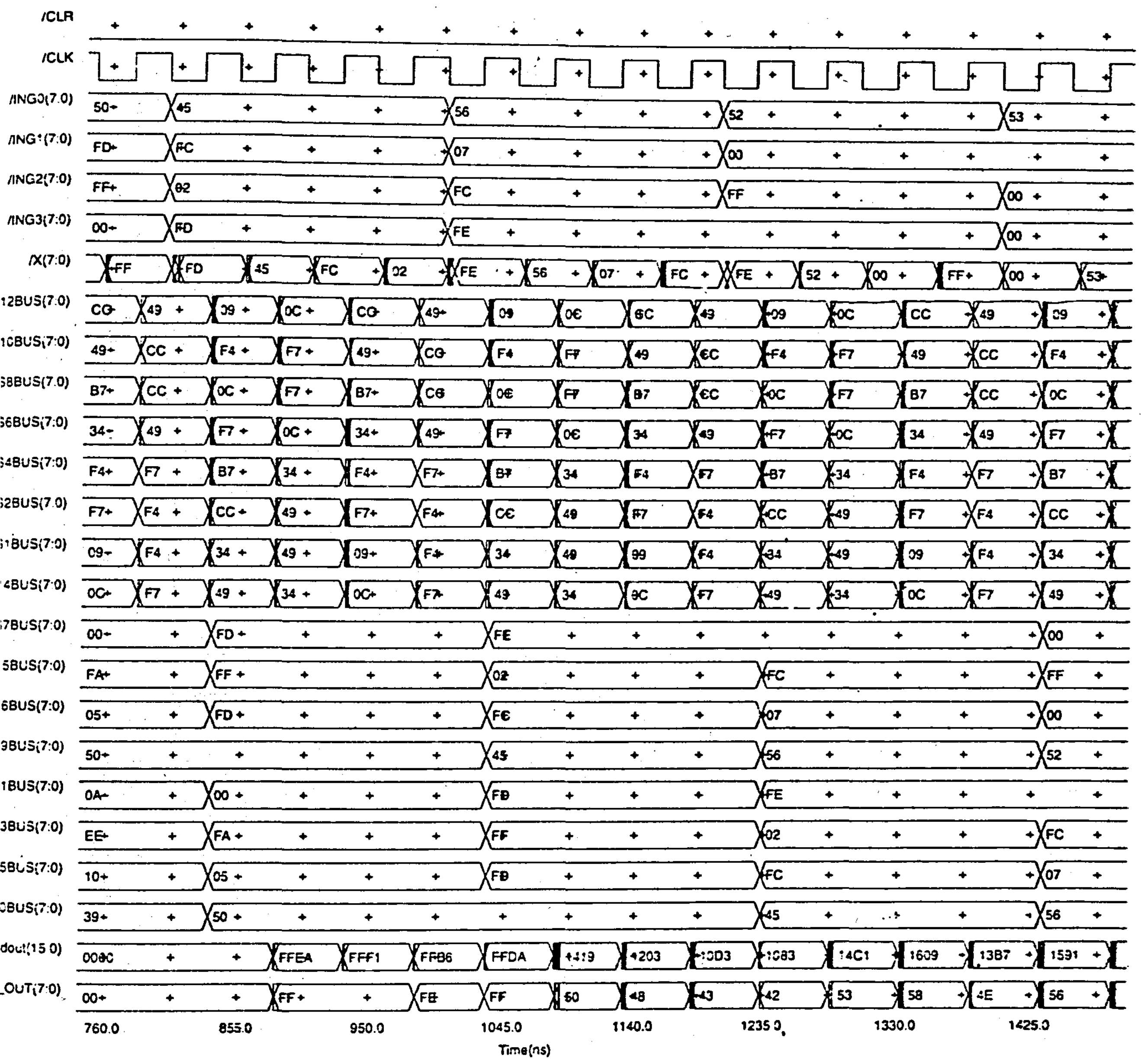


FIGURE 4.2b. Timing of the Synthesis Filter (Part 2)

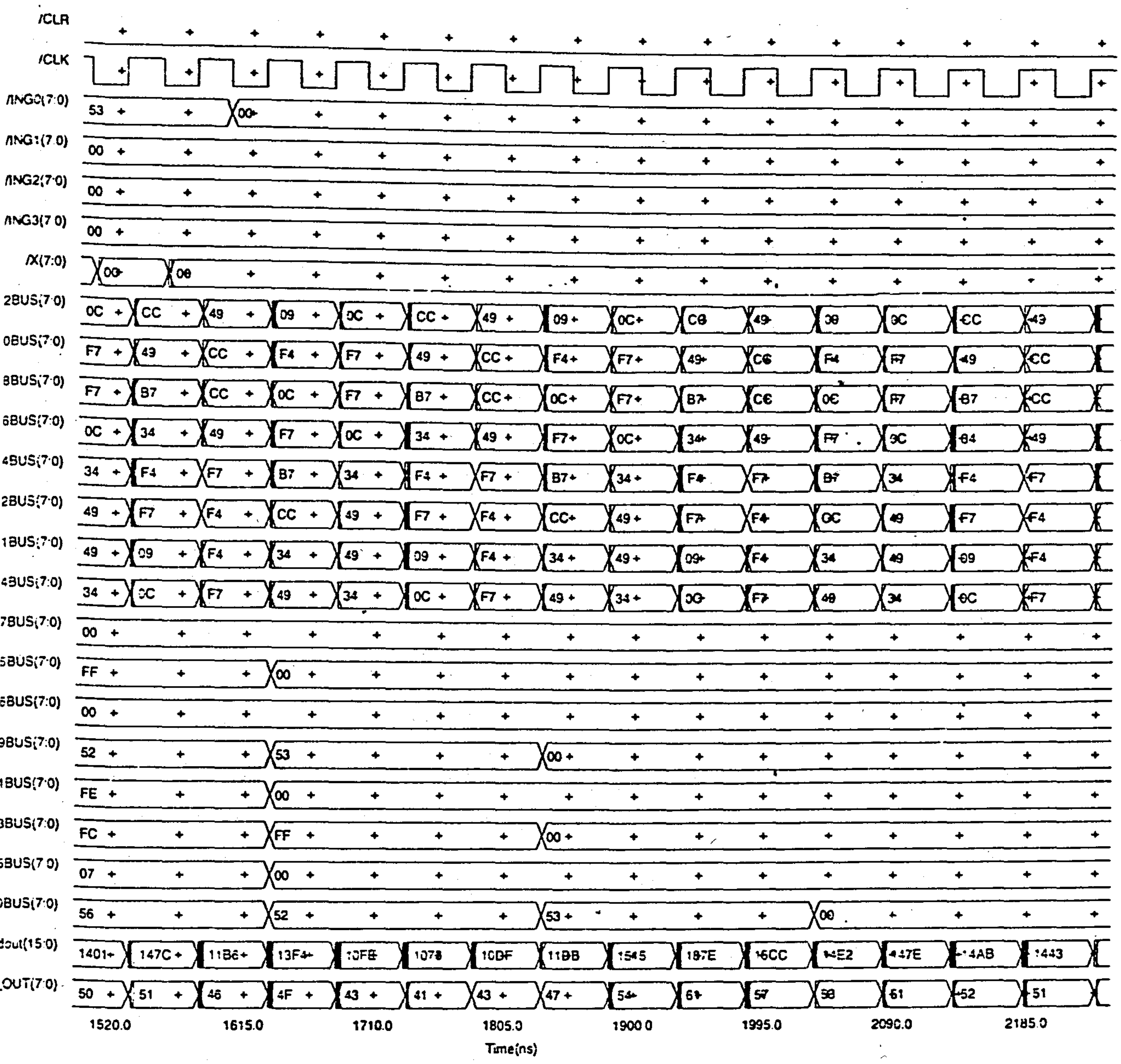


FIGURE 4.2c. Timing of the Synthesis Filter (Part 3)

In this simulations, it is observed that, if both circuits are functioning properly and perfect reconstruction can be achieved at the output of the synthesis part. After success in this step (MBAND_IN in Figure 4.1a-b vs. RCSTR_OUT in Figure 4.2a-c), a gray scale "Lenna" image (Figure 4.3.) (8-bit data for each pixel) in size of 128x128 is fed to the analysis filter as a stream, since this design is 1D (one dimension). Then, the outputs of the analysis filter are fed to the synthesis filter and the final reconstructed output is obtained as in Figure 4.4. with 6.67 RMS and 2.6% Percentage errors.

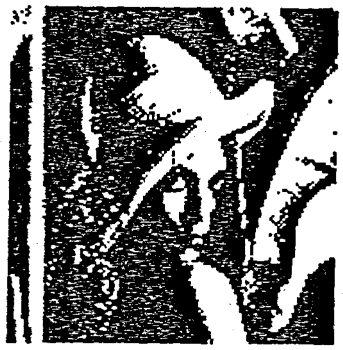


Figure 4.3. Original Lenna input image

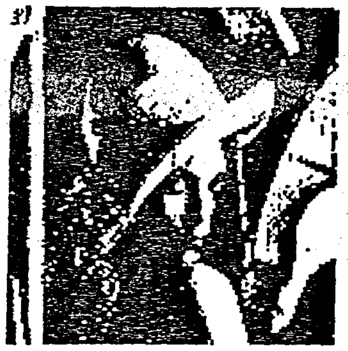


Figure 4.4. Reconstructed image

Latency of the overall system is 31 clock cycles; from the input of the analysis filter to the output of the synthesis filter. Circuits for both analysis and synthesis parts are capable of running with 33 Mhz. clock.

As mentioned in Section 3.1 quantization is done by a quantizer value of 0.99218, for having the inputs multiplied with a number in power of 2, and 128 (2^7) is found. This shows that, if we shift outputs of both analysis and synthesis parts right by 7 then we will not have a problem. However, if we shift the output of the analysis part right by 7 then we can possibly miss the sign of the resulting number. Since the input for the analysis part has no sign, the sign for analysis output stands on the MSB. Therefore, we should shift the output right by 8 for the analysis part and shift the output right by 6 for the synthesis part. In this way we may only expect for a sign missing at the output of the synthesis part.

After all these simulation steps, IC layout creation begins. Like all previous hardware steps, IC creation is done by "bottom-up" methodology, but with bigger groups. First a multiplier's, an adder's and switches' chip layouts are prepared, then the FIR filter's and finally the topmost level circuit's chip layout are created (Figure 4.5.).

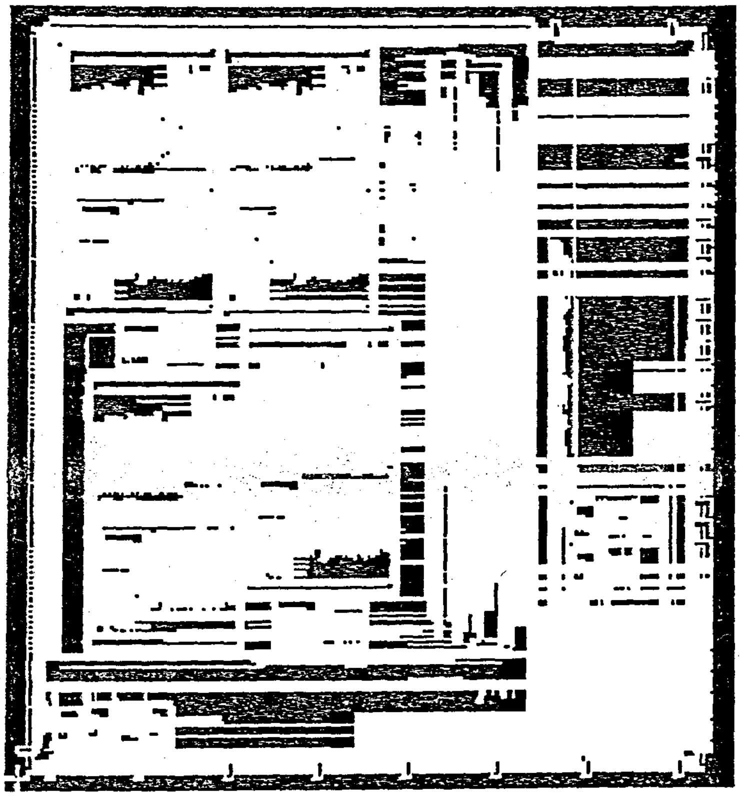


Figure 4.5. Chip Layout of Synthesis Filter

5. CONCLUSION

The biggest potential of wavelets is on applications for signal compression. Since this 1D design is prepared for speech compression, the operation speed is very high for speech or any other type of data elsewhere the image. To observe real operation error level, simulations should be done with image. Speech samples would not give an idea about real error level, because of slower variations in speech signal than image.

As it's mentioned before, even with 50% compression rate in 8 bit operation total percentage error does not exceed 0.0105 for our image sample (Figure 3.4.). Besides, this design is completely suitable for channel coding applications requiring subband coded signal [29].

Because of creating the IC in such a hierarchical way, as mentioned in Section 4, resulting chip layout appears to be bigger than the one created from a flat hierarchy. But the transistor count of the topmost circuit and the configuration of the design platform are major bottlenecks for the process with flat hieararchy. In this design transistor counts for analysis part is 145,795 and for synthesis part is 151,499. Therefore, in this design the method with tree structured hierarchy is prefered.

This design has been made for 8 bit words. Although this seems to provide adequate accuracy for speech processing applications, most of the music and image signals are represented with 16 bits. To use this design for 16 bit operation, a small change on VHDL codes and running the steps mentioned earlier will be enough. But for a good operation in 16 bit, multipliers which happen to be the bottleneck of the circuit should be changed to full custom or semi-full custom design from VHDL based ones. In 8 bit operation this bottleneck does not make that much sence.

The future work for this design will include design of 16 bit version with full custom multipliers and 2-D architectures for image compression applications.

REFERENCES

[1] S.Mallat, "A Theory For Multiresolution Signal Decomposition: The Wavelet Representation," *IEEE Trans. on Pattern Analysis and Machine Intell.* , Vol.11, No.7, pp. 674-693, July 1989.

[2] O.Rioul and M.Vetterli, " Wavelets And Signal Processing," *IEEE Singal Processing Magazine* , pp. 14-38, October 1991.

[3] P.M.Cassereau, D.H.Staelin and G.Jager, "Encodign of Images Based on Lapped Orthogonal Transform," *IEEE Trans. Comm.*, vol.37, no.2, pp. 189-193, Feb.1989.

[4] M.Smith and T.P.Barnwell, "Exact Reconstruction For Tree-Structured Subband Coders," *IEEE Trans. Acoust.,Speech, Signal Processing*, vol.34, pp. 434-441, June 1986.

[5] M.Vetterli and J.Kovacevic, "Wavelets and Subband Coding," *Prentice Hall PTR*, pp. 1-3, 1995.

[6] P. Goupillaud, A. Grossman and J. Morlet, "Cycle-Octave and Related Transforms in Seismic Signal Analysis," *Geoexploration*, Vol.23, pp. 85-102, Elsevier Science Pub., 1984.

[7] Y. Meyer, "Ondelettes et Operateurs, Tome I.," *Ondelettes*, Herrmann Ed., 1990.

[8] I. Daubechies, "Orthonormal Bases of Compactly Supported Wavelets," *Comm. in Pure and Applied Math.*, Vol.41, pp. 909-996, 1988.

[9] G. Battle, "A Block Spin Construction of Odelettes. Part I: Lemarie Functions," *Comm. Math. Phys.*, Vol.110, pp. 601-615, 1987

[10] P. G. Lemarie and Y.Meyer, "Ondelettes et bases Hilbertiennes," *Rev. Math. Iberoamericana*, Vol.2, pp. 1-18, 1986.

[11] I. Daubechies, "The Wavelet Transform, Time-Frequency Localization and Signal Analysis," *IEEE Trans. on Info. Theory*, Vol.36, No.5, pp. 961-1005, Sept. 1990.

[12] A. Haar, "Zur Theorie der Orthogonalen Funktionen-systeme," [in German] *Math. Annal.*, Vol.69, pp. 331-371, 1910.

[13] J. Littlewood and R. Paley, "Theorems on Fourier Series and Power Series," *Proc. London Math. Soc.*, Vol.42, pp. 52-89, 1937.

[14] A. Calderon, "Intermediate Spaces and Interpolation, the Complex Method," *Studia Math.*, Vol.24, pp. 113-190, 1964.

[15] A. Croisier, D. Esteban and C. Galand, "Perfect Channel Splitting by Use of Interpolation, Decimation, Tree Decomposition Techiniques," *Int. Conf. on Information Sciences/Systems*, Patras, pp. 443-446, Aug. 1976.

[16] R. E. Crochiere, S. A. Weber and J. L. Flanagan, "Digital Coding of Speech in Subbands," *Bell Syst. Tech. J.*, Vol.55, pp. 1069-1085, Oct. 1976.

[17] M. J. T. Smith and T. P. Barnwell, "Exact Reconstruction for Tree-Structured Subband Coders," *IEEE Trans. on Acoust., Speech and Signal Proc.*, Vol.34, pp. 434-441, June 1986.

[18] F. Mintzer, "Filters for Distortion-Free Two-Band Multirate Filter Banks," *IEEE Trans. on Acoust., Speech and Signal Proc.*, Vol.33, pp. 626-630, June 1985.

[19] M. Vetterli, "Filter Banks Allowing Perfect Reconstruction," *Signal Processing*, Vol.10, No.3, pp. 219-244, April 1986.

[20] P. P. Vaidyanathan, "Quadrature Mirror Filter Banks, M-Band Extensions and Perfect Reconstruction Techniques," IEEE ASSP Magazine, Vol.4, No.3, pp. 4-20, July 1987.

[21] P. J. Burt and E. H. Adelson, "The Laplacian Pyramid as a Compact Image Code," *IEEE Trans. on Com.*, Vol.31, No.4, pp. 532-540, April 1983.

[22] R.J.Schalkoff, "Digital Image Processing and Computer Vision," *John WILEY & Sons INC.*, pp. 188-189, 1989.

[23] M.Antonini, M.Barlaud, P.Mathieu and I.Daubechies, "Image Coding Using Vector Quantization in the Wavelet Transform Domain," in *Proc. of IEEE Int. Conf. Acoust. Speech Signal Proc.* Albuquerque, NM, pp. 2297-2300, Apr.3-6, 1990.

[24] O.Alkın and H.Çağlar, "Design of Efficient M-Band Coders With Linear-Phase And Perfect Reconstruction Properties," *IEEE Trans. Acoust., Speech, Signal Processing,* Vol.43, No.7, pp. 1579-1590, July 1995.

[25] P. M. Embree and B. Kimble, " C Language Codes For Digital Signal Processing ," *Prentice Hall PTR.* , pp. 157-161, 1991.

[26] G.Dundar and K.Rose, "The Effects of Quantization on Multilayer Neural Networks"

[27] M.K.İbrahim, "Radix-2n Multiplier Structures: A Structured Design Methodology," *IEE Proceedings-E,* Vol.140, No.4, pp. 185-190, 1993.

[28] Y.Atabek, G.Dündar, S.Balkır, H.Çağlar, E.Anarım "M-Bantlı Dalgacık Dönüşümlerini Gerçekleyecek Mimarilerin Tasarımı," *3.Sinyal Işleme ve Uygulamaları Kurultayı Sinyal Işleme Bildiri Kitabı (B),* pp. 286-290, 26-28 Nisan 1995 (in Turkish).

[29] M.Schilpp, A.Netter, W.Rupprecht, E.Bogenfeld "Parallel And Serial Concatenated Codes For Digital HDTV," *Proceedings of International Conference On Telecommunications ICT'96,* Vol.2, pp. 616-619, 14-17 April 1996.

APPENDIX A

VHDL

VHDL

In the first half of 1980's, MC68000 had been designed on a wall with paper and pencil. Today's increasing demand on digital IC's that are smaller in size but larger in complexity, pushes designers to use computer aided design (CAD) tools. As the size and complexity increases, drawing circuits by a CAD tool and keeping the program under control gets harder. Besides, this kind of design may take long time. At this point, thought of a higher level programing language has brought a description language into the electronic design area. In VHDL (Very high speed IC Hardware Description Language), designers describe behaviors or structures of their hardware with very simple commands (i.e. Figure Apx.A.1.), fitting the standard hardware description format (IEEE 1076). By the help of this format, designers can synthesize their VHDL codes with a VHDL Synthesis program and obtain the schematic diagram of their design. Available software for VHDLs include simulators and hardware synthesis programs. A simulation program can be used for design verification, while a synthesizer is used for automatic hardware generation.

In VHDL, the designer can build up a huge chip by describing the components from the bottom to the top hierarchically. By this way, designer describes a component only for once, even it may be used many times in the circuit. If that component needs to be changed, then applying that change only for one of them effects the others also.

In this thesis, the whole design is made in VHDL, totally hierarchically and simulated to test its functionality. Afterwards, it is synthesized to generate the schematic diagram and finally optimized to map the design into the target technology library.

```
LIBRARY mgc_portable ;
USE mgc_portable.qsim_logic.ALL ;

entity four_bit_mux is

 port(a, b: in qsim_state_vector(3 downto 0);  sel: in qsim_state ;
        xout: out qsim_state_vector(3 downto 0));

end four_bit_mux;

architecture behavioral of four_bit_mux is

 begin
  mux:process(sel, a, b)
   begin
    case sel is
     when '0' => xout <= a;
     when '1' => xout <= b;
     when others => xout <= "XXXX";
    end case;
  end process mux;

end behavioral;
```

Figure Apx.A.1.   A VHDL Sample (4 Bit Multiplexer)

APPENDIX B

C & BATCH CODES

INPUT COEFFICIENTS AND BATCH FILES

ANALYSIS COEFFICIENTS FOR NO COMPRESSION CASE (in1)

```
-0.067371764 0.094195111 0.40580489 0.56737176 0.56737176 0.40580489 0.094195111 -0.067371764
-0.094195111 0.067371764 0.56737176 0.40580489 -0.40580489 -0.56737176 -0.067371764 0.094195111
-0.094195111 -0.067371764 0.56737176 -0.40580489 -0.40580489 0.56737176 -0.067371764 -0.094195111
-0.067371764 -0.094195111 0.40580489 -0.56737176 0.56737176 -0.40580489 0.094195111 0.067371764
```

ANALYSIS COEFFICIENTS FOR 25% COMPRESSION CASE (in1110)

```
-0.067371764 0.094195111 0.40580489 0.56737176 0.56737176 0.40580489 0.094195111 -0.067371764
-0.094195111 0.067371764 0.56737176 0.40580489 -0.40580489 -0.56737176 -0.067371764 0.094195111
-0.094195111 -0.067371764 0.56737176 -0.40580489 -0.40580489 0.56737176 -0.067371764 -0.094195111
```

ANALYSIS COEFFICIENTS FOR 50% COMPRESSION CASE (in1100)

```
-0.067371764 0.094195111 0.40580489 0.56737176 0.56737176 0.40580489 0.094195111 -0.067371764
-0.094195111 0.067371764 0.56737176 0.40580489 -0.40580489 -0.56737176 -0.067371764 0.094195111
```

SYNTHESIS COEFFICIENTS (in2)

```
-0.067371764 0.094195111 0.40580489 0.56737176 0.56737176 0.40580489 0.094195111 -0.067371764
0.094195111 -0.067371764 -0.56737176 -0.40580489 0.40580489 0.56737176 0.067371764 -0.094195111
-0.094195111 -0.067371764 0.56737176 -0.40580489 -0.40580489 0.56737176 -0.067371764 -0.094195111
0.067371764 0.094195111 -0.40580489 0.56737176 -0.56737176 0.40580489 -0.094195111 -0.067371764
```

```
cp in1 inputs
qnt
cp coeffs coeffs.mb
cp in2 inputs
qnt
cp coeffs coeffs.rc
mb
channel
rc
calc
```

```
cp in1110 inputs
qnt
cp coeffs coeffs.mb
cp in2 inputs
qnt
cp coeffs coeffs.rc
mb
channel
rc
calc
```

```
cp in1100 inputs
qnt
cp coeffs coeffs.mb
cp in2 inputs
qnt
cp coeffs coeffs.rc
mb
channel
rc
calc
```

QNT.C

```
#include <stdio.h>
#define ROUND(a)   (((a) < 0) ? (int)((a)-0.5) : (int)((a)+0.5))

    FILE *fp,*fq,*fr,*fs;

main()
{

    int b,c,d,e,f,g,h,n,r,s,t,bits,qcoef[32];
    float z,v[32],y[32],coef[32];

        b=0;
          c=0;
        d=0;
        h=0;
        r=0;
        z=0;

      for (g=0; g<32; g++)
       {
        v[g]=0;
        y[g]=0;
        coef[g]=0;
        qcoef[g]=0;
         }

      if((fp=fopen("bitnum","w"))==NULL)
        { printf("file cannot be opened\n");
         exit(0); }
     if((fq=fopen("coeffs","w"))==NULL)
        { printf("file cannot be opened\n");
         exit(0); }
      if((fr=fopen("inputs","r"))==NULL)
        { printf("file cannot be opened\n");
         exit(0); }
      if((fs=fopen("out.qnt","w"))==NULL)
        { printf("file cannot be opened\n");
         exit(0);}

     printf(" \n");
     printf(" # of bits for the coefficients to be quantized...:\t");
     scanf("%d",&e);

    bits=e-1;
    t=32;
    z=0.9921876;
```

```
 for (f=0; f<t; f++)
      fscanf(fr,"%f",&coef[f]);

 r=(1<<(bits))-1;
  fprintf(fp,"%d\t",e);

 for (f=0; f<t; f++)
    {
     qcoef[f]=ROUND(r*coef[f]/z);
     y[f]=(z*qcoef[f]/r);
     v[f]=coef[f]-y[f];
      fprintf(fs,"%f => %x\t",coef[f],qcoef[f]);
      fprintf(fq,"%1.8f  ",y[f]);
    }
 fclose(fp);
 fclose(fq);
 fclose(fr);
 fclose(fs);
}
```

MB.C

```
#include <stdio.h>
FILE *fp,*fq,*fr,*fs,*ft,*fu;
main()

{
 int b,c,d,e,f,g,h,l,m,n,r,s,t,v,y,bits,o[5000],p[5000],x[5000],a[5000][7],qcoef[4][7];
 float z,coef[4][7],q[5000],u[5000][7];

 b=0;
 c=0;
 d=0;
 h=0;
 r=0;
 t=0;
 z=0;
 for (f=0; f<5000; f++)
  for (g=0; g<16; g++)
    for (c=0; c<4; c++)
        { a[f][g]=0;
         u[f][g]=0;
         coef[c][g]=0;
         qcoef[c][g]=0;
         x[f]=0;
         p[f]=0;
         q[f]=0;
        }

  if((fp=fopen("error1","w"))==NULL)
   { printf("error file cannot be opened\n");
    exit(0); }
 if((fq=fopen("coeffs.mb","r"))==NULL)
  { printf("coeffs file cannot be opened\n");
    exit(0); }
  if((fr=fopen("bitnum","r"))==NULL)
   { printf("bitnum file cannot be opened\n");
    exit(0); }
 if((fs=fopen("mbandin","r"))==NULL)
  { printf("mbandin file cannot be opened\n");
   exit(0); }
 if((ft=fopen("out.mb","w"))==NULL)
  { printf("output file cannot be opened\n");
   exit(0); }
 if((fu=fopen("link","w"))==NULL)
  { printf("link file cannot be opened\n");
       exit(0); }
/* Number of coefficients in each filter */
t=8;
/* four filters */
 fscanf(fr,"%d",&e);
for (g=0; g<4; g++)
  for (f=0; f<t; f++)
    fscanf(fq,"%f",&coef[g][f]);
fclose(fq);

 fscanf(fs,"%d",&d);
for (f=0; f<d; f++)
  fscanf(fs,"%d",&x[f]);
fclose(fs);

for (g=0; g<4; g++)
  {
    for (f=0; f<t; f++)
       fprintf(fp,"%1.8f\t",coef[g][f]);
     fprintf(fp,"\n");
  }
   fprintf(fp,"\n");
y=t+d-1;

for (f=0; f<y; f+=4)
  {
    for (h=0; h<4; h++)
       {
        b=f;
         m=f+h;
         for (g=0; g<8; g++)
          {
           q[m]+=(x[b]*coef[h][g]);
            fprintf(fp,"%f\t%d\t%f\n",q[m],x[b],coef[h][g]);
           b--;
           if (b<0)
       break;
          }
         fprintf(fp,"\t%2d.output :%3.8f\n",m,q[m]);
       }
  }
 fprintf(fu,"%d\t%d\n",e,d+t-1);
for (f=0; f<y; f++)
  fprintf(ft," %f\t",q[f]);
 fprintf(ft,"\n");
fclose(fr);
fclose(fp);
fclose(ft);
fclose(fu);
}
```

RC.C

```
#include <stdio.h>

FILE *fp,*fq,*fr,*fs,*ft;

main()
{

 int b,c,d,e,f,g,h,l,m,n,r,s,t,v,w,y,bits,a[5000],p[5000],qcoef[4][7];
 float z,coef[4][7],u[5000][7],q[5000],x[5000],o[4][5000],filout[4][5000];

 b=0;
 c=0;
 d=0;
 h=0;
 r=0;
 t=0;
 z=0;

 for (f=0; f<5000; f++)
  for (g=0; g<16; g++)
   for (c=0; c<4; c++)
      { a[f]=0;
       u[f][g]=0;
       coef[c][g]=0;
       qcoef[c][g]=0;
       filout[c][f]=0;
       o[c][f]=0;
       x[f]=0;
       p[f]=0;
       q[f]=0;
      }

 if((fp=fopen("error2","w"))==NULL)
  { printf("error file cannot be opened\n");
   exit(0); }
 if((fq=fopen("coeffs.rc","r"))==NULL)
  { printf("coeffs file cannot be opened\n");
   exit(0); }
 if((fr=fopen("link","r"))==NULL)
  { printf("link file cannot be opened\n");
   exit(0); }
 if((fs=fopen("out.rc","w"))==NULL)
  { printf("rcstrout file cannot be opened\n");
   exit(0); }
 if((ft=fopen("out.mb","r"))==NULL)
  { printf("output file of mband cannot be opened\n");
   exit(0); }
 fscanf(fr,"%d",&e);
 /*  Number of coefficients in each filter */
t=8;

 /*  four  filters */

for (g=0; g<4; g++)
  for (f=0; f<t; f++)
    fscanf(fq,"%f",&coef[g][f]);
fclose(fq);

bits=e;
 printf("%d \n",bits);
 fprintf(fs,"%d\t",bits);

 /*   "The number of inputs you're going to enter"  */
 /*   --Inputs (values varies between 0 & 255)--   */

 fscanf(fr,"%d",&d);
for (f=0; f<d; f++)
  fscanf(ft,"%f",&x[f]);

 fprintf(fp,"\n\t INPUT MATRIX AFTER UPSAMPLING\n");
for (c=0; c<4; c++)
 {
   fprintf(fp,"\n");
  for (h=c; h<d; h+=4)
     {
      o[c][h]=x[h];
       fprintf(fp,"%f\t",o[c][h]);
     }
 }

 fprintf(fp,"\n");

for (h=0; h<d; h++)
 {
   fprintf(fp,"\n");
  for (c=0; c<4; c++)
     fprintf(fp,"%f\t",o[c][h]);
 }

 fprintf(fp,"\n\t AFTER FILTERING\n");
for (c=0; c<4; c++)
 for (f=c; f<d; f++)
  {
     b=f;
     g=f-c;
```

```
        for (h=0; h<8; h++)
         {
          filout[c][g]+=o[c][b]*coef[c][h];
          b--;
          if (b<0)
            break;
         }
     }

     for (f=0; f<d; f++)
        {
          fprintf(fp,"\n");
         for (c=0; c<4; c++)
           fprintf(fp,"%f\t",filout[c][f]);
        }

  for (f=0; f<d; f++)
   {
     for (c=0; c<4; c++)
        q[f]+=filout[c][f];
     fprintf(fs,"%f\t",q[f]);

   }

  fclose(fp);
  fclose(fr);
  fclose(fs);
  fclose(ft);
}
```

CHANNEL.C

```
#include <stdio.h>
#define ROUND(a)   (((a) < 0) ? (int)((a)-0.5) : (int)((a)+0.5))

    FILE *fp,*fq,*fr,*fs;

main()
{

    int b,c,d,e,f,g,h,n,r,s,t,y[5000],z,bits,qout[5000];
    float v[5000],out[5000];

        b=0;
         c=0;
        d=0;
        h=0;
        r=0;
        z=0;

      for (g=0; g<5000; g++)
      {
       v[g]=0;
       y[g]=0;
       out[g]=0;
       qout[g]=0;
        }

     if((fp=fopen("bitnum","w"))==NULL)
       { printf("file cannot be opened\n");
        exit(0); }
     if((fq=fopen("out.chn","w"))==NULL)
       { printf("file cannot be opened\n");
        exit(0); }
     if((fr=fopen("out.mb","r"))==NULL)
       { printf("file cannot be opened\n");
        exit(0); }
     if((fs=fopen("link","r"))==NULL)
       { printf("file cannot be opened\n");
              exit(0); }


     printf(" \n");
     printf(" # of bits for the channel to be quantized...:\t");
     scanf("%d",&e);

     fscanf(fs,"%d",&b);
     fscanf(fs,"%d",&t);
    bits=e-1;
```

```
 z=255;

  for (f=0; f<t; f++)
        fscanf(fr,"%f",&out[f]);

  r=(1<<(bits))-1;
   fprintf(fp,"%d\t",e);

  for (f=0; f<t; f++)
     {
      qout[f]=ROUND(r*out[f]/z);
      y[f]=(z*qout[f]/r);
      v[f]=out[f]-y[f];
      fprintf(fq,"%d  ",y[f]);
     }
 fclose(fp);
 fclose(fq);
 fclose(fr);
 fclose(fs);
}
```

CALC.C

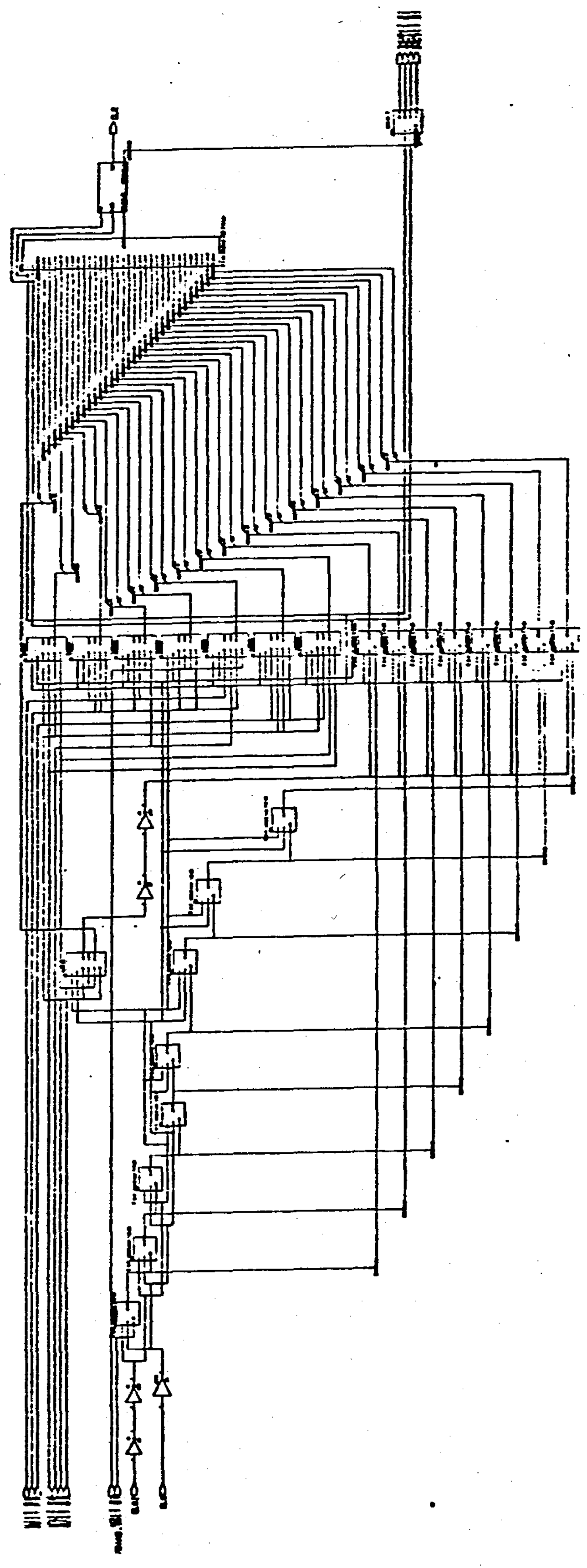
```
#include <stdio.h>
#include <math.h>
#define ROUND(a)   (((a) < 0) ? (int)((a)-0.5) : (int)((a)+0.5))

main()
{

 FILE *fa,*fb,*fc,*fp;

 int b,c,d,e,f,g,h,n,r,s,t;
 float z,v,y,q[5000],o[5000];
 double i,j,k,l,p[5000];



 b=0;
 c=0;
 d=0;
 h=0;
 r=0;
 t=0;
 i=0;
 j=0;
 k=0;
 l=0;
 z=0;

 for (f=0; f<5000; f++)
  {
   o[f]=0;
   p[f]=0;
   q[f]=0;
  }

  if((fa=fopen("mbandin","r"))==NULL)
   { printf("file cannot be opened\n");
    exit(0); }
 if((fb=fopen("out.rc","r"))==NULL)
   { printf("file cannot be opened\n");
    exit(0); }
 if((fc=fopen("ratio","a"))==NULL)
   { printf("file cannot be opened\n");
    exit(0); }
 if((fp=fopen("error","w"))==NULL)
   { printf("file cannot be opened\n");
    exit(0); }

 fscanf(fb,"%d",&e);
```

```
  fscanf(fa,"%d",&d);
 c=d-4;
  for (h=0; h<c; h++)
   {
     fscanf(fa,"%f",&q[h]);
      fprintf(fp,"%d\t%f\n",h,q[h]);
   }
 g=d+3;
  for (h=0; h<g; h++)
   {
     fscanf(fb,"%f",&o[h]);
      fprintf(fp,"%d\t%f\n",h,o[h]);
   }

  fprintf(fp,"\n\n");
  fprintf(fp,"errors    :\n----------\n");

 for (f=0; f<c; f++)
  { b=f+7;
   p[f]=q[f]-o[b];
    fprintf(fp,"between %f and %f =%1.9f\n ",q[f],o[b],p[f]);
  }
 for (f=0; f<c; f++)
  v+=q[f];
 z=v/c;

  fprintf(fp,"\n\n");

 i=0;
 for (f=0; f<c; f++)
  i+=(p[f]*p[f]);
 k=i/c;
 j=sqrt(k);
 y=j/z;
  fprintf(fp,"input avr.= %f\n",z);
  fprintf(fp,"RMS error = %1.9f\n",j);
 fprintf(fp,"Percentage  error = %1,9f\n",y);
 fprintf(fc," %d\t",e);
 fprintf(fc," %1.9f\n",y);

 fclose(fa);
 fclose(fb);
 fclose(fc);
 fclose(fp);
}
```

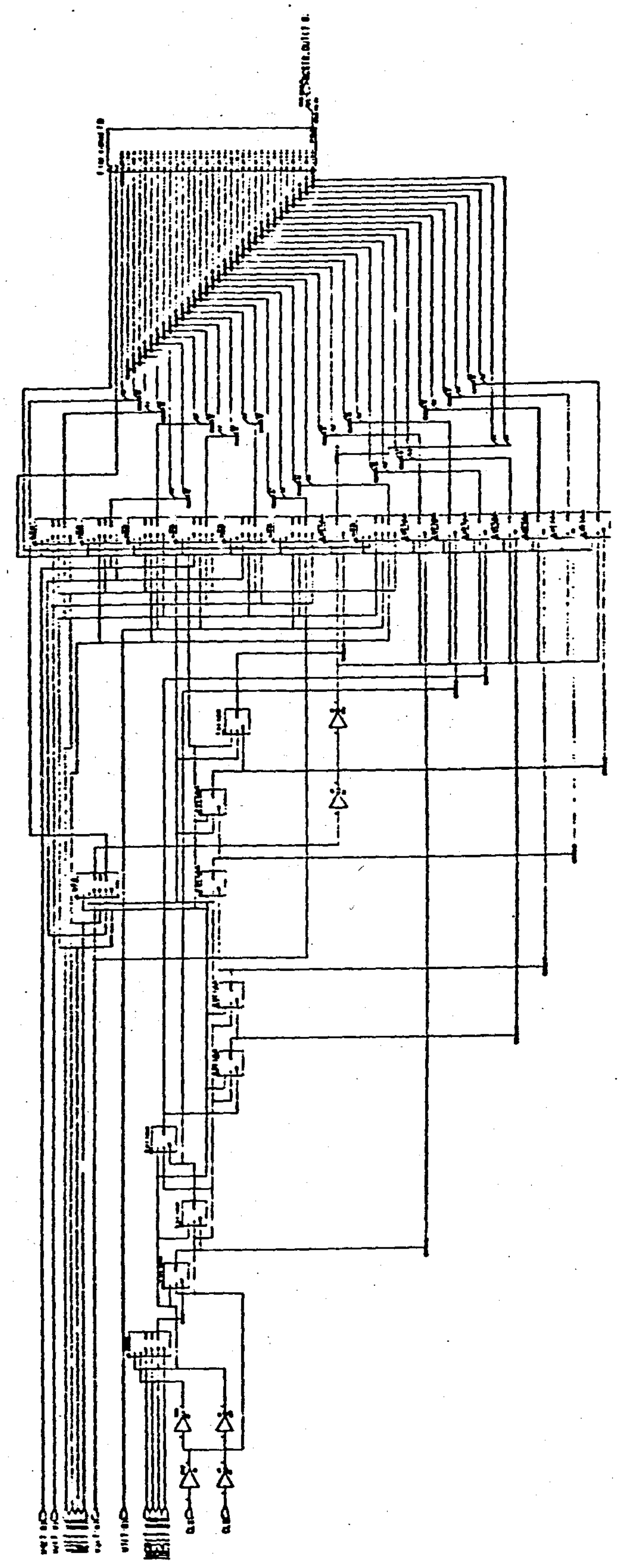
APPENDIX C

SCHEMATIC DIAGRAMS

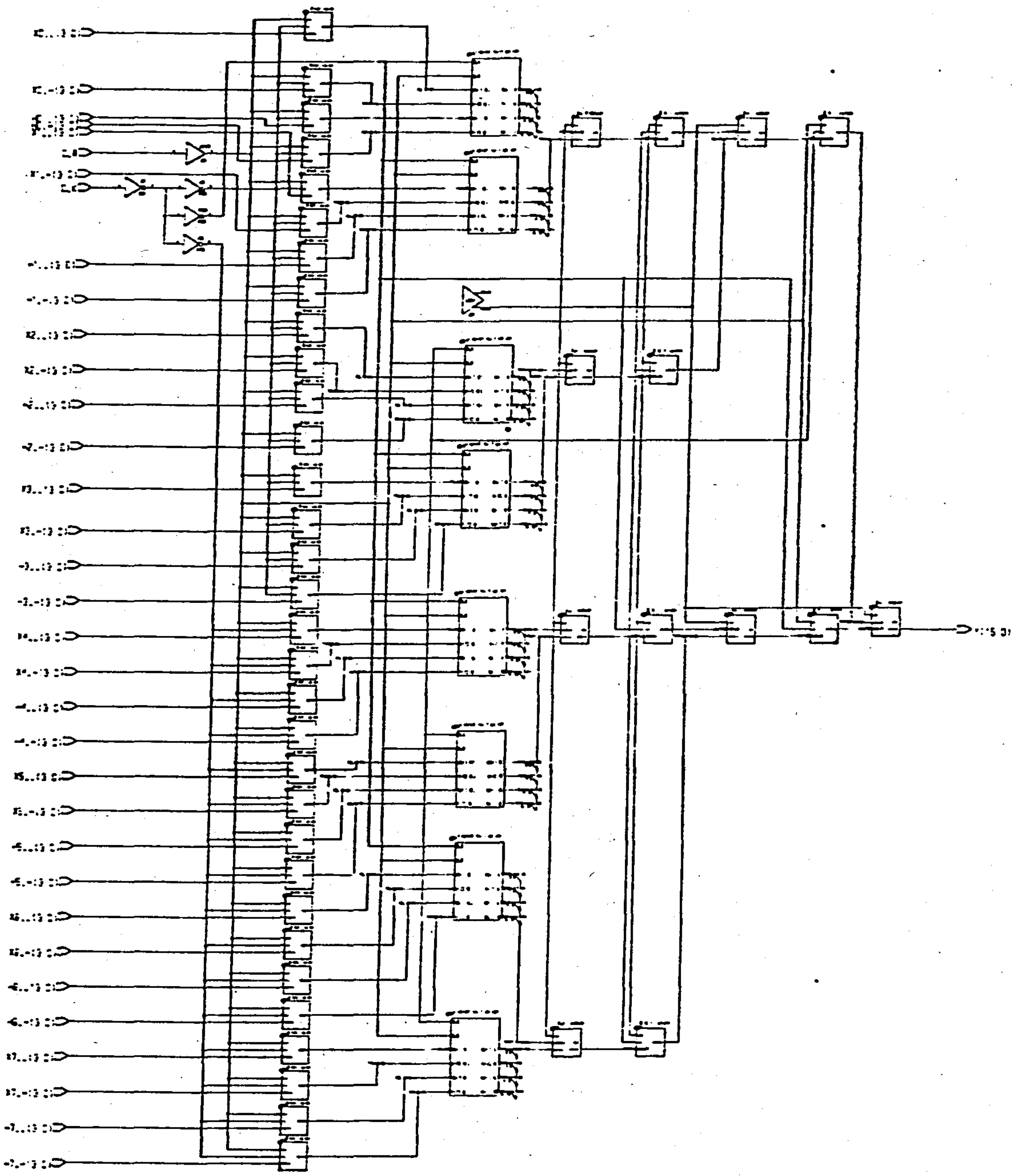
ANALYSIS FILTER



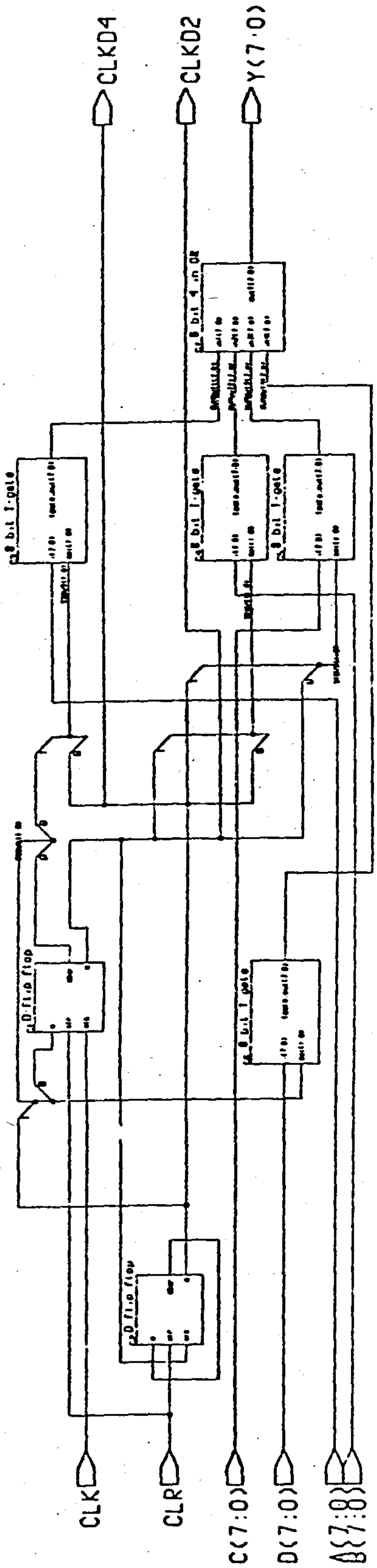
SYNTHESIS FILTER



FIR FILTER



SWITCHING CIRCUIT


CLKD4
CLKD2
Y(7:0)
8 bit 4 in OR
8 bit T-gate
8 bit T-gate
8 bit T-gate
D flip flop
8 bit T-gate
D flip flop
CLK
CLR
C(7:0)
D(7:0)
A(7:0)
B(7:0)