ARCHITECTURES AND IMPLEMENTATIONS FOR SPEECH ENHANCEMENT

by

Gökhan Coşgül

BS. in E.E., Boğaziçi University, 1998

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Electronics Engineering

Boğaziçi University

2000

ACKNOWLEDGMENTS

I would like to thank to my thesis supervisors Assoc. Prof. Günhan Dündar and Assoc. Prof. Levent Arslan for their endless support and motivation.

I would also like to thank to Prof. Dr. Ömer Cerid for his support and being a jury member.

I would like to thank to Prof. Dr. Fikret Gürgen and Prof. Dr. Avni Morgul for being jury members.

ARCHITECTURES AND IMPLEMENTATIONS FOR SPEECH ENHANCEMENT

ABSTRACT

Complexity of a high performance speech enhancement algorithm has been reduced while retaining its quality. This modification leads to faster implementation as well as lower cost.

The reduced complexity speech enhancement algorithm has taken as a typical digital signal processing application. Thus, architectures and implementation techniques for blocks involved in this algorithm have been studied. In addition, thanks to the design methodology employed, these implementations can easily be ported into other algorithms involving the same blocks.

ARCHITECTURES AND IMPLEMENTATIONS FOR SPEECH ENHANCEMENT

ÖZET

Yüksek performanslı bir ses iyileştirme algoritmasının, kalitesi korunarak karmaşıklığı düşürüldü. Bu değişiklik sayesinde daha hızlı calışması ve ucuzlaması sağlandı.

Bu düşük karmaşıklıktaki ses iyileştirme algoritması, örnek bir sayısal im işleme uygulaması olarak seçilerek kapsadığı modüller için mimariler ve gerçeklemeler incelendi. Ayrıca, kullanilan tasarım metodu sayesinde, bu gerçeklemelerin, aynı modülleri içeren başka elektronik tasarımlarda da kolaylıkla kullanılabilinmesi sağlandı.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

k	Data point number in frequency domain
m	Stage number in FFT
n	Data point number in time domain
N	Number of the data points
p	Order of the predictor coefficients in LPC
$\varepsilon\{\}$	Expected value operator
Σ	Summation operator
$\psi(n)$	Window function
ξ	Mean squared error
$\propto$	Proportional to

1. INTRODUCTION

Digital signal processing (DSP) is used in numerous applications such as video compression, digital set-top box, digital audio, wireless communications, biomedical signal processing and speech processing which is of interest in this thesis. Table 1.1 shows some of the common speech related DSP algorithms and corresponding applications [1].

TABLE 1.1. Speech algorithms and applications.

ALGORITHM	APPLICATIONS
Speech coding and decoding	Digital cellular phones, secure communications, multimedia computers
Speech recognition	Robotics and automotive applications, personal communication systems
Speech Synthesis	Advanced user interfaces, robotics
Noise Cancellation	Professional audio, digital cellular phones

Digital signal processing systems are implemented in two basic ways: software in general purpose computers, and special purpose hardware. The software approach is relatively straight-forward.

Recently, the field of DSP has found wider interest through the advances in DSP applications and in VLSI (Very Large Scale Integration) technologies. Real-time DSP applications impose additional challenges on the implementations of the DSP systems. This results in the need for application specific integrated circuits (ASIC) to tackle real-time implementations of DSP algorithms.

In many speech communication systems, the presence of background interference causes the quality or intelligibility of speech to degrade [2]. The quality of speech is extremely important in data conversion, transmission, or reproduction. The purpose of many enhancement algorithms is to reduce background noise, improve speech quality, or suppress channel or speaker interference. If one can enhance quality and/or intelligibility of noisy speech, this also contributes to improved performance in other speech applications, such as speech coding, speech recognition etc. [2]. With the advent of wireless digital communication systems, noise suppression has become even more important. This is because coding algorithms become less immune to background noise as bit rates go down. If we examine the literature, it is clear that most currently available algorithms can not tackle the challenges implied by today's speech applications [3], [4]. These are either simple FIR based algorithms whose performances are insufficient [3] or too complex [5] to be implemented in real-time with a general DSP or an ASIC when low power is a requirement.

This thesis can be considered as having two aspects. First part is the signal processing whereas the second is the VLSI implementation part.

In the first part of the thesis, the complexity of a powerful algorithm, modified Wiener Filtering [6], has been reduced while retaining its quality. This enables real-time implementation on a moderate programmable digital signal processor as well as less cost for an ASIC approach.

In the second part, architectures and implementations for blocks, which are also common in many DSP algorithms, that are involved in the new algorithm are explored. Thanks to the design methodology employed, these architectures as well as implemented blocks can easily be ported to other VLSI designs.

The organization of the thesis is as follows. Section 2 explains the speech processing part. In section 3, design methodology used in implementation can be found. Sections 4, 5, 6, 7, 8, and 9 clarify the implemented blocks which are windowing, fast Fourier transform (FFT), finite impulse response (FIR) filter, square-root and division, multiply accumulate unit, and output block, respectively. Section 10 concludes the thesis.

2. ALGORITHM DESCRIPTION

This section explains the speech enhancement algorithm that is the concern of this thesis.

2.1. Wiener Filtering

In many practical applications we are given an input signal, which consists of the sum of desired signal and an undesired noise or interference, and we are asked to design a filter that will suppress the undesired interference component. In such a case, the objective is to design a system that filters out the additive interference while preserving the characteristics of the desired signal.

Short-term Wiener filtering is an approach in which a frequency weighting for an optimum filter is first estimated from noisy speech, y(n). The linear estimator of the uncorrupted speech s(n), which minimizes the MSE (Mean Square Error) criterion, is obtained by filtering y(n) with a non-causal Wiener filter. The filter requires a priori knowledge of both speech and noise statistics, and therefore must also adapt to changing characteristics. In a single-channel framework, noise statistics must be obtained during silent frames. Also, since noise-free speech is not available, a priori statistics must be based upon y(n), resulting in an iterative estimation scheme.

To obtain the transfer function of an iterative Wiener filter, we begin with $\underline{s}$, $\underline{d}$, and $\underline{y}$, which are stochastic processes representing speech, noise, and noisy speech, respectively. Let $\underline{s}(n)$, $\underline{d}(n)$, and $\underline{y}(n)$ represent random variables from the respective processes, and s(n), d(n), and y(n) denote realizations.

$$y(n) = s(n) + d(n) \tag{2.1}$$

We want to formulate a linear filter with which to produce an estimate of s(n), call s'(n) that is optimal in the MSE sense. That is, we desire a filter with impulse response h(n) such that with input s(n) the output is an estimator s'(n) for which

$$\xi = \varepsilon\{[\underline{s}(n)-\underline{s}'(n)]^2\} \tag{2.2}$$

is minimized. In frequency domain, a Wiener Filter has the transfer function shown below

$$H(\omega) = \sqrt{\frac{\underline{P}_s(\omega)}{\underline{P}_s(\omega)+P_n(\omega)}} \tag{2.3}$$

where $\underline{P}_s(\omega)$ is the power spectrum of the estimated uncorrupted speech and $P_n(\omega)$ is the power spectrum of the noise.

2.2. Modified Wiener Filtering

Traditional Wiener filters are first proposed for speech enhancement in [4]. Later, various modifications to the algorithm were proposed by a number of other researchers [6].

In [6], a modified Wiener filter is proposed on which our implemented algorithm is based. This filter uses a noise suppression factor that is time-varying and is computed based on the frame-by-frame signal-to-noise ratio (SNR). The lowest

Wiener filter gain is clamped to a preset minimum threshold. The clean speech power spectrum estimate is calculated from the linear predictive coding (LPC) model spectrum of the noisy speech $P_Y(\omega)$ with only a gain modification:

$$P_s(\omega) = \frac{E_y - E_n}{E_y} P_y(\omega) \tag{2.4}$$

where, E_Y and E_n are, respectively, energies of the noisy speech and noise and $P_s(\omega)$ is the power spectrum of the clean speech. The Wiener filter expression then reduces to:

$$H(\omega) = \sqrt{\frac{P_y(\omega)}{P_y(\omega) + \frac{E_y}{E_y - E_n} \alpha P_n(\omega)}} \tag{2.5}$$

Next, the factor multiplying $P_n(\omega)$ in the above expression is made inversely dependent on SNR and is allowed to change from frame to frame. This will ensure stronger suppression during high-energy (e.g. vowels) frames which are not corrupted as much to begin with. The desired SNR dependence is achieved simply by replacing α (a constant determining noise suppression factor) with $E_n\alpha/E_Y$. Then, the expression for $H(\omega)$ becomes:

$$H(\omega) = \sqrt{\frac{P_y(\omega)}{P_y(\omega) + \frac{E_n}{E_y - E_n} \alpha P_n(\omega)}} \tag{2.6}$$

Unlike other Wiener filtering approaches, the MWF (Modified Wiener Filter) method is non-iterative and hence computationally attractive. The SNR-dependent noise suppression factor gives MWF the ability to suppress those parts of the degraded signal where speech is not likely to be present, and not to suppress and hence not to distort the voiced speech as much.

2.3. Implemented Algorithm

MWF algorithm is modified further to reduce the complexity. First of all, instead of the LPC model spectrum, a smoothed FFT spectrum is used. Secondly, the transfer function is handled in magnitude domain, which results in (2.7) for H(ω). The final block diagram of the implemented algorithm is shown in FIGURE 2.1.

$$H(\omega) = \frac{P_y(\omega)}{P_y(\omega) + \frac{E_n}{E_y - E_n}\alpha P_n(\omega)} \tag{2.7}$$

In spite of these modifications, the quality of the algorithm is retained by changing the method that is used for updating the noise spectrum. Furthermore, α is chosen to be high to achieve better performance. Spectrograms of a clean speech signal, the degraded speech signal, MWF applied speech signal and the speech signal enhanced by the implemented algorithm are shown in FIGURE 2.2, FIGURE 2.3, FIGURE 2.4, and FIGURE 2.5, respectively.

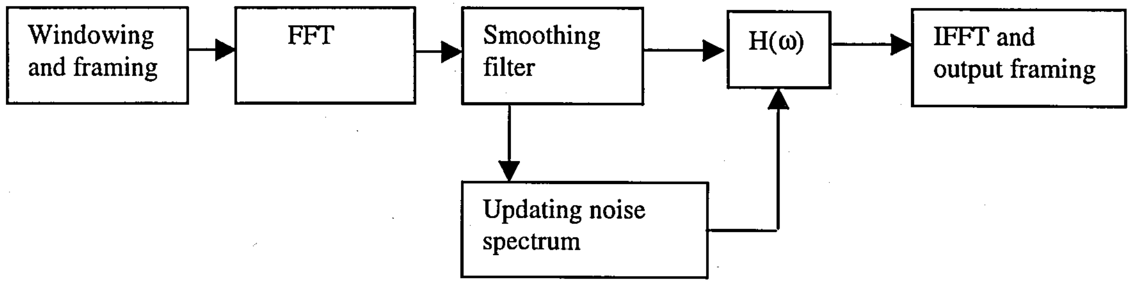

FIGURE 2.1. Block diagram of the implemented algorithm.

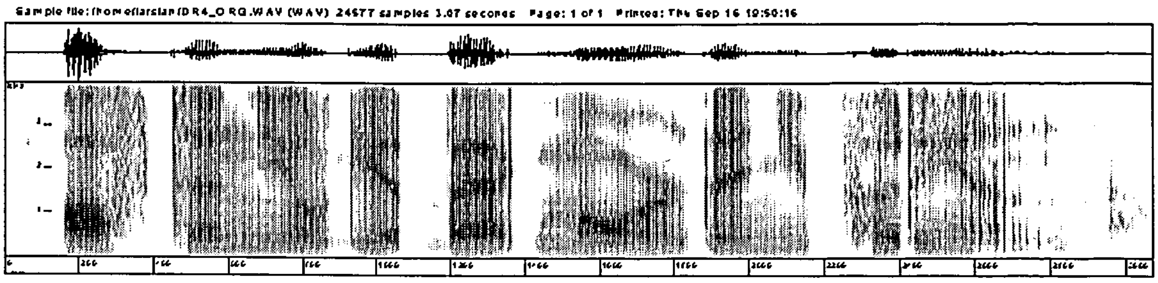

FIGURE 2.2. Spectrogram of the original speech.

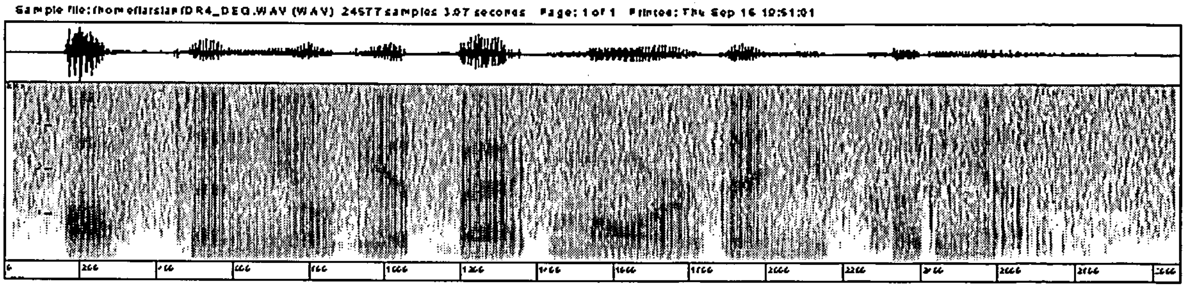

FIGURE 2.3. Spectrogram of the degraded speech.

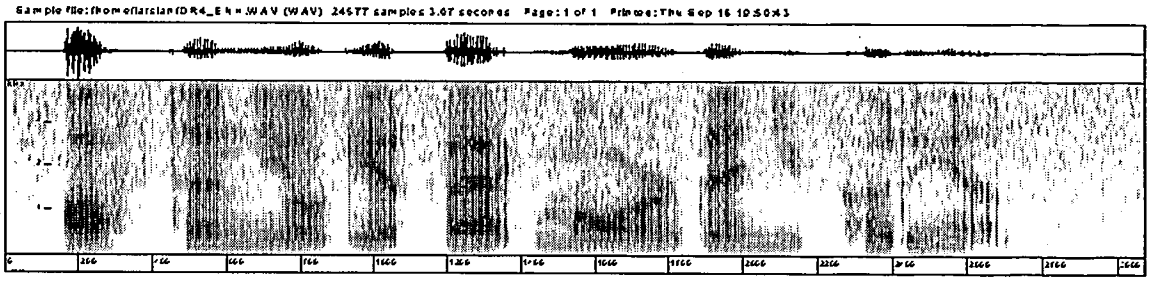

FIGURE 2.4. Spectrogram of the enhanced speech with MWF.

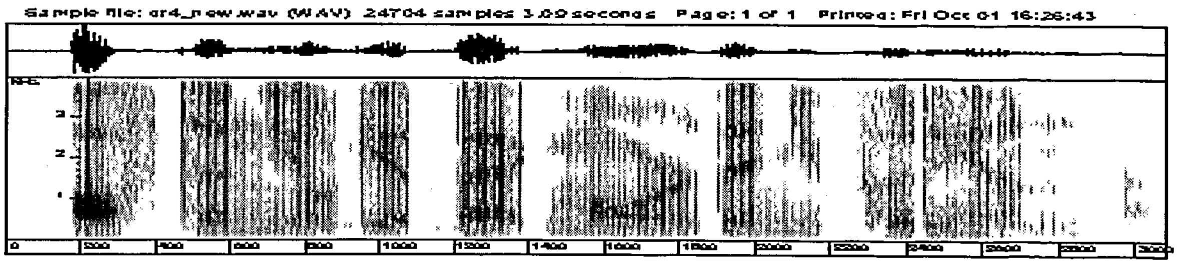

FIGURE 2.5. Spectrogram of the enhanced speech with the implemented algorithm.

In addition, a subjective MOS test is conducted with 11 listeners. Mean values of MOS scores are shown in TABLE 2.1 TABLE 2.2 shows the standard deviations for MOS scores. All values are over 5.

TABLE 2.1. Averages of MOS scores.

	Degraded	MWF	Implemented
Sample 1	2.18	2.54	4.09
Sample 2	2.00	3.09	2.63
Sample 3	1.91	2.45	2.72
Sample 4	1.91	2.63	2.54
Overall	2.00	2.68	2.99

TABLE 2.2. Standard Deviations of MOS scores.

	Degraded	MWF	Implemented
Sample 1	1.07	0.93	1.13
Sample 2	0.77	1.04	0.92
Sample 3	1.22	0.52	0.64
Sample 4	0.94	1.02	0.82
Overall	1.00	0.88	0.88

3. DESIGN METHODOLOGY

Hardware design has recently undergone dramatic changes in design methodology, especially with proliferation of hardware description languages (HDLs) that promote the integration of the design methodology into a unified environment[7].

Several motivations to the use of VHDL (Very high speed integrated circuits Hardware Description Language) are listed below[7]:

- Completeness - Systems that are captured in VHDL can be maintained and upgraded for their lifetime period with no need to switch to another modeling language.
- Technology Independence - VHDL models can be kept technology independent until the very last stages of the synthesis process, providing the capability to retarget the same model to a variety of vendor technologies.
- Portability - Being a standard, VHDL allows the portability and the reuse of VHDL models across a wide range of design tools.
- Abstraction levels - VHDL supports a wide range of design abstraction levels and hierarchy. Hence, the same language is used for the entire design process, bridging the gap between design teams.
- Excellent documentation - Documentation is a very important issue with respect to future upgrades and maintenance. VHDL allows the documentation to be done during design process, reflecting the true design intent.

Besides, another important advantage of using VHDL is parameterization of the design units. In other words, a square-root unit can be written such that it can take the square-root of any word length by changing generic map.

In this thesis, for the design of all the blocks, VHDL is used. However before implementing the blocks, the original algorithm (MWF) which was written in C programming language is ported to MATLAB. Hence, original algorithm is modified using MATLAB to achieve the same performance with reduced complexity. Besides, MATLAB is used for functional modeling of each block before implementing in VHDL.

Once functional verifications of each block is accomplished, hardware implementation is the next stage. For this stage Mentor Graphics' tools are used. QuicksimII and QHDL are the tools for digital simulation and VHDL simulation, respectively. AutologicII is used for synthesizing, technology mapping and optimizing the blocks. IC Station is the tool used for generating layouts.

Although most of the design units are parameterized in this thesis, blocks are assumed to operate on an 8-KHz 16-bit signed speech signal.

All the blocks presented in this thesis are mapped to MIETEC 0.7um 3.3V process and all the area reports are according to this technology. This technology has a 426.87 um^2 unit cell area that must be multiplied with the total area number in the area report given for each block.

For simulation waveforms it must be noted that all values are decimal and for the inputs negative numbers are shown in two's complement decimal format due to the simulator used.

4. WINDOWING

This section elaborates the windowing and overlapping that is used as an input block for various algorithms.

4.1. Theory

In the design of FIR filters and in spectral analysis, it is needed to apply time weighting to a function before going into frequency domain. This is mostly due to smooth the time domain function sharpened by the glitches. This is achieved by the windowing functions. In mathematical terms, windowing is nothing but the multiplication of the signal by a "window" sequence of finite-length, $\psi(n)$.

4.1.1. Hanning Window

The Hanning window is simply the raised cosine or sine-squared function. It is defined as

$$\Psi_N(n) = \sin^2 \frac{\pi.n}{N_1 - 1}, \qquad n = 0, 1, \ldots, N_1 - 1. \qquad (4.1)$$

It has a -31dB peak sidelobe and a 18dB/octave rolloff. The Hanning window is probably the most widely used window function in spectrum analysis because of its excellent rolloff rate.

4.1.2. Hamming Window

The Hamming window is designed to minimize the peak sidelobe level while maintaining approximately the same mainlobe width as the Hanning window. This window is defined by

$$\Psi_N(n) = \frac{1}{2}\left[1 - \cos\frac{2\pi.n}{N_1}\right], \qquad n = 0, 1, \ldots\ldots, N_1\text{-}1. \qquad (4.2)$$

It has a peak sidelobe of -41dB and a rollof rate of 6dB/octave. The low peak sidelobe level of the Hamming window is of more importance in the design of FIR filters rather than spectrum analysis[8].

4.1.3. Blackman Window

The ideas behind the Hanning and Hamming windows are combined in the Blackman window, given by

$$\Psi_N(n) = 0.42 - 0.5\cos\frac{2\pi.n}{N_1 - 1} + 0.08\cos\frac{4\pi.n}{N_1 - 1}, \qquad n = 0, 1, \ldots\ldots, N_1\text{-}1. \qquad (4.3)$$

It has a -57dB peak sidelobe and a 18dB/octave rolloff rate. Use of this window sacrifices spectral resolution to gain additional attenuation.

4.2. Implementation

Block diagram of the input block of the system is depicted in FIGURE 4.1. This block is designed with the assumption of input is parallel and applies the 256 point Hanning window with overlapping to the input data. Shift register is used to implement the overlap-add method, coefficient ROM (Read Only Memory) acts as a look-up table for the window function.

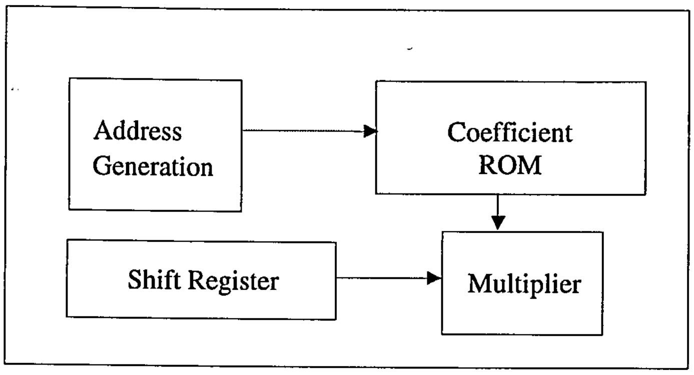

FIGURE 4.1 Block diagram of the windowing and overlapping.

4.2.1. Shift Register

A parameterized 128x16-bit wide shift register is designed. This shift register is used to achieve the overlap-add method which basically implements the (4.4).

$$\begin{aligned} &frm(1:N/2)=frm(N/2+1:N) \\ &frm(N/2+1:N)=input(N/2) \end{aligned} \tag{4.4}$$

where *frm* denotes the output of the shift register and N is the length of the window function as well as the framing.

Synthesis results for shift register bank is as follows:

```
Number of nets           :        259
Number of cells          :        128
Number of operators      :          0

Combinational area       :          0    (  0 %)
Buffer/Inverter area     :          0    (  0 %)
Sequential area          :     811.52    (100 %)
Operator area            :          0    (  0 %)
                             -----------
Total area               :     811.52
```

4.2.2. Address Generation

In order to implement a window function, Hanning window in our case, it is required to multiply the input with the appropriate window coefficient. This block addresses a coefficient ROM and secures the right implementation of a window function.

4.2.3. Multiplier

For a multiplier, an operator from numeric_signed library is used. This enables a signed multiplication in a synthesizable manner.

4.2.4. Overall Windowing and Overlapping

VHDL simulation results for the overall windowing and overlapping function are shown in FIGURES 4.2 , 4.3 and 4.4.

The signals shown in simulation waveforms can be listed as follows:

/mout:	Output of the multiplier.
/frout:	Output of the shift register.
/hanrout:	Output of the coefficient ROM.
/res:	Data written to the RAM (Random Access Memory).
/gx:	ROM and RAM address.
/gwr:	Read/write signal for RAM.
/atlast:	Acknowledge signal for the end of operation.

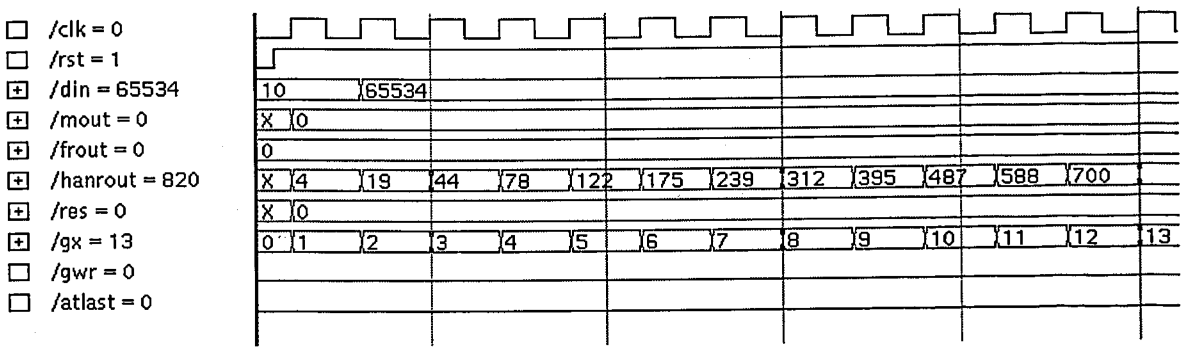

FIGURE 4.2. Simulation results for the first input frame.

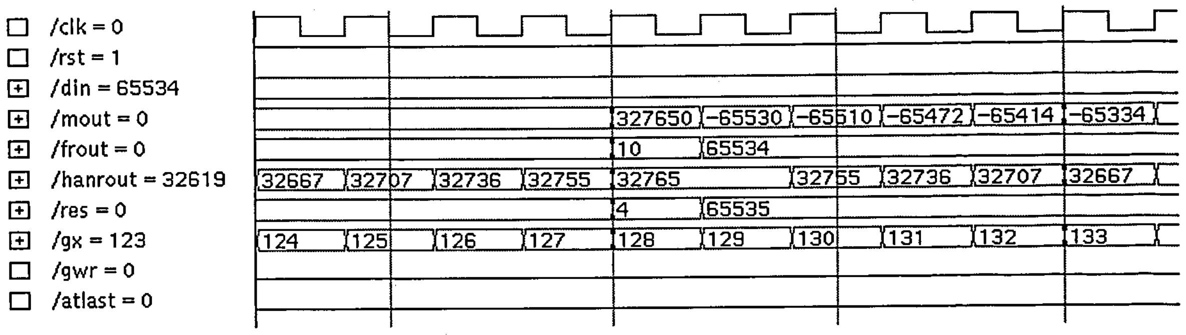

FIGURE 4.3. Simulation results for first inputs arrived.

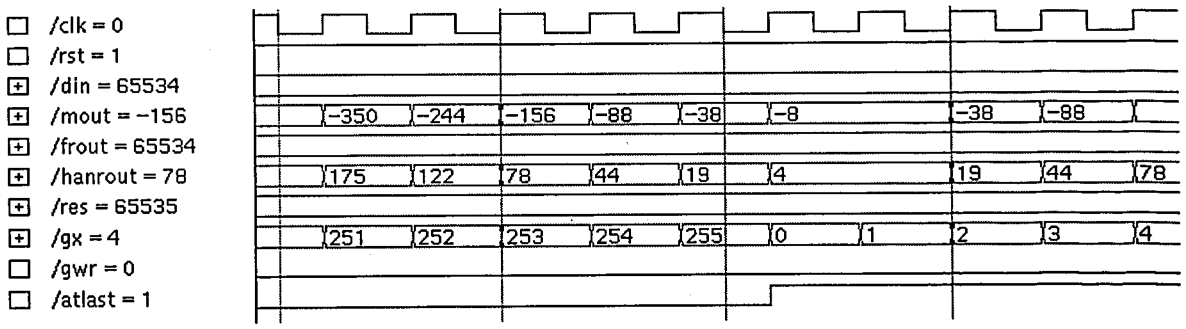

FIGURE 4.4. Simulation results for the windowing function of the first frame.

FIGURE 4.2 shows the beginning of a new speech data, denoted by /din. /mout is the output of the multiplier which has the inputs /frout, output of the shift register, and /hanrout, output of the Hanning window coefficient ROM. /res is the truncated data of /mout that is written to a RAM, which has addressed by /gx and has the read/write signal /gwr.

FIGURE 4.3 verifies the overlapping of 128 samples and FIGURE 4.4 shows that after 256 point windowing is accomplished the unit acknowledges the end signal.

It must be noted that, for this architecture while the odd numbered frames are correctly written to the RAM, even numbered frames are written in reverse order. However this not a very important problem. A design approach that tackles this problem will be illustrated in the FFT block.

5. FFT and IFFT

This section explains the basics of the FFT and inverse FFT and elaborates the hardware implementation.

5.1. Theory

In many areas of science and engineering, the representation of signal or other functions by sums of sinusoids or complex exponentials leads to convenient solution to problems and often to greater insight into physical phenomena than is available by other means. Such representations- Fourier representation as they are commonly called- are useful in signal processing for two basic reasons. The first is that for linear systems it is very convenient to determine the response to a superposition of sinusoids or complex exponentials. The second reason is that the Fourier representation often serves to place in evidence certain properties of the signal that may be obscure or at least less evident in the original signal. A common example is the interference from domestic power lines, which, though masked in the time domain, appears clearly as discrete harmonics of 50-60 Hz in the frequency domain [9].

Therefore, Fourier transform based signal processing, which is often simply termed "spectral analysis", is used extensively for modern-day engineering tasks, including speech recognition, vibration analysis, and biomedical engineering.

The Discrete Fourier Transform (DFT) plays an important role in many applications of digital signal processing, including linear filtering, correlation analysis, and spectrum analysis. A major reason for its importance is the existence of efficient algorithms for computing the DFT.

In essence, the DFT is simply a mapping of one ordered set of N complex numbers to a different ordered set-the former conveying time domain information, the latter frequency domain information. The precise definition of the DFT is

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-2\pi nk/N} \qquad (5.1)$$

where n is used as the sequence member index (sample number) in the input discrete signal and k as the index for the transformed signal.

On the other hand, inverse DFT (IDFT) has the equation shown below

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) \cdot e^{2\pi nk/N} \qquad (5.2)$$

Thus, a DFT processor may be used to calculate the IDFT by simply conjugating the input samples, and then conjugating the result [9].

The DFT expression can be expressed in matrix form as:

$$\begin{bmatrix} X(0) \\ X(1) \\ . \\ . \\ X(k) \\ X(N-1) \end{bmatrix} = \begin{bmatrix} W(0,0) & W(0,1) & \dots & W(0,n) & \dots & W(0,N-1) \\ W(1,0) & W(1,1) & \dots & W(1,n) & \dots & W(1,N-1) \\ . & . & & . & & . \\ . & . & & . & & . \\ W(k,0) & W(k,1) & \dots & W(k,n) & \dots & W(k,N-1) \\ W(N-1) & W(N-1,1) & \dots & W(N-1,n) & \dots & W(N-1,N-1) \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ . \\ . \\ x(n) \\ x(N-1) \end{bmatrix} \qquad (5.3)$$

where n and k are used for the matrix column and row indices respectively. W(k,n) is termed the *twiddle factor* and is usually denoted as

$$W(k,n) = W_N^i \qquad (i = nk). \tag{5.4}$$

It is a unit-magnitude complex number whose real and imaginary parts are the cosine and sine of the angle (-2πi/N) radians. Thus,

$$W_N^i = e^{-j2\pi ni/N} = \cos(2\pi i/N) - j\sin(2\pi i/N). \tag{5.5}$$

To compute DFT as defined in (5.1), we must do N^2 complex multiplication assuming table look-up method for twiddle factors is employed. This requirement makes DFT algorithms based on the (5.1) prohibitively expensive except for very small N.

On the other hand, number of multiplications can be reduced by 25 percent using a well-known trick for multiplying complex numbers [10]. Although we are looking for much better speed-ups than this, this technique will be reviewed here, since it is the first and simplest. The product of (a+jb)(c+jd) is normally written as (ac-bd)+j(ad+bc). Let P = d(a-b), Q = a(c-d) , and R = b(c+d). Finding P, Q, and R requires only three multiplications, and the product is (P+Q)+j(P+R). However, it must be noted that saving in multiplications is bought at the price of an increase in the number of additions; this trade-off is typical of these fast algorithms.

The Fast Fourier Transform, developed originally by Cooley and Tukey in 1965, is a significantly less computationally intensive method of evaluating the DFT, and thus particularly attractive for real-time spectral analysis using DSP technology. The FFT reduces the number of complex multiplications involved N^2 for the DFT to order of $N\log_2(N)$ [9, 10, 11]. There are number of types of FFT algorithm, but all share the common constraint of only working for certain values of N. If this constraint is unacceptable, the full-blown DFT must be implemented. Some of the well known FFT algorithms are radix-2 algorithms, non radix-2 algorithms, split radix algorithms, and Winograd algorithm [9, 10, 11].

5.1.1. Radix-2 Algorithms

The FFT algorithm is based on the principle of computing a large transform via a number of smaller, more manageable transforms. There are in fact two methods to achieve this result. The first, which is called decimation-in-time (DIT), implements the two N/2-point transforms using the even and odd elements of the input sequence respectively. These two transforms are then merged by a further N/2 two-point DFT to generate the desired output elements. The second method, called decimation-in-frequency (DIF), performs the same two sets of operations, but in reverse order. The input samples are first processed in pairs by N/2 two-point transforms, followed by the two N/2-point transforms which generate odd and even components of the output sequence directly. Both methods are in-place, that is the same memory locations used for storing the DFT inputs can be overwritten, first with intermediate results, and then with the output results. FIGURE 5.1 and FIGURE 5.2 illustrate the DIT and DIF decomposition algorithms respectively.

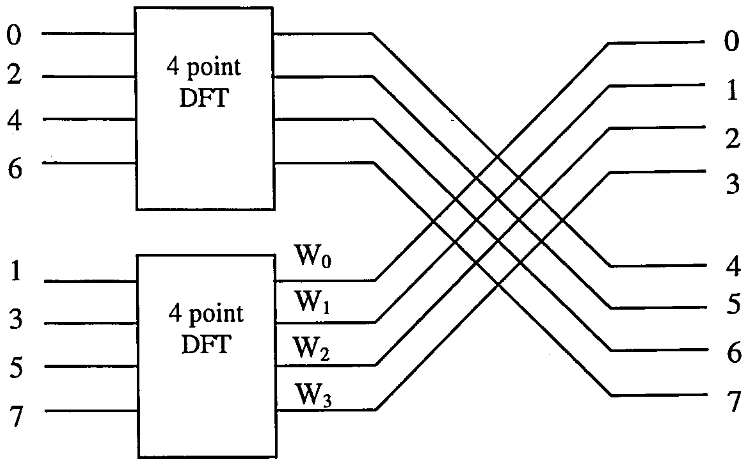

FIGURE 5.1. Decimation-in-time FFT algorithm.

In the DIT algorithm the input is out of sequence (scrambled) and the output is in correct order. In the DIF reverse is true.

Normally the two-point DFT and twiddle multiplication are combined in a single processing operation, which is termed "butterfly".

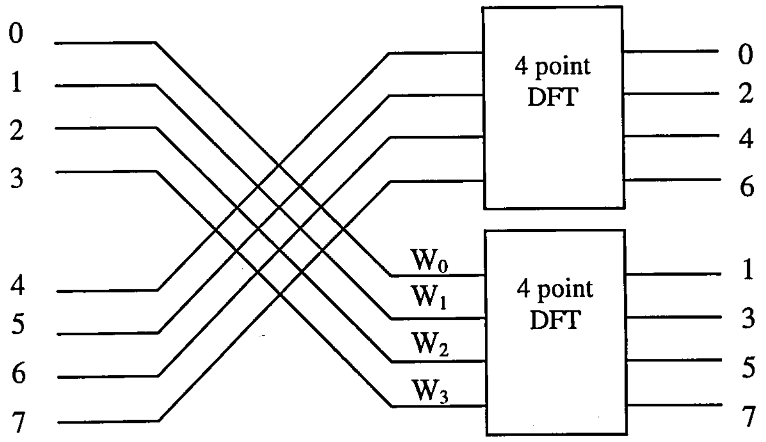

FIGURE 5.2. Decimation-in-frequency FFT algorithm.

In the DIT butterfly the input data is twiddled before the DFT. In the DIF butterfly, the data is twiddled after the butterfly.

5.1.2. Non-radix-2 Algorithms

Radix-2 is by far the most commonly used FFT algorithm, despite the fact that it suffers from the significant limitation of only working for sequence lengths that are integer powers of 2. The FFT algorithm can be expanded for other N`s that are powers of other natural numbers [9, 10, 11].

5.1.3. Split-radix Algorithm

If we investigate the radix-2 decimation-in-frequency flow graph shown in FIGURE 5.2., it can be seen that the even-numbered points of the DFT can be computed independently of the odd-numbered points. This suggests the possibility of using different computational methods for independent parts of the algorithm, with the objective of reducing the number of calculations [11]. The split-radix FFT (SRFFT) algorithm exploits this idea by using both a radix-2 and a radix-4 decomposition in the same FFT algorithm. In this implementation, the even-numbered points require no complex multiplications at all; hence a four-point butterfly at the cost of two complex multiplications. For further information [10, 11, 12, 13] can be referred.

5.1.4. Winograd Algorithm

Winograd presented an algorithm that reduces the number of multiplications below the number required for the FFT. His algorithm builds on Rader's prime-N transform and the Good-Thomas mixed-radix transform [10]. Further information can be found in [14, 15].

Comparison of operations involved in various algorithms are shown in TABLE 5.1 [12, 16].

TABLE 5.1. Computations involved in various FFT algorithms.

	Radix-2		Radix-4		Split-radix		WFTA	
Length	Mul	Add	Mul	Add	Mul	Add	Mul	Add
16	28	148	24	144	20	148		
63							198	1394
64	332	964	264	920	196	964		
252							792	6584
256	2316	5380	1800	5080	1284	5380		
512	5644	12292			3076	12292		

5.2. Implementation

In the implementation of 256-point complex FFT, radix-2 decimation-in-time algorithm is used. Data flow diagram of the implemented DIT algorithm for an 8-point FFT is shown in FIGURE 5.3. In this diagram, each square denotes a butterfly unit. The butterfly unit is depicted in FIGURE 5.4. Each arrow in FIGURE 5.4. means a multiplication. Furthermore, TABLE 5.2 outlines a detailed analysis of the coefficients of a 256-point complex FFT. In TABLE 5.2 **m** is the stage number and **ERF** is the repeat factor for each coefficient.

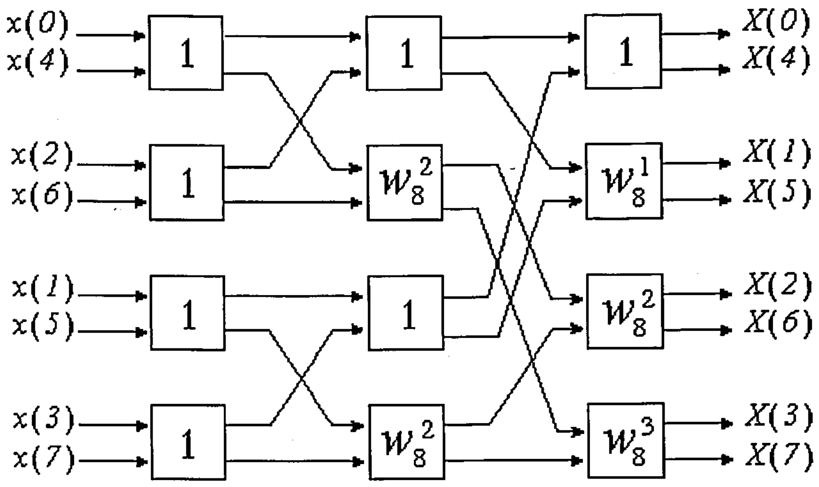

FIGURE 5.3. DIT data flow diagram of the 8-point FFT.

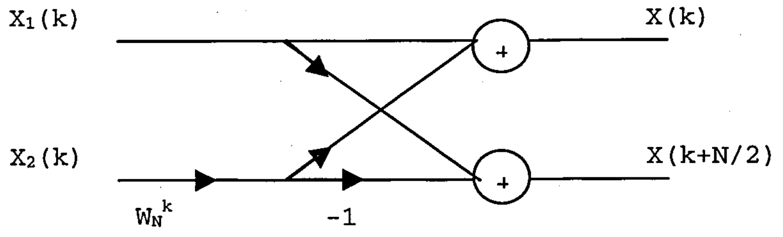

FIGURE 5.4. Butterfly unit.

TABLE 5.2. Coefficients involved in each stage of 256-point FFT.

N	m	ERF	Radix-2 Twiddle Factor Exponents													
256	1	128	0													
	2	64	0	64												
	3	32	0	32	64	96										
	4	16	0	16	32	48	64	80	96	112						
	5	8	0	8	16	24	32	40	48	56	64	72	80	88	96		
	6	4	0	4	8	12	16	20	24	28	32	36	40	44	48	...	124
	7	2	0	2	4	6	8	10	12	14	16	18	20	22	24	...	126
	8	1	0	1	2	3	4	5	6	7	8	9	10	11	12	...	127

The powers of W_N involved in computing the *m*th stage from the (m-1)th stage are given by

$$W(N{:}m{:}Nk/r^m;t) = W_N^{Nk/r^m}, \qquad k = 0,1,2,\ldots,2^{m-1}-1 \quad t = 0,1,2,\ldots,(N/2)-1 \tag{5.6}$$

where t is the butterfly index for each stage. There are $2^{m-1}-1$ distinct twiddle factors per stage, which are repeated according to

$$R = r^{M-m} \text{ MOD } r^{m-1} \tag{5.7}$$

where $M = \log_2(N)$.

Block diagram of the FFT block is shown in FIGURE 5.5.

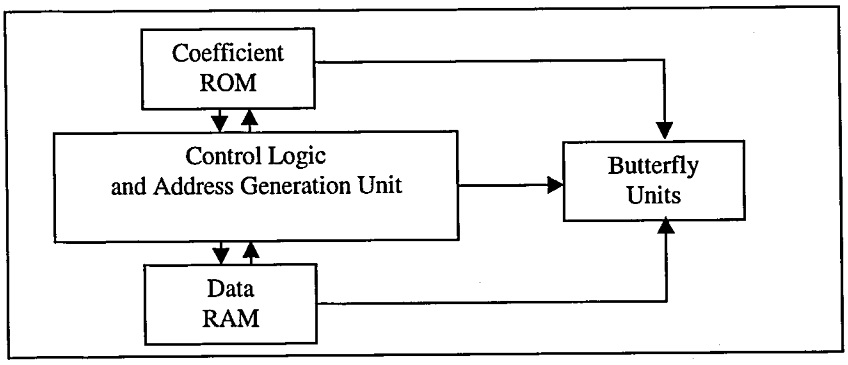

FIGURE 5.5. Block diagram of the FFT unit.

5.2.1. Control Logic

This unit has designed based on Moore type FSM (Finite State Machine). We can summarize the operation in six phases where each phase lasts one clock cycle.

- Reads the first input to the butterfly from RAM and reads coefficient from the ROM.
- Reads the second input to the butterfly from RAM.
- Send inputs to the butterfly unit.

- Waits for butterfly operation to be concluded.
- Writes one of the outputs of the butterfly to the RAM.
- Writes other output to the ROM.

According to the scheme explained above it can be seen that the period of one clock cycle is limited by the half of butterfly operation, which is specifically half of delay of one 16x16 bit multiplication and a 16-bit addition.

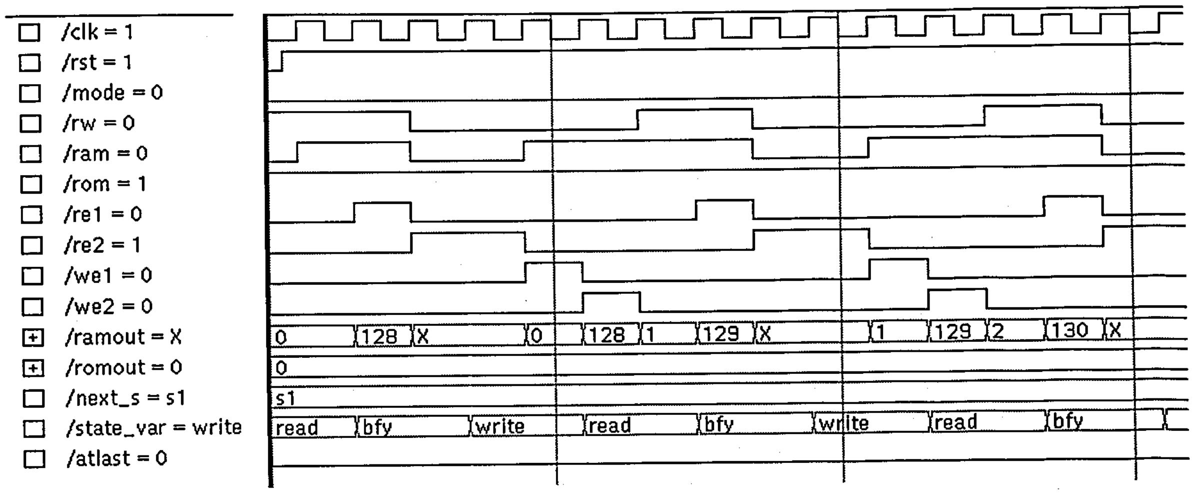

FIGURE 5.6. Addressing of stage 1.

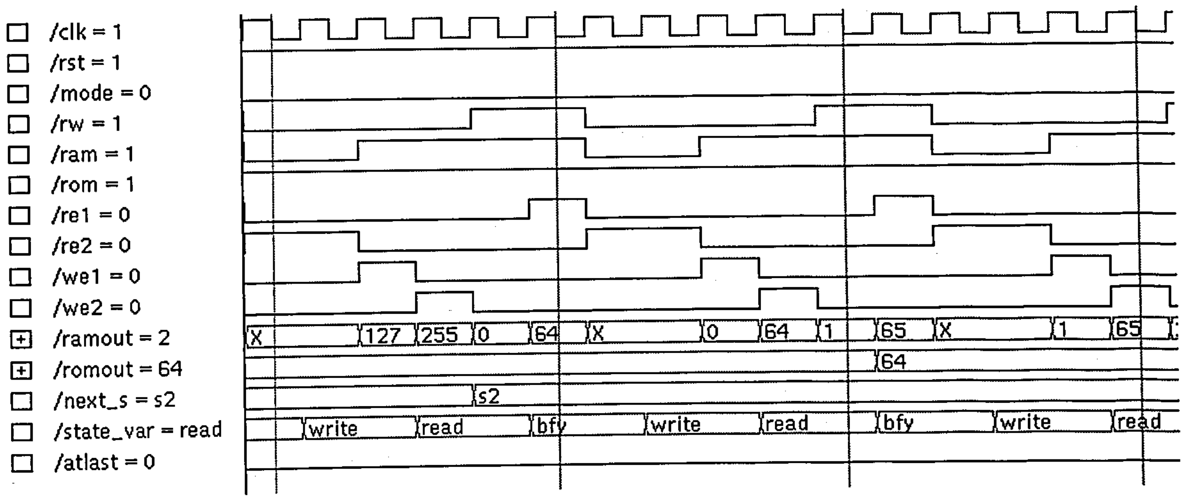

FIGURE 5.7. Transition from stage 1 to stage 2.

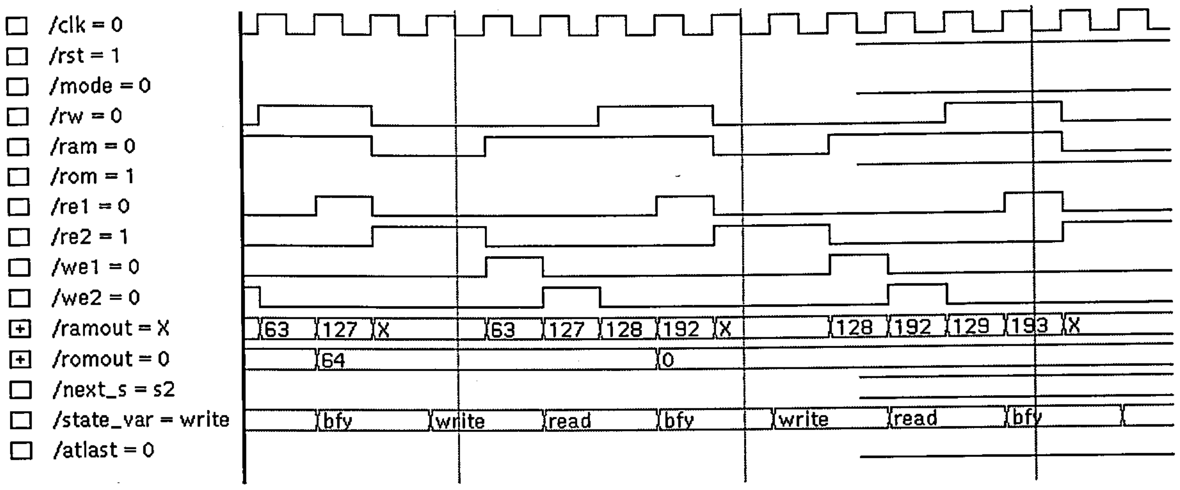

FIGURE 5.8. Transition between two pairs.

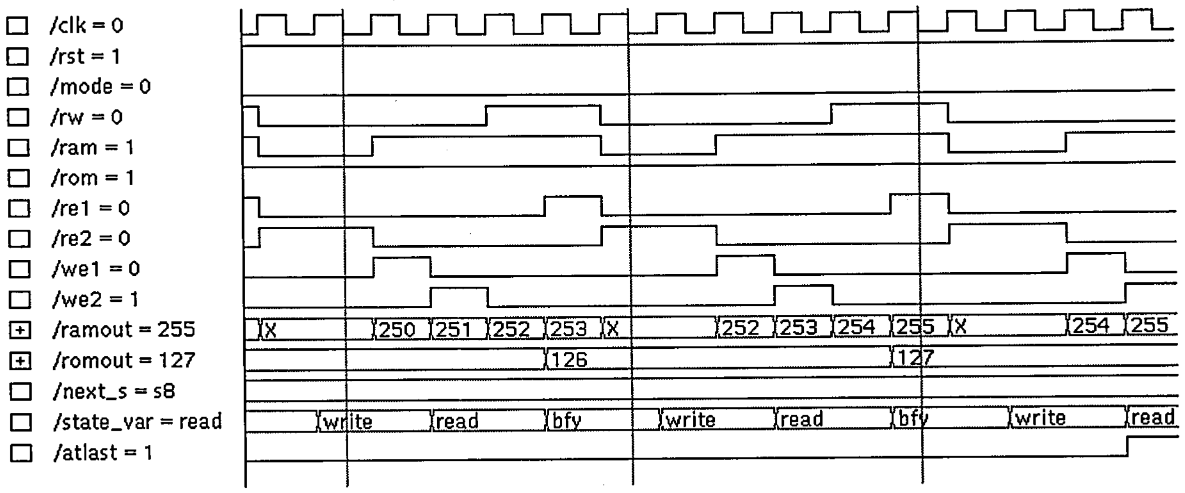

FIGURE 5.9. End of the FFT operation.

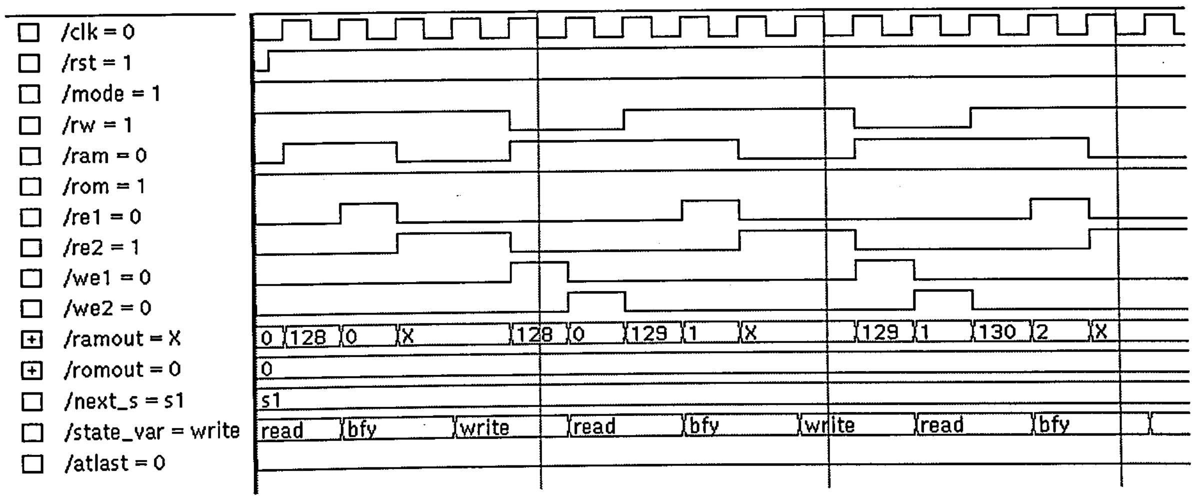

FIGURE 5.10. Addressing in reverse mode.

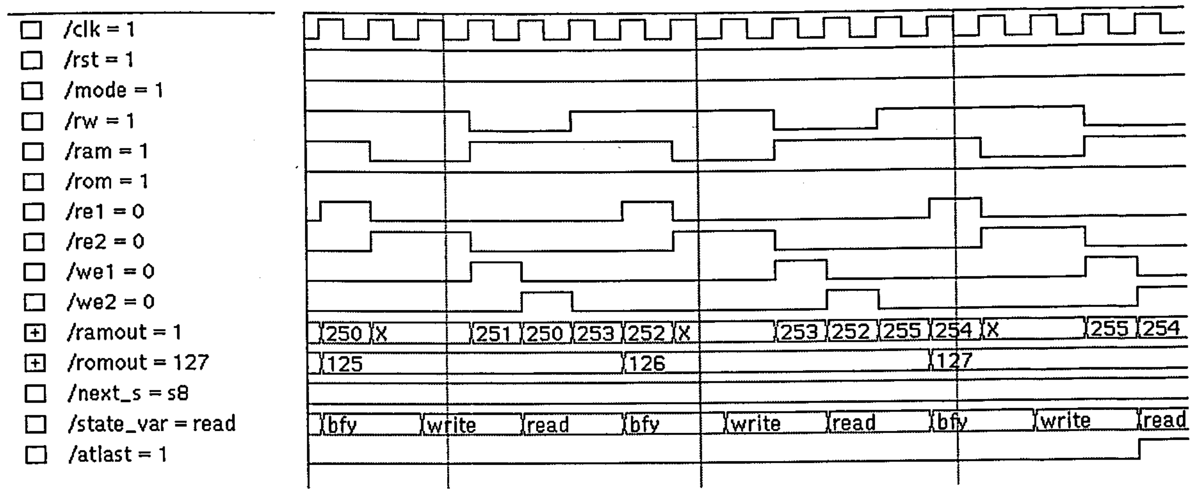

FIGURE 5.11. End of FFT in reverse mode.

Optimization results for control unit of FFT:

```
Number of nets          :        632
Number of cells         :        581
Number of operators     :          7

Combinational area      :     691.51    ( 42 %)
Buffer/Inverter area    :       87.1    (  5 %)
Sequential area         :     578.95    ( 35 %)
Operator area           :     283.86    ( 17 %)
                            -----------
Total area              :    1641.42
```

Critical path is 7.69 nsec.

5.2.2. Butterfly

The implemented butterfly unit is exactly same as the FIGURE 5.4. The simulation results are shown in FIGURE 5.12.

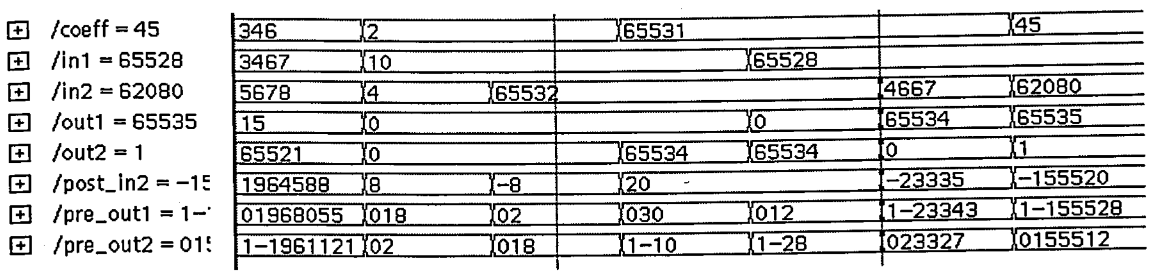

FIGURE 5.12. QHDL simulation for butterfly unit.

Synthesis results are as follows:

```
Number of nets            :       148
Number of cells           :      1036
Number of operators       :         3

Combinational area        :    1809.21    ( 46 %)
Buffer/Inverter area      :     142.04    (  4 %)
Sequential area           :          0    (  0 %)
Operator area             :    1947.25    ( 50 %)
                               -----------
Total area                :     3898.5
```

Critical path is 20.8 nsec.

5.2.3. Overall FFT/IFFT Design

For the overall FFT simulation a test bench is written that instantiates all the blocks needed. The design consists of two butterfly units to calculate real and imaginary parts concurrently. Twiddle factors are generated using MATLAB in 16-bit two's complement format and included in the simulation as ROM's that are addressed by /romad and has the outputs /t_rrom_dout (real part) and /t_irom_dout (imaginary part). The overall FFT design concept can be explored in FIGURES 5.13, 5.14, 5.15, 5.16.

The signals shown in simulation waveforms can be listed as follows:

t_mode: Mode of the operation.
t_cs_ram: Chip select signal for RAMs.
t_cs_rom: Chip select signal for ROMs.
t_atlast: Acknowledge for the end of operation.
t_rram_dout: Output of RAM of the real part of data.
t_iram_dout: Output of RAM of the imaginary part of data.
t_rrom_dout: Output of ROM of the real part of twiddle factor.
t_irom_dout: Output of ROM of the imaginary part of twiddle factor.
t_rram_din: Input to RAM of the real part of data.
t_iram_dout: Input to RAM of the imaginary part of data.
t_ramad: Address of the RAM.
t_romad: Address of the ROM.
t_bfy1_in1: First input of the butterfly unit of the real part.
t_bfy1_in2: Second input of butterfly unit of real part.
t_bfy2_in1: First input of the butterfly unit of imaginary part.
t_bfy2_in2: Second input of the butterfly unit of imaginary part.
t_bfy1_out1: First output of butterfly unit of real part.
t_bfy1_out2: Second output of butterfly unit of real part.
t_bfy2_out1: First output of the butterfly unit of imaginary part.
t_bfy2_out2: Second output of the butterfly unit of imaginary part.

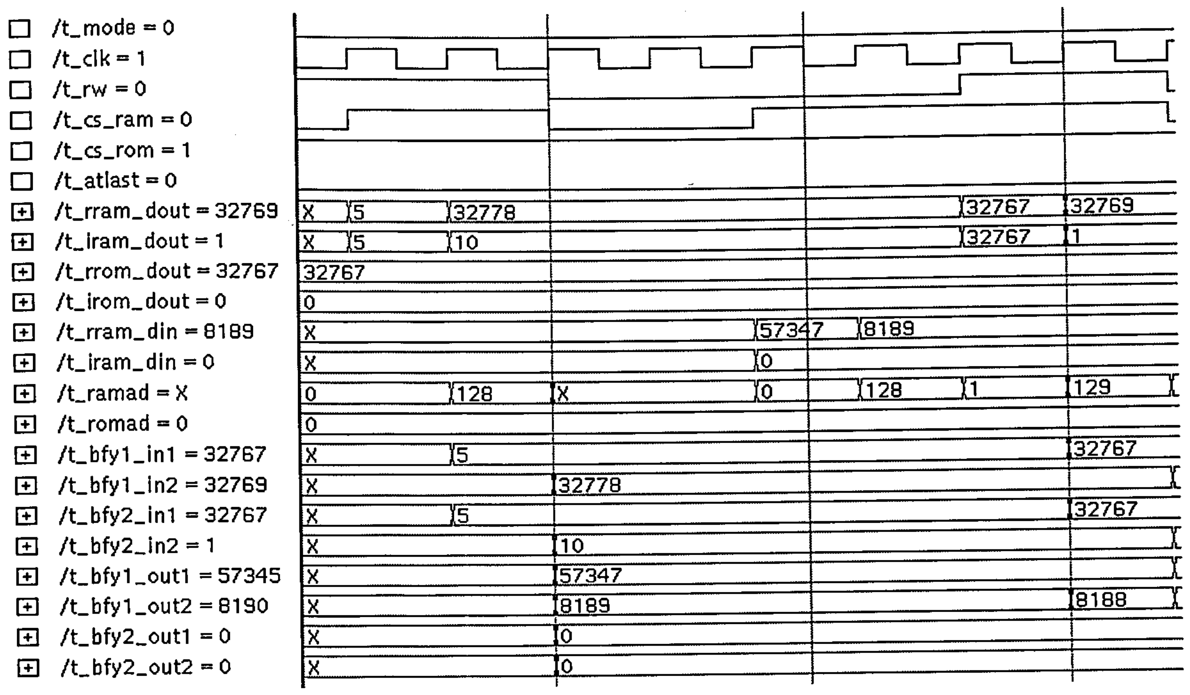

FIGURE 5.13. Overall FFT simulation.

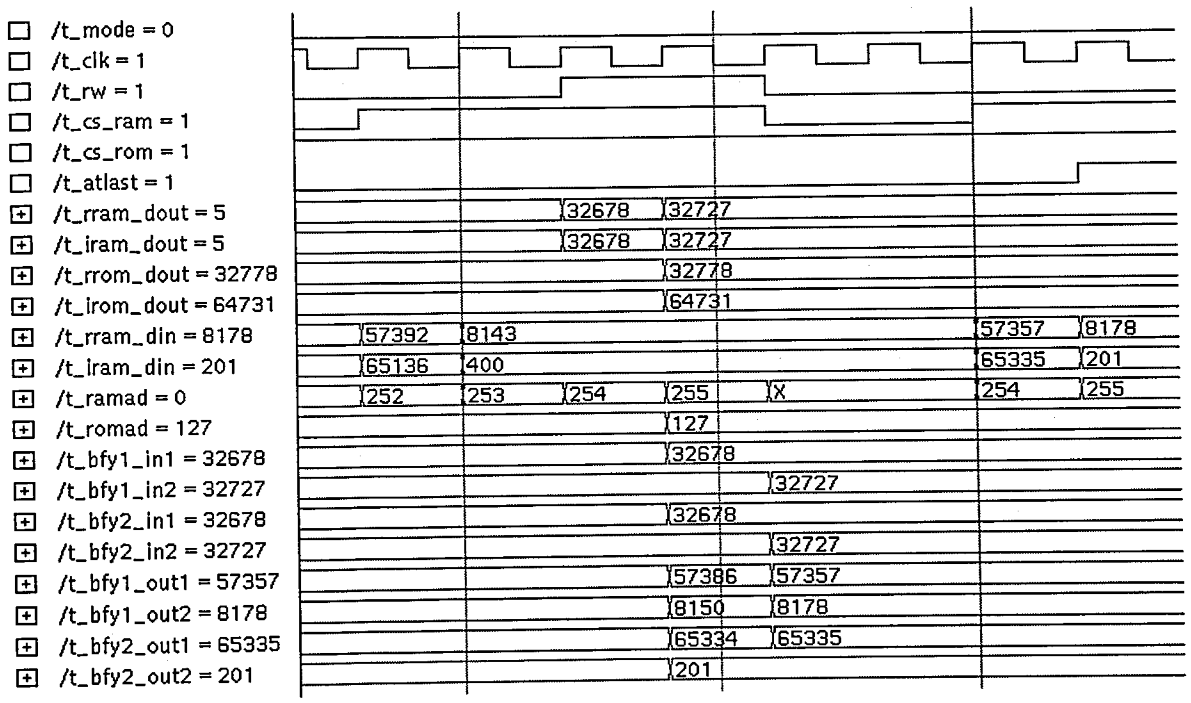

FIGURE 5.14. End of the 256 point complex FFT.

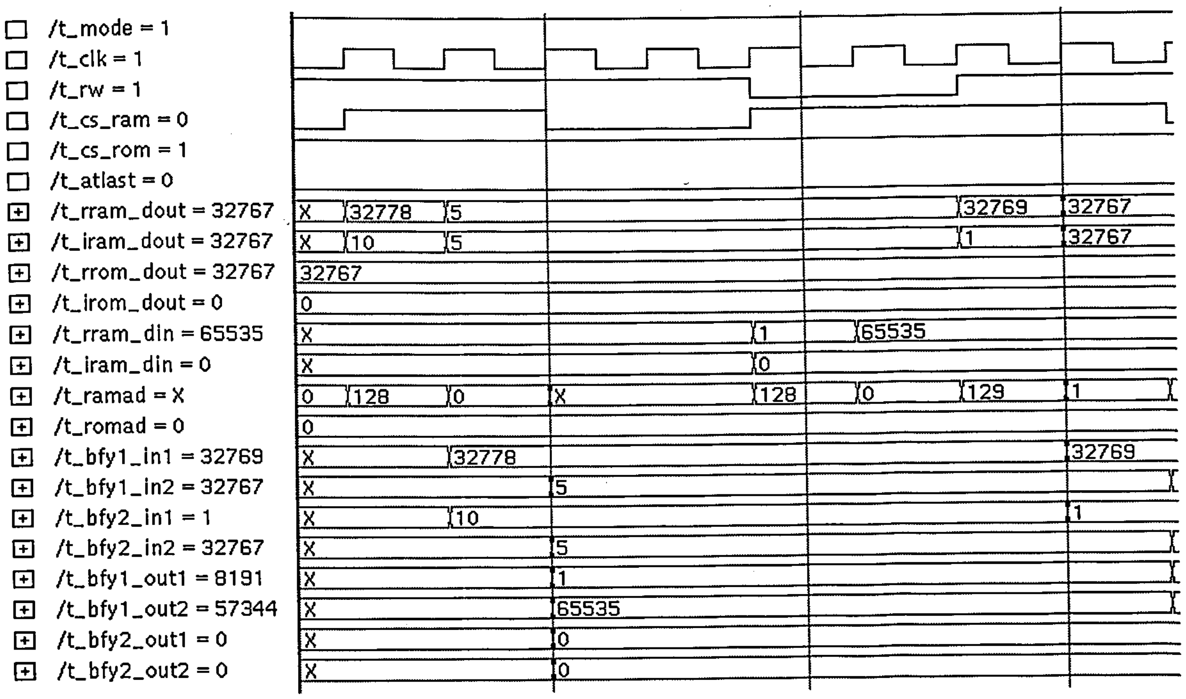

FIGURE 5.15. Overall FFT simulation in reverse order.

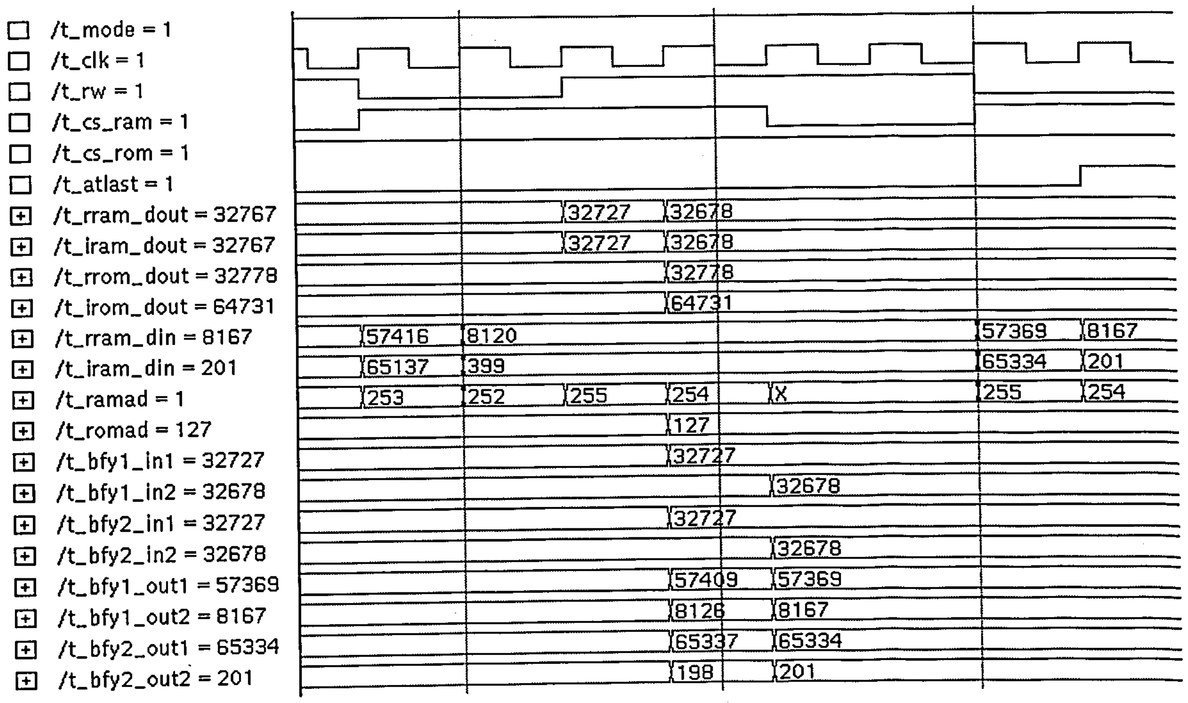

FIGURE 5.16. End of the 256 point complex FFT in reverse order.

6. FIR Filter

This section elaborates the constant coefficient finite impulse response filter implementation without multipliers.

6.1. Theory

In general, an FIR system is described by the difference equation

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k) \qquad (6.1)$$

or, equivalently, by the system function

$$H(z) = \sum_{k=0}^{M-1} b_k z^{-k} \qquad (6.2)$$

Furthermore, the unit sample response of the FIR system is identical to the coefficients $\{b_k\}$, that is,

$$h(n) = \begin{cases} b_n, \ldots\ldots\ldots\ldots\ldots 0 \le n \le M-1 \\ 0, \ldots\ldots\ldots\ldots\ldots\ldots \textit{otherwise} \end{cases} \qquad (6.3)$$

It must be noted that the length of the filter is M. There are several methods for implementing an FIR system. Four methods are as follows: direct form, cascade-form, frequency sampling realization, and lattice realization[11].

We can represent the FIR filter in block diagram form as shown in FIGURE 6.1.

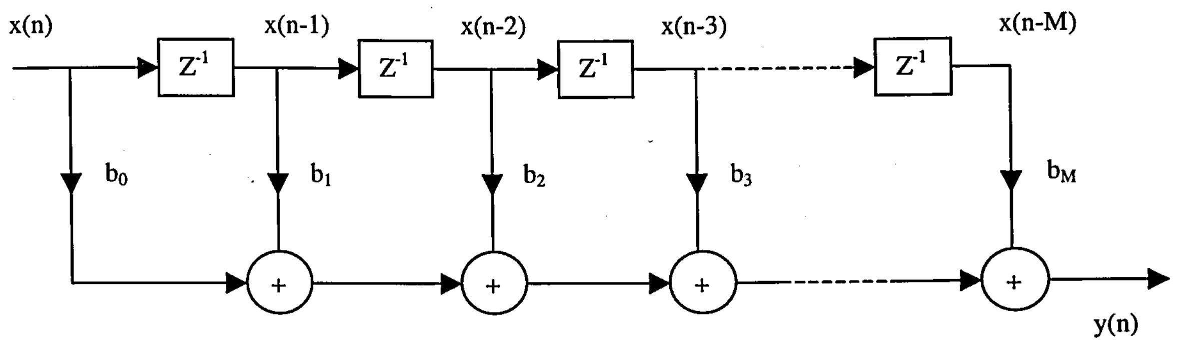

FIGURE 6.1. Data flow diagram of a FIR filter.

Such a diagram, often called a digital filter structure depicts the operations required to compute each value of the output sequence from values of the input sequence [17]. The basic elements of the diagram depict means for addition, multiplication of sequence values by constants (constants indicated on the branches imply multiplication), and storage of past values of the input sequence. Thus the block diagram gives a clear indication of the complexity of the system.

6.2. Implementation

As it can been seen from (6.1) and (6.2) realization of FIR filters requires multiplications. However, multipliers occupy huge areas. If the FIR system is a constant filter, it is usually implemented without multipliers where multiplications are represented by a series of adders and/or subtractors and shifters. This kind of multiplier representation is very effective in terms of area, delay, and power compared to general multipliers [18].

Even though systems without multipliers are cost effective than general ones, it is always required to make these systems even more effective. In these systems, number of "one's" in the representation of the constants determines the number of adders required in the implementation. Thus, if one can reduce the number of "one's" in the representation, it is possible to reduce the number of adders, which generally achieved by using Canonic Signed Digit (CSD) representation. It is proved that CSD representation requires, on average, 33 percent fewer adders than standard binary representation [19]. Also, CSD representation has the nice property of standard binary representation that is uniqueness [20].

A 6-tap symmetrical coefficient FIR filter that can be used for smoothing the FFT spectrum, was designed using MATLAB. TABLE 6.1 summarizes the representations of the coefficients normalized to signed 16-bit input format. For CSD representation $\underline{1}$ denotes -1.

TABLE 6.1. Representation of coefficients in various number systems.

Coefficient	Decimal	2's complement	CSD
1	-236	1111111100010100	0000000$\underline{1}$00010100
2	-436	1111111001001100	000000$\underline{1}$001010$\underline{1}$00
3	17056	0100001010100001	0100001010100000

Block diagram of the FIR filter is illustrated in FIGURE 6.2.

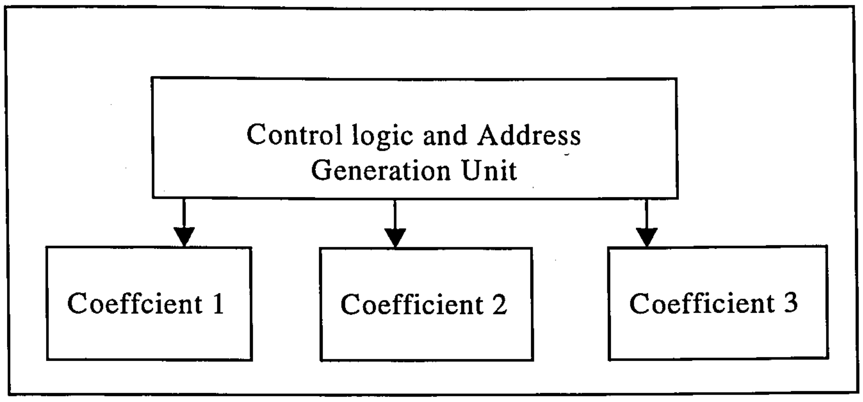

FIGURE 6.2. Block diagram of the FIR unit.

6.2.1. Control Logic

Control logic in FIR maintains the correct address generation for RAM and scheduling the dataflow of the inputs to the coefficient units in order to achieve (6.2).

6.2.2. Coefficient 1

Coefficient 1 as well as other two coefficients are implemented using CSD representations of the coefficients without multipliers. FIGURE 6.3. shows the QHDL simulation result for coefficient 1 in decimal.

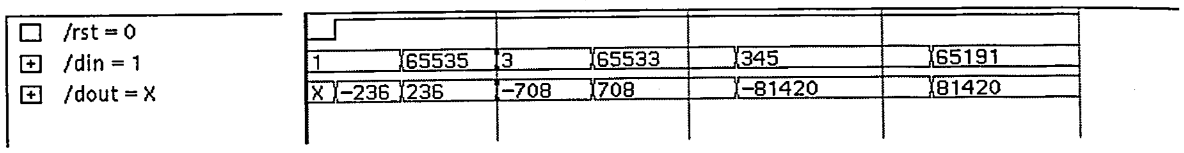

FIGURE 6.3. VHDL simulation waveform for coefficient 1.

Optimization results are as follows :

Number of nets	:	86
Number of cells	:	167
Number of operators	:	2

```
Combinational area      :     235.02   ( 45 %)
Buffer/Inverter area    :      40.87   (  8 %)
Sequential area         :       3.67   (  1 %)
Operator area           :     244.63   ( 47 %)
                           -----------
Total area              :     524.19
```

Total combinational delay is 8.68ns.

6.2.3. Coefficient 2

QHDL simulation result is shown in FIGURE 6.4 for this coefficient.

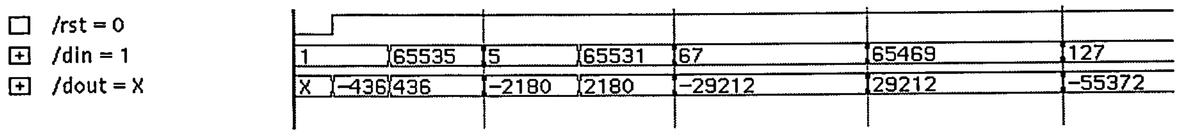

FIGURE 6.4. VHDL simulation waveform for coefficient 2.

Optimization results are as follows :

```
Number of nets          :        109
Number of cells         :        208
Number of operators     :          3

Combinational area      :     313.75   ( 44 %)
Buffer/Inverter area    :      54.27   (  8 %)
Sequential area         :       3.67   (  1 %)
Operator area           :     335.43   ( 47 %)
                           -----------
Total area              :     707.12
```

Total combinational delay is 11.752 nsec.

6.2.3. Coefficient 3

QHDL simulation result can be shown in FIGURE 6.5. which is exactly same as the one listed in TABLE 6.1 since there is no $\underline{1}$ in the CSD representation.

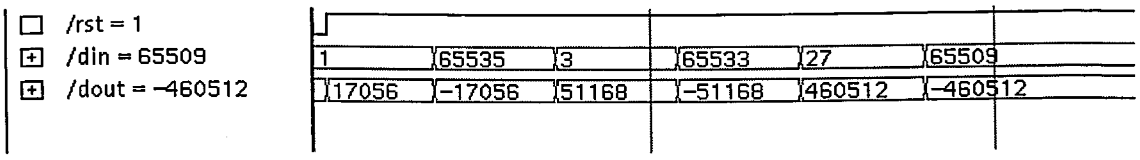

FIGURE 6.5. VHDL simulation waveform for coefficient 3.

Optimization results are as follows :

```
Number of nets            :        110
Number of cells           :        221
Number of operators       :          3

Combinational area        :     309.26   ( 48 %)
Buffer/Inverter area      :      32.16   (  5 %)
Sequential area           :          0   (  0 %)
Operator area             :     306.84   ( 47 %)
                             -----------
Total area                :     648.26
```

Total combinational delay is 11.012 nsec.

6.2.4. Overall FIR filter

For overall FIR filter simulation a test bench entity is used. FIGURE 6.6.

The signals shown in simulation waveforms can be listed as follows:

t_ram_dout: Output of the data RAM.
t_ram_adres: Address of the data RAM.
t_result: Result of the filtering.
t_coeff1: Result of multiplication with first coefficient.
t_coeff2: Result of multiplication with second coefficient.
t_coeff3: Result of multiplication with third coefficient.

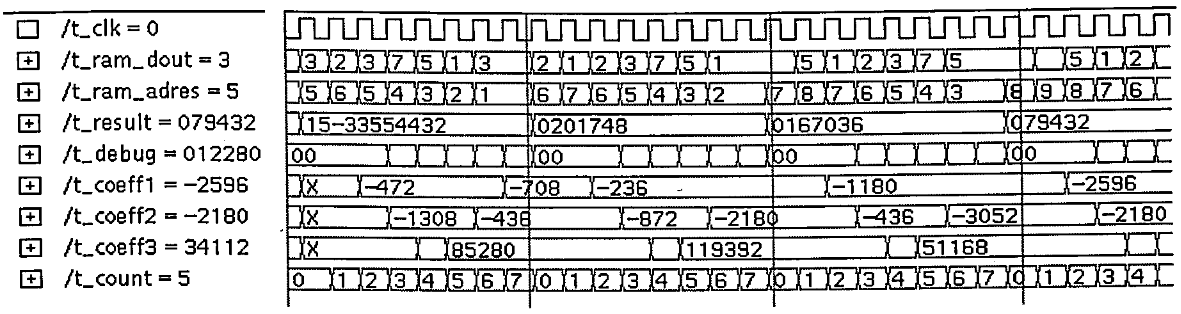

FIGURE 6.6. Overall FIR filter simulation waveform.

Optimization results are as follows :

```
Number of nets              :        984
Number of cells             :       1260
Number of operators         :         11

Combinational area          :    1554.53    ( 36 %)
Buffer/Inverter area        :     235.17    (  5 %)
Sequential area             :    1353.41    ( 31 %)
Operator area               :    1159.44    ( 27 %)
                              -----------
Total area                  :    4302.55
```

Delay on output data is 26.03 nsec.

7. DIVISION AND SQUARE-ROOT

In general computation systems, the frequency of division and square-root operations tends to be considerably lower than that of addition and multiplication. Hence, many arithmetic processors do not include hardware support for these operations [21]. Software routines for division and square-root can be up to an order of magnitude slower than multiplication and addition and as a result, the time weighted impact of division and square-root on processor utilization can be significant [21]. However, therefore it may be needed to implement these units in hardware for some applications.

7.1. Divider

Division of two binary numbers can be easily implemented by using shifters and a subtractor. However, this approach is rather slow and inconvenient for many DSP applications. Real-time digital signal processing requires high performance implementation of division. This can only be achieved by the design of fast and efficient algorithms which address practical VLSI architectural design issues [21].

There are basically two kind of division algorithms, restoring and non-restoring. If we assume two numbers A and B then the implementation of the restoring division is as follows:

- Always do A - B, and add B back again if the result is negative.
- On the next step subtract B/2 (shifted B).

However an optimization can be applied to this scheme in the way that instead of adding B back, add B/2 if the dividend A is negative. This is called non-restoring division. It must noted that the quotient gets a "1" bit when the result is positive, a "0" bit otherwise.

To illustrate the non-restoring division, following example can be used.

130 / 11: A = 130 = 10000010, B = 11 = 00001011

A	B'	A-B' or A+B'	Q
10000010	10110000	11010010	00000000
11010010	01011000	00101010	00000001
00101010	00101100	11111110	00000010
11111110	00010110	00010100	00000101
00010100	00001011	00001001	00001011

Q = 00001011 = 11, R = 00001001 = 9

For the division algorithm the one presented in [22] is used. It is specifically a non-restoring two's complement division algorithm.

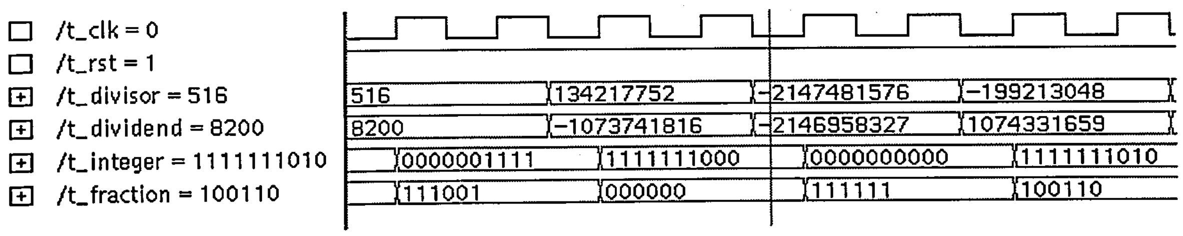

FIGURE 7.1. Simulation results for divider block.

Optimization results are for divider:

Number of nets	:	1268
Number of cells	:	2554
Number of operators	:	17

```
Combinational area      :    4039.17   ( 53 %)
Buffer/Inverter area    :      388.6   (  5 %)
Sequential area         :     101.44   (  1 %)
Operator area           :    3157.59   ( 41 %)
                          ------------
Total area              :     7686.8
```

Critical path is 3.2 nsec.

7.2. Square-root

We have implemented a novel square-root algorithm. This algorithm is iterative and multiplication based. Let's represent an m bit binary number in the form $(NSP)_{m/2}(SP)_{m/2}$. Then, algorithm has the flowchart shown in FIGURE 7.2.

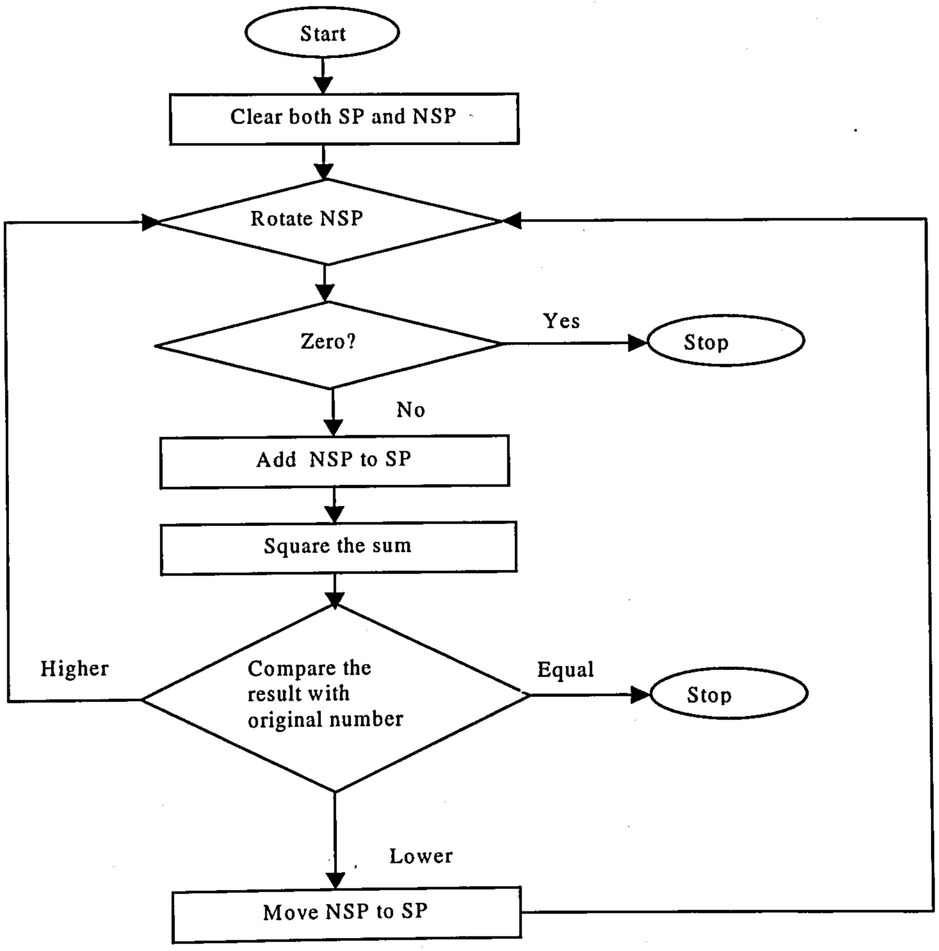

FIGURE 7.2 Flowchart of the square-root algorithm.

This square-root algorithm has been implemented in a parameterized manner to be used with variable input word lengths. FIGURE 7.3. shows some of the results for some square-root operations.

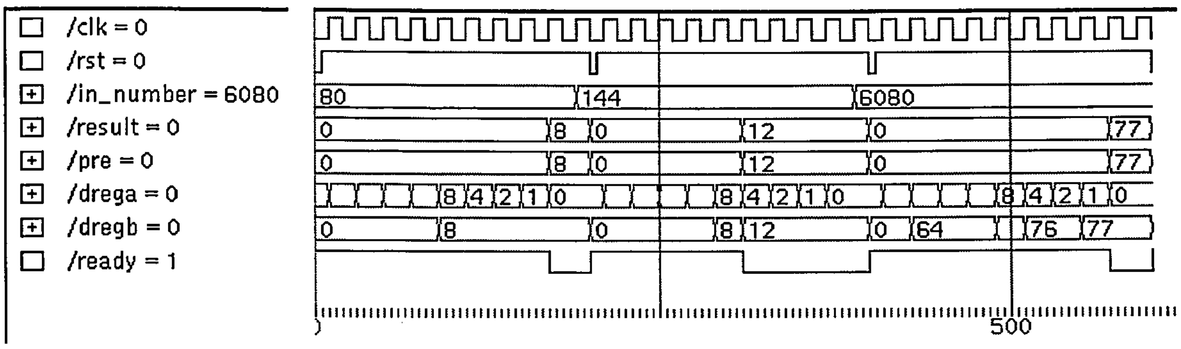

FIGURE 7.3. VHDL simulation waveform for square-root.

Below optimization results can be found.

```
Number of nets           :      340
Number of cells          :     1265
Number of operators      :        3

Combinational area       :   1953.42    ( 41 %)
Buffer/Inverter area     :     247.9    (  5 %)
Sequential area          :    488.54    ( 10 %)
Operator area            :    2055.2    ( 43 %)
                            -----------
Total area               :   4745.06
```

Total combinational delay is 8.02 nsec.

8. MULTIPLY ACCUMULATE UNIT

Multiply accumulate unit (MAC) is one of the most critical units in a multi-purpose digital signal processor as well as in a digital signal processing ASIC due to the nature of the DSP algorithms. Hence, we have implemented a parametric signed MAC that can be used with variable input widths. FIGURE 8.1 shows the QHDL simulation results.

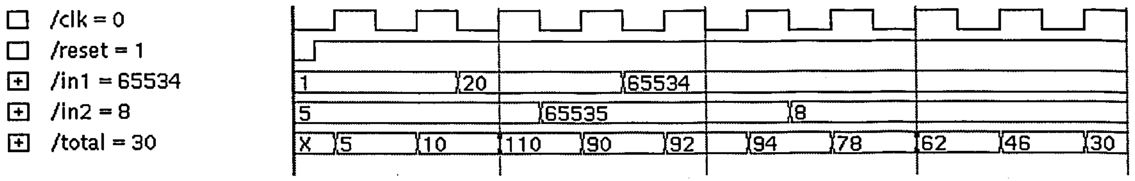

FIGURE 8.1. VHDL simulation waveform for MAC.

Synthesis and optimization results for the MAC are as follows:

```
Number of nets            :       162
Number of cells           :       967
Number of operators       :         2

Combinational area        :   1682.09    ( 45 %)
Buffer/Inverter area      :    101.17    (  3 %)
Sequential area           :    202.88    (  5 %)
Operator area             :   1783.26    ( 47 %)
                            -----------
Total area                :    3769.4
```

Critical timing path is 4.72 nsec.

9. OVERLAP-ADD METHOD

In practical applications involving linear filtering of signals, the input sequence x(n) is often a very long sequence. This is especially true in some real-time signal processing applications concerned with signal monitoring and analysis [11].

However in digital domain, it is not possible to process very long sequence of data due to hardware limitations. Therefore, to cope with the block processing of the data there are several methods. The two common methods are overlap-add and overlap-save [11, 17].

In our implementation, overlap-add method is utilized which basically performs (9.1).

$$\text{outbuf}(N/2) = \text{outbuf}(N/2) + \text{input\_to\_block}(1:N/2). \quad (9.1)$$

FIGURE 9.1. shows the simulation result for this block.

/final: Output of the block.

/frout: Output of the shift registers.

/dout: Output of the RAM.

/gx: Address of the RAM.

/gwr: Read/write signal for the RAM.

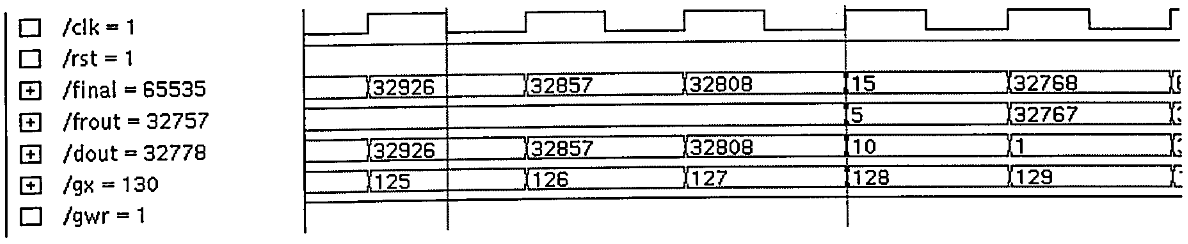

FIGURE 9.1. Simulation waveforms for the overlap-add.

Optimization results for this block is as follows:

```
Number of nets          :        87
Number of cells         :      2141
Number of operators     :         2

Combinational area      :    123.19    (  1 %)
Buffer/Inverter area    :     10.72    (  0 %)
Sequential area         :     13035    ( 98 %)
Operator area           :     118.6    (  1 %)
                          -----------
Total area              :   13287.6
```

Another method to implement overlap-add method can be to address the data that is resident in the RAM in an appropriate way to avoid shift register which consumes a lot of silicon area.

10. CONCLUSION

In this thesis, a high quality reduced complexity speech enhancement algorithm that is suitable for both software and hardware implementation is introduced. In addition, common DSP blocks, which are also involved in the proposed algorithm are implemented using VLSI design techniques.

The complexity of the MWF algorithm is successfully reduced. This modification, which consists of replacing LPC with a smoothing low pass filter, results an important advantage on software and hardware implementation if the computational burden of LPC shown in TABLE 10.1 considered [17] where N is the number of data points in the analysis and p is the order of predictor coefficients. In addition, a floating point operation count is made using MATLAB on a Pentium-III 500MHz computer running Linux and 17396 flops obtained for LPC method whereas 3072 flops for FIR filter method.

TABLE 10.1. Computational considerations in LPC solutions.

		METHOD		
		Covariance	**Autocorrelation**	**Lattice**
Storage	Data	N	N	3N
	Matrix	$\propto p^2/2$	$\propto p$	-
	Window	-	N	-
Multiplication	Windowing	-	-	-
	Correlation	$\propto Np$	$\propto Np$	-
	Matrix Solution	$\propto p^3$	$\propto p^2$	$5Np$

In the design methodology, new tools like QHDL and AutoLogicII are introduced. In addition, parameterized VHDL coding techniques as well as synthesizable IEEE libraries are employed for most of the blocks.

In the design of a window function and overlap-add method, a novel but a simple architecture that uses a shift register is introduced. The generic design of these blocks enables these implementations to be used in various designs.

Since being one of the major building blocks of many DSP algorithms, many implementations of FFT can be found. Although we did not try to make it very fast since our blocks are mostly concentrated on speech processing it may be useful to compare our design with some others. TABLE 10.2 summarizes the characteristics of some FFT processor designs.

TABLE 10.2. Comparison of various FFT designs

DESIGN	Technology	Datapath Width	Execution Time(1024 pt)	Equivalent Gates	Area (mm^2)
PDSP16510A (Plessey)	1.4um	16	98usec	≈26k	22
ETH, Zurich	0.5um	32	80usec	≈250k	167
Inventra Mentor	0.7um	20	90usec	27K	≈20
Our FFT	0.7um	16	≈400usec	≈12K	≈9

In the implementation of the FIR filter multiplierless design techniques to exploit the constant coefficients are illustrated.

A fully generic divider in terms of input word length and output format has been accomplished. This implementation gives great flexibility to division, thus can be used in many various applications.

A novel square-root algorithm is designed. The optimization results show that a full custom design will be able to compete with many proposed fast square-root architectures.

The summary of all the implemented blocks in this thesis can be found in TABLE 10.3.

TABLE 10.3. Summary of the designed blocks.

Block	**Number of cells used**	**Area (mm^2)**	**Method**
Windowing	3042	7.5	Shift register and multiplier
FFT	12k	9	Radix-2 DIT
FIR filter	1260	2	Constant coefficient without multipliers
Divider	2554	3.8	Non-restoring
Square-root	1265	2.4	Multiplicative
MAC	967	1.9	-
Overlap-add	2141	6.5	Shift register and adder

As stated in section 3 layouts of the designed blocks are generated using IC Station.

Consequently, this thesis addresses important points in the wireless communication and system-on-a-chip era. While the proposed algorithm can be used in many demanding mobile communication applications, the designed blocks can be part of any embedded application specific integrated circuit.

APPENDIX

EXAMPLE LAYOUT

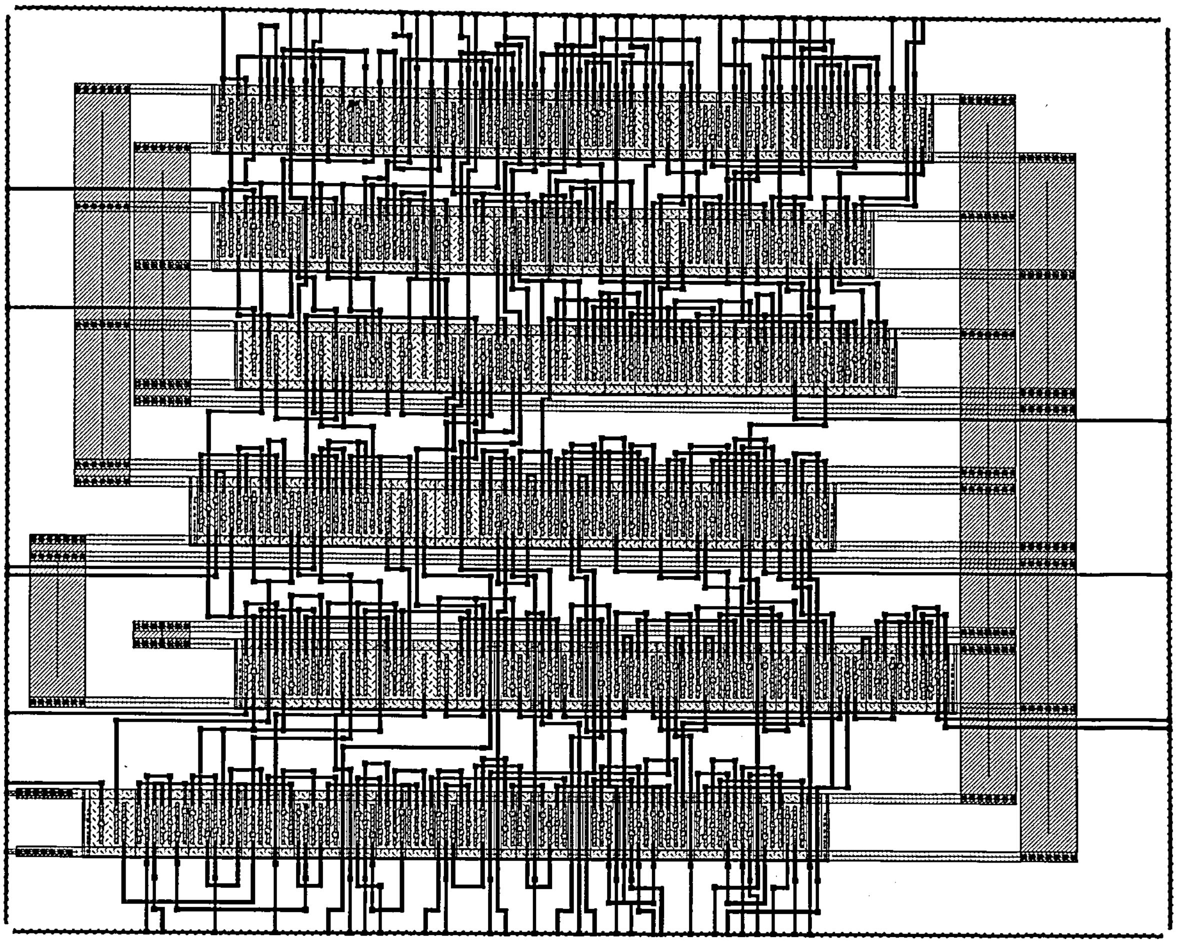

FIGURE A.1. Layout of the 16-bit adder circuit.

REFERENCES

1. Parhi, K. K., *VLSI Digital Processing Systems*, Wiley, New York,1999.

2. Lim, J. S. (editor), *Speech Enhancement*, Prentice Hall, NJ, 1983.

3. Boll, S.F., "Suppression of acoustic noise in speech using spectral subtraction," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-27, No. 2, pp. 113-120, April 1979.

4. Lim, J. and A. V. Oppenheim, "All-pole modeling of degraded speech," *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-26, No. 3, pp. 197-210, June 1978.

5. Ephraim, Y., "A Minimum mean square error approach for speech enhancement," *Proc. IEEE Int. Conf. Acoust, Speech Signal Processing*, pp. 829-832, 1990.

6. Arslan, L., A. McCree, and V. Viswanathan, "New methods for adaptive noise suppression," *IEEE Proceedings of ICASSP*, Vol. 1, pp. 812-815, 1995.

7. Romdhane, M. S. Ben, Madisetti, K. Vijay, john W. Hines, *Quick-Turnaround ASIC Design in VHDL*, Kluwer Academic Publishers, Boston, 1996.

8. Jackson, Leland B., *Digital filters and signal processing : with MATLAB exercises*, Kluwer Academic Publishers, Boston, 1996.

9. Bateman, Andrew, and Warren Yates, *Digital Signal Processing Design*, Computer Science Press, New York, 1989.

10. Haddad, Richard A., and Thomas W. Parsons, *Digital signal processing : theory, applications, and hardware*, Computer Science Press, New York, 1991.

11. Proakis, John G., and Dimitris G. Manolakis, *Digital signal processing : principles, algorithms, and applications*, Macmillan, New York, 1992.

12. Richards, Mark A., "On Hardware Implementation of the Split-Radix FFT," *IEEE Trans. on Acoust, Speech, and Signal Processing*, Vol 36, No. 10, pp. 1575-1581, 1988.

13. Duhamel, P., and H. Hollmann, "Split radix FFT algorithm," Elect. Letters, Vol. 20, No. 1, pp. 14-16, 1984.

14. Winograd, S., "On computing the discrete Fourier transform," *Math. Comput.*, Vol. 32, pp. 175-199, 1978.

15. Macleod, M. D., N. L. Bragg, "Fast Hardware Implementation of the Winograd Fourier Transform Algorithm," *Elect. Letters*, Vol. 19, No. 2, pp. 363-365, 1983.

16. Burrus, C. Sidney, and Peter W. Eschenbacher, "An In-Place, In-Order Prime Factor FFT Algorithm," IEEE Trans. on Acoust, Speech, and Signal Processing, Vol. Assp-29, No. 4, pp. 806-811, 1981.

17. Rabiner, Lawrence R., and Ronald W. Schafer, *Digital Processing of Speech Signals*, Prentice Hall, New Jersey, 1978.

18. Yurdakul, Arda, "Development of a High Level Synthesis tool Speciallized on FIR-Based Multirate Systems," PhD. Dissertation, Bogaziçi University, 1999.

19. Garner, H. L., "Number systems and aritmetic," *Advanced Computers*, Vol. 6, pp 131-194, 1965.

20. Hwang, K., *Computer Arithmetic: Principles, Architectures and Design*, John Wiley & Sons, NY, 1979.

21. McCanny, J. V. and S. E. McQuillan, "Fast VLSI Algorithms for Division and Square Root," *Journal of VLSI Signal Processing*, Vol. 8, pp. 151-158, 1994.

22. Keshab K. Parhi, "A systematic Approach for Design of Digit-Serial Signal Processing Architectures," *IEEE Trans. on Circuits and Systems*, Vol. 38, No. 4, April 1991.

REFERENCES NOT CITED

Borgatti, M., Felici, M., Ferrari, A., and R. Guerrieri, "A Low-Power Integrated Circuit for Remote Speech Recognition," *IEEE J. of Solid-State Circuits*, Vol. 33, No. 7, July 1998.

Wang, C.-C., Huang, C.-J., and G.-C. Lin, "A Chip Design of Radix-4/2 64b/32b Signed and Unsigned Integer Divider Using Compass Cell Library," *Electronics Letters*, Vol. 32, No. 17, pp. 1526-1527, 1996.

El-Sharkawy, Mohammed, *Real Time Signal Processing Applications with Motorola's DSP56000 Family*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

Hansen, J.H.L. and M. A. Clements, "Constrained iterative speech enhancement with application to speech recognition," *IEEE Trans. on Signal Processing*, Vol. 39, No. 4, pp. 795-805, 1991.

Despain, A. M., "Very fast Fourier transform algorithms hardware for implementation", *IEEE Trans. on Computers*, Vol. C-28, No. 5, pp. 333-340, May 1979.

Baas, B. M., "A Low-Power, high-performance, 1024-Point FFT processor", *IEEE Journal of Solid-State Circuits*, Vol. 34, No. 3, pp.380-387, March 1999

Cooley J. W. and J. W. Tukey, "An Algorithm for the Machine Calculation of Complex Fourier Series," *Math. Computing*, Vol. 19, pp. 297-301, April 1965.