ZERO CROSSING BASED

SPECTRAL ANALYSIS

A Dissertation Submitted in Candidature
for the Degree of

MASTER OF PHILOSOPHY

of the University of Wales

by

JOHN A SHERRINGTON

Faculty of Technology,
Gwent College of Higher Education

February, 1996

GCHE

GWENT COLLEGE OF HIGHER EDUCATION

DECLARATION:

This work has not previously been accepted in any substance for any degree and is not concurrently submitted in candidature for any degree

Signed ..... J.A. Sherrington ..... (Candidate)

Date ..... 29/2/96 .....

STATEMENT 1:

This thesis is the result of my own investigations, except where otherwise stated.

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed ..... J.A. Sherrington ..... (Candidate)

Date ..... 29/2/96 .....

STATEMENT 2:

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... J.A. Sherrington ..... (Candidate)

Date ..... 29/2/96 .....

NB: *Candidates on whose behalf a bar on access has been approved by the University (see Appendix 2), should use the following version of Statement 2:*

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, **after expiry of a bar on access approved by the University of Wales on the special recommendation of the Constituent/Associated Institution.**

Signed ..... (Candidate)

Date .....

ACKNOWLEDGEMENTS

To my wife Dorothy who has shown great patience and support during the long hours of my studies and to Gurvinder Singh Baicher, Dr Hefin Rowlands and Dr Geoff Roberts for their advice and encouragement.

ZERO CROSSING BASED SPECTRAL ANALYSIS

SUMMARY

This dissertation contains a study of the timed zeros, both real and complex, of a band-limited signal. Mathematics essential to the study is explained and the method of using a simple invertible transform to convert complex zeros to real zeros, otherwise known as zero crossings, is considered. The work of key researchers in the field is reviewed giving a brief historical background.

A practicable means of reconstructing a signal from zero crossings times is devised using the Texas Instruments TMS320C30 floating point digital signal processor simulator. Four methods of spectral analysis using zero crossing times are also reviewed and a practicable scheme using the same processor is devised. Spectral analysis of a reconstructed signal from zero crossings is shown to be the best of the methods tried.

The reconstruction of a signal from zero crossing times is also shown to be an alternative means of analogue to digital conversion and a method of compressing data representing a band-limited signal by a factor of two is suggested.

ZERO CROSSING BASED SPECTRAL ANALYSIS

TABLE OF CONTENTS

ZERO CROSSING BASED SPECTRAL ANALYSIS

TABLE OF CONTENTS

ZERO CROSSING BASED SPECTRAL ANALYSIS

TABLE OF CONTENTS

ZERO CROSSING BASED SPECTRAL ANALYSIS

TABLE OF CONTENTS

ZERO CROSSING BASED SPECTRAL ANALYSIS

TABLE OF CONTENTS

ZERO CROSSING BASED SPECTRAL ANALYSIS

LIST OF FIGURES

ZERO CROSSING BASED SPECTRAL ANALYSIS

LIST OF FIGURES

Page No

ZERO CROSSING BASED SPECTRAL ANALYSIS

LIST OF TABLES

Page No

CHAPTER 1 INTRODUCTION

1.1 INTRODUCTION

Spectrum analysers are widely used in signal processing applications and many schemes are available. The conventional elements for digital spectral analysis of a signal are an analogue anti-aliasing filter followed by an analogue to digital converter (A/D) to represent the signal as a set of numbers in digital form followed by a fast Fourier transform which represents the spectrum as the output which in turn must be converted from its digital form to analogue form for the purpose of displaying the desired result. The anti-aliasing filter is essential for correct sampling according to the sampling theorem, which states that the sampling frequency must be greater than twice the maximum frequency contained within the signal to be sampled. The signal to be converted must therefore be band-limited.

The A/D process relies upon multilevel quantisation of samples at the sampling rate where the width of the sampling pulse cannot be infinitely small and the number of levels of quantisation cannot be infinitely great which subsequently leads to quantisation errors. The process thus suffers from inherent inaccuracies and requires sophisticated hardware to produce the desired conversion in as short a time as possible.

It has been well established [2], [5], [6], [10] that the timed zeros of a band-limited signal are informational attributes of that signal and can be used to both reconstruct the original signal and directly determine the discrete Fourier coefficients which

yields the spectrum of the signal. If the timed zeros can be measured with simple hardware then it is an attractive proposition to investigate the possibility of eliminating the analogue to digital conversion process since measuring time is inherently simpler and more accurate than multilevel quantisation.

1.2 AIMS AND OBJECTIVES

The purpose of this dissertation is to investigate the nature of timed zeros of band-limited signals and review the work of other researchers to gain as wide an insight into the mathematics of the subject as possible. Once the general principles are understood, practicable schemes for signal reconstruction and spectral analysis from zero crossing times are to be devised using the TMS320C30 floating point digital signal processor simulator.

1.3 ENVIRONMENT AND TOOLS

In order to study zero crossing techniques several tools are required and all have to work within the chosen environment. A system for demonstrating various digital signal processing functions has been developed at Gwent College of Higher Education (GCHE) using the TMS320C25 fixed point processor simulator working in the Microsoft Disk Operating System (MS-DOS) environment. The Lotus 1-2-3 Version 2.01 spreadsheet is used to generate input data for the simulator and to display the output data from the simulator (see APPENDIX G). This system proved to be highly effective enabling a large variety of functions to be studied and has been adapted for this project to work with the TMS320C30 floating point processor simulator [1]. In addition, the more up-to-date Microsoft spreadsheet package 'Excel'

version 5 has been used for demonstrating some aspects of zero crossing techniques and for printing charts; see APPENDIX A for a more detailed description.

1.4 OUTLINE OF THE STUDY

Chapter 2 briefly reviews the work of other researchers since 1947.

Chapter 3 gives a description of signal reconstruction from zero crossing values.

Chapter 4 describes spectral analysis by an indirect method which first reconstructs a signal from zero crossing values and then uses a traditional fast Fourier transform. Three methods of direct spectral analysis using only the values of zero crossings are also considered in detail.

Chapter 5 discusses the possible applications of the findings of this study and reviews some applications studied by other researchers.

Chapter 6 contains the conclusions drawn from this study.

CHAPTER 2 REVIEW OF PREVIOUS WORK

2.1 INTRODUCTION

The subject of zero-crossing has been considered by many researchers with a view to practical applications for almost 50 years and mathematicians such as Titchmarsh [2] have contributed theoretically many years earlier.

2.2 INFINITE CLIPPING OF SPEECH SIGNALS

Licklider and Pollack [3] describe experiments demonstrating that infinite clipping of speech signals has little effect on the intelligibility of the information contained in the spoken message. The "intelligibility" was determined by carefully constructed tests using 5 standard lists of 50 monosyllabic words each and each list then recorded by one speaker in 5 different orders randomly selected. The recordings were then heard by 5 listeners via a low distortion audio amplifier and then via 10 different arrangements of a differentiator, an infinite clipper and an integrator. Each listener's success or failure to recognise a word were then recorded and a measure of intelligibility was thus determined. One of the circuit arrangements is shown in Figure 2-1.

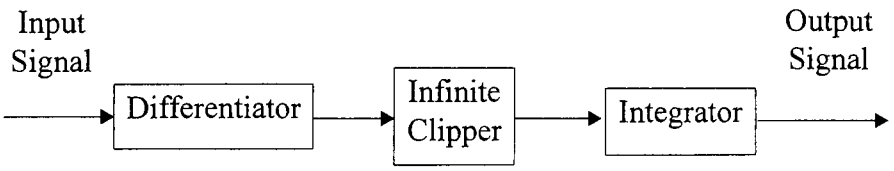


Figure 2-1 *Speech Signal Distorting Circuit with Infinite Clipper*

The result of the tests were that if an undistorted signal were first differentiated (i.e. passed through a first order high pass filter), infinitely clipped and then integrated (i.e. passed through a first order low pass filter) the intelligibility was as high as 97%. It was also observed that the listeners recognition performance improved with practice but was not due to them remembering word sequences. It also appeared that a measure of the harmonic distortion was not related to intelligibility.

This demonstration was a dramatic example of the importance of zero crossing times since the clipped speech signal only consisted of rectangular waves. It became clear that the apparently highly distorted signal contained sufficient information or attributes to convey complicated speech patterns with a high degree of accuracy.

2.3 SAMPLING THE ZEROS OF BAND-LIMITED SIGNALS

Bond and Cahn [4] studied the traditional sampling theorem related to band-limited signals and developed a sampling theorem which utilised information related to the zeros of an input signal. The sampling theorem states that the sampling rate must be at least twice the bandwidth of the signal, known as the Nyquist rate, for a signal to be reconstructed from those samples. The samples normally occur at regular time intervals and contain a measure of the amplitude of the signal at the sampling instant.

In their paper [4] the concept of complex zeros is considered in some detail and they explain that, in general , the zero crossing rate of a signal is less than the Nyquist rate hence it cannot be completely recovered from the zero crossing values. The deficiency in the number of zeros was explained earlier by Titchmarsh [2] indicating

that zeros do occur at the Nyquist rate for a band-limited signal but they are either real or occur in complex conjugate pairs.

Bond and Cahn [4] pointed out that "continuous band-limited functions generated by natural information sources will include complex zeros". Real zeros are readily detectable since they are the points where the signal crosses the time axis, known as zero crossings. Complex zeros on the other hand are not straightforward to detect and a means of calculating the real and imaginary parts had not been discovered prior to 1958. The experiments conducted by Licklider and Pollack [3] were on signals with the complex zeros eliminated thus consisting of real zeros only.

2.4 A UNIFIED THEORY OF MODULATION

Two papers by H B Voelcker [5][6] are a wide ranging reappraisal of modulation theory which up to the time of his publications had concentrated on amplitude and angular modulation. The possibility of modulating a signal in a generalised way with amplitude and angular parameters simultaneously is examined in detail and equations which describe the relationships applying to such band-limited signals are developed.

The significant outcome of the papers as far as this study is concerned is that the equations turned out to be totally dependent on the real and complex zeros of the modulated signal. The zeros, both real and complex, of a band-limited signal are thus shown to be the full informational attributes of the signal. The following simple product relationship describes a band-limited signal:

$$s(t) = s_{CZ}(t) \,.\, s_{RZ}(t) \qquad \text{..... (2-1)}$$

where $s_{CZ}(t)$ denotes the product of all the complex zeros and $s_{RZ}(t)$ the product of all the real zeros. If all the zeros are complex then $s_{RZ}(t) = 1$ and if all the zeros are real then $s_{CZ}(t) = 1$ where both $s_{RZ}(t)$ and $s_{CZ}(t)$ are band-limited signals [6]. The expansion of the real and complex terms are shown in equations (2-2) and (2-3):

$$s_{CZ}(t)=\prod_{i=1}^{n_C}[\cosh\Omega|\sigma_i|-\cos\Omega(t-\tau_i)] \quad \text{..... (2-2)}$$

$$s_{RZ}(t)=\prod_{(k,L)=1}^{n_R}\left[\cos\Omega\left(\frac{\tau_k-\tau_L}{2}\right)-\cos\Omega\left(t-\frac{\tau_k+\tau_L}{2}\right)\right] \quad \text{..... (2-3)}$$

There are two roots associated with each complex zero and τ_i is the real time value of the 'i th' root and σ_i is the imaginary time value. In a similar way there are two roots associated with each real zero, τ_k and τ_L are the real time values.

The real zeros are simple to detect since they are the times at which the signal crosses the time axis, better known as zero crossings. Complex zeros, which are associated with the minima of the signal, are very difficult to identify accurately. Voelcker [6] suggests that if the complex zeros are converted to real zeros using an invertible transform then the transformed signal can be reconstructed from a knowledge of the zero crossing times only. The original signal can then be realised by applying the inverse of the transform.

An infinitely clipped speech signal has the complex zeros eliminated and only the zero crossings of the signal are preserved hence some of the attributes defining the original signal are absent. The missing attributes account for the loss in quality of the

signal but the real zeros contain sufficient information for it to be intelligible as demonstrated by Licklider and Pollack [3].

2.5 REAL-ZERO INTERPOLATION

Andrew Sekey [7] examined the possibility of improving the quality of high speed facsimile transmission using zero crossing techniques. He further develops Voelcker's relationship of equations (2-2) and (2-3) into a more manageable form with the scaling constants 2^{2n} and $|C_n|$ included as shown in equation (2-4).

$$s(t)=2^{2n}|C_n|\underbrace{\prod_{(CZP)}[\cosh\Omega\sigma_k-\cos\Omega(t-\tau_k)]}_{S_{CZ}(t)\geq 0}\underbrace{\prod_{(RZ)}\sin\frac{\Omega}{2}(t-\tau_k)}_{S_{RZ}(t)} \quad \text{..... (2-4)}$$

where the first product term represents the conjugate complex zero pairs and the second represents the real zeros or zero crossing times which occur at times τ_k. If there are no complex zeros in s(t) then equation (2-4) reduces to:

$$s(t) = 2^{2n}|C_n|\prod_{(RZ)}\sin\frac{\Omega}{2}(t-\tau_k) \quad \text{..... (2-5)}$$

where:

- $s(t)$ = The band-limited signal
- $2n$ = Total number of zero crossings
- $|Cn|$ = Half the magnitude of the highest frequency component
- Ω = Fundamental angular frequency (the lowest frequency component)
- τ_k = Zero crossing times

This equation is of particular importance to this study which is primarily concerned with the reconstruction of signals containing real zeros (i.e. zero crossings) only. All

the parameters are easily realised and the values of τ_k can be measured with simple electronic circuits [8] [9].

A derivation of this important equation (2-5) has not been found in any publication but is provided by the author in APPENDIX B.

2.6 ENTIRE FUNCTIONS

Requicha [10] discusses in detail 'entire functions' which are functions analytic over a complex plane [11: APP 1 & 2]. The paper is largely concerned with fundamental mathematical issues drawing from the work of Voelcker and many others but also refers to many practical engineering applications that can be described by an entire function. An application highly relevant to this study and discussed in his paper is an alternative method of analogue to digital conversion using zero interpolation.

2.7 ZERO CROSSING BASED SPECTRUM ANALYSIS

Kay and Sudhaker [12] bring together the work of many researchers in a relatively simple way such that methods of signal reconstruction and discrete Fourier transform coefficient generation from zero crossing times can be examined using straightforward simulation tools.

2.8 SUMMARY

The subject of zeros and zero crossings has been researched for well over 70 years and even longer if the more fundamental nature of entire functions and Fourier series, which is a particular example, is considered. One aim of this study is to explore the possibility of calculating spectra directly from the real zeros of signals which only

contain real zeros but during the course of research for this dissertation, the importance of signal reconstruction became a dominant factor.

The applications which have been reviewed include new methods of modulation, data transmission, an alternative method of analogue to digital conversion and spectral analysis which covers a wide range of engineering activity. The techniques have also been used in image processing and optical signal analysis [8],[9],[13].

CHAPTER 3 SIGNAL RECONSTRUCTION

3.1 INTRODUCTION

The reconstruction of a signal from real zero crossing values has been proved possible as already outlined in Chapter 2. Kay and Sudhaker [12] suggested the simplest form of the essential equations provided that all the complex zeros were converted to real zeros by using the simple invertible transform of adding an extra signal (i.e. a transforming signal) to the input signal [6].

The input signal is first bandwidth limited by a bandpass analogue filter in a similar manner to an anti-aliasing filter in an analogue to digital converter system. The transforming signal is then added, the resulting real zero crossings are detected and their times measured. The zero crossing time values are then processed and the result of the processing is then converted back to an analogue output signal. The system is shown in Figure 3-1

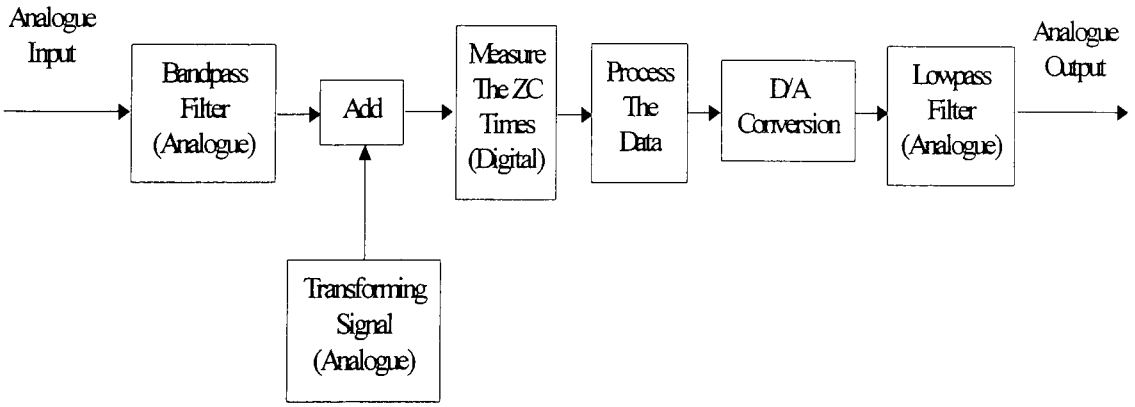


Figure 3-1 *System Block Diagram*

The frequency of the transforming signal must be equal to or greater than the maximum frequency of the band-limited input signal and its magnitude large enough to ensure that all the complex zeros of the input signal are 'forced' to become real zeros (i.e. zero crossings). If all the complex zeros are converted to real zeros the total number is equal to 2M where $M = 2\pi W/\omega_0$ and $\omega_0 = 2\pi/T$ where W is the bandwidth and T is the period of the fundamental frequency [12].

An example of a signal which contains both real and complex zeros is shown in Figure 3-2. It consists of the fundamental and the first 4 odd harmonics of a truncated square wave. The complex zeros which exist in conjugate pairs occur at the minima points of the signal [5]. Real zeros occur at the intersection of the signal and the time axis.

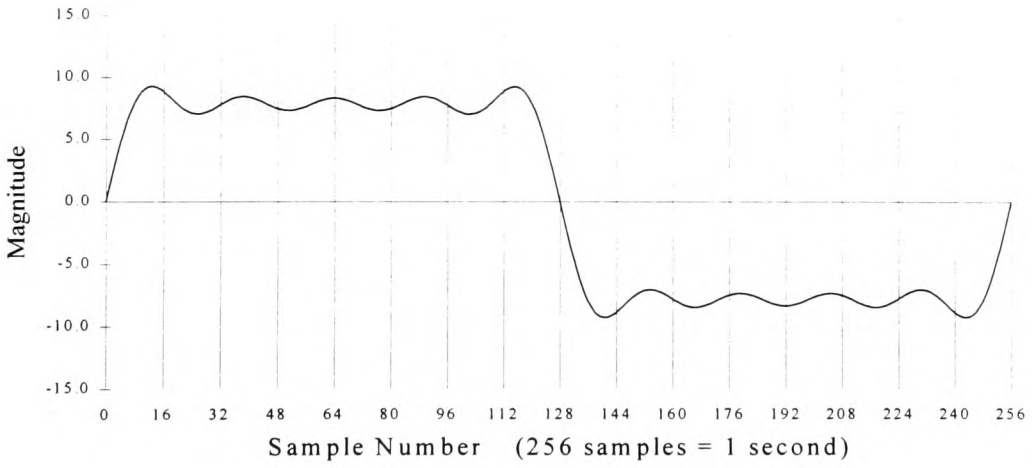


Figure 3-2 *The Input Signal*

The resultant signal after transformation is shown in Figure 3-3. It is clear that as the magnitude of the transforming signal is increased the minima on the positive side of the time axis and the maxima on the negative side move towards the time axis. When the magnitude is sufficiently high all the minima occur on the negative side of the time axis and all the maxima occur on the positive side thus creating two real zero crossings per complex zero of the original signal thus giving a full complement of 2M real zeros.

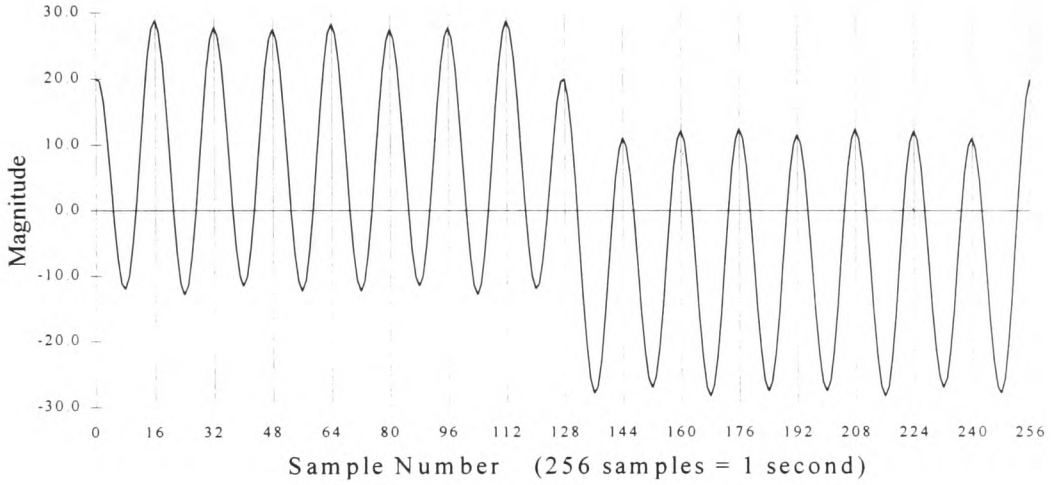


Figure 3-3 *The Input plus Transforming Signal*

In this example the frequency of the transforming signal is 16 times that of the fundamental of the original signal giving 32 zero crossings.

3.2 CALCULATING ZERO CROSSING VALUES

The signals shown in Figures 3-2 and 3-3 were generated using a spreadsheet with 256 discrete time samples. It is also possible to estimate the positions along the time

axis where the signal intersects using the same spreadsheet; these time values are locations of the real zeros.

The procedure is first to detect if a change of sign in the signal magnitude occurs at two consecutive sample times. The zero crossing time is then estimated by linear interpolation as indicated in figure 3-4.

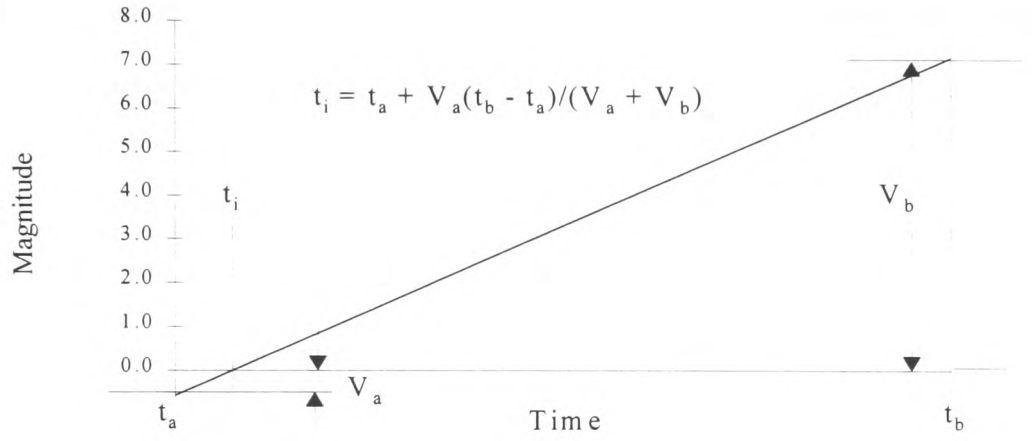


Figure 3-4 *Linear Interpolation of Zero Crossing Value*

There is clearly some error using this form of estimation but if the number of samples is increased the zero crossing value calculation is more accurate. There are more accurate interpolation algorithms [14: Ch 3.3] but they place greater demands on computation. It was found that linear interpolation was simple to implement and gave sufficient accuracy to adequately demonstrate the principles of zero crossing techniques and has thus been used throughout this study. The spreadsheet used for zero crossing value generation is shown in APPENDIX C.

A practical hardware design for measurement of the positions of real zeros has been proposed using operational amplifiers and a high resolution pulse counter [9] which can easily measure zero crossing times to within one part in 65536 or better. The circuit is suitable for real time systems.

3.3 RECONSTRUCTION ALGORITHM

Reconstruction of a signal after transformation from the resulting zero crossing values is achieved by the following relationship [12] (see also APPENDIX B):

$$s(t) = 2^{2M}|C_{2M}|\prod_{i=1}^{2M}\sin(t-t_i)\frac{\pi}{T} \quad \text{..... (3-1)}$$

where:
- s(t) = Reconstructed signal.
- 2M = Number zero crossings.
- $|C_M|$ = Half the magnitude of transforming signal
- t = Sample time
- t_i = Zero crossing time
- T = Period of the fundamental frequency

This equation as it stands requires a large number of sine functions which is computationally intensive since an iterative polynomial technique is normally employed to achieve a result [15: pp. 273-279] for each function call. It will be shown later that for spectral analysis as many samples as there are zero crossings are required but for reconstruction, oversampling simplifies the design of the analogue low pass filter on the output. If the number of zero crossings were 2M the number of calls required would be $4M^2$ for spectral analysis and $8M^2$ for reconstruction with 2 times oversampling.

The number of trigonometric function calls can be reduced to 8M and 16M respectively if equation (3-1) is modified as follows:

$$s(t) = 2^{2M}|C_{2M}|\prod_{i=1}^{2M}\left[\sin\left(\frac{t\pi}{T}\right)\cos\left(\frac{t_i\pi}{T}\right)-\cos\left(\frac{t\pi}{T}\right)\sin\left(\frac{t_i\pi}{T}\right)\right] \quad \text{..... (3-2)}$$

The values of 't' and 't_i' are known hence the trig functions can be calculated before the product expression is evaluated. The task is thus reduced to multiplication and addition or subtraction of floating point numbers. If processing is carried out on a continuous basis the values of $\sin(t\pi/T)$ and $\cos(t\pi/T)$ need only be calculated once and stored in an array hence, apart from the initial overhead, the number of trig function calls is reduced to 4M and 8M respectively.

3.4 SEQUENTIAL CALCULATION

The straightforward way to calculate equation (3-2) is to store all the values of 't_i' in an array of 2M memory locations and then calculate the product expression for 2M values of 't' thus producing 2M reconstruction samples. This method has the disadvantage that the calculation cannot start until all 2M values of 't_i' are detected and stored hence there is a delay for both the detection and calculation processes.

An alternative method is to store the results of the first term of equation (3-2) calculated for 2M values of 't'. The stored results are then multiplied by the results for the second term with the products re-stored in the same array. The array has thus accumulated the product for the first two terms. The process is then repeated until all 2M terms have been included with the final results yielding the completed 2M reconstruction samples. Each multiplication stage takes the same time hence the

reconstruction process can be evenly spread over the detection period of the zero crossings.

This latter method is likely to be the most suitable for a practical application. A further and essential improvement is to move the factor of 2^{2M} inside the product expression since its magnitude would be far too high even for a floating point processor (i.e. $2^{512} = 1.157 \times 10^{77}$) to calculate separately (See section 3.1.6.3 for more detail).

The number of multiplications can be further reduced if the values of 2 $\sin(t\pi/T)$ and 2 $\cos(t\pi/T)$ are initially stored. A 'C' code program for the TMS320C30 processor simulator for sequential reconstruction is shown in APPENDIX D.

3.5 LOOK-UP TABLES

It is often desirable to eliminate the need for polynomial trig function calculations which require many operations [15: pp. 292-293]. If the values are pre-calculated and stored in a look-up table the speed of computation can be increased and the writing of an assembler program is relatively simple. Only one operation is required to obtain any trig value.

Equation (3-2) can be used as a basis for a look-up table but it would be very large for a high resolution. If there were 256 zero crossings a resolution of 1/8192 would be possible with a table of 4097 values since all values for sine and cosine functions can be calculated from one quadrant of a sinewave. This table would give a very poor reconstruction with only 32 values per Nyquist interval (see Figure 3-5).

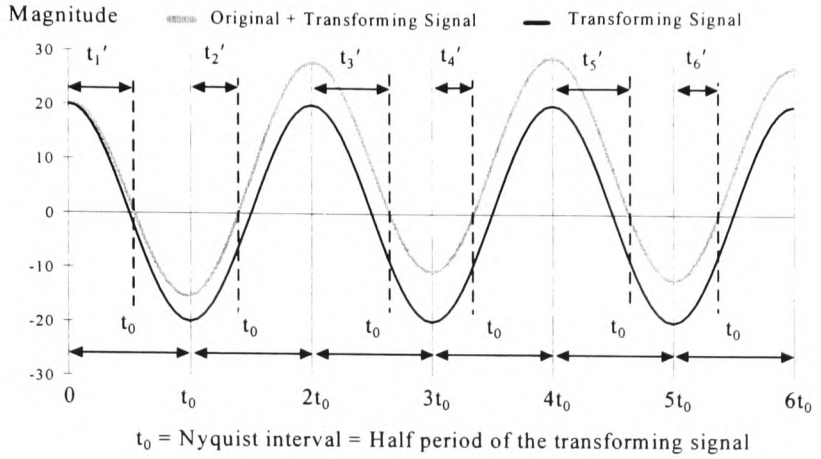


Figure 3-5 *Zero Crossings and the Nyquist Interval*

The size of the table can be substantially reduced if the values of the zero crossings are related to the Nyquist interval since only one zero crossing can occur within it [9] as shown in Figure 3-5.

The zero crossing times are: $t_i = n\,t_o + t_i'$ where $n = 0, 1, 2 \ldots 2M$ and $2M$ is the total number of zero crossings within the resolution period. The values for sines and cosines can thus be obtained from the simple trig identities:

$$\sin(\pi t_i/T) = \sin(n\pi t_o/T)\cos(\pi t_i'/T) + \cos(n\pi t_o/T)\sin(\pi t_i'/T) \quad \ldots\ldots\ldots\ldots \quad (3\text{-}3)$$

$$\cos(\pi t_i/T) = \cos(n\pi t_o/T)\cos(\pi t_i'/T) + \sin(n\pi t_o/T)\sin(\pi t_i'/T) \quad \ldots\ldots\ldots\ldots \quad (3\text{-}4)$$

Three quadrants of $\sin(n\pi t_o/T)$ give the values of $\cos(n\pi t_o/T)$ and $\sin(n\pi t_o/T)$ where $t_o/T=1/2M$. There are $2M$ values of $\sin(\pi t_i'/T)$ since t_i' ranges in integers from 0 to $2M$, similarly for $\cos(\pi t_i'/T)$; hence $2M$ values per Nyquist interval. Reconstruction from 256 zero crossings therefore requires three small look-up tables with a total size of 897 locations (385+256+256) giving a resolution of 1/65536 with 256 values per

Nyquist interval. This increases the resolution by a factor of 8 with table size of less than a quarter of that using the basic equation (3-2).

There are clearly multiply and add operations to yield the values but the calculation times are much less than that for polynomial sine and cosine calculations. Figure 3-6 shows the relationship between the estimated execution time [16: Ch 6.7] and the number of zero crossings for reconstruction where the number of samples is equal to the number of zero crossings. The 'C' codes for the TMS320C30 simulator for sequential reconstruction for both polynomial calculation and using look-up tables is shown in APPENDIX D.

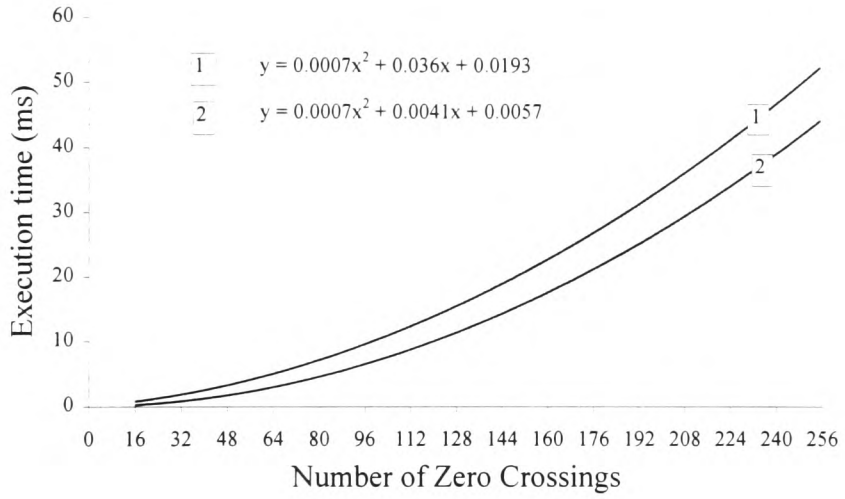


Figure 3-6 *Execution Time Vs Zero Crossings for Reconstruction*
Trace 1: Reconstruction by polynomial calculation
Trace 2 : Reconstruction using look-up tables

Figure 3-6 shows that the execution time is pre-dominantly dependent on the square of the number of zero crossings which reflects the method of calculation. The program consists of three stages. The first is the calculation of the sines and cosines of the zero crossing values, the second is the evaluation of the product expression and

the third stage is to output the results. The execution time of the first and third stages is clearly directly proportional to the number of zero crossings but the second stage is proportional to the square because there are as many samples generated as there are zero crossings. This applies to both the polynomial and look-up table methods of calculation. The best fit curves shown in Figure 3-6 also show this.

3.6 OVERSAMPLING

A reconstruction of the signal shown in Figure 3-2 using the TMS320C30 'C' code is shown in Figure 3-7. The result is a very close approximation to the original signal with only 32 samples generated.

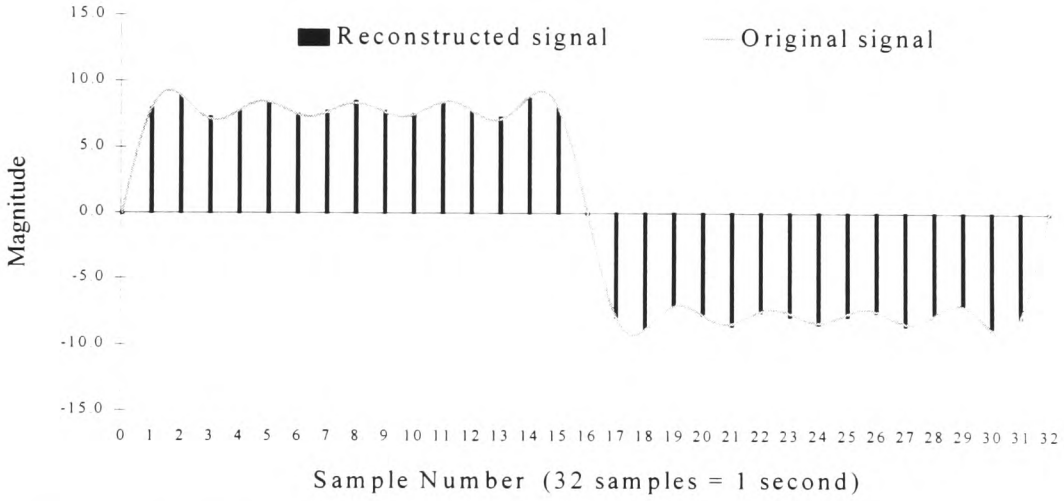


Figure 3-7 *Reconstruction of Truncated Squarewave from 32 Zero Crossings*

If the number of samples is the same as the number of zero crossings the information is sufficient for spectral analysis using a Fourier or Hartley transform or similar (see Chapter 4). If the requirement is to reproduce the signal, as many samples as there are zero crossings is theoretically sufficient but an ideal low pass filter after the output of

the digital to analogue converter (D/A) would be needed to recover the processed signal (see Figure 3-1). Figure 3-8 shows the result of reconstructing a signal from 32 zero crossings with only one component of constant magnitude at the maximum frequency of the band (i.e. 15Hz) and only 32 samples generated. Figures 3-9 and 3-10 show how the reconstruction of the same signal improves as the number of samples per zero crossing increases (i.e. 2 and 4 times oversampling).

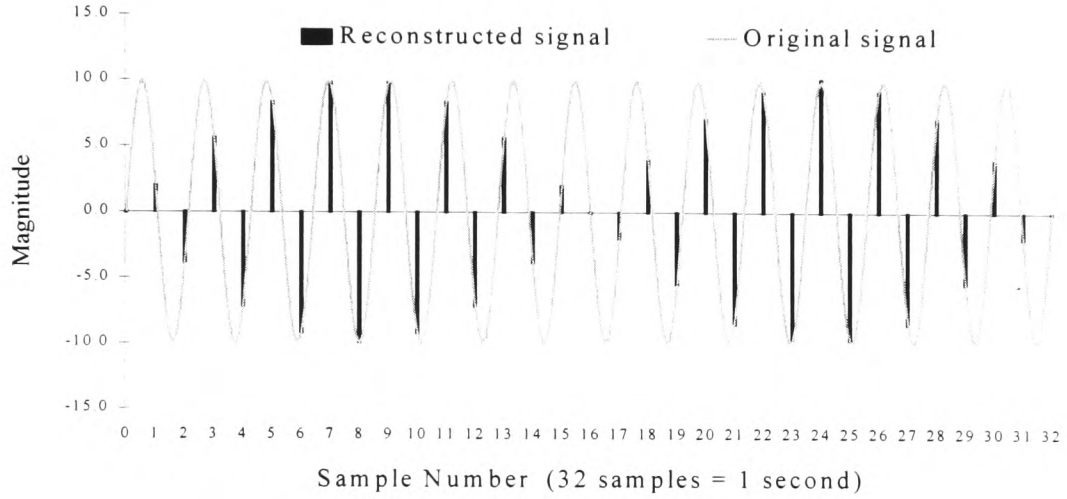


Figure 3-8 *Reconstruction from 32 Zero Crossings and 32 Samples*

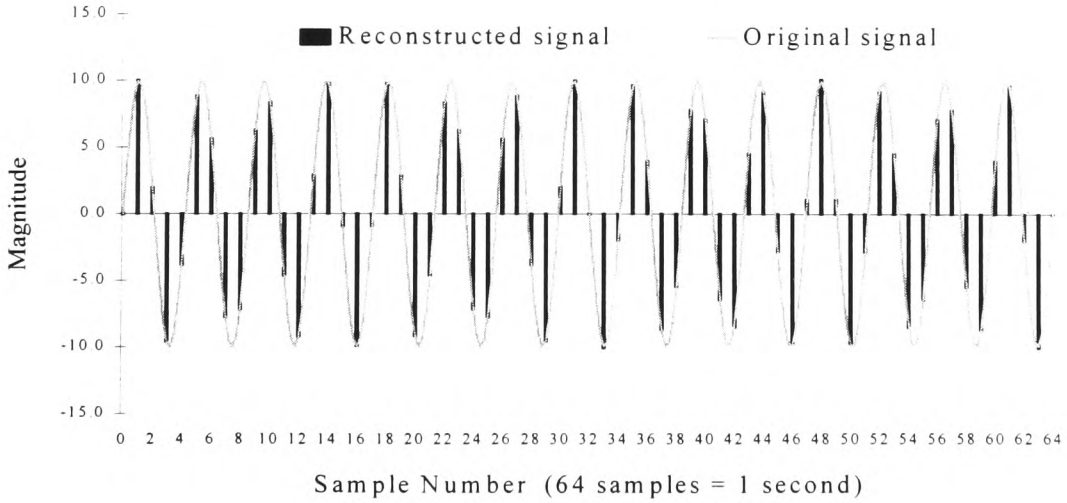


Figure 3-9 *Reconstruction from 32 Zero Crossings and 64 Samples*

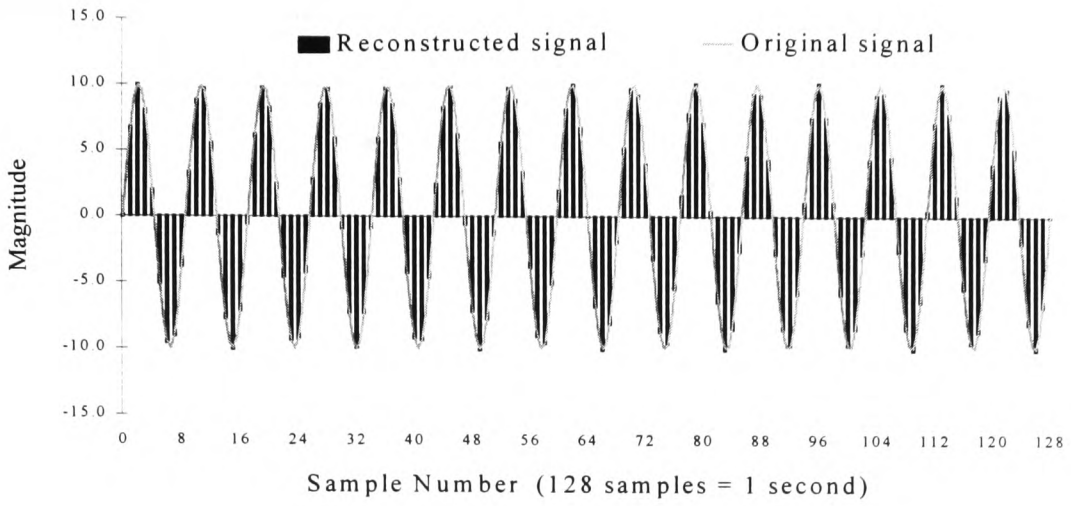


Figure 3-10 *Reconstruction from 32 Zero Crossings and 128 Samples*

The execution time increases linearly with the number of samples for a fixed number of zero crossings. The method of oversampling used here is to use the reconstruction equation but other methods are possible [17] provided that the minimum number of samples at the Nyquist rate have been generated. Oversampling simplifies the design of the low pass filter required to produce the analogue output.

3.7 FREQUENCY RESOLUTION

Important aspects of the discrete Fourier transform are the minimum frequency and frequency interval that can be resolved. If, for example, a transform has 256 points, all frequency components contained within the periodic signal to be transformed must repeat exactly within the period represented by 256 evenly spaced samples of the signal. If the minimum frequency were 100 Hz all other frequencies must be integer multiples of 100 Hz with the maximum resolved being 100 x 256 / 2 = 12800 Hz.

The maximum frequency is characterised by 2 samples which conforms to the sampling theorem (i.e. the Nyquist sampling rate).

The same situation applies to zero crossings. If a periodic signal is to be reconstructed with a resolution of 100 Hz and has a maximum frequency component of 12800 Hz then the number of zero crossings must be 12800 x 2 / 100 = 256 (i.e. 2 zero crossings per period of the maximum frequency or 1 zero crossing per Nyquist interval). The frequency that must be added to ensure the required number of zero crossings is 12800 Hz provided that the input signal can be band-limited to 12800 Hz. An ideal bandpass filter would be required to ensure that all frequencies below 100 Hz and above 12800 Hz were rejected from the input signal before the transforming signal were added (see Figure 3-11).

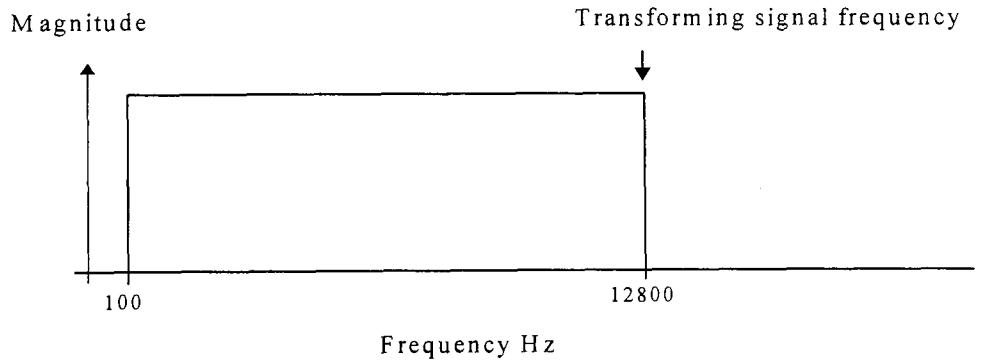


Figure 3-11 *Frequency Response after Ideal Bandpass Filter Stage*

This is not possible to achieve since there must always be a finite attenuation outside the bandwidth of any practical filter. The higher the attenuation rate the more elaborate the filter design has to be with attendant compromises of phase and magnitude response in the bandpass region.

A practical situation is shown in Figure 3-12 where the transforming signal is twice the upper cut-off frequency of the input bandpass filter. In this case the filter must attenuate the input signal to a very low value at just less than the transforming signal frequency. An attenuation of 36 dB/octave is a stringent specification but possible with a 6th order Butterworth filter response [18: Ch 6][19: pp. 560-578].

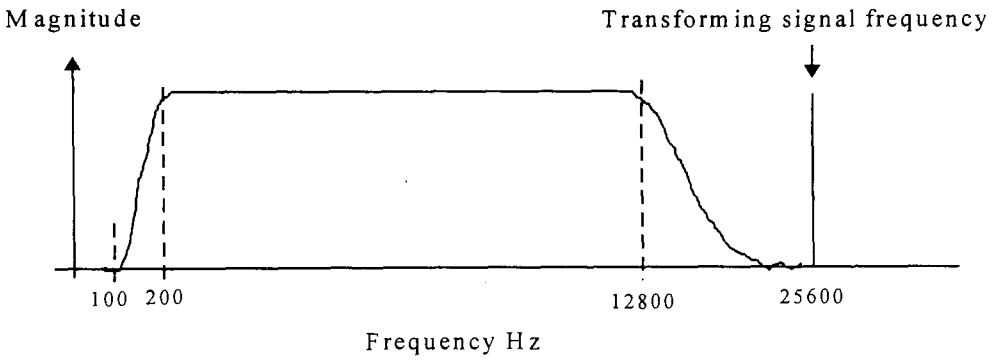


Figure 3-12 *Frequency Response after Practical Bandpass Filter Stage*

This means that the number of zero crossings required is twice the ideal situation because unwanted frequencies above 12800 Hz may be present in the original signal hence the calculation time increases but the resolution of 100 Hz remains the same. An increase in resolution to 50 Hz requires that the period over which the zero crossings are detected is doubled hence the number of zero crossings is doubled. This clearly shows that the higher the required frequency resolution the higher the calculation time becomes.

The obvious question to be asked at this stage is what happens if the input signal contains a component whose frequency is not an integer multiple of the resolution frequency? Figure 3-13 shows the resulting reconstruction from 32 zero crossings of a 7.5 Hz signal with a resolution frequency of 1.0 Hz and 8 times oversampled.

Figure 3-14 shows the difference between the reconstructed and original signal. The distortion is a maximum at the beginning and end of the sampling period but becomes smaller towards the middle.

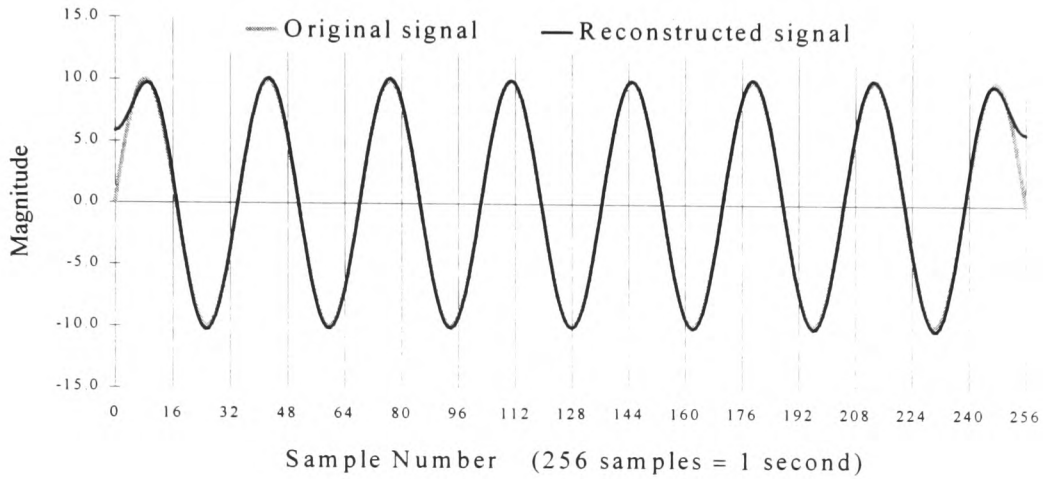


Figure 3-13 *Reconstruction of Signal Outside Resolution Frequency*

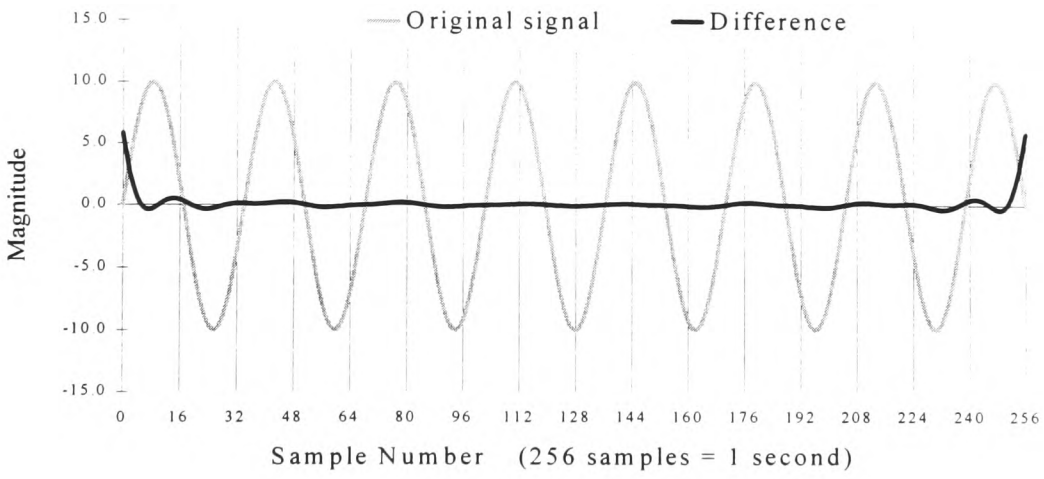


Figure 3-14 *Difference Between Reconstructed Signal and Original*

Figures 3-15 and 3-16 show the resolution effect exaggerated by using only 8 zero crossings, one signal component at 1.5 Hz, a resolution of 1.0 Hz and with 4 consecutive cycles hence 4 consecutive zero crossing measuring periods. The signal repeats every 2 cycles. The difference signal shown in Figure 3-16 indicates clearly that there are noise spikes at the beginning and end of each zero crossing measuring period.

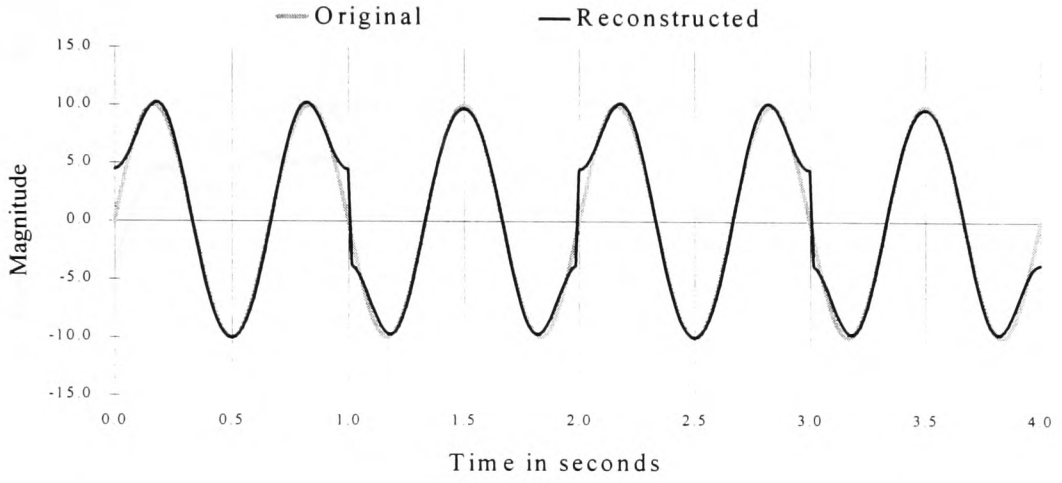


Figure 3-15 *4 Cycles of a Signal Outside the Resolution Frequency*

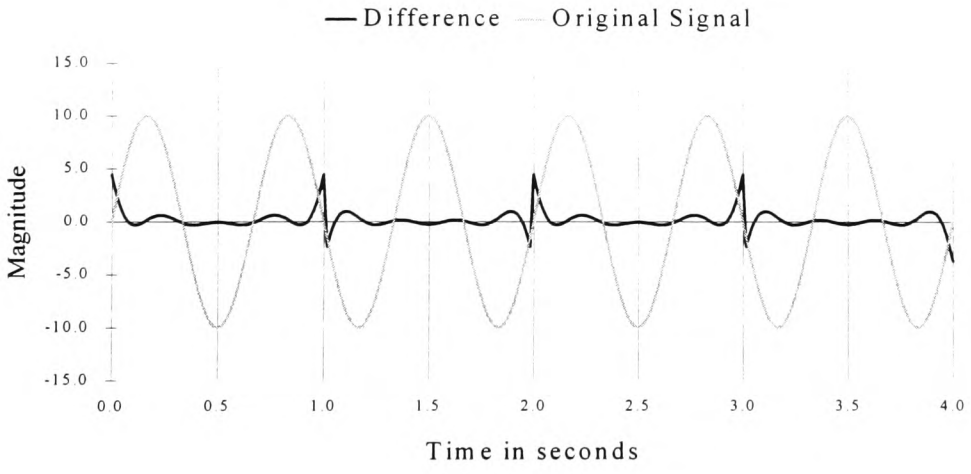


Figure 3-16 *Difference and Original for 4 Cycles*

Figure 3-17 shows the truncated square wave with the third harmonic replaced by a frequency component of 2.5 Hz. This alters the shape of the truncated squarewave but the reconstruction is almost the same as the original. Figure 3-18, where the magnitude scale has been increased, shows the difference which again is larger at the start and finish of the sampling period than it is in the middle.

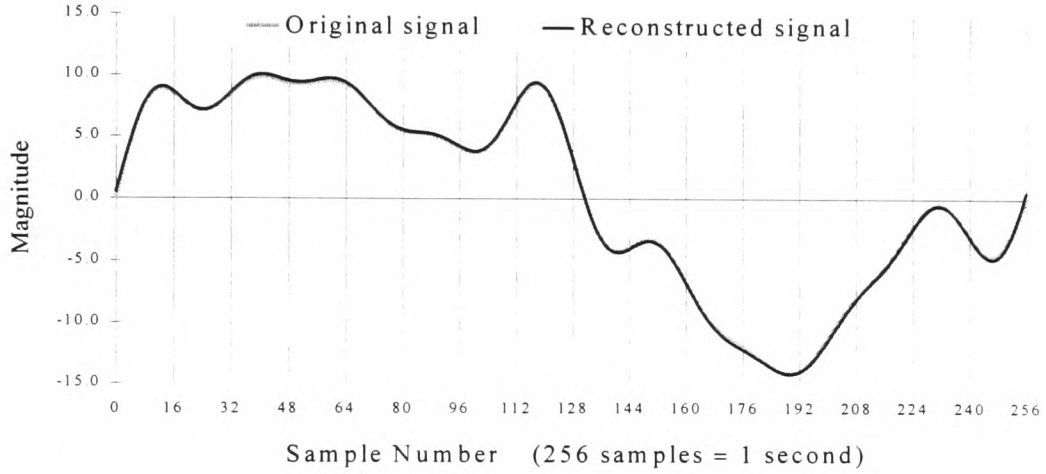


Figure 3-17 *Original and Reconstructed Modified Truncated Squarewave Signal*

The error shown in Figure 3-18 is small since only one component of the five is outside the resolution frequency (N.B. the magnitude range is ± 4.0). In Figure 3-14 the error is larger because the only component is outside the resolution frequency (N.B. the magnitude range is ± 15.0). The resolution error can only be reduced by increasing the number of zero crossings to reconstruct the original signal. It is not a function of the accuracy of measurement of the individual zero crossing times which

is analogous to the measurement of signal magnitude at sample intervals in traditional analogue to digital converters.

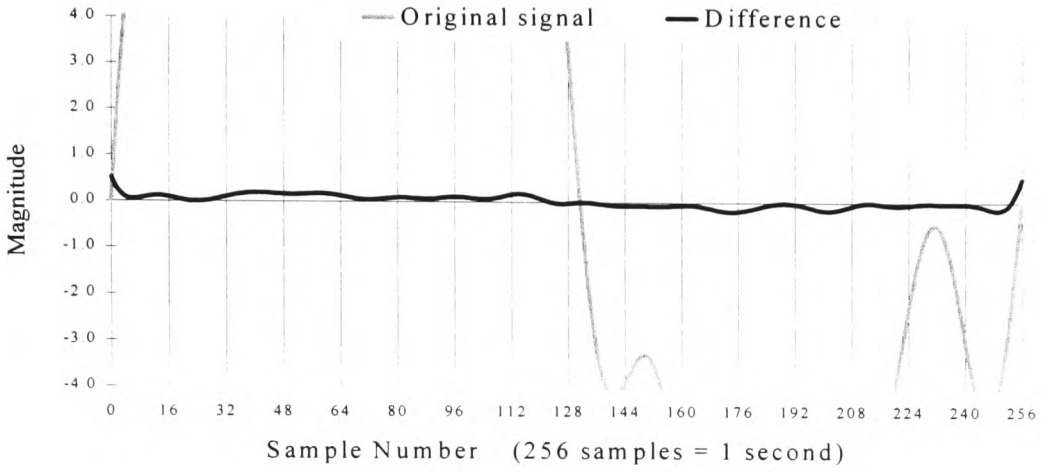


Figure 3-18 *Difference Between the Original and the Reconstructed Modified Truncated Squarewave Signal*

3.8 ACCURACY

The accuracy of reconstruction is predominantly dependent on the accuracy of measuring the positions of the zero crossings but is also a function of the magnitude of the transforming signal and the calculation process itself.

3.8.1 Reconstruction Accuracy

In order to assess the accuracy of reconstruction some quantitative measure is required. A useful method is suggested in the MATLAB Signal Processing Tool Box [20: 1-58] where the 'Euclidean norm' [21: Ch 19.4 pp. 1024] of the difference between the original signal and the reconstructed signal is divided by the 'norm' of the original signal. This is in effect dividing the RMS of the difference by the RMS

of the original thus comparing the power of the error signal with the power of the original signal:

$$\text{Error}\% = 100\,\frac{\text{norm}(\text{Reconstruction}-\text{Original})}{\text{norm}(\text{Original})} = 100\sqrt{\frac{\sum(\text{Reconstruction}-\text{Original})^2}{\sum(\text{Original})^2}}$$

This relationship which is simple to calculate on a spreadsheet has been used throughout this study.

Figure 3-19 shows how the accuracy of reconstruction is affected by the number of samples used to estimate the value of zero crossing times by linear interpolation. The smallest number of samples must clearly be equal to or greater than twice the number of zero crossings since the times must lie between 2 consecutive sample points. The error falls dramatically when the number of samples is doubled or quadrupled.

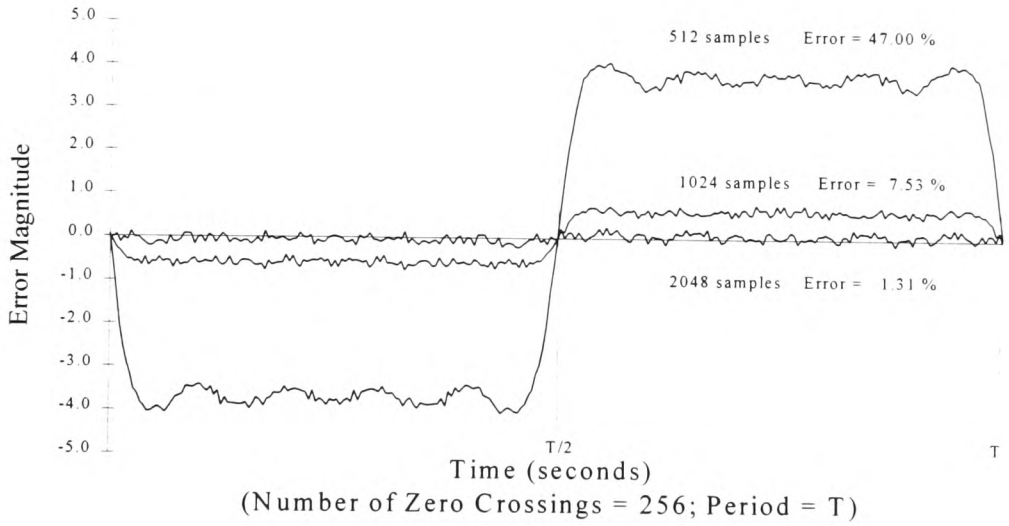


Figure 3-19 *Difference Between Reconstructed and Original Signals Vs Time*

The values for Figure 3-19 were generated using a spreadsheet (see APPENDIX C) where there was a limit of 2048 samples but in a practical system the accuracy of the positions of the zero crossing times would be limited by the hardware circuit [9].

Figure 3-20 shows the relationship between reconstruction accuracy and the number of samples for 32 and 256 zero crossings with the resolution of the times kept constant at one part in 65536 (i.e. 2^{-16}).

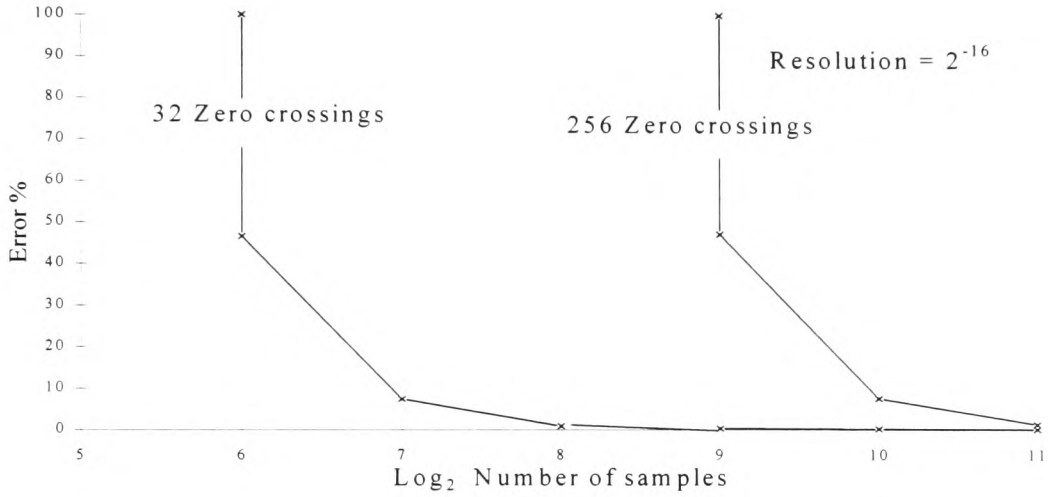


Figure 3-20 *Accuracy of Reconstruction Vs Number of Samples*

Figure 3-21 shows the relationship between reconstruction accuracy and the resolution of the zero crossing times (i.e. from 2^{-7} to 2^{-17}) with the number of samples kept constant at 2048.

It is obvious that the higher the resolution the more accurate the reconstruction but Figure 3-21 shows that the resolution must be at least one part in 8 times the number

of zero crossings giving an error of approximately 15.0%. If the resolution is one part in 256 times, the error reduces to approximately 1.0%.

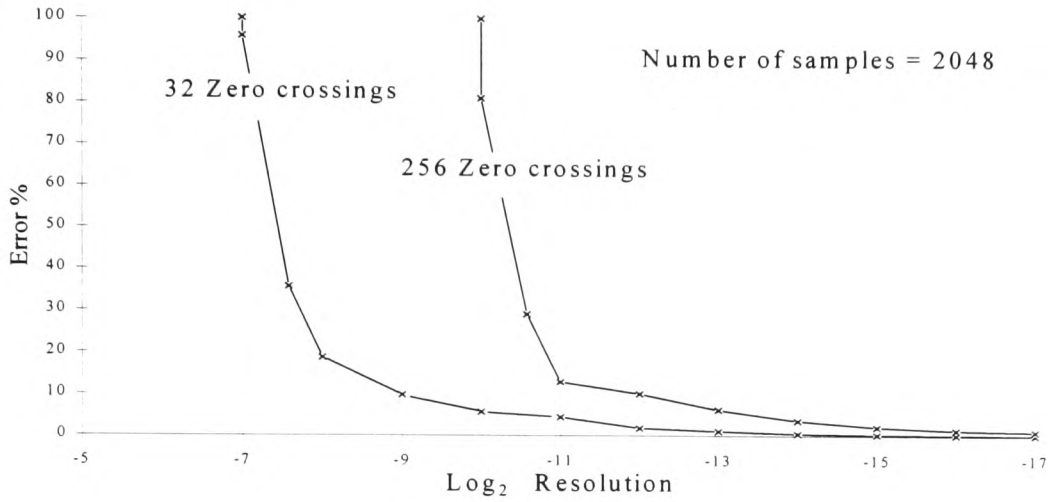


Figure 3-21 *Accuracy of Reconstruction Vs Resolution*

3.8.2 The Transforming Signal

The magnitude of the transforming signal also affects the reconstruction accuracy. As the magnitude of the transforming signal increases the deviation of the zero crossings about the half period times of the transforming signal becomes smaller (see Figure 3-5). This results in a loss of sensitivity as suggested by Kay and Sudhaker [12]. There is however a practical difficulty if the zero crossings are estimated using linear interpolation. The estimation is more accurate if the transformed signal intersects the time axis when the rate of change is a maximum. If the magnitude of the transforming signal is only just enough to force a full complement of real zeros then the rate of change for some intersections will be low and the estimation will poor. In

a practical system the real zero crossing times would be measured by a hardware circuit to a higher degree of accuracy than used in this study [9].

3.8.3 The Processor Accuracy

The nature of the reconstruction algorithm of equation (3-1) presents two computational problems when the number of zero crossings becomes large. The first is the maximum rate at which the value of the product reduces when a large number of consecutive terms with values less than unity are multiplied together. If the values are small enough the processor would return a value of zero before the product calculation has been completed hence all subsequent terms are then multiplied by zero giving an incorrect result when the calculation is complete. This problem is greatly reduced by including the factor of 2 in each term as already indicated in section 3.4.:

$$s(t) = \frac{A}{2} \prod_{i=1}^{2M} 2\sin(t - t_i)\frac{\pi}{T} \quad \text{..... (3-5)}$$

where:
s(t) = Reconstructed signal.
2M = Number zero crossings.
A = Magnitude of the transforming signal
t = Sample time
t_i = Zero crossing time
T = Period of the fundamental frequency

The problem, however, is not eliminated since the larger the number of zero crossings, the more individual terms there are, hence convergence to zero is more likely. The second problem occurs when a large number of consecutive terms greater than unity are multiplied together which happens because the factor of 2 has been

included. Figure 3-22 shows how the individual terms of equation (3-5) alter with the zero crossing time.

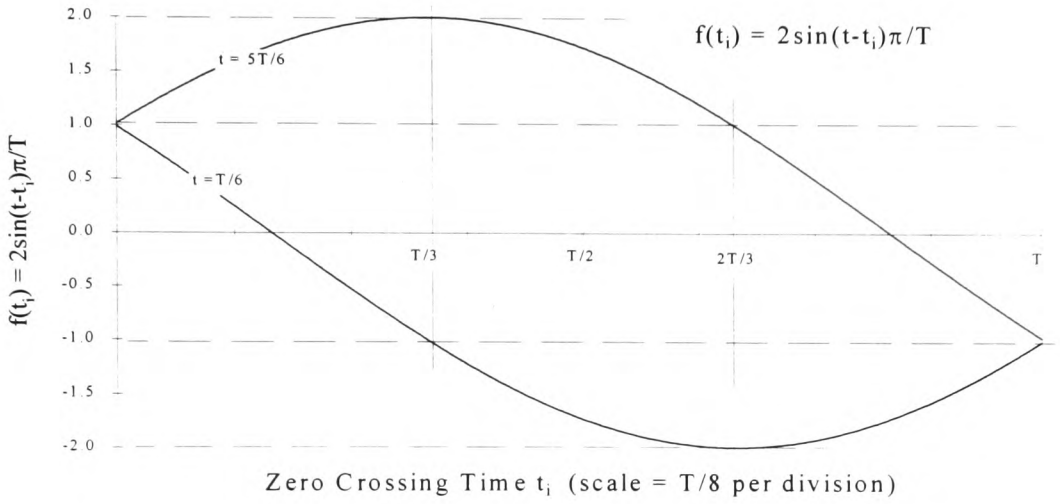


Figure 3-22 *Individual Product Term Vs Zero Crossing Time*

Figure 3-22 shows that with the initial value at unity, the maximum number of consecutive terms less than unity occurs when 't' is held constant at T/6 and the maximum number of terms greater than unity occurs when 't' is held constant at 5T/6. These, respectively, are the worst case conditions for convergence to zero and divergence to overflow.

Figure 3-23 shows, for 256 zero crossings, how the value of the complete product expression of equation (3-5) alters as the available zero crossing times increase. The logarithm of the function has been used to compress the scale since the magnitude becomes very large. The maximum value is $8.2 \times 10^{+35}$ and the minimum 1.2×10^{-36} which is well within the range of the TMS320C30. Figure 3-22 and Figure 3-23 were generated by a spreadsheet (see APPENDIX D).

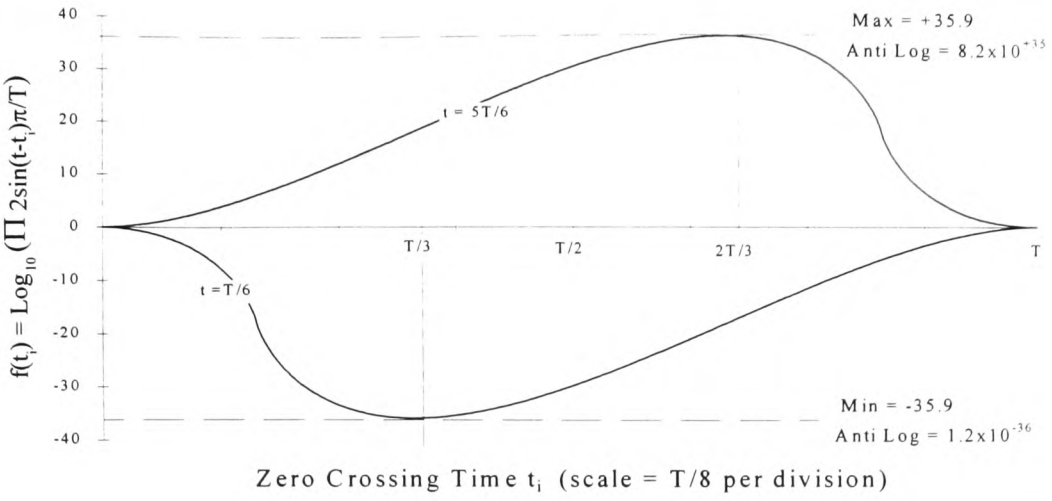


Figure 3-23 *Log_{10} of Product Vs Zero Crossing Time (for 256 Zero Crossings)*

A close approximation to the minimum and maximum values of the product function can be simply determined if the portion of the curve of 2 sin $(t-t_i)\pi/T$ between the values ±1.0 is assumed to be linear (see Figure 3-22). The number of N equally spaced values between zero and +1.0 must be T/6 rounded to the nearest integer. The magnitude of each value is proportional to N hence:

$$\text{Product of N terms} = \frac{1}{N}.\frac{2}{N}.\frac{3}{N}. \ldots \frac{(N-1)}{N}.\frac{N}{N} = \left(\frac{1}{N}\right)^N N! \quad \ldots\ldots\ldots\ldots \quad (3\text{-}6)$$

where N = T/6 (i.e. for 256 zero crossings N ≒ 256/6 = 43 rounded to nearest integer)

If there are 2N values equally spaced between ±1.0 then the product of 2N values gives the approximate minimum value for the product expression hence:

$$\text{Minimum value} \approx \left\{\left(\frac{1}{N}\right)^N N!\right\}^2 \quad \ldots\ldots\ldots\ldots \quad (3\text{-}7)$$

for example if N = 43 the minimum value $\approx 1.213 \times 10^{-35}$ cf. the spreadsheet result: 1.206×10^{-36}.

If the input signal consists only of the transforming signal $\cos(t\pi/T)$ then:

$$\cos\left(\frac{t\pi}{T}\right) = \frac{1}{2}\prod_{i=1}^{2M} 2\sin(t - t_i)\frac{\pi}{T} = \frac{\sqrt{3}}{2}$$

where $t = \frac{T}{6}$

The final result is the product of all the individual terms between ± 1.0 multiplied by all the terms greater than 1.0 (i.e. the minimum value multiplied by the maximum value) hence:

$$\text{Maximum value} = \frac{2\cos\left(\frac{t\pi}{T}\right)}{\text{Minimum value}} \approx \frac{\sqrt{3}}{\left\{\left(\frac{1}{N}\right)^N N!\right\}^2} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3\text{-}8)$$

for example if N = 43 the maximum value $\approx 1.427 \times 10^{+35}$ cf. the spreadsheet result: $8.29 \times 10^{+35}$.

The estimated values are within one order of the results calculated by the spreadsheet and thus compare favourably.

If the number of zero crossings were doubled to 512, the maximum and minimum values would be approximately $2.19 \times 10^{+71}$ and 7.91×10^{-72} respectively which greatly exceeds the precision of the TMS320C30. A processor capable of higher arithmetic precision would be needed to cope with this range of numbers but a limit to the number of zero crossings that can be successfully processed is soon reached.

The experience gained from this project is that if the initial term of the product is 0.5, then 256 zero crossings can be calculated, with no overflow or underflow, by the Texas Instruments TMS320C30 floating point processor [22: Ch 4.3.2. pp. 4-6].

Both the problems of overflow and underflow therefore become apparent when the number of zero crossings is greater than 256. One method which eliminates both effects is to split the calculation into two stages with an initial value of 0.5. The first stage multiplies all the odd numbered terms together with the result lying between ±1.0. The second stage multiplies the result of the first stage by all the even numbered terms with the final result lying between ±1.0. This serves to reduce the rate of convergence to zero and the divergence to overflow at the same time but of course with an execution time penalty. This method works for 512 zero crossings but the calculation needs to be split into 4 stages for 1024. The number of stages therefore increases by 2 if the number of zero crossings increases by 2.

This scheme has been tried successfully for 512 and 1024 zero crossings using a 'C' program running in MS-DOS. If the program is set to use single precision floating point arithmetic, overflow and zero convergence occur for 512 zero crossings with a single multiplying stage giving similar characteristics to the TMS320C30 processor (see APPENDIX D). If the program is set to use double precision arithmetic, 512 zero crossings can be processed using a single multiplying stage but overflow occurs if the number of zero crossings is increased to 1024 which demonstrates the futility of merely increasing the precision without increasing the number of multiplying stages.

Other methods of tackling the problems are possible such as checking after each multiplication if overflow or zero convergence is likely to occur and then multiplying or dividing by a suitable factor to prevent failure. A record would have to be kept of these factors such that they can be used to calculate the final result. This scheme is more complicated and likely to be significantly less efficient in terms of execution time than the split multiplying stage method just described.

The calculation accuracy of the values for sines and cosines also affect the reconstruction accuracy. If the precision is between 4 and 6 decimal places, the error is significant. If the precision is 7 or more decimal places, the reconstruction accuracy does not improve. The major factor affecting reconstruction accuracy is the accuracy of measurement of the zero crossing times.

3.9 SUMMARY

The method of reconstruction from zero crossings using an invertible transform has been demonstrated. Simulation using a spreadsheet is very simple and rapidly gives an insight into the nature of reconstruction. The means of estimating the zero crossing values using linear interpolation is very basic but sufficiently accurate for demonstration purposes.

The modification to the reconstruction equation (3-1) to equation (3-2) considerably speeds up the computation time for the TMS320C30 digital signal processor. Sequential calculation enables the reconstruction calculation to build up as the zero crossing values arrive using the time between them to carry out the multiplications thus improving the efficiency of computation. Relating the zero crossing times to the

Nyquist interval greatly reduces the size of look-up tables which further reduces the computation time.

Any degree of oversampling is possible and is only limited by the speed of the processor or processors in parallel calculating the reconstruction. Oversampling simplifies the design of the low pass filter after digital to analogue conversion (see Figure 3-1).

Analogue to digital conversion using zero crossing times is a valid alternative to traditional methods whose accuracy depends on amplitude measurement. The difference is that time must be accurately measured; however circuits to achieve this [9] are simpler than those for amplitude measurement.

Equations (3-7) and (3-8) provide a useful means of estimating the minimum and maximum values of the product calculation so that it can be determined what floating point number range is required for a particular number of zero crossings. The method of splitting the calculation into 2, 4 ,8 ... etc. stages enables a processor with a fixed floating point number range to increase the maximum number of zero crossings it can handle by factors 2, 4, 8 ... etc. The only limitations are the accuracy of the zero crossing values and the number of decimal places to which the sine and cosine values have been calculated.

CHAPTER 4 SPECTRAL ANALYSIS

4.1 INTRODUCTION

Spectral analysis can be performed in a variety of ways once the input signal has been converted from an analogue signal to a series of digital samples (i.e. Analogue to digital conversion A/D). This chapter considers four methods of analysis. The first is fast Fourier transform (FFT) analysis of a reconstructed signal, the second is direct extraction of Fourier coefficients by comparing the coefficients of a power series with those of a product function, the third is to use Newton's formula and the final method is a sequential version of the second method. The latter three have all been suggested by Kay and Sudhaker [12] but are considered in more detail in sections 4.2, 4.3 and 4.4 . Figure 4-1 shows block diagrams of the four methods considered.

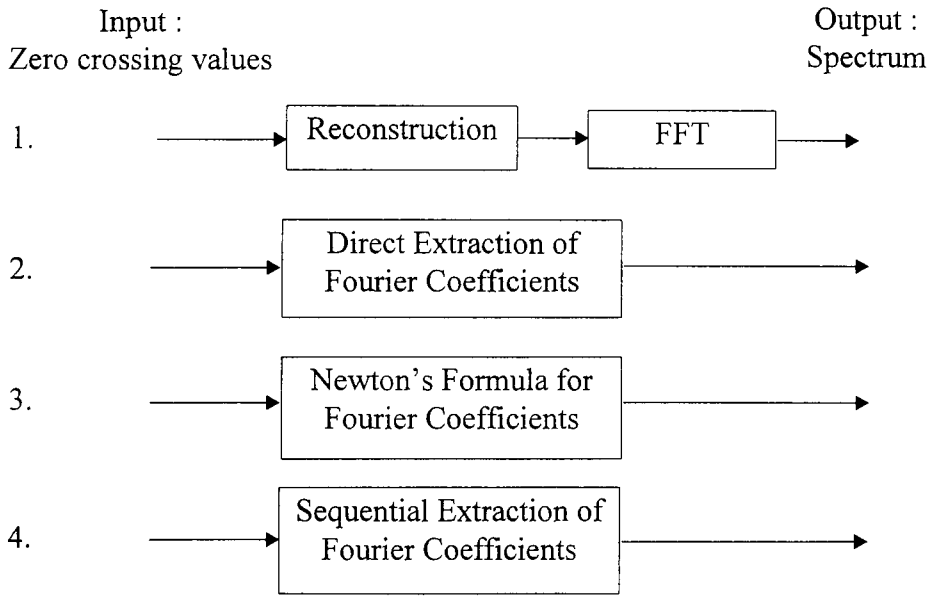


Figure 4-1 *The Four Methods of Spectral Analysis Considered*

4.2 SPECTRAL ANALYSIS FROM RECONSTRUCTED SIGNALS

Chapter 3 describes in detail the reconstruction of a signal from real zero crossings times, which are not evenly spaced along the time axis, and demonstrates that the result consists of a series of digital samples, which are evenly spaced, similar to that produced by the traditional methods of A/D conversion. The major difference is that the samples are derived from the measurement of real zero crossing times rather than measurement of signal amplitude. The samples produced by either method can be equivalent provided the measurement resolution is the same [10], hence spectral analysis of the samples can be performed by a fast Fourier transform (FFT) [23] [25], fast Hartley transform (FHT) [24] or Winograd transform [26].

The sampling rate in traditional A/D conversion must be at least twice the frequency of the highest frequency component of the band-limited input signal for spectral analysis (i.e. the Nyquist rate). A similar situation applies to zero crossings where the number of samples produced must be equal to the number of real zero crossings provided the signal has been transformed such that it has full complement of real zero crossings. This also conforms to the Nyquist sampling theorem since the frequency of the transforming signal is equal to or greater than the highest frequency component of the input signal hence there are at least 2 zero crossings per period of the highest frequency component of the input.

The input signal shown in Figure 4-2 is a truncated squarewave with the magnitude of the fundamental equal to 10 units but the harmonic components have been phase shifted by -60°, +60°, -45°, +45° and -30°. Figure 4-3 shows the first 16 points of the magnitude spectrum of the reconstructed signal derived from 256 zero crossings and

Figure 4-4 shows the phase spectrum. The signal is reconstructed by the TMS320C30 simulator (see APPENDIX D).

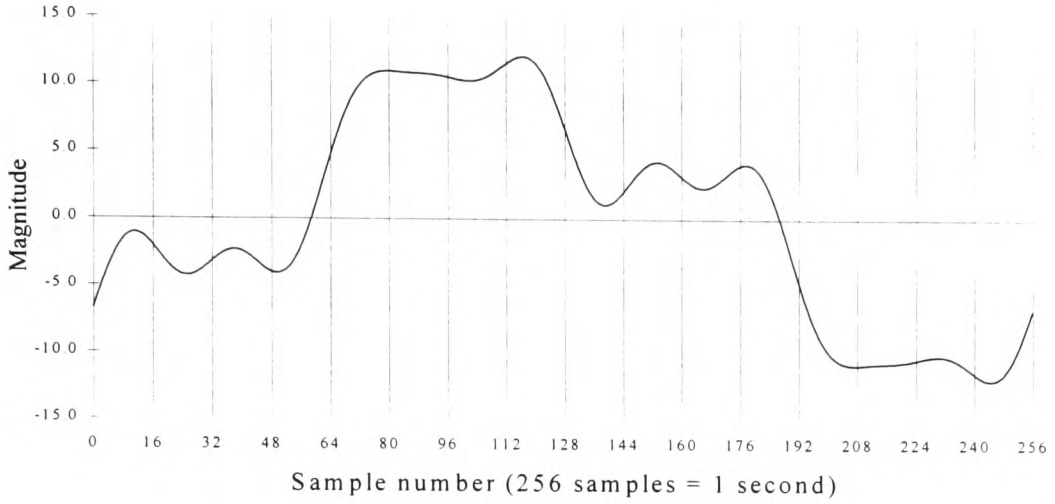


Figure 4-2 *Truncated Squarewave Input Signal with Phase Shifted Components*

Figures 4-3 and 4-4 show that magnitude and phase spectra can be produced with an error of 2.15% with 256 zero crossings and only 8 times the number of samples (i.e. 2048) for linear interpolation. The error can be significantly reduced if the measurement of the zero crossing time is improved by increasing the number of samples as indicated in Figure 3-20.

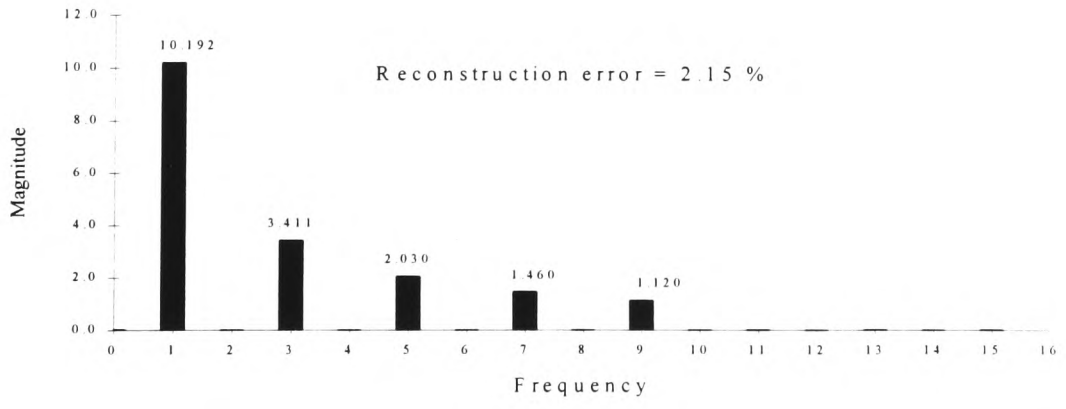


Figure 4-3 *Magnitude Spectrum of the Truncated Squarewave*

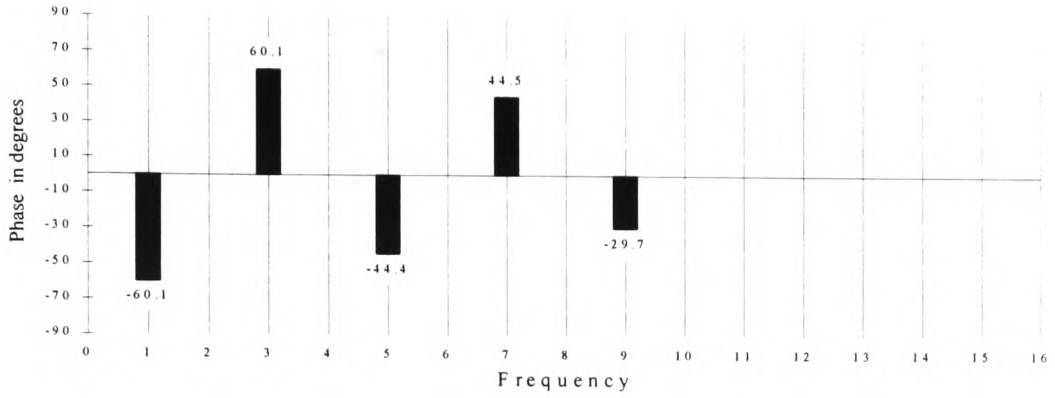


Figure 4-4 *Phase Spectrum of the Truncated Squarewave*

Figure 4-5 shows an input signal which has 5 high frequency components but none of them shifted in phase. Figure 4-6 shows the magnitude spectrum of the reconstructed signal with the reconstruction error just under 1.2%.

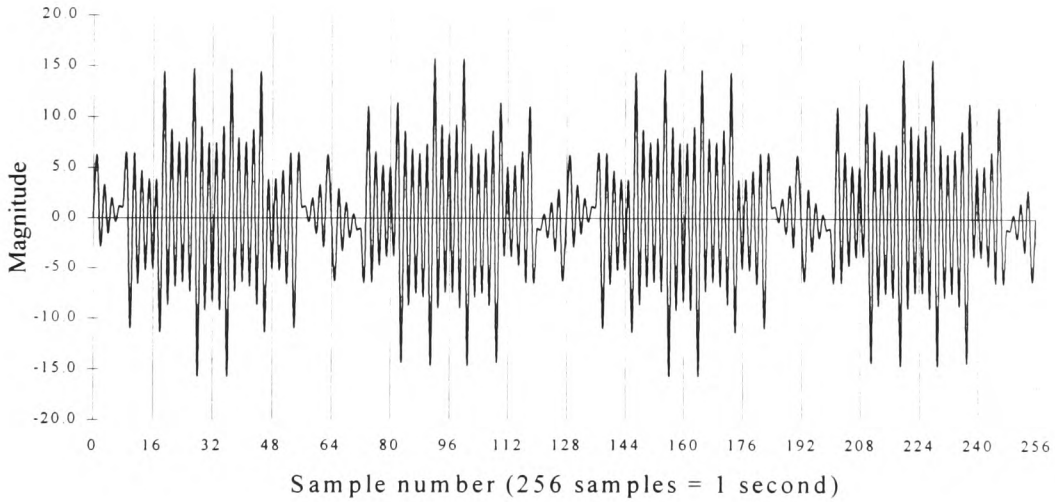


Figure 4-5 *Signal with High Frequency Components*

These two figures have been included to demonstrate that reconstruction and spectral analysis work equally well over the entire spectral range.

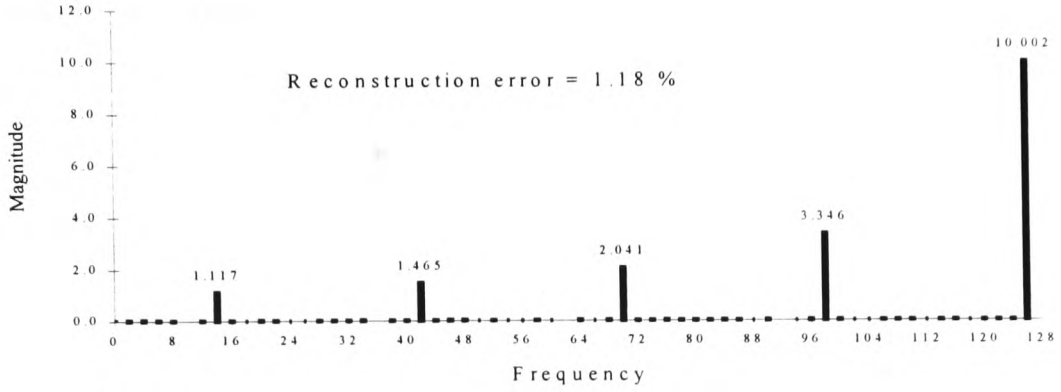


Figure 4-6 *Magnitude Spectrum of the High Frequency Signal*

The FFT function built into the Microsoft Excel V5.0 spreadsheet package has been used to produce Figures 4-3, 4-4 and 4-6.

The signal examples used so far only contain components which are integer multiples of the fundamental resolution frequency. Figure 4-7 shows the magnitude spectrum of a single frequency component of 64.5 Hz which is outside the resolution frequency of 1Hz with a magnitude of 10 units for both the original and reconstructed signal.

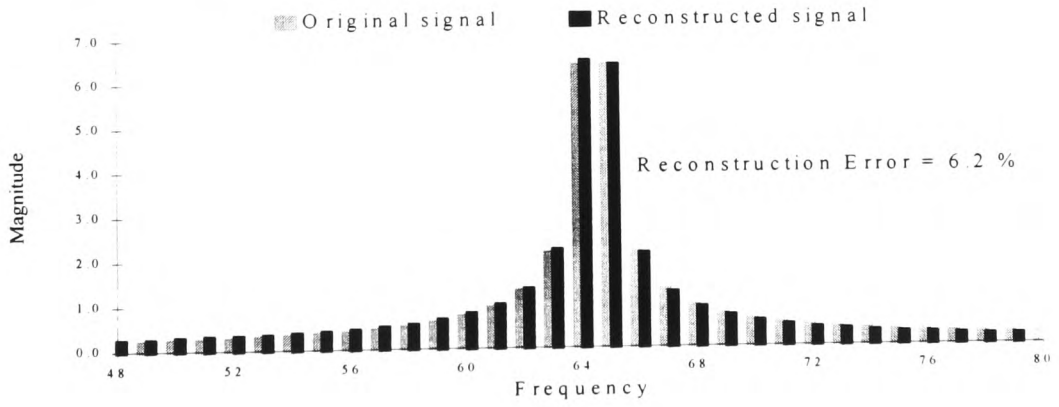


Figure 4-7 *Spectrum of a Signal Outside the Resolution Frequency*

Figure 4-7 demonstrates that the spectral leakage resulting from the 256 point FFT carried out on the reconstructed signal is almost the same as that for the original signal. If the measurement of the zero crossings were more accurate the results would be closer. There are windowing techniques, which have not been pursued in this study, for accurately determining the frequency of signals outside the resolution interval [27]. There are no restrictions using this method of analysis other than those that already apply to traditional A/D conversion.

4.3 DIRECT EXTRACTION OF FOURIER COEFFICIENTS

The Fourier coefficients can be directly calculated from real zero crossing values 't_i' by comparing the coefficients of Z from the following equations [12]:

$$x(z) = C_{(M+1)} Z^{-(M+1)} \prod_{i=1}^{2(M+1)} (Z - Z_i)$$

Where $Z_i = e^{j \omega_0 t_i}$ and t_i are the zero crossings times and ..... (4-1)
$2(M+1)$ = the total number of zero crossings

also

$$x(z) = \sum_{m=-(M+1)}^{(M+1)} C_m Z^m \qquad \text{..... (4-2)}$$

Equations (4-1) and (4-2) have taken into account that there are 2M zeros in the original signal with a further 2 zeros added for the transforming signal which ensures that all zeros are real. A simple example of how the coefficient extraction is achieved can be demonstrated if M = 1 hence:

$$x(z) = C_2 Z^{-2} \prod_{i=1}^{4} (Z - Z_i)$$

$$= C_2 Z^{-2} (Z-Z_1)(Z-Z_2)(Z-Z_3)(Z-Z_4)$$

$$= C_2 \{ Z^2 - (Z_1+Z_2+Z_3+Z_4) Z^1 + (Z_1Z_2 + Z_1Z_3 + Z_1Z_4 + Z_2Z_3 + Z_2Z_4 + Z_3Z_4)\, Z^0$$
$$(Z_1Z_2Z_3 + Z_1Z_2Z_4 + Z_1Z_3Z_4 + Z_2Z_3Z_4)\, Z^{-1} + (Z_1Z_2Z_3Z_4)\, Z^{-2} \}$$

Expanding equation (4-2):

$$x(z) = \sum_{m=-2}^{2} C_m Z^m$$

$$= C_2Z^2 + C_1Z^1 + C_0Z^0 + C_{-1}Z^{-1} + C_{-2}Z^{-2}$$

Comparing coefficients of equations (4-1) and (4-2) for M = 1:

$C_2 = \frac{A}{2}$ where A is the amplitude of the transforming signal

$C_1 = -C_2(Z_1 + Z_2 + Z_3 + Z_4)$

$C_0 = +C_2(Z_1Z_2 + Z_1Z_3 + Z_1Z_4 + Z_2Z_3 + Z_2Z_4 + Z_3Z_4)$

$C_{-1} = -C_2(Z_1Z_2Z_3 + Z_1Z_2Z_4 + Z_1Z_3Z_4 + Z_2Z_3Z_4)$

$C_{-2} = +C_2(Z_1Z_2Z_3Z_4)$

Since $Z_i = e^{j\omega_0 t_i} = \cos \omega_0 t_i + j \sin \omega_0 t_i$ all the coefficients can be calculated from the known values of 't_i'.

As a simple example let the input signal be a single sinusoid and the transforming signal to force real zeros be a sinusoid of twice the frequency then:

$s(t) = A \sin \omega_0 t$ and $x(t) = A \sin 2\omega_0 t$

$s'(t) = A \sin \omega_0 t + A \sin 2\omega_0 t = A \sin \omega_0 t (1 + 2 \cos \omega_0 t) = 0$ for zero crossing

If $\omega_0 = 2\pi / T$ and $T = 1$

then:

$t = 0\ , \frac{1}{2}$ and 1 for $A\sin\omega_0 = 0$ and

$t = \frac{1}{3}$ and $\frac{2}{3}$ for $(1+2\cos\omega_0 t) = 0$ hence

$t_1 = \frac{1}{3};\quad t_2 = \frac{1}{2};\quad t_3 = \frac{2}{3};\quad t_4 = 1$

The values of 'Z_i' can thus be calculated from these values of 't_i' and the Fourier coefficients 'C_i' are then:

$C_2 = \frac{A}{2}$ where A is the amplitude of the transforming signal

$C_1 = -C_2(-1) = +\frac{A}{2}$

$C_0 = +C_2(0) = 0$

$C_{-1} = -C_2(+1) = -\frac{A}{2}$

$C_{-2} = +C_2(-1) = -\frac{A}{2}$

This is the expected result since the amplitudes of both the input and the transforming signals were both 'A'. Figure 4-8 shows the double sided spectrum for the example described with the spectral lines for the transforming signal included (i.e. C_{-2} and C_2).

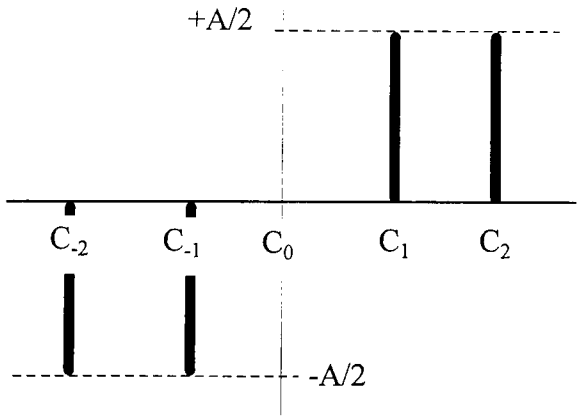


Figure 4-8 *Spectrum for Simple Sinusoid with 4 Zero Crossings*

In practice the spectral lines for the transforming signal can be omitted and only a single sided spectrum need be calculated due to the exact symmetry which holds true for any number of zero crossings.

Kay and Sudhaker [12] suggest that this method becomes impractical if the total number of zero crossings, which includes the contribution of the transforming signal, is greater than 10 (i.e. 2 (M + 1) > 10). It is also suggested that the number of partial products increase with the factorial of 2(M+1) but it can be deduced from the expansion of equations (4-1) and (4-2) that the number of partial product terms to produce a single spectral line is the combination of all 'Zi' values taken 'r' at a time (i.e. ${}^{n}C_{r}$) where n = 2(M+1) and r is the spectral line number. The total number of partial products to produce all lines of the spectrum follows from Pascal's triangle and is 2^{n} which is considerably less than n! but still very large. If the number of zero crossings were 32 then the total number of partial products would be 2^{32} (i.e. 4.29×10^{9}) and the 17th spectral line would require a maximum of ${}^{32}C_{16}$ (i.e. 6.0×10^{8}) partial products added together. The calculations to produce a complete spectrum would take a very long time even with modern DSP processors. This method has been covered in some detail because, in spite of its impracticability, the principle of comparing coefficients is demonstrated.

4.4 NEWTON'S FORMULA TO CALCULATE FOURIER COEFFICIENTS

The second method of spectral analysis directly calculated from real zero crossing values uses Newton's formula [28: Ch 1-18]. The calculation consists of two stages :

Stage 1: $$p_{(k)} = \sum_{i=1}^{L} Z_i^k \quad \text{where } k = 1,2,\cdots L$$

$$p_{(1)} = Z_1 + Z_2 + Z_3 + Z_4 + \cdots Z_L$$

$$p_{(2)} = Z_1^2 + Z_2^2 + Z_3^2 + Z_4^2 + \cdots Z_L^2$$

$$\vdots$$

$$p_{(L)} = Z_1^L + Z_2^L + Z_3^L + Z_4^L + \cdots Z_L^L$$

Stage 2: $$a_{(k)} = -\frac{1}{k} \sum_{i=1}^{k} p_{(i)}\, a_{(k-i)} \quad \text{where } k = 1,2,\cdots L$$

$$a_{(1)} = -\frac{1}{1}(p_{(1)}\, a_{(0)})$$

$$a_{(2)} = -\frac{1}{2}(p_{(1)}\, a_{(1)} + p_{(2)}\, a_{(0)})$$

$$a_{(3)} = -\frac{1}{3}(p_{(1)}\, a_{(2)} + p_{(2)}\, a_{(1)} + p_{(3)}\, a_{(0)})$$

$$\vdots$$

$$a_{(L)} = -\frac{1}{L}(p_{(1)}\, a_{(L-1)} + p_{(2)}\, a_{(L-2)} + p_{(3)}\, a_{(L-3)} + \cdots p_{(L)}\, a_{(0)})$$

Where $a_{(0)}$ = Magnitude of the transforming signal.

The number of partial products to be calculated is reduced to only $\frac{L^2}{2}$ where L = the total number of zero crossings. Calculation of the values of Z_i^k is easily carried out using a spreadsheet if they are converted to harmonic form (i.e. $Z_i^k = e^{jk\omega_0 t_i} = \cos k\omega_0 t_i + j \sin k\omega_0 t_i$) and each value calculated using the polynomial method for sine and cosine values. An alternative is to calculate all the values of '$\cos \omega_0 t$' and then recursively calculate the values of '$\cos k\omega_0 t$' using the relationship '$\cos k\theta = 2\cos(k-1)\theta \cos\theta - \cos(k-2)\theta$'. The advantage is that there is effectively only one multiplication and one subtraction operation required for each value since the multiplication by 2 need only be done once per zero crossing. This is clearly much faster than using a polynomial calculation. A look-up table for all the

possible values of 'cos $\omega_0 t$' similar to that described in section 3.5 is also possible. The first stage can be performed sequentially as the zero crossing values arrive hence the second stage can follow immediately after the arrival of the last value in the sequence.

The second stage of the calculation can be performed quickly if the real part of Z_i^k only is used, otherwise the evaluation would involve the multiplication together of a very large number of pairs of complex numbers (i.e. both $p_{(i)}$ and $a_{(k-i)}$ would be complex). If the equations of the second stage are expanded the results would be exactly the same for each spectral line $a_{(k)}$ as the comparison of coefficients method discussed in section 4.3 which in turn would require the evaluation of 2^n partial products.

A spreadsheet (see APPENDIX E) is used to evaluate the method using only the real part of Z_i^k . Figure 4-9 shows an input signal which consists of the fundamental of magnitude 10 units, the first 4 odd harmonics of a truncated squarewave and a dc

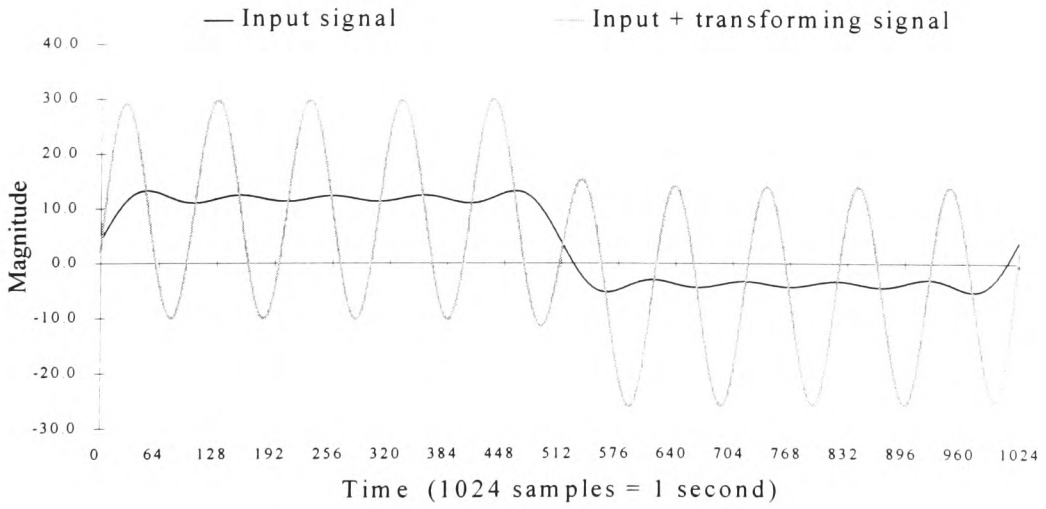


Figure 4-9 *Truncated Squarewave with DC Component and Zero Phase Shifts*

component of 2 units with none of the components phase shifted. The resulting complete spectrum is shown in Figure 4-10 which yields the correct magnitude values and sign similar to an FFT but the DC component is absent.

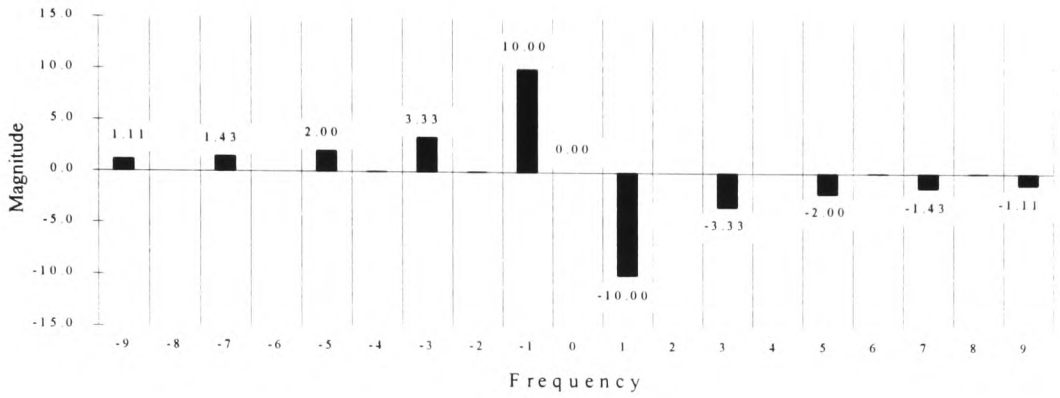


Figure 4-10 *Spectrum of Zero Phase Shifted Squarewave*

Figure 4-11 shows a signal similar to that of Figure 4-9 except that all components are phase shifted by 90°. The resulting spectrum is shown in Figure 4-12. In this case the magnitudes are correct including the dc component.

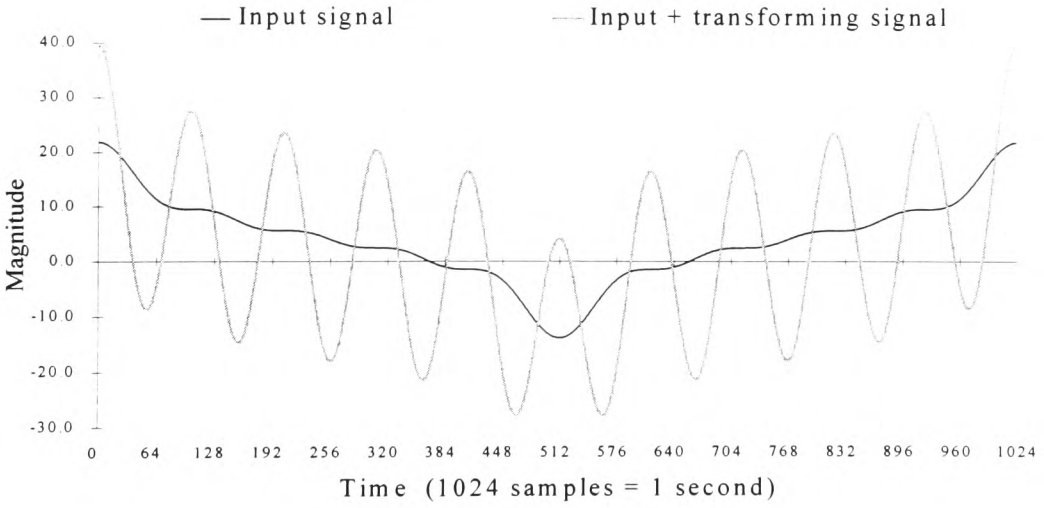


Figure 4-11 *Truncated Squarewave with DC Component and 90° Phase Shifts*

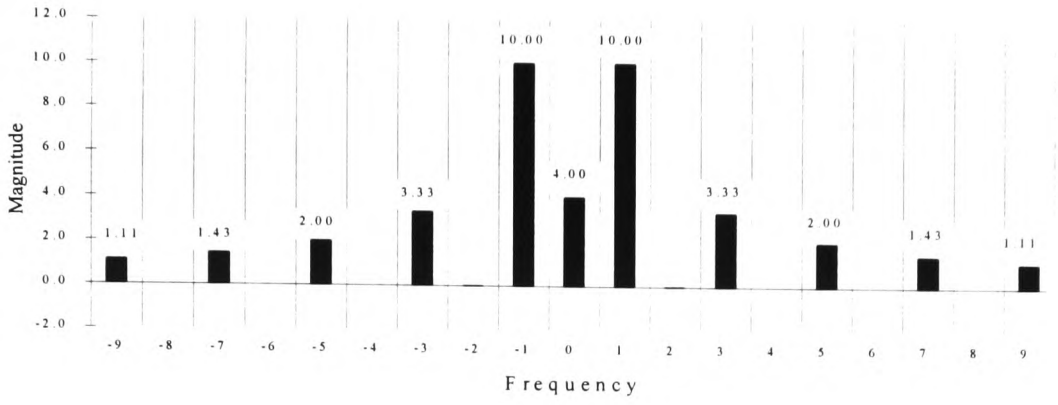


Figure 4-12 *Spectrum of 90° Phase Shifted Squarewave*

Further tests have shown that if all components are phase shifted by the same angle, the first half of the spectrum always yields the correct magnitudes except for the dc component because all the imaginary parts of Z_i^k sum to zero except the dc component. The dc component can be regarded as a cosine wave of zero frequency which explains why the correct spectrum is yielded for all components shifted by 90°. If any component has a different phase to the others the real and imaginary parts of Z_i^k must be taken into account to correctly yield the spectrum. The Newton's formula method is limited in its application but has been used successfully in optics where signals to be analysed were known to be coherent [8].

The Newton formula method is recursive and is therefore prone to accumulated errors hence care must be taken to ensure that the accuracy of measuring the zero crossing is high and the precision of the processor used is adequate. The recursion method for calculating the first stage does not introduce a significant error with 64 zero crossings.

4.5 SEQUENTIAL EXTRACTION OF FOURIER COEFFICIENTS

The third method, also outlined in Kay and Sudhaker's paper [12], of spectral analysis directly calculated from real zero crossing values uses a sequential method of comparing the coefficients of 'Z' from the following equations:

$$P_{(k)}(Z) = \prod_{i=1}^{k}(Z - Z_i) = Z^k + \sum_{i=1}^{k} a_i^{(k)} Z^{k-i}$$

$$P_{(k+1)}(Z) = (Z - Z_{k+1}) \prod_{i=1}^{k}(Z - Z_i) = Z^{k+1} + \sum_{i=1}^{k+1} a_i^{(k)} Z^{k+1-i}$$

comparing the coefficients of Z for $P_{(k)}(Z)$ and $P_{(k+1)}(Z)$ yields:

$$a_i^{(k+1)} = a_i^{(k)} - Z_{k+1}\, a_{i-1}^{(k)}$$
$$a_{k+1}^{(k+1)} = \quad - Z_{k+1}\, a_k^{(k)}$$

where $a_0^{(k)} = 1$ for all values of k and
$a_i^{(0)} = 0$ i.e. no zero crossings collected

The subscript of 'a' denotes the 'i th' spectral line and the superscript denotes the number of zero crossing values collected. The calculation is completed only when k+1 = 2(M+1) for example $a_3^{(6)}$ represents the third spectral line after 6 zero crossing values have been collected. If the total number of zero crossings is 6 then the spectral line is complete. A simple example to demonstrate this algorithm is to let the total number of zero crossings equal 4:

$$a_1^{(1)} = a_1^{(0)} - Z_1 a_0^{(0)} = -Z_1 a_0^{(0)} \qquad = -Z_1$$
$$a_1^{(2)} = a_1^{(1)} - Z_2 a_0^{(1)} = -Z_1 a_0^{(0)} - Z_2 a_0^{(1)} \qquad = -Z_1 - Z_2$$
$$a_1^{(3)} = a_1^{(2)} - Z_3 a_0^{(2)} = -Z_1 a_0^{(0)} - Z_2 a_0^{(1)} - Z_3 a_0^{(2)} \qquad = -Z_1 - Z_2 - Z_3$$
$$a_1^{(4)} = a_1^{(4)} - Z_4 a_0^{(3)} = -Z_1 a_0^{(0)} - Z_2 a_0^{(1)} - Z_3 a_0^{(2)} - Z_4 a_0^{(3)} \quad = -Z_1 - Z_2 - Z_3 - Z_4$$

$$a_1 = -(Z_1 + Z_2 + Z_3 + Z_4)$$

$$a_2^{(1)} = a_2^{(0)} - Z_1 a_1^{(0)} = 0$$
$$a_2^{(2)} = a_2^{(1)} - Z_2 a_1^{(1)} = -Z_2 a_1^{(1)} = Z_1 Z_2$$
$$a_2^{(3)} = a_2^{(2)} - Z_3 a_1^{(2)} = -Z_2 a_1^{(1)} - Z_3 a_1^{(2)} = Z_1 Z_2 + Z_1 Z_3 + Z_2 Z_3$$
$$a_2^{(4)} = a_2^{(3)} - Z_4 a_1^{(3)} = -Z_2 a_1^{(1)} - Z_3 a_1^{(2)} - Z_4 a_1^{(3)} = Z_1 Z_2 + Z_1 Z_3 + Z_2 Z_3 + Z_1 Z_4 + Z_2 Z_4 + Z_3 Z_4$$

$$a_2 = Z_1 Z_2 + Z_1 Z_3 + Z_2 Z_3 + Z_1 Z_4 + Z_2 Z_4 + Z_3 Z_4$$

In general:

$$a_i^{(L)} = -\sum_{k=i}^{L} Z_k a_{i-1}^{(k-1)} \quad \text{..... (4-3)}$$

Equation (4-3) represents the completed 'i th' spectral line and L = the total number of zero crossings. Table 4-1 shows the resulting pattern of equation (4-3) for L = 6.

	$a_1^{(6)}$	$a_2^{(6)}$	$a_3^{(6)}$	$a_4^{(6)}$	$a_5^{(6)}$	$a_6^{(6)}$
@ t_1	$-Z_1 a_0^{(0)}$					
@ t_2	$-Z_2 a_0^{(1)}$	$-Z_2 a_1^{(1)}$				
@ t_3	$-Z_3 a_0^{(2)}$	$-Z_3 a_1^{(2)}$	$-Z_3 a_2^{(2)}$			
@ t_4	$-Z_4 a_0^{(3)}$	$-Z_4 a_1^{(3)}$	$-Z_4 a_2^{(3)}$	$-Z_4 a_3^{(3)}$		
@ t_5	$-Z_5 a_0^{(4)}$	$-Z_5 a_1^{(4)}$	$-Z_5 a_2^{(4)}$	$-Z_5 a_3^{(4)}$	$-Z_5 a_4^{(4)}$	
@ t_6	$-Z_6 a_0^{(5)}$	$-Z_6 a_1^{(5)}$	$-Z_6 a_2^{(5)}$	$-Z_6 a_3^{(5)}$	$-Z_6 a_4^{(5)}$	$-Z_6 a_5^{(5)}$

Table 4-1 *Sequential Build Up of Fourier Coefficients*

Table 4-2 shows the sequential build up of the Fourier coefficients or spectral lines completely in terms of Z_i which then can be calculated from the cosines and sines of the zero crossing values.

	$a_1^{(6)}$	$a_2^{(6)}$	$a_3^{(6)}$	$a_4^{(6)}$
@ t_1	$-Z_1$			
@ t_2	$-Z_2$	$Z_2(Z_1)$		
@ t_3	$-Z_3$	$Z_3(Z_1+Z_2)$	$-Z_3(Z_2Z_1)$	
@ t_4	$-Z_4$	$Z_4(Z_1+Z_2+Z_3)$	$-Z_4(Z_2Z_1+Z_3Z_1+Z_3Z_2)$	$Z_4(Z_3Z_2Z_1)$
@ t_5	$-Z_5$	$Z_5(Z_1+Z_2+Z_3+Z_4)$	$-Z_5(Z_2Z_1+Z_3Z_1+Z_3Z_2+Z_4Z_3)$	$Z_5(Z_4Z_3Z_2Z_1+\cdots$
@ t_6	$-Z_6$	$Z_6(Z_1+Z_2+Z_3+Z_4+Z_5)$	$-Z_6(Z_2Z_1+Z_3Z_1+Z_3Z_2+Z_4Z_3+Z_5Z_4)$	$Z_6(Z_5Z_4Z_3Z_2Z_1+\cdots$

Table 4-2 *First Four Fourier Coefficients (Spectral Lines) in Terms of Z_i*

It can be seen from Table 4-2 that each completed spectral line is the sum of each column. Each entry of the first column is simply the zero crossing values $-Z_i$. The next column starts with the product of Z_2 and the first entry of the previous column (i.e. $Z_2\ Z_1$), the next entry of the column is the product of Z_3 and the sum of the first two entries of the previous column (i.e. $Z_3\ (\ Z_1 + Z_2\ )$). This recursive process continues until the last zero crossing value is collected when the full spectrum is yielded.

The spectral lines must be calculated in terms of their 'real' and 'imaginary' components using the following relationships:

If $Z_1 = x_1 + j\ y_1$ and $Z_2 = x_2 + j\ y_2$ then $Z_2\ Z_1 = (\ x_2\ x_1 - y_2\ y_1\ ) + j\ (\ x_2\ y_1 + y_2\ x_1\ )$

where $x_1 = \cos \omega_0\ t_1$ and $y_1 = \sin \omega_0\ t_1$

Table 4-3 shows the first two spectral lines in terms of the 'real' and 'imaginary' components.

Real $a_1^{(6)}$ Real $a_2^{(6)}$

@ t_1	$-x_1$	
@ t_2	$-x_2$	$x_2 (x_1) - y_2 (y_1)$
@ t_3	$-x_3$	$x_3 (x_1+x_2) - y_3 (y_1+y_2)$
@ t_4	$-x_4$	$x_4 (x_1+x_2+x_3) - y_4 (y_1+y_2+y_3)$
@ t_5	$-x_5$	$x_5 (x_1+x_2+x_3+x_4) - y_5 (y_1+y_2+y_3+y_4)$
@ t_6	$-x_6$	$x_6 (x_1+x_2+x_3+x_4+x_5) - y_6 (y_1+y_2+y_3+y_4+y_5)$
	Imaginary $a_1^{(6)}$	Imaginary $a_2^{(6)}$
@ t_1	$-y_1$	
@ t_2	$-y_2$	$x_2 (y_1) + y_2 (x_1)$
@ t_3	$-y_3$	$x_3 (y_1+y_2) + y_3 (x_1+x_2)$
@ t_4	$-y_4$	$x_4 (y_1+y_2+y_3) + y_4 (x_1+x_2+x_3)$
@ t_5	$-y_5$	$x_5 (y_1+y_2+y_3+y_4) + y_5 (x_1+x_2+x_3+x_4)$
@ t_6	$-y_6$	$x_6 (y_1+y_2+y_3+y_4+y_5) + y_6 (x_1+x_2+x_3+x_4+x_5)$

Table 4-3 *Fourier Coefficients in Terms of 'Real' and 'Imaginary' Components*

4.5.1 Spreadsheet and TMS320C30 'C' Code Realisation

Once the Fourier coefficients have been expressed in the form shown in Table 4-3 a simple spreadsheet and 'C' programs to run either in MS-DOS or the TMS320C30 floating point processor simulator can be created (see APPENDIX F for details). The magnitude spectrum is calculated from the modulus of the 'real' and 'imaginary' components and the phase spectrum from the argument.

Figure 4-13 shows the familiar truncated squarewave input signal with a magnitude of 10 units for the fundamental frequency and the first four odd harmonics. Figures 4-14 and 4-15 show the resulting spectrum calculated on an 'Excel 5' spreadsheet from 56 and 60 zero crossing values respectively using the recursive sequential method. The accuracy from 56 zero crossings is fairly good with an error of only 0.5% in the magnitude of the fundamental frequency and a small error at zero

frequency as can be seen Figure 4-14. Figure 4-15 shows a significant increase in the errors with the fundamental magnitude error at over 8.0% and large errors between the expected frequencies. If the number of zero crossings were further increased the spectrum would be unrecognisable.

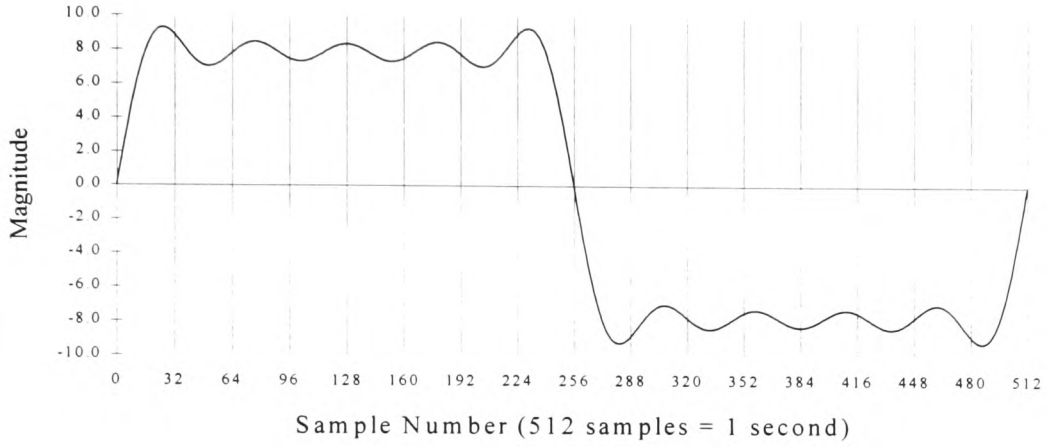


Figure 4-13 *Input Signal - Truncated Squarewave*

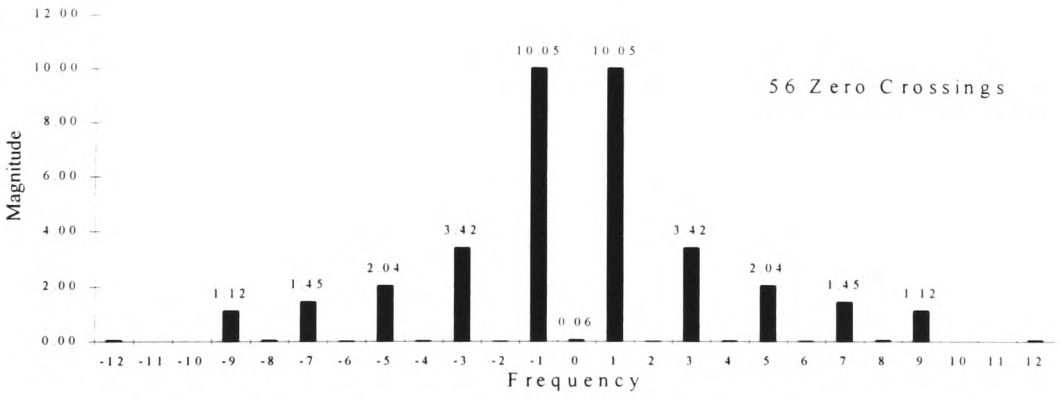


Figure 4-14 *Spectrum Calculated on a Spreadsheet from 56 Zero Crossings*

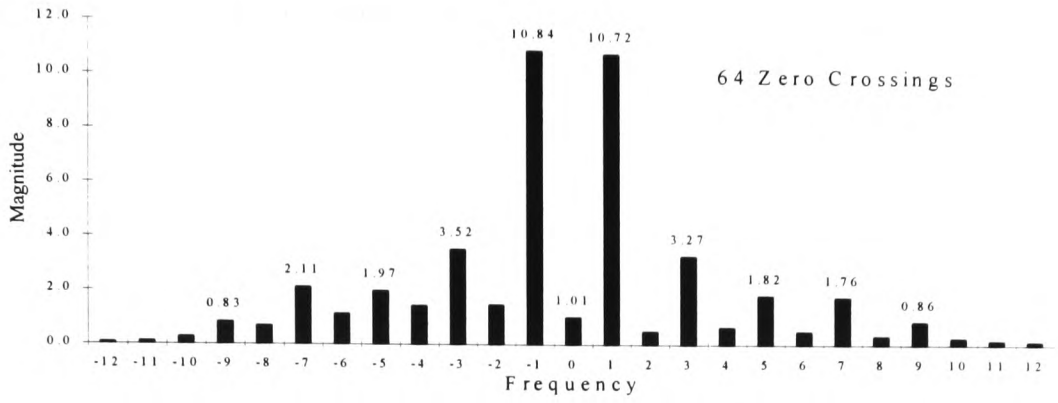


Figure 4-15 *Spectrum Calculated on a Spreadsheet from 64 Zero Crossings*

The reason for the rapid build up of errors is due to the recursive nature of the algorithm and is not a function of the accuracy of measuring the values of the zero crossings. As already explained the calculation of a spectral line depends upon the calculation of the previous one hence any resulting error is included and clearly increases as the number of spectral lines increase. The maximum error occurs towards the centre of the spectrum where the lowest frequencies occur and the number of partial products are a maximum (see section 4.3). The precision of the spreadsheet is high where the largest floating point numbers that can be represented are $\pm$ 9.9999 x 10^{99} and the smallest are $\pm$ 1.000 x 10^{-99}.

The spectra calculated by the TMS320C30 floating point processor simulator from 26 and 30 zero crossing values are shown in Figures 4-16 and 4-17 respectively. It can be seen from Figure 4-16 that the errors are small although there is a significant error at zero frequency. Figure 4-17 shows how the errors increase dramatically when the number of zero crossings is increased to 30. The reason for the relatively poor result compared with the spreadsheet is the much lower precision of the TMS320C30 where

the largest floating point numbers that can be represented are ± 3.4028236 x 10^{38} and the smallest are ± 5.8774717 x 10^{-39} [22: Ch 4.3.2 pp 4-6].

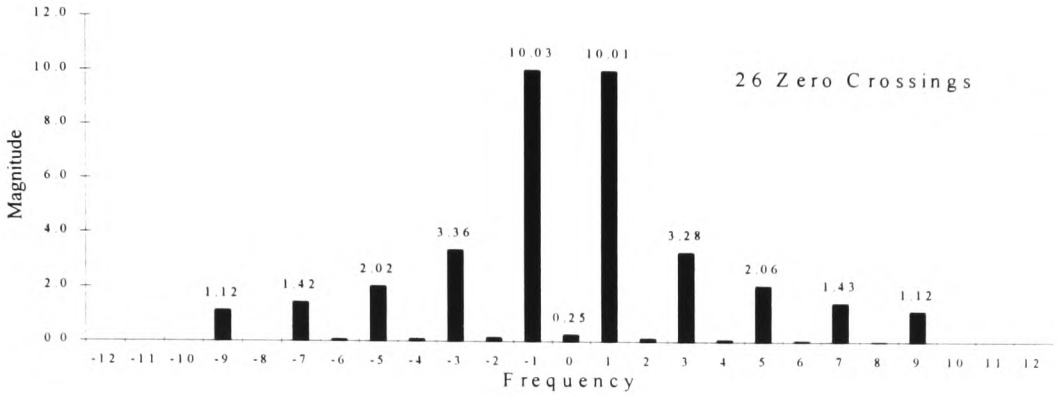


Figure 4-16 *Spectrum Calculated by the TMS320C30 for 26 Zero Crossings*

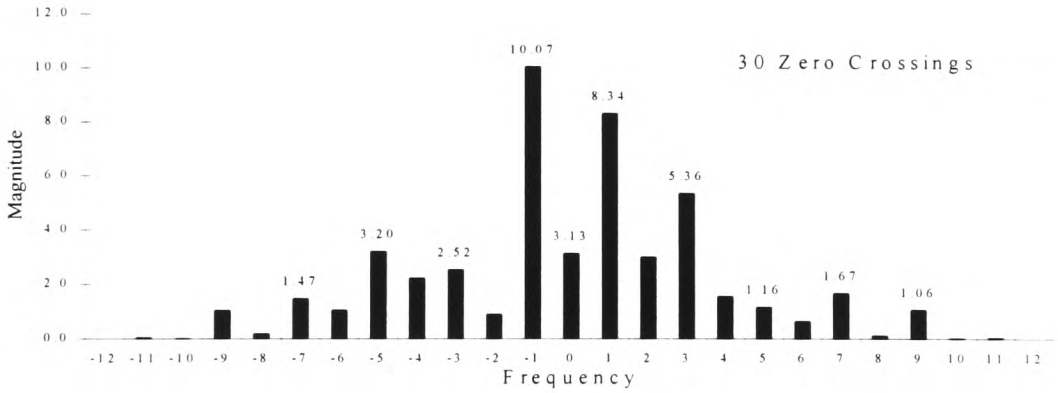


Figure 4-17 *Spectrum Calculated by the TMS320C30 from 30 Zero Crossings*

4.5.2 Program Execution Time

A 'C' program which runs on the TMS320C30 processor simulator has been written for the sequential method using look-up tables (see APPENDIX F). Figure 4-18 shows that direct calculation of the single sided spectrum above 40 zero crossings is

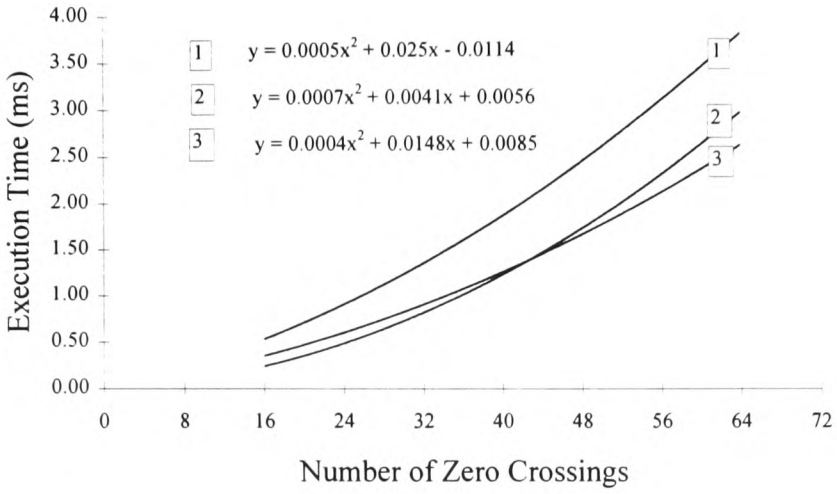


Figure 4-18 *Execution Time for Spectral Analysis and Reconstruction Vs Number of Zero Crossings (using the TMS320C30 with look-up tables)*

Trace 1: Full Spectrum
Trace 2: Reconstruction
Trace 3: Single Sided Spectrum

faster than reconstruction but the recursion errors in calculating the former are so great above 30 zero crossings that the result is unintelligible. The execution time advantage for direct calculation from zero crossing times is therefore completely lost.

4.6 SUMMARY

The four methods of spectral analysis indicated in Figure 4-1 have been considered in detail. The spectrum calculated on the basis of first reconstructing the signal from zero crossings and then using a traditional FFT, described in section 4.2, is the most practicable of the methods considered in this study. The reconstruction algorithm used is non-recursive hence computational errors are small. Spectra generated from 256 zero crossing values estimated to an accuracy of 1 part in 2^{16} using linear interpolation from 2048 samples have been obtained using the TMS320C30 processor simulator with errors of less than 3.0%. The accuracy could be enhanced with more

accurate measurement of the zero crossing values and larger numbers of zero crossings can be accommodated as discussed in Chapter 3.8.3.

Direct extraction of Fourier coefficients is impracticable as explained in section 4.3 due to the very large number of partial products that have to be calculated. Newton's formula method, described in section 4.4, is practicable [8] but limited to coherent signals and is ultimately prone to recursion errors. The final method of sequential calculation, described in section 4.5, is the most flexible for direct calculation of spectra from zero crossing values since information of magnitude, frequency and phase is easily obtained but the recursion errors limit its use to less than 30 zero crossings which is a severe restraint.

CHAPTER 5 APPLICATIONS

5.1 INTRODUCTION

Zero crossing techniques have been used in a variety of applications in image processing [13], speech recognition [30], modulation systems and more fundamentally could be used as a means of analogue to digital conversion [10]. The latter enables the techniques to be used in almost any area of digital signal processing in conjunction with already well established analysis algorithms.

5.2 ANALOGUE TO DIGITAL CONVERSION

Signal reconstruction described in Chapter 3 demonstrates that analogue to digital conversion can be achieved by transforming a signal and then measuring the resulting real zero crossing times. This is in contrast to the traditional method of sampling and measuring the magnitudes of the signal at the sampling times. The major advantage of the zero crossing method is that hardware to measure time intervals [9] is simpler and inherently more accurate than that which measures amplitude. An effective zero crossing detector circuit has been suggested by Saloma and Haeberli [8][9] which could be used, with minor modifications, incorporating the TMS320C30 processor.

5.3 DATA COMPRESSION

The discussion on look-up tables in Chapter 3 section 3.5 demonstrates that the zero crossing values can be defined as an 8 bit binary number (i.e. 256 values) which is added to the Nyquist interval, also defined as an 8 bit binary number as indicated by

equations (3-3) and (3-4). The combined result is a 16 bit number (i.e. 65536 values). If the number of zero crossings is known and fixed then the values can be represented by only 8 bits since the Nyquist intervals can be built into the processor memory. This is a form of data compression without losing any information and is therefore not an approximation. This could be used to halve the transmission bandwidth of a digital audio signal or to double the data density of a recording medium such as 'Compact Disc' [17]. The output filter design of the reconstruction device would be greatly simplified if over sampling is included as described in Chapter 3 section 3.6.

The reconstructing device could be a standard digital signal processor such as the TMS320C30, but in order to obtain the high throughput the program would have to be written in speed optimised assembler code running on several processors in parallel since the algorithm is ideally suited to parallel processing. An alternative would be to design an integrated circuit, incorporating a look-up table, dedicated to the reconstruction algorithm which could be a reasonable proposition for the mass market.

5.4 SPECTRUM ANALYSERS

Spectrum analysers are widely used for industrial and scientific instrumentation. Chapter 4 demonstrates that there are several methods of performing spectral analysis which are based on zero crossing techniques. The best approach appears to be the initial reconstruction of the signal from zero crossing values and then analysing the digital samples using an FFT. Programs to perform an FFT which run on general purpose digital signal processors are available. As a typical example the Texas Instruments TMS320C30 takes 3.75 ms to complete a 1024 point complex FFT [29: Ch 6.5 pp. 71]. A more recent DSP processor with an on-chip look-up table achieves

the same computation in 67μ s [26]. Silicon integrated circuits are also available which are specifically designed to perform high resolution complex FFTs at very high speeds. A recent example is a device which performs an 8192 complex FFT in 400μ seconds [26].

The methods of spectral analysis, so far examined, using zero crossing values directly all have serious limitations. The Newton's formula method however has some applications but is limited to coherent signals [8].

5.5 DEMODULATION

It is possible to use zero crossings techniques to reconstruct an amplitude modulated signal as described in chapter 3 and then process the resulting digitised waveform but this would be an over-complicated and computationally intensive method. Zero crossing detectors, however, have been used as part of a demodulator for frequency modulated (FM) signals [31: pp. 322],[32: pp. 71],[33].

The frequency modulated signal is first limited in the usual manner, a zero crossing detector measures the instant that the limited signal crosses the time axis which produces a squarewave. The time between the positive edges of the squarewave is then measured by a counter whose output is proportional to the instantaneous frequency. The output signal is then converted to an analogue signal using a standard D/A converter system.

An alternative is to generate a pulse of fixed width at each positive edge of the squarewave using a multivibrator circuit and then passing the output of this circuit through an averaging circuit which is in effect a low-pass filter. This gives an

analogue output which is proportional to the instantaneous frequency of the FM signal which in turn represents the original modulation signal. The block diagram of a zero crossing FM demodulator is shown in Figure 5.1.

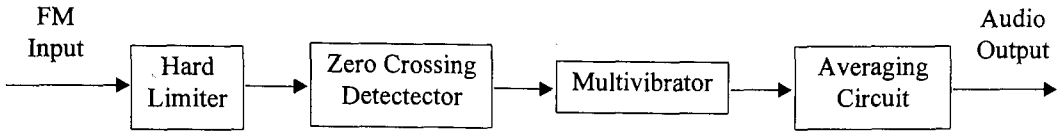


Figure 5-1 *Zero Crossing FM Demodulator Block Diagram* [31]

In this application there is no process of signal reconstruction required since the frequency deviation of the FM signal is proportional to the amplitude of the modulating signal and the rate of change of frequency of the signal is proportional to the frequency of the modulating frequency.

5.6 ZERO SYNCHRONOUS MODULATION (ZSM)

The work of Kay and Sudhaker [12] highlighted the method of invertible transforms to convert complex zeros to real zeros thus greatly simplifying the problem of reconstructing a signal from a combination of its real and complex zeros. There are however applications where the estimation of the real and imaginary parts of complex zeros of a signal is essential. Lockhart et al [34-38] describe methods for locating complex zeros using linear filters with complex transfer functions. The filters convert complex zeros of the input signal to real zeros thus determining the real parts which occur at the time of the minima of the input signal. The real parts can thus be easily determined using zero crossing detectors. The filter parameters are such that a real zero can only be produced at fixed values of the imaginary part of the input signal

hence several filters are required to detect complex zeros with different imaginary values. The values obtained can then be used to generate the conjugate of the complex zero to replace the existing zero. If a binary data source is used to switch the conjugate generator on and off, a phase or frequency modulated signal results which does not significantly affect the amplitude modulated envelope. The general arrangement for phase modulation is shown in Figure 5-2.

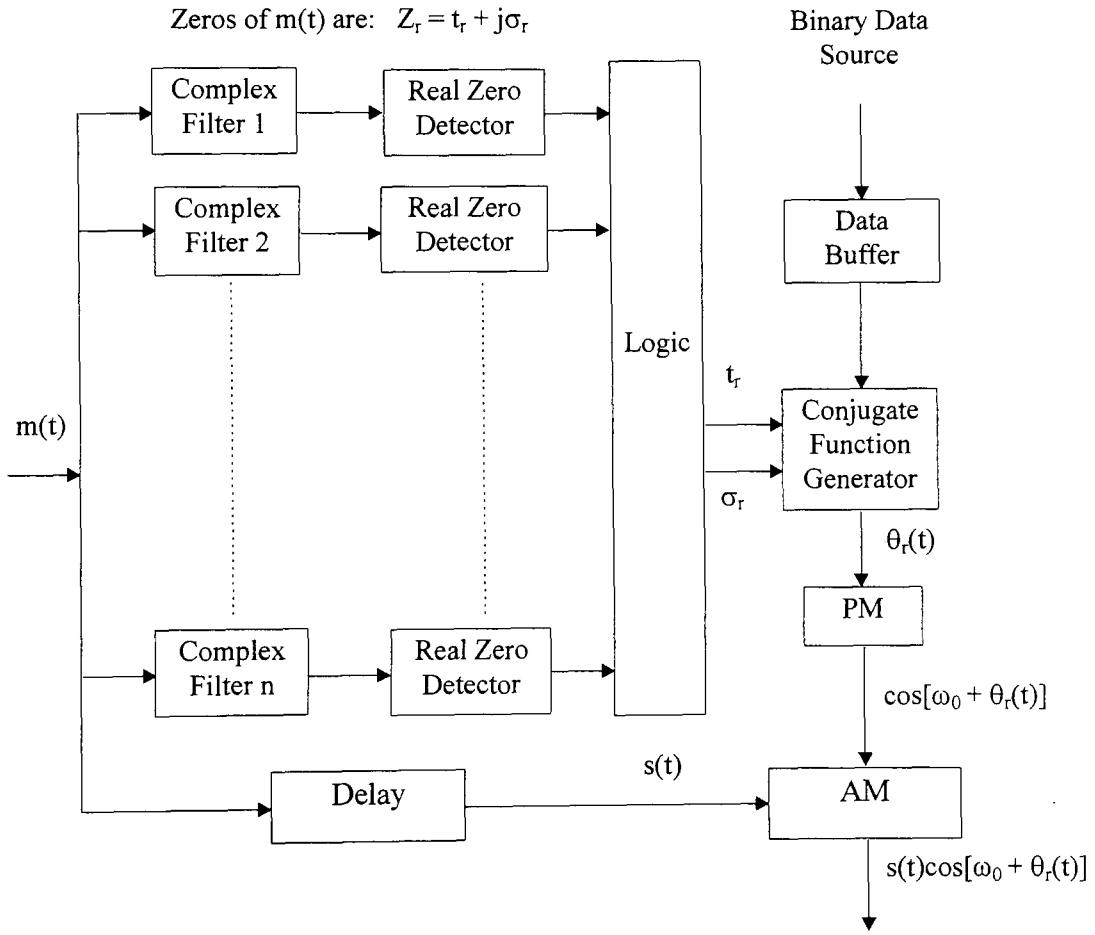


Figure 5-2 *Zero Synchronous Modulation Scheme* [36]

This modulation system depends on complex zeros occurring in conjugate pairs [5][6], and the envelope of a double side band amplitude modulated signal, which consists only of complex zeros, depends on the modulus of the signal $|s(t)|$ [36]:

$$\text{If } z_r = t_r + j\sigma_r \qquad z_r^* = t_r - j\sigma_r$$

$$f_r = [1 - e^{j\Omega(t-z_r)}] \qquad f_r^* = [1 - e^{j\Omega(t-z_r^*)}]$$

$$s(t) = K \prod_{r=1}^{N} f_r f_r^*$$

$$|f_r| = |f_r^*|$$

$$|f_r^*||f_r^*| = |f_r||f_r^*|$$

It can be seen from these equations that if 'f_r' is replaced by its conjugate the modulus of the original signal remains unchanged.

The modulation is achieved by multiplying the delayed signal s(t) by the conjugate function c(t) whose output is either an angle $\theta_r(t)$ for phase modulation (PM) or its derivative $\theta_r'(t)$ for frequency modulation (FM) i.e.:

$$c(t) = \left(\frac{f_r}{f_r^*}\right)^{\pm 1} \quad \text{where} \quad \arg c(t) = \theta_r(t) = \pm 2 \arg f_r = \pm 2 \tan^{-1}[(t - t_r)/\sigma_r]$$

$$\text{and} \quad \theta_r'(t) = \pm 2\,\sigma_r / [(t - t_r)^2 + \sigma_r^2]$$

The final output signal is thus angle and amplitude modulated simultaneously without significantly affecting the bandwidth. This method of modulation is known as 'Zero Synchronous Modulation' (ZSM) because the modulation is synchronised with the occurrence in real time of the complex zeros

The amplitude modulated part of the signal can be envelope detected in the usual way and the data part of the signal detected by any of the conventional frequency demodulators [34].

The advantage of this system over other hybrid modulation schemes is that the data rate that can be achieved is much higher since simple phase or frequency modulation increases the bandwidth of the output signal unless the data rate is very low [34].

5.7 SUMMARY

The applications included in this chapter have been chosen to give an overview of the possibilities suggested in Chapters 3 and 4 which are mainly aimed at analogue to digital conversion (A/D) and data compression which in themselves could be used in a wide variety of engineering applications.

Zero Synchronous Modulation (ZSM) is a system which enables binary data and an amplitude modulated (AM) signal to be transmitted simultaneously with the same bandwidth as the AM signal transmitted on its own. Section 5.6 briefly describing ZSM is included because the application demonstrates almost all of the mathematical principles involved in the study of zeros and modulation.

Analogue to digital conversion from zero crossings is a key application since it leads to a wide variety of digital signal processing applications where it forms the first step in the process.

Spectral analysis of a signal reconstructed from zero crossings is a practicable proposition with the inherent accuracy of zero crossing time measurement.

Generation of Fourier coefficients directly from zero crossings does not appear to be as attractive according to the three methods outlined in Chapter 4.

CHAPTER 6 CONCLUSIONS

6.1 CONCLUSIONS

This study reinforces the theoretical assertion that all the attributes of a real band limited signal can be represented by its zeros, both complex and real, just as the Nyquist samples or Fourier series coefficients do. Transforming a band-limited input signal by adding another signal of suitable magnitude and a frequency that is equal to or higher than the highest frequency component of the original in order to convert complex zeros to real zeros proves to be highly effective. Accurate zero crossing time measurement on a practical system should enable large numbers of zero crossings to be used for band-limited signal reconstruction.

Signal reconstruction and spectral analysis using zero crossing time values have been extensively researched in this dissertation. Four methods of spectral analysis are considered and are as follows:

1. Fast Fourier transform of reconstructed signals (Chapter 4.2)

2. Direct extraction of Fourier coefficients by comparing the Fourier series with a product function (Chapter 4.3)

3. Using Newton's formula to calculate the Fourier coefficients (Chapter 4.4)

4. Sequential extraction of Fourier coefficients (Chapter 4.5)

The first method of spectral analysis proves the most practicable of the four with few limitations. The other three all have serious limitations but the Newton's formula method can be used successfully if the signals to be analysed are known to be coherent [8].

Analogue to digital conversion from zero crossing times is an alternative to the traditional methods with the advantage that measuring zero crossing times with a counter is inherently more accurate than measuring signal amplitude. Detecting zero crossing times can be achieved with an electronic circuit incorporating simple operational amplifiers, logic elements and an accurate sinewave generator for the transforming signal [9]. A further bonus is that time measured to a resolution of one part in 65536 can be represented by an 8 bit binary number rather than 16 thus achieving a compression ratio of 2:1. High speed A/D conversion can be achieved using parallel processing which is ideally suited to the reconstruction algorithm.

The study includes several techniques which optimise the execution of programs that calculate a reconstruction or spectrum of an input signal. These include separating the reconstruction equation into four components, which speeds up program execution times, the generation of efficient look-up tables and protection from overflow and underflow such that any number of zero crossings can be used for calculation.

Methods of reconstruction and spectral analysis were studied using the Lotus 1-2-3 V2.01 and Microsoft Excel V5.0 spreadsheet packages together with the Texas Instruments TMS320C30 floating point processor simulator in conjunction with the TMS320C3X 'C' compiler [16][39][40]. The use of these software packages has

resulted in a practicable scheme for reconstruction and spectral analysis from zero crossing time values.

The original aims of this study were to investigate timed zeros, in particular zero crossings, review the work of other researchers and produce a practicable scheme for spectral analysis. These aims have been achieved.

6.2 FURTHER WORK

There is much to be done to show that A/D conversion using zero crossing times performs as well as if not better than the more traditional methods. A converter working in real time needs to be built and a theoretical analysis of such a system working in a noisy environment carried out and compared with experimental figures.

An efficient program written in assembler language to run on a target machine working in real time in conjunction with a zero crossing detector would fully yield the potential speed of the reconstruction algorithm. The source code for the programs in this dissertation are written in the high level 'C' language, which is then compiled to generate assembler code.

A working A/D converter incorporating parallel processing would greatly increase the bandwidth of an input signal. A working model would demonstrate this and could lead to the design of an application specific integrated circuit (ASIC) to further improve conversion time.

The method of using the invertible transform to convert complex zeros to real zeros may be a useful technique in speech recognition [30].

The work on ZSM is a particularly interesting example of the use of zeros. It depends on the estimation of the real and imaginary components of the zeros contained in a signal using an array of complex filters. This method quite clearly works well but requires a large amount of signal processing. There is some scope for finding a simpler alternative to complex filtering.

All the methods for calculating a spectrum directly from zero crossings proved to be severely limited due to recursion errors. Finding a fast algorithm for calculating Fourier coefficients directly from zero crossing times which does not suffer from computation errors would be an attractive goal.

REFERENCES

[1] CHASSAING, R.: ' Digital Signal Processing with C and the TMS320C30, (John Wiley & Sons, 1992), ISBN 0-471-55780-3

[2] TITCHMARSH, E.C.: ' The Zeros of Certain Integral Functions' *Proc. Lond. Maths. Soc.*, 14 May 1926, vol 25, pp. 283-302

[3] LICKLIDER, J.C.R., and POLLACK, I.: ' Effects of Differentiation, Integration, and Infinite Peak Clipping upon the Intelligibility of Speech' *J. Acoust. Soc. Amer.*, 1948, **20-1**, pp. 42-51

[4] BOND, F.E., and CAHN, C.R.: 'On sampling the zeros of bandwidth limited signals', *IRE Trans. Inform. Theory.*, 1958, **IT-4**, pp. 110-113

[5] VOELCKER, H.B.: 'Towards a unified theory of modulation, Part I: Phase envelope relationships', *Proc. IEEE.*, 1966, **54-3**, pp. 340-353

[6] VOELCKER, H.B.: 'Towards a unified theory of modulation, Part II: Zero manipulation', *Proc. IEEE.*, 1966, **54-5**, pp. 735-755

[7] SEKEY, A.: 'A computer simulation study of real-zero interpolation', *IEEE. Trans.*, 1970, **AU-18**, pp. 43-54

[8] SALOMA, C., and HAEBERLI, P.: 'Optical spectrum analysis from zero crossings', *OPTICS LETTERS,* 1991, **16-19**, pp. 1535-15

[9] SALOMA, C., and DARIA, V.R.: 'Performance of a zero-crossing spectrum analyzer', *OPTICS LETTERS,* 1993, **18-17**, pp. 1468-1470

[10] REQUICHA, A.A.A.: ' The Zeros of Entire Functions: Theory and Engineering Applications', *Proc. IEEE.*, 1980, **68-3**, pp. 308-328

[11] BAHER, H.: ' Analogue & Digital Signal Processing', (John Wiley & Sons, 1990), ISBN 0-471-92342-7

[12] KAY, S.M., and SUDHAKER, R.: 'A zero crossing-based spectrum analyzer', *IEEE. Trans.*, 1986, **ASSP-34**, pp. 96-104

[13] SALOMA, C., and HAEBERLI, P.: 'Two-dimensional image reconstruction from Fourier coefficients computed directly from zero crossings', *APPLIED OPTICS,* 1993, **32-17**, pp. 3092-3093

[14] PRESS, H., VETTERLING, W.T., TEUKOLSKY, S.A., and FLANNERY, B.P.: ' Numeric Recipes in C - The Art of Scientific Computing', (Cambridge University Press, 1988 2nd edn), ISBN 0-521-43108-5

[15] TEXAS INSTRUMENTS: ' Digital Signal Processing Applications with the TMS320 Family: Theory, Algorithms, and Implementations' (Texas Instruments, 1990, vol. 3, document no. SPRA017)

[16] TEXAS INSTRUMENTS: ' TMS320C3X C Source Debugger User's Guide' (Texas Instruments, 1993, revision G, document no. SPRU053B)

[17] WATKINSON, J.R.: ' Compact disc players - 2', *ELECTRONICS & WIRELESS WORLD, Nov 1985,* pp. 29-33

[18] HEATH ZENITH - CONTINUING EDUCATION.:, ' Electronics Technology Series - ' Active Filters' ', (Heath Company, Benton Harbour, Michigan, 1979 1st edn)

[19] POULARIKAS, A.D., and SEELY, S.: ' Signals and Systems', (PWS Engineering, Boston, 1985)

[20] KRAUSS, T.P., SHURE, L., and LITTLE, J.N.: ' Signal Processing Toolbox User,s Guide', (The MATH WORKS Inc, 1992)

[21] KREYSZIG, E.: 'Advanced Engineering Mathematics', (John Wiley & Sons, 1988, 6th edn.), ISBN 0-471-62787-9

[22] TEXAS INSTRUMENTS.: ' TMS320C3X User's Guide' (Texas Instruments, 1991, revision E, document no. SPRU031B)

[23] COOLEY, J.W., and TUKEY, J.W.: ' An Algorithm for the Machine Calculation of Complex Fourier Series', *Maths. Comput.,* 1965, **19**, pp. 297-301

[24] BRACEWELL, R.N., ' The Hartley Transform', (Oxford University Press, 1986), ISBN 0-195-03969-6

[25] BLAIR, G.M.: ' A review of the discrete Fourier transform Part 1: Manipulating the powers of two', *Electron. & Commun. Eng. J.,*August 1995, **7-4**, pp. 169-177

[26] BLAIR, G.M.: ' A review of the discrete Fourier transform Part 2: Non-radix algorithms, real transforms and noise', *Electron. & Commun. Eng. J.,*October 1995, **7-5**, pp. 187-194

[27] BROWN, L.A., KNIGHT, J.A.G., CROFT, A., and HARGREAVES, M.: 'Acoustic Imaging Using Nearfield Holography, *IEE. Proc. Int. Conf. Acoustic Sensing and Imaging,* 1993, pp. 239-244

[28] GIACOLLETO, L.J.: ' Electronics Designers Handbook', (McGraw-Hill, New York, 1977 2nd edn, pp. 1-18, 1-19)

[29] JONES, N.B., and MCK WATSON, J.D.: ' Digital Signal Processing: principles, devices and appications', (Peter Peregrinus on behalf of the IEE., London, 1991), ISBN 0-86341-210-6

[30] RUBENSTEIN, R.H.: ' Time Domain Analysis Yields Powerful Voice Recognition', *New Electronics on Campus,* Autumn 1994, pp. 6-8

[31] LATHI, B.P.: ' Modern Digital and Analogue Communication Systems', (Saunders College Publishing, 1989 2nd edn.), ISBN 0-03-029802-4

[32] DUNLOP, J. and SMITH, D.G.: ' Telecomunications Engineering', (Van Nostrand Reinhold (UK),1984), ISBN 0-442-30586-9

[33] WILEY, R.G., and CARMICHAEL, W.R.: 'A practical procedure for estimation of instantaneous frequency', *IEEE. Trans. Inst. Measurement., 1981,* **IM-30 (1)**, pp. 76-79

[34] LOCKHART, G.B.: ' A Spectral Theory for Hybrid Modulation', *IEEE. Trans. Comms.*, 1973, **COM-21**, vol 7, pp. 790-800

[35] LOCKHART, G.B., and AL-JALILI, Y.O.: ' Method for Superimposing Data on Amplitude Modulated Signals' *ELECTRONICS LETTERS*, 1982,**18-9**, pp. 379-381

[36] LOCKHART, G.B., BAYES, T., and MEHDI, A.A.A.: ' Embedding Data in DSB-AM Using Zero-Synchronous Modulation' *ELECTRONICS LETTERS*, 1987, **23-14**, pp. 745- 746

[37] AL-JALILI, Y.O., and LOCKHART, G.B.: ' Towards Real -Time Implementation of a Radio-Data System for AM Sound Broadcasting' *IEE. Proc.*, 1989, **136-1-1**, pp. 20-24

[38] AL-JALILI, Y.O.: 'Analysis and detection algorithms for complex time zeros of bandlimited signals', *IEE. Proc-1.*, 1991, **138**, pp. 189-200

[39] TEXAS INSTRUMENTS: ' TMS320 Floating-Point DSP Assembly Language Tools User's Guide' (Texas Instruments, 1991, revision A, document no. SPRU035A)

[40] TEXAS INSTRUMENTS: ' TMS320C30 Optimizing C Compiler Reference Guide' (Texas Instruments, 1990, revision D, document no. SPRU034D)

APPENDIX A

ENVIRONMENT AND TOOLS

APPENDIX A ENVIRONMENT AND TOOLS

TABLE OF CONTENTS

APPENDIX A ENVIRONMENT AND TOOLS

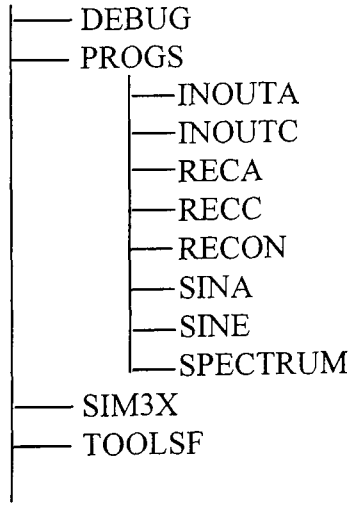
A.1 MICROSOFT DISK OPERATING SYSTEM (MS-DOS)

It is useful to create an ordered directory structure for the project such that files associated with the simulator, batch files to improve the efficiency of program running and user design files are kept separately from each other. If this is done it is a simple matter to find where design files are and keep them up-to-date.

Before the TMS320C30 simulator can be run it is first necessary to repeat several operations such as compiling and linking the 'C' source file. This can be time consuming especially when several modifications or versions of a program are to be run. The instructions for running can be included in a batch file such that only one main command is necessary to invoke a simulation. Section A.1.1 contains the directory structure, the batch files and a typical design directory.

A.1.1 The Directory Structure

Directory PATH listing
Volume Serial Number is 2B1F-0FF9
C:DSF



A.1.2 The Batch Files

The following is the directory which holds the batch files that manage the TMS320C30 simulator program which is held in DEBUG, SIM3X and TOOLSF. The individual design programs for reconstruction, spectral analysis etc. are held in the PROGS directory.

```
Volume in drive C has no label
 Volume Serial Number is 2B1F-0FF9
 Directory of C:\DSF

ACLF        BAT         360    25/04/95     10:47
ASLF        BAT         287    12/05/95     10:30
DSF         BAT       1,000    27/10/95     12:43
CLEAN       BAT         362    24/04/95      8:28
QUIT        BAT         253    27/10/95     12:40
.           <DIR>              10/03/95     18:03
..          <DIR>              10/03/95     18:03
DEBUG       <DIR>              10/03/95     18:03
PROGS       <DIR>              10/03/95     18:03
SIM3X       <DIR>              10/03/95     18:03
TOOLSF      <DIR>              10/03/95     18:03

          11 file(s)          2,262 bytes
             31,940,608 bytes free
```

A.1.2.1 Batch File to Set up the Environment for the TMS320C30

This batch program sets up the path for the individual design directory, checks that the directory exists, sets up the environment variables (SET commands) required by the TMS320C30 simulator and protects against an incorrect directory specification. Simply typing DSF <design name> at the C: prompt is all that is required.

```
@echo OFF
::      DSF.BAT for Drive C:  J A Sherrington 13/6/95  Page 1 of 1
::         Batch file for running DSF programs held on Drive C:

PATH C:\DOS;C:\MOUSE;C:\LOTUS;C:\PKZIP; C:\DSF;C:\DSF\TOOLSF;
C:\DSF\SIM3X;C:\DSF\DEBUG;C:\F-PROT
cls
echo.
if NOT '%1' == ' ' goto load
echo.
echo ***** ENTER A VALID SOURCE CODE DIRECTORY NAME ON *******
 echo                    ***** THE COMMAND LINE ******
echo.
echo ************** PLEASE TRY AGAIN ************************
goto done
:load
if NOT EXIST C:\DSF\PROGS\%1\NUL  goto directory_error
echo.
echo ****** SETTING UP THE ENVIRONMENT FOR THE TMS320C30 *******
echo                          *** DEVELOPMENT SYSTEM ***
echo.
SET PROG=%1
SET C_DIR=C:\DSF\TOOLSF
SET D_DIR=C:\DSF\SIM3X
SET D_SRC=C:\DSF\PROGS\%PROG%
SET D_OPTIONS=-b
c:
cd C:\DSF\PROGS\%PROG%
dir
goto done
:directory_error
echo.
echo ******** DIRECTORY C:\DSF\PROGS\%1  DOES NOT EXIST **********
echo.
echo          ************** PLEASE TRY AGAIN ********************
:done
echo.
```

A.1.2.2 Batch File to Assemble Compile and Link a 'C' Program

This batch file controls the running of the TMS320C30 simulator 'C' compiler such that it can be run from any directory. There is a single command designed by Texas Instruments which carries out the same set of commands in one line but it only works in the TOOLSF directory which is not convenient if several designs are to use the

same simulator. It is essential to keep the design files separate from the simulator program files to avoid accidental deletion or corruption. Typing ACLF at the C: prompt invokes this batch file. Once the source file is compiled the simulator is invoked by typing SIM3X at the C: prompt.

```
@echo off
::                    ACLF.BAT
cls
echo.
echo.
echo  ******* ASSEMBLING COMPILING AND LINKING PROGRAM:
echo                      ******* %PROG%.C *********
echo.
echo                ************* PLEASE WAIT **************
echo.
ac30   -q      %prog%.c
opt30  -qo2    %prog%.if
cg30   -qo     %prog%.opt
asm30  -lq     %prog%.asm
lnk30  -q      %prog%.cmd
```

A.1.2.3 Batch File to Clear Unwanted Files from the Program Directory

Several files are generated during assembly and compiling which are not needed for running the program again. The following batch file deletes them from the current directory which can be initiated by typing CLEAN at the C: prompt.

```
@echo OFF
::              CLEAN.BAT
cls
echo.
echo.
echo   DELETING THE UNWANTED FILES OF PROGRAM *** %PROG% ***

if EXIST      *.OBJ        del *.OBJ
if EXIST      *.MAP        del *.MAP
if EXIST      *.OUT        del *.OUT
if EXIST      *.IP         del *.IP
if EXIST      *.T0         del *.T0
```

```
if EXIST      *.LST          del *.LST
dir
echo.
```

A.1.2.4 Batch File to Quit the Program and Return to the Root Directory

This batch file creates an orderly return to the root directory by restoring the original path and clearing the SET environment variables enabling other applications to be run. The file is initiated by typing QUIT at the C: prompt.

```
@ECHO OFF
::   QUIT.BAT for Drive C:  J A Sherrington 13/6/95  Page 1 of 1

PATH  C:\WINDOWS;C:\DOS;C:\MOUSE;C:\PKZIP;C:\LOTUS;C:\TC\BIN;
C:\WP51;C:\DSP;C:\DSF;C:\F-PROT
SET PROG=
SET D_DIR=
SET D_SRC=
SET C_DIR=
SET D_OPTIONS=
cd\
cls
```

A.1.2.5 Typical Design Directory

The following is a typical design directory which includes the simulator command file, the linker command file, the 'C' source code file, many experimental source files, print files, the assembly file etc. Spreadsheet files which provide input data to the simulator and display output data from the simulator are also contained in this directory. The essential files are in bold characters.

```
Volume in drive C has no label
 Volume Serial Number is 2B1F-0FF9
 Directory of C:\DSF\PROGS\RECON
```

RECON	AN1	10,717	11/10/95	17:29
RECON	AN2	16,215	11/10/95	17:33
RECON	ANP	1,258	13/10/95	17:57
RECON	AS1	10,781	13/10/95	15:54
RECON	AS2	11,162	13/10/95	17:54
RECON	AS3	11,557	13/10/95	17:55
RECON	AS4	13,148	13/10/95	17:55
RECON	AS5	16,527	15/10/95	18:35
RECON	**ASM**	**23,974**	**27/10/95**	**10:54**
RECON	ASP	1,312	25/10/95	18:39
RECON	AX1	10,780	12/10/95	17:30
RECON	AX2	69,469	12/10/95	17:32
RECON	**C**	**16,489**	**27/10/95**	**10:49**
RECON	CMD	1,093	29/08/95	9:41
SIMINIT	CMD	897	30/05/95	16:46
RECON	**OP1**	**396**	**27/10/95**	**10:55**
RECON	**PRN**	**910**	**27/10/95**	**10:54**
Z_2048	PRN	41,020	23/10/95	14:54
RECON_SP	PRN	4,369	13/10/95	23:36
ORIGIN	PRN	5,140	27/10/95	10:54
TAB	PRN	15,027	13/10/95	18:16
SCREEN	**SCR**	**1,124**	**15/03/95**	**15:51**
TAB_CODE	WK1	110,484	26/10/95	13:47
RECON	**WK1**	**260,431**	**15/10/95**	**18:32**
ZC_RECON	WK1	371,542	27/10/95	10:54
ZC_2048	WK1	415,503	23/10/95	14:54
FFT_256	WK1	199,413	01/09/95	10:29
ZC	**WK1**	**379,935**	**25/10/95**	**18:42**
.	<DIR>		24/04/95	8:29
..	<DIR>		24/04/95	8:29

30 file(s) 2,020,673 bytes
32,022,528 bytes free

A.2 TMS320C30 SIMULATOR

The TMS320C30 is fully described in the Texas Instruments data books [15],[16],[39] and [40] but some of the essential elements particular to this dissertation are described in more detail in the following sub-sections.

A.2.1 The Simulator Command File (SIMINIT.CMD)

The following file must be invoked for the simulator to run correctly. It contains information about the processor memory map [16: pp. 3-13], the name of the input file for the simulator, the VDU configuration and the names and locations of the input and output ports. It is particularly important to include the block of memory which holds the memory mapped registers for the IO to function correctly.

The input port is allocated the location 0x0808040 and the output port 0x0808041 which is part of the block 0x0808000 to 0x0809800 (i.e. 6k of RAM). Note that memory allocations must not overlap.

```
; TMS320C30 SIMULATOR COMMAND FILE FOR THE RECON.C PROGRAM

ma 0x000000,0x04000,ram
ma 0x808000,0x0010,ram     ; 0x808010 to 0x80801F reserved memory
ma 0x808020,0x0020,ram
ma 0x808042,0x17be,ram     ; This one and above block defines some of
                           ; the memory mapped registers. These memory maps
                           ; are must for normal execution of simulator.
                           ; If you define your own siminit, please include
                           ; these two.
ma 0x809800,0x400,ram
ma 0x809c00,0x400,ram

map on
load      RECON.OUT        ; Input file name for the simulator
sconfig   SCREEN.SCR       ; VDU configuration file name

ma 0x808040,0x001,IPORT    ; Configure address 0x808040 as an input  port
ma 0x808041,0x001,OPORT    ; Configure address 0x808041 as an output port

mc 0x808040,RECON.PRN,READ  ; Connect the input  port to RECON.PRN
mc 0x808041,RECON.OP1,WRITE ; Connect the output port to RECON.OP1

ba exit
```

A.2.2 The Linker Command File (RECON.CMD)

The linker command file gives the TMS320C30 'C' compiler information about memory configuration, where to start the program, where the standard function libraries are, the name of the output file required and where various sections of the program are.

```
/*    RECON.CMD      COMMAND FILE       */

-cr                 /* SPECIFIES INTERNAL RAM MODE   */
-e c_int00          /* SPECIFIES ENTRY POINT         */
RECON.OBJ           /* OBJECT CODE                   */
-l  rts.lib         /* RUN-TIME SUPPORT LIBRARY      */
-o RECON.OUT        /* LINKED COFF OUTPUT FILE       */
-m RECON.MAP        /* MEMORY MAP FILE               */

MEMORY
{
VECS  : org = 0           len = 0x40     /* VECTOR LOCATIONS  */
 RAM  : org = 0x40        len = 0x3FC0
 RAM  : org = 0x809800    len = 0x800
 RAM  : org = 0x808020    len = 0x17e0
}

SECTIONS
{
 .data:   {} > RAM        /* DATA SECTION IN RAM           */
 .text:   {} > RAM        /* CODE                          */
 .cinit:  {} > RAM        /* INITIALIZATION TABLES         */
 .stack:  {} > RAM        /* SYSTEM STACK                  */
 .bss:    {} > RAM        /* BSS SECTION IN RAM            */
 vecs:    {} > VECS       /* RESET & INTERRUPT VECTORS     */
}
```

A.2.3 Input Data Format

The input data file (e.g. RECON.PRN) must only contain signed integers with no comments or any other text. The comments added here follow the ';' symbol to

explain the range of numbers allowed. The input can be in decimal integer format or the 8 field hexadecimal equivalent. It is more convenient to use decimal format when generated from a spreadsheet.

```
2147483647   ; 2^31-1 largest +ve integer          : hex equivalent  0x7fffffff
0000000003
0000000002
0000000001
0000000000   ; 2^32    equivalent to zero          : hex equivalent 0x00000000
4294967295   ; 2^32-1  smallest -ve integer  ie -1 : hex equivalent 0xffffffff
4294967294
4294967293
4294967292   ; 2^32 - 4    ie -4                   : hex equivalent 0xfffffffc
2147483648   ; 2^31  the largest -ve integer       : hex equivalent  0x80000000
```

A.2.4 Output Data Format

The output data is written to the output data file (e.g. RECON.OP1) in the 8 field hexadecimal format only with no comments or any other text. The hexadecimal numbers must therefore be converted back to decimal in the output spreadsheet before the data can be graphically displayed.

```
0x7fffffff          ; Equivalent to 2147483647 or 2^31-1
..
0x00000001
0x00000000
0xffffffff          ; Equivalent to -1
0xfffffffe          ; Equivalent to -2
..
0x80000000          ; Equivalent to -2147483648 or 2^31
```

A.2.5 Floating Point Numbers

It is not possible to enter floating point numbers as input (e.g. 234.456) directly as already indicated in section A.2.5 but must be first multiplied by 1000 and thus

converted to an integer (i.e. 234456). Once this has been done and entered into the program the numbers can then be converted to floating point format and all floating point operations can be performed on the numbers.

The result of any floating point arithmetic performed by the program will be in floating point format and thus must be converted to an integer before writing to the output data file. The integers in the output file, which will be in hexadecimal format, can then be converted to decimal in the spreadsheet and then divided by the appropriate factor (e.g. 10^3) to obtain the result in floating point decimal format (i.e. 234.456)

A.2.6 'C' Programming

The following program is a simple example of how input data is entered in and how it is written to an output file. The use of the 'volatile int' pointers is to ensure that a 'C' compiler and optimizer cannot alter the memory-mapped addresses for the input/output files. The location of the input files is specified in the simulation command file as input and output ports. In this example one input and two output ports (i.e. INOUTC.IPX, INOUTC.OP1 and INOUTC.OP2) have been specified in the simulation command file for this program. In the example of section A.2.1 only one of each is specified (i.e. RECON.PRN and RECON.OP1)

```
/* INOUTC.C   DEMONSTRATING THE INPUT AND OUTPUT FUNCTION */

main()
{
volatile int *INPUT_1    = (volatile int *) 0x808040;      /* Input  channel 1 */
volatile int *OUTPUT_1 = (volatile int *) 0x808041;        /* Output channel 1 */
```

```
volatile int *OUTPUT_2 = (volatile int *) 0x808042;        /* Output channel 2 */
int count;
int value[10];

for (count=0; count <10; count++)
 value[count]=*INPUT_1;    /* Read  data from INOUTC.IPX ascending order */

for (count=0; count <10; count++)
 *OUTPUT_1 = value[count];  /* Write data to INOUTC.OP1 ascending order */

for (count=0; count <10; count++)
 *OUTPUT_1 = value[9-count]; /* Write data to INOUTC.OP1 descending order */

for (count=0; count <10; count++)
 *OUTPUT_2 = value[9-count]; /* Write data to INOUTC.OP2 descending order */

for (count=0; count <10; count++)
 *OUTPUT_2 = value[count];  /* Write data to INOUTC.OP2 ascending order */
}
```

A.3 SPREADSHEETS

The Lotus 1-2-3 Version 2.01 is a rather old application program but is sufficiently flexible to be suitable for this study. It is extremely easy to export individual columns or rows of data to an external text file and to import data from a text file to a particular area of the spreadsheet. This facility is particularly important when working in conjunction with the TMS320C30 processor simulator which operates in the MS-DOS environment.

A more up-to-date spreadsheet application program such as Microsoft's 'Excel' Version 5 which works in the 'Windows' environment offers many advantages over Lotus 1-2-3 Version 2.01. The graphics facilities are much superior and has some very useful functions such as decimal to hexadecimal conversion and vice versa and a built in fast Fourier transform (FFT). The only disadvantage is the rather awkward method of importing and exporting data. This can be overcome by writing suitable

macros but in spite of this it was decided to adopt a compromise by using Lotus 1-2-3 when working directly with the processor simulator thus taking advantage of the simple interfacing but using 'Excel' Version 5 for printing graphs and running stand-alone spreadsheets. Lotus 1-2-3 graphs can be easily imported into 'Excel' and then modified as desired. Details of the spreadsheets for input data generation and conversion of output data for display to work in conjunction with the TMS320C30 simulator are given in APPENDIX C.

Stand-alone spreadsheets have been extensively used to study the algorithms specific to this project and to provide models for devising 'C' code for both 'Turbo C' and the Texas Instrument's 'C' compiler for the TMS320C30 simulator which both operate in the MS-DOS environment. Details of stand-alone spreadsheets for reconstruction and spectral analysis are given in APPENDIX D, APPENDIX E and APPENDIX F.

APPENDIX B

THE RECONSTRUCTION EQUATION

APPENDIX B THE RECONSTRUCTION EQUATION

If a signal s(t) is represented by a Fourier series then:

$$s(t)=\sum_{k=-n}^{+n} C_k e^{jk\Omega t} \quad \text{..... (B-1)}$$

$$s(t)=C_{-n} e^{-jn\Omega t}\left\{1+\frac{C_{-n+1} e^{j\Omega t}}{C_{-n}}+\frac{C_{-n+2} e^{j2\Omega t}}{C_{-n}}+\frac{C_{-n+3} e^{j3\Omega t}}{C_{-n}}+\cdots\frac{C_n e^{j2n\Omega t}}{C_{-n}}\right\} \text{..... (B-2)}$$

Equation (B-1) can be written in factorial form as follows:

$$s(t)=C_{-n} e^{-jn\Omega t}\prod_{k=1}^{2n}[1-a_k e^{j\Omega t}] \quad \text{where} \quad a_k=e^{-j\Omega\xi_k} \quad \text{and} \quad \xi_k=\tau_k+j\sigma_k$$

$$s(t)=C_{-n} e^{-jn\Omega t}\prod_{k=1}^{2n}[1-e^{j\Omega(t-\tau_k-j\sigma_k)}] \quad \text{..... (B-3)}$$

If the factors of equation (B-3) are separated into two groups, one containing only complex roots where $\sigma_k \neq 0$ and the other containing only real roots where $\sigma_k = 0$ then:

$$s(t)=C_{-n} e^{-jn\Omega t}\prod_{k=1}^{2n-n_R}[1-e^{j\Omega(t-\tau_k-j\sigma_k)}]\prod_{L=1}^{n_R}[1-e^{j\Omega(t-\tau_L)}] \quad \text{..... (B-4)}$$

where:

n_R = The number of real zeros
n_C = The number of pairs of complex conjugate zeros
$2n$ = The total number of zeros

and:

$$n = n_C + \frac{n_R}{2} \quad \text{..... (B-5)}$$

If a single pair of conjugate complex zeros is considered the following applies:

$z_i = \tau_i + j\sigma_i$ and $z_i^* = \tau_i - j\sigma_i$ i.e. the conjugate of z_i then:

$$[1-e^{j\Omega(t-z_i)}][1-e^{j\Omega(t-z_i^*)}] = 1-e^{j\Omega(t-\tau_i-j\sigma_i)}-e^{j\Omega(t-\tau_i+j\sigma_i)}+e^{j2\Omega(t-\tau_i)}$$

$$= e^{j\Omega(t-\tau_i)}\left\{-[e^{\Omega\sigma_i}+e^{-\Omega\sigma_i}]+e^{-j\Omega(t-\tau_i)}+e^{j\Omega(t-\tau_i)}\right\}$$

$$\therefore\ [1-e^{j\Omega(t-z_i)}][1-e^{j\Omega(t-z_i^*)}] = -2e^{j\Omega t}\, e^{-j\Omega\tau_i}\left\{\cosh\Omega\sigma_i - \cos\Omega(t-\tau_i)\right\} \quad \text{..... (B-6)}$$

If a single real zero is considered then the following applies:

$$[1-e^{j\Omega(t-\tau_r)}] = e^{j\frac{\Omega}{2}(t-\tau_r)}\left\{e^{-j\frac{\Omega}{2}(t-\tau_r)} - e^{j\frac{\Omega}{2}(t-\tau_r)}\right\} \quad \text{then}$$

$$[1-e^{j\Omega(t-\tau_r)}] = -2j\, e^{j\frac{\Omega}{2}t}\, e^{-j\frac{\Omega}{2}\tau_r}\, \sin\frac{\Omega}{2}(t-\tau_r) \quad \text{..... (B-7)}$$

Since there must be an even number of real zeros, 'j' is always squared, hence for a pair of real zeros:

$$[1-e^{j\Omega(t-\tau_{r1})}][1-e^{j\Omega(t-\tau_{r2})}] = -2^2\, e^{j\Omega t}\, e^{-j\frac{\Omega}{2}(\tau_{r1}+\tau_{r2})} \sin\frac{\Omega}{2}(t-\tau_{r1})\sin\frac{\Omega}{2}(t-\tau_{r2})$$

Substituting equations (B-6) and (B-7) into equation (B-4) yields:

$$s(t)=C_{-n}e^{-jn\Omega t}\prod_{k=1}^{n_C}(-2)e^{j\Omega t}e^{-j\Omega\tau_k}\left\{\cosh\Omega\sigma_k-\cos\Omega(t-\tau_k)\right\}.$$

$$\prod_{L=1}^{n_R}(-2j)e^{j\frac{\Omega}{2}t}e^{-j\frac{\Omega}{2}\tau_L}\sin\frac{\Omega}{2}(t-\tau_L) \quad \text{..... (B-8)}$$

hence:

$$s(t)=2^{(n_C+n_R)}C_{-n}\,e^{-j\Omega t\left(n-n_C-\frac{n_R}{2}\right)}e^{-j\frac{\Omega}{2}P}\prod_{k=1}^{n_C}\left\{\cosh\Omega\sigma_k-\cos\Omega(t-\tau_k)\right\}\prod_{L=1}^{n_R}\sin\frac{\Omega}{2}(t-\tau_L)$$

..... (B-9)

where:

$$e^{-j\frac{\Omega}{2}P}=e^{-j\frac{\Omega}{2}(2\tau_{K=1}+2\tau_{K=2}+\cdots 2\tau_{K=nC})}\,e^{-j\frac{\Omega}{2}(\tau_{L=1}+\tau_{L=2}+\cdots\tau_{L=nR})} \quad \text{..... (B-10)}$$

Equation (B-10) is the square root of the product of all the roots of equation (B-3). There are two equal values of τ_k for the real parts of the complex conjugate zero pairs which add together and the imaginary parts σ_k cancel, in addition there are an even number of real zeros τ_L. The product of all the roots is equal to the coefficient of the last term of equation (B-2) hence:

$$e^{-j\frac{\Omega}{2}P}=\sqrt{\frac{C_n}{C_{-n}}}=\sqrt{\frac{C_n}{C_n^*}}$$

$$\therefore\ C_{-n}e^{-j\frac{\Omega}{2}P}=\sqrt{C_{-n}C_n}=\sqrt{C_n^*C_n}=|C_n| \quad \text{..... (B-11)}$$

but from equation (B-5) $n_C+\frac{n_R}{2}=n$, hence substituting equations (B-5) and (B-11) in equation (B-9) yields:

$$s(t) = 2^{(n_C + n_R)} \left| C_n \right| \prod_{k=1}^{n_C} \{\cosh\Omega\sigma_k - \cos\Omega(t-\tau_k)\} \prod_{L=1}^{n_R} \sin\frac{\Omega}{2}(t-\tau_L) \quad \text{..... (B-12)}$$

Equation (B-12) is almost the same as equation (2-4) in Chapter 2 except for the scaling factor of $2^{(n_c + n_R)}$ which is equal to 2^n if all the zeros are complex. If all the zeros are real the scaling factor becomes 2^{2n}. Equation (2-5) in Chapter 2 used in this study for the reconstruction of signals containing real zeros only is therefore correct and proves to be so in practice. It remains to be shown in practice that equation (B-12) is correct for signals containing either complex conjugate zero pairs only or a combination of real and complex zeros.

APPENDIX C

INPUT/OUTPUT DATA

APPENDIX C INPUT/OUTPUT DATA

TABLE OF CONTENTS

APPENDIX C INPUT/OUTPUT DATA

C.1 INPUT DATA SPREADSHEET AND LINEAR INTERPOLATION

Figure C-1 shows part of the Lotus 1-2-3 V 2.01 spreadsheet which generates the input signal to be reconstructed, adds the transforming signal, detects and calculates the zero crossing times and then, using a simple macro, sorts the zero crossing values in ascending order. Another macro is used to transfer the zero crossing times data to a text file which then serves as an input to the TMS320C30 simulator.

C.1.1 Cell Formulae to Produce the Input Data

Cell A21: 1

Cell B21: +$B$7*@SIN($R$8*$A21+$R$9)+$C$7*@SIN($S$8*$A21+$S$9)+ $D$7*@SIN($T$8*$A21+$T$9)+$E$7*@SIN($U$8*$A21+$U$9)+ $F$7*@SIN($V$8*$A21+$V$9)+$B$11+$E$11*(@RAND-0.5)

Cell C21: +B21+$G$7*@SIN($W$8*$A21+$W$9)

Cell D21: @IF(C20*C21<=0,(A20+C20/(C20-C21))*$E$5,1000000)

Cell E21: 2764.8185236

Cell F21: @IF(E21>$E$17,"",E21-A21*$F$17)

Cell A21 contains the time increment. Cell B21 contains the formula to generate the input signal. Cell C21 contains the transforming signal added to the input signal. Cell D21 contains the formula to detect the zero crossing and carry out the linear interpolation. This is done by using an @IF function which tests whether a change of sign has occurred. If the statement is true the linear interpolated crossing time value is

returned; if false, a large number is returned which simplifies the sort routine. Cells E20 to E52 contain 32 zero crossing values in ascending order ready for transfer to a input text file. Cell F21 contains the formula to change the data suitable for 'C' code which uses look-up tables to reconstruct the signal (see APPENDICES D and F).

256-Point Zero Crossing Study
Generates Zero Crossing Values Only

No of Samples N =	1024	Res Factor =	64			
Magnitudes of x(t)	10.000	3.333	2.000	1.429	1.111	20.000
Frequencies in Hz	1	3	5	7	9	16
Phase Shift degrees	0	0	0	0	0	90
DC Component =	0.000	Noise Factor =	0			

	Normalised Time n	Original Signal	Input Signal x(n)	Zero Times t(i)	Zeros in Ascending Order t(i)	
					65536	2048 = Table Resolution
Formulae		0.000	20.000	1000000	32	32 = Number of Zero Crossings
					2000	2000 = Magnitude of the Added Signal
	0	0.000	20.000	1000000	1201	1201
	1	0.307	20.210	1000000	2765	717
	2	0.613	20.229	1000000	5374	1278
	3	0.919	20.057	1000000	6932	788
	4	1.223	19.701	1000000	9500	1308
	5	1.526	19.164	1000000	10991	751
	6	1.827	18.457	1000000	13559	1271
	7	2.126	17.586	1000000	15091	755
	8	2.422	16.564	1000000	17677	1293
	9	2.715	15.403	1000000	19209	777
	10	3.005	14.117	1000000	21777	1297
	11	3.291	12.719	1000000	23268	740
	12	3.574	11.227	1000000	25836	1260
	13	3.851	9.657	1000000	27394	770
	14	4.125	8.026	1000000	30003	1331
	15	4.393	6.353	1000000	31567	847
	16	4.656	4.656	1000000	33657	889
	17	4.913	2.953	1000000	36155	1339
	18	5.165	1.263	1000000	37616	752
	19	5.410	-0.395	1201	40179	1267
	20	5.650	-2.004	1000000	41707	747
	21	5.882	-3.546	1000000	44295	1287
	22	6.108	-5.003	1000000	45835	779
	23	6.327	-6.361	1000000	48405	1301
	24	6.539	-7.603	1000000	49899	747
	25	6.743	-8.717	1000000	52469	1269
	26	6.940	-9.689	1000000	54009	761
	27	7.129	-10.509	1000000	56597	1301
	28	7.311	-11.167	1000000	58125	781
	29	7.484	-11.655	1000000	60688	1296
	30	7.650	-11.966	1000000	62149	709
	31	7.807	-12.096	1000000	64647	1159
	32	7.957	-12.043	1000000	1000000	
	33	8.098	-11.806	1000000	1000000	
	34	8.231	-11.385	1000000	1000000	
	35	8.356	-10.783	1000000	1000000	
	36	8.472	-10.005	1000000	1000000	

Figure C-1 *Input Data and Linear Interpolation Spreadsheet*

C.1.2 Macro to Sort Data

The following macro copies the values in cells D20 to D1044 to cells E20 to E1044 and then sorts them in ascending order. The graph showing the input signal is then automatically displayed.

```
Type Alt+A to
run this macro
{CALC}
/RVD20..D1044~E20~
/DSDE20.E1044~
PE20~A~G
{CALC}{GRAPH}
{GOTO}A7~
```

C.1.3 Macro to Transfer Data

The following macro transfers data to text file RECON.PRN, copies the original signal data to text file ORIGIN.PRN and then automatically saves the entire spreadsheet, quits the spreadsheet and returns to the MS-DOS prompt.

```
Type Alt+T to
run this macro
/PF
RECON.PRN~RR
F19..F275~
OOUQGQ
/PF
ORIGIN.PRN~RR
K20..K276~
OOUQGQ
/FS~R/QY
```

This macro demonstrates how simple it is to transfer a range of data within a spreadsheet to an external text file and then to return to the MS-DOS prompt ready to run the simulator. It is not as simple using Microsoft Excel which works in the

'Windows' environment. Lotus 1-2-3 V 2.01 was used exclusively for generating input data for the simulator for these reasons.

C.2 OUTPUT DATA SPREADSHEET

Figure C-2 shows part of the Lotus spreadsheet which imports data, in hexadecimal format, generated by the TMS320C30, converts it to floating point decimal format which can then be graphically displayed.

Lotus 1-2-3- Hexadecimal Conversion Facility

Time	Input Data	Field Separation					Hexadecimal Number Conversion					Recovered Signal Decimal
0	0x00000000	0	0	0	0	0	0	0	0	0	0	0.000
1	0x0000031c	0	0	3	1	c	0	0	3	1	12	7.960
2	0x00000378	0	0	3	7	8	0	0	3	7	8	8.880
3	0x000002c6	0	0	2	c	6	0	0	2	12	6	7.100
4	0x0000030a	0	0	3	0	a	0	0	3	0	10	7.780
5	0x0000034a	0	0	3	4	a	0	0	3	4	10	8.420
6	0x000002ec	0	0	2	e	c	0	0	2	14	12	7.480
7	0x000002ff	0	0	2	f	f	0	0	2	15	15	7.670
8	0x00000342	0	0	3	4	2	0	0	3	4	2	8.340
9	0x000002ff	0	0	2	f	f	0	0	2	15	15	7.670
10	0x000002ec	0	0	2	e	c	0	0	2	14	12	7.480
11	0x0000034a	0	0	3	4	a	0	0	3	4	10	8.420
12	0x0000030a	0	0	3	0	a	0	0	3	0	10	7.780
13	0x000002c6	0	0	2	c	6	0	0	2	12	6	7.100
14	0x00000378	0	0	3	7	8	0	0	3	7	8	8.880
15	0x0000031c	0	0	3	1	c	0	0	3	1	12	7.960
16	0x00000000	0	0	0	0	0	0	0	0	0	0	0.000
17	0xfffffce1	f	f	c	e	1	####	15	12	14	1	-7.990
18	0xfffffc87	f	f	c	8	7	####	15	12	8	7	-8.890
19	0xfffffd3a	f	f	d	3	a	####	15	13	3	10	-7.100
20	0xfffffcf5	f	f	c	f	5	####	15	12	15	5	-7.790
21	0xfffffcb5	f	f	c	b	5	####	15	12	11	5	-8.430
22	0xfffffd12	f	f	d	1	2	####	15	13	1	2	-7.500
23	0xfffffd01	f	f	d	0	1	####	15	13	0	1	-7.670
24	0xfffffcbb	f	f	c	b	b	####	15	12	11	11	-8.370
25	0xfffffd01	f	f	d	0	1	####	15	13	0	1	-7.670
26	0xfffffd12	f	f	d	1	2	####	15	13	1	2	-7.500
27	0xfffffcb5	f	f	c	b	5	####	15	12	11	5	-8.430
28	0xfffffcf5	f	f	c	f	5	####	15	12	15	5	-7.790
29	0xfffffd3a	f	f	d	3	a	####	15	13	3	10	-7.100
30	0xfffffc87	f	f	c	8	7	####	15	12	8	7	-8.890
31	0xfffffce1	f	f	c	e	1	####	15	12	14	1	-7.990
32	0x00000000	0	0	0	0	0	0	0	0	0	0	0.000

Figure C-2 *Output Data Conversion Spreadsheet (Lotus 1-2-3)*

Hexadecimal conversion cannot be performed directly by the Lotus 1-2-3 spreadsheet package hence the procedure of separating out each character of the hex number into an individual cell, converting it to a decimal integer and then adding the resulting numbers multiplied by the appropriate factor of 2^n to yield the full decimal number. Figure C-3 shows part of the Excel spreadsheet to achieve the same function.

	A	B	C	D	E	F	G	H
1	**This file is designed to display the output of the TMS320C30 simulator**							Error % =
2								
3	Time t_1	Time t_2	Time t_3	Time t_4	Hex Data	Signed Decimal Integer	Reconstructed	Original
4								
5	0	0	0	0	0000029d	669	6.69000000	0.00000000000
6	1	0	0	1	000003b4	948	9.48000000	9.99924701840
7	2	0	1	1	00000019	25	0.25000000	-0.24541228530
8	3	0	1	2	ffffbff	4294966271	-10.25000000	-9.99322384590
9	4	1	1	2	00000045	69	0.69000000	0.49067674350
10	5	1	1	3	000003e4	996	9.96000000	9.98118112900
11	6	1	2	3	ffffffc6	4294967238	-0.58000000	-0.73564563630
12	7	1	2	4	ffffbf8	4294966264	-10.32000000	-9.96312612180
13	8	1	2	4	00000077	119	1.19000000	0.98017140370
14	9	1	2	5	000003e6	998	9.98000000	9.93906970000
15	10	1	3	5	fffffff91	4294967185	-1.11000000	-1.22410675250
16	11	1	3	6	ffffc05	4294966277	-10.19000000	-9.90902635420
17	12	2	3	6	00000099	153	1.53000000	1.46730474520
18	13	2	3	7	000003c1	993	9.93000000	9.87301418150
19	14	2	4	7	fffff5b	4294967131	-1.65000000	-1.70961888830
20	15	2	4	8	ffffc26	4294966310	-9.86000000	-9.83105487420
21	16	2	4	8	000000ce	206	2.06000000	1.95090322100
22	17	2	4	9	000003db	987	9.87000000	9.78317370700
23	18	2	5	9	fffff1b	4294967067	-2.29000000	-2.19101240250
24	19	2	5	10	ffffc20	4294966304	-9.92000000	-9.72939952180
25	20	3	5	10	000000cc	236	2.36000000	2.42980180000
26	21	3	5	11	000003c7	967	9.67000000	9.66976471020
27	22	3	6	11	ffffcef1	4294967025	-2.71000000	-2.66712757580
28	23	3	6	12	ffffc36	4294966326	-9.70000000	-9.60430519380
29	24	3	6	12	0000012b	299	2.99000000	2.90284677370
30	25	3	6	13	000003bd	957	9.57000000	9.53306040320
31	26	3	7	13	ffffcc7	4294966983	-3.13000000	-3.13681740530
32	27	3	7	14	ffffc3d	4294966333	-9.63000000	-9.45607325340
33	28	4	7	14	00000151	337	3.37000000	3.36889853530
34	29	4	7	15	000003b9	953	9.53000000	9.37339011860
35	30	4	8	15	ffffe95	4294966933	-3.63000000	-3.59895036680
36	31	4	8	16	ffffc61	4294966369	-9.27000000	-9.28506080410
37	32	4	8	16	0000017e	382	3.82000000	3.82683432520
38	33	4	8	17	000003a5	933	9.33000000	9.19113851620
39	34	4	9	17	ffffc6d	4294966893	-4.03000000	-4.05241314160
40	35	4	9	18	ffffc7b	4294966395	-9.01000000	-9.09167983020
41	36	5	9	18	000001b6	438	4.38000000	4.27555093600
42	37	5	9	19	00000386	902	9.02000000	8.98674465610
43	38	5	10	19	ffffc3d	4294966845	-4.51000000	-4.49611329830
44	39	5	10	20	ffffc8c	4294966412	-8.84000000	-8.87639620310
45	40	5	10	20	000001cd	461	4.61000000	4.71396737010
46	41	5	10	21	00000378	888	8.88000000	8.76070094090
47	42	5	11	21	ffffc05	4294966789	-5.07000000	-4.92898192420
48	43	5	11	22	ffffc93	4294966419	-8.77000000	-8.63972856010
49	44	6	11	22	0000020f	527	5.27000000	5.14102744390
50	45	6	11	23	00000354	852	8.52000000	8.51355192980

Figure C-3 *Output Data Conversion Spreadsheet (Excel)*

C.2.1 Cell Formulae to Import and Convert Data (Lotus)

```
A9:   +A8+1

C9:   '0x0000031c

E9:   @MID($C9,5,1)
F9:   @MID($C9,6,1)
G9:   @MID($C9,7,1)
H9:   @MID($C9,8,1)
I9:   @MID($C9,9,1)

K9:   @IF(E9="f",65536,0)
L9:   @IF(F9="a",10,@IF(F9="b",11,@IF(F9="c",12,@IF(F9="d",13,
      @IF(F9="e",14,@IF(F9="f",15,@VALUE(F9)))))))
M9:   @IF(G9="a",10,@IF(G9="b",11,@IF(G9="c",12,@IF(G9="d",13,
      @IF(G9="e",14,@IF(G9="f",15,@VALUE(G9)))))))
N9:   @IF(H9="a",10,@IF(H9="b",11,@IF(H9="c",12,@IF(H9="d",13,
      @IF(H9="e",14,@IF(H9="f",15,@VALUE(H9)))))))
O9:   @IF(I9="a",10,@IF(I9="b",11,@IF(I9="c",12,@IF(I9="d",13,
      @IF(I9="e",14,@IF(I9="f",15,@VALUE(I9)))))))
P9:   (+L9*16^3+M9*16^2+N9*16^1+O9*16^0-K9)/$S$1
```

Cell A9 contains the time value, Cell B9 contains the output data in hexadecimal format, Cell C9 to I9 separates the characters, Cells K9 to O9 convert each hex character to a decimal integer and cell P9 calculates the floating point decimal number (N.B. divided by the factor in cell S1).

C.2.2 Cell Formulae to Import and Convert Data (Excel)

```
D5:   '0000029d
E5:   =HEX2DEC(+E5)
F5:   =IF(+F5>1000000000,+F5-2^32,+F5)/100
```

Cell D5 contains the hexadecimal data generated by the TMS320C30 simulator. Cell E5 converts the data to a signed decimal integer and cell F5 converts the data to floating point format. This is clearly a much simpler spreadsheet and is used for

importing output data generated by the TMS320C30 simulator, producing graphs and Fourier transforms throughout this study.

C.2.3 Macro for Importing Data (Lotus)

This macro imports the original signal held in ORIGIN.PRN, the output data generated by the TMS320C30 simulator held in RECON.OP1 and then automatically displays a graph comparing the original signal with the signal reconstructed by the TMS320C30 simulator.

```
Type Alt+R to
receive data
{GOTO}Q8~
/FIN
ORIGIN.PRN~
{GOTO}C8~
/FIT
RECON.OP1~
/GNU
{?}~
Q{HOME}
```

C.2.4 Macro for Importing Data (Excel)

This macro in Microsoft Excel V4.0 format performs the same function as the Lotus

	A
1	Macro3 (f)
2	=OPEN.TEXT("C:\DSF\PROGS\RECON\RECON.OPm",2,1,1,3,FALSE,FALSE,FALSE,FALSE,FALSE,TRUE,"x",{1,9;2,2})
3	=SELECT("R1C1:R257C1")
4	=COPY()
5	=ACTIVATE("RECON_2.XLS")
6	=FORMULA.GOTO("R5C5")
7	=PASTE.SPECIAL(3,1,FALSE,FALSE)
8	=OPEN.TEXT("C:\DSF\PROGS\RECON\ORIGIN.PRh",2,1,1,3,FALSE,FALSE,FALSE,FALSE,FALSE,FALSE,,{1,1})
9	=SELECT("R1C1:R257C1")
10	=COPY()
11	=ACTIVATE("RECON_2.XLS")
12	=FORMULA.GOTO("R5C8")
13	=PASTE.SPECIAL(3,1,FALSE,FALSE)
14	=FORMULA.GOTO("R1C1")
15	=FOURIER([RECON_2.XLS]Recon!$G$5:$G$260, [RECON_2.XLS]Recon!$I$5, FALSE, FALSE)
16	=FOURIER([RECON_2.XLS]Recon!$H$5:$H$260, [RECON_2.XLS]Recon!$P$5, FALSE, FALSE)
17	=RETURN()

macro but also automatically performs a fast Fourier transform on the original input data and the output data. The data must be copied into a separate Excel worksheet and then copied into the main worksheet which is more complicated than the Lotus macro. Data from different files can be imported by simply changing the file name in the macro (e.g. in cells A2: and A8:)

C.3 SUMMARY

This appendix shows how a complete environment for simulation can be achieved using entirely MS-DOS based applications and does not suffer the disadvantages of working in Microsoft Windows with the TMS320C30 simulator which is designed to run in MS-DOS. The major disadvantage is the rather limited graphics facility of Lotus 1-2-3 V 2.01 and the absence of the hex to decimal conversion. Microsoft Excel V 5.0 has both these facilities and a built-in fast Fourier transform function which is essential to this project. A 256 point FFT spreadsheet in Lotus 1-2-3 V 2.01 has been written by the author in a previous project but is not so convenient to use.

A compromise has been adopted in this project where the convenience of generating input data using the 'Lotus' spreadsheet is exploited but then using 'Excel' for importing output data, data analysis and display.

APPENDIX D

SIGNAL RECONSTRUCTION

SIGNAL RECONSTRUCTION

TABLE OF CONTENTS

APPENDIX D SIGNAL RECONSTRUCTION

D.1 RECONSTRUCTION - SPREADSHEET

Figure D-1 shows part of the spreadsheet which demonstrates signal reconstruction from zero crossing values. The product term is split into four elements as described in Chapter 3.1.2 equation (3-2). Many different graphs have been produced by this spreadsheet and others similar to it.

	A	B	C	D	E	F	G	H	I	J	K	L
1		Reconstruction of 8 Sine Waves Added Together from 32 Zero Crossings										
2												
3		For the Product = P (Sin(πt/256).Cos(πt(i)/256) - Cos(πt/256).Sin(πt(i)/256))										
4												
5												Carrier
6	Magnitude		10.00	3.33	2.00	1.43	1.11	0.91	0.77	0.67	Magnitude	20.00
7	Frequency		1.00	3.00	5.00	7.00	9.00	11.00	13.00	15.00	Frequency	16.00
8	Phase (degrees)		0	0	0	0	0	0	0	0	Phase	90
9	Dc Component		0.00		Noise Factor		0.00				Number of Samples	256
10												
11	Time	Input	Input + Added		Zero Values		Rounding Factor =		7			Reconstructed
12		Signal	Signal		$t_{(i)}$	Sin(πt/256)	Cos(πt$_{(i)}$/256)	Cos(πt/256)	Sin(πt$_{(i)}$/256)	Product		Signal
13												
14	0	0.00	20.00	1000.00	5.05	0.0000000	0.9980792	1.0000000	0.0619501	1.6615	1.2019	-0.03
15	1	1.95	20.42	1000.00	10.87	0.0122715	0.9911221	0.9999247	0.1329550	0.9861	2.0691	1.93
16	2	3.80	17.94	1000.00	21.06	0.0245412	0.9667886	0.9996988	0.2555775	0.5396	3.3220	3.78
17	3	5.46	13.11	1000.00	26.93	0.0368072	0.9458769	0.9993224	0.3245255	0.2580	5.0810	5.46
18	4	6.86	6.86	1000.00	37.05	0.0490677	0.8983921	0.9987955	0.4391943	0.0917	7.4922	6.87
19	5	7.95	0.29	1000.00								
20	6	8.71	-5.44	5.05								
21	7	9.14	-9.34	1000.00								
22	8	9.27	-10.73	1000.00								
23	9	9.16	-9.32	1000.00								
24	10	8.87	-5.28	1000.00								
25	11	8.47	0.81	10.87								
26	12	8.04	8.04	1000.00								
27	13	7.64	15.30	1000.00								
28	14	7.33	21.47	1000.00								
29	15	7.13	25.61	1000.00								
30	16	7.07	27.07	1000.00								
31	17	7.13	25.61	1000.00								
32	18	7.29	21.43	1000.00								
33	19	7.52	15.18	1000.00								
34	20	7.78	7.78	1000.00								
35	21	8.03	0.38	1000.00								
36	22	8.23	-5.91	21.06								
37	23	8.36	-10.12	1000.00								
38	24	8.41	-11.59	1000.00								
39	25	8.36	-10.11	1000.00								
40	26	8.25	-5.89	1000.00	210.99	0.3136817	-0.8512800	0.9495282	0.5247117	-0.0006	1017.9235	8.29
41	27	8.08	0.43	26.93	221.00	0.3253103	-0.9091865	0.9456073	0.4163892	0.0000	1166.1102	8.13
42	28	7.89	7.89	1000.00	226.99	0.3368899	-0.9373067	0.9415441	0.3485057	0.0006	1329.2238	7.94
43	29	7.70	15.36	1000.00	236.98	0.3484187	-0.9728931	0.9373390	0.2312552	0.0010	1507.8382	7.76
44	30	7.55	21.69	1000.00	243.00	0.3598950	-0.9873042	0.9329928	0.1588408	0.0013	1702.4366	7.60
45	31	7.45	25.93	1000.00	252.76	0.3713172	-0.9992083	0.9285061	0.0397832	0.0014	1913.3888	7.50
46	32	7.42	27.42	1000.00	1000.00	0.3826834		0.9238795		0.0013	2140.9267	7.47
47	33	7.45	25.93	1000.00	1000.00	0.3939920		0.9191139		0.0011	2385.1543	7.50

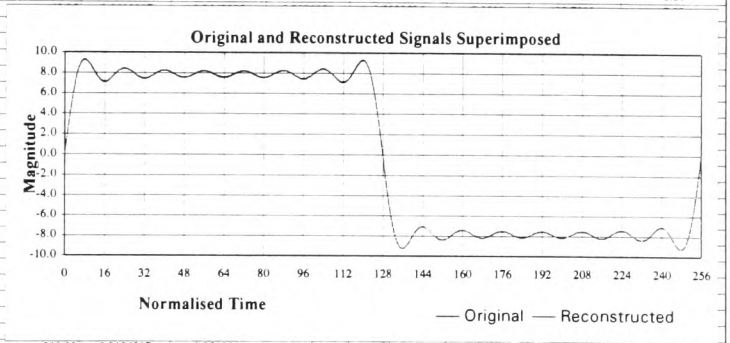


Figure D-1 *Spreadsheet Demonstrating Reconstruction from 32 Zero Crossings*

This spreadsheet has been used as a model for several 'C' programs which run in the both the TMS320C30 simulator and MS-DOS environments which are described in sections D.2 and D.3 of this APPENDIX.

D.1.1 Cell Formulae

The cell formulae for generating the signals and linear interpolation are almost the same as for the Lotus spreadsheets in APPENDIX C. The formula for multiplying 16 of the partial products together is as follows:

```
= ($F14*$G$30-$H14*$I$30)*($F14*$G$31-$H14*$I$31)*
  ($F14*$G$32-$H14*$I$32)*($F14*$G$33-$H14*$I$33)*
  ($F14*$G$34-$H14*$I$34)*($F14*$G$35-$H14*$I$35)*
  ($F14*$G$36-$H14*$I$36)*($F14*$G$37-$H14*$I$37)*
  ($F14*$G$38-$H14*$I$38)*($F14*$G$39-$H14*$I$39)*
  ($F14*$G$40-$H14*$I$40)*($F14*$G$41-$H14*$I$41)*
  ($F14*$G$42-$H14*$I$42)*($F14*$G$43-$H14*$I$43)*
  ($F14*$G$44-$H14*$I$44)*($F14*$G$45-$H14*$I$45)* 2^16
```

The last term is the square root of the scaling factor (i.e. 2^{16}) the next column in the spreadsheet contains the other 16 partial products and the rest of the scaling factor. The column containing the reconstruction multiplies the two partial products together and then subtracts the transforming signal thus yielding the reconstructed signal.

D.1.2 The Zero Crossing Sort and Fourier Transform Macro (Excel V-4)

The following is a macro in Excel Version 4.0 which sorts the data and invokes the built-in fast Fourier transform function.

	A
1	Zeros (t)
2	=CALCULATE.NOW()
3	=OPTIONS.CALCULATION(3,,,0.001,,FALSE)
4	=FORMULA.GOTO("R14C4")
5	=SELECT("R14C4:R271C4")
6	=COPY()
7	=SELECT("R14C5")
8	=PASTE.SPECIAL(3,1,FALSE,FALSE)
9	=SORT(1,"R14C5",1,,,,,0,1,FALSE)
10	=CALCULATE.NOW()
11	=FOURIER([ZERO_CR5.XLS]RECON_32!$L$14:$L$269, [ZERO_CR5.XLS]RECON_32!$N$14, FALSE, FALSE)
12	=FORMULA.GOTO("R1C1")
13	=OPTIONS.CALCULATION(3,,,0.001,,FALSE)
14	=CALCULATE.NOW()
15	=WORKBOOK.SELECT("Graph 5","Graph 5")
16	=RETURN()

D.2 RECONSTRUCTION - 'C' PROGRAMS FOR TMS320C30

D.2.1 Code for Signal Reconstruction Using Polynomial Calculation

```
/* RECON.ASP:                                                    8/10/95

Sequential reconstruction of a signal from zero crossing values using the polynomial
calculation method                                                          */

#include <math.h>
#define PI 3.141592654        /* The value of π                            */

main()
{
 volatile int    *INPUT_1 = (volatile int *) 0x808040;      /* Input  channel 1 */
 volatile int *OUTPUT_1 = (volatile int *) 0x808041;        /* Output channel 1 */

 register int i,j,n,M;
 static double CAS[385];
 double SINB;
 double COSB;
 float  z,t;
 static float reconstruct[256];

 t = PI/(float)*INPUT_1;      /* t = π / T  The minimum time increment        */
 n = *INPUT_1;                /* n = Number of zero crossings                 */
 M = *INPUT_1;                /* M = (magnitude of added frequency)x100       */

 for(i=0 ;i<=n; i++)          /* Initialises the output array                 */
 reconstruct[i]=0.5;

 z = PI/(float)n;
 for(i=0 ;i<=3*n/2; i++)      /* Creates the sample time CAS[385] table       */
 CAS[i]=2*sin(i*z);

 for(i=0 ;i<n; i++)
  {
   z = (float)*INPUT_1*t;
   SINB=sin(z);
   COSB=cos(z);
   for(j=0 ;j<=n; j++)
   reconstruct[j]*=CAS[j]*COSB-CAS[j+n/2]*SINB;
  }
 z=1.0;
 for(i=0;i<=n;i++,z=-z)              /* Outputs the reconstructed original signal */
 *OUTPUT_1 = M*(reconstruct[i]-z);
}
```

D.2.2 Code for Signal Reconstruction Using a Look-up Table

```
/* RECON.AS1:                                                    15/10/95

Sequential reconstruction of a signal from zero crossing values using  a three
quadrant  look-up table.                                                    */

#define     N       32    /*   Number of zero crossings                   */

/* Period (i.e. N*256)              T  =     8192                          */
/* Minimum size of CAS[S]           S  =      49                           */
/*                                  t_i' =  Zero crossing time             */
/*                                  t_0  =  Nyquist interval time          */

main()
{
 volatile int    *INPUT_1= (volatile int *)0x808040;     /* Input  channel  */
 volatile int *OUTPUT_1= (volatile int *)0x808041;       /* Output channel  */

 static float CAS[49]={/*   cas (π n t_0 / T) sample time table: t_0 / T=1/N */

 0.0000000000 , 0.0980171403 , 0.1950903220 , 0.2902846773 ,
 0.3826834324 , 0.4713967368 , 0.5555702330 , 0.6343932842 ,
 0.7071067812 , 0.7730104534 , 0.8314696123 , 0.8819212643 ,
 0.9238795325 , 0.9569403357 , 0.9807852804 , 0.9951847267 ,
 1.0000000000 , 0.9951847267 , 0.9807852804 , 0.9569403357 ,
 0.9238795325 , 0.8819212643 , 0.8314696123 , 0.7730104534 ,
 0.7071067812 , 0.6343932842 , 0.5555702330 , 0.4713967368 ,
 0.3826834324 , 0.2902846773 , 0.1950903220 , 0.0980171403 ,
 0.0000000000 ,-0.0980171403 ,-0.1950903220 ,-0.2902846773 ,
-0.3826834324 ,-0.4713967368 ,-0.5555702330 ,-0.6343932842 ,
-0.7071067812 ,-0.7730104534 ,-0.8314696123 ,-0.8819212643 ,
-0.9238795325 ,-0.9569403357 ,-0.9807852804 ,-0.9951847267 ,
-1.0000000000 };

 static float SIN[256] = {  /*          2*sin (π t_i'/ T)  table          */

 0.0000000000 , 0.0007669904 , 0.0015339806 , 0.0023009707 ,
 0.0030679604 , 0.0038349496 , 0.0046019383 , 0.0053689263 ,
                              ...
                              ...
 0.1899269907 , 0.1906905009 , 0.1914539830 , 0.1922174370 ,
 0.1929808627 , 0.1937442601 , 0.1945076289 , 0.1952709691 };

 static float COS[256] = {  /*          2*cos(π t_i'/ T)   table          */

 2.0000000000 , 1.9999998529 , 1.9999994117 , 1.9999986764 ,
 1.9999976469 , 1.9999963233 , 1.9999947055 , 1.9999927936 ,
```

```
                    ...
                    ...
1.9909615110 , 1.9908885285 , 1.9908152532 , 1.9907416851 ,
1.9906678243 , 1.9905936707 , 1.9905192243 , 1.9904444852 };

int  n,j,k,M;
float SINB;                    /* 2*sin (π[n t0 + ti'] /T ) = 2*sin (π ti /T )   */
float COSB;                    /* 2*cos (π[n t0 + ti'] /T ) = 2*cos (π ti /T )   */
static float reconstruct[N+1];

M=*INPUT_1;          /* M = (magnitude of transforming signal)x100              */
for(n=0; n<=N; n++)  /* Initialises the output array                            */
reconstruct[n]=0.5;
for(n=0; n<N; n++)   /* Reconstructs the original signal                        */
 {
 k=*INPUT_1;         /*   k = Zero crossing time ti'                            */
 SINB=SIN[k]*CAS[n+N/2]+COS[k]*CAS[n];
 COSB=COS[k]*CAS[n+N/2]-SIN[k]*CAS[n];
 for(j=0 ;j<=N ; j++)
  {
  k=j+N/2;
  reconstruct[j]*=CAS[j]*COSB-CAS[k]*SINB;
  }
 }
k=1;
for(n=0 ;n<=N ; n++,k=-k)          /* Outputs the reconstructed signal          */
*OUTPUT_1=M*(reconstruct[n]-(float)k);
}
```

D.2.3 Code for Demonstrating Oversampling

```
/* RECON.ASR:                                                          16/11/95

Sequential Reconstruction of a signal from zero crossings values using
the polynomial sine/cosine calculation method.

This program is designed to demonstrate oversampling.  For no oversampling
(S=1) 7 zero values are inserted between each sample value, for 2 times
(S=2) 3 are inserted, for 4 times (S=4) 1 is insert and for 8 times none
are inserted.                                                           */

#include <math.h>
#define PI 3.141592654      /* The value of π                              */
#define  S   1              /* The samples multiplier                      */

main()
```

```
{
volatile int   *INPUT_1 = (volatile int *) 0x808040;      /* Input  channel 1    */
volatile int *OUTPUT_1 = (volatile int *) 0x808041;       /* Output channel 1    */

register int i,j,n,M;
static double CAS[385];
double SINB;
double COSB;
float  z,t;
static float reconstruct[256];

t = PI/(float)*INPUT_1;       /* t = π / T  The minimum time increment          */
n = *INPUT_1;                 /* n = Number of zero crossings                   */
M = *INPUT_1;                 /* M = (magnitude of added frequency)x100         */

for(i=0 ;i<=S*n; i++)         /* Initialises the output array                   */
reconstruct[i]=0.5;

z = PI/(float)n;
for(i=0 ;i<=S*n*3/2; i++)     /* Creates the sample time CAS[385] table         */
CAS[i]=2*sin(i*z/S);

for(i=0 ;i<n; i++)
  {
  z = (float)*INPUT_1*t;
  SINB=sin(z);
  COSB=cos(z);
  for(j=0 ;j<=S*n; j++)
  reconstruct[j]*=CAS[j]*COSB-CAS[j+S*n/2]*SINB;
  }
for(i=0 ;i<=S*n ;i++)
  {
  *OUTPUT_1 = M*(reconstruct[i]-cos(PI*i/S));
  for(j=0; j<8/S-1 ;j++)  *OUTPUT_1 = 0;
  }
*OUTPUT_1 = M*(reconstruct[i]-cos(PI*i/S));
}
```

D.2.4 Code for Signal Reconstruction from Greater than 256 Zero Crossings

/* RECON.ASS: 5/12/95

Sequential reconstruction of a signal from zero crossing values using the polynomial calculation method.

```
The calculation is split into 2 to prevent overflow and convergence to zero when the
number of zero crossings is large (greater than 256).                          */

#include <math.h>
#define PI 3.141592654        /* The value of π                                */

main()
{
 volatile int    *INPUT_1 = (volatile int *) 0x808040;      /* Input  channel 1   */
 volatile int *OUTPUT_1 = (volatile int *) 0x808041;        /* Output channel 1   */

 register int i,j,k,n;
 static double CAS[385];
 double SINB;
 double COSB;
 float  z,t,M;
 static float recon_1[512],recon_2[512];

 t = PI/(float)*INPUT_1;       /* t = π/T  The minimum time increment          */
 n = *INPUT_1;                 /* n = Number of zero crossings                 */
 M = *INPUT_1;                 /* M = (magnitude of added frequency)x100       */
 for(i=0 ;i<=n; i++)           /* Initialises the output arrays                */
   {
   recon_1[i]=0.5;
   recon_2[i]=0.5;
   }
 z = PI/(float)n;
 for(i=0 ;i<=3*n/2; i++)       /* Creates the sample time CAS[] table          */
 CAS[i]=2*sin(i*z);

 for(i=0 ;i<n/2; i++)
   {
   z = (float)*INPUT_1*t;
   SINB=sin(z);
   COSB=cos(z);
   for(j=0 ;j<=n; j++)
   recon_1[j]*=CAS[j]*COSB-CAS[j+n/2]*SINB;

   z = (float)*INPUT_1*t;
   SINB=sin(z);
   COSB=cos(z);
   for(j=0 ;j<=n; j++)
   recon_2[j]*=CAS[j]*COSB-CAS[j+n/2]*SINB;
   }
 z=1.0;
 for(i=0;i<=n;i++,z=-z)              /* Outputs the reconstructed original signal */
 *OUTPUT_1 = M*(2.0*recon_1[i]*recon_2[i]-z);
}
```

```
reconstruct[j]*=CAS[j]*COSB-CAS[j+S*n/2]*SINB;
  }
 for(i=0 ;i<=S*n ;i++)
 *OUTPUT_1 = M*(reconstruct[i]-cos(PI*i/S));
}
```

D.3 RECONSTRUCTION - 'C' PROGRAMS FOR MS-DOS

The following code has been written to run in MS-DOS directly rather than the simulator. It has the facility to split the calculation up into 4 stages but could be adapted to any number by minor changes to function "reconstruct" on page D-10.

```
/*********************************************************************
Author          J A Sherrington.
Installation    Gwent College of Higher Education.
Date Written            22 November 1995.
Source Code             RECON.C

*********************************************************************

This program reads a text file containing the zero crossing values of a signal and the
magnitude and frequency of a "Transforming Signal". The original waveform is then
reconstructed and written to "OUTPUT.PRN".

************************ Header Files ******************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/************************ Definitions *******************************/

#define T  65536              /*  The period of the fundamental                */
#define PI 3.1415926535       /*  The value of π                               */
#define Z  1024               /*  Maximum number of zero crossings expected    */

/*********************     The Main Program     ***********************/

main()
{
 int  read_input      (char *in,float*p);
 void reconstruct     (int b,float *r,float *s);
 void write_output_1  (int c,char *out,float *p);
```

```
int num_zero_crossings;
float  *x;
float *y;

x=(float *)malloc((Z+2)*sizeof(float));
y=(float *)malloc((Z+2)*sizeof(float));

num_zero_crossings=read_input("input.prn",x);
reconstruct(num_zero_crossings,x,y);
write_output_1(num_zero_crossings,"output.prn",y);

free(x);free(y);
return 0;
}

/********************** End of Main Program    ************************/



/********************** Function "read_input" ***************************

The input file is opened and the text is converted to floating point numbers. The
numbers are then processed and the results are stored in array x[j].

*************************************************************************/

int read_input(char *in,float *p)
{
 register int i,j;
 FILE *fp;
 char  ch,*input;

 input=(char *)malloc(20*sizeof(char));
 if((fp=fopen(in,"rt"))==NULL)
  {
  puts("\nCannot open source file\n");
  exit(1);
  }
 i=0; j=0;
 while(!feof(fp))
  {
  ch=getc(fp);
  if(ch!='\n')  input[i++]=ch;
  else
    {
    p[j]=atof(input);            /* Converts characters to floating point numbers */
    for(i=0; i<20; i++)
```

```
    input[i]='\0';                    /* Clears character buffer input[i]        */
    if(p[j]) j++;                     /* If p[j] is non-zero increment j         */
    i=0;
   }
 }
free(input);
fclose(fp);
return (j-1);                         /* Returns the number of zero crossings    */
}
```

/********** Function "reconstruct" **********

The original waveform is reconstructed using the zero crossing values, the amplitude of the maximum frequency and the maximum frequency. The results stored in y[j].

**********/

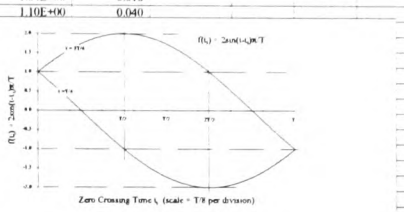
```
void  reconstruct(int n,float *INPUT,float *OUTPUT)
{
 register int i,j,k;

 float COSA,SINA,*SINB,*COSB,h;

 SINB=(float *)malloc(n*sizeof(float));
 COSB=(float *)malloc(n*sizeof(float));

 for(i=0; i<=n; i++)                  /* n = Number of zero crossings            */
  {
   SINB[i]=2*sin(PI*INPUT[i+1]/T);
   COSB[i]=2*cos(PI*INPUT[i+1]/T);
  }
 h=1.0;
 for(j=0; j<=n; j++,h=-h)             /* h = cos(2*π*f_c*t/S) = cos(PI*j) = -1 or +1 */
  {                                   /* s = number of samples = n               */
   SINA=sin(PI*j/n);                  /* n = number of zero crossings            */
   COSA=cos(PI*j/n);
   OUTPUT[j]=0.5;
   for(k=0; k<4; k++)          /* 4 = number of multiplication stages     */
         {
          for(i=k; i<n; i+=4)
          OUTPUT[j]*=(SINA*COSB[i]-COSA*SINB[i]);
         }
   OUTPUT[j]=INPUT[0]*(OUTPUT[j]-h);       /* Subtracts transforming signal */
  }
 free(SINB);free(COSB);
}
```

```
/******************** Function "write_output_1" ************************
The array y[i] is written to the output file.
*****************************************************************/

void write_output_1(int n,char *out,float *p)
{
 register int i;
 FILE *fp;
 if((fp=fopen(out,"wt"))==NULL)
  {
   puts("\nCannot open destination file\n");
   exit(1);
  }
 for(i=0; i<=n; i++)
  {
   fprintf(fp,"%12.6lf\n",p[i]);
   p[i]=0;                              /* clears buffer p[i] */
  }
 fclose(fp);
}

/******************** End of Complete Program ************************/
```

D.4 Product Evaluation Spreadsheet

Part of the Excel V 5.0 spreadsheet for evaluating the product expression as discussed in Chapter 3.1.7.3 is shown in Figure D-2. The zero crossing values were obtained from a signal with only the transforming signal present. The partial product is achieved by the formula ' =2*SIN((E$10-$B21)*$C$10)', the accumulated product is achieved by cell formula ' =ABS(C21*D20)' and LOG_{10} , by ' =LOG(D21)'.

	A	B	C	D	E	F	G	H	I	J	K	L
1												
2		**Product Evaluation for** $f(t_i) = \prod 2\sin(t-ti)_{\pi}/T$										
3												
4												
5			LOG	ACTUAL								
6		MAX =	35.919	8.2896E+35								
7		MAX =	0.000	1.0000E+00								
8		MIN =	-35.919	1.2063E-36								
9												
10		π/T =	4.79369E-05			10923 = Value of t for minimum (i.e T/6)						
11						54613 = Value of t for maximum (i.e. 5T/6)						
12		65536	= Period T									
13		256	= Number of zero crossings									
14												
15				for t = T/6			for t = 5T/6					
16												
17	Sample	Zero Crossing	Partial Product	Accumulated	Accumulated	Partial Product	Accumulated	Accumulated				
18	Number	Time (t_i)	$2\sin(t-ti)\pi/T$	Product (Abs)	Product (Log_{10})	$2\sin(t-ti)\pi/T$	Product (Abs)	Product (Log_{10})				
19												
20	0	0	1.0000	1.00E+00	0.000	1.0000	1.00E+00	0.000				
21	1	128	0.9894	9.89E-01	-0.005	1.0106	1.01E+00	0.005				
22	2	384	0.9679	9.58E-01	-0.019	1.0317	1.04E+00	0.018				
23	3	640	0.9464	9.06E-01	-0.043	1.0527	1.10E+00	0.040				
24	4	896	0.9247	8.38E-01	-0.077	1.0734						
25	5	1152	0.9029	7.57E-01	-0.121	1.0941						
26	6	1408	0.8809	6.67E-01	-0.176	1.1145						
27	7	1664	0.8588	5.72E-01	-0.242	1.1348						
28	8	1920	0.8366	4.79E-01	-0.320	1.1550						
29	9	2176	0.8142	3.90E-01	-0.409	1.1749						
30	10	2432	0.7917	3.09E-01	-0.510	1.1947						
31	11	2688	0.7691	2.37E-01	-0.624	1.2143						
32	12	2944	0.7464	1.77E-01	-0.751	1.2337						
33	13	3200	0.7236	1.28E-01	-0.892	1.2529						
34	14	3456	0.7007	8.99E-02	-1.046	1.2719						
35	15	3712	0.6776	6.09E-02	-1.215	1.2908						
36	16	3968	0.6545	3.99E-02	-1.400	1.3094						
37	17	4224	0.6312	2.52E-02	-1.599	1.3279						
38	18	4480	0.6079	1.53E-02	-1.815	1.3461						
39	19	4736	0.5845	8.94E-03	-2.049	1.3642						
40	20	4992	0.5610	5.01E-03	-2.300	1.3820						
41	21	5248	0.5374	2.69E-03	-2.570	1.3997	5.03E+01	1.702				
42	22	5504	0.5137	1.38E-03	-2.859	1.4171	7.13E+01	1.853				
43	23	5760	0.4899	6.78E-04	-3.169	1.4343						
44	24	6016	0.4661	3.16E-04	-3.500	1.4513						
45	25	6272	0.4422	1.40E-04	-3.855	1.4681						
46	26	6528	0.4182	5.85E-05	-4.233	1.4846						
47	27	6784	0.3942	2.30E-05	-4.637	1.5010						
48	28	7040	0.3701	8.53E-06	-5.069	1.5171						
49	29	7296	0.3460	2.95E-06	-5.530	1.5330						
50	30	7552	0.3218	9.49E-07	-6.023	1.5486						
51	31	7808	0.2975	2.82E-07	-6.549	1.5640						
52	32	8064	0.2732	7.72E-08	-7.113	1.5792						
53	33	8320	0.2489	1.92E-08	-7.717	1.5941						
54	34	8576	0.2245	4.31E-09	-8.365	1.6088						
55	35	8832	0.2001	8.63E-10	-9.064	1.6233						
56	36	9088	0.1757	1.52E-10	-9.819	1.6375						
57	37	9344	0.1512	2.29E-11	-10.640	1.6515						
58	38	9600	0.1267	2.90E-12	-11.537	1.6652						
59	39	9856	0.1022	2.97E-13	-12.527	1.6787						
60	40	10112	0.0777	2.31E-14	-13.637	1.6919						
61	41	10368	0.0532	1.23E-15	-14.911	1.7049						
62	42	10624	0.0286	3.51E-17	-16.454	1.7176	6.73E+05	5.828				
63	43	10880	0.0041	1.44E-19	-18.843	1.7300	1.17E+06	6.066				



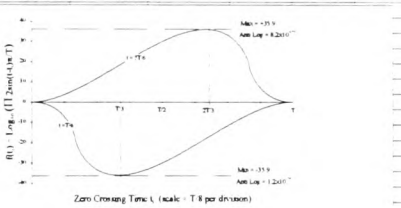


Figure D-2 *The Product Evaluation Spreadsheet*

APPENDIX E

NEWTON'S FORMULA

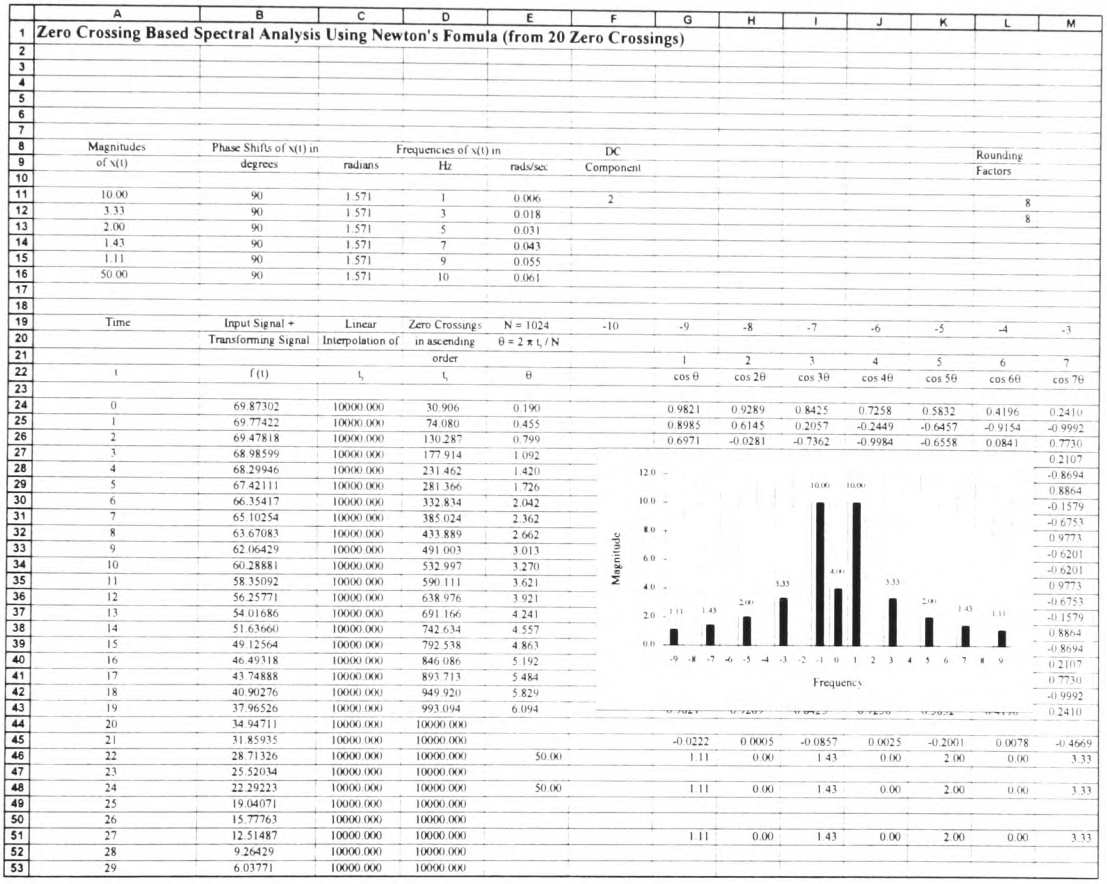
NEWTON'S FORMULA

TABLE OF CONTENTS

APPENDIX E NEWTON'S FORMULA

E.1 NEWTON'S FORMULA - SPREADSHEET

Figure E-1 shows part of the Microsoft Excel V5.0 spreadsheet which calculates the spectrum of a transformed input signal from its zero crossing values using Newton's formula as described in Chapter 4.1.3.



	A	B	C	D	E	F	G	H	I	J	K	L	M
1	**Zero Crossing Based Spectral Analysis Using Newton's Fomula (from 20 Zero Crossings)**												
8	Magnitudes	Phase Shifts of x(t) in	Frequencies of x(t) in			DC							
9	of x(t)	degrees	radians	Hz	rads/sec	Component						Rounding Factors	
11	10.00	90	1.571	1	0.006	2							8
12	3.33	90	1.571	3	0.018								8
13	2.00	90	1.571	5	0.031								
14	1.43	90	1.571	7	0.043								
15	1.11	90	1.571	9	0.055								
16	50.00	90	1.571	10	0.061								
19	Time	Input Signal +	Linear	Zero Crossings	N = 1024	-10	-9	-8	-7	-6	-5	-4	-3
20		Transforming Signal	Interpolation of	in ascending order	$\theta = 2\pi t_i / N$		1	2	3	4	5	6	7
22	t	f(t)	t_i	t_i	θ		cos θ	cos 2θ	cos 3θ	cos 4θ	cos 5θ	cos 6θ	cos 7θ
24	0	69.87302	10000.000	30.906	0.190		0.9821	0.9289	0.8425	0.7258	0.5832	0.4196	0.2410
25	1	69.77422	10000.000	74.080	0.455		0.8985	0.6145	0.2057	-0.2449	-0.6457	-0.9154	-0.9992
26	2	69.47818	10000.000	130.287	0.799		0.6971	-0.0281	-0.7362	-0.9984	-0.6558	0.0841	0.7730
27	3	68.98599	10000.000	177.914	1.092								0.2107
28	4	68.29946	10000.000	231.462	1.420								-0.8694
29	5	67.42111	10000.000	281.366	1.726								0.8864
30	6	66.35417	10000.000	332.834	2.042								-0.1579
31	7	65.10254	10000.000	385.024	2.362								-0.6753
32	8	63.67083	10000.000	433.889	2.662								0.9773
33	9	62.06429	10000.000	491.003	3.013								-0.6201
34	10	60.28881	10000.000	532.997	3.270								-0.6201
35	11	58.35092	10000.000	590.111	3.621								0.9773
36	12	56.25771	10000.000	638.976	3.921								-0.6753
37	13	54.01686	10000.000	691.166	4.241								-0.1579
38	14	51.63660	10000.000	742.634	4.557								0.8864
39	15	49.12564	10000.000	792.538	4.863								-0.8694
40	16	46.49318	10000.000	846.086	5.192								0.2107
41	17	43.74888	10000.000	893.713	5.484								0.7730
42	18	40.90276	10000.000	949.920	5.829								-0.9992
43	19	37.96526	10000.000	993.094	6.094								0.2410
44	20	34.94711	10000.000	10000.000									
45	21	31.85935	10000.000	10000.000			-0.0222	0.0005	-0.0857	0.0025	-0.2001	0.0078	-0.4669
46	22	28.71326	10000.000	10000.000	50.00		1.11	0.00	1.43	0.00	2.00	0.00	3.33
47	23	25.52034	10000.000	10000.000									
48	24	22.29223	10000.000	10000.000	50.00		1.11	0.00	1.43	0.00	2.00	0.00	3.33
49	25	19.04071	10000.000	10000.000									
50	26	15.77763	10000.000	10000.000									
51	27	12.51487	10000.000	10000.000			1.11	0.00	1.43	0.00	2.00	0.00	3.33
52	28	9.26429	10000.000	10000.000									
53	29	6.03771	10000.000	10000.000									

Figure E-1 *Spreadsheet Demonstrating Spectra by Newton's Formula*

The spreadsheet consists of two main sections. The first section, contained in columns A to E, generates the input signal, adds the transforming signal, calculates the zero crossing values by linear interpolation and finally sorts the zero crossing values in ascending order. The second section, contained in columns F to Z,

calculates the two stages of Newton's formula and creates a graphical output displaying the resulting spectrum.

E.1.1 Signal Generation and Zero Crossing Calculation

Signal generation and the addition of the transforming signal is contained in the cells of column B with the values of time contained in column A. Interpolation of the zero crossing values is contained in column C. The cell formulae are as follows:

Cell A25: 1
Cell B25: = $A$11*SIN($E$11*$A25+$C$11)+$A$12*SIN($E$12*$A25+$C$12)
+$A$13*SIN($E$13*$A25+$C$13)+$A$14*SIN($E$14*$A25+$C$14)
+$A$15*SIN($E$15*$A25+$C$15)+$A$16*SIN($E$16*$A25+$C$16)
+$F$11
Cell C25: = IF(B25*B26<0,A25+B25/(B25-B26),10000)

Sorting the zero crossing values in ascending order is performed by the following macro:

	A
1	Macro1 (s)
2	=CALCULATE.NOW()
3	=SELECT("R24C3:R1048C3")
4	=COPY()
5	=SELECT("R24C4")
6	=PASTE.SPECIAL(3,1,FALSE,FALSE)
7	=SORT(1,"R24C4",1,,,,,0,1,FALSE)
8	=CALCULATE.NOW()
9	=SELECT("R1C1")
10	=RETURN()

The column containing the unsorted zero crossing formulae is first copied into a memory buffer and the actual values, in floating point number format, are copied into the next column. The numbers in this column are then sorted in ascending order using the Excel sort facility.

E.1. 2 Spectrum Calculation Using Newton's Formula

Some of the cell formulae from the spreadsheet are as follows:

Cell G43: = ROUND(COS(G$21*$E43),$L$12)
Cell G45: = SUM(G24:G43)
Cell G46: = -1/G$21*($G$45*$A$16)
Cell G48: = ROUND(ABS(G46),$L$12)

Cell H43: = ROUND(COS(H$21*$E43),$L$12)
Cell H45: = SUM(H24:H43)
Cell H46: = -1/H$21*($G$45*G46+$H$45*$A$16)
Cell H48: = ROUND(ABS(H46),$L$12)

Cell Y43: = ROUND(COS(Y$21*$E43),$L$12)
Cell Y45: = SUM(Y24:Y43)
Cell Y46: = -1/Y$21*($G$45*X46+$H$45*W46+$I$45*V46+$J$45*U46+
$K$45*T46+$L$45*S46+$M$45*R46+$N$45*Q46+
$O$45*P46+$P$45*O46+$Q$45*N46+$R$45*M46+
$S$45*L46+$T$45*K46+$U$45*J46+$V$45*I46+
$W$45*H46+$X$45*G46+$Y$45*$A$16)
Cell Y48: = ROUND(ABS(Y46),$L$12)

The real parts of $Z_i^k = \cos(k 2\pi t_i / N)$ (e.g. cell G43) are contained in the cell range G24 to Z43. The values of 't_i' increase down the column and the index 'k' increases along the row. The sums $\sum_{i=1}^{L} Z_i^k$ are contained in the cells at the bottom of the columns (e.g. cell G45 contains $\sum_{i=1}^{l} Z_i^l$). The spectral lines $a_{(k)} = -\frac{1}{k}\sum_{i=1}^{k} p_{(i)}\, a_{(k-i)}$ are then calculated in the cells of row 46 (e.g. cells G46, H46 and Y46). The absolute values of the spectral lines are then calculated in the cells of row 48.

APPENDIX F

SEQUENTIAL SPECTRUM GENERATION

SEQUENTIAL SPECTRUM GENERATION

TABLE OF CONTENTS

APPENDIX F SEQUENTIAL SPECTRUM GENERATION

F.1 SEQUENTIAL SPECTRUM - SPREADSHEET

The spreadsheet to calculate the spectrum from 10 zero crossing time values using the sequential algorithm described in Chapter 4.5 , equation (4-3) and Table 4-3 is shown in Figures F-1 and F-2. The full spreadsheet is split into two worksheets within the same workbook, the first sheet (Figure F-1) calculates the zero crossing values in the same way as already described in APPENDIX C and APPENDIX D and the second sheet (Figure F-2) calculates the spectrum.

	A	B	C	D	E	F	G	H	I	J
1	**Reconstruction of 4 Sine Waves Added Together from 10 Zero Crossings**									
2										
3	Magnitude	10.00	6.00	4.00	2.00		Carrier Magnitude	=	30.00	
4	Frequency	1.00	2.00	3.00	4.00		Carrier Frequency	=	5.00	
5	Phase (degrees)	0	0	0	0		Carrier Phase	=	90	
6	Dc Component	0.00		π/180 =	0.0174533		Number of Samples =		256	
7										
8					Zero Crossings					
9				Linear	in Ascending					
10	Time	Input Signal	Input +Carrier	Interpolation of	Order					
11	t	f(t)	f(t) + c(t)	$t_{(i)}$	$t_{(i)}$					
12										
13	0	0.00	30.00	1000.00	16.82					
14	1	1.03	30.80	1000.00	34.02					
15	2	2.06	31.16	1000.00	65.61					
16	3	3.07	31.06	1000.00	88.25					
17	4	4.08	30.53	1000.00	115.44					
18	5	5.06	29.59	1000.00	141.05					
19	6	6.03	28.26	1000.00	165.13					
20	7	6.97	26.57	1000.00	193.70					
21	8	7.88	24.55	1000.00	214.11					
22	9	8.76	22.25	1000.00	245.88					
23	10	9.61	19.71	1000.00	1000.00					
24	11	10.41	16.99	1000.00	1000.00					
25	12	11.18	14.12	1000.00	1000.00					
26	13	11.90	11.16	1000.00						
27	14	12.57	8.17	1000.00						
28	15	13.19	5.19	1000.00						
29	16	13.76	2.28	16.82						
30	17	14.29	-0.50	1000.00						
31	18	14.76	-3.12	1000.00						
32	19	15.17	-5.52	1000.00						
33	20	15.53	-7.66	1000.00						
34	21	15.84	-9.51	1000.00						
35	22	16.09	-11.03	1000.00						
36	23	16.29	-12.20	1000.00						
37	24	16.44	-12.99	1000.00						
38	25	16.53	-13.39	1000.00						
39	26	16.58	-13.39	1000.00						
40	27	16.57	-12.99	1000.00						
41	28	16.52	-12.19	1000.00						
42	29	16.43	-11.00	1000.00						
43	30	16.29	-9.44	1000.00						
44	31	16.11	-7.54	1000.00	1000.00					
45	32	15.90	-5.31	1000.00	1000.00					
46	33	15.65	-2.81	1000.00	1000.00					

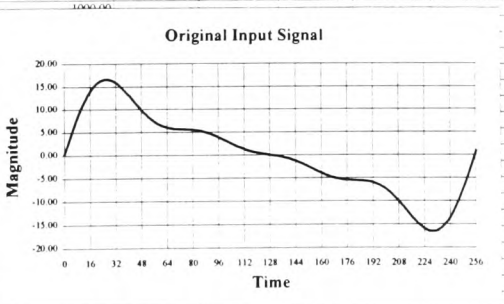
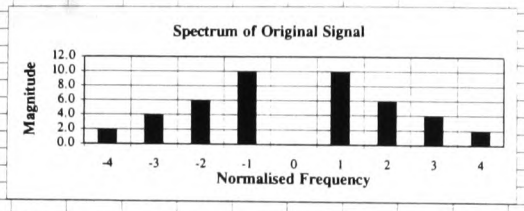


Figure F-1 *Worksheet for Calculating the Zero Crossing Times $t_{(i)}$*

Spectrum for 4 Sinewaves Added Together from 10 Zero Crossings

Fourier Coefficient	$a_{(0)}$	$a_{(1)}$	$a_{(2)}$	$a_{(3)}$	$a_{(4)}$	$a_{(5)}$	$a_{(6)}$	$a_{(7)}$	$a_{(8)}$	$a_{(9)}$	$a_{(10)}$
Frequency	-5	-4	-3	-2	-1	0	1	2	3	4	5
Re parts of $a_{(i)}$	30.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	30.00
Im parts of $a_{(i)}$	0.00	2.00	-4.00	6.00	-10.01	0.00	10.01	-6.00	4.00	-2.00	0.01
Magnitude of $a_{(i)}$	30.00	2.00	4.00	6.00	10.01	0.00	10.01	6.00	4.00	2.00	30.00



NB. Real part $Z_{(i)}$ = Cos(2 π $t_{(i)}$ / 256) and Imag part Z(i) = Sin(2 πt(i) / 256)

Zero Crossing Values $t_{(i)}$	i	Real $Z_{(i)}$									
16.82	1	0.92									
34.02	2	0.67	0.32								
65.61	3	-0.04	-1.20	-0.96							
88.25	4	-0.56	-2.64	-1.56	0.31						
115.44	5	-0.95	-1.84	2.58	2.97	0.00					
141.05	6	-0.95	1.00	5.11	-1.75	-3.09	0.32				
165.13	7	-0.61	2.90	0.25	-6.10	3.11	1.84	-0.94			
193.70	8	0.04	2.10	-4.22	1.85	3.12	-4.41	1.63	0.29		
214.11	9	0.52	0.23	-1.86	3.12	-3.14	1.87	-0.02	-1.15	0.97	
245.88	10	0.97	-0.86	0.67	-0.39	0.00	0.39	-0.67	0.86	-0.97	1.00

	Imag $Z_{(i)}$									
1	0.40									
2	0.74	0.95								
3	1.00	1.54	0.28							
4	0.83	0.08	-2.13	-0.95						
5	0.30	-2.53	-3.52	1.00	1.00					
6	-0.31	-3.12	1.65	5.08	-1.07	-0.95				
7	-0.79	-1.09	5.34	-1.81	-4.35	2.49	0.33			
8	-1.00	1.62	1.30	-5.34	4.71	-0.20	-2.09	0.96		
9	-0.86	1.88	-1.86	0.49	1.29	-2.54	2.63	-1.50	0.25	
10	-0.25	0.54	-0.87	1.20	-1.58	1.53	-1.07	0.67	-0.31	0.00

Figure F-2 *Worksheet for Calculating the Spectrum from 10 Zero Crossings*

F.1.1 Cell Formulae

The essential formulae for calculating the spectrum are as follows:

Cell A25: =SPEC_10S!$E14
Cell C25: =COS(A25*2*PI()/256)
Cell D25: =$C25*SUM(C$24:C24)-$C39*SUM(C$38:C38)

Cell C39: =SIN(A25*2*PI()/256)
Cell D39: =$C25*SUM(C$38:C38)+$C39*SUM(C$24:C24)

C5: =SUM(C24:C33)*$B$5
C6: =SUM(C38:C69)*$B$5

D7: =SQRT(C5^2+C6^2)

The worksheet containing the spectrum calculation is linked to the worksheet containing the zero crossing times calculation hence the formula of cell A25 which fetches the zero crossing time from the first work sheet named SPEC_10S.

Cell C25 contains the real part of $Z_{(i)}$ and cell C39 the imaginary part. Cells D25 and D39 contain the first terms of the real and imaginary parts of the second spectral line in accordance with Table 4-3 of Chapter 4.5. These two cells are then copied down the columns and along the rows in a triangular pattern as shown in Figure F-2.

Cell C5 sums all the real parts of the second spectral line and cell C6 the imaginary parts. Cell C7 calculates the modulus of the complex number implied by cells C5 and C6 yielding the magnitude of the frequency component. These three cells are copied along the rows to give the complete spectrum as shown in the inset of Figure F-2.

F.1.2 Macro for Building Spectral Lines

It is a simple task to copy the spectral line cells as indicated in Figure F-2 for only 10 zero crossing values but for 64, the task would be very tedious and prone to errors. A better method is to use a macro so that most of the task is automatic.

The first column of the real and imaginary parts are first created by copying the cells down the column. The cursor is then placed on the second cell down the column (e.g. cell D26 of Figure F-2) and the triangular pattern of cells is achieved by running the following macro as many times as is required to complete the triangle.

	A
1	Spectral line (a)
2	=SELECT("RC:R[7]C")
3	=COPY()
4	=SELECT("RC[1]")
5	=PASTE()
6	=SELECT("R[1]C")
7	=RETURN()
8	

A similar macro is used to create the spreadsheet for calculating spectra from up to 64 zero crossings as shown in Figure F-3 where 56 zero crossings have been calculated.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Zero Crossing Based Spectrum (up to 64 Zero Crossings)														
2															
3	Spectral Line a(i)	0	1	2	3	4	5	6	7	8	9	10	11	12	13
4	Frequency	-32	-31	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19
5	Frequency	-30	-29	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17
6	Frequency	-28	-27	-26	-25	-24	-23	-22	-21	-20	-19	-18	-17	-16	-15
7	Re parts of $a_{t(i)}$	20.00	0.00	-0.06	0.00	0.02	0.00	0.02	0.00	0.09	0.00	0.03	0.00	-0.01	0.00
8	Im parts of $a_{t(i)}$	0.00	-0.02	0.00	0.03	0.00	0.00	0.00	0.01	0.00	-0.02	0.00	0.04	0.00	0.01
9	Magnitude of $a_{t(i)}$	20.00	0.02	0.06	0.03	0.02	0.00	0.02	0.01	0.09	0.02	0.03	0.04	0.01	0.01
10															
11															
12	Zero Crossng		Real		NB Real $Z_{t(i)} = \cos(2\pi t_{t(i)}/16384)$				NB Imag $Z_{t(i)} = \sin(2\pi t_{t(i)}/16384)$					Resolution =	
13	Values $t_{t(i)}$	i	$Z_{t(i)}$												
14	0		-1												
15	80	1	-0.9981181												
16	203	2	-0.9879034	-0.98											
17	388	3	-0.9560453	-1.84	-0.87										
18	490	4	-0.9302050	-2.55	-2.25	-0.63									
19	677	5	-0.8681904	-2.93	-3.39	-1.49	-0.16								
20	788	6	-0.8228498	-3.12	-3.84	-1.18	0.79	0.43							
21	969	7	-0.7362978	-2.78	-2.27	3.46	7.18	4.43	0.93						
22	1078	8	-0.6772222												
23	1264	9	-0.5657318												
24	1370	10	-0.4968970												
25	1554	11	-0.3698924												
26	1665	12	-0.2895506												
27	1848	13	-0.1527972												
28	1955	14	-0.0712696												
29	2141	15	0.0712696												
30	2248	16	0.1527972												
31	2431	17	0.2895506												
32	2542	18	0.3698924												
33	2726	19	0.4968970												
34	2832	20	0.5657318												
35	3018	21	0.6772222												
36	3127	22	0.7362978												
37	3308	23	0.8228498												
38	3419	24	0.8681904												
39	3606	25	0.9302050												
40	3708	26	0.9560453												
41	3893	27	0.9879034												
42	4016	28	0.9981181												
43	4163	29	0.9986799	-0.92	-158.85	48.47	4175.38	-756.33	-42784.03	5427.76	224712.84	-21527.96	-688567.40	51821.70	1317945.24
44	4334	30	0.9833851	-4.21	-152.28	326.02	3892.69	-6739.95	-38715.62	60855.96	196776.85	-294963.11	-580870.18	853840.91	1063898.95
45	4439	31	0.9655942	-6.49	-139.50	531.67	3198.64	-11127.67	-26790.72	99855.68	101381.22	-475620.39	-157689.49	1340342.07	-78675.94
46	4629	32	0.9175962	-9.60	-114.78	737.59	1966.89	-14342.23	-6913.42	117227.96	-48096.62	-491706.31	461389.38	1147279.41	-1619237.32
47	4736	33	0.8819213	-11.40	-90.29	833.02	738.43	-14675.99	11689.95	101586.72	-171705.43	-310703.11	874618.93	278954.29	-2297003.62
48	4918	34	0.8077528	-13.55	-55.29	875.01	-709.95	-12685.46	29025.07	58354.33	-247533.69	6667.27	918262.56	-864030.53	-1585798.55

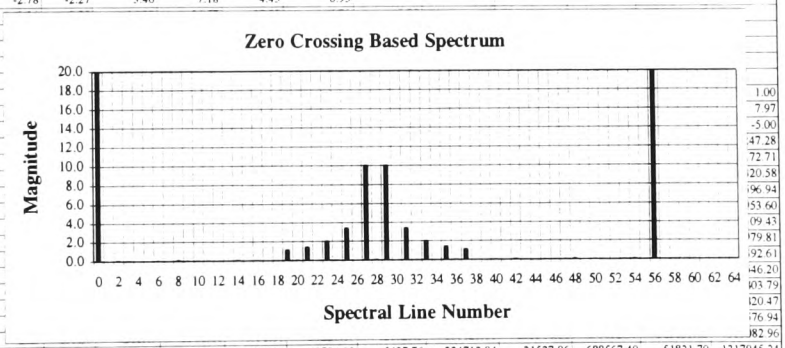


Figure F-3 *Worksheet for Calculating the Spectrum from 56 Zero Crossings*

F.2 SEQUENTIAL SPECTRUM - 'C' PROGRAMS FOR TMS320C30

The spreadsheets described in section F.1 form useful models for the 'C' codes contained in the following subsections. The input data is imported from the files

generated by the Lotus 1-2-3 spreadsheet described in APPENDIX C section C1. The output data is imported from files generated by the TMS320C30 into an Excel spreadsheet similar to that described in APPENDIX C Figure C-3 for graphical display.

F.2.1 Code for Double Sided Spectra Using Polynomials

```
/* SPECTRUM.DP:                                                    1/10/95

Sequential spectral analysis of a signal from zero crossing values producing the
double sided spectrum using polynomial sine/cosine calculation                  */

#include <math.h>
#define MAX 64                  /* The maximum number of zero crossings possible */
#define t 7.66990393e-4         /* t = 2π / T  the minimum time increment where  */
                                /* T = 8192 = The normalised time period         */
main()
{
 volatile int    *INPUT_1 = (volatile int *) 0x808040;     /* Input  channel 1   */
 volatile int *OUTPUT_1 = (volatile int *) 0x808041;       /* Output channel 1   */

 register int i,j,k=0;          /* Counter registers                          */
 int n;                         /* The number of zero crossings               */
 int A;                         /* Magnitude of added signal                  */
 double Rp;                     /* Real part input data buffer                */
 double Ip;                     /* Imaginary part input data buffer           */
 double a[MAX],b[MAX];          /* Calculation buffers                        */

 n = *INPUT_1;
 A = *INPUT_1;
 for(i=0; i<n; i++,k++)
  {
  Ip = *INPUT_1*t;
  Rp =  cos(Ip);
  Ip =  sin(Ip);
  a[k] = 0;
  b[k] = 0;
  for(j=k; j>=0; j--)
   {
   a[j+1] += (Rp*a[j]-Ip*b[j]);
   b[j+1] += (Rp*b[j]+Ip*a[j]);
   }
  a[0] += Rp;
```

```
  b[0] += Ip;
  }
for(i=0 ;i<n-1 ;i++)          /* Magnitude of the spectrum written to the output */
*OUTPUT_1 = A*sqrt(a[i]*a[i]+b[i]*b[i]);
}
```

F.2.2 Code for Single Sided Spectra Using Polynomials

The following code is a simple modification of the double sided spectrum code such that only the first half of the spectrum is generated thus reducing the execution time. The magnitudes of the second half of the spectrum are identical to the first.

```
/* SPECTRUM.SP:                                                    1/10/95

Sequential spectral analysis of a signal from zero crossing values producing the
single sided spectrum using polynomial sine/cosine calculation                */

#include <math.h>
#define MAX 64              /* The maximum number of zero crossings possible */
#define t 7.66990393e-4     /* t = 2π / T  the minimum time increment where   */
                            /* T = 8192 = The normalised time period         */
main()
{
volatile int    *INPUT_1 = (volatile int *) 0x808040;     /* Input  channel 1  */
volatile int *OUTPUT_1 = (volatile int *) 0x808041;       /* Output channel 1  */

register int i,j,k=0;           /* Counter registers                           */
int n;                          /* The number of zero crossings                */
int A;                          /* Magnitude of added signal                   */
double Rp;                      /* Real part input data buffer                 */
double Ip;                      /* Imaginary part input data buffer            */
float  a[MAX],b[MAX];           /* Calculation buffers                         */

n = *INPUT_1;
A = *INPUT_1;
for(i=0; i<n; i++,k++)
  {
  Ip = *INPUT_1*t;
  Rp =  cos(Ip);
  Ip =  sin(Ip);
  if(k>=n/2) k=n/2;             /* Limits the number of spectral lines to n/2  */
```

```
   a[k] = 0;
   b[k] = 0;
   for(j=k; j>=0; j--)
    {
     a[j+1] += (Rp*a[j]-Ip*b[j]);
     b[j+1] += (Rp*b[j]+Ip*a[j]);
    }
   a[0] += Rp;
   b[0] += Ip;
  }
 for(i=n/2-1 ;i>=0 ;i--)          /* Magnitude of the spectrum written to the output */
 *OUTPUT_1 = A*sqrt(a[i]*a[i]+b[i]*b[i]);
}
```

F.2.3 Code for Double Sided Spectra Using Look-up Tables

Code to produce spectra from zero crossings using look-up tables is similar to that which uses polynomials to calculate the sine and cosine values. The look-up tables, which contain all the values necessary to calculate a spectrum from a fixed number of zero crossings, are generated on a simple spreadsheet.

The entire code can be contained within a spreadsheet and then exported to a text file which can be subsequently compiled in the usual way. Generating tables for different numbers of zero crossing values is achieved by altering the value of N prior to exporting to the text file . The following code for 24 zero crossings is complete except that only part of the look-up tables SIN[400] and COS[400] are shown:

```
/* SPECTRUM.D24:                                                    16/10/95

Sequential spectral analysis of a signal from zero crossing values producing a
double sided spectrum using a five quadrant look-up table.                    */

        /* Number of zero crossings     N =      24                          */
        /* Period (constant)            T =    8192                          */
        /* Minimum size of CAS[S]       S =      31                          */
        /*                              t_i' =  zero crossing time           */
        /*                              t_0 =   Nyquist interval time        */
```

```
#include <math.h>

main()
{
 volatile int    *INPUT_1= (volatile int *)0x808040;      /* Input  channel      */
 volatile int *OUTPUT_1= (volatile int *)0x808041;        /* Output channel      */

static float CAS[31]={/*  cas (2π n t0 / T) sample time table: t0 / T= 1/N        */

 0.0000000000 , 0.2588190451 , 0.5000000000 , 0.7071067812 ,
 0.8660254038 , 0.9659258263 , 1.0000000000 , 0.9659258263 ,
 0.8660254038 , 0.7071067812 , 0.5000000000 , 0.2588190451 ,
 0.0000000000 ,-0.2588190451 ,-0.5000000000 ,-0.7071067812 ,
-0.8660254038 ,-0.9659258263 ,-1.0000000000 ,-0.9659258263 ,
-0.8660254038 ,-0.7071067812 ,-0.5000000000 ,-0.2588190451 ,
 0.0000000000 , 0.2588190451 , 0.5000000000 , 0.7071067812 ,
 0.8660254038 , 0.9659258263 , 1.0000000000 };

static float SIN[400]={/*   sin (2π ti'/ T )  Zero crossing table          */

 0.0000000000 , 0.0007669903 , 0.0015339802 , 0.0023009692 ,
 0.0030679568 , 0.0038349426 , 0.0046019261 , 0.0053689070 ,
                          ...
                          ...
 0.2961508882 , 0.2968833852 , 0.2976157074 , 0.2983478546 ,
 0.2990798263 , 0.2998116220 , 0.3005432414 , 0.3012746840 };

static float COS[400]={/*  cos (2π ti'/T )  Zero crossing table          */

 1.0000000000 , 0.9999997059 , 0.9999988235 , 0.9999973528 ,
 0.9999952938 , 0.9999926466 , 0.9999894111 , 0.9999855873 ,
                          ...
                          ...
 0.9551411683 , 0.9549137425 , 0.9546857549 , 0.9544572058 ,
 0.9542280951 , 0.9539984231 , 0.9537681899 , 0.9535373956 };

 int i,j,k=0,A,n,z;
 double Rp;                 /* Real part input data buffer                    */
 double Ip;                 /* Imaginary part input data buffer               */
 double a[32],b[32];        /* Calculation buffers                            */

 n = *INPUT_1;              /* n = Number of zero crossings                   */
 A = *INPUT_1;              /* A = Magnitude of transforming signal           */
 for(i=0; i<n; i++,k++)
  {
  z=*INPUT_1;               /* z = Zero crossing time ti'                     */
  Ip=CAS[i]*COS[z]+CAS[i+n/4]*SIN[z];
```

```
  Rp=CAS[i+n/4]*COS[z]-CAS[i]*SIN[z];
  a[k] = 0;
  b[k] = 0;
  for(j=k; j>=0; j--)
   {
    a[j+1] += (Rp*a[j]-Ip*b[j]);
    b[j+1] += (Rp*b[j]+Ip*a[j]);
   }
  a[0] += Rp;
  b[0] += Ip;
 }
for(i=0 ;i<n-1 ;i++)                    /* Outputs magnitude of spectrum    */
*OUTPUT_1 = A*sqrt(a[i]*a[i]+b[i]*b[i]);
}
```

F.2.4 Code for Single Sided Spectra Using Look-up Tables

```
/* SPECTRUM.S24:                                              16/10/95

Sequential spectral analysis of a signal from zero crossing values  producing a
single sided spectrum using a five quadrant look-up table.                  */

        /* Number of zero crossings     N =      24                        */
        /* Period (constant)            T =    8192                        */
        /* Minimum size of CAS[S]       S =      31                        */
        /*                              t_i' =  zero crossing time         */
        /*                              t_0 =   Nyquist interval time      */

#include <math.h>

main()
{
volatile int    *INPUT_1= (volatile int *)0x808040;     /* Input  channel   */
volatile int *OUTPUT_1= (volatile int *)0x808041;       /* Output channel   */

static float CAS[31]={/*   cas (2π n t0 / T )  sample time table: t0 / T = 1/ N   */

0.0000000000 , 0.2588190451 , 0.5000000000 , 0.7071067812 ,
0.8660254038 , 0.9659258263 , 1.0000000000 , 0.9659258263 ,
0.8660254038 , 0.7071067812 , 0.5000000000 , 0.2588190451 ,
0.0000000000 ,-0.2588190451 ,-0.5000000000 ,-0.7071067812 ,
-0.8660254038 ,-0.9659258263 ,-1.0000000000 ,-0.9659258263 ,
-0.8660254038 ,-0.7071067812 ,-0.5000000000 ,-0.2588190451 ,
0.0000000000 , 0.2588190451 , 0.5000000000 , 0.7071067812 ,
0.8660254038 , 0.9659258263 , 1.0000000000};
```

```
static float SIN[400]={/*   sin (2π t_i' / T)   Zero crossing table                */

 0.0000000000 , 0.0007669903 , 0.0015339802 , 0.0023009692 ,
 0.0030679568 , 0.0038349426 , 0.0046019261 , 0.0053689070 ,
                              ...
                              ...
 0.2961508882 , 0.2968833852 , 0.2976157074 , 0.2983478546 ,
 0.2990798263 , 0.2998116220 , 0.3005432414 , 0.3012746840 };

static float COS[400]={/*   cos (2π t_i' / T)   Zero crossing table                */

 1.0000000000 , 0.9999997059 , 0.9999988235 , 0.9999973528 ,
 0.9999952938 , 0.9999926466 , 0.9999894111 , 0.9999855873 ,
                              ...
                              ...
 0.9551411683 , 0.9549137425 , 0.9546857549 , 0.9544572058 ,
 0.9542280951 , 0.9539984231 , 0.9537681899 , 0.9535373956 };

 int i,j,k=0,A,n,z;
 double Rp;                  /* Real part input data buffer                    */
 double Ip;                  /* Imaginary part input data buffer               */
 double a[32],b[32];         /* Calculation buffers                            */

 n = *INPUT_1;               /* n = Number of zero crossings                   */
 A = *INPUT_1;               /* A = Magnitude of transforming signal           */
 for(i=0; i<n; i++,k++)
  {
  z=*INPUT_1;                /* z = Zero crossing time t_i'                    */
  Ip=CAS[i]*COS[z]+CAS[i+n/4]*SIN[z];
  Rp=CAS[i+n/4]*COS[z]-CAS[i]*SIN[z];
  if(k>=n/2) k=n/2;
  a[k] = 0;
  b[k] = 0;
  for(j=k; j>=0; j--)
    {
    a[j+1] += (Rp*a[j]-Ip*b[j]);
    b[j+1] += (Rp*b[j]+Ip*a[j]);
    }
   a[0] += Rp;
   b[0] += Ip;
  }
 for(i=n/2-1 ;i>=0 ;i--)                    /* Outputs magnitude of spectrum   */
 *OUTPUT_1 = A*sqrt(a[i]*a[i]+b[i]*b[i]);
}
```

APPENDIX G

PUBLISHED PAPERS

APPENDIX G PUBLISHED PAPERS

G.1 About the Published Papers

This appendix contains papers published during the course of this project.

G.1.1 A DSP Laboratory Based on a Spreadsheet and TMS320C25 Simulation Software

This paper is concerned with teaching digital signal processing (DSP) using the fixed point TMS320C25 processor and has been included because the system described in it was adapted to suit the TMS320C30 floating point processor which was used in this project. It was presented at the IEE Electronics Colloquium on ' The Teaching of Digital Signal Processing (DSP) in Universities' held in Savoy Place, London on February 16, 1995.

G.1.2 Zero Crossing Techniques for Signal Reconstruction and Spectral Analysis Using the TMS320C30

The second paper gives a brief overview of zero crossing techniques with particular emphasis on using the TMS30C30 and was included in the proceedings of the 'Texas Instruments Fifth Annual TMS320 Educators Conference' held in Houston, Texas in August 10-11, 1995.

G.1.3 Spectral Analysis Using Zero Crossing Techniques

This paper gives more detail on how to implement a zero crossing solution to achieve analogue to digital conversion with emphasis on using look-up tables to speed up computation time. This paper has been accepted for the ' IEE First Communications Conference, Muscat' to be held at the Sultan Qaboos University, Muscat, Oman, on March 11-13, 1996.

G.1.4 Learning about Digital Signal Processing Using Spreadsheets and Simulation Software

The fourth paper is an article included in the IEE 'Engineering, Science and Education Journal', February 1996, volume 5, number 1.

A DSP LABORATORY BASED ON A SPREADSHEET AND TMS320C25 SIMULATION SOFTWARE

G Singh Baicher and J A Sherrington*

Introduction

There have been rapid advances in the field of Digital Signal Processing (DSP) in recent times and several hardware devices, specifically designed to perform the required processing tasks at high speed, have become available. This development highlights the need for the teaching of DSP at an earlier stage of the student's education which in turn demands a simple, low cost , 'hands-on' laboratory experience approach to the subject.

This paper suggests methods of using a spreadsheet package to develop DSP algorithms and also provide a source of input data to the Texas Instruments TMS320C25 Simulator [1] and converts output data from the simulator to give a graphical display of the results. This approach provides an inexpensive but powerful tool for students and other researchers to develop, debug and test DSP algorithms and eliminates the need for a specially designed hardware development system.

Hardware And Software Requirements

The suggested education package is designed to run on a standard IBM compatible PC (286, 386 or 486) with the MS-DOS version 5.0 (or later) operating system and this is the only hardware required for each student. If it is required to run a developed DSP program in real time on a TMS320C25 processor only one target system is required for a large number of students since a simulated program can be easily converted into a format suitable for the target machine.

The following software packages are required:

- The Lotus 1-2-3- Spreadsheet version 2.01 (or later) or the ASEASY Shareware program which is Lotus 1-2-3 compatible.
- TMS 320C25 Fixed Point Assembler [2,3].
- TMS 320C25 Fixed Point Linker [2,3].
- TMS 320C25 Simulator/Debugger [1].
- CONVERT.EXE [4] converts Lotus/Aseasy output to Q-15 format for the TMS320C25 simulator.

The Spreadsheet As A Mathematical Tool

The spreadsheet is an extremely versatile calculation tool which can be used to solve problems ranging from a simple balance sheet to complicated statistical and scientific functions. The applications used by the authors of this paper include a simple rectifier, low-pass FIR and IIR filters, a Fast Fourier Transform and a Fast Hartley Transform. The spreadsheet thus enables the student to easily model a large variety of DSP functions and produce a graphical display without having to learn a programming language such as 'C' or Pascal.

A spreadsheet package is very simple to learn and enables the student to see how a function works in the minimum of time [5].

* G Singh Baicher and J A Sherrington are with the Gwent College of Higher Education

The Spreadsheet As A Link To The TMS320C25 Simulator

It is a simple matter to generate a sinewave or a summation of several sinewaves with different magnitudes, frequencies and phase shifts using a spreadsheet and it is also easy to represent the result of this summation in a 'quasi Q-15 Hex' format which can be written to an external text file. The contents of this text file can be converted to true 'Q-15 Hex' format using the CONVERT.EXE program and then written to an input file for the TMS320C25 simulator.

The reverse conversion can also be performed on the output data generated by the simulator from 'Hex' to 'quasi Hex' format which can then be imported from another text file into the spreadsheet worksheet. In more sophisticated spreadsheet packages such as Excel 5 the data need only be represented in Q-15 decimal format and directly converted to 'Hex' using a built-in function.

This process provides the essential link between the spreadsheet and the TMS320C25 simulator. and thus provides the versatility offered by the approach suggested in this paper.

The Software Simulator and Debugger (SIM2X)

At the heart of the DSP software development system is the simulator/debugger program supplied by Texas Instruments. This program runs the application program in a simulated environment for the TMS320C25 device. The software includes a number of debugging features and memory and register displays that form a visual aid for single-stepping through the TMS assembly coded program.

The input to the simulator is taken from an input file which consists of sample values taken from the spreadsheet program and 'converted' to the hexadecimal 'Q-15' format as required by the simulator program.

The simulator program is run from the MS-DOS prompt using the command 'SIM2X'. The simulator features a powerful and comprehensive set of commands which allows the user to:

- Load and run the application program.
- Define the input and output ports.
- Assign data files to ports.
- Define the program starting point.
- Set breakpoints.

Full details of all commands of the simulator are given in [1]

Software Tools: Assembler, Linker and 'C' Compiler

The assembler translates assembly language source code into an object file in the Common Object File Format (COFF). The assembler will take up to 8 options and can be invoked by a command such as:

```
dspa PROG.ASM PROG.OBJ -v25 -s
```

Where 'dspa' is the command to run the assembler. 'PROG.ASM' is the source file, 'PROG.OBJ' is the COFF file generated, '-v25' specifies the code for the device TMS320C25 and '-s' requires the assembler to place all defined symbols in the object file's symbol table.

The linker creates an executable COFF object file by resolving all addresses between input files. It also allocates sections into the target system's memory map.

In most cases a linker command file is first created using a screen editor such as MS-DOS EDIT. This file allows the user to put in the relevant linking information such as 'MEMORY' and 'SECTIONS' directives to customise the applications program. The command to invoke the linker in this manner is:

'dsplnk LINK.CMD' where LINK.CMD is the linker command file name.

Full details of the assembler and linker programs are given in [2].

The 'C' compiler comprises three separate programs:

- The Parser (dspac.exe).
- The Code Generator (dspcg.exe).
- The Optimiser (dspopt.exe).

The 'C' compiler tools are run in the above order. The result is a text file in TMS320C25 assembler code which can then be assembled and linked to generate the executable COFF object file.

Tutorial -1 The Rectifier Program REC

This is a full-wave rectifier program which reads in a set of sample values representing a sine wave signal on an input port, it then makes each corresponding value positive and returns the result to an output port. The source file for this program is shown in Figure 1.

```
S0  .set  0060h              ; store input data S0 in Block B2
S1  .set  0061h              ; store output data S1 in Block B2
          .text
BEGIN     LARK AR4,49        ; load AR4 with 49 (50 samples for simulator)  ***
          SOVM
          OUT  S1,PA1        ; Output contents of S1.
START     BIOZ NEWSAM        ; is input sample present ?
          B    START         ; loop until it is.
NEWSAM    IN   S0,PA0        ; read new sample and store in data memory (DM) location shown by S0
          LACC S0,16         ; load accum. with left shifted (x16) data in S0
          ABS
          SACH S1,0          ; Store contents of high accummulator in DM shown by S1
          OUT  S1,PA1        ; Send  data word to Output port
          LARP AR4           ; AR pointer to AR4 ***
          BANZ START         ; if AR4 NOT=0 decrement and goto START else ***
LASTOP    .end               ; Set label for end of program, for debugger.
```

Figure 1 Rectifier Program REC

The input signal is generated on the spreadsheet file REC.WK1 and saved as file REC.IN. An MS-DOS batch file called MKREC.BAT is then invoked which commands the following operations to be carried out:

- 'CONVERT' contents of file REC.IN to the hexadecimal 'Q-15' format and place in file REC.IP.
- 'ASSEMBLE' the source code REC.ASM and create an object file called REC.OBJ
- 'LINK' the object file using command file LINK25.CMD to create the executable COFF file REC.OUT.
- Loads the simulator/debugger using the simulator command file SIM25.CMD.

At this stage the simulator screen displays the REC program which is loaded and ready to run. The student can single-step the program and get a visual indication of data being input and stored in a register before finally being applied to the output port. The program is run by pressing function key 'F5'. An output file

from the simulator called REC.OP is then generated. To quit the simulator simply type <quit> at the simulator command prompt.

Batch file MKREC.BAT will automatically 'CONVERT' the contents of REC.OP to the 'quasi hexadecimal' format and place the result in file REC.RES which is suitable for importing into the spreadsheet.

The spreadsheet program is then loaded with the worksheet file REC.WK1 and file REC.RES is imported into the spreadsheet and displayed as an output waveform which is shown in Figure 2.

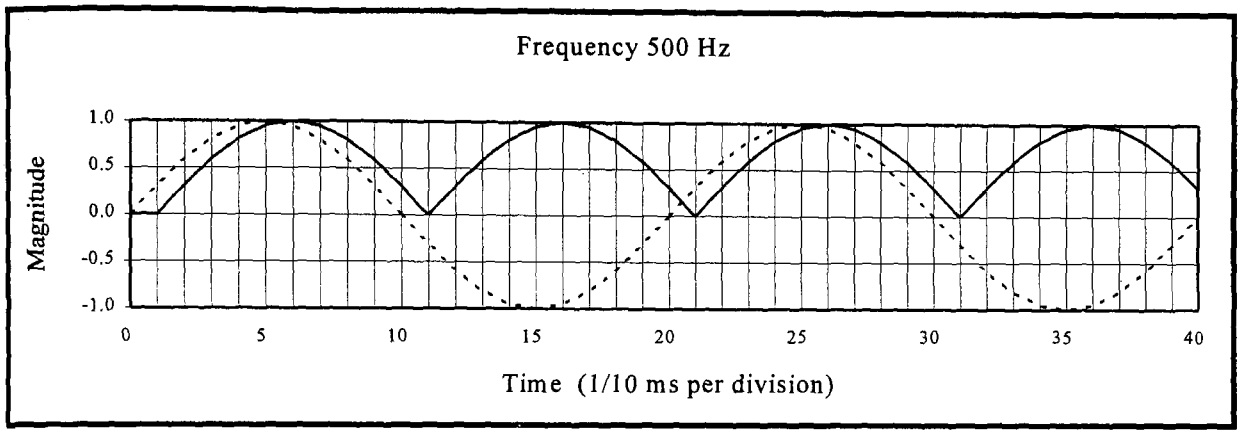


Figure 2 The REC Program Output

Tutorial -2 The FIR Low-pass Filter Program FIRLPF

This tutorial is split into two parts. In the first part a spreadsheet file is created that calculates the FIR low-pass filter coefficients using the Fourier series technique. This worksheet file called FIRLPFIL.WK1 also displays the magnitudes and phase response for the filter. The cut-off frequency and sampling frequency can be altered to any given values. The magnitude response of the filter is shown in Figure 3.

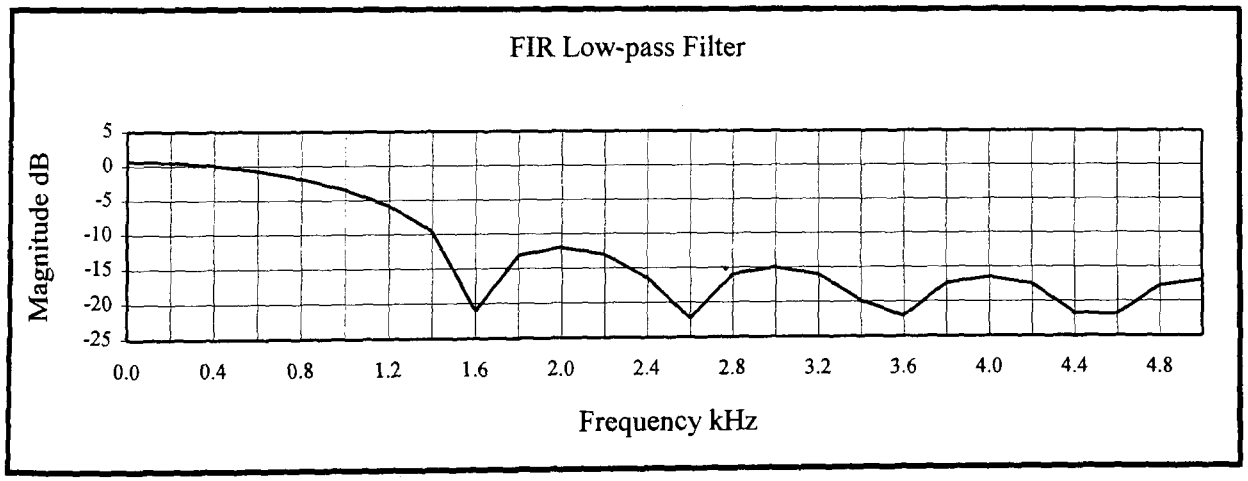


Figure 3 Magnitude response of the FIR Low-pass Filter

The second part of this tutorial uses another spreadsheet FIRLPF.WK1 to generate a composite waveform using the 1st, 3rd and 5th harmonics of a sinusoidal signal. This represents the input signal whose sample values are saved in a file called FIRLPF.IN. Part of the TMS320C25 assembly code that calculates equation (1) is shown in Figure 4:

$$y(n) = a(0) \times (n-10) + a(1) \times (n-9) + \ldots + a(10) \times (n) \quad \text{(1)}$$

```
                .
                .
                .
*   Input Signal
*
LOOP    BIOZ NXTXN    ; BIO goes low when x(n) exists
        B   LOOP      ; loop till new sample is available
NXTXN   IN  XN,PA0    ; input new x(n) at port 0
        LAR AR6,XN    ; load AR6 with new x(n)    ***
*
*   Main program
*
        LARP AR1      ; AR1 for indirect addressing
        LRLK AR1,778  ; load AR1 with DMA 778 i.e. x(n-10)
        MPYK 0        ; product register = 0
        ZAC           ; zero accumulator
        RPTK 10       ; repeat next instruction 10+1 times
        MACD 0ff00h,*- ; calculate y(n)
        APAC          ; accumulate last product
        SACH YN,1     ; left shift 1, store upper 16 bits
*
*   Output Signal
*
        LAR  AR7,YN   ; load AR7 with new y(n)    ***
        OUT  YN,PA1   ; output result to port 1
        LARP AR5      ; AR pointer to AR5
        BANZ LOOP     ; if AR5 NOT=0 decrement, goto LOOP else next ***
        .end
```

Figure 4 Part of the Assembler Code for the FIRLPF Program

In order to execute the filtering program on the simulator, an MS-DOS batch file called MKLPF.BAT is invoked. This batch file performs identical operations to MKREC.BAT detailed in Tutorial-1. The output signal (i.e. low-pass filtered) is displayed as shown in Figure 5.

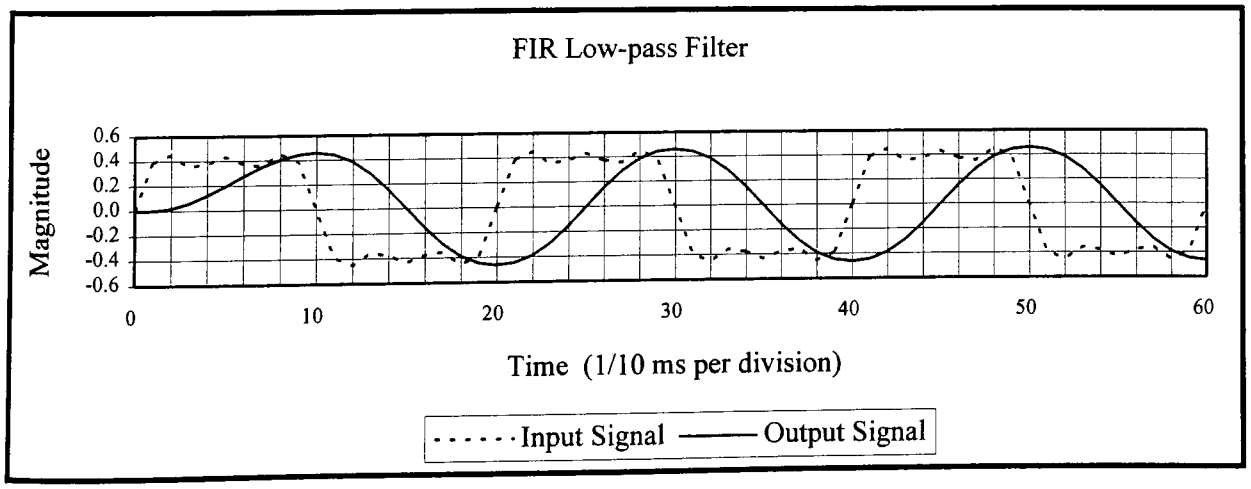


Figure 5 Input and Output Waveforms of the FIR Low-pass Filter

Conclusions

This paper briefly outlines the software development system that has been developed using software methods entirely. The only hardware required is an average IBM compatible PC. Students working on this system are each provided with a "Users Guide and Tutorial" [6] accompanied by a floppy disk containing the various batch and program files. The guide gives a basic understanding of the device architecture, memory maps and an instruction on the use of the assembler, linker and 'C' compiler.

Following simple steps within the "User Guide" the students are able to grasp a working understanding of the code development methods and its implementation to an application program. MS-DOS batch files have proved immensely useful in quickly moving from the spreadsheet to the 'dsptools' environment and cuts down on the tedium of running several programs individually.

The TMS code developed and tested in the software environment by the student can then be converted to a form which can be run in real time on a hardware target system which includes an Analogue Interface Board (AIB). The Texas Instruments utility which enables this is the 'dsphex -t' program which generates a 'TI tagged' object format file which can be loaded into the target machine.

Development and debugging takes most of the time and is therefore better done using independent PC stations without cluttering the entire laboratory with test and measurement equipment hence only one target system is required in the laboratory to serve all students.

This software development system has been used in the Gwent College of Higher Education for the past 3 years and has proved very effective. A number of final year projects involved development of advanced applications programs including 'Bandwidth Reduction of Voice Modulated Signals', Telephone Speech Encryption' and 'Fast Fourier Transform'.

An extended study of the 'Hartley Transform' has also been carried out using the techniques described in this paper [7]. This study included a simulation of a 256 point Fast Hartley Transform and an implementation of a 64 point FHT in real time on the target system.

References

[1] Texas Instruments, 'TMS 320C2X Source Debugger - User's Guide', 1991, document number SPRUO07B.

[2] Texas Instruments, 'TMS 320 Fixed Point DSP Assembly Language Tools - User's Guide', 1990, document number SPRUO18B.

[3] Texas Instruments, 'TMS 320C2X- User's Guide', 1990, document number SPRUO14B.

[4] Gwent College of Higher Education, 'CONVERT.EXE', an internally developed 'C' program, C Day, August 1993.

[5] C C BISSELL, 'Spreadsheets in the teaching of information engineering', IEE, Engineering Science and Education Journal , April 1994, pp 89-96.

[6] Gwent College of Higher Education, 'A User's Guide and Tutorial to the TMS320C25 DSP Development System', an internally published document, Nov 1994.

[7] Gwent College of Higher Education, 'Investigation and Implementation of the Hartley Transform', Final Year B.Eng (Hons) Project report , J A Sherrington, June 1994.

ZERO CROSSING TECHNIQUES FOR SIGNAL RECONSTRUCTION AND SPECTRAL ANALYSIS USING THE TMS320C30

J A Sherrington and G Singh Baicher
Gwent College of Higher Education, Allt-yr-yn Campus, PO Box 180, Newport, Gwent, NP9 5XR, UK

Abstract

This paper briefly describes how a band limited signal, which has been transformed such that the signal has a full complement of real zeros, can be reconstructed and analyzed using zero crossing technique algorithms. This technique offers an alternative to the traditional methods of spectral analysis such as Fourier or Hartley Transforms and has the advantage of avoiding the use of an analogue to digital converter. The method of developing the code using a spreadsheet is described together with a description of its implementation using the TMS320C30 simulator package.

1 Introduction

The major steps required for the zero crossing method are to limit the bandwidth in a similar manner to an anti-aliasing filter, the imaginary zero crossings (the minima) are then converted to real zero crossings by adding a signal, of suitable magnitude and phase, that is equal to or greater than the maximum frequency of the band limited input signal. The positions of real zero crossings are informational attributes of the signal and can be processed to reconstruct and analyse that signal[1]. The block diagram in Figure 1 shows the process.

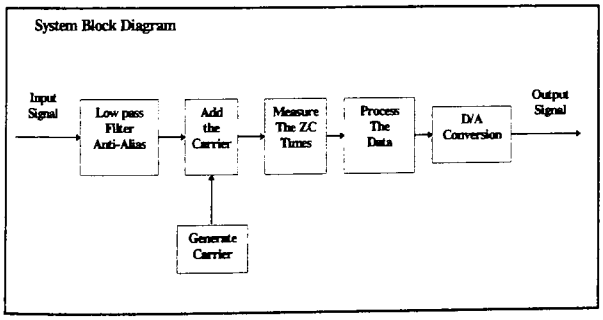


Figure 1 System Block Diagram

The input signal used as an example is 8 sinewaves added together and magnitude weighted for a square wave as shown in Figure 2. It will be noticed that there are only 2 real zero crossings in this signal after the first crossing at time zero but there are several minima which are called complex zeros. If a suitable signal is added the number of zero crossings will be equal to 2M where $M = 2\pi W/\omega_0$ and $\omega_0 = 2\pi/T$. W is the bandwidth and T is the period of the fundamental frequency[2].

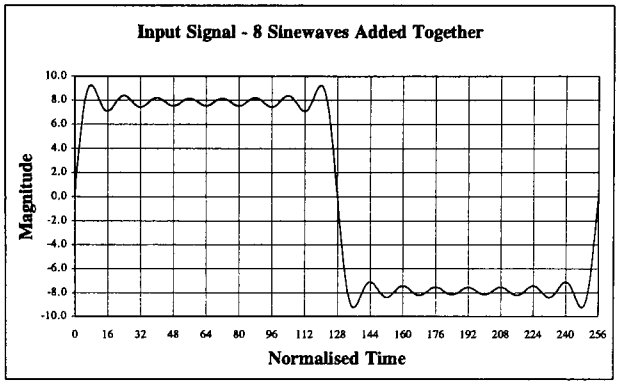


Figure 2 The Input Signal

The resultant signal after the carrier has been added is shown in Figure 3.

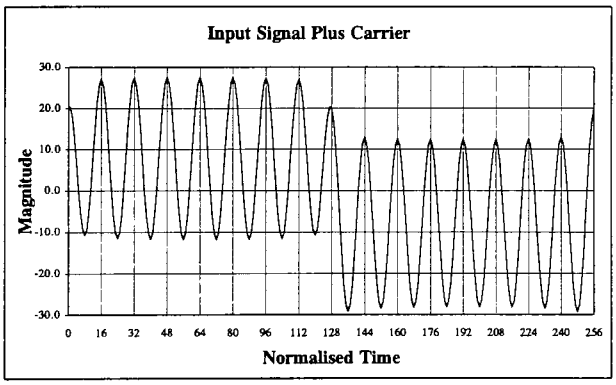


Figure 3 The Input Signal Plus Carrier

The carrier signal in this case is 16 times the fundamental hence the number of zero crossings is 32 and they are all real. A magnified view of Figure 3 is shown in Figure 4. This view shows two zero crossings where the resultant signal is not amplified. Amplification would ensure the position of the zero on the time axis and reduce the chance of noise causing distortion. A level detector and counter can then be used to calculate the time of the zero crossing.

2 Algorithms

Once the values of the zero crossings have been determined any form of processing can take place. The

first process could be the reconstruction of the original signal since it is possible that the received signal could be in zero crossing data form or as a purely digital signal.

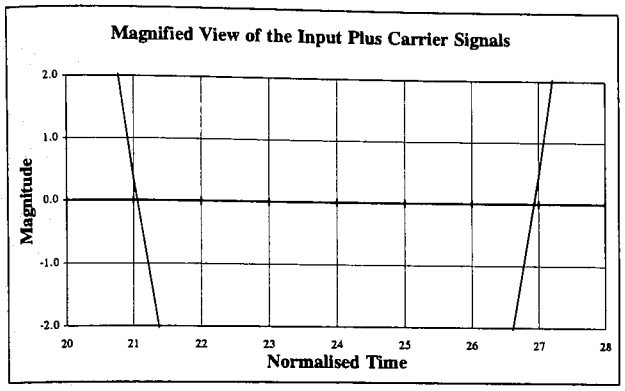


Figure 4 Magnified View of Input and Carrier

The reconstructed signal could then be analysed for spectral content using the well established fast Fourier or fast Hartley transforms. It is to be noted that no A/D conversion has been used in the process.

The equation to reconstruct the original signal from the zero crossing values is as follows[2] :

$$s(t) = 2^{2M} |C_M| \prod_{i=1}^{2M} \sin(t - t_i)\frac{\pi}{T} \qquad (1)$$

where s(t) = The reconstructed signal.
2M = The number zero crossings.
$|C_M|$ = Half the magnitude of the carrier frequency
t = The sample time
t_i = The zero crossing time
T = The period of the fundamental frequency

Calculation of the spectrum can be performed directly from the zero crossing values without the reconstruction process, again with no A/D conversion.

The equation to calculate the input signal frequency spectrum directly from the zero crossing values can be derived from the following equation[2] :

$$x(z) = |C_{M+1}| Z^{-(M+1)} \prod_{i=1}^{2(M+1)} (Z - Z_i) \qquad (2)$$

where $Z_i = e^{j\omega_0 t_i}$ and t_i are zero crossing times
x(z) = The input signal in the frequency domain
2M = The number of zero crossings
$|C_{M+1}|$ = Half the magnitude of the carrier frequency

The task is therefore to devise methods of calculating equations (1) and (2) using the TMS320C30 floating point processor simulator (Version 2.2).

3 The Role of the Spreadsheet

The spreadsheet is a very versatile calculation tool with many mathematical functions available and above all excellent graphical facilities. Figures 2, 3 and 4 were generated by the Microsoft Excel (Version 5) spreadsheet package. Its role can be extended to generate a complete model of the zero crossing process hence ideas can be tried out before programs need be written for the TMS320C30 simulator.

Figure 5 shows an example of the graphical display from a spreadsheet model for signal reconstruction.

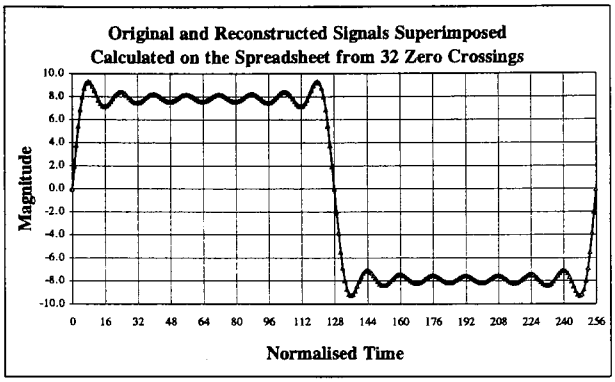


Figure 5 Original and Reconstructed Signals Superimposed

Figure 6 shows the graphical display of the spectrum of the same input signal calculated directly from zero crossing values.

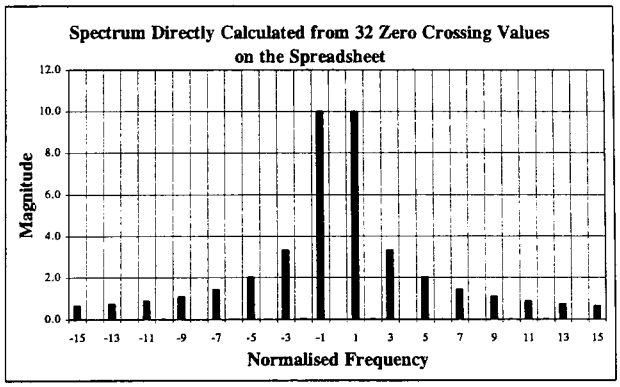


Figure 6 Spectrum of the Input Signal

The generation of the zero crossing values and the spectrum algorithm are both carried out on the spreadsheet. Once the spreadsheet models have been completed they can greatly assist in the generation of 'C' programs that will run on the TMS320C30 simulator.

The spreadsheet can also be used to generate zero crossing values in the correct format and export them to a text file that can be used as input data to the simulator.

The output data generated by the simulator can then be imported into the spreadsheet via a simple text file and displayed using its graphics facilities.

4 Data Format for the TMS320C30

The TMS320C30 simulator[3] can only accept input data as positive integers either in decimal or hexadecimal formats. The text file must only contain integers between the permitted ranges. Negative numbers produced on the spreadsheet can be represented by adding them to 2^{32} (i.e. 4294967296) to give the correct format as shown in Figure 7.

Decimal Integer Value	Input File Integer Value	8-Field Hex Code Value
214748367	214748367	7FFFFFFF
:		:
:	000000002	00000002
:	000000001	00000001
000000000	000000000	00000000
:	4294967295	FFFFFFFF
:	4294967294	FFFFFFFE
:	4294967293	FFFFFFFD
:	:	:
-214748367	214748369	80000001
-214748368	214748368	80000000

Figure 7 Input/Output Data Range

The output data produced by the TMS320C30 must first be converted in the program code to positive integers before it is written to the output text file. This requires that calculation results as floating point numbers be multiplied by a scaling factor to convert them to integers.

The output data produced by the TMS320C30 simulator is in hexadecimal format only hence it has to be converted to decimal format in the spreadsheet. The conversion is simple using Microsoft Excel since the hex/dec function is available, earlier versions of Lotus 123 require the use of string functions which is rather more involved.

5 The Simulator Command File

The TMS320C30 simulator must be instructed so that it knows where in the memory map the essential routines are[4]. The device is memory mapped hence the IO ports must be clearly defined with no overlapping of the memory map space. A typical simulator command file is shown in Figure 8.

This command file is compatible with the memory map defined in the TMS320C3X User Guide[5].

```
; TMS320C30 SIMULATOR COMMAND FILE FOR
;      THE RECONSTRUCTION PROGRAM

ma 0x000000,0x04000 ram
ma 0x808000,0x0010,ram
; 0x808010 to 0x80801F reserved memory

ma 0x808020,0x0020,ram
ma 0x808042,0x17be,ram

ma 0x809800,0x400,ram
ma 0x809c00,0x400,ram

map on
load RECON.OUT
sconfig screen.scr

ma 0x808040,0x001,IPORT
; Configures address 0x808040 as an input  port

ma 0x808041,0x001,OPORT
; Configures address 0x808041 as an output port

mc 0x808040,RECON.PRN,READ
; Connects the input  port to RECON.PRN

mc 0x808041,RECON.OP1,WRITE
; Connects the output port to RECON.OP1

ba exit     ; Adds a break point to halt the program
```

Figure 8 Typical Simulator Command File

6 The Linker Command File

It will be noted from Figure 8 that the simulator requires a file RECON.OUT, in Common Object File Format (COFF), be loaded. To produce this, the file containing the 'C' source code must be assembled, optimised, compiled and linked[6]. The instruction to link the source code calls the linker command file. A typical example is shown in figure 9.

```
/* LINKER COMMAND FILE FOR THE */
    RECONSTRUCTION PROGRAM */

-cr                 /* Specifies internal ram mode */
-e c_int00          /* Specifies the entry point   */
 RECON.OBJ          /* Object code                 */
-l  rts.lib         /* Run-time support library    */
-o RECON.OUT        /* Linked coff output file     */
-m RECON.MAP        /* Memory map file             */
MEMORY
{
VECS : org = 0          len = 0x40 /* Vectors     */
RAM : org = 0x40        len = 0x3FC0
RAM : org = 0x809800 len = 0x800
RAM : org = 0x808020 len = 0x17e0
}
SECTIONS
{
.data:   {} > RAM    /* Data section in RAM    */
.text:   {} > RAM    /* Code                   */
.cinit:  {} > RAM    /* Initialization tables  */
.stack:  {} > RAM    /* System stack           */
.bss:    {} > RAM    /* Bss section in ram     */
vecs:    {} > VECS   /* Reset& interrupt vectors */
}
```

Figure 9 Typical Linker Command File

7 'C' Code for Signal Reconstruction

The code for the reconstruction of a signal is shown in Figure 10.

The first major feature of the code is that the expression '$\sin(t-t_i)$' has been changed, using a simple trig identity, to '$\sin(t).\cos(t_i)-\cos(t).\sin(t_i)$'. This greatly reduces the number of calls to the trig functions and thus makes the program more efficient.

The second feature is that the number of samples produced is the same as the number of zero crossings hence the value of the carrier signal is either +F or -F and thus no call to the sin() function is required to subtract the carrier signal.

It is possible to devise a program to use a look-up table since the zero crossing values are integers ranging between 0 and 16384. This will further improve the run time on a target machine but at the expense of memory space. The table however need have only 4096 locations since all the positive values of sine and cosine are contained in the first quadrant.

```
/* RECONSTRUCTION OF A SIGNAL FROM ZERO
    CROSSING VALUES */

#include <math.h>
#define MAX 128
#define F 1e-3    /* Scaling factor to prevent overflow */
#define PI 3.141592654    /*    π    */
#define t 3.83495197e-4    /* π/8192  */

main()
{
 volatile int    *INPUT_1 = (volatile int *) 0x808040;
 volatile int *OUTPUT_1 = (volatile int *) 0x808041;

 register int i,j,n;
 int   a[MAX];
 double SINA;
 double COSA;
 double SINB[MAX];
 double COSB[MAX];
 float  h,M,reconstruct;

              /* Number of zero crossings */
 n=*INPUT_1;
       /* M = (magnitude of carrier frequency)/F  */
M=*INPUT_1/F;
            /* Stores the zero crossing data  */
  for(i=0 ;i<n; i++)  {
   a[i] = *INPUT_1;
   COSB[i] = cos(a[i]*t);
   SINB[i]  = sin(a[i]*t);
  }
 h=F;
          /* Reconstructs the original signal */
for(i=0;i<=n;i++,h=-h)
  {
   SINA  = sin(i*PI/n);
   COSA = cos(i*PI/n);
   reconstruct = 0.5*F;
   for(j=0 ;j<n; j++)
   reconstruct *= 2.0*(SINA*COSB[j]-COSA*SINB[j]);
   *OUTPUT_1    = M*(reconstruct-h);
   }
} .
```

Figure 10 The Code for Signal Reconstruction

8 'C' Code for the Spectrum Calculation

The program for calculating the spectrum directly from the zero crossing values is shown in Figure 11. The major feature of this program is the 'while loop' which could have been difficult to derive without the assistance of the spreadsheet model.

```
/* SPECTRUM FROM ZERO CROSSING VALUES */

#include <math.h>
#define F 1e-3   /* Scaling factor to prevent overflow */
#define MAX 64
#define t 3.83495197e-4  /* t = 2π/T and  T = 16384  */
main()
{
 volatile int    *INPUT_1 = (volatile int *) 0x808040;
 volatile int *OUTPUT_1 = (volatile int *) 0x808041;

 register int k,i,j;    /* Counter registers                 */
 int n;                 /* The number of zero crossings      */
 int A;                 /* Magnitude of carrier signal       */
 double Rp[MAX]; /* Real part input data buffer          */
 double Ip[MAX]; /* Imaginary part input data buffer    */
 float  Or[MAX];   /* Real part output data buffer       */
 float  Oi[MAX];   /* Imaginary part output data buffer */
 float a[MAX],b[MAX],c[MAX],d[MAX];
      /* Enters the actual number of zero crossings  */
 n = *INPUT_1;
      /* Magnitude of the carrier signal copied to A */
 A = *INPUT_1;
 for(i= 0 ;i<n; i++)
  {
  Ip[i] = *INPUT_1*t;  /* Zero crossing data in Rp[i]  */
  Rp[i] = cos(Ip[i])*F;  /* Real part of Zi in Rp[i]       */
  Ip[i] = sin(Ip[i])*F;   /* Imaginary part of Zi in Ip[i] */
  Or[i] = 0;              /* Output buffer Or[i] cleared  */
  Oi[i] = 0;              /* Output buffer Oi[i] cleared  */
   a[i] = Rp[i];
   b[i] = Ip[i];
  }
 k=0;
 while(k<n-1)   /* Spectrum calculation while loop */
  {
  for(i=0; i<n-k; i++)
   {
   Or[k]+= a[i];   /* Writes accumulated results to */
   Oi[k]+= b[i];   /* the output buffers             */
    c[i] = a[i];
    d[i] = b[i];
    a[i] = 0;
    b[i] = 0;
   }
  k++;
  for(j = 0; j<n-k; j++)
  for(i = 0; i<=j; i++)    /* Calculates the spectrum */
  {
   a[j] += (Rp[j+k]*c[i]-Ip[j+k]*d[i])/F;
   b[j] += (Rp[j+k]*d[i]+Ip[j+k]*c[i])/F;
  }
 }
```

```
/* Spectrum  magnitude written to the output */
for(i=0 ;i<n-1 ;i++)
 *OUTPUT_1 = A*sqrt(Or[i]*Or[i]+Oi[i]*Oi[i])/F;
}
```

Figure 11 The Code for the Spectrum Calculation

9 Performance - Speed and Accuracy

The speed of the program has been assessed by using the clock facility of the simulator where each clock tick is assumed to take 60ns[4]. A graph of estimated run-time against the number of zero crossings is shown figure 12.

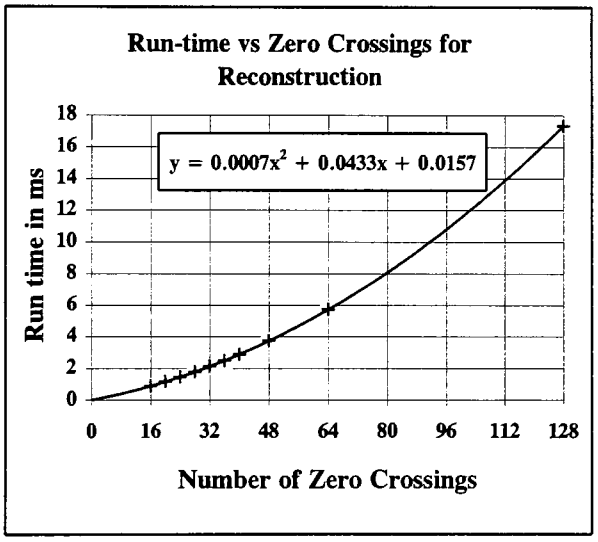


Figure 12 Graph of Estimated Run Time vs the Number of Zero Crossings for Reconstruction

The graph shows the run-time increasing predominantly with the square of the number of zero crossings when the value is large according to the best fit polynomial. The accuracy up to 32 Zero crossings compares well with results obtained from the spreadsheet model (see Figure 5).

A graph of estimated run-time against zero crossings for calculating the full spectrum of the original signal is shown in Figure 13. It shows that the run-time increases predominantly with the cube of the number of zero crossings for large numbers of zero crossings. This suggests that for the algorithm used in this program it may be quicker to reconstruct the signal and then perform a fast Fourier or fast Hartley transform.

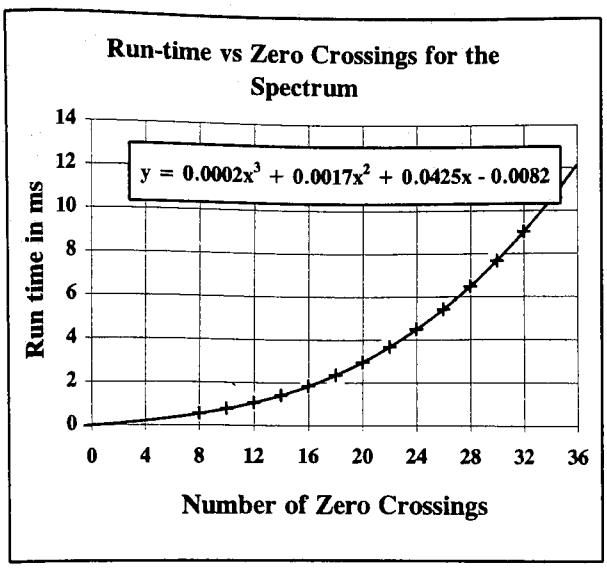


Figure 13 Run-time vs Zero Crossing for the Spectrum Calculation

A further factor that must be considered is the accuracy of the calculation. Figure 14 shows the spectrum calculated by the TMS320C30 from 24 zero crossings giving a result close to that expected. The input signal consists of the first 9 harmonics of a squarewave.

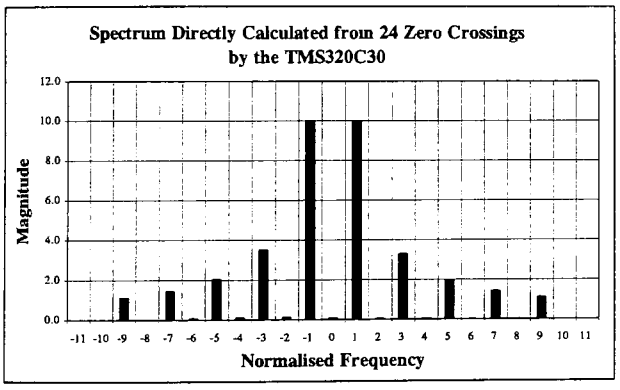


Figure 14 Spectrum from 24 Zero Crossings

Figure 15 shows the spectrum of the same input signal calculated from 28 zero crossings. It is clear from this figure that the errors are very large. If the number of zero crossings is increased to 32 the spectrum obtained is completely unusable due to the errors in the calculation.

If Figure 14 is compared with the result obtained from the spreadsheet (see Figure 6) it can be seen that spreadsheet gives much a better result for 32 zero crossings. The spreadsheet shows that each spectral line is the resultant sum of numbers that depends on the calculation of the previous spectral line. As the frequency decreases the maximum value of a number within the sum becomes larger. Positive numbers are approximately matched by negative numbers hence the differences are the significant factor. In the case of 32 zero crossings the maximum value within a sum is in the order of 4000 but each part of the sum must be calculated to an accuracy of 7 decimal places to obtain the result shown in Figure 6.

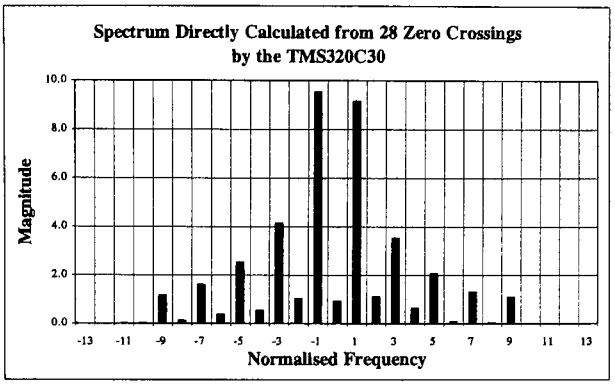


Figure 15 Spectrum form 28 zero Crossings

The program used for the TMS320C30 in this exercise clearly does not achieve this order of accuracy.

10 Conclusion

The reconstruction algorithm could be used in a wide range of applications since it has the general advantage that no A/D conversion is required to produce digital samples of an input signal. The accuracy required is well within the capacity of the TMS320C30 and spectral analysis using the traditional fast transforms gives satisfactory results. Data compression of audio signals, modulation and spectral analysis are feasible applications.

The spectrum algorithm is more limited due to the very high order of accuracy required but it could be an advantage to use it in harmonic analysis of signals containing up to 10 harmonics (i.e. 20 zero crossings).

References

1 Bond F E and Cahn C R, "On Sampling the Zeros of Bandwidth Limited Signals", IRE Transactions on Information Theory,Vol. IT-4, pp. 110-113, Sept1958.

2 Kay S M and Sudhaker R, "A Zero Crossing-Based Spectrum Analyzer", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-34 No 1 February 1986.

3 Texas Instruments, "TMS320C30X C Source Debbugger User's Guide 1993", Document number SPRU053B.

4 Chassaing R (1992), "Digital Processing with C and the TMS320C30 ", John Wiley & Sons Inc, ISBN 0-471-55780-3.

5 Texas Instruments, "TMS320C30X User's Guide 1991", Document number SPRU031B.

6 Texas Instruments "TMS320C30 Optimizing C Compiler Reference Guide 1990", Document number SPRU034D.

SPECTRAL ANALYSIS USING ZERO CROSSING TECHNIQUES

J A Sherrington and G Singh Baicher
Gwent College of Higher Education
Allt-yr-yn Campus, PO Box 180, Newport, Gwent, UK, NP9 5XR

Key Words:
Zero Crossings, Signal Reconstruction, Spectral Analysis.

Abstract:
A practical method for spectral analysis using 'zero-crossing' techniques is investigated. Such a method avoids the need for an analogue to digital converter (A/D) at the input stage by using a transformed signal for the detection of real zeros. The A/D is thus replaced by using a zero-crossing detector circuit. A Texas Instruments TMS320C30 floating point device simulator is used for processing the zero-crossing times for signal reconstruction and spectral analysis.

Introduction

It is well established that time zeros of bandlimited signals are informational attributes of the signal[1-4]. These time zeros may be real (i.e. points of intersection of the time axis) or complex (i.e. minima points). In general, complex zeros are not easily determined although their detection algorithms have been investigated[5]. However, by using a simple invertible transform technique it is possible to convert the input signal to a form that contains only real zeros. The resulting positions of real zero-crossings along the time axis can then be easily and accurately detected[6].

The invertible transformation method used in this investigation is to add a signal of suitable magnitude and frequency to the original bandlimited signal[7]. Figure 1 shows a composite input signal waveform consisting of a fundamental and the first four odd harmonics with their magnitudes weighted to form a truncated squarewave.

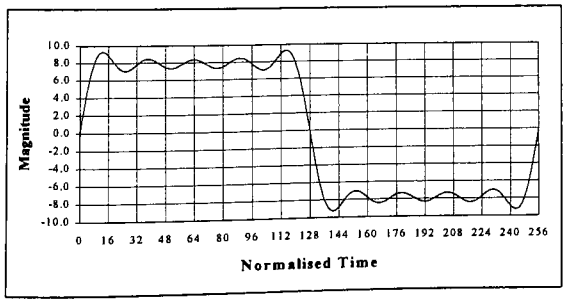


Figure 1 The Input Signal

This waveform comprises real and complex zeros. The transforming signal to be added to the composite signal must be equal to or greater than the maximum frequency of the input signal and its magnitude large enough to ensure all complex zeros are 'forced' to become real. The resultant signal after transformation is shown in Figure 2.

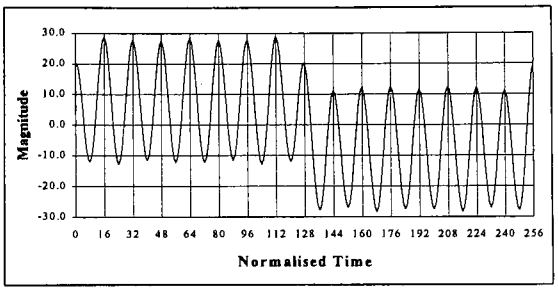


Figure 2 The Input Plus Transforming Signal

The resulting positions of the real zero-crossings along the time axis can then be processed to reconstruct and analyse that signal. The block diagram in Figure 3 shows the process.

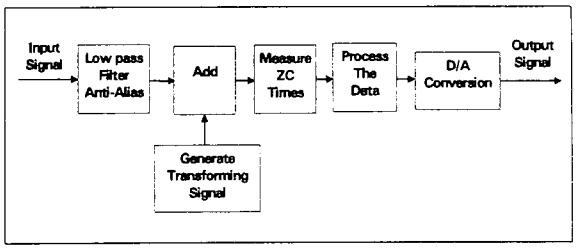


Figure 3 System Block Diagram

Signal Reconstruction

If it is assumed that the transformed input signal s(t) has only real zeros (i.e. all complex zeros are pre-processed to real zeros) then the signal can be reconstructed from the real zero-crossing times t_i using the following relationship[7]:

$$s(t) = 2^{2M} |C_M| \prod_{i=1}^{2M} \sin(t - t_i)\frac{\pi}{T} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (1)$$

where:
s(t) = Reconstructed signal.

2M = Number zero crossings.
$|C_M|$ = Half the magnitude of transforming signal
t = Sample time
t_i = Zero crossing time
T = Period of the fundamental frequency

This equation as it stands requires a large number of sine function calculations. As many samples are required as there are zero crossings for spectral analysis and twice that number for signal reconstruction. If the number of zero crossings were 256 the number of sine function calls would be 65536.

The number of sine function calls can be significantly reduced if equation (1) is modified as follows:

$$s(t) = 2^{2M}|C_M|\prod_{i=1}^{2M}\left[\sin\left(\frac{t\pi}{T}\right)\cos\left(\frac{t_i\pi}{T}\right)-\cos\left(\frac{t\pi}{T}\right)\sin\left(\frac{t_i\pi}{T}\right)\right] \quad \text{..... (2)}$$

The values of $(t\pi/T)$ are fixed and if all the values of $(t_i\pi/T)$ are known the sines and cosines need only be calculated once prior to the product expression being evaluated. The total number of sine and cosine function calls is reduced to 1024 for 256 zero crossings. The product is therefore the result of simple multiplications and additions.

A look-up table is also possible using this relationship but it would be very large. If there were 256 zero crossings with a resolution of 1Hz and 8192 possible values of t_i a look-up table of 4097 values would be required. This would not give a good reconstruction since there are only 32 possible values per Nyquist interval (i.e. half the period of the transforming signal). The size of the look-up table can be substantially reduced if the values of the zero crossings are related to the Nyquist interval[6].

This depends on the fact that only one zero crossing will occur within a Nyquist interval t_0 of the transforming signal as shown in Figure 4.

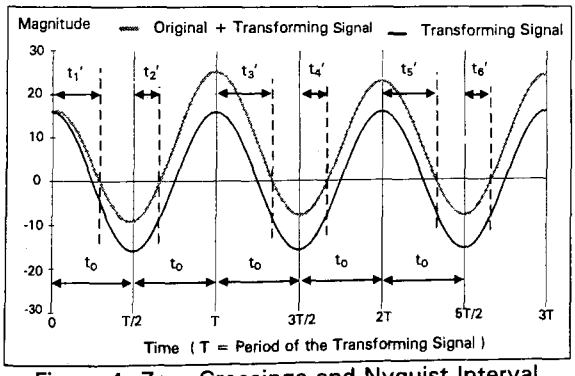


Figure 4 Zero Crossings and Nyquist Interval

The zero crossing times are: $t_i = n\,t_o + t_i'$ where n = 0, 1, 2 ... 2M and 2M is the total number of zero crossings within the resolution period. The calculation for sines and cosines can thus be obtained from the simple trig identities:

$$\sin(t_i\pi/T) = \sin(nt_o\pi/T)\cos(t_i'\pi/T) + \cos(nt_o\pi/T)\sin(t_i'\pi/T)$$

$$\cos(t_i\pi/T) = \cos(nt_o\pi/T)\cos(t_i'\pi/T) + \sin(nt_o\pi/T)\sin(t_i'\pi/T)$$

where T = the fundamental period.

Only one quadrant of the cosine is required for all values of the sine and cosine of $nt_o\pi/T$ since n is an integer and $nt_o\pi/T$ ranges from 0 to π but a 3 quadrant table is more efficient in terms of speed. The table for the sine values required for 256 zero crossings consist of the first 256 values of SIN ($t_i'\,\pi$ /T) since t_i' ranges in integers from 0 to 256. A similar situation applies for the cosine values of the zero crossings.

Code to achieve reconstruction can thus be written using three small look-up tables without penalty in program execution time. With this arrangement the total table size for 256 zero crossings is 897 locations (385 + 256 + 256) giving a resolution of 65536 points for the fundamental period compared with a table size of 4097 giving a resolution of only 8192 points. In this example there are 256 values per Nyquist interval. The 'C' code program for reconstruction, using a look-up table, written for the TMS320C30 floating point processor simulator is shown in Figure 5.

```
/* RECON.C: SIGNAL RECONSTRUCTION
   FROM ZERO CROSSING VALUES USING
   A 3 QUADRANT LOOK-UP TABLE        */

#define     N         32

     /* Period (i.e. n*256)          T =  8192  */
     /* Minimum size of CAS[S]   S = 49      */

main()
{
 volatile int  *INPUT_1 =(volatile int *)0x808040;
 volatile int *OUTPUT_1 =(volatile int *)0x808041;

static float CAS[49] ={/* cas(nπt/T)  table: t/T =/N*/
 0.0000000000,0.0980171403 , ... -1.0000000000 };

 static float SIN[256] ={  /* 2*sin(ti*π/T) table  */
 0.0000000000 , 0.0007669904 , ... 0.1952709691};

 static float COS[256] = {  /* 2*cos(ti*π/T) table  */
 2.0000000000,1.9999998529 , ... 1.9904444852 };

    int  i,j,k,M;
    float SINB;                          /* sin(ti*π/T) */
    float COSB;                          /* cos(ti*π/T) */
    static float reconstruct[N + 1];

  /* M = (magnitude of transforming signal)x100  */

M = *INPUT_1
                                          Continued ...
```

```
for(i=0; i<=N; i++) /* Initialises the output array */
reconstruct[i]=0.5;
for(i=0; i<N; i++) /* Reconstructs original signal*/
  {
   k=*INPUT_1;    /* k = Zero crossing time ti */
   SINB=SIN[k]*CAS[i+N/2]+COS[k]*CAS[i];
   COSB=COS[k]*CAS[i+N/2]-SIN[k]*CAS[i];
  for(j=0 ;j<=N ; j++)
   {
    k=j+N/2;
    reconstruct[j]*=CAS[j]*COSB-CAS[k]*SINB;
    }
  }
  k=1;
  /* Outputs the reconstructed signal */
 for(i=0 ;i<=N ; i++,k=-k)
  *OUTPUT_1=M*(reconstruct[i]-(float)k);
 }
```

Figure 5 TMS320C30 'C' Code for Reconstruction using Look-up Tables

Spectral Analysis

Once the signal has been reconstructed spectral analysis can be performed using a fast Fourier transform (FFT) that will yield frequency, magnitude and phase. Very efficient FFT programs have been written, for example a 1024 point radix-4 FFT executes in 3.75 ms using a TMS320C30 processor[9]. A single chip discrete Fourier transform (DFT) which achieves an 8192 point complex DFT in 400μs has recently been reported[10].

It is also possible to perform spectral analysis directly from the zero crossing values of the transformed signal without first reconstructing the signal by using the following relationship[7]:

$$X(z) = |C_{M+1}| z^{-(M+1)} \prod_{i=1}^{2(M+1)} (z - z_i) \quad \ldots\ldots\ldots\ldots (3)$$

where:

$z_i = e^{j\omega_0 t_i}$

t_i = Zero crossing times

$x(z)$ = Input signal in the frequency domain

$2M$ = Number of zero crossings

$|C_{M+1}|$ = Half the magnitude of transforming signal

The Fourier coefficients are obtained by comparing the coefficients of z in equation (3) with the coefficients of z of the complex Fourier series in equation (4):

$$X(z) = \sum_{m=-(M+1)}^{M+1} c_m z^m \quad \ldots\ldots\ldots\ldots (4)$$

This process is well described by Kay and Sudhaker[7] who suggest two methods. The first uses Newton's formula which works well for coherent signals, a common situation in optics where such signals are analysed[8]. For non-coherent signals the real and imaginary parts of z_i must be separated which proves not to be computationally efficient and suffers from calculation accuracy problems.

The second method is sequential and yields the coefficients such that the real and imaginary parts are easily separated. This method, however, has the major disadvantage that the processor accuracy required is very high if the number of zero crossings is greater than 26.

The performances of the reconstruction and spectral analysis algorithms (for both single and double sided spectra) are shown in Figure 6. This indicates that the execution times increase approximately with the square of the number of zero crossings for both algorithms.

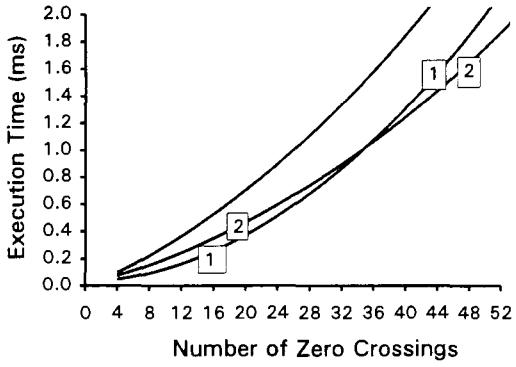


Figure 6 Zero Crossings Vs Execution Time

Trace 1 Reconstruction.
Trace 2 Single sided spectrum.
Trace 3 Double sided spectrum.

The 'C' code for the TMS320C30 simulator to directly calculate the discrete Fourier transform coefficients from real zero-crossing times by the sequential method using a 5 quadrant look-up table is shown in Figure 7.

The single sided spectra derived from 24 and 28 zero crossings for the signal shown in Figure 1 are shown in Figures 8 and 9 respectively. Severe calculation errors are clearly shown in Figure 9. The errors increase rapidly as the number of zero crossings increase.

```
/* SPECTRUM.C: SINGLE SIDED SPECTRAL
 ANALYSIS FROM ZERO CROSSING VALUES
 USING A 5 QUADRANT LOOK-UP TABLE*/

   /* Number of zero crossings N =      24   */
   /* Period (constant)          T =    8192  */
   /* Minimum size of CAS[S]     S =      31   */

#include <math.h>
                                   ...Continued
```

```
main()

{
volatile int   *INPUT_1  = (volatile int *)0x808040;
volatile int *OUTPUT_1  = (volatile int )0x808041;

static float CAS[31] = {/*cas(2nπt/T)table t/T = 1/N*/
0.0000000000,0.2588190451 , ... 1.0000000000 };

static float SIN[400] = {/*sin(2πti/T)table */
0.0000000000,0.0007669903 ,... 0.3012746840 };

static float COS[400] = {/* cos(2πti/T)table */
1.0000000000,0.9999997059 ,... 0.9535373956 };

  int i,j,k=0,A,n,z;
  float Rp;       /*  Real part input data buffer       */
  float Ip;       /*  Imaginary part input data buffer*/
  float  a[32],b[32];          /* Calculation buffers  */

  n  =  *INPUT_1; /* n  =  Number of zero crossings*/
  A  =  *INPUT_1;
  /* A  =  Magnitude of transforming signal          */

for(i=0; i<n; i++,k++)
 {
  z=*INPUT_1;      /* z  =  Zero crossing time ti  */
  Ip=CAS[i]*COS[z]+CAS[i+n/4]*SIN[z];
  Rp=CAS[i+n/4]*COS[z]-CAS[i]*SIN[z];
   if(k>=n/2)  k=n/2;
   a[k]  =  0;
   b[k]  =  0;
    for(j=k; j>=0; j--)
      {
       a[j+1]  +=  (Rp*a[j]-Ip*b[j]);
       b[j+1]  +=  (Rp*b[j]+Ip*a[j]);
      }
     a[0]  +=  Rp;
     b[0]  +=  Ip;
    }
       /* Outputs magnitude of spectrum   */
   for(i=n/2-1 ;i>=0 ;i--)
   *OUTPUT_1  =  A*sqrt(a[i]*a[i]+b[i]*b[i]);
  }
```

Figure 7 TMS320C30 'C' Code for Direct Generation of the Spectrum

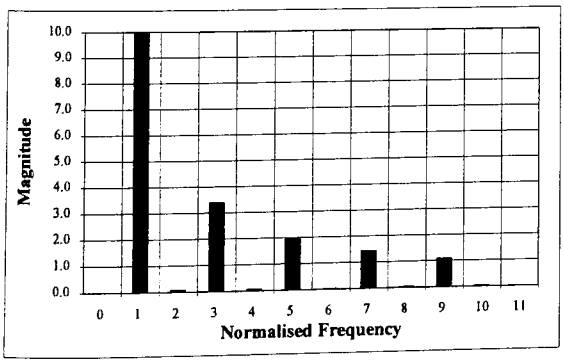


Figure 8 Spectrum from 24 Zero Crossings

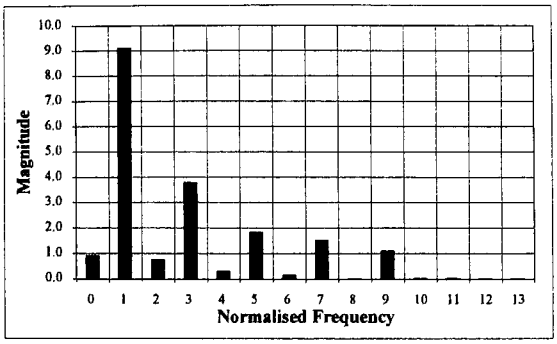


Figure 9 Spectrum from 28 Zero Crossings

Conclusion

Spectral analysis can be successfully performed using zero crossing techniques which exploit the advantage that measuring zero crossing times is inherently simpler and more accurate than analogue to digital conversion.

Direct spectral analysis for up to 26 zero crossings is feasible but in general it is better to first reconstruct the signal and then use a fast Fourier transform to yield the spectrum. This latter approach gives satisfactory results with 256 zero-crossings using the TMS320C30 simulator.

The efficient use of memory for look-up tables suggested in this paper offers the possibility of developing an assembly code which will improve the speed of reconstructing the signal. A further improvement could be achieved with parallel processing producing several output samples simultaneously thus improving the throughput.

References

[1] BOND, F.E., and CAHN, C.R.: 'On sampling the zeros of bandwidth limited signals', *IRE Trans. Inform. Theory*, 1958, **IT-4**, pp. 110-113

[2] VOELCKER, H.B.: 'Towards a unified theory of modulation, Part I: Phase envelope relationships', *Proc IEEE*, 1966, **54-3**, pp. 340-353

[3] VOELCKER, H.B.: 'Towards a unified theory of modulation, Part II: Zero manipulation', *Proc IEEE*, 1966, **54-5**, pp. 735-755

[4] SEEKY, A.: 'A computer simulation study of real-zero interploation',*IEEE Trans*, 1970, **AU-18**, pp. 43-54

[5] AL-JALILI, Y.O.: 'Analysis and detection algorithms for complex time zeros of bandlimited signals', *IEE Proc-1*, 1991, **138**, pp. 189-200

[6] SALOMA, C., and DARIA, V.R.: 'Performance of a zero-crossing spectrum analyser', *OPTICS LETTERS*, 1993, **18-17**, pp. 1468-1470

[7] KAY, S.M., and SUDHAKER, R.: 'A zero crossing-based spectrum analyser', *IEEE Trans*, 1986, **ASSP-34**, pp. 96-104

[8] SALOMA, C., and HAEBERLI, P.: 'Optical spectrum analysis from zero crossings', *OPTICS LETTERS*, 1991, **16-19**, pp. 1535-1537

[9] JONES, N.B., and MCK WATSON, J.D.: 'Digital Signal Processing and Applications, principles, devices and applications' (Peter Peregrinus on behalf of the IEE, 1991, ISBN 0 86341 2106)

[10] BIDET,E.,CASTELAIN,D.,JOANBLANQ,C., and SENN,P.: 'A fast single-chip implementation of 8192 complex point FFT', *IEEE J. Solid-State Circuits*, March 1995, **30**, (3) pp. 300-305

Learning about digital signal processing using spreadsheets and simulation software

by G. Singh Baicher and J. A. Sherrington

The ready availability of low-cost high-speed digital signal processor chips has resulted in their widespread use in many areas of industrial and consumer electronics for processes such as filtering, coding, estimation and spectral analysis. This article explores the opportunities of learning about digital signal processing (DSP) using a spreadsheet and a specialised simulation software package, developed by Texas Instruments for their TMS320C25 processor, that mimics the device in terms of its functional capabilities. A finite impulse response (FIR) digital filter is used as a typical example to demonstrate the ideas presented. Application programs can thus be entirely tested in a software environment before being used on a real-time target system.

Introduction

Digital signal processing (DSP) has seen rapid advances in recent years with a number of advanced engineering applications moving away from their traditional analogue processing base towards digital processing methods. In response to this demand, a number of colleges and universities have now included DSP as part of their core curriculum at the undergraduate level of the electrical/electronic programme.

A number of different strategies can be implemented for the effective teaching of DSP at this level. An interesting outcome of a recent IEE Colloquium[1] on 'The teaching of DSP in universities' was the obvious split between the undergraduate and postgraduate educators. Those teaching at the undergraduate level mostly emphasised strongly the importance of 'real-time' systems as a laboratory back-up support to reinforce theoretical concepts. Postgraduate educators were content with using simulation methods to cover advanced DSP applications; especially favoured was MATLAB. However, in either case a fairly hefty budget within the department must be allocated to support the resourcing of such a DSP laboratory. Another problem with real-time systems is that they are based on specific processors which become obsolete in a few years as newer and faster processors are developed and supersede the older ones.

This article discusses a software simulation method for implementing a complementary DSP laboratory that can be used in conjunction with a real-time system such that the implementation of the 'hands-on' feature can be most effective, efficient and economic. For the effective teaching of DSP there are three distinct stages that students must follow, these are:

- learning about theoretical concepts and being able to develop conceptual outcomes graphically
- learning about DSP hardware/software and possibly being able to write a programme code in machine or high-level language
- being able to run a specific applications program and to interact signals on a target system.

Theoretical concepts in the first stage are best covered by formal instruction in a classroom situation, however the second and third stages are best suited to a laboratory environment.

Simulation methods are sometimes frowned upon in scientific and engineering circles due to the 'lack-of-feel' of the real-world interaction and there is some truth in this. On the other hand there is a substantial overhead imposed in implementing 'real-time' systems. Many times such an imposition becomes

counter-productive to giving the broadest exposure to learning about concepts, systems and practice. In effectively understanding DSP and its applications, a large part of the learning time and experience is devoted to recognising the specific processor architecture, developing a machine or high-level language code and implementing it on a target system. It would therefore be a significant time-saving exercise if the signal generation, and code development/debugging and execution could be carried out in a software environment. What then remains to be done is to implement the code on a real-time system with interaction with real signals.

In this article, the process of software simulation uses a spreadsheet program for generation and display of signal waveforms and a Texas Instruments TMS320C25 Simulator[2] for processing the signals as though a real DSP device were being used. In recent years Texas Instruments' DSP devices, both the fixed-point and floating-point types, have established themselves as market leaders and have been particularly popular in the educational environment due to competitive pricing for educational institutions and the availability of simulator packages and low-cost real-time target systems. This was also evident at the IEE colloquium[1], where a number of papers that were presented used the TMS320C25 processor as the basis for the development of DSP target systems. Added to this is the fact that a number of other well known suppliers (such as Loughborough Sound Images and Atlanta Signal Processors) are producing value-added TI DSP products for general and specialist applications.

In an educational environment, there are significant advantages in using software simulation methods for teaching DSP. Some of these are that:

- the only hardware required is an average-performance IBM-compatible PC
- there is the possibility for an 'open-access' student learning centre
- it is easier to upgrade or adapt to newer processors at minimal cost
- scaling and aliasing problems are readily resolved
- only one hardware target system is required for real-time processing of signals using the code developed, debugged and tested using the simulator software.

Spreadsheet as a calculator and graphical display

Spreadsheets have been used extensively in mathematical, scientific and engineering fields for the purpose of calculating physical quantities, mathematical modelling and signal analysis[3–5]. The most useful aspects of a spreadsheet package, apart from its ubiquitous status and low cost, are its graphical display and recalculation features. An entire set of figures is recalculated by changing the value of a single 'cell', which may be the cut-off frequency of a filter or some scaling factor.

There are several other generic mathematical packages, such as Mathcad, Mathematica and MATLAB, that are increasingly becoming popular in the educational and industrial sectors. However, in terms of cost and flexibility, a spreadsheet package is probably the most suitable for developing examples relating to a specific application that the students themselves may first try out. For the purpose of this article we have used the Lotus 1-2-3 (Version 2.01) package although other packages, such as Microsoft Excel (Version 5) and Shareware ASEASY (Lotus 1-2-3 compatible), could also be used. The spreadsheet has been used for the following processes:

- developing and displaying a 'simulated' input signal, which may be a single sinusoid of a certain frequency or a composite waveform formed by adding a number of sinusoids each with a different amplitude, frequency and phase
- developing and displaying the magnitude response of a finite impulse response (FIR) digital filter and calculating the coefficients
- 'importing' the processed output signal into the spreadsheet and then displaying it
- developing a 256-point fast Fourier transform (FFT) and using this for spectral analysis of the input and output signals.

An FFT is a computationally efficient form of the discrete Fourier transform that translates periodic signals from the time domain to the frequency domain.

FIR low-pass filter on a spreadsheet

The design of digital filters is extensively covered in standard textbooks on DSP[6,7]. The articles on digital signal processing by Grant and McDonnell[8,9] are particularly good due to their conceptual clarity without involving the rigours of extensive mathematical detail. We do not intend to cover here the fundamental concepts or design criteria and implementation of DSP applications but rather to consider the software and simulation methods to achieve a 'feel' for DSP techniques and their implementation. For this we will show how to bridge the gap between a spreadsheet package and a DSP device simulator for executing a DSP application entirely in software.

For the purpose of explaining the full process involved, we will use the example of a FIR digital filter of the form shown in Fig. 1. The transfer function of such a filter[8] is given by:

$$H(z) = \sum_{n=0}^{N} a(n) z^{-n} \qquad (1)$$

where $a(n)$ is the nth multiplier coefficient and z^{-n} implies n sample delays. Note that in Fig. 1 z^{-1}

represents a unit sample time delay of T seconds and the sampling frequency of the input signal is given by $f_s = 1/T$ Hz.

From eqn. 1, the magnitude response is obtained as:

$$|H(\omega)| = a\left(\frac{N}{2}\right) + 2\sum_{n=0}^{(N/2)-1} a(n)\cos\left\{\left(\frac{N}{2} - n\right)\omega t\right\} \quad (2)$$

where ω is the radial frequency (i.e. $\omega = 2\pi f$ radians/s).

A simple method of designing an FIR filter is based upon approximating the desired response with a finite Fourier series. Thus, the filter length must be limited to some finite value given by $(N + 1)$, as indicated in Fig. 1. In this example we will use $N = 20$, resulting in a 21-tap FIR filter.

The impulse response of the approximated transfer function for a 'low-pass' FIR filter is given by the series:

$$h(m) = \frac{1}{m\pi}\sin\left(\frac{2\pi m f_c}{f_s}\right) \quad (3)$$

where $m = -10, -9, -8, \ldots, -1, +1, +2, \ldots, +10$, f_c = the cut-off frequency, f_s = the sampling frequency and $h(0) = 2f_c/f_s$. The above series represents coefficients of the FIR filter in its non-causal form. A causal filter implementation is achieved by introducing a delay of $N/2$ (i.e. 10) samples. The filter coefficients are then given by:

$$a(n) = h\left(n - \frac{N}{2}\right) \quad (4)$$

where $n = 0, 1, 2, \ldots, 20$. Note that the impulse response is symmetrical about $a(N/2)$ since $a(n) = a(N - n)$.

The truncation of the infinite series for implementation of an FIR filter amounts to multiplying it by a rectangular window function that is defined as:

$$w_R(m) = \begin{cases} 1 & |m| \le \frac{N}{2} \\ 0 & \text{elsewhere} \end{cases}$$

Several other window functions exist that generate the corresponding Fourier transform with reduced sidebands but wider main lobe. The Hamming and Blackman window functions are defined respectively as:

$$w_H(m) = \begin{cases} 0{\cdot}54 + 0{\cdot}46\cos\left(\frac{2\pi m}{N}\right) & |m| \le \frac{N}{2} \\ 0 & \text{elsewhere} \end{cases}$$

and

$$w_B(m) = \begin{cases} 0{\cdot}42 + 0{\cdot}50\cos\left(\frac{2\pi m}{N}\right) + 0{\cdot}08\cos\left(\frac{4\pi m}{N}\right) & \\ & |m| \le N/2 \\ 0 & \text{elsewhere} \end{cases}$$

The above equations are implemented on a spreadsheet to give a graphical display of the magnitude response as shown in Fig. 2. The calculated filter coefficients are updated according to the selected cut-off frequency. These coefficient values are subsequently used in the development of the DSP assembler code for implementing the FIR low-pass filter.

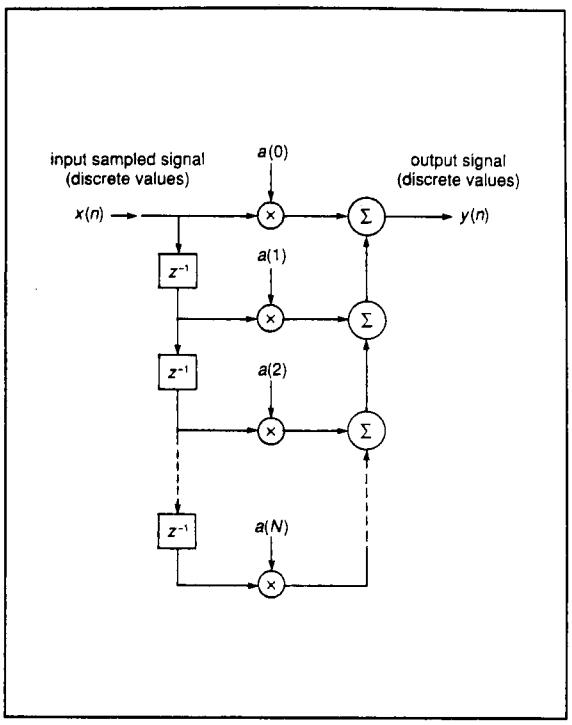


Fig. 1 FIR digital filter network diagram

The TMS320C25 device simulator

This software package forms the hub for signal processing applications. The TMS320C25 Simulator mimics the hardware DSP chip in its functional features. The Simulator also includes a debugger of the source code with several useful features such as:

- reverse assembly of the code for identifying source code location
- loading and running application programs
- single stepping through the source code and adding break points
- program and data memory location and content identification
- registers and their content display
- reading from an input ASCII file and writing to an output file.

The two commonly used simulators supplied by Texas Instruments are the fixed-point TMS320C2X/5X and the floating-point TMS320C3X/4X. The main features for both of these types of simulators are similar although there are important differences in the way the signal sample values are interacted. The fixed-point device uses an IN and OUT instruction from its op-code set to read data from and write data to an ASCII file.

	A	B	C	D	E	F	G	H	I	J	K
1				FIR Low-pass Filter							
2											
3				21 coefficients							
4											
5	Cut-off	Coeff	Coeff	Freq	wT	Mag'd	Mag'd dB	Phase	3 dB	Normalised	Blackman Window
6	freq kHz		Scaled by	kHz	Samp freq	Scaling	Scaling	degrees	point	dB Magnitude	
7	I		2^15		16000	1	0				
8											
9	-10	-0.0225	-738	0.00	0.00	1.1043	0.4308	0.00	-3.0	0.00000	0.00000
10	-9	-0.0135	-444								0.00919
11	-8	0.0000	0								0.04021
12	-7	0.0174	570								0.10139
13	-6	0.0375	1229								0.20077
14	-5	0.0588	1927								0.34000
15	-4	0.0796	2608								0.50979
16	-3	0.0980	3212								0.68917
17	-2	0.1125	3688								0.84923
18	-1	0.1218	3992								0.96025
19	0	0.1250	4096								1.00000
20	1	0.1218	3992								0.96025
21	2	0.1125	3688								0.84923
22	3	0.0980	3212								0.68917
23	4	0.0796	2608								0.50979
24	5	0.0588	1927								0.34000
25	6	0.0375	1229								0.20077
26	7	0.0174	570								0.10139
27	8	0.0000	0								0.04021
28	9	-0.0135	-444								0.00919
29	10	-0.0225	-738								0.00000
30				1.31	0.52	0.1598	-7.9639	32.34	-3.0	-8.39469	
31				1.38	0.54	0.1005	-9.9768	25.31	-3.0	-10.40761	
32				1.44	0.56	0.0477	-13.2126	18.28	-3.0	-13.64345	
33				1.50	0.59	0.0023	-26.4564	11.25	-3.0	-26.88719	

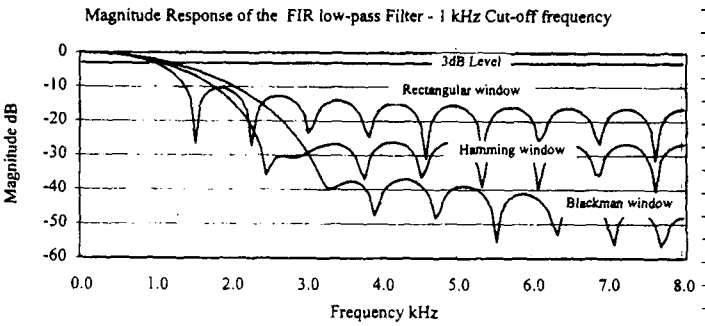


Fig. 2 Magnitude frequency response of a 21-tap FIR low-pass filter

On the other hand the floating-point device uses a memory mapped I/O feature for reading and writing data. The op-code instruction sets of the fixed-point and floating-point processors are entirely different, so a careful assessment of the application is essential before deciding on the type of device that may be most beneficial. The specific application program is first written in source code, which is then assembled and linked[10,11]. This generates an executable file in 'common object file format' (COFF) form. This COFF file is then loaded into the simulator and the source code can then be debugged and finally run. Alternatively, the application program can be written in a high-level language such as 'C' and then compiled. The 'C' compiler generates the assembler source code file, which may then be assembled and linked to generate the COFF file for running on the simulator.

Input and output signal samples

For digitally processing real signals, an analogue-to-digital (A/D) converter is required at the input stage to convert the input signal into sampled digital values. The processed signal values are then passed through a digital-to-analogue (D/A) converter to the output. For real-time processing, sampled input values must be processed and transferred to the output before a new input signal value is accepted for processing.

In using a simulator package, the input signals are generated in the spreadsheet and the sample values are saved sequentially in an ASCII MS-DOS file with each sample value on a new line. The input sample values must be in a 'Q-15' format. This is a 4-field hexadecimal representation of a sample value within the range +1 (approximately) and −1. This requirement is specific to the TMS320C25 simulator and so an additional stage of transforming the decimal signal value in the spreadsheet to the Q-15 format must be applied. This transformation can be done entirely in the spreadsheet or it can be done using some utility program specially written for this purpose. We have performed this transition partly in the spreadsheet and then used an in-house developed 'C' program to convert to the required Q-15 format. Table 1 shows the data values and their Q-15 representations.

The processed signal is output by the simulator in the same Q-15 format, which must then be converted back to its decimal equivalent values and imported into the spreadsheet and displayed. Although the whole process of 'converting' and 'importing' data files from one format to another may appear to be somewhat tedious, in practice the extensive use of MS-DOS batch files and spreadsheet macros makes the process relatively simple. A block diagram of the complete system is shown in Fig. 3.

In Gwent College, this procedure has been used for the last few years by the final year BEng students. Each student is provided with a 'User's guide and tutorial' handbook along with a floppy disk containing the batch files and applications programs. Within a few hours the students are able to gain a quick understanding of the process involved and are able to guide

themselves effectively from start to finish of a simple full-wave rectifier program that forms the first tutorial.

FIR low-pass filtering—using the device simulator

In order to demonstrate the process of filtering using the TMS320C25 Simulator, an input composite signal waveform is generated on the spreadsheet. This signal is formed by adding together a fundamental signal of 500 Hz and its 3rd, 5th, 7th and 9th harmonics.

The FIR low-pass filter coefficients previously calculated from the spreadsheet model of this filter are used in the assembler source code file. The executable COFF file for the filter is loaded into the simulator and run. The simulator processes the input signal samples and generates an output file containing the processed values. The values in the output file are then converted to a form suitable for the spreadsheet, imported into the spreadsheet and subsequently displayed. These waveforms are shown in Fig. 4, which indicates the complete low-pass filtering process.

Table 1: Q-15 format equivalent hexadecimal codes

Signal magnitude	Q-15 value	Q-15 4-field hex code
+0·9999695	32767	7FFF
:	:	:
:	00002	0002
:	00001	0001
0·00000000	00000	0000
:	65535	FFFF
:	65534	FFFE
:	65533	FFFD
:	:	:
−1·00000000	32768	8000

Spectral analysis

For processed signals to be properly analysed, their Fourier transform must be determined to give a clear representation of frequency content. The spreadsheet implementation of an FFT is shown by Chapman[5]. We have developed a 256-point FFT using the Lotus 1-2-3 spreadsheet package for spectral analysis of signals. Using a 386 25 MHz PC, this FFT recalculates

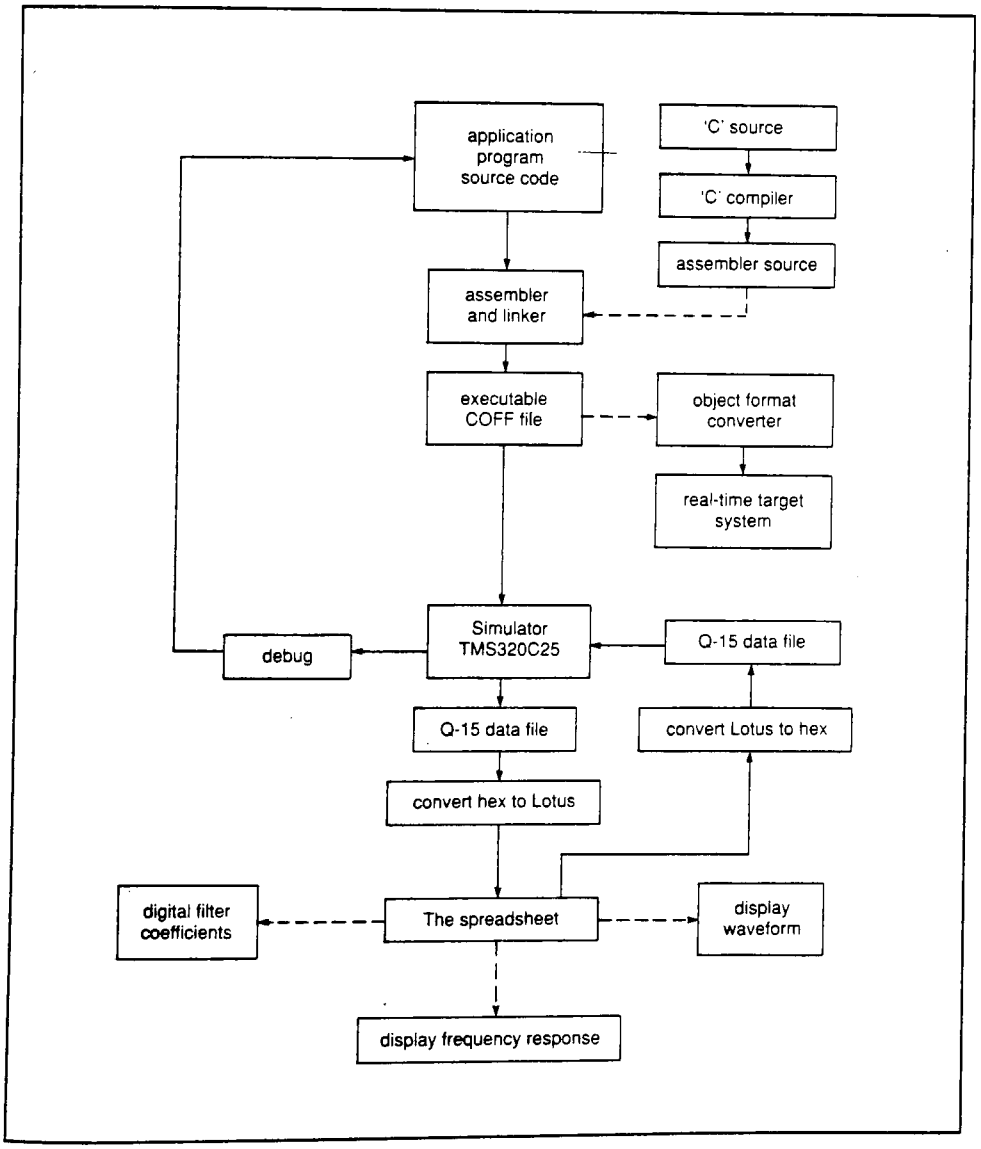


Fig. 3 System block diagram

in approximately 5 seconds. A display of input and output spectra for the FIR filter example is shown in Fig. 5.

The frequency response of any filter network is obtained by taking the Fourier transform of its time domain impulse response. The impulse applied to the input of the FIR filter is generated in the spreadsheet by filling the first cell for the input signal by 1·0000 followed by 0·0000 for all the remaining input cells. Once again the filter program is run on the simulator to generate a time domain impulse response as shown in Fig. 6. The FFT of the impulse response is shown in Fig. 7, which is the magnitude response of the filter (using a rectangular window) and is identical to the magnitude response obtained from the spreadsheet model of the filter shown in Fig. 2.

Real-time processing

For real-time processing of signals a DSP starter kit[12] (DSK) from Texas Instruments is an ideal low-cost hardware target system for testing/debugging and running the program developed for the simulator environment (educational cost of the starter kit is approximately £70). Small modifications are needed to run the programs on the DSK. The oscilloscope output of a full-wave rectifier coded for the DSK is shown in Fig. 8. The oscilloscope output of a double-sided frequency spectrum running a program supplied by Texas Instruments for the DSK is shown in Fig. 9.

The DSK board is connected to a PC using an asynchronous RS-232 serial link. An external 9 V AC transformer is required to power up the DSK. Software supplied with the DSK kit includes an assembler and a debugger. This is not a COFF assembler; however COFF object files can be loaded and run on the DSK. The DSK assembler does not go through a linker process to create an output file; instead, it uses special directives in the source code to generate an absolute address during the assembly process. Small programs can be easily created and run and larger programs can be developed by chaining files together. The DSK debugger uses a window and menu type interface for ease of loading and executing program code. Other

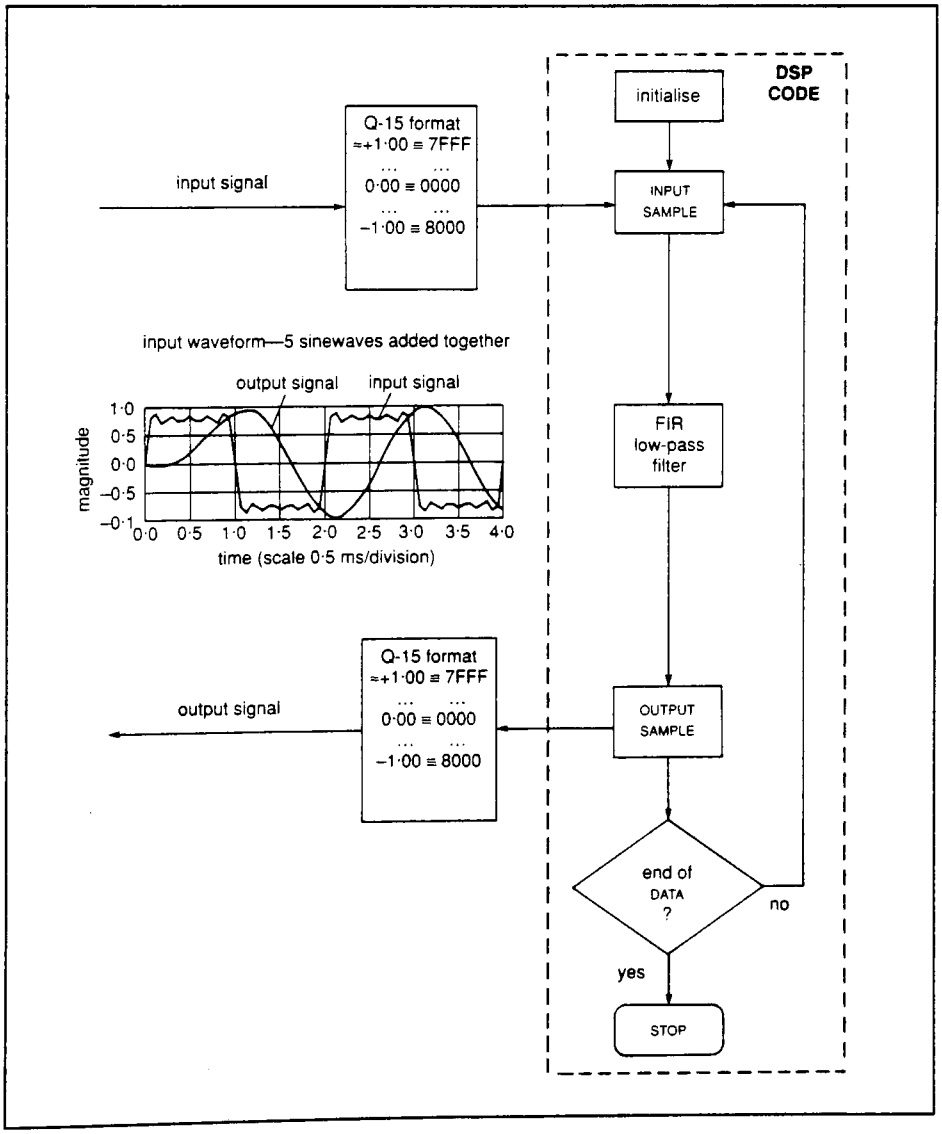
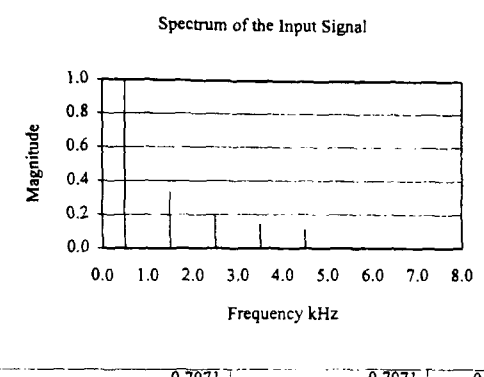


Fig. 4 The low-pass filtering process

	H	I	J	K	L	M	N	O	P	Q
3				Fast Fourier Transform 256 points - Radix 4						
4	Phase Noise Threshold =	1.00E-07								
5	N = 256, n = 0 .. 63 for Stage 1		Stage 1		Stage 2		Stage 3		Stage 4	
6	N = 64, n = 0 .. 15 for Stage 2		Real	Imag	Real	Imag	Real	Imag	Real	Imag
7	N = 16, n = 0 ... 3 for Stage 3									
8										
9	2*PI =	6.2831853072	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
10	PI/180 =	0.0174532925	0.7957	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
11	180/PI =	57.2957795131	0.8876	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
12			0.7103	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
13	COS(2*PI*n/16)	SIN(2*PI*n/16)	0.7789	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
14	0.9239	0.3827	0.8414	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
15	0.7071	0.7071	0.7491	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
16			'652	0.000(						00
17			1349	0.000(						00
18			'652	0.000(						00
19			'491	0.000(						00
20			1414	0.000(						00
21			'789	0.000(						00
22			'103	0.000(						00
23			1876	0.000(						00
24			'957	0.000(						00
25			1000	0.000(						00
26			'957	0.000(						00
27			1876	0.000(						00
28			'103	0.000(						00
29			'789	0.000(						00
30			1414	0.000(						00
31			'491	0.000(						00
32			'652	0.000(						00
33	0.7071	0.7071	-0.8349	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
34	0.6344	0.7730	-0.7652	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
35	0.5556	0.8315	-0.7491	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000



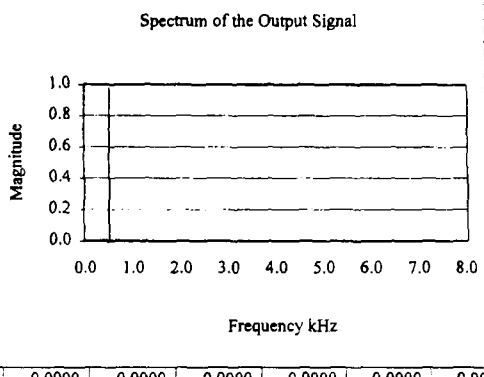


Fig. 5 Spectra of the input and output signals to and from the 21-tap FIR filter

features include single-step, breakpoint and run-time halt capabilities.

Conclusions

A novel yet powerful approach to learning about DSP has been demonstrated in this article. For students in an educational environment and for other professionals interested in getting to grips with DSP systems and their applications, this approach can be very useful given the low cost and flexibility that it offers.

The power of spreadsheets and their application in engineering is once again demonstrated here. Smaller applications can easily be handled by students using the spreadsheet approach, which gives a great deal of confidence by removing the repetitive aspects of problem solving. However, a large spreadsheet model such as the 256-point FFT, although feasible as demonstrated in this article, is unnecessary for students to develop unless this is their main objective. The

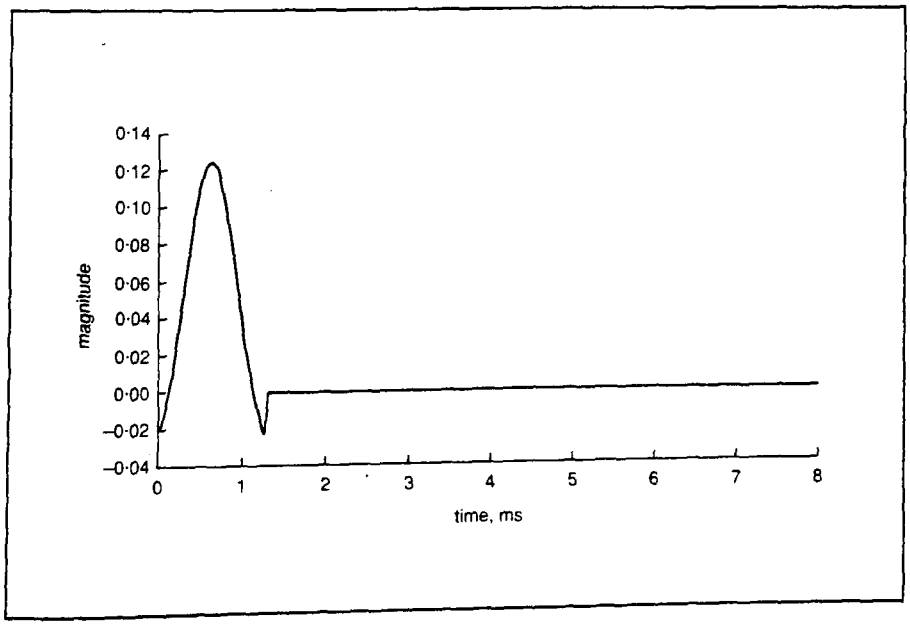


Fig. 6 Time domain impulse response of the 21-tap FIR low-pass filter

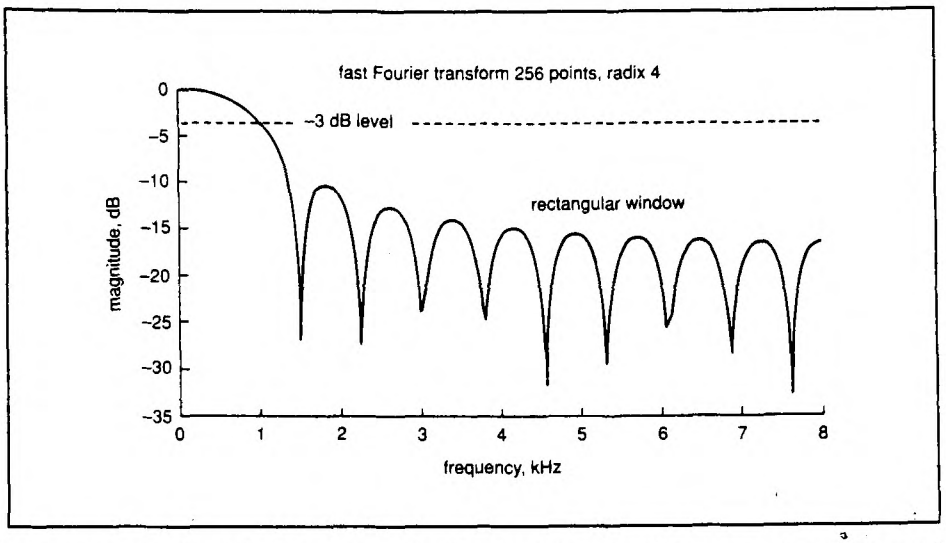


Fig. 7 Spectrum of the 21-tap FIR low-pass filter impulse response

Microsoft Excel for Windows package has a built-in FFT function which may be used directly to analyse signals. It also features decimal-to-hexadecimal and the reversé conversion functions, which make it ideal for this simulation approach.

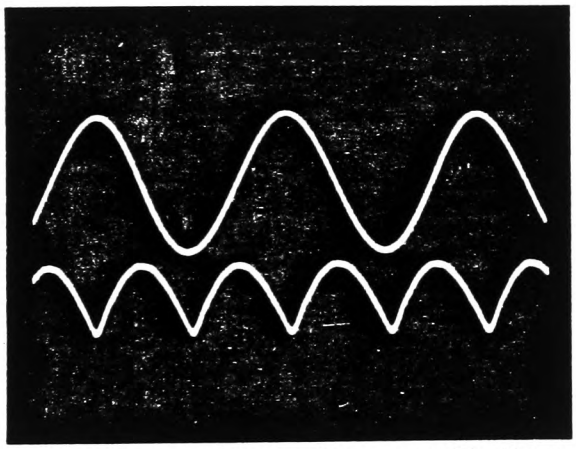


Fig. 8 Oscilloscope traces for a full-wave rectifier. Upper trace—input signal; lower trace—output signal

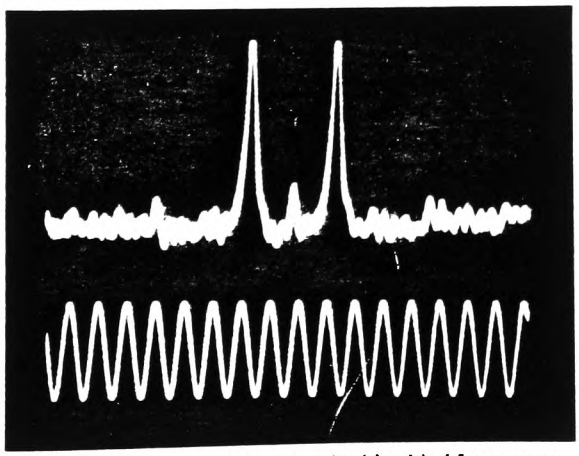


Fig. 9 Oscilloscope traces for a double-sided frequency spectrum. Upper trace—256-point FFT frequency spectrum; lower trace—2 kHz input sinusoid

Finally, the interaction between a spreadsheet and a commercial DSP software package has been demonstrated thereby showing the possibilities that are available and may be exploited to the full by students and professionals alike.

References

1 IEE Electronics Division Colloquium on 'The teaching of digital signal processing in universities', 16th February 1995, Digest No. 1995/036

2 Texas Instruments: 'TMS320C2X C source debugger—user's guide', 1991, document number SPRUO7B

3 STANTON, B. J., DROZDOWSKI, M. J., and DUNCAN, T. S.: 'Using spreadsheets in student exercises for signal and linear systems analysis', *IEEE Trans. on Education*, February 1993, **E-36**, (1), pp.62–68

4 KOLK, W. R., and LERMAN, R. A.: 'Non-linear system dynamics' (Van Nostrand Rheinhold, 1992)

5 CHAPMAN, D. A.: 'Spreadsheet demonstration of discrete and fast Fourier transforms', *Int. J. Electr. Eng. Educ.*, July 1993, **30**, (3), pp.211–215

6 JOHNSON, J. R.: 'Introduction to digital signal processing' (Prentice Hall, 1989)

7 LYNN, P. A.: 'Digital signals, processors and noise' (Macmillan, 1992)

8 GRANT, P. M.: 'Digital signal processing. Part 1: Digital filters and the DFT', *Electron. & Commun. Eng. J.*, February 1993, **5**, (1), pp.13–21

9 GRANT, P. M., and McDONNELL, J. T. E.: 'Digital signal processing. Part 2: Spectral analysis', *Electron. & Commun. Eng. J.*, August 1993, **5**, (4), pp.212–220

10 Texas Instruments: 'TMS320 fixed point DSP assembly language tools—user's guide', 1990, document number SPRUO18B

11 Texas Instruments: 'TMS320C2X—user's guide', 1990, document number SPRUO14B

12 Texas Instruments: 'DSK starter kit', part no. TMDS 3200026

The authors are with the Faculty of Technology, Gwent College of Higher Education, Allt-yr-yn Campus, PO Box 180, Newport, NP9 5XR, UK.