SOFTWARE ANALYSIS OF MUSICAL INSTRUMENT TONES

Clifford S. Elion.

A Dissertation submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg for the degree of Master of Science.

Johannesburg 1983.

SOFTWARE ANALYSIS OF MUSICAL INSTRUMENT TONES

ABSTRACT

Computer based signal processing techniques provide important tools for the research of naturally occurring timbres. A system to digitally record and analyse natural sounds is described. The system interfaces to a large computing resource which supports spectral analysis software - allowing evaluation, and continual upgrading of the analysis techniques. Fourier transform methods are applied to pitch determination and spectral analysis of selected musical instruments. The final output of the analysis is a graphical representation of the harmonic and spectral landscapes of the various instruments.

Declaration

I declare that this dissertation is my own, unaided work. It is being submitted for the degree of Master of Science in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other university.

C.S. ELION

14th day of February, 1983.

Acknowledgements

I would like to express my gratitude to Mr. J.M. Van Coller for accepting the responsibility of supervising this research, for his interest and sound advice. I am also indebted to Professor H. Hanrahan who is responsible for the core of the graphics and signal processing software, and to the postgraduate students in the Electrical Engineering department during 1982 for many enlightening discussions. Finally I would like to thank my parents, Allan and Joan Elion for moral and financial support throughout my university career, and Chasey Agnew for her patience, inspiration and good cooking.

Preface

Music is undoubtedly one of man's oldest activities. Today in its various forms it is one of his most common activities.

Music production and reproduction has been influenced throughout history by the state of technology. Modern digital computers provide a new and fertile ground for the research and production of musical sounds.

The complex nature of musical tones has been considered by the likes of Pythagoras and Galileo, but only with the production of the first electronic music synthesizers was the inadequacy of simple waveforms in the generation of synthesized tones appreciated.

It is apparent that the prerequisite for a music synthesis system is an understanding of the nature and perception of musical tones. The following text describes the development and comission of a musical tone analysis environment, and its use in obtaining information regarding the structure of musical tones and their related *perceptual effects* - the goal being to synthesize new sounds with waveforms based on those of natural instrument sounds.

> "a single intoned sound is meaningless in the sense of a *musical investigation* when, after onset or any other change, it becomes stationary or reaches a value which remains unchanged in time...
> ...in the laws of nature there is a relationship between the stationary sound structure and that of sound movement, or *the onset process.*"
>
> Fritz Winkel - on the dynamic nature of musical tones.

List of Abbreviations

ACIA - Asynchronous Communications Interface Adaptor
ADC - Analog to Digital Converter
CLI - Command Line Interpreter
CPU - Central Processing Unit
CV - Control Voltage
DASMA - Data Acquisition System for Musical tone Analysis
DRAM - Dynamic Random Access Memory
DMA - Direct Memory Access
I/O - Input/Output
OEPS - Open Ended Problem Solver
PDL - Program Description Language
VCA - Voltage Controlled Amplifier
VCF - Voltage Controlled Filter
VCO - Voltage Controlled Oscillator

Table of Contents

APPENDIX A: DASHA Users Manual
APPENDIX B: Acoustic Characteristics of Selected Resonant Bodies
APPENDIX C: Analysis Software
APPENDIX D: Program Description Language
APPENDIX E: System Software FORTH Listing
APPENDIX F: DASHA Circuit diagrams

List of Illustrations

Presentation of material

Chapter 1 introduces the historical background of electronic music synthesis and looks at some modern synthesis techniques.

Chapter 2 evaluates the requirements of music analysis and synthesis systems by an examination of the perceptual characteristics of the human auditory system. Two mathematical models for the synthesis of musical tones are presented.

Chapter 3 details the evolution of a design specification and the subsequent design of an audio frequency data capture and analysis system for musical instrument tone analysis.

Chapter 4 introduces the BASIC programming language. The design of the test, operating and maintenance software is flowcharted and presented in a program description language.

Chapter 5 discusses the application of fast Fourier transform techniques to selected musical tones. The results are discussed and presented in graphical form.

Chapter 6 evaluates the analysis environment and provides suggestions for future work in the field of electronic music analysis and synthesis.

CHAPTER 1

INTRODUCTION

This chapter introduces the historical background of electronic music synthesis and looks at some modern synthesis techniques.

1.1 A Brief History of Music Synthesis

Research into the generation of musical sounds is not new. Early musical instruments were developed by changing the lengths of strings, changing the sizes and shapes of soundboards and by changing the material of the component parts. Evaluation of the sounds generated by these instruments was purely by ear.

By the end of the nineteenth century, the development of acoustic instruments had leveled off, and the instruments of the orchestra were firmly established in the concert halls and music chambers of the world.

A radically new instrument was introduced in 1906 when the Telharmonium was installed in the Telharmonic Hall in New York. The Telharmonium used more than one hundred alternators (each capable of generating as much as 15 kilowatts of power) to generate music which was transmitted via telephone lines to subscribers. Electronic music had arrived.

The invention of the vacuum tube led to smaller and more usable electronic music synthesizers. One instrument that attracted much interest during the 1920s was the Theremin. The Theremin departed from convention in its user interface - instead of being played via a keyboard, it was played by moving the hands in the space surrounding the instrument.

The Hammond organ developed in the 1930s used the same principle of sound generation as the Telharmonium; however instead of large mechanical generators, tone wheels induced waveforms in adjacent coils.

The Solovox developed by the Hammond company was the first instrument to use modern synthesis techniques. It used a single tone generator and an assortment of waveshaping, filtering, frequency dividing and envelope-shaping circuits.

Early synthesizers, although innovative, produced unnatural sounds and were thus delegated to providing weird sounds for special effects and for background music in films.

Largely as a result of Dr. Robert Moog, whose name has become synonymous with synthesizers, second generation synthesizers appeared in the late 1960s. These synthesizers were modular in construction, and more important, the modules were controlled by a common parameter: voltage.

1.1.1 Modern Techniques of Music Synthesis

The modules and mechanisms of second-generation synthesizers have dictated the evolution of modern music synthesis techniques. These modules fall into three categories (figure 1.1):

1. Waveform generation - modules consisting of various waveform oscillators whose frequencies are controlled by an input voltage, and noise generators for percussive and wind type sounds.

2. Waveform modification - filters, amplifiers and mixers to modify the harmonic content of the waveforms - under voltage control.

3. Waveform control - modulation hardware to give vibrato and tremolo effects.

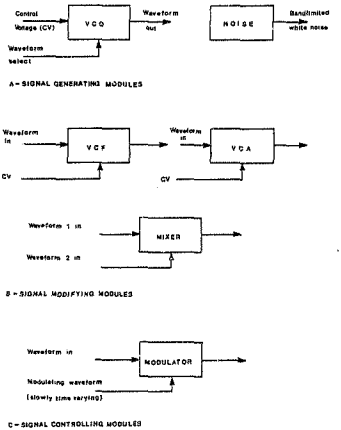


Figure 1.1 : Electronic Music Synthesis modules.

Second-generation synthesizers incorporated an assortment of these modules interconnected with patch cords somewhat like an old fashioned telephone switchboard. Control signals and audio signals were carried on two distinct signal paths. The control path carried signals corresponding to the changes in the parameters of the signal on the audio path. An example of voltage controlled synthesis is illustrated by figure 1.2. The control voltage (CV) controls a VCO as well as a VCF on the same audio signal path. A variation in the control voltage causes a change in generated frequency and a proportional change in the frequency band of the VCF. The output audio signal thus varies in frequency while maintaining a constant harmonic envelope. This is perceived as a varying pitch of consistent tone.

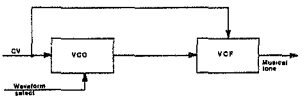


Figure 1.2 : Typical signal patch path in a voltage controlled synthesizer.

Signal patching allowed the maximum use of the resources of a few synthesis elements. It was difficult, however, to repeat intricate patch patterns and level settings with absolute accuracy - setting up a particular sound required time-consuming iterative methods. Users were therefore restricted to a small subset of the sounds when different patches were required in rapid succession.

The development of synthesizers with internally wired signal paths provided a partial solution to these restrictions. The modules in these instruments were wired together to provide the most versatile signal paths. Some had provision for limited variation of the patches. The result was a more usable instrument with reduced versatility, but increased repeatability.

The late 19/Os saw the introduction of digital technology in electronic music analysis and synthesis - initially only in the form of storage and control systems for analog synthesizers. The control parameters were stored in digital memory, alleviating the tedium of repeated signal patching and parameter setting. Once a particular sound was found, the musician could assign it to a switch or button which, when activated, would "program" the synthesizer with the parameters stored in the memory addressed by the switch.

Digital computer technology is only beginning to influence electronic music synthesis. More recent synthesizers range from analog peripherals for the Apple microcomputer to dedicated sixteen-bit microprocessor-based systems employing advanced digital signal processing techniques.

1.2 Electronic Music Synthesis Techniques

1.2.1 Terminology

As an introduction to the techniques and terminology used in modern music research, it is useful to classify the various techniques of electronic music analysis and synthesis. This terminology will be used throughout this dissertation.

1. Synthesis-elements - the waveform generation, modification and controlling modules used in an electronic music synthesizer.

2. Tone-description parameters - the values of the functions used to control the timbres generated by the synthesis-elements.

3. Parameter-definition - the method used to derive the tone-description parameters.

4. Synthesis-technique - the method of using the tone-description parameters to control the synthesis-elements.

5. Technology - the Analog or Digital implementation of the synthesis-technique and parameter-definition.

The technology associated with the various synthesis techniques is given in table 1.1.

PARAMETER DEFINITION	SYNTHESIS TECHNIQUE	TECHNOLOGY
Direct	Subtractive	Analog
Analysis-based	Additive	Digital
Analysis-based	Subtractive	Digital
Direct	Waveshaping	Digital
Direct	Musique Concrete	Digital/Analog

Table 1.1: Common Synthesis Techniques and their implementations.

Most synthesizers use a combination of two or more of these techniques. Subtractive synthesis schemes, for example, often employ waveshaping techniques such as frequency modulation.

1.2.2 Parameter Definition

1.2.2.1 Direct Synthesis - Direct synthesis is probably the oldest method of parameter definition, and has been the most widely used commercial technique over the last decade. With direct synthesis the synthesist intuitively controls the tone-description parameters.

Direct synthesis can be implemented on second generation analog synthesizers. The synthesist iteratively sets up the sound he desires by defining signal paths with patch cords and parameter settings on potentiometers and switches. Direct synthesis has also been implemented on digital computers. Several special purpose computer languages have been developed to implement the parallel processes of music synthesis.

These music synthesis languages define a series of computational modules, each representing a synthesis element. The modules are linked together (patched) to produce the output waveform. A "piece" of music is written as a program consisting of two parts:

1. Instrument definition - the various computational modules necessary to describe the sampled data waveform of the sound.

2. Event list - the sequence of musical events, their starting times, durations, and pitches.

These languages are useful for developing and evaluating signal paths which may later be either hard-wired or implemented on dedicated computer music systems. The languages are not limited to direct synthesis, in fact the computational modules may be fed with tone-description parameters derived from musical tone analysis.

1.2.2.2 Analysis-Based Synthesis - Analysis-based synthesis relies on the extraction of the tone-description parameters from the waveforms generated by an acoustic or electro-acoustic musical instrument. Digital computers are invaluable in the analysis of these musical tones. Many computationally efficient digital signal processing algorithms may be used to analyse musical sounds.

The aim of analysis-based synthesis systems is to create new musical sounds with the timbral character of natural instruments. Natural instrument sounds are thus a subset of the tones available on an analysis-synthesis system. The analysis system can be evaluated by the closeness of the synthesized sound to the original.

1.2.2.3 Musique Concrete - Musique concrete differs from both analysis-based and direct synthesis. No waveform generation takes place; instead, signal processing techniques are used to modify the sounds generated by musical instruments.

Musique concrete originated from the use of tape recorders to generate echo, phase shifting, flanging and chorus effects. The tape recorder in modern musique concrete has been replaced by digital computing systems which provide greater flexibility. Manipulation such as changing the pitch of a sound without changing

its duration, and changing the duration of a sound without changing its pitch are possible on digital systems.

1.2.3 Synthesis Techniques

1.2.3.1 Additive Synthesis - Involves the synthesis of a complex waveform by the addition of its component parts. Fourier Synthesis is the special case of additive synthesis where the component parts are the spectral components (sinusoids) of the synthesized waveform.

1.2.3.2 Subtractive Synthesis - The resultant waveform is generated by filtering out unwanted portions of a harmonically rich waveform.

1.2.3.3 Waveshaping Synthesis - Mathematical translations and relations make it computationally efficient to produce complex waveforms with time varying harmonic content. Le Brun [1] provides a comprehensive analysis of the existing techniques including Chowning's frequency modulation, see Chowning [2]. Although analysis-based waveform synthesis techniques are virtually unexplored, direct methods have been used to match the steady state harmonic spectra of many wind instruments.

CHAPTER 2

ANALYSIS-BASED MUSIC SYNTHESIS

The requirements of music analysis and synthesis systems are evaluated by an examination of the perceptual characteristics of the human auditory system. Two models for analysis-based music synthesis are presented.

2.1 The Perception of Musical Sound

The perception of musical tone is dependant on three parameters: pitch, loudness and timbre.

Pitch and loudness are readily understood. Timbre is best explained as the cumulative effect of everything besides pitch and loudness (the American Standards Association (1960) defines timbre as "that attribute of auditory sensation in terms of which a listener can judge that two sounds similarly presented and having the same loudness and pitch are dissimilar").

A music score conveys information to the musician regarding all three of these parameters:

1. Pitch - by the location of the notes on the staff,
2. Loudness - by expressions such as Pianissimo and Fortissimo, and
3. Timbre - by expressions such as Agitato, Brilliante and Legato which describe an emotion that the piece is to convey - usually interpreted into the timbral qualities of the sounds.

Acoustic musical instruments provide physical contact between the musician and the acoustic mechanisms of the instrument - allowing experienced musicians to convey the emotive qualities of music by adding timbral-altering effects such as damping and vibrato.

Electronic Music Synthesizers are known for their lack of timbral control. New sounds soon become boring and notably synthetic, due to their timbral rigidity.
Three levels of timbral control are currently implemented on keyboard controlled synthesizers:

1. At the lowest level, the timbre is unvarying. More advanced techniques may base the rigid timbres on the characteristics of an acoustic instrument.

2. More interesting sounds are generated when some form of random or linear interpolation is used to vary the timbres of individual notes.

3. State-of-the-art experimental keyboards use two-dimensional velocity-sensing keys and keys capacitively coupled to the striking object (finger) to extract as much information as possible from the manner in which the key is struck. This information is used to control the timbral variations of the sound, enhancing the degree of communication between the musician and the instrument - allowing emotive timbral control through this interface.

Musical tone analysis is complicated by the perceptual inconsistencies of the human senses. To understand the effects of acoustic variations on the human auditory system, and to determine the extent to which data describing these variations need be specified, an examination of the perceptual characteristics of this complex hearing mechanism is presented in this chapter.

It should be noted that analytic descriptions of musical stimuli rely on perception. It is therefore not surprising that an enormous amount of experimentation is required to determine which perceptual data are reliable. Most experimental techniques are based on the presentation of musical tones of known characteristics to a jury of listeners ranging from the naive to the musically proficient, and evaluating their descriptions of the tones.

2.1.1 The Perception of Pitch

The pitch of a complex tone is always perceived as that of a pure tone at the frequency of the fundamental (which may not necessarily exist in the tone itself). The pitch perceived when the fundamental is not present in the complex tone is known as residue pitch.

Residue pitch has been extensively although not conclusively investigated. The classical theory is that pitch is attributed to the high relative loudness of the fundamental which may either exist in the tone itself or be introduced by non-linear effects of the ear. Contemporary theory is that the ear detects pitch from the periodicity of the soundwave.

After providing evidence that the non-linear distortion of the ear is too small to induce a perceived pitch corresponding to the missing fundamental, Plomp [3] conducted experiments with two complex stimuli presented successively. The second represented a 10% decrease in fundamental frequency and a 10% increase in the frequency of the second and higher harmonics. Observing that the 10% harmonic increase influenced the perceived pitch more than the 10% fundamental decrease, Plomp concluded that the pitch of complex tones is based on the periodicity of the soundwave, rather than the frequency of the "fundamental" component of the wave.

Many pitch-detection algorithms rely on the periodicity of the complex tone to estimate its pitch. Time-domain techniques analyse the periodicity of the time waveform itself, while in the frequency domain the most successful techniques estimate the pitch by analysing the periodicity of the frequency spectrum.

2.1.2 Consonance, Dissonance and Critical Bandwidth

Pythagoras is said to have been the originator of the theory of musical harmony. He discovered that tones produced by the vibration of strings with the integer length ratios 1:1, 1:2, 2:3 and 3:4 produced more appealing harmonies than other ratios. These intervals are called consonances. Other ratios, 4:5, 3:5, 5:6 and 5:8 are now included as "imperfect consonances" (It is on the character of the tones and harmonies generated by these intervals that western music has developed).

Plomp and Levelt [4] define consonance as "the peculiar sensorial experience associated with isolated tone pairs having simple frequency ratios".

Terhardt [5] defines consonance as the "undisturbed simultaneous sounding of pure tones", and defines the disturbing element which destroys consonance as roughness. Roughness is associated with the beats present when two pure tones of close frequency are sounded together. Terhardt claims that the consonance referred to by Plomp et al is governed by the frequency difference (critical bandwidth) rather than on the frequency ratio of pure tone pairs.

The consonance of complex tones sounded together is dependant upon the number of partials (see page 2-8) of the tones producing beat frequencies. This number is minimised when the fundamental frequencies of the complex tones are related by a ratio of small integers.

Ohm's acoustical law describes the ability of the human ear to decompose a complex tone into its sinusoidal components. To ascertain the number of distinguishable partials Plomp [6] conducted experiments using multitone signals, and showed that partials could be distinguished only if their frequency separation exceeds a critical bandwidth. For frequencies above 500 Hz a critical bandwidth is about 1/4 octave.

Pierce [7] used digital computer techniques to generate an experimental musical scale consisting of eight tones with a frequency ratio r, such that $\log(r)=1/8$ (the equally tempered music scale has r, such that $\log(r)=1/12$). He found that in accordance with the theory of critical bandwidth, the tones generated by partials separated by at least 1/4 octave sounded more consonant than tones generated by partials separated by 1/8 octave. Pierce concluded that "by providing music with tones that have accurately specified but nonharmonic partial structures, the digital computer can release western music from the tyranny of 12 tones without throwing consonance overboard".

2.1.3 Effect of Phase on Timbre

Plomp and Steeneken [8] found that the maximum effect of phase on timbre is smaller than a change of 2 dB of sound pressure level, and is less for high than for low frequencies. Their results also

Plomp and Levelt [4] define consonance as "the peculiar sensorial experience associated with isolated tone pairs having simple frequency ratios".

Terhardt [5] defines consonance as the "undisturbed simultaneous sounding of pure tones", and defines the disturbing element which destroys consonance as roughness. Roughness is associated with the beats present when two pure tones of close frequency are sounded together. Terhardt claims that the consonance referred to by Plomp et al is governed by the frequency difference (critical bandwidth) rather than on the frequency ratio of pure tone pairs.

The consonance of complex tones sounded together is dependant upon the number of partials (see page 2-8) of the tones producing beat frequencies. This number is minimised when the fundamental frequencies of the complex tones are related by a ratio of small integers.

Ohm's acoustical law describes the ability of the human ear to decompose a complex tone into its sinusoidal components. To ascertain the number of distinguishable partials Plomp [6] conducted experiments using multitone signals, and showed that partials could be distinguished only if their frequency separation exceeds a critical bandwidth. For frequencies above 500 Hz a critical bandwidth is about 1/4 octave.

Pierce [7] used digital computer techniques to generate an experimental musical scale consisting of eight tones with a frequency ratio r, such that $\log(r)=1/8$ (the equally tempered music scale has r, such that $\log(r)=1/12$). He found that in accordance with the theory of critical bandwidth, the tones generated by partials separated by at least 1/4 octave sounded more consonant than tones generated by partials separated by 1/8 octave. Pierce concluded that "by providing music with tones that have accurately specified but nonharmonic partial structures, the digital computer can release western music from the tyranny of 12 tones without throwing consonance overboard".

2.1.3 Effect of Phase on Timbre

Plomp and Steeneken [8] found that the maximum effect of phase on timbre is smaller than a change of 2 dB of sound pressure level, and is less for high than for low frequencies. Their results also

indicated that the effect of phase on timbre is independant of the harmonic amplitude pattern and the loudness.

2.1.4 Effect of Harmonic Variations on Timbre

The question of the effect of the temporal evolution and cessation of harmonic structures in the recognition of musical instrument timbre was investigated by Berger [9]. Several wind instrument tones were recorded and played back, both unaltered and backward, with the attack and delay portions either removed or filtered. He concluded that the attack and release of a waveform seem to give the *listener important clues to the recognition of the instrument.*

On the basis of earlier work on the perception of timbre, Grey [10] used a method of analysis-based additive synthesis in a series of experiments to *determine a measure of both the discriminability and* perceptual distance between tones produced by analysis-synthesis or data-reduced synthesis techniques and the original tones upon which they were based.

Time-varying amplitude and frequency functions were obtained for each harmonic of a complex tone. The tone was then synthesized by summing a set of harmonic sinusoids controlled in time by the analysed functions.

The most relevant investigation carried out by Grey was on the extent to which data reduction techniques could be applied to the massive amount of data describing a single musical instrument tone. He synthesized four versions of the original tones using

1. Complex-synthesis - where the complex amplitude and frequency functions resulting from the analysis were used to control the parameters of a set of sinusoids.

2. Line-segment approximation - where four to eight line-segments per function were used to model the complex functions.

3. Cut-attack approximation - where a line-segment aproximation as above was used, with initial low amplitude segments in the attack excluded.

4. Constant-frequencies approximation - where a line-segment approximation was used with constant frequencies

substituted for the time-variant frequency functions.

The perceptual studies revealed that the extent of the data-reduction and the nature of the approximation (line-segment, cut-attack or constant-frequencies) have different, discriminable effects on the perceptual distance between the synthesized waveforms and the originals. The most successful data-reduction technique was the line-segment approximation in both the time and frequency domain. The cut-attack approximation was judged most discriminable. The constant frequency approximation, which is used in most additive synthesis schemes, was found to be very discriminable in several cases.

Mathews and Pierce [11] could not conclusively decide whether the perception of harmonic effects resulted from

1. Residue Pitch,
2. Critical Bandwidth, or
3. Cultural Brainwashing.

The perception of musical tones is clearly a complex phenomenon - an understanding of which is necessary as a starting point for the development of musical tone synthesis systems capable of timbrally rich sounds.
The following points are considered important:

1. Tones separated by a critical bandwidth are perceived as consonant.
2. The pitch of complex tones results from the effect of the periodicity of the soundwave and not from any frequency present in the soundwave.
3. The relative phases of the harmonics of a complex tone have a small effect on the perception of the tone.
4. The attack portion of a natural musical instrument tone is very important for the recognition of the sound.
5. Line-segment approximations can be applied as data-reduction techniques to both the frequency and amplitude description functions of the complex musical tone.

2.2 Analysis-Based Synthesis Models

Associated with the acoustic waveforms of most musical instruments are frequency spectra exhibiting partial frequencies at near-integer multiples of the fundamental frequency. It is often convenient to refer to these partials as harmonics even though they do not always occur at exactly integer multiples of the fundamental.

In the following text the term partial refers to activity at unconstrained frequencies, and harmonic is used to describe partials at frequencies which are near-integer multiples of the fundamental.

This type of harmonic generation is characteristic of the vibration modes of reeds, organ pipes and stretched strings (appendix B).

Musical tones exhibiting harmonic spectra are termed voiced, while percussive instruments such as bells and drums which are characterized by inharmonic spectra are termed unvoiced.

2.2.1 The Subtractive Synthesis Model

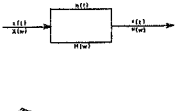
Acoustic musical instruments exhibit properties similar to those of the speech production mechanism. On the basis of these similarities, the mathematical equivalent developed for the production of synthetic speech can easily be applied to musical tone synthesis.

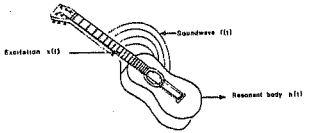
Consider the physical structure of any musical instrument. The instrument has two important components.

1. A source of mechanical vibration - reed, string, skin etc.

2. A resonant body - bowl, drum, tube etc.

The source of mechanical vibration produces an excitation waveform which is acoustically coupled to the resonant body. The acoustic circuit of the instrument can be represented by the electrical analog of figure 2.1.





ELECTRIC CIRCUIT ANALOGUE OF MUSICAL INSTRUMENT

Figure 2.1 Analogy between electric circuit frequency response and musical instrument resonance.
x(t)=excitation waveform, h(t)=impulse response of resonant body, f(t)=output signal (musical tone), X(w)=frequency spectrum of input (excitation waveform), H(w)=transfer function (resonances of instrument), F(w)=frequency spectrum of output (acoustic waveform).

In accordance with the electrical analogy, the resonances of the musical instrument's body cause enhancements or depressions in the frequency spectrum associated with the excitation waveform. The enhanced (resonant) frequency bands form spectral "humps" (figure 2.2) which are known as formant regions.

If the formant regions are represented by a cascade of bandpass filters, the formants occur where the centre frequencies of the bandpass filters would lie.

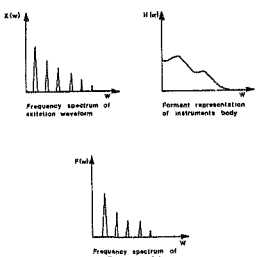


Figure 2.2: Spectrum of excitation waveform and formants associated with musical instrument tones.

The parameters of the acoustic wave vary slowly with time so that for a short section of the wave the parameters can be approximated as stationary. The musical instrument tone f(t) can thus be viewed as the time convolution of the excitation waveform x(t) with the impulse response h(t) of the instruments resonant body.

$$f(t)=x(t)*h(t) \qquad \text{--------(1)}$$

And in terms of the Fourier Transform.

$$F(w)=X(w).H(w) \qquad \text{--------(2)}$$

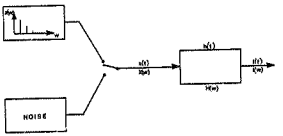


Figure 2.3: Model for subtractive synthesis of musical instrument tones.

Figure 2.3 shows the model for subtractive synthesis of musical instrument sounds. The excitation waveform x(t) can be either

1. a harmonically rich waveform having spectral components at every partial frequency for voiced sounds, or
2. bandlimited white noise for unvoiced sounds.

X(w) is fed into a formant filter which models the frequency response H(w) of the instrument. The formant filter dictates the harmonic content of the synthesized waveform by "carving" out the unwanted harmonics of X(w). This removal of unwanted harmonics gives the technique the name subtractive synthesis. Natural timbres are achieved when H(w) and X(w) are time varying functions controlled by the tone-description parameters resulting from a musical tone analysis.

2.2.2 Pitch and Formant Detection

Many algorithms have been proposed - the most widely used being cepstrum pitch detection. The cepstrum technique is particularly useful for the subtractive synthesis model as it provides pitch, formant and voiced/unvoiced information.

2.2.3 Cepstrum Pitch Determination

Applying Discrete Fourier Transform analysis to a finite record of the digitized musical instrument tone f(t) of period T, the square of the magnitude of the Fourier Transform is the Power Spectrum $|F(w)|^2$ consisting of harmonics spaced at 1/T Hz. The Power Spectrum is thus periodic and equal to:

$$|F(w)|^2 = |X(w)|^2 . |H(w)|^2 \quad \text{--------(3)}$$

The time domain convolution of the excitation waveform and the impulse response of the resonant body of the instrument has now been converted to a multiplication in the frequency domain.

For the purpose of analysis, it is convenient to separate these two parameters. Taking the logarithm:

$$\log |F(w)|^2 = \log [|X(w)|^2 . |H(w)|^2] \quad \text{--(4)}$$

$$= \log [|X(w)|^2] + \log [|H(w)|^2] \quad \text{--(5)}$$

The significance of this is that the time domain convolution has been transformed into a linear combination of the two expressions (addition). This is illustrated in Figure 2.4(a). Deconvolution of waveforms in this manner is known as Homomorphic filtering and is described by Oppenheim et al [12] and specifically for the speech waveform by Oppenheim and Schafer [13].

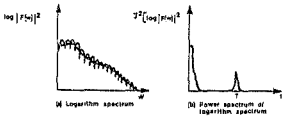


Figure 2.4: Homomorphic processing of slowly time varying acoustic waveforms.

The effect of the frequency response of the instrument's body is to produce the low frequency ripple in the logarithm spectrum, the "humps" of which are the formants. The high frequency ripple superimposed upon the frequency response is the logarithmic power spectrum of the excitation waveform - it exhibits a strong periodicity. The Fourier transform of the logarithm power spectrum

shows a peak corresponding to the periodicity of these high frequency ripples as well as activity related to the low frequency ripple (Figure 2.4(b)).

This representation is known as the cepstrum. Noll [14] and Childers et al [15] provide detailed discussions and applications information on this technique. Cepstral quantities exist in the time domain with similar properties to spectral quantities in the frequency domain. The names given to the cepstral quantities are derived from those of the parallel spectral quantities:

spectrum	-------	cepstrum
frequency	-------	quefrency
phase	-------	saphe
amplitude	-------	gamnitude
filtering	-------	liftering
harmonic	-------	rahmonic
period	-------	repiod

Cepstrum and quefrency are, however, the only terms in common usage.

Cepstral analysis provides the parameters for the subtractive synthesis model. The peak at T in figure 2.4(b) shows activity at a quefrency of T seconds, this corresponds to the high frequency ripple and is the pitch-period of the excitation waveform. The low time activity is due to the frequency response of the instrument. The formants are obtained by short-pass liftering (low-pass filtering) the logarithm power spectrum.

Zero high-quefrency activity indicates a non-periodic spectrum and thus an unvoiced excitation waveform.

2.2.4 The Additive Synthesis Model

Consider again the musical instrument tone f(t). This signal can be expressed in terms of any set of complete orthogonal functions, $\phi(t)$,such that:

$$f(t)=\sum_{k} f_k \phi_k(t) \qquad \text{--------(6)}$$

where: f_k: is the weighting factor of the function $\phi_k(t)$

f(t) can thus be represented by a summation of signals $\phi_k(t)$ of amplitudes f_k. This is the basis of the additive synthesis models.

A wide variety of orthogonal functions are available, such as exponential functions, trigonometric functions and Walsh functions.

Two find immediate application:

1. Exponential functions (Fourier) and
2. Walsh functions

Walsh functions are generated from a set of non-uniform-duty-cycle rectangular functions taking on the values +1 and -1. The periodicity of Walsh functions is measured by the average number of zero crossings per second (zps) and is known as sequency.

Walsh transforms are computationally more efficient than their Fourier counterpart due to the binary nature of the Walsh functions (sine and cosine functions are replaced by +1 and -1).

However it is useful to develop the additive synthesis model using Fourier transform methods because frequency is conceptually easily understood. This does not prohibit the use of other techniques to implement the sound analysis and synthesis. A hypothetical analysis-synthesis system could perform all analysis and synthesis computations using Walsh transforms and, when user interaction is required, represent the sequency information in terms of frequency through the use of Walsh/Fourier conversions.

Analysis-Based Synthesis Models

Moorer [16] proposes a versatile model for additive synthesis using frequency domain techniques.

$$F(n)=\sum_{k=1}^{M} A_k(n)\sin\{nT[kw+2\pi D_k(n)]\} \quad --(7)$$

Where:

F(n): Value at time nT

n : sample number.

T : Time between consecutive samples (Sampling period).

w : Fundamental frequency of tone (radians).

k : Harmonic (partial) number.

M : Number of partials.

A (n) : Amplitude of partial k at time nT (slowly time varying).

D (n) : Frequency deviation of harmonic k at time nT (slowly time varying).

Frequency analysis methods are applied to a digitized natural instrument tone F(n) to determine the slowly time-varying partial frequencies $[kw+2\pi D_k(n)]$ and their amplitudes $A_k(n)$. These functions can then be approximated by data reduction techniques and used to drive sound-generation hardware similar to that shown in figure 2.5.

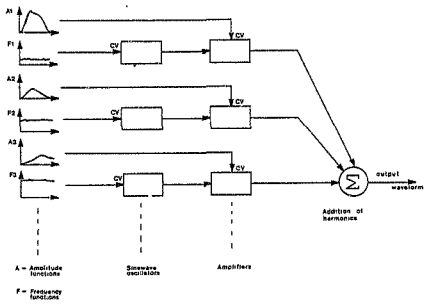


Figure 2.5: Additive synthesis model for the generation of complex tones using spectral components (sinewaves). Each partial is controlled in amplitude and frequency by the tone-description parameters (Ak and Fk, where k=harmonic number) derived from a spectral analysis.

Frequency domain methods allow the musician/synthesist to vary the timbral characteristics of natural sounds and intuitively to generate new musical tones with the perceptual qualities of natural musical instruments.

CHAPTER 3

DEVELOPING A MUSICAL TONE ANALYSIS SYSTEM

A system capable of audio frequency data capture and analysis is required for the proposed examination of the characteristics of musical instrument tones. In this chapter the evolution of a design specification and the subsequent design of an audio frequency data capture and analysis system for musical instrument tone analysis is presented.

3.1 Evolution of a Specification

This dissertation is concerned with discrete-time models of musical tones - the broad term objectives being:

1. to analyse musical instrument tones using digital signal processing techniques,

2. to obtain a quantitative representation of the characterizing parameters of the musical instrument tones, and

3. to create an environment for evaluating different analysis techniques.

Electronic music is only beginning to exhibit the influence of digital signal processing techniques in the analysis and synthesis of musical tones; therefore, rather than a dedicated musical instrument tone analysis system, a general purpose data analysis system constitutes an ideal environment for developing music analysis and related digital signal processing strategies.

A system satisfying these requirements would:

1. digitize audio frequencies up to about 18 KHz,

2. subsequently process these signals with user-programmable digital signal processing techniques,

3. allow interactive programming of analysis algorithms for their evaluation.

These requirements can be met by using a large computing resource in conjunction with an audio frequency data capture facility. The design of this data acquisition system is presented here.

3.1.1 Resources

A large library of digital signal processing software has been developed in the department as part of a larger "Open Ended Problem Solver" (OEPS) package. (see Hanrahan [17]). Consultation with the author of the package led to the realisation that the signal processing software available on OEPS and the structure of the package made it a powerful environment for the analysis of musical tones. All that was needed was a data acquisition system interfacing it to the analog environment. An ANALOGIC MP6912 12 bit linear Pulse Code Modulation (PCM) ADC was made available for this purpose.

OEPS resides on the departmental Data General Eclipse S/140 minicomputer and is written in BASIC. The task was to interface the ADC to the ECLIPSE, to establish a general purpose data capture system for the OEPS software - primarily for musical instrument tone analysis.

The ECLIPSE, being a multi-user system and often under considerable demand, was not to be burdened with the overhead of controlling the data capture system. The hardware was thus designed to operate independantly of the host computer - as a stand alone unit capable of the necessary data manipulation prior to the data being transferred to the host computer.

3.1.2 Data Storage

Storage of any significant amount of digitized audio waveform requires a large digital memory. State-of-the-art 64 Kilobit (Kbit) Dynamic Random Access Memory (DRAM) provides a compact and economic solution to large memory arrays; however, the use of these devices places certain constraints on system design. Unlike static RAMS which can be written into and read from with minimal control of the read/write process, DRAMs require accurate timing waveforms during the read/write and subsequent refresh cycles to ensure data retention.

3.2 Data Acquisition System: Functional Specification

For purposes of abbreviation the Data Acquisition System is referred to as DASMA (Data Acquisition System for Musical tone Analysis (Figure 3.1)).

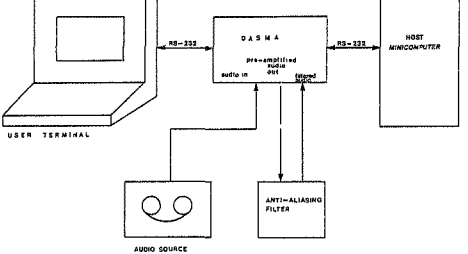


Figure 3.1: Functional representation of the data acquisition system environment.

The system functions can be summarised as follows:

1. DASMA communicates with the user and the host minicomputer over two separate serial data links.

2. Data transfer to the host is under a high level but simple protocol so that DASMA may readily communicate with other host computers besides the ECL SE.

3. Host-dependant activities such as direct memory access or interrupting are avoided by including a large temporary data storage facility in the data acquisition hardware. This memory is expandable in 64 kilobyte blocks. - The digitized data is stored in this memory and then transferred to the host under full program control.

4. The software creates a user-friendly environment through a command line interpreter, allowing user entry of simple commands which set the various functional modes.

5. Programmable functions include the data link baudrate as well as the ADC sampling frequency - which is both software and hardware programmable.

6. Analog to digital conversions are software enabled, and are initiated when analog signals above a preset threshold are detected.

7. A terminal simulation program allows direct user interaction with the host.

3.3 Data acquisition System: Technical specification

System performance and hardware costs were main considerations determining the specification and the subsequent design. Use was made of suitable resources available from the university - particular items being the ADC and the card frame in which DASMA is housed.

3.3.1 Analog to Digital Conversion

Audio frequencies lie between 20 and 20000 Hz - a target sampling frequency is therefore about 50 KHz. (This could become a standard for professional digital audio systems).
Blesser [16] reports a maximum design goal of 90 dB dynamic range for perceptually lossless digital representation of the audio waveform.

The MAC-DAC Pod12 provides a dynamic range of approximately 72 dB (12 bit ADC) and a maximum sampling frequency of 100 KHz. This dynamic range is comparable with professional quality analog recording equipment.

3.3.2 Data Storage

Data is stored in a memory array consisting of 64 Kilobyte x 12 bit blocks of Dynamic Random Access Memory (DRAM), only one of which is installed in the prototype (although addressing facilities for three more are provided).

The required memory size is determined from the following equation.

MEMORY SIZE = SAMPLING RATE x DATA DURATION

Where: Memory size is measured in Kilobytes (Kbytes)

Sampling rate in Kilohertz (KHz)

Data duration in seconds (s)

The prototype can convert and store 2 seconds of audio data at a sampling frequency of 32 KHz.

3.3.3 System Architecture

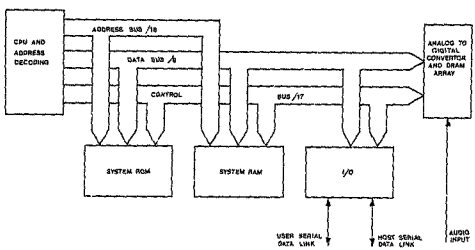


Figure 3.2 : Functional representation of the DASRA processor architecture.

The architecture of DASRA is best understood through an examination of figure 3.2. The system is shown to exhibit a standard processor bus structure. System activities are centered around the CPU address, control and data busses. The ADC and surrounding circuitry constitute a second hardwired ADC and Direct Memory Access (ADC/DMA) processor controlled directly by the CPU (shown in greater detail in figure 3.5, page 3-11).

1. The CPU controls all data acquisition and manipulation prior to the data being transmitted to the host computer. It consists of a 6502 microprocessor running at a 1 MHz clock rate plus 10 Kilobytes of system software in ROM.

2. A FORTH compiler, resident in ROM, forms the core of the system software which at a high-level emulates a Command Line Interpreter (CLI) at the user terminal.

3. At Power up, a menu of functions is displayed on the user terminal. These are all system level software functions. The menu may be recalled at any time by pressing the ESC key on the user terminal. A users manual and recommended operating procedure is given in appendix A. Development software written in FORTH may be interactively programmed

by the user. Utility software has been written and is documented in chapter 4.

4. Two RS-232 data links communicate between DASMA, the user terminal and the host computer. The data link protocols are software programmable. Default operation is as a 2400 baud bidirectional data link with no parity check. DASMA is as portable as the RS-232 standard.

5. The analog interface amplifies 100 mV to 1 volt signals with 20 to 20000 KHz bandwidth to -10 to +10 volt signals. The circuitry consists of a variable gain input preamplifier, connections for an external pre-sampling filter, and signal detection circuitry which provides a trigger pulse to initiate analog to digital conversions.

6. The analog to digital converter (ANALOGIC MP 6912) is configured for a +10 to -10 volt input voltage and produces linear PCM 12 bit 2's complement binary output with a sampling frequency variable up to 60 KHz under program control.

7. The DRAM is interfaced to the ADC data bus. When data acquisition is enabled by the CPU, the DRAM interface provides address and control information to write the 12 bit data to the DRAM under DMA control.

8. The DASMA software occupies addresses 9FE0H to 9FFFH on the 6502 processor memory map (figure 3.3, page 3-9). A series of data latches at these addresses completely controls the interaction between the CPU and the ADC/DMA processor, the dynamic memory array, and the serial I/O devices (Asynchronous Communication Interface Adaptors).

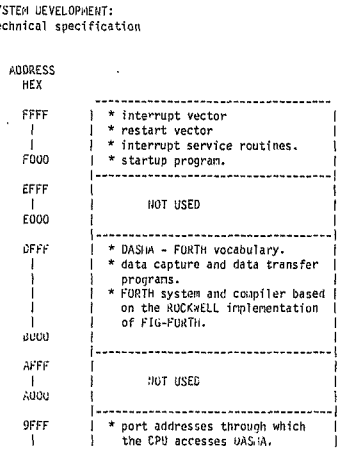


FIGURE 3.3: DASHA 6502 CPU memory map.

3.4 Detail of the Design

3.4.1 Physical construction

ASDA is housed in a VERO 3U. 160 mm deep card frame with guide rails for 34 VERO card slightly spaced, and half that loosely spaced. The cards used are VERO D.I.P board no. 10-0145 A. These cards plug into edge connector VERO 14-0207E, which has 42 pins.

The size of the cards prevents the use of a uniform bus structure - instead the required card inter-connections are hardwired and each card has its unique slot in the card frame.
Figure 3.4 shows an illustration of the physical dimensions of the card frame.

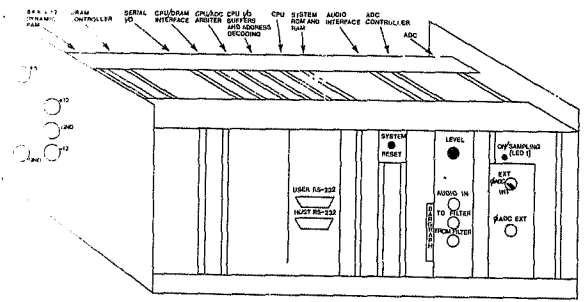


Figure 3.4: Illustration of the data capture system.

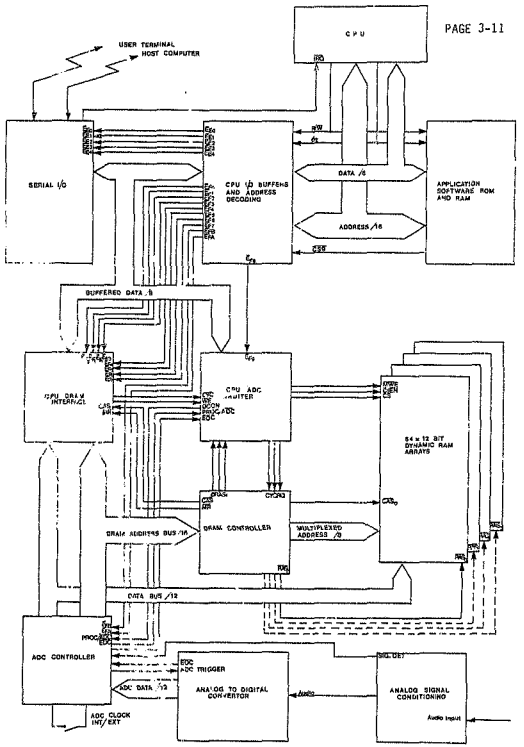


Figure 3.5: Functional block diagram of DASMA showing detail of the CPU, the ADC/DRM processor and the separate processor busses.

3.4.2 The Central Processing Unit (CPU)

The CPU consists of a 1 MHz 6502 microprocessor and a FORTH-based operating system residing in 10 kilobytes of ROM. System software is designed to provide user-friendly interaction via a menu-driven command line interpreter (described in chapter 4 - System Software).

3.4.3 System Synchronization Clock

The synchronization clock provides all the timing (excluding the CPU clock) throughout DASHA (Figure 3.6).
All DRAM cycle occur syncronously to this clock.

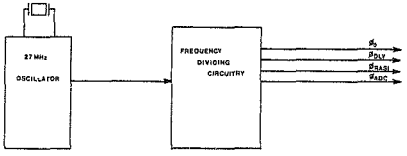


Figure 3.6: System synchronization clock generation module. A 27 MHz 3rd-overtone, series-resonant quartz crystal forms the heart of the system timing.

The system clock is divided synchronously to provide the following timing waveforms:

1. Refresh Clock (ϕ_0) - this 27 MHz clock provides accurate edges at multiples of 35 ns for DRAM timing waveforms.

2. Delay Clock (ϕ_{DLY}) - this 13.5 MHz clock is used to extend very quick pulses generated during system activity - usually by clocking the signals through D-type flip-flops.

3. RASI Clock (ϕ_{RASI}) - this 1.68 MH. clock is used to initiate refresh cycles during burst mode refresh, and to synchronize CPU command signals with DRAM refresh cycles.

4. ADC Clock (ϕ_{ADC}) - this 105.5 KHz clock divided down by the ADC controlling circuitry (under program control) is used as the sampling frequency for the analog to digital

converter.

3.4.4 Processor I/O Buffers and address decoding

This is the communications interface between the CPU and the DRAM, the ADC/DAC processor, and the serial I/O unit. Device addresses are decoded for addressing peripheral circuitry.

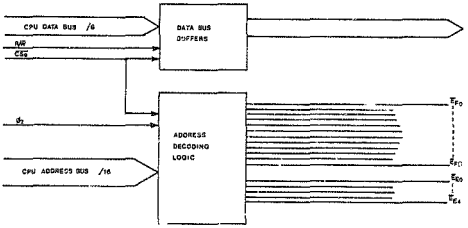


Figure 3.7: Processor I/O Buffers and Address Decoding.

ADDRESS RANGE (HEX)	FUNCTION	SYMBOL
9FE0 to 9FE7	Serial data I/O control	$\overline{E}_{E0}$ to $\overline{E}_{E7}$
9FF0 to 9FF7	Interface to DRAM	$\overline{E}_{F0}$ to $\overline{E}_{F7}$
9FF8 to 9FFF	Various system controls	$\overline{E}_{F8}$ to $\overline{E}_{FF}$

Table 3.1: Address bus decoded to control functions
$\overline{E}_{E0}$ through $\overline{E}_{FF}$

Reliable propagation of bidirectional data is ensured by bidirectional bus drivers at strategic intervals between the CPU and the peripherals.

3.4.5 CPU/ADC Arbiter

The CPU/ADC arbiter resolves CPU and ADC DRAM access requests and synchronizes the requests to the current activity of the DRAM.

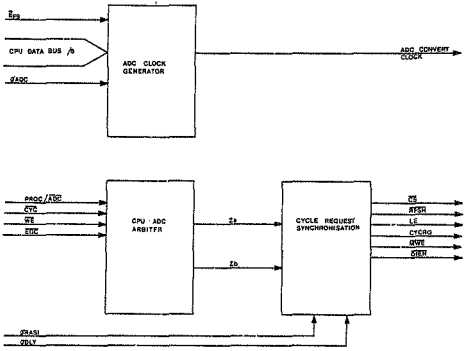


Figure 3.d: CPU/ADC Arbiter Block diagram. All memory access cycle requests are synchronized to the current activity of the DRAM.

1. INPUTS

   1. $\overline{E}_{Fb}$ - latches the 8 bit number on the CPU data bus into the ADC clock generator.

   2. ϕADC - 105,5 KHz clock - divided by the number in the ADC clock generator latch to give the ADC convert clock.

   3. PROC/$\overline{ADC}$ - indicates to the arbiter whether DRAM access requests are from the CPU or the ADC/DMA processor as follows:

      1 - CPU

      0 - ADC/DMA processor.

   4. $\overline{CYC}$ - indicates CPU-initiated DRAM access cycles - its state is reflected on Z_a when PROC/$\overline{ADC}$=1.

5. $\overline{WE}$ - indicates whether the CPU DRAM access cycle is a read or write cycle - its state is reflected on Zb when PROC/$\overline{ADC}$=1.

6. $\overline{EOC}$ - initiates DRAM write cycles when PROC/$\overline{ADC}$=0 - its state is reflected on Za and Zb when PROC/$\overline{ADC}$=0.

7. ϕ_{RASI} - executes continuous DRAM refresh cycles and is used to synchronize all DRAM access requests to the refresh cycles.

8. ϕ_{OLY} - used to derive short timing pulses from input state changes.

2. OUTPUTS

1. ADC CONVERT CLOCK (ϕ_{CON}) - sampling rate clock for analog to digital conversions.

2. $\overline{CS}$ - chip select for memory data bus drivers.

3. $\overline{RFSH}$ - halts continuous refresh after finishing the current refresh cycle - synchronously with (ϕ_{RASI}).

4. LE - indicates that the address information on the DRAM address bus is valid.

5. CYCRQ - synchronous DRAM access-cycle request.

6. $\overline{WE}$ - controls the read/write lines of the DRAM.

7. $\overline{DIEN}$ - direction control for the bidirectional memory data bus drivers.

The DRAM is normally undergoing continuous refresh cycles controlled by ϕ_{RASI} . Memory access cycles occur as follows:

1. CPU memory access: PROC/$\overline{ADC}$ = 1
$\overline{CYC}$ and $\overline{WE}$ are reflected on Za and Zb respectively and control the DRAM-access cycles completely.

2. ADC memory access: PROC/$\overline{ADC}$ = 0
$\overline{EOC}$ is reflected on Za and Zb and controls the DRAM-access cycles completely.

3. Za =0 - initiates a memory-access cycle of type (read or write) dependant upon Zb .

4. $\overline{RFSH}$ is pulled high on the completion of the current memory refresh cycle.

5. The LE pulse is generated on the next ϕ_{RASI} cycle and a CYCRQ pulse slightly later (controlled by ϕ_{DLY})

These output signals control the next memory access cycle as follows:

1. $\overline{CS}$ - selects the memory data bus drivers.

2. $\overline{RWE}$ - read/write cycle indicator to DRAM.

3. $\overline{OTEN}$ - direction control for memory data bus drivers.

4. $\overline{RFSH}$ - halts [illegible] of refresh cycles.

5. LE - latches the address of the current DRAM access cycle into the address multiplexing circuitry.

6. CYCRQ - initiates a DRAM access request via the DRAM controller.

Figure 3.10 on page 3-21 illustrates the waveform timing during DRAM read and write cycles.

3.4.6 CPU/Dynamic RAM Interface

Four blocks of 64K x 12 bit DRAM exist on the 6502 memory map as an I/O port at addresses 9FF0H to 9FFFH inclusive (the prototype uses only one block of DRAM - expansion is readily implemented by providing the extra address lines A_0 and A_1 - see DRAM controller).

This I/O port consists of three registers - each consisting of data latches accessed by the CPU through control lines $\overline{E_{F5}}$ through $\overline{E_{F7}}$. The registers hold information completely describing the current, previous or next memory cycle.

1. Control register - a transparent latch accessed by $\overline{E_{F5}}$. The two least significant bits of the CPU data bus are reflected on its output as $\overline{CYC}$ and $\overline{WE}$ - when $\overline{E_{F5}}$ is pulsed. These outputs are reset by a low going transition on $\overline{RR}$. The state of $\overline{CYC}$ and $\overline{WE}$ determine the DRAM activity (when the CPU is in control of the DRAM).

2. DRAM address register - an 18 bit latch with tri-state outputs which are enabled when a CPU DRAM access cycle is requested. Address information is written into this register as one 2 bit and two 8 bit words by the CPU writing to locations 9FF0H, 9FF1H and 9FF3H. This information represents the DRAM address at which the next DRAM access cycle will occur.

3. DRAM Data Register - fuctions as a 12 bit bidirectional data latch with tri-state outputs, it uses two unidirectional latches connected back to back - referred to as the write-data-register, and the read-data-register. Data is written to the write-data-register by the CPU as one 4 bit and one 8 bit data word at CPU addresses 9FF3H and 9FF4H. This data is reflected on the DRAM data bus as one 12 bit data word during a CPU-initiated DRAM write cycle.
During a DRAM read cycle the 12 bit data is written into the read-data-register by the DRAM controlling circuitry via $\overline{CAS}_0$. This data can then be read by the processor as two 8 bit bytes at locations 9FF6H and 9FF7H.

Note that in the software documentation (Chapter 4) these registers are referred to as:

1. the control-register,

2. the address-register,

3. the write-data-register, and

4. the read-data-register respectively.

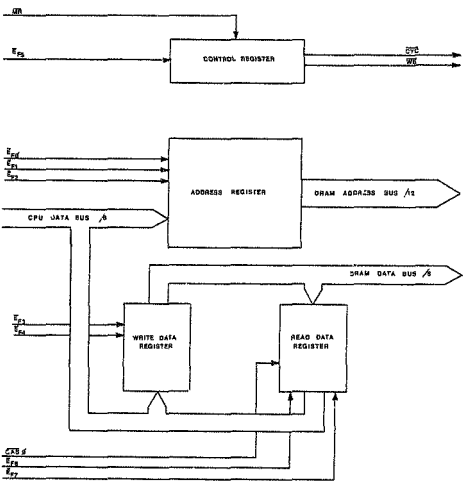


Figure 3.9: Dynamic RAM Interface Block Diagram. The CPU/DRAM interface is the only control and data link between the CPU and the DRAM. All CPU-initiated DRAM access cycles occur via this interface.

1. Memory Write Cycle

    1. The CPU writes information describing the data and the address at which it is to be written in DRAM to the data and address registers.

    2. the binary number 11 (three) is then written to the control register, the negative edge of $\overline{E}_{P5}$ causes $\overline{CYC}$ and $\overline{WE}$ to go low.

$\overline{CYC}$=0 enables the output of the address-register putting the DRAM address on the memory address bus.

$\overline{WE}$=0 enables the outputs of the write-data-register, putting the data on the memory data bus.

$\overline{CYC}$ and $\overline{WE}$ propagate through the system to the DRAM controller causing a memory-write cycle. On completion of which $\overline{MR}$ is pulled low by the DRAM controller resetting the contents of the control register causing $\overline{CYC}$ and $\overline{WE}$ to go high and inactive.

2. The Memory Read Cycle executes in a similar manner to the write cycle.

   1. Data representing the DRAM address which is to be read is written to the address-register,

   2. the binary number 10 (two) is written to the control-register causing $\overline{CYC}$ to go low while $\overline{WE}$ remains high. $\overline{CYC}$=0 puts the memory address on the DRAM address bus.

   3. The states of $\overline{CYC}$ and $\overline{WE}$ cause the DRAM controller to execute a memory-read cycle during which data is put on the DRAM data bus. $\overline{CAS}_0$ is then pulled low by the DRAM controller causing the data to be latched into the read-data-register.

   4. A valid read cycle is completed about 370 ns after $\overline{CYC}$ goes low. The processor may thus fix the next instruction (a minimum of 500 ns later) read the read-data-register and subsequently process the data.

3.4.7 Dynamic RAM Controller

Industry standard 64 Kbit DRAMs require refreshing every 2 ms. This can be implemented in several ways - in this case the DRAMs undergo continuous refresh. Any DRAM read or write cycle is accomplished by:

1. the CPU or ADC controller requesting the read/write cycle

($\overline{CYC}$=0 or $\overline{EOC}$=0),

2. latching the read/write commands until completion of the current refresh cycle,

3. halting the refresh circuitry ($\overline{RFSH}$=1),

4. executing the read/write cycle,

5. resetting the cycle request ($\overline{RR}$=0), and

6. restarting the refresh circuitry ($\overline{RFSH}$=0).

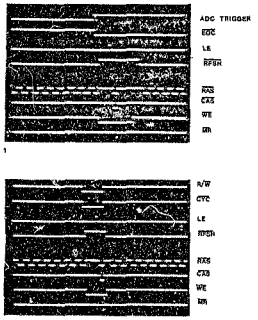


Figure 3.10. DRAM access cycles:
1. ADC-write cycle
2. CPU-write cycle

DRAM refresh cycles are of the order of five times as fast as CPU instruction cycles. Time is thus available to synchronize the processor cycle requests to the refresh circuitry as described above.

The DRAM Controller is built around an Advanced Micro Devices AM2964 DRAM controller (VLSI integrated circuit). The AM2964 provides the following functions:

1. Multiplexing the memory address bus.
2. Row address decoding - selection of one of four memory banks.
3. Refresh address generation.

Further circuitry (figure 3.11) uses a serial-in-parallel-out shift register (clocked by ϕ_0) to generate the DRAM timing waveforms.

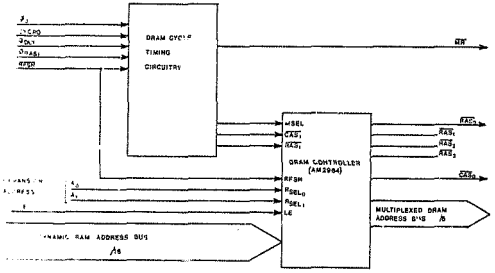


Figure 3.11: Dynamic RAM Controller

1. INPUTS
   1. ϕ_0 - 27MHz clock used to generate timing waveforms.
   2. CYCRQ - A pulse on this input initiates DRAM access cycles.
   3. ϕ_{DLY} - Used to extend the $\overline{RR}$ pulse.
   4. ϕ_{RASI} - Continuously refreshes the DRAM when idling.
   5. $\overline{RFSH}$ - When high terminates continuous mode refresh.

6. A$_8$and A$_9$ - Expansion memory addresses select addressed 64 Kbyte memory bank.

7. LE - Latches the address of the memory access cycle into the Am2964.

8. ADDRESS BUS - 16 bit address of memory access cycle in the DRAM bank selected by A$_8$ and A$_9$.

2. OUTPUTS

1. $\overline{RR}$ - Low going pulse indicates the completion of a memory access cycle.

2. $\overline{RAS_0}$- Row Address Strobe for DRAM bank 0.

3. $\overline{RAS_1}$ - Row Address Strobe for DRAM bank 1.

4. $\overline{RAS_2}$ - Row Address Strobe for DRAM bank 2.

5. $\overline{RAS_3}$ - Row Address Strobe for DRAM bank 3.

6. $\overline{CAS_0}$- Column Address Strobe, used for all [illegible].

7. MULTIPLEXED ADDRESS - 8 bit multiplexed DRAM address bus.

3.4.5 Analog to Digital Converter Controller

The ADC controller is the logical link between the DRAM and the analog to digital converter. When initialised by the CPU it takes complete control of the analog to digital conversion process and the writing of digitized data to the DRAM under a DMA environment.

The ADC controller interfaces to the DRAM data and address busses through tri-state data latches.

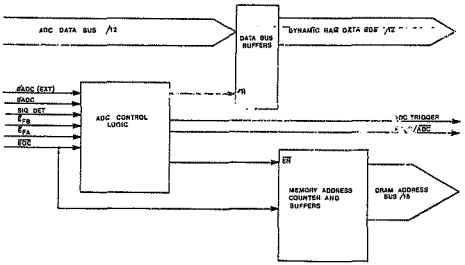


Figure 3.12 Analog to Digital Converter Controller block diagram.

1. INPUTS

   1. ADC data bus - 12 bit data directly from the outputs of the ADC.

   2. $\phi_{ADC(EXT)}$ - connection for an external ADC conversion (sampling) rate clock.

   3. ϕ_{ADC} - Internal ADC conversion clock, programmable by th

   4. SIG.DET - A pulse is generated on this line by the analog signal detection circuitry. If the ADC controller is in the convert-standby mode when this pulse occurs, the system will begin analog to digital conversions at a sampling rate determined by ϕ_{ADC}.

   5. $\overline{E}_{FB}$ - Sets the system in the convert standby mode.

   6. $\overline{E}_{FA}$ - Used by the CPU (under user direction) to abort an analog to digital conversion command.

   7 $\overline{EOC}$ - When low - data from the ADC is valid.

2. OUTPUTS:

1. Memory data bus - Tri-state drivers onto the memory data bus allow the ADC controller to disconnect from the DRAM data bus when not active.

2. Memory address bus - Tri-state drivers onto the memory address bus allow the ADC controller to disconnect from the DRAM address bus when not active.

3. ADC TRIGGER - Triggers ADC conversions when the ADC controller enables conversions.

4. PROC/$\overline{ADC}$ - Indicates to system circuitry whether the ADC controller or the CPU is accessing the DRAM.

Analog to digital conversions are activated as follows:

$\overline{E_{FB}}$ is pulled low setting the ADC controller in the convert-standby mode. Any subsequent analog input signal will enable conversions (via SIG.SET) - analog to digital conversions execute at a sampling rate determined by ADC TRIGGER. The CPU can abort a conversion via $\overline{E_{FA}}$.

$\overline{EOC}$ is generated by the ADC after every conversion (when digitized data is valid) and used to latch the data and control the outputs of the address and data bus drivers - then to generate a DRAM write cycle (via the DRAM controller). $\overline{EOC}$ is also used to increment the address counters so that digitized data is written to sequential addresses in DRAM.

3.4.9 Serial Input/Output

This unit has two Asynchronous Communications Interface Adaptors (ACIAs) interfaced to two RS-232 standard data drivers facilitating serial communication between the user terminal, the host computer and DASMA.

A baud-rate clock generator is provided and is programmable by the CPU.

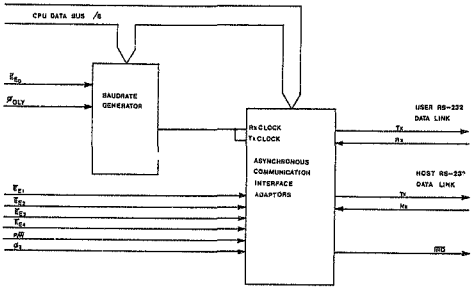


Figure 3.13 Serial Input/Output Interface.

1. INPUTS

    1. CPU DATA BUS.

    2. ϕ_2 - CPU phase two clock, enables data latches in the ACIAs.

    3. R/$\overline{W}$ - CPU read/write indicator.

    4. ϕ_{DLY} - 150 KHz clock used by the baudrate generator.

    5. User RS-232 - serial data link with the user terminal.

    6. Host RS-232 - serial data link with the host computer.

    7. $\overline{E}_{E_0}$ through $\overline{E}_{E_4}$ - control lines used to enable the baudrate generator data latch and the various ACIA functions.

2. OUTPUTS

    1. CPU DATA BUS.

    2. User RS-232.

    3. Host RS-232.

3.4.10 Analog Signal Interface

This unit interfaces the analog to digital converter to the audio source. It consists of:

1. an first-stage input preamplifier of variable gain 0-14 dB,
2. a second stage amplifier of fixed gain of 27 dB,
3. signal detection circuitry, and
4. a bargraph peak level indicator.

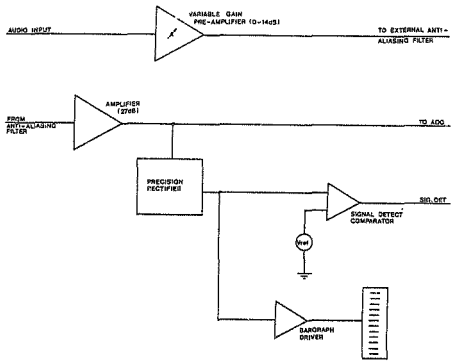


Figure 3.14: Analog Interface circuitry. The audio source connects to the first stage amplifier whose output must be connected via an external anti-aliasing filter to the input of the second stage amplifier.

The output of the second stage amplifier drives the ADC. This output is also rectified and fed into the bargraph driver and the signal detect circuitry.

The signal detect circuit consists of a comparator which generates SIG.DET when an input signal above 0.3 volts is detected.

The bargraph provides an indication of the peak level of the audio input to the ADC. The input signal should not exceed 75% of the maximum displacement of the bargraph - the analog signal to the ADC is then maintained within the correct limits.

CHAPTER 4

SYSTEM SOFTWARE

This chapter introduces the FORTH programming language. The design of the test, operating and maintenance software is flowcharted and presented in a Program Description Language (PDL) as high-level documentation.

4.1 Introduction to FORTH

FORTH is best described as a programming-environment because it has all the functions of:

1. a high-level programming language,
2. an assembler,
3. an operating system,
4. a system development tool, and is often referred to as
5. a mental illness.

The FORTH environment can operate at any of these levels depending upon the needs of the user.

Standard implementations of FORTH provide a high-level programming language with structured programming constructs and a machine level FORTH-based assembler. These do not directly provide for number-crunching operations.

FORTH programs are stored as words in a vocabulary. A word (program) is executed by typing the word in at the keyboard of the user terminal. The FORTH vocabulary is expandable - built up by creating new words as combinations of other words already existing in the FORTH system vocabulary.

FORTH allows intimate programmer/processor interaction by including machine-level programming constructs in the standard FORTH vocabulary and the FORTH-based assembler which simulates that of the host processor. Procedure calls between high-level FORTH and low-level FORTH-assembler routines are implemented by passing parameters between the FORTH parameter stack and a storage area accessed by the accumulator during FORTH assembler level routines.

High-level FORTH programs are typically defined as words in the vocabulary which describe their function. A program can be constructed of a string of words describing the action of that program.

For example - a process controller may regulate the temperature of a plant by controlling the flow of a coolant through a radiator system by:

```
BEGIN
    TESTEMP
    IFHOT
    TAPON
    IFCOOL
    TAPOFF
AGAIN
```

Where the words TESTEMP, IFHOT, TAPON etc. are all pre-defined FORTH words in the control systems application software. BEGIN and AGAIN are standard FORTH constructs which implement an infinite loop.

Good FORTH programming practice is to develop small simple words. These words are easily debugged using the FORTH interpreter, and are then compiled together as high-level words under a name describing the composite function of the procedures. In the above example the string of words could be compiled under one word - CONTROLTEMP. Subsequent execution of the program is by simply typing CONTROLTEMP at the user terminal.

FORTH facilitates an ideal development environment for small microprogrammed systems. The system code as well as compiled FORTH code is extremely compact. The entire FORTH system can reside in 8k of ROM and a typical application program in another 2k.

4.1.1 The FORTH Interpreter and DASMA Software

FORTH inherently operates as a Command Line Interpreter (CLI) through the FORTH word INTERPRET, which picks out words from an input stream, finds their definitions in the FORTH dictionary, and executes them.

DASMA issues a sign-on message to the user terminal when reset, or when ESC is pressed during execution of system software. The message is in the form of a menu of user functions - prompting the user with the DASMA system functions. These words allow the execution of data capture and transfer - by typing the required word at the user terminal (see appendix D).

High-level use of the system is in response to the menu-functions. At a low level the user can use the FORTH interpreter and compiler to develop further system software. Flowchart 1 on page 4-7 illustrates a simplified working of the DASM text interpreter under normal operation.

4.2 Program Description Language

Flowcharts are provided as conceptual descriptions of the software functions. Associated with the flowcharts Program Description Language (PDL) programs are provided. The system code is listed in appendix E.

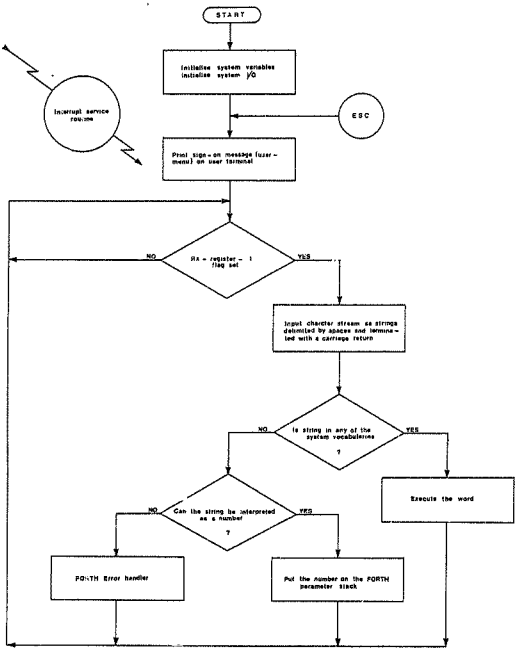
A PDL is useful for developing and documenting software. It provides a "structured-english" description of the software functions. The PDL used here obeys programming constructs based on those proposed by Young [19].

Translation from PDL to FORTH is simplified by implementing the FORTH parameter stack as a PDL data-structure. The PDL and its structures are dealt with in appendix D.

4.3 Software Documentation

The system software is described on four functional levels:

1. System Bootstrap Routines.
2. Utility Routines.
3. Test Routines.
4. User Software.



FLOWCHART 1: The FORTH Threaded Interpreter and DASHA operating system (OS). START indicates a power-up or system reset.

4.3.1 System Bootstrap

The bootstrap routine initializes the AIM 65 FORTH and configures it to match the OASIS hardware, it is written at assembler level and includes

1. initialisation of system variables,
2. initialisation of I/O mechanisms,
3. system dependant I/O routines, and
4. interrupt service routine.

4.3.1.1 System Variables - See PDL listing.

4.3.1.2 I/O Initialisation - See PDL listing.

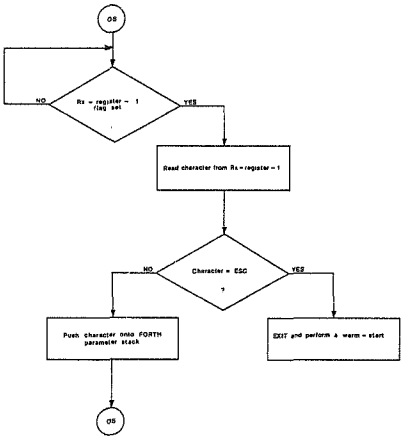
4.3.1.3 System Dependant I/O - The FORTH interpreter communicates with the user via four I/O routines which map the FORTH I/O into that of OASIS.

FORTH WORD	ASSEMBLER ROUTINE
KEY	INTERM
EMIT	OUTERM
?TERMINAL	TERCHCK
CR	CRLF

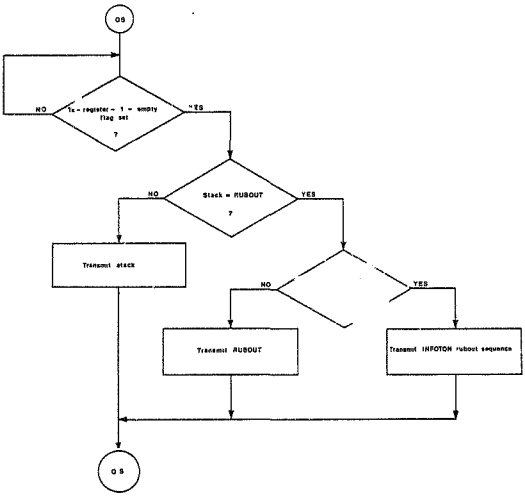
These routines are patched into the AIM 65 FORTH and point to the corresponding assembler routines indicated above.

1. KEY - Leaves the ASCII value of the next key depressed at the user terminal on the stack.
2. EMIT - Transmits the ASCII encoded character on top of the stack to the user terminal.
3. ?TERMINAL - Tests the user keyboard for actuation of any key and returns a true flag if a key is depressed.
4. CR - Transmits a carriage return and line feed to the user

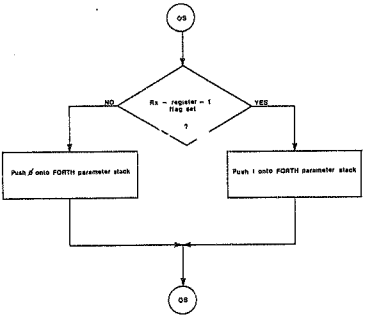
terminal.



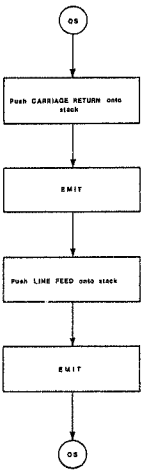
FLOWCHART 2: KEY - user terminal input routine (input-from-user-terminal).



FLOWCHART 3: EDIT - user terminal output routine (output-to-user-terminal).



FLOWCHART 4: ?TERMINAL - user terminal activation detect routine (terminal-cneck).



FLOWCHART 5: CR - issues carriage return line feed to user terminal (carriage-return).

4.3.1.4 Interrupting the CPU - Is necessary when the CPU is not fast enough to receive a continuous string of characters from the host computer under program control. A 256 character circular queue is used to buffer the character stream.
The queue is written into by the interrupt service routine when

1. the CPU interrupt mask is cleared,
2. ACIA 2 (host serial link) IRQ is enabled, and
3. when the host transmits serial data to ACIA 2.

The queue is subsequently read under program control.

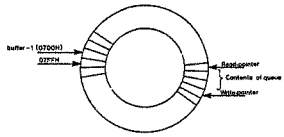
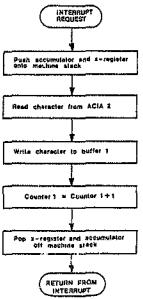


FIGURE 4.1: Circular queue used as a 256 character buffer on the host serial link.

The queue starts at machine address 0700H and uses 8 bit pointers - providing 256 locations folding back at address 07FFH.

KPT1 - address of next datum to be read (read-pointer).
MPT1 - address at which next datum will be written (write-pointer).
CNT1 - number of data bytes in the queue.



FLOWCHART 5: Interrupt Service Routine.

4.3.2 Utility Routines

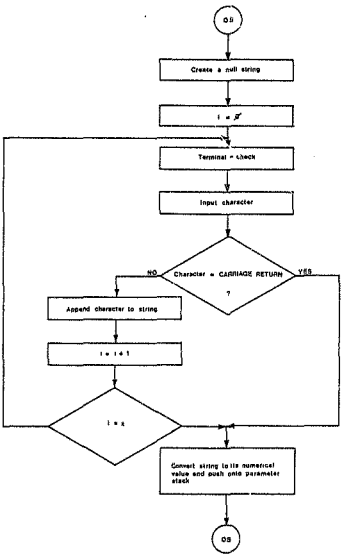
Utility software constitutes the routines necessary for the implementation of high-level I/O functions:

1. Input of ASCII coded numerics from the user terminal.
2. Dumping object code to an intelligent prom programmer.
3. Writing and reading 12 bit data to and from the DRAM.

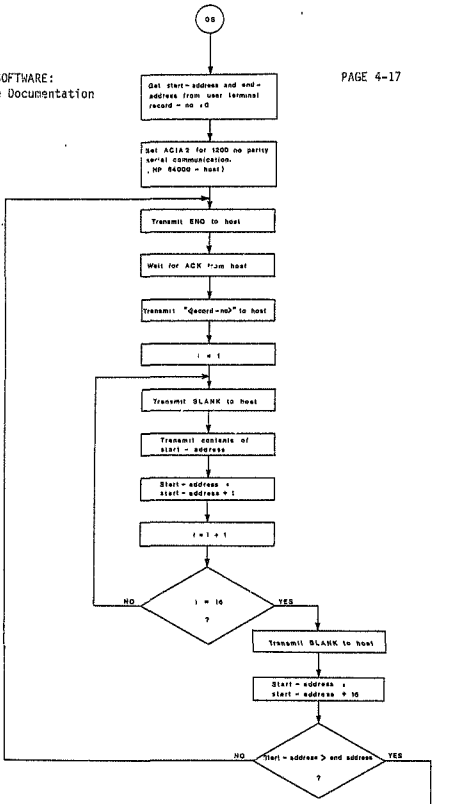
The utility module contains the FORTH definitions listed in table 4.1.

FORTH dictionary entry	PDL procedure name
IN-1	in-x (x=1)
IN-2	in-x (x=2)
IN-3	in-x (x=3)
IN-4	in-x (x=4)
O/P	dram-write
I/P	dram-read
SETUP	setup-acias
BAUDRATE	set-baudrate
STAT-1	status-1
STAT-2	status-2
OUTREM	out-remote
INREM	in-remote
ASC	byt-ascii
ASCOUT	word-ascii-out
DUMP	dump-object

TABLE 4.1: Utility FORTH system words and their PDL procedure names.



FLOWCHART 7: Input of ASCII coded numerics from user terminal (in-x).



FLOWCHART 8: DUMP - Dump object code under ENQ ACK protocol (dump).

4.3.3 Test Routines

The development of DASHA initially centered on the design of the dynamic RAM - extensive memory testing was required to verify the reliability of the DRAM and its controlling circuitry. The requirements of the memory test routines were based on the following:

1. Data is always written to the DRAM at sequential address locations by the ADC controller after each analog to digital conversion.

2. Data is always read from sequential address locations by the CPU.

The test proceeds as follows:

1. Data is written into the DRAM - setting sequential bit cells at sequential DRAM locations beginning with the first bit of location 0000H, ie:

   001H is written to address 0000H
   002H is written to address 0001H
   004H is written to address 0002H
   etc.

2. The data is read back and verified.

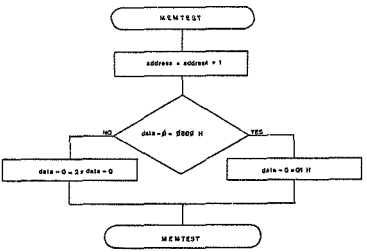
3. Step one is repeated, except that the next bit cell of every byte is set and the others reset, ie. the first operation sets the second bit cell of location one by writing 002H to address 0000H.

4. This is repeated until all sequential bit cell patterns have been tested and verified.

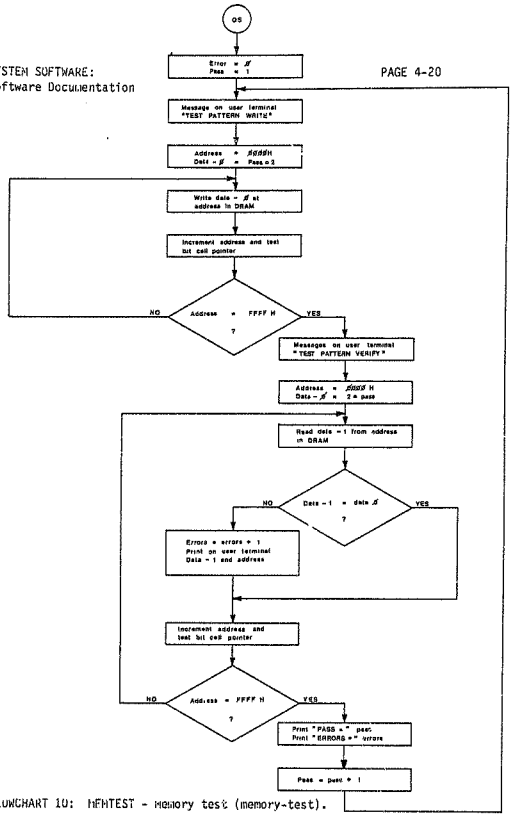
5. Appropriate messages convey the number of errors detected and the status of the test procedure.

FORTH dictionary entry	PDL procedure name
INC	increment-address-and-bit-cell-pointer
T1	write-test-pattern
T2	verify-test-pattern
MEMTEST	memory-test

TABLE 4.2: Test FORTH system words and their PDL procedure names.



FLOWCHART 9: INC - Memory test (increment-and-test-bit-cell-pointer).



FLOWCHART 10: MEMTEST - Memory test (memory-test).

4.3.4 User Routines

User software includes all those routines necessary to execute analog to digital conversions and then transmit the data to the host computer under program control. The main routines include:

1. Terminal simulation - DASHA is used as a dumb terminal to allow the user to communicate directly with the host computer.

2. Analog to digital conversion is controlled by three routines:

    1. Initialisation of the sampling frequency.

    2. Setting the system into a convert-standby mode - subsequent analog input triggers the ADC.

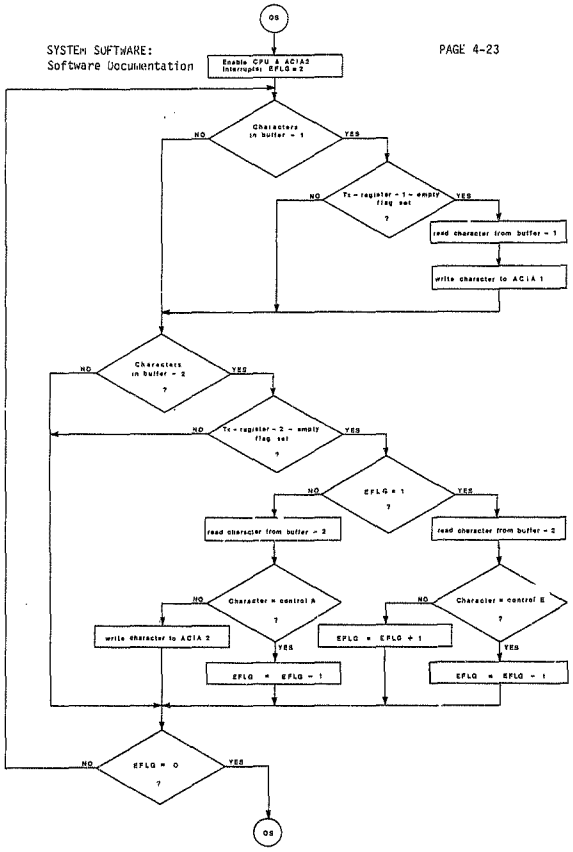
    3. Aborting the conversion.

3. Graphically examining the acquired data.

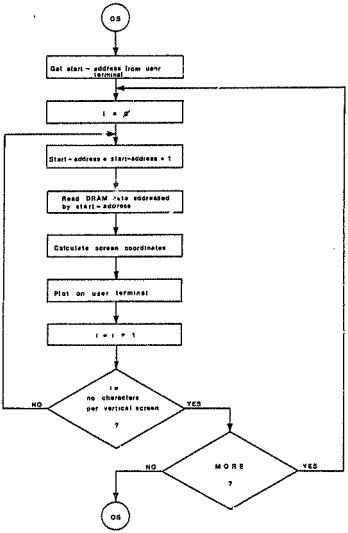
4. Transmission of digitized data to the host computer.

FORTH dictionary entry	PDL procedure name
TERSIM	terminal-simulation
COMP	convert-twelve-two-sixteen-bit-two's-complement
P1	calculate-plot-position
P2	plot-on-sreen
PLOT	rough-plot
SAMP-FREQ	set-sampling-frequency
INIT-SAMP	initialize-conversions
RESET-SAMP	abort-conversions
UP1	prompt-host
UP2	wait-for-prompt
UPLOAD	upload-to-host

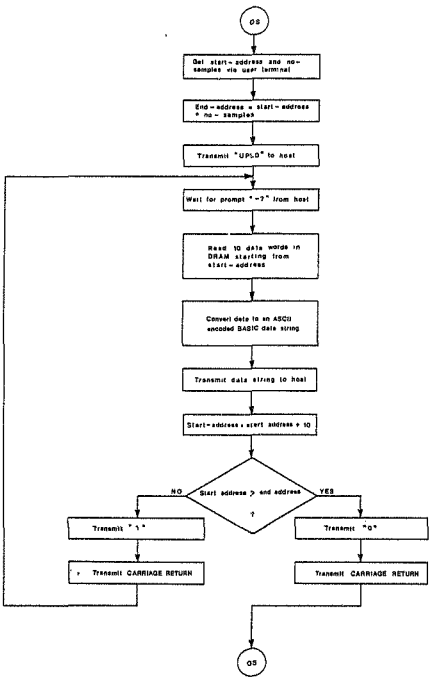
TABLE 4.3: User FORTH system words and their PDL Procedure names.



FLOWCHART 11: TERSIM - (terminal-simulation).



FLOWCHART 12: PLOT - rough plot on user terminal (rough-plot).



FLOWCHART 13: UPLOAD - formatted data to host computer (upload).

4.4 PDL Implementation of DASMA Application Software

```
SYSTEM DASMA Application Software

TYPE data-pointer: 16 bit;
     address-pointer: 16 bit;
     data : 16 bit;
     address: 16 bit;
     stack: 16 bit;

VAR (d0: data-pointer) := 0000H;
    (d1: data-pointer) := 0000H;
    (errors: data-pointer) := 0000H;
    (pass: data-pointer) := 0001H;
    (no-samples: data-pointer) := 0014H;
    (maximum: data-pointer) := 0800H;
    (minimum: data-pointer) := F800H;
    (no-columns: data-pointer) := 0041H;
    (position: data-pointer) := 0000H;
    (axis: data-pointer) := 0000H;
    (sample-clock: data-pointer) := 0000H;
    (baud-clock: data-pointer) := 00D8H;
    (stat-1: data-pointer) := 0000H;
    (stat-2: data-pointer) := 0000H;
    (dat-1: data-pointer) := 0000H;
    (dat-2: data-pointer) := 0000H;
    (counter-1: data-pointer) := 0000H;
    (counter-2: data-pointer) := 0000H;
    (read-pointer-1: data-pointer) := 0000H;
    (write-pointer-1: data-pointer) := 0000H;
    (escape-flag: data-pointer) := 0000H;
    (address-from: address-pointer) := 0000H;
    (address-to: address-pointer) := 0000H;

CONST buffer-1 : address;
      buffer-2 :address;
      read-data-register: data-pointer;
      write-data-register: data-pointer;
      address-register: address-pointer;
      control-register: data-pointer;
```

```
PROCEDURE input-output-initialisation
[**********************************]
[initialises CPU, sets baudrate generator at 2400]
[baud default, initialises ACIA's for 7 character]
[half-clock rate, no-parity check - communication]

BEGIN
    initialise machine stack to 00FFH
    clear decimal mode
    set interrupt mask
    baudclock := 00D4H
    stat-1 := stat-2 := 00FFH
    stat-1 := stat-2 := 0002H
END


PROCEDURE input-from-user-terminal
[*********************************]
OUTPUTS: stack

BEGIN
    REPEAT
        do nothing
    UNTIL [rx-register-1 flag set]
    read character from rx-register-1
    IF [character = ESC]
    THEN
        exit and perform a warm start
    ELSE
        stack := character
    ENDIF
END
```

```
PROCEDURE output-to-user-terminal
[*********************************]
INPUTS: stack

BEGIN
    REPEAT
        do nothing
    UNTIL [tx-register-1 empty flag set]
    IF [stack = RUBOUT]
    THEN
        IF [infoton flag is set]
        THEN
            send infoton rubout sequence
        ELSE
            send RUBOUT
        ENDIF
    ELSE
        send stack
    ENDIF
END

PROCEDURE terminal-check
[***********************]
[Checks for actuation of any key at the user     ]
[terminal. If key has been depressed returns a   ]
[TRUE flag on the FORTH parameter stack.         ]

OUTPUTS: stack

BEGIN
    IF [rx-register-1 flag set]
    THEN
        stack := 1
    ELSE
        stack := 0
    ENDIF
END

PROCEDURE carriage-return
[************************]
[issues a carriage return - line feed to the user]
[terminal.                                       ]

BEGIN
    stack := CARRIAGE-RETURN
    output-to-user-terminal
    stack := LINE-FEED
    output-to-user-terminal
END
```

```
PROCEDURE interrupt-service
[************************]
[On the condition that:                          ]
[  1. CPU interrupts are enabled,                ]
[  2. ACIA-2 interrupt is enabled, and           ]
[  3. ACIA-2 receives a character                ]
[the flow of program execution is halted, and    ]
[this routine writes the data in rx-register-2 to]
[buffer-2.                                       ]

BEGIN
    push accumulator onto machine stack
    push x-register onto machine stack
    read character from rx-register-2
    (0700H + write-pointer) := character
    counter-1 := counter-1 + 1
    pull x-register from machine stack
    pull accumulator from machine stack
    return from interrupt
END
```

```
MODULE utility

DEFINE in-1; in-2; in-3; in-4;
       dram-write; dram-read;
       byte-ascii; word-ascii-out;
       dump;


PROCEDURE in-x
[**********]
[reads x ascii encoded characters from the user  ]
[terminal and stores them as a string in a FORTH ]
[storage area called PAD. After x characters have]
[been input, or the input stream is terminated   ]
[with a CARRIAGE-RETURN, the string is converted ]
[to its numerical value in the current number    ]
[base and the value is placed on the FORTH       ]
[parameter stack.                                ]

OUTPUTS: stack(1)

BEGIN
    create a null string called string
    WHILE i NOT = x DO
        terminal-check
        input-from-user-terminal
        IF [stack NOT= CARRIAGE-RETURN]
        THEN
            append stack to string
        ELSE
            i := x
        ENDIF
    END
    convert string to numeric
    stack := numeric
END
```

```
MODULE utility

DEFINE in-1; in-2; in-3; in-4;
       dram-write; dram-read;
       byte-ascii; word-ascii-out;
       dump;


PROCEDURE in-x
[************]
[Reads x ascii encoded characters from the user  ]
[terminal and stores them as a string in a FORTH ]
[storage area called PAD. After x characters have]
[been input, or the input stream is terminated   ]
[with a CARRIAGE-RETURN, the string is converted ]
[to its numerical value in the current number    ]
[base and the value is placed on the FORTH       ]
[parameter stack.                                ]

OUTPUTS: stack(1)

BEGIN
    create a null string called string
    WHILE i NOT = x DO
        terminal-check
        input-from-user-terminal
        IF [stack NOT= CARRIAGE-RETURN]
        THEN
            append stack to string
        ELSE
            i := x
        ENDIF
    END
    convert string to numeric
    stack := numeric
END
```

```
PROCEDURE dram-write
[******************]
[by writing data, address and control information]
[to the write-data-register, address-register and]
[the control-register, a dynamic RAM write cycle ]
[is executed by the DRAM controller hardware.    ]

INPUTS: stack(2)

BEGIN
    (write-data-register) := stack
    (address-register) := stack
    (control-register) := 3
END
PROCEDURE dram-read
[*****************]
[by writing address and control information to   ]
[the address-register and the control-register   ]
[the DRAM controller hardware executes a DRAM    ]
[read cycle.                                     ]

INPUTS: stack(1)
OUTPUTS: stack(1)

BEGIN
    (address-register) := stack
    (control-register) := 2
    stack := (read-data-register)
END

PROGRAM dump
[**********]
[Allows dumping of object code from CPU RAM over ]
[a serial data link to an intelligent prom       ]
[programmer under an ENQ ACK protocol. The object]
[is converted to ascii and formatted as an object]
[file generated on the HP 64000 system.          ]

INPUTS: user terminal prompted

PROCEDURE setup-acias
[*********************]
[Sets up acias for 1200, no-parity serial        ]
[communication with the HP64000.                 ]

BEGIN
    baudclock := A8H
    stat-1 := stat-2 := 02H
END
```

```
PROCEDURE out-remote
[******************]
INPUTS: stack(1)

BEGIN
    REPEAT
        do nothing
    UNTIL [Tx-register-2-empty flag set]
    (data-2) := stack
END

PROCEDURE in-remote
[******************]
OUTPUTS: accumulator

BEGIN
    REPEAT
        do nothing
    UNTIL [Rx-register-2 flag set]
    accumulator := (data-2)
END

PROCEDURE ascii
[*************]
INPUTS: accumulator
OUTPUTS: accumulator

BEGIN
    IF [accumulator < 0AH]
    THEN
        accumulator := accumulator + 30H
    ELSE
        accumulator := accumulator + 36H
    ENDIF
END
```

```
PROCEDURE word-ascii-out
[**********************]
INPUTS: stack(1)

BEGIN
    dupstack
    stack := stack AND F0H
    stack := stack / 10H
    ascii
    out-remote
    stack := stack AND 0FH
    ascii
    out-remote
END

END MODULE utility
```

```
MODULE test

DEFINE increment-address-and-test-bit-cell-pointer;
       write-test-pattern;
       verify-test-pattern;
       memory-test;

PROGRAM memory-test
[*****************]
[A walking bit memory test routine is executed   ]
[on the Dynamic RAM. The test verifies correct   ]
[operation of the DRAM under sequential read and ]
[write conditions and issues the appropriate     ]
[messages to the user terminal.                  ]

INPUTS: user terminal prompted
OUTPUTS: test status information

PROCEDURE increment-address-and-bit-cell-pointer
[*********************************************]
BEGIN
    adr0 := adr0 + 1
    IF [d0=000H]
    THEN
        d0 := 1
    ELSE
        d0 := d0 + 2
    ENC
END


PROCEDURE write-test-pattern
[*************************]
BEGIN
    print "TEST PATTERN WRITE"
    adr0 := 0
    REPEAT
        stack := adr0
        stack := d0
        dram-write
        increment-address-and-bit-cell-pointer
    UNTIL [adr0 = FFFFH]
END
```

```
PROCEDURE verify-test-pattern
[**************************]
BEGIN
    adr0 := 0
    REPEAT
        stack := adr0
        dram-read
        d1 := stack
        IF [d0=d1]
        THEN
            increment-address-and-bit-cell-pointer
        ELSE
            errors := errors + 1
            PRINT d0
            PRINT adr0
            increment-address-and-bit-cell-pointer
        END
    UNTIL [adr0=FFFFH]
    PRINT "PASS" pass "COMPLETE" errors "ERRORS"
END


BEGIN
    errors := 0
    pass := 1
    d1 := 1
    adr0 := 0
    REPEAT
        d0 := d1
        write-test-pattern
        d0 := d1
        verify-test-pattern
        d1 := 2*d1
        pass := pass + 1
    UNTIL [d1=1000H]
END

END MODULE test
```

```
MODULE user
[********]
DEFINE  convert-twelve-to-sixteen-bit-two's-complement;
        calculate-plot-position;
        plot-point-on-screeen;
        rough-plot;


PROGRAM rough-plot
[***************]
[A rough plot of the data in DRAW is output on    ]
[the user terminal.                               ]

PROCEDURE convert-twelve-to-sixteen-bit-two's-complement
[*************************************************]
INPUTS: stack(1)
OUTPUTS: stack(1)

BEGIN
    dupstack
    dO := stack AND 8OOH
    IF [dO=8OOH]
    THEN
        stack := -((stack XOR FFFFH) + 1)
    END
END


PROCEDURE plot-point-on-screen
[***************************]
BEGIN
    FOR i := 1 TO cols DO
        CASE i OF
            position: PRINT "*"
            axis: PRINT "!"
        ELSE
            PRINT " "
        END
    END
END
```

```
PROCEDURE calculate-plot-position
[*****************************]
BEGIN
    FOR i := 0 TO no-samples DO
        stack := adr0 + i
        dram-read
        convert-twelve-to-sixteen-bit-two's-complement
        d0 := stack
        position := ((d0 - (min-value)) * (column-no))
                  / ((max-value) - (min-value))
        axis := ((0 - (min-value)) * (column-no))
                / ((max-value) - (min-value))
        plot-point-on-screen
    END
END


BEGIN
    PRINT "START ADDRESS"
    in-4
    adr0 := stack
    REPEAT
        calculate-plot-position
        adr0 := adr0 + (no-samples)
        PRINT "MORE"
        in-1
    UNTIL [stack := "N"]
END
```

```
PROGRAM terminal-simulation
[************************]
[The DASMA system is made to look transparent,   ]
[allowing direct communication between the user  ]
[terminal and the host computer. Exit from this  ]
[program is by typing control-A, control-E at the]
[user terminal.                                  ]
    .
BEGIN
    enable CPU and ACIA 2 interrupts
    IF [counter-1 NOT= 0]
    THEN
        IF [tx-register-1-empty flag set]
        THEN
            stack := (buffer-1 + read-pointer)
            (data-1) := stack
            counter-1 := counter-1 - 1
            read-pointer := read-pointer + 1
        ENDIF
    ENDIF
    IF [rx-register-1 flag set]
    THEN
        stack := (data-1)
        (buffer-2) := stack
        counter-2 := counter2 + 1
    ENDIF
    IF [counter-2 NOT= 0]
    THEN
        IF [tx-register-2-empty flag set]
        THEN
            IF [escape-flag = 1]
            THEN
                stack := (buffer-2)
                counter-2 := counter-2 - 1
                IF [stack = control-E]
                THEN
                    escape-flag := escape-flag - 1
                ELSE
                    (data-2) := control-A
                    escape-flag := escape-flag + 1
                ENDIF
            ELSE
                stack := (buffer-2)
                counter-2 := counter-2 - 1
                IF [stack = contro-1A]
                THEN
                    escape-flag := escape-flag - 1
                ELSE
                    (data-2) := stack
```

```
                ENDIF
            ENDIF
        ENDIF
    ENDIF
UNTIL [escape-flag = 0]
END
PROGRAM upload
[***********]
[This routine communicates directly with an OEPS ]
[routine implemented on the Data General ECLIPSE ]
[in BASIC. OEPS must be waiting for              ]
[response to its CODE ? prompt, the response is  ]
[UPLU. The routine commences to upload user      ]
[specified number of data bytes from the DRAM    ]
[to a BASIC data file specified on the ECLIPSE   ]
[by the OEPS routine FILE.                       ]

PROCEDURE to-host
[***************]
INPUTS: accumulator

BEGIN
    REPEAT
        do nothing
    UNTIL [TX-register-2-empty flag set]
    (data-2) := accumulator
END

PROCEDURE read-host
[*****************]
OUTPUTS: accumulator

BEGIN
    accumulator := (read-pointer + buffer-1)
    counter-1 := counter-1 + 1
    read-pointer-1 := read-pointer-1 + 1
END

PROCEDURE transmit
[*****************]
INPUTS: stack(1)

BEGIN
    accumulator := stack
    to-host
END
```

```
PROCEDURE ascii-ajust-transmit
[*****************************]
[Converts a two's complement decimal number to   ]
[its ascii value and transmits it to the host    ]

INPUTS: stack(1)

BEGIN
    dupstack
    IF [stack < 0]
    THEN
        stack := "-"
        transmit
    ENDIF
    stack := stack /MOD 1000
    stack := stack + 48
    transmit
    stack := stack /MOD 100
    stack := stack + 48
    transmit
    stack := stack /MOD 10
    stack := stack + 48
    transmit
END

PROCEDURE prompt-to-host
[***********************]
BEGIN
    stack := "U"
    transmit
    stack := "P"
    transmit
    stack := "L"
    transmit
    stack := "D"
    transmit
END
```

```
PROCEDURE wait-for-host-prompt
[**************************]
BEGIN
    escape-flag := 2
    REPEAT
        IF [counter-1 NOT= 0]
        THEN
            IF [e-    -flag NOT= 1]
            THEN
                read-host
                IF [stack = " "]
                THEN
                    escape-flag := escape-flag - 1
                ELSE
                    escape-flag := escape-flag + 1
                ENDIF
            ELSE
                read-host
                IF [stack = "?"]
                THEN
                    escape-flag := escape-flag - 1
                ENDIF
            ENDIF
        ENDIF
    UNTIL [escape-flag = 0]
END
```

```
BEGIN
    print "HEX STARTING ADDRESS"
    in-4
    adr0 := stack
    print "HEX NUMBER OF SAMPLES"
    in-4
    adr1 := adr0 + stack
    send-prompt-to-host
    REPEAT
        wait-for-host-prompt
        FOR i := 0 TO 9 DO
            stack := adr0
            dram-read
            ascii-adjust-transmit
            adr0 := adr0 + 1
            stack := ","
            transmit
        END
        IF [adr0 > adr1]
        THEN
            stack := "1"
            transmit
            stack := "carriage-return"
            transmit
            stack := "0"
        ELSE
            stack := "0"
            transmit
            stack := "carriage-return"
            transmit
            stack := "1"
        ENDIF
    UNTIL [stack = 1]
terminal-simulation
END
```

CHAPTER 5

TONE ANALYSIS AND ASSESSMENT

The application of Fast Fourier Transform (FFT) techniques to the analysis of selected musical tones and an assessment of the results thereof is presented in this chapter.

5.1 Review of the Analysis Requirements

The state of current research in music perception, analysis and synthesis is outlined in chapter two.

Frequency domain representations of musical tones are seen to provide insight into the relationship between the harmonic evolution of musical tones and the resultant timbral effects.

The literature reveals two important factors:

1. Natural instrument tones exhibit time varying partial structures which to be accurately represented must be observed at intervals not greater than 3ms.

2. The ear takes a finite time to process these sounds. This time is equivalent to an integration time of about 50ms.

The 3ms perturbations in the musical tones are thus considered unimportant in the perception of the tones. To graphically represent the tone-description parameters, with the intention of subsequently using these parameters to synthesize musical tone, an average value over every 50ms time interval is sufficient. Grey [10] has shown that a line-segment approximation to the temporal and harmonic variations of the original waveform is sufficient for synthesizing data reduced tones.

Graphical representations of each tone are presented in four different forms:

1. Amplitude-time plot - showing the envelope of the complete waveform analysed (amplitude vs time).

2. Time-Window - showing a steady state portion of the waveform (amplitude vs time).

3. Amplitude-time-spectrum plot - showing the time evolution of the amplitude spectrum of the musical tone (amplitude vs frequency vs time).

4. Harmonic-profile - showing the temporal evolution of the amplitude of each partial tone at nearly integral multiples of the fundamental frequency (amplitude vs time vs frequency).

Note that the names appearing under each graphic plot refer to the

file in which the data was originally stored on the host computer. Only the instrument type is of relevance.

5.2 Data Capture

The original tones were played by musicians who were asked to play short duration tones in the region of A above middle-C (approx. 440 Hz). The exact pitch of the tone was unimportant as the analysis can determine the harmonic relationships relative to the fundamental frequency (and thus the pitch). The analysis provided general harmonic structures for each instrument rather than for each note of an instrument.

Short duration tones, which show strong attack and decay characteristics and a small steady state region, were sufficient for the analysis.

The tones were recorded on a Nagra-IV tape recorder using BASF or SCOTCH low noise tape. Acoustic instruments were recorded in the department's acoustic laboratory using Bruel and Kjaer capacitor microphones. The electric and electro-acoustic piano were recorded directly via their line outputs.

The sounds were digitized and transferred to BASIC data files on the ECLIPSE using DASNA.

5.2.1 Digitization Conditions

Sampling frequencies and Fourier record lengths were determined from the characteristics of the tone under analysis. For an N-point transform of a waveform of maximum frequency Xw sampled at Fs Hz, the time duration of each window of analysis was T=N/Fs seconds. The frequency resolution of the resultant spectrum is 1/T Hz.

It was assumed that a waveform contained a maximum of fifteen dominant harmonics. Therefore the sampling frequency was based on the Nyquist rate of the fifteenth harmonic of the original waveform, ie 2x(15xXw). Analysis window lengths were chosen according to these resolution requirements. A Kron-Hite second order low-pass filter was used to remove components above the Nyquist frequency from the original tone.

5.3 Signal Analysis

The software used in the following analysis is documented in appendix C. The relevant macros which performed the spectral analysis and provided graphical plots were:

1. MAC$TPLOT - which plots a selected portion of the time waveform.

2. MAC$SPEC - which applies a fourier analysis to successive windows (of N points length) of the time waveform.

3. MAC$SPLOT - which plots a three-dimensional plot of the amplitude-time-spectrum computed by MAC$SPEC.

4. MACSHAR - which uses a cepstrum technique to determine the pitch-period of the time waveform (from the Fourier spectra resulting from MAC$SPEC) and using the pitch-period information calculates the amplitude variation of each partial frequency of the time waveform.

5. MACSHPLOT - which plots a three-dimensional harmonic spectrum of the data computed by MACSHAR.

5.4 Observations and Conclusions

5.4.1 Clarinet

The temporal evolution shown in the amplitude-time plot exhibited the characteristic slow attack of the clarinet waveform. This is a manifestation of the mechanical construction of the instrument and is attributed to the slow build up of the air column in the acoustic tube when excited by air blown past the reed.

The fundamental component showed the highest energy content. explains the bright clear sound of the instrument. amplitude-time-spectrum plot and the harmonic profile showed a suppression of the even harmonics. This is characteristic of a pipe closed at one end (appendix 3).

The steady state portion of the waveform existed only when the excitation was prolonged, and showed steady partial amplitudes until the excitation was removed. The decay showed constant partial amplitudes and a slowly decreasing fundamental amplitude relative to these partials. Then a rapid decrease in all partial activity - the higher frequency partials decay more rapidly than the lower frequency partials.

5.4.2 Acoustic Guitar

The guitar waveform was that of a plucked string and showed a faster attack than the clarinet. The peak amplitude was reached in about eight cycles of the excitation waveform. The excitation was in the form of an impulse and not prolonged like that of the clarinet. The guitar therefore did not exhibit a steady state region.

The partial structure depended on the intensity of the string being plucked, and showed energy at all partial frequencies at near-integer multiples of the fundamental.

Three breakpoints existed in the decay portion of the harmonic profile. Immediately following the attack, a rapid decrease in low partial energy was seen. The partials decayed slowly, the fundamental decaying faster than the other partials. Then a rapid

decrease in all harmonic activity - the higher frequency partials decayed more rapidly than the low frequency partials.

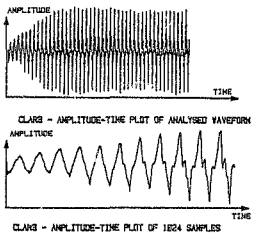
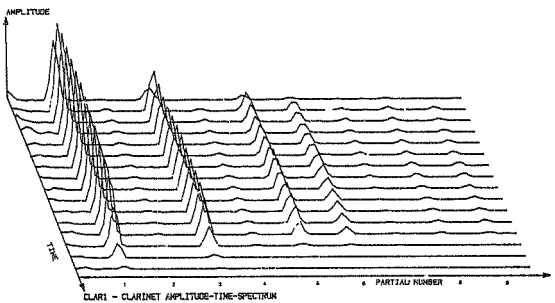


Figure 5.1: Clarinet amplitude-time waveforms.



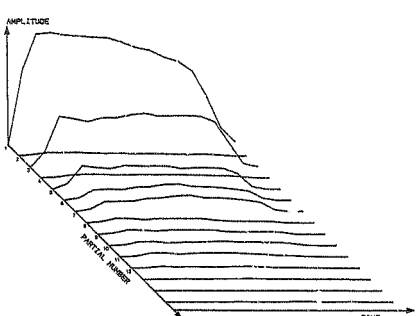
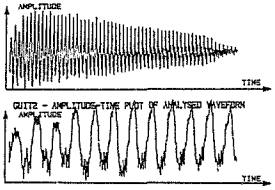
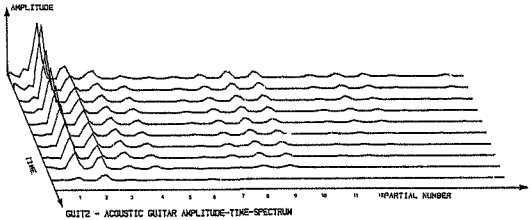


Figure 5.2: Clarinet amplitude-time spectrum and harmonic profile.



GUITZ - AMPLITUDE-TIME PLOT OF 1024 SAMPLES

Figure 5.3: Acoustic guitar amplitude-time waveforms.



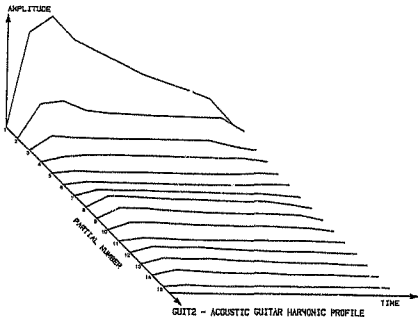


Figure 5.4: Acoustic guitar amplitude-time spectrum and harmonic profile.

5.4.3 Acoustic piano

The piano being a struck string instrument, like the guitar exhibited a rapid temporal attack.

The harmonic evolution of the piano was the most complex of all the tones analysed. It showed activity at all partial frequencies of near-integer multiples of the fundamental frequency. The second harmonic was particularly strong, being larger than the fundamental in certain cases. This was more pronounced for low frequencies than for higher frequencies, and can be attributed to the decreased acoustical coupling between the soundboard and the air in the low-frequency range - due to the relatively small acoustical-radiation resistance. The acoustical-radiation resistance decreases as the ratio of the dimensions of the vibrating surface to the wavelength decreases.

The piano showed no steady state harmonic region, and the relative energies in the partials decreased more rapidly with increasing partial frequency.

An examination of the amplitude-time-spectrum plot of PIANO1 showed that at onset a wide spread of inharmonic partial frequencies existed. These account for the percussive nature of a piano sound.

5.4.4 Electric Piano

This instrument provided useful reference information. The waveforms were electronically generated, and this manifested itself in the results of the analysis.

The regularity of the evolution and cessation of the harmonics of the electric piano contrasted with those of the natural instruments.

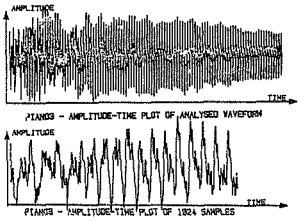


Figure 5.5: Acoustic piano, amplitude-time waveforms.

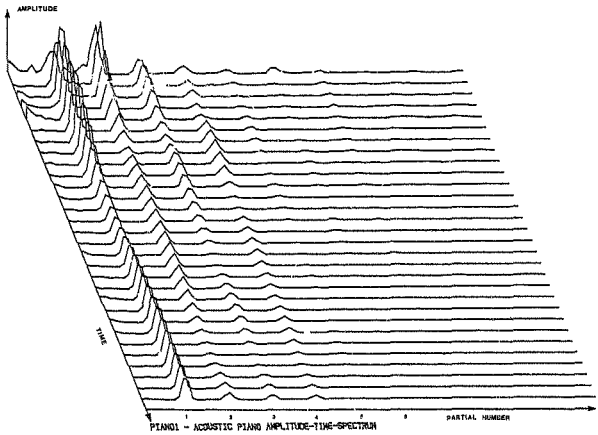


Figure 5.6: Acoustic piano, amplitude-time spectrum.

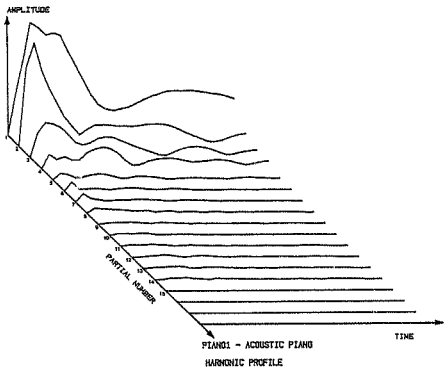


Figure 5.7: Acoustic piano, harmonic profile.

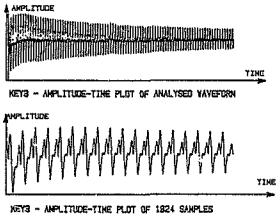


Figure 5.8: Electric piano, amplitude-time waveforms.

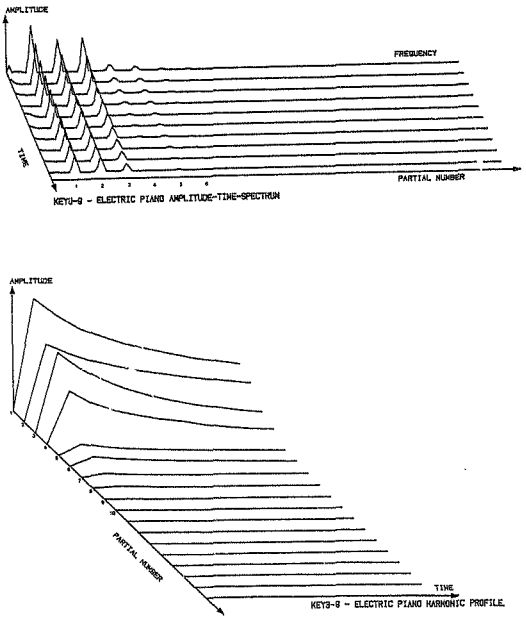


Figure 5.9: Electric piano, amplitude-time spectrum and harmonic profile.

CHAPTER 6

PROJECT OVERVIEW

The evolution of the musical tone analysis system and the results of some tone analyses have been documented in the preceding text. This chapter evaluates the musical tone analysis environment which has been commissioned. Suggestions for future work in the field of music analysis and syntnesis are also included.

6.1 The Future of Electronic Music

The final evaluation of the analysis environment will depend on its ability to determine the tone-description parameters to drive a synthesis system, and the accuracy with which these parameters can synthesize natural timbres.

Hopefully the university will continue with projects which will "close the loop". The references should provide an adequate starting point for any person interested in pursuing this type of research. It is unfortunate, however, that our present industry does not encourage research in music analysis and synthesis. At present much attention is being paid to this subject in Japan, Britain and America. But even in these countries, funding for music research is supplied by large commercial corporations - government funding, such as that for military research, is not generally available. Under these conditions this research will rely on individual interest.

6.2 Suggestions for Future Work

DASnA provides an open-ended development and evaluation environment for digital analysis of musical tones. The researcher can make great use of DASnA in its prototype form. There are numerous improvements which could be made to both the hardware and the software. DASnA, however, functions adequately, and it is felt that more important work would be the evaluation of analysis software and the development of a synthesis system.

6.2.1 Hardware

The prototype hardware has proved functionally adequate. No memory errors were detected over the three month evaluation period. When using the Nagra IV as the audio signal source, the audio interface supplied too much gain, and a Boss two channel mixer was used to attenuate the input signal. The audio input can be driven directly by a microphone.

Further improvements could be made by incorporating VLSI circuitry in the CPU/URAM interface to replace the latching circuitry at CPU addresses 9FF0H thru 9FF7H.

6.2.2 System Software

This software allows the user a large degree of control over the sampling and subsequent transfer of data. Cosmetic improvements could be made to enhance the user friendliness of the command line interpreter.

6.2.3 Analysis Software

The interface with the DEPS environment provides access to much signal processing software which, as is the nature of DEPS, will be continually upgraded. DASim, in effect, can benefit from these software revisions, and thus with minimum effort can be kept up to date. The analysis programs implemented here merely indicate the application of Fourier transform based spectral analysis techniques to musical tone analysis. The analysis software is the area which shows greatest potential for further investigation.

Analysis software evaluation would be most effective if a synthesis system existed for evaluating the parameters obtained from the analysis. Such a system could finally lead to the development of a stand-alone system performing the analysis and synthesis in real-time.

6.2.4 Synthesis Hardware

Additive synthesis has the advantage that only sinewave oscillators with amplitude controlling functions are required to generate the musical timbres; thus a synthesis system can be implemented using limited special purpose hardware. Alles [20] describes an oscillator useful for additive or frequency-modulation synthesis. The oscillator uses a 16-bit arithmetic unit controlled by a 4 Mhz Z-80 microprocessor. It is time-division-multiplexed to provide the function of 32 sinewave oscillators - each with frequency and amplitude modulation capabilities. Each sampled-data sinewave partial has a sampling rate of 32 kHz.

This type of hardware would be invaluable for implementing additive synthesis using the tone-description parameters obtained from DASMA.

6.2.5 A Real-Time System

A major problem of current electronic music synthesizers is the musician/machine interface. Popular synthesizers use a conventional keyboard for this purpose. Attempts to extend the interface to other popular instruments, such as the guitar, have had only limited success due mainly to the difficulty in detecting the pitch of the played notes. The most successful guitar synthesizer uses multiple pick-ups, each pick-up covering a small range of the instrument's frequency spectrum. This, however, requires the use of a specially adapted guitar.

A more advanced real-time pitch detection algorithm such as a bit-slice computation or a hardware cepstrum analysis could greatly extend the musician/machine interface by allowing any electric or electro-acoustic instrument to be used as the pitch source. The pitch is calculated in real time and the required musical tone is then synthesized.

The analysis and synthesis techniques evaluated through the use of DASMA could lead to the construction of a dedicated music synthesis system, which could prove to be of commercial value for use in music and film recording studios.

REFERENCES

REFERENCES

[1] Le Brun, M. Digital Waveshaping Synthesis. J. Audio Eng. Soc. Vol.27, Pg.250, (1979).

[2] Chowning, J.M The Synthesis of Complex Audio Spectra by Means of Frequency Modulation. J. Audio Eng. Soc. Vol.21, Pg.526, (1973).

[3] Plomp, R. Pitch of Complex Tones. J. Acoustical Soc. America. Vol.41, Pg.1526, (1967).

[4] Plomp, R. and Levelt, W.J.M. Tonal Consonance and critical Bandwidth. J. Acoustical Soc. America. Vol.38, Pg.548, (1965).

[5] Terhardt, Pitch Consonance and Harmony. J. Acoustical Soc. America. Vol.55, Pg.1061, (1974).

[6] Plomp, R. The Ear as a Frequency Analyzer. J. Acoustical Soc. America. Vol.36, Pg.1628, (1964).

[7] Pierce, J.R Attaining Consonance in Arbitrary Scales. J. Acoustical Soc. America. Vol.40, Pg.249, (1966).

[8] Plomp, R. and Steeneken, H.J.M. Effect of Phase on the Timbre of Complex Tones. J. Acoustical Soc. America. Vol.46 Pg.409, (1969).

[9] Berger, K.W. Some Factors in the Recognition of Timbre. J. Acoustical Soc. America. Vol.36, Pg.1888, (1964).

[10] Grey, J.M. An Exploration of Musical Timbre. Ph.D. dissertation. Dept of Phsychology, Stanford University, (1975). Distributed as Department of Music Report no. STAN-M-2.

[11] Mathews, M.V. and Pierce, T.R. Harmony and Nonharmonic Partials. J. Acoustical Soc. America. Vol.68, Pg.1252, (1980).

[12] Oppenheim, A.V. Schafer, R.W. and Stockham, T.G. Non-Linear Filtering of Multiplied and Convolved Signals. Proc. IEEE. Vol. 56, No. 8, Pg. 1264, (1968).

[13] Oppenheim, A.V. and Schafer, R.W. Homomorphic Analysis of

Speech. IEEE. Trans. on Audio and Electroacoustics, Vol. AU-16, Pg.221, (1968).

[14] Noll, A.M. Cepstrum Pitch Determination. J. Acoustical Soc. America. Vol.41, Pg.293, (1967).

[15] Childers, D.G. et al The Cepstrum: A Guide to Processing. Proc. IEEE. Vol.65, Pg.1428, (1977).

[16] Moorer, J.A. Signal Processing Aspects of Computer Music : A Survey. Proc. IEEE. Vol. 65, Pg. 1108, (1977).

[17] Hanrahan, H. An Open Ended Problem Solving and T[illegible]ng Package. Published by the Mech Eng. University of the Witwate[illegible] (1982).

[18] Blesser, B.A. Digitization of Audio: A Comprehensive Examination of Theory, Implementation and Current Practice. J. Audio Eng. Soc. Vol.26, Pg. 739, (1978).

[19] Young, S. P-notation: High level description language for software design. Microprocessors and Microsystems,
part 1. Vol. 4, No. 7, Pg. 267. (1980).
part 2. Vol. 4, No. 8, Pg. 307. (1980).
part 3. Vol. 4, No. 8, Pg. 3[illegible]. (1980).
part 4. Vol. 4, No. 10, Pg. [illegible] (1980).

Selected bibliography

[20] Alles, H.G. Real-Time Implementation of Digital Music Synthesis. Proc. IEEE. Vol. 64, No.4, Pg. 436, (1980).

[21] Rozenburg, M. Linear Sweep Synthesis. Computer Music Journal. Vol. 6, No. 3, Pg. 65, (1982).

[22] Broderson, R.W. A 500-Stage CCD Transversal Filter for Spectral Analysis. IEEE. Journal of Solid State Circuits. Vol. SC-11, No. 11, Pg. 75 (1976).

[23] Hutchins, B.A. Experimental Electronic Music Devices Employing Walsh Functions. Journal of the Audio Eng. Soc. Vol. 21, No. 3, Pg. 349, (1973).

[24] Winkel, F. Music, Sound and Sensation a Modern Exposition. Dover Publications, Inc. New York. (1967).

[25] Olson, H.F. Musical Engineering. McGraw-Hill Book Company, Inc. New York. (1952).

APPENDIX A

DASHA USERS MANUAL

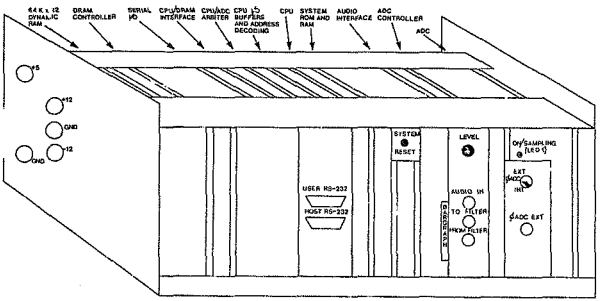


Figure A.1: Illustration of DAS1A.

A.1 Setting the System up

A.1.1 Power Supplies

Connect a 5v 3-amp regulated supply and a +12v 0 -12v 1-amp regulated supply as indicated in figure A.1.

A.1.2 Data Links

The system reset or power-up sets the data link protocol to a default baudrate of 2400 baud with 7-bit characters and one stop bit.

1. Connect a terminal to the user RS-232 link via a null-modem RS-232 connector, set to match the above protocol.

2. Connect a null-modem RS-232 connector to the host RS-232 link. If the ECLIPSE is being used as the host, this is identical to any RS-232 connector on any of the terminals.

A.2 Power-Up

On power-up LED 1 lights up and DASHA responds with a menu of user functions and the the FORTH prompt - OK. The FORTH interpreter will then respond to any user input.

1. If LED 1 is lit but there is no response at the user terminal, check the data link protocol of the terminal.

2. If the host is not the ECLIPSE, or the baudrate of the ECLIPSE port is other than 2400 baud:

   1. Type BAUDRATE

      DASHA responds with a menu of standard baudrates,

   2. respond with the correct input.

   Note that both the host and user terminal operate at the same baudrate, so changing the baudrate requires that the terminal baudrate be ajusted as well.

   Both data links should now function - if not the protocol of the host link may be modified by writing the correct data (see Motorola microprocessor components handbook - MC6850) to

   1. User ACIA by - HEX (data) STAT1 C!

   2. Host ACIA by - HEX (data) STAT2 C!

A.3 Terminal Simulation Program

Direct communication with the host can be initiated by typing TERSIM - all subsequent user-terminal inputs are relayed to the host and all host outputs are relayed to the user terminal. This mode of communication can be terminated by typing control-A control-E in at the user terminal.

A.3.1 Initialising UEPS

Type TERSIM

Sign on to the ECLIPSE in the usual way. Make sure that the DASNA directory is available. Run single precision BASIC - this allows increased length Fourier transforms to be used.

Type X BASICSP

Type ENTER "MAIN"

Type RUN

The ECLIPSE responds with: CODE ?

Now terminate the terminal simulation mode by typing control-A control-E.

A.4 Data Acquisition

Connect the audio source and anti-aliasing filter as shown in Figure A.1.

A.4.1 Sampling Frequency

Determine the required sampling frequency.

Type HEX SAMP-FREQ

DASNA responds with a menu of sampling frequencies. Type in the correct response.

A.4.2 Audio Level

Turn on the audio source and check that it does not cause the bargraph to exceed 75% of its maximum value. If it does the input signal can be attenuated via the level control (figure A.1).

Type INIT-SAMP

DASMA is now set in the convert-standby mode and will sample and save analog information following any input above the threshold voltage, and will continue sampling until 64 k memory is filled or the conversion is aborted by typing RESET-SAMP.

Note - when LED 1 (figure A.1) is not lit the system is sampling data and storing it in memory. Do not carry out any operations which use the DRAM until this light is lit; otherwise the data will be corrupted.

Typing RESET-SAMP at any stage cau[illegible]urn to the DASMA system and halts conversions.

A.5 Evaluating the data

Type PLOT

DASMA responds with START ADDRESS ?. The response to this determines the beginning of the data which will be graphically displayed on the user terminal.

The PLOT routine allows the user to check the data in DRAM prior to it being loaded down to the ECLIPSE. It is useful to determine the number of samples that contain the entire audio waveform, this number should be noted for data transfer to the ECLIPSE.

A.6 Uploading the Data

Type TERSIM

Make sure that the ECLIPSE is prompting with CODE ?

Type FILE

Enter a destination filename for the data. The ECLIPSE prompts again with CODE ?.

Type control-A control-E to return to the DASHA system.

Type UPLOAD

DASHA responds with NO OF SAMPLES ? - enter the required number of samples - DASHA will begin uploading the data to the ECLIPSE. A count of the current data byte being transferred is displayed on the user terminal.

After data transfer DASHA is in the terminal-simulation mode and OEPS responds with CODE ?.

The data is now saved in an ascii data file on the ECLIPSE and subsequent processing may be carried out using OEPS or the DASHA analysis-software.

APPENDIX B

ACOUSTIC CHARACTERISTICS OF SELECTED RESONANT BODIES

B.1 Strings

The musical tones of guitars, violins, pianos, banjos etc. are all characterised by the modes of vibration of stretched strings fixed at both ends. Vibrations in these strings occur at all harmonic frequencies.

$$Fn = fxn \qquad n=1,2,3...$$

Where: f - fundamental frequency
n - harmonic number
Fn - frequency of harmonic n

B.2 Reeds

Reeds vibrate with the modes of vibration of a bar fixed at one end. They are used to excite acoustical resonators in clarinets, oboes, saxophones, etc.

A bar fixed at one end exhibits nonharmonic partial frequencies at the following ratios of the fundamental frequency f.

f; 6,267f; 17,55f; 34,39f

B.3 Pipes

Open pipes - The partial frequencies generated by the air column in open pipes are at harmonics of the fundamental.

$$Fn = fxn \qquad n=1,2,3...$$

Closed pipes - The partial frequencies generated by the air column in pipes closed at one end are at odd harmonics of the fundamental.

$$Fn = f(2n-1) \quad n=1,2,3...$$

B.4 Strings on Acoustic Resonators

Acoustic resonators such as soundboards or Helmholtz resonators excited by stretched strings cause certain frequency bands of the excitation waveform to be amplified or attenuated according to the resonant characteristics of the acoustic resonator.

B.5 Reeds in Acoustic Tubes

The effective length of the resonant air column is changed by opening and closing holes in the acoustic tube causing a change in the resonant frequency of the air column.

The reed is coupled to the air column, which controls the motion of the reed and hence the fundamental vibration of the reed.

APPENDIX C

ANALYSIS SOFTWARE

The analysis software is documented in the standard OEPS format.

MACSSPEC - Fourier Analysis of Windowed Data

1. Authorship

C.S.Elion January 1983

2. Purpose

This macro calculates the fourier spectra of successive time windows of the source waveform. The resultant spectra are stored in a data file.

3. Method

The macro reads data from the source file, applies a Hamming window to user specified lengths of the data - which are then Fourier transformed resulting in frequency spectra at time intervals determined by the data window lengths. The resultant spectra are saved in a binary file under the same name as the source file with the extension "SP".

The macro uses five subroutines:

1. RDFILE - read data from the source file into the E array.
2. WIN - apply a Hamming window to the E array.
3. FTSH - (OEPS) shuffle the E array into bit reversed order.
4. FTSH - (OEPS) execute the Fast Fourier Transform.
5. MAGN - calculate the magnitude of the Fourier spectrum.

4. Calling sequence

Normal via the code MACSSPEC

5. Initial state

Immaterial, the macro prompts the user for all variable data.

6. Input

Console:

1. Name of source file.

2. N1 - window length index, window length is calculated as W=2**N1.

3. W0 - number of windows, this should be calculated by the user according to the no of samples in the source file (see Caution - below).

7. Output

Console:

1. Name of file in which spectral data is saved.

File:

1. Spectral data saved.

8. Final state

Data saved in file.

9. Caution

Be sure to calculate the correct data lengths or an error message will be displayed when the program attempts to read data beyond the end of the source file.

For example:

If 2000 hex samples have been uploaded from DASNA to the ECLIPSE - this implies that there are 16 time windows of 512 samples each ie.:

N1=9 WO=16

10. Listing

```
1000 REM>>>>>>>>>>>>>>>>>ACCESPEC
1010 PRINT "SOURCE FILENAME"
1020 INPUT SR$
1030 PRINT "RECORD LENGTH INDEX <10"
1040 INPUT R1
1050 PRINT "NO. RECORDS"
1060 INPUT R0
1070 LET SR$=SR$,STR$(R1),"SP"
1080 PRINT "DESTINATION FILE:",SR$
1090 LET R2=2*R1
1100 LET R$=R$/2
1110 LET R0=R0-1
1120 DIM A(R2),V(R$)
1130 OPEN FILE (1,4),SR$
1140 OPEN FILE (2,1),SR$
1150 WRITE FILE (2),R1,R0
1160 FOR R=1 TO R0
1170   PRINT "ACCESFILE1"
1180   GOSUB 2000
1190   PRINT "ACCESS."
1200   GOSUB 2000
1210   PRINT "ACCESTOP"
1220   GOSUB 2000
1230   PRINT "ACCEPT"
1240   GOSUB 2000
1245   FOR I=1 TO R2
1250     LET F(1,I)=F(1,I)/R2
1260     LET F(2,I)=F(2,I)/R2
1270   NEXT I
1280   PRINT FILE (2),V
1290 NEXT R
1300 CLOSE
1310 GOTO 9000
1320 REM <<<<<<<<<<<<<<<<ACCESPEC
```

MACSHAR - Determine Harmonic Profile Data

1. Authorship

C.S.Elion January 1983

2. Purpose

This macro calculates and saves data which represents the time evolution of the individual harmonics of the source waveform.

3. Method

The macro uses a cepstrum technique to determine the pitch-period of the original tone by calculating the inverse fourier transform of the log amplitude spectrum of the original tone at a point where its time waveform is approximately steady state. The pitch-period information is used to calculate the time evolution of each harmonic from spectral data previously calculated by the macro MACSSPEC. The resultant data is saved in a binary data file under the name of the source file with the extension "H".

The macro uses six subroutines:

1. FTST - (OEPS) sort the E array prior to inverse fourier transform.

2. FTSH - (OEPS) shuffle E array into bit reversed order.

3. FTFT - (OEPS) perform the Fast Fourier Transform.

4. MAGN - calculate the magnitude of the fourier spectrum (cepstrum) and find the index of the peak value after the dc component (pitch period) in the cepstrum.

5. AMPL - use the peak value calculated in MAGN to extract harmonic amplitude information from the fourier spectra of the source waveform.

6. SAVE - save the data calculated by AMPL in a binary data file.

4. Calling sequence

Normal via code MACSHAR

5. Initial state

The data file created by executing MACSSPEC must exist.
The user is prompted for all variables.

6. Input

Console:

1. HO - The number of harmonics to be calculated.

File:

1. Data file containing spectral information calculated by MACSSPEC.

7. Output

Console:

1. Name of file in which harmonic data is stored.

File:

1. Harmonic data.

8. Final state

Harmonic data stored in file.

9. Caution

MACSSPEC must be executed prior to this macro.

10. Listing

```
1000 REM <<<<<<<<<<<<<<<<<<<ALGMALGMAN
1010 PRINT "SOURCE FILENAME"
1020 INPUT S$
1030 LET S9$=S$,"M"
1040 LET S8$=S$,"SP"
1050 PRINT "READING FROM",S$
1060 PRINT "DESTINATION FILE",S8$
1070 PRINT "NO OF VARIABLES"
1080 INPUT NO
1090 OPEN FILE (1,3),S9$
1100 READ FILE (1),A(1,1),A0
1110 LET R=INT(A0/5)
1120 LET N1=R+1
1130 LET N2=2*N1
1140 LET N3=N2/2
1150 LET N4=N2-1
1160 DIM V(N2)
1170 FOR I=0 TO R
1180   NXT READ FILE (1),V
1190 NEXT I
1200 FOR I=1 TO N2
1210   LET C(1,I)=CONV(V(I))
1220   LET C(2,I)=0
1230 NEXT I
1240 ENTER "ALSPEC"
1250 GOSUB 2000
1260 ENTER "ALSPEC"
1270 GOSUB 2000
1280 ENTER "ALSPEC"
1290 GOSUB 2000
1300 ENTER "ALSAMPLE"
1310 GOSUB 2000
1320 CLOSE
1330 OPEN FILE (1,3),S8$
1340 READ FILE (1),A(1,1),A0
1350 LET N2=2*N1
1360 LET N3=N2/2
1370 LET N5=N2/2
1380 FOR K=1 TO A0
1390   NXT READ FILE (1),V
1400   ENTER "ALSAMPLE"
1410   GOSUB 2000
1420 NEXT K
1430 CLOSE
1440 ENTER "ALSDATA"
1450 GOSUB 2010
1460 GOTO 0600
1470 REM <<<<<<<<<<<<<<<<<<<ALGMALGMAN
```

MAC$SPLOT - Plot of Amplitude-Time-Spectrum

1. Authorship

C.S.Elion January 1983

2. Purpose

This macro plots a perspective of the spectral data calculated by MAC$SPEC. The data is plotted in such a manner as to give the impression of a three dimensional plot of Amplitude-vs-Frequency-vs-Time.
The amplitude scale may be linear, logarithmic or squared.

3. Method

The macro uses the OEPS graphics facility. The 3-D perspective is created by shifting the screen window after plotting each spectral window.

The macro uses two subroutines:

1. PLOTV - plot the contents of the V array.

2. TWV - calculate a data limit window for the V array.

4. Calling sequence

Normal via the code MAC$SPLOT

5. Initial state

The OEPS graphics must be initialised by GINIT.

6. Input

Console:

1. S8$ - Name of source file.

2. S7$ - LOG, LIN or POW - indicates log, linear or power

spectrum plot.

3. K8 - number of points of each spectrum to be plotted.

4. K4, K5, K6, K7 - coordinates of the plot on the display/plotter.

7. Output

Console:

1. Plot of the contents of the source file on the display/plotter.

8. Final state

All variables used are changed.

9. Caution

The spectra must have been calculated by MACSSPEC prior to using this macro.

10. Listing

```
1000 REM>>>>>>>>>>>>>>>>>>>>ACOUSPEC.BAS
1010 PRINT "SOURCE FILENAME"
1020 INPUT S$
1030 LET S9$=S$+",SP"
1040 PRINT "READING FROM:",S9$
1050 PRINT "(LOG), (LINEAR OR (POWER SPECTRUM"
1060 INPUT S7$
1061 PRINT "NO POINTS"
1062 INPUT N9
1070 LET N9=500
1080 PRINT "XREF XREF AL YL"
1090 INPUT N4,N5,N6,N7
1100 LET Z$="#SA"
1110 LET Z9(1)=N4
1120 LET Z9(2)=N5
1130 LET Z9(3)=N6
1140 LET Z9(4)=N7
1150 LET Z9(9)=4
1160 GOSUB 7000
1170 OPEN FILE (1),S9$
1180 READ FILE (1),A,A9
1190 LET A2=Z(1)
1200 LET A3=Z(2)
1210 LET A2=Z(2)
1220 DIM V(N9)
1221 LET A2=N4
1230 GOSUB 1410
1240 FOR A=1 TO A9
1250   CALL READ FILE (1),V
1260   IF S7$="LOG" THEN GOSUB 1330
1270   IF S7$="POW" THEN GOSUB 1370
1280   ENTER "ACK=CONT"
1290   GOSUB 2000
1300 NEXT A
1310 CLOSE
1320 GOTO 9000
1330 FOR I=1 TO N5
1340   LET V(I)=LOG(V(I))
1350 NEXT I
1360 RETURN
1370 FOR I=1 TO N5
1380   LET V(I)=V(I)^2
1390 NEXT I
1400 RETURN
1410 FOR I=1 TO N5 STEP 2
1420   LET V(I)=N4
1430   IF S7$="LOG" THEN LET V(I)=LOG(N4)
1440   IF S7$="POW" THEN LET V(I)=N4^2
1450 NEXT I
1460 FOR I=2 TO N5 STEP 2
1470   LET V(I)=0
1480   IF S7$="LOG" THEN LET V(I)=LOG(1/N4)
1490   IF S7$="POW" THEN LET V(I)=-1
1500 NEXT I
1510 ENTER "ACK=NV"
1520 GOSUB 2000
1530 RETURN
```

MACSHPLOT - Plot of Harmonic Profile

1. Authorship C.S.Elion January 19..

2. Purpose

This macro plots a perspective of the harmonic activity of the source waveform as calculated by MAC$HAR.

3. Method

The macro uses the OEPS graphics and creates a 3-D impression by shifting the screen window after plotting each harmonic.

The following subroutines are used:

1. GET - reads data from the source file into the H matrix.

2. PLOT!1 - plots the contents of the H matrix, moving the screen window after plotting each harmonic - to give a 3-D visual effect.

4. Calling sequence

Normal via code MACSHPLOT.

5. Initial state

The OEPS graphics must be initialised by GINIT.

6. Input

Console:

1. S$$ - name of source file.

File:

1. Data for harmonic plot.

MACSHPLOT - Plot of Harmonic Profile

1. Authorship C.S.Elion January 1983

2. Purpose

This macro plots a perspective of the harmonic activity of the source waveform as calculated by MACSHAR.

3. Method

The macro uses the OEPS graphics and creates a 3-D impression by shifting the screen window after plotting each harmonic.

The following subroutines are used:

1. GET - reads data from the source file into the H matrix.

2. PLOT11 - plots the contents of the H matrix, moving the screen window after plotting each harmonic - to give a 3-D visual effect.

4. Calling sequence

Normal via code MACSHPLOT.

5. Initial state

The OEPS graphics must be initialised by GINIT.

6. Input

Console:

1. S8$ - name of source file.

File:

1. Data for harmonic plot.

7. Output

Console:

1. S2S - name of destination file.

Graphic plot of the harmonic profile of the source data on the console/plotter.

8. Final state

All variables used are changed.

9. Caution

The harmonic data must have been calculated by HACSHAR.

10. Listing

```
1000 REM>>>>>>>>>>>>>>>>>>>>>MAC$HARPLOT
1010 PRINT "SOURCE FILENAME"
1020 INPUT S8$
1030 LET S2$=S8$,"H"
1040 PRINT "READING FROM:",S2$
1050 LET Z$="#8A"
1060 PRINT "(LIN)EAR OR (LOG) PLOT"
1070 INPUT S7$
1080 PRINT "XREF YREF XL YL"
1090 INPUT K4,K5,K6,K7
1100 LET Z9(1)=K4
1110 LET Z9(2)=K5
1120 LET Z9(3)=K6
1130 LET Z9(4)=K7
1140 LET Z9(9)=4
1150 GOSUB 7000
1160 ENTER "AC$BFI"
1170 GOSUB 2000
1180 LET N2=2↑N1
1190 LET N3=N2/2
1200 LET N0=N2-1
1210 ENTER "AC$PLOTII"
1220 GOSUB 2000
1230 GOTO 8200
1240 REM <<<<<<<<<<<<<<<<<<<<MAC$HARPLOT
```

MACSTPLOT - Temporal Plot of Source Waveform

1. Authorship

C.S.Elion January 1983

2. Purpose

This macro allows the source waveform to be plotted. Its usefulness is in that the amplitude envelope of the waveform may be graphically examined.

3. Method

The macro uses the DEPS graphics to plot the waveform.

The following subroutines are used:

1. RDFILE - reads data from source file into the E array.

2. PLOTV - plots the contents of the V array.

3. TMV - calculates a data limit window from the contents of the V array.

4. Calling sequence

Normal via code MACSTPLOT.

5. Initial state

The DEPS graphics must be initialised by GINIT.

6. Input

Console:

1. S8$ - name of source data file.

2. N1 - index for calculation of the number of points in each time window

MACSTPLOT - Temporal Plot of Source Waveform

1. Authorship

C.S.Elion January 1983

2. Purpose

This macro allows the source waveform to be plotted. Its usefulness is in that the amplitude envelope of the waveform may be graphically examined.

3. Method

The macro uses the OEPS graphics to plot the waveform.

The following subroutines are used:

1. RDFILE - reads data from source file into the E array.
2. PLOTV - plots the contents of the V array.
3. TWV - calculates a data limit window from the contents of the V array.

4. Calling sequence

Normal via code MACSTPLOT.

5. Initial state

The OEPS graphics must be initialised by GINIT.

6. Input

Console:

1. SB$ - name of source data file.
2. N1 - index for calculation of the number of points in each time window

3. WO - number of windows of length 2**N1 to be plotted.

4. K6,K7,K8,K9 - coordinates of the plot on the display/plotter.

7. Output

Plot of selected number of windows on the display/plotter.

8. Final state

All variables used changed.

9. Caution

Be sure to calculate the correct data lengths or an error message will be displayed when the program attempts to read data beyond the end of the source file.

For example:

If 2000 hex samples have been uploaded from DASHA to the ECLIPSE - this implies that there are 16 time windows of 512 samples each ie.:

N1=9
WO=16

10. Listing

```
1000 REM>>>>>>>>>>>>>>>>>>>>MACSIPLOT
1010 PRINT "SOURCE FILENAME"
1020 INPUT S$
1030 PRINT "WINDOW LENGTH INDEX"
1040 INPUT N1
1050 PRINT "NO. WINDOWS"
1060 INPUT N0
1070 PRINT "XREF YREF XL YL"
1080 INPUT X9,X8,X7,X6
1090 LET N2=2^N1
1100 LET N3=N2-1
1110 LET Z9(1)=X9
1120 LET Z9(2)=X8
1130 LET Z9(5)=X7
1140 LET Z9(4)=X6
1150 LET Z9(9)=4
1160 LET Z$="#8#"
1170 GOSUB 7000
1180 GOSUB 1370
1190 OPEN FILE (1,5),S$
1200 FOR K=1 TO N0
1210   ENTER "ALSMOFILE1"
1220   GOSUB 2000
1230   FOR I=1 TO N2
1240     LET V(I)=B(1,I)
1250   NEXT I
1260   ENTER "ALSPLOTV"
1270   GOSUB 2000
1280   LET Z9(1)=X9+X7
1290   LET Z9(2)=X8
1300   LET Z9(5)=X7
1310   LET Z9(4)=X6
1320   LET Z$="#8#"
1330   LET Z9(9)=4
1340   GOSUB 7000
1350 NEXT K
1360 GOTO 0200
1370 FOR I=1 TO 45 STEP 2
1380   LET V(I)=2048
1390   LET V(I+1)=-2048
1400 NEXT I
1410 ENTER "ALSTWV"
1420 GOSUB 2000
1430 RETURN
1440 REM<<<<<<<<<<<<<<<<<<<<MACSIPLOT
```

RDFILE - Read Ascii Coded Data File

1. Authorship

C.S.Elion January 1983

2. Purpose

DASHA saves data in an ascii file via the routine UPLD. The resultant data records are 10 datum long and the data file has sequential access only. This means that because of the sequential nature of the file, only sequential records of 10 datum length can normally be read. It is required, however to read data lengths which are a power of 2 long for the purposes of spectral analysis. READFILE allows reading of any length data records from the source file.

3. Method

RDFILE implements a temporary data buffer in the X array. If, say, 512 datum are required from the source file, then 520 datum are read from the file and the 8 remaining datum are saved in the x-array until a further 512 datum are required. Then the first 8 datum are taken from the x-array, and the remainder read from the source file as above.

4. Calling sequence

Normal via RDFILE

5. Initial state

The input data file must be opened on channel 1. The following variables must be initialised:

1. K - window number to be read.

2. N2 - number of datum required per read.

6. Input

Ascii coded data from the opened file.

7. Output

nil

8. Final state

N2 numbers from file(1) in the E array.

9. Caution

The sequential file is read by this routine in a loop where K is the loop index and is incremented from 1 to the number of windows contained in the file.

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>>>>>>RDFILE
2010 OPEN FILE (1,3),S$
2020 LET K1=K
2030 LET K1=N2*K1
2040 LET K2=K1-N2
2050 LET I=0
2060 FOR J=0 TO K1 STEP 10
2070   IF J<K2 THEN GOTO 2110
2080   INPUT FILE (1),E(1,I+1),E(1,I+2),E(1,I+3),E(1,I+4),E(1,I+5),E
2090   LET I=I+10
2100   GOTO 2120
2110   INPUT FILE (1),A,B,C,D,E,F,G,H,O,P
2120 NEXT J
2130 CLOSE FILE (1)
2140 FOR I=1 TO N2
2150   LET E(2,I)=0
2160 NEXT I
2170 RETURN
2180 REM<<<<<<<<<<<<<<<<<<<<<<<<<RDFILE
```

WIN - Hamming Window

1. Authorship

C.S.Elion January 1983

2. Purpose

The subroutine applies a hamming window to the time data prior to spectral analysis.

3. Method

The Hamming window is given by:

$$W(n)=X(n)*(0.54+0.46*COS\{[(I-1)/N2-0.5]*2\pi\}$$

4. Calling sequence

Normal via WIN

5. Initial state

1. N2 - length of data to be windowed.

2. E(1,I) - holds data to be windowed.

6. Input

nil

7. Output

nil

8. Final state

Data in E array is windowed by a Hamming window.

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>>main
2010 FOR I=1 TO N2
2020   LET E(J,I)=E(J,I)*(.54+.46*COS(((I-1)/N2-.5)*6.28318))
2030 NEXT I
2040 RETURN
2050 REM<<<<<<<<<<<<<<<<<<<<<main
```

MAGN - Magnitude of Spectral Data

1. Authorship

C.S.Elion January 1983

2. Purpose

The routine calculates the magnitude of the fourier spectrum and finds the index of the largest spectral component (excluding the dc component).

3. Method

The complex aritmetic utility is invoked to calculate the magnitude.
An assumption is made that the snectral components exist at frequencies greater than 1/25th of the sampling frequency. The magnitude of the largest spectral component is thus found by searching the magnitude of the snectrum from frequency values above 1/25th of the sampling frequency to 1/2 the sampling frequency.

4. Calling sequence

Normal via code MAGN

5. Initial state

1. E(1,I) - real values of fourier data.
2. E(2,I) - imaginary values of fourier data.
3. N2 - number of samples in E array .
4. N3 - N2/2.

6. Input

nil

7. Output

nil

8. Final state

1. V(I) - Magnitude of fourier analysis data.

2. [illegible] - W2 divided by the index of the max value of the V [illegible]y (pitch).

9. Caution

nil

10. Listing

```
2000 REM >>>>>>>>>>>>>>>>>>>>>ACCMAG<
2010 ENTER "ACROBACAM12"
2020 LET v0=0
2030 FOR I=1 TO 25
2040   LET x2=E(1,I)
2050   LET y2=E(2,I)
2060   GOSUB 5100
2070   LET v(I)=x5
2080 NEXT I
2090 FOR I=1+1 TO (25/50) TO 25
2100   IF ABS(V(I))>V0 THEN GOSUB 2150
2110 NEXT I
2120 RETURN
2130 LET v0=ABS(v(I))
2140 LET A0=I
2150 RETURN
2160 REM<<<<<<<<<<<<<<<<<<<<<ACCMAG>
```

AMPL - Amplitude of Spectral Harmonics

1. Authorship

C.S.Elion January 1983

2. Purpose

This subroutine determines the maximum value of the amplitudes of the fourier partials at near-integer multiples of the fundamental. The data is calculated from Fourier-transformed portions of the source waveform, and is saved in the H-matrix.

3. Method

The pitch (P) determined by the subroutine MAGN is used to search the fourier spectrum at frequencies (F) in the range:

$$P-P/2 < F < P+P/2$$

The resulting values are stored in the H-matrix.

4. Calling Sequence

Normal via code AMPL

5. Initial State

1. HO=number of partials to be extracted
2. NJ=half the number of fourier points
3. AO=pitch of source waveform
4. V(I)=fourier power spectrum of source waveform

6. Input

nil

7. Output

nil

8. Final State

H(K,I)=harmonic amplitude information

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>>>>>AMPL
2010 FOR I=1 TO 40
2020   LET V0=0
2030   LET B0=INT((AUX1(I)-A0/2)
2040   LET C0=INT((AUX1(I)+A0/2)
2050   IF C0>63 THEN LET C0=63
2060   FOR J=B0 TO C0
2070     IF ABS(V(J))>V0 THEN LET V0=ABS(V(J))
2080   NEXT J
2090   LET M(A,1)=V0
2100 NEXT I
2110 RETURN
2120 REM<<<<<<<<<<<<<<<<<<<<<<<<AMPL
```

SAVE - Save Data in H-Matrix

1. Authorship

C.S.Elion January 1983

2. Purpose

This subroutine opens a binary file and writes the contents of the H-matrix to it.

3. Method

Self-explanatory

4. Calling Sequence

normal via code SAVE

5. Initial State

1. S9S - destination filename
2. NI - window length index
3. NO - no. of harmonics
4. WO - no. of windows
5. H(I,J) - harmonic amplitude data

6. Input

nil

7. Output

1. Data from H-matrix.

8. Final State

Harmonic data saved in binary file.

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>SAVE
2010 OPEN FILE (1,1),59A
2020 WRITE FILE (1),N1,A0,H0
2030 FOR I WRITE FILE (1),M
2040 CLOSE
2050 RETURN
2060 REM<<<<<<<<<<<<<<<SAVE
```

GET - Read Data File into H-matrix

1. Authorship

C.S.Elion January 1983

2. Purpose

Used to read data files previously saved by SAVE into the H-matrix.

3. Method

Self-explanatory

4. Calling Sequence

normal via GET

5. Initial State

1. S2S - Source filename

6. Input

File:

1. data in the H-matrix.

7. Output

nil

8. Final State

Data read into H-matrix

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>GET
2010 OPEN FILE (1,5),525
2020 READ FILE (1),N1,N0,M0
2030 MAT READ FILE (1),M
2040 CLOSE FILE (1)
2050 IF S/$="CIN" THEN GOTO 2120
2060 FOR I=1 TO N0
2070    FOR J=1 TO M0
2080       IF M(I,J)=0 THEN LET M(I,J)=1
2090       LET M(I,J)=LOG(M(I,J))
2100    NEXT J
2110 NEXT I
2120 RETURN
2130 REM<<<<<<<<<<<<<<<GET
```

PLOT11 - Perspective Plot of Harmonic Data

1. Authorship

C.S.Elion January 1983

2. Purpose

Used to plot the harmonic landscape from data in the H-matrix.

3. Method

A three dimensional impression of the harmonic landscape is created by shifting the data limit window after each harmonic is plotted.

4. Calling Sequence

Normal via code PLOT11

5. Initial State

1. NO - number of harmonics
2. WO - number of time windows
3. N1 - window length index
4. N2 - window length = 2**N1

6. Input

nil

7. Output

Console:

1. Graphic plot on display/plotter.

nil

8. Final State

Plot on terminal/plotter

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>PLOT11
2010 LET V0=0
2020 FOR J=1 TO 40
2030   FOR K=1 TO 40
2040     IF ABS(M(K,J))>V0 THEN LET V0=ABS(M(K,J))
2050   NEXT K
2060 NEXT J
2070 LET K4=40
2080 LET K2=K0+1
2090 LET Y(1)=0
2100 LET Y(K0)=200
2110 LET Z$="APP"
2120 LET Z1$="IT"
2130 LET Z2(1)=0
2140 GOSUB 7000
2150 LET Z$="ANL"
2160 GOSUB 7000
2170 FOR J=1 TO 40
2180   LET T(J)=0
2190   FOR K=1 TO 40
2200     LET T(J)=T(J)+M(K,J)
2210   NEXT K
2220   LET Z$="APL"
2230   LET Z1(1)=0
2240   LET Z1(2)=0
2250   LET Z1(3)=.01
2260   LET Z1$="C"
2270   GOSUB 7000
2280   LET Z$="ASY"
2290   LET K4=K4+.5
2300   LET K5=K5-.5
2310   LET Z1(1)=K1
2320   LET Z1(2)=K5
2330   LET Z1(3)=K5
2340   LET Z1(4)=K7
2350   LET Z1(5)=0
2360   GOSUB 7000
2370 NEXT J
2380 LET K2=K6
2390 RETURN
2400 REM >>>>>>>>>>>>>PLOT11
```

PLOTV - Plot Data and Shift Data Limit Window

1. Authorship

C.S.Elion January 1983

2. Purpose

This subroutine is used by ..BSSPLOT to create a perspective plot of the fourier transformed data from the source waveform.

3. Method

Each spectrum is plotted and the data limit window is subsequently shifted - giving a three dimensional impression of the data.

4. Calling Sequence

Normal via code PLOTV.

5. Initial State

1. V(I) - data to be plotted.

2. N2 - no. of points.

6. Input

nil

7. Output

Console:

1. plot on display/plotter.

8. Final State

Plot on terminal/plotter.

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>PLOT
2010 LET Z$="A"
2020 LET Z9(9)=0
2030 LET Z9(1)=0
2040 LET Z1$="C"
2050 GOSUB 7000
2060 LET Z$="B"
2070 LET K4=K4+.2
2080 LET K5=K5-.5
2090 LET Z9(1)=K4
2100 LET Z9(2)=K5
2110 LET Z9(3)=K6
2120 LET Z9(4)=K7
2130 LET Z9(9)=4
2140 GOSUB 7000
2150 RETURN
2160 REM <<<<<<<<<<<<<<<<<<<<PLOT
```

TWV - data limit window from V-array

1. Authorship

C.S.Elion January 1983

2. Purpose

Calculates a data limit window to scale the data in the V-array to the screen window.

3. Method

Uses graphics core routine " DL".

4. Calling Sequence

Normal via code TWV.

5. Initial State

1. V(I) - plot data

2. N2 - max dimension of V-array

6. Input

nil

7. Output

nil

8. Final State

Data limit window calculated.

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>>TAV
2010 LET Z$="#PV"
2020 LET Z1$="IV"
2030 LET Z9(9)=0
2040 GOSUB 7000
2050 LET Z$="#DL"
2060 GOSUB 7000
2070 RETURN
2080 REM<<<<<<<<<<<<<<<<<<<<<TAV
```

UPLD - Upload Data from DASHA

1. Authorship

C.S.Elion January 1983

2. Purpose

This subroutine interfaces to the DASHA system by interacting with the DASHA-FORTH routine, UPLOAD. It loads data transmitted over the RS-232 link by DASHA into an ASCII data file.

3. Method

The data format on the RS-232 link is 10 datum plus one flag. UPLD saves the data in a file, checks the flag, if the flag indicates more data to come it sends the BASIC INPUT statement prompt to DASHA (space-?-space).

4. Calling Sequence

Normal via code UPLD (usually called by the DASHA routine which transmits the characters U P L D to the Eclipse).

5. Initial State

The sequential access file must be opened on channel 1.

6. Input

1. A,B,C,D,E,F,G,H,I,J - ASCII encoded data.

2. K - status flag:
K=1 indicat[illegible] to be transmitted by DASHA.
K=0 indica[illegible] of the data stream.

7. Output

File:

1. ASCII encoded data.

8. Final State

Data saved in sequential access file.

9. Caution

nil

10. Listing

```
2000 REM>>>>>>>>>>>>>>>>>>>>>UPLD
2010 LET S$="--####"
2020 INPUT A,B,C,D,E,F,G,H,I,J,K
2030 PRINT FILE (1),USING S$,A;",";H;",";C;",";D;",";E;",";F;",";G;
2040 IF K=1 THEN GOTO 2020
2050 CLOSE
2060 RETURN
2070 REM<<<<<<<<<<<<<<<<<<<UPLD
```

APPENDIX D

A PROGRAM DESCRIPTION LANGUAGE (PDL)

D.1 Describing FORTH using a PDL

High level FORTH programs are usually descriptive and thus easily understood. The high-level words in the FORTH vocabulary often consist of other high-level FORTH words. The example of chapter 4 can be extended:

```
: CONTROLTEMP
  BEGIN
      TESTTEMP
      IFHOT
      TAPON
      IFCOOL
      TAPOF
  AGAIN ;
```

This defines the word CONTROLTEMP whose function is easily understood by an examination of its component procedures (words). These words could, on the other hand, be constructed from low-level FORTH words or even FORTH-assembler instructions - making their appearance low-level and unreadable.

A Program Description Language is useful for developing and documenting software. It provides a "structured-english" description of the software functions. The PDL obeys programming constructs based on those proposed by Young [19]. Further modifications of the PDL to suit the FORTH environment are included in this appendix.

Translation from PDL to FORTH is simplified by implementing the FORTH parameter stack as a PDL data-structure. The FORTH parameter stack allows parameters to be passed between high level FORTH words. Parameters pushed on the stack by any previous operation remain on the stack until a subsequent operation pops them off and uses them.

The PDL stack is used as follows:

```
stack := expression

expression := stack
```

The former evaluates the expression and pushes its value onto the stack. The latter removes the top value off the stack and assigns its value to the expression.

The following programming structures are used throughout the PDL.

1. (0F4AH) - indirection - refers to the data contained at CPU address 0F4AH.

2. [number1<number2] - conditional - a conditional statement associated with conditional program execution.

3. [comment] - comment - may be placed anywhere in the program

4. data-pointer - an address at which variable data is held and is addressed by indirection.

5. address-pointer - a constant address at which variable address information is held and addressed by indirection.

Variables are created by assigning constant values to symbols - the constant value being the machine address at which the variable data is stored in RAM. The constant values are data-pointers or address-pointers and are used to address the data or address variable.

Variables are declared in the PDL by:

```
(dd:data-pointer) := 0
```

This indicates a data variable of type data, pointed to by constant dd of type data-pointer which is initially assigned the value 0. Software routines are defined on three levels:

1. MODULE

2. PROGRAM

3. PROCEDURE

These are the same as defined by Young.

D.2 Parameter Passing

All parameters are defined at the beginning of the software listing and are universal to all modules, programs and procedures - as are FORTH parameters universal to FORTH words.

FORTH parameters are often passed via the FORTH parameter stack. Assembler level parameters are passed via the accumulator. The PDL facilitates these two levels of parameter passing by:

```
INPUTS: stack(2)
OUTPUTS: stack(1)
```

or

```
INPUTS: accumulator
```

The former is used in describing a FORTH procedure requiring two items on the stack prior to its execution - and one parameter left on the stack due to its execution.

The latter indicates an assembly level routine which operates on the contents of the accumulator.

The lack of the INPUTS or OUTPUTS statement prior to a routine indicates no parameters passed to or from the routine.

D.3 FORTH words used in the PDL

1. DUP - Duplicates the top value on the stack.

2. /MOD - Leaves the remainder and signed quotient of n1/n2. The remainder has the sign of the dividend, where n1 and n2 are the uppermost numbers on the stack prior to execution.

3. KEY - Leaves the ascii value of the next user terminal key struck.

4. EMIT - Transmits the ascii character uppermost on the stack to the user terminal.

5. ?TERMINAL - Tests the user terminal keyboard for actuation, leaves a true flag to indicate actuation.

6. CR - Transmits a carriage return and line feed to the user terminal.

APPENDIX E

SOFTWARE LISTING

```
;
;
;         ASSEMBLY LANGUAGE BOOTSTRAP
;
;
            '650X'
        PROG EQU 0F000H

FORTH     EQU 0D000H
RAUDL     EQU 09FE0H
STAT1     EQU 09FE1H
DAT1      EQU 09FE2H
STAT2     EQU 09FE3H
DAT2      EQU 09FE4H
ESC       EQU 01BH
CR        EQU 00DH
LF        EQU 00AH
CNT1      EQU 0ABH
RPT1      EQU 0ACH
WPT1      EQU 0AEH


INIT    LDX  0FFH       ; processor and ACI
        TXS             ; initialization st
        CLD             ; clear decimal mode
        SEI             ; clear interrupt mask
        LDA  0D4H
        STA RAUDL       ; baud rate 2400 default
        LDA  0FFH
        STA STAT1
        STA STAT2       ; reset ACIAs
        LDA  003H
        STA STAT1
        STA STAT2       ; set ACIAs
;
;       INITIALIZE FORTH VARIABLES
;
        LDA  014H       ; NSAMP
        STA 00274H
        LDA  000H
        STA 00275H
        LDA  000H       ; MAXI
        STA 00276H
        LDA  000H
        STA 00277H
        LDA  000H       ; MINI
        STA 00278H
        LDA  0F8H
        STA 00279H
```

```
        LDA  041h       ; COLS
        STA 0027AH
        LDA  000H
        STA 0027BH
        STA 000FEH      ; rubout flag

        JMP FORTH

;
;       I/O ROUTINES
;
OUCHCK  LDA STAT1       ; poll status reg until
        AND  002H       ; TRANSMIT buffer empty
        BEQ OUCHCK
        RTS

INCHCK  LDA STAT1       ; poll status reg until
        AND  001H       ; RECEIVE buffer full
        BEQ INCHCK
        RTS

OUTERM  PHA             ; check status ACIA
        JSR OUCHCK
        PLA
        CMP  08BH       ; if char is RUBOUT
        BEQ RUBOUT      ; to rubout routine
        STA DAT1        ; output data
        RTS

INTERM  JSR INCHCK      ; check status ACIA
        LDA DAT1        ; get data
        CMP  ESC        ; if data is ESC
        BEQ ESCAPE      ; to escape routine
        RTS


RUBOUT  LDA 000FFH      ; check rubout flag
        BEQ RUB1        ; if infoton RUB1
        LDA  008H
        STA DAT1
        RTS

RUB1    LDA  01AH       ; implements rubout
        JSR OUTERM      ; on infoton terminal
        LDA  02BH
        JSR OUTERM
        LDA  01AH
```

```
        JSR OUTERM
        RTS

TRCHCK  LDA STAT1       ; code for FORTH word
        ANB  001H       ; ?TERMINAL
        JMP 08517H

CRLF    LDA  CR         ; code for FORTH word
        JSR OUTERM      ; CR
        LDA  LF
        JSR OUTERM
        JMP 08054H

ESCAPE PLA              ; fix machine stack
        PLA             ; and perform a warm
        JMP 08003H      ; start

;
;       INTERRUPT SERVICE ROUTINE
;
INTRPT PHA              ; save accumulator
        TXA
        PHA
        LDX WPT1
        LDA DAT2        ; get data
        STA 0700H,X     ; save in buffer
        INC CNT1
        INX             ; increment buffer pointer
        TXA
        STA WPT1
        PLA
        TAX
        PLA
        RTI
```

```
(**************************************************)
(*********DATA ACQUISITION SYSTEM FOR MUSICAL*********)
(**********************TONE ANALYSIS**********************)
(**************************************************)
(**********************SYSTEM SOFTWARE**********************)
(**************************************************)
(The software operates on four levels:          )
(               1. Bootstrap Routines           )
(               2. Utility Routines             )
(               3. Test Routines                )
(               4. User Routines                )
(The bootstrap programs are all assembly level  )
(as documented above. Utility, Test and user routines)
(are written in FORTH and FORTH-assembler where )
(execution speed is required.                   )
(**************************************************)
(**********************UTILITY ROUTINES**********************)
(**************************************************)

HEX
9FF2 CONSTANT SAMPRT
9FE0 CONSTANT BIGLOCK
9FF1 CONSTANT STAT1
9FE3 CONSTANT DAT1
9FF3 CONSTANT STAT2
9FF4 CONSTANT DAT2
0700 CONSTANT BUFF1
B0   CONSTANT BUFF2
A8   CONSTANT CNT1
AC   CONSTANT RPT1
A1   CONSTANT WPT1
AE   CONSTANT EFLG
AF   CONSTANT CNT2
9FE8 CONSTANT ALATCH
9FE9 CONSTANT DLATCH
9FE5 CONSTANT CTLATCH
9FE7 CONSTANT DLATCH1
338 CONSTANT ADRF
2AA CONSTANT ADRT
2A6 CONSTANT ADM
2A2 CONSTANT D0
2A4 CONSTANT D1
2A6 CONSTANT D2
2A8 CONSTANT D3
2AA CONSTANT D4
2AC CONSTANT ERR
2AE CONSTANT PASS
2B0 CONSTANT ADR0
2B2 CONSTANT ADR1
```

```
274 CONSTANT NSAMP
276 CONSTANT MAXI
278 CONSTANT MINI
27  CONSTANT COLS
27C CONSTANT POS
27E CONSTANT AXIS


(*******************USER TERMINAL I/O**********************)

: IN-1          (read 1 ascii coded number)
PAD 1 EXPECT    (from input device and leave)
0 0 PAD 1 -     (decimal value on the)
(NUMBER)        (parameter stack)
2DROP ;


: IN-2          (read 2 ascii coded numbers)
PAD 2 EXPECT    (from input device and leave)
0 0 PAD 1 -     (decimal values on the)
(NUMBER)        (parameter stack)
2DROP ;


: IN-3          (read 3 ascii coded numbers)
PAD 3 EXPECT    (from input device and leave)
0 0 PAD 1 -     (decimal values on the)
(NUMBER)        (parameter stack)
2DROP ;


: IN-4          (read 4 ascii coded numbers)
PAD 4 EXPECT    (from input device and leave)
0 0 PAD 1 -     (decimal values on the)
(NUMBER)        (parameter stack)
2DROP ;

(*********************DRAM I/O ROUTINES*********************)

: O/P          (Writes the top word on the parameter)
DLATCH !       (stack to the location in DRAM addressed)
ALATCH !       (by the next word on the stack)
3 CTLATCH C! ;


: I/P          (Reads data from the DRAM location)
ALATCH !       (addressed by the word on top of the stack)
1 CTLATCH C!
DLATCH1 @
```

```
;

(**************OJECT CODE DUMP ROUTINES****************)

HEX             (Initialise ACIAs for 1200 baud)
CODE SETUP      (no-parity serial communication)
XSAVE STX,      (with intelligent PROM programmer)
A8   LDA,       (i.e. HP 64000.)
9FE0 STA,
FF   LDA,
9FE1 STA,
9FE2 STA,
02   LDA,
9FE1 STA,
9FE3 STA,
XPAUS LDX,
NEXT JMP,
END-CODE


CODE OUCHCK     (Reads status register ACIA2)
XSAVE STX,      (until Tx-register empty flag)
BEGIN,          (is set.)
9FE3 LDA,
02   AND,
0= NOT UNTIL,
XSAVE LDX,
NEXT JMP,
END-CODE


CODE INRCH      (Read status register ACIA2)
XSAVE STX,      (until Rx-register full flag)
BEGIN,          (is set.)
9FE3 LDA,
01   AND,
0= NOT UNTIL,
9FE4 LDA,
XPAUS LDX,
PUSHAA JMP,
END-CODE


: OUTREM        (Output character on the stack)
  OUCHCK        (via ACIA2.)
  9FE4 C! ;


CODE ASC        (Convert the data word on the)
```

```
1   LDA,          (parameter stack to its ASCII)
SETUP JSR,        (equivalent.)
N LDA,
A   CMP,
0< IF,
   30   ADC,
ELSE,
   37   ADC,
ENDIF,
PUSHAA JMP,
END-CODE


: ASCOUT          (Output the ASCII value of the)
  DUP F0 AND      (data word on the parameter stack)
  10 /            (via ACIA2.)
  ASC OUTREM
  0F AND
  ASC OUTREM ;


: DUMP            (Format the object code in CPU RAM:
  HEX             (at locations ADRF to ADRT as a PROM)
  0 ADM !         (listing file on the HP 9000 - then)
  ." FROM ? "     (transmit this data.)
  IN-4
  CR
  ADRF !
  ." TO ? "
  IN-4
  CR
  ADRT !
  SETUP
  BEGIN
     05 OUTREM
     BEGIN
        INREM
        0A =
     UNTIL
     3C OUTREM
     ADM DUP 1 + C@ ASCOUT C@ ASCOUT
     3F OUTREM
     10 0 DO
         20 OUTREM
         ADRF DUP @ C@ ASCOUT
         1 SWAP +!
     LOOP
     0D OUTREM
     10 ADM +!
```

```
    ADRS @ ADRT @ )
  UNTIL
  @ ADR ! ;

(**********************************************************)
(**********************TEST ROUTINES***********************)
(**********************************************************)

: INC                   (Increments test address and checks)
ADRR @ 1 + ADRR !       (if 12 bits have been tested.)
D0 @ 800 = IF
             1 D0 !
           ELSE
             D0 @ 2 * D0 !
THEN ;

: T1                            (Write memory test pattern.)
CR ." TEST PATTERN WRITE "
@ ADRR !
BEGIN
    ADRR @ D0 @ C/P
    INC
    ADRR @ -1 =
UNTIL
;

: T2                            (Verify data pattern in the)
CR ." TEST PATTERN VERIFY " (DRAM.)
ADRR ADRR !
BEGIN
    ADRR @ T/P DUP
    D0 @ = IF
           DROP
           INC
           ELSE
              ERR @ 1 + ERR !
              ADRR @ . .
              INC
THEN
ADRR @ -1 =
UNTIL
CR ." PASS " PASS @ . ." COMPLETE " ERR @ . ." ERRORS"
CR ;

: MEMTEST       (Perform walking-bit memory test)
```

```
0 ERR !             (routine to verify reliable)
1 PASS !            (operation of the DRAM.)
1 D1 !
6 ADRG !
BEGIN
    D1 @ D0 !
    T1
    D1 @ D0 !
    T2
    D1 @ 2 * DUP D1 !
    PASS @ 1 + PASS !
    1000 =
     UNTIL ;

(**************************************************************)
(****************UTILITY ROUTINES******************************)
(**************************************************************)

(**************************PLOT ROUTINE************************)

: CONV         (Convert 12 bit to 16 bit 2's complement)
DUP            (representation.)
[illegible] AND
[illegible]   IF
     [illegible] NEGATE
THEN ;


: [illegible]                        (Plot points on screen.)
[illegible] @ 1 + 1 DO
    [illegible] @ I = IF ." *" ELSE
    AXIS @ I = IF ." I" ELSE
    SPACE THEN THEN
  [illegible]
[illegible]


: [illegible]                        (Calculate position of)
[illegible] 1 + 0 DO                 (points on screen.)
    ADRG @ I + 1/P
    COMP D2 !
    D2 @ MINI @ - COLS @ M*
    MAXI @ MINI @ - M/ POS ! DROP
    0 MINI @ - COLS @ M*
    MAXI @ MINI @ - M/ AXIS ! DROP
    [illegible]
LOOP ;
```

```
: PLOT                  (Rough plot of NSAMP data values)
." START ADDRESS = ?"   (in DRAM beginning from ADR0.)
IN-A ADR0 !
CR
DEGIN, P,
NSAMP @ ADR0 @ + ADR0 !
." MORE ?"
PAD 1 EXPECT
PAD @ 4E =
CR
UNTIL ;

(**************TERMINAL SIMULATION ROUTINE************)

CODE TERSIM     (Terminal simulation routine)
XSAVE STX,      (DASMA allows direct user interaction)
02   LDA,       (with the host computer by relaying any)
EFLA STA,       (input at the user terminal to the)
0   LDA,        (host data link, and any input at the host)
TAY,            (data link to them user terminal via a)
TAX,            (circular buffer.)
RPT1 STA,
WPT1 STA,
CNT1 STA,
CNT2 STA,
02   LDA,
STAT2 STA,
CLI,
BEGIN,
     CNT1 LDX,
     0= NOT IF,
          STAT1 LDA,
          02   AND,
          0= NOT IF,
               RPT1 LDX,
               BUFF1 ,X LDA,
               DAT1 STA,
               CNT1 DEC,
               INX,
               RPT1 STX,
          ENDIF,
     ENDIF,
     STAT1 LDA,
     01   AND,
     0= NOT IF,
          DAT1 LDA,
          BUFF2 STA,
          CNT2 INC,
     ENDIF,
```

```
        CNT2 LDA,
        0= NOT IF,
                STAT2 LDA,
                02   AND,
                0= NOT IF,
                        EFLG LDA,
                        01   CMP,
                        0= IF,
                           BUFF2 LDA,
                           CNT2 DEC,
                           05   CMP,
                           0= IF,
                              EFLG DEC,
                           ELSE,
                              01   LDA,
                              DAT2 STA,
                              EFLG INC,
                           ENDIF,
                        ELSE,
                           BUFF2 LDA,
                           CNT2 DEC,
                           01   CMP,
                           0= IF,
                              EFLG DEC,
                           ELSE,
                              DAT2 STA,
                           ENDIF,
                        ENDIF,
                ENDIF,
        ENDIF,
        EFLG LDA,
    0= UNTIL,
    SEI,
    XSAVE LDX,
    NEXT JMP,
    END-CODE

    (**********GENERAL COMMUNICATIONS ROUTINES**********)

    CODE TOCPSE         (Data in the accumulator is)
    PHA,                (transmitted over the host serial)
    BEGIN,              (link when the Tx-register empty)
    STAT2 LDA,          (flag [ACIA2] is set.)
    02   AND,
    0= NOT UNTIL,
    PLA,
    DAT2 STA,
    RTS,
    END-CODE
```

```
CODE TRMIT              (Puts the top value on the)
1   LDA,                (parameter stack into the)
SETUP JSR,              (accumulator and transmits)
N LDA,                  (it to the host.)
' TOCPSE JSR,
NEXT JMP,
END-CODE

FORTH

: TRANSMIT                      (Converts a four figure)
COMP DECIMAL DUP                (decimal number into an)
0<                              (ASCII encoded string and)
IF                              (transmits it over the)
    45 TRMIT NEGATE             (host data link.)
THEN
1000 /MOD 48 + TRMIT
100  /MOD 48 + TRMIT
10   /MOD 48 + TRMIT
          48 + TRMIT HEX ;

ASSEMBLER

CODE RDCHR              (Read character from circular)
XSAVE STX,              (buffer - ie character from)
RPT1 LDX,               (host.)
BUFF1 ,X LDA,
CNT1 DEC,
INX,
RPT1 STX,
XSAVE LDX,
STS,
END-CODE


CODE UP1                (Transmit code to ECLIPSE to)
0   LDA,                (initiate OPPS routine)
RPT1 STA,               (AC$UPLD.)
WPT1 STA,
CNT1 STA,
CLI,
55  LDA,
' TOCPSE JSR,
50  LDA,
' TOCPSE JSR,
4C  LDA,
' TOCPSE JSR,
```

```
    44   LDA,
    ' TOCPSE JSR,
    0D   LDA,
    ' TOCPSE JSR,
    NEXT JMP,
    END-CODE


    CODE UP2            (Wait for promt from host)
    02   LDA,           (space ? space)
    EFLG STA,
    BEGIN,
        CNT1 LDA,
        0= NOT IF,
                EFLG LDA,
                01   CMP,
                0= IF,
                   ' RDCHR JSR,
                   20   CMP,
                   0= IF,
                      EFLG DEC,
                   ELSE,
                      EFLG INC,
                   ENDIF,
                ELSE,
                   ' RDCHR JSR,
                   3F   CMP,
                   0= IF,
                      EFLG DEC,
                   ENDIF,
                ENDIF,
        ENDIF,
        EFLG LDA,
    0= UNTIL,
    NEXT JMP,
    END-CODE

    FORTH

    : UPLOAD                            (Upload DRAM data to)
    HEX                                 (the host, beginning)
    CR .' HEX STARTING ADDRESS ? ' CR   (at address ADR0 and)
    IN-4 CR                             (ending at address)
    ADR0 !                              (ADR1 - Under strict)
    CR .' HEX NO OF SAMPLES ? '  CR     (handshake conditions.)
    IN-4 CR
    ADR0 @ +
    ADR1 !
    UP1
```

```
    BEGIN
        UP2
        -9 D4 !
        BEGIN
             ADR0 @ I/P
             TRANSMIT
             1 ADR0 +!
             20 TRMIT
             D4 @ DUP 1 + D4 !
        0= UNTIL
        ADR0 @ ADR1 @ U<
        IF
            31 TRMIT
            0D TRMIT
            0
        ELSE
            30 TRMIT
            0D TRMIT
            1
        THEN
        ADR0 @ , 0D EMIT
    UNTIL
    TERMINAL ;


    : STAT-1
    FF DUP STAT1 C! STAT2 C!    (reset ACIA1 and ACIA2 and set)
    01 DUP STAT1 C! STAT2 C! ;  (baudclock in divide by)
                                (1 mode)

    : STAT-2                    (reset ACIA1 and ACIA2 and set)
    FF DUP STAT1 C! STAT2 C!    (baudclock in divide by)
    02 DUP STAT1 C! STAT2 C! ;  (1 mode)


    : BAUDRATE                  (set the baudrate of the RS 232)
    CR                          (links between DASHA and the user)
       ." ENTER NO. CORRESPONDING TO REQD. BAUD RATE " CR
       ." 0    9600' CR        (terminal and DASHA and the host.)
       ." 1    7200' CR
       ." 2    4800' CR
       ." 3    2400' CR
       ." 4    1800' CR
       ." 5    1200' CR
    ."   ? "
    IN-1 DUP 0 = IF STAT-1 D4 BDCLOCK C! ELSE
         DUP 1 = IF STAT-1 C5 BDCLOCK C! ELSE
         DUP 2 = IF STAT-2 EA BDCLOCK C! ELSE
```

```
      DUP 3 = IF STAT-2 D6 BDCLOCK C! ELSE
      DUP 4 = IF STAT-2 C5 BDCLOCK C! ELSE
      DUP 5 = IF STAT-2 A8 BDCLOCK C! ELSE
THEN THEN THEN THEN THEN THEN
DROP CR ;


: SAMP-FREQ       (Set ADC sampling frequency.)
CR
HEX
." ENTER NO CORRESPONDING TO REQD. SAMPLING FREQ. "
CR
." 0    6.7   KHZ" CR
." 1    7.0   KHZ" CR
." 2    7.5   KHZ" CR
." 3    8.1   KHZ" CR
." 4    8.9   KHZ" CR
." 5    9.6   KHZ" CR
." 6   10.5   KHZ" CR
." 7   11.7   KHZ" CR
." 8   13.2   KHZ" CR
." 9   15.1   KHZ" CR
." A   17.6   KHZ" CR
." B   21.1   KHZ" CR
." C   26.4   KHZ" CR
." D   35.1   KHZ" CR
." E   52.7   KHZ" CR
." F IS ZERO FREQ."
CR
."  ? "
IN-1 SAMPRT C! CR ;


: INIT-SAMP             (Set DASMA into convert-standby mode.)
9FFA C@ DROP ;


: RESET-SAMP            (Abort analog to digital conversion)
9FFB C@ DROP ;          (or abort convert-standby mode.)


: OPTIONS                                   (Sign on message.)
CR
." THE FOLLOWING ARE VALID COMMANDS : " CR CR
." SYSTEM" 8 SPACES ." UTILITY " CR CR
." RAMP-FREQ" 7 SPACES ." BAUDRATE"
." INIT-SAMP" 7 SPACES ." MEMTEST" CR
." RESET-SAMP" 6 SPACES ." DUMP" CR
```

```
    ." PLOT" C SPACES ." I/P " CR
    ." TERSIM" A SPACES ." O/P" CR
    ." UPLOAD" CR ;

    : MSGE
    CR
    ." FORTH BASED AUDIO FREQUENCY DATA AQUISITION SYSTEM "
    CR
    CR
    OPTIONS ;

    FINIS
```

APPENDIX F

DASHA CIRCUIT DIAGRAMS

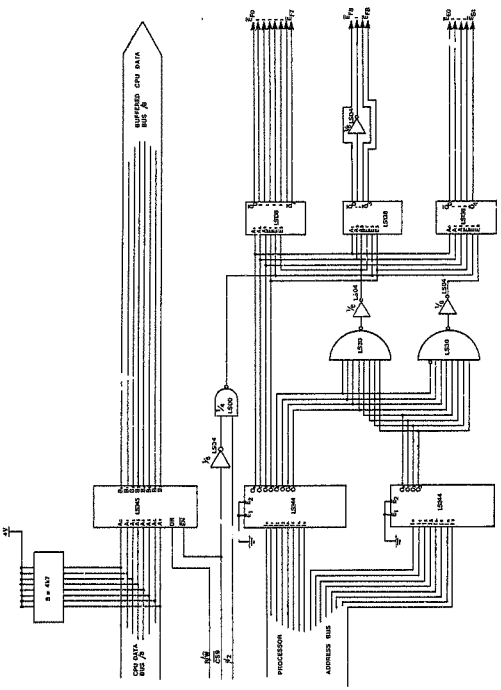


Figure F.1: Processor I/O buffers and address decoding circuitry.

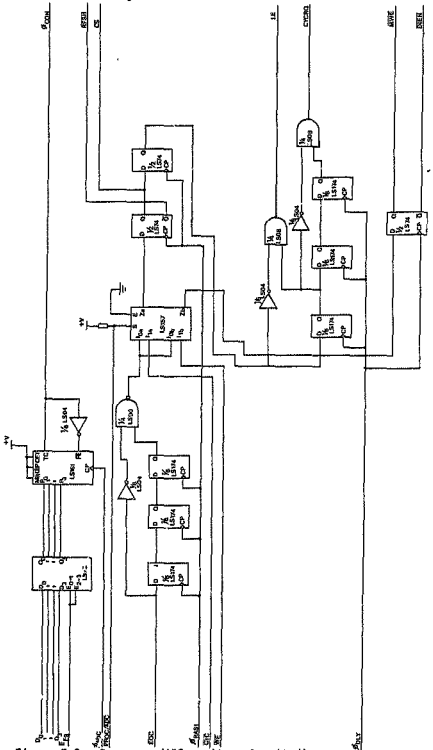


Figure F.2: Processor/ADC arbiter circuit diagram.

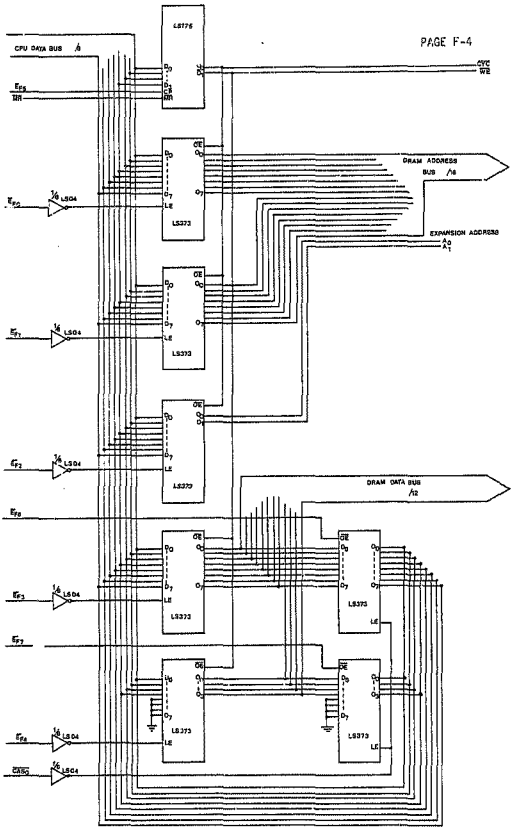


Figure F.3: Dynamic RAM interface circuit diagram.

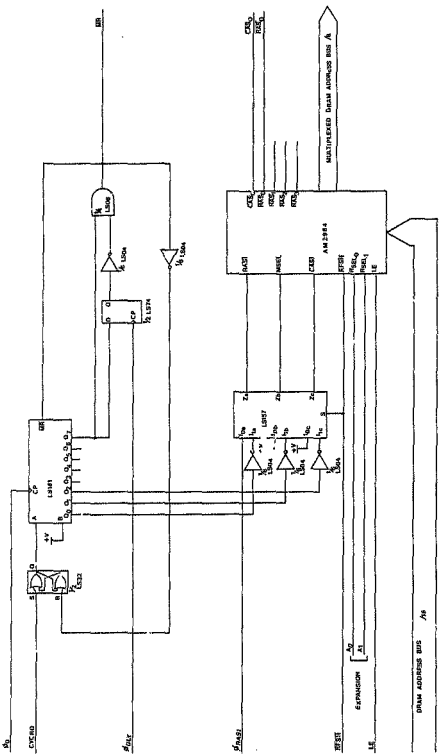


Figure F.4: Dynamic RAM controller circuit diagram.

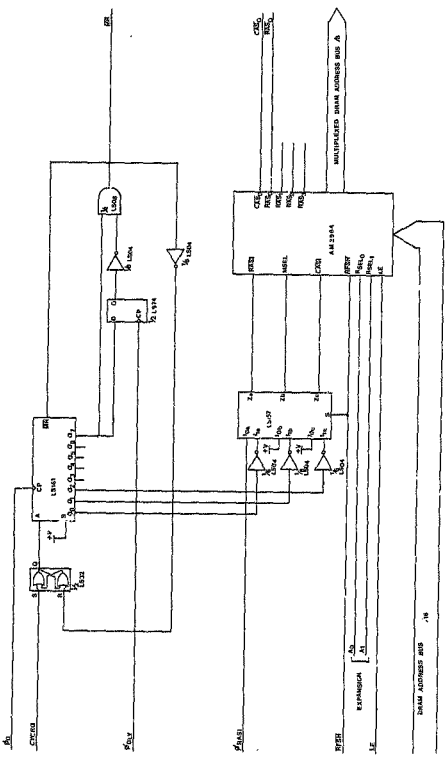


Figure F.4: Dynamic RAM controller circuit diagram.

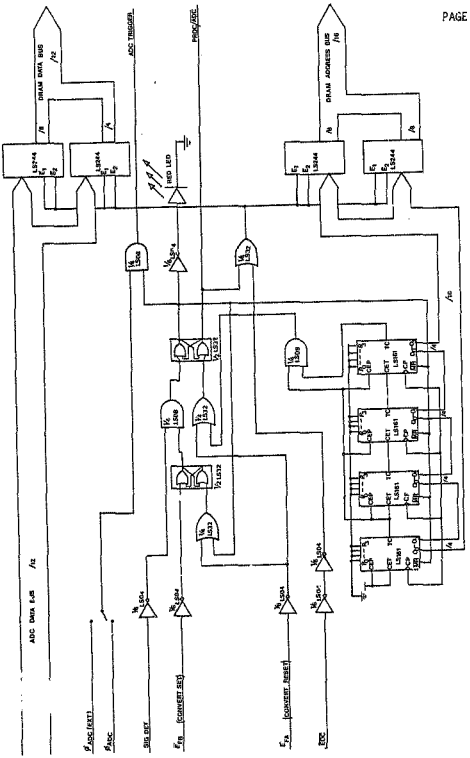


Figure F.5: Analog to digital converter controller.

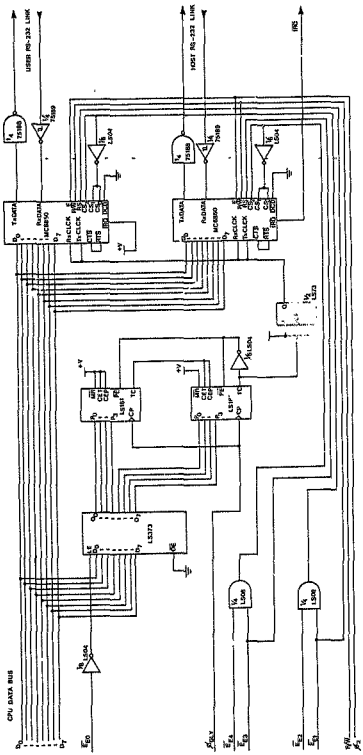


Figure F.6: Serial Input/Output interface circuit diagram.

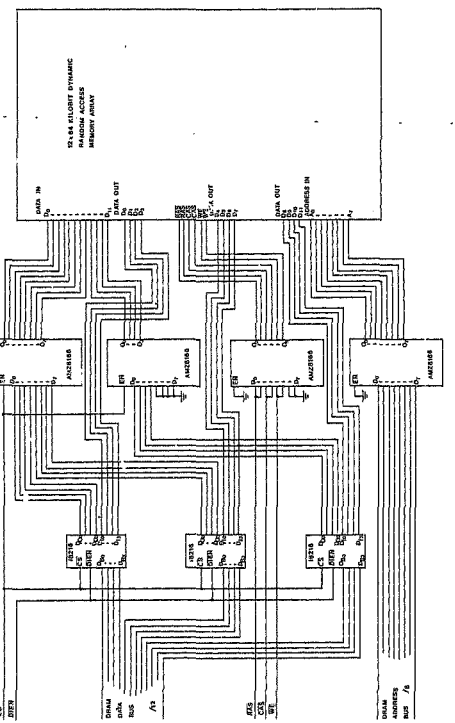


Figure F.7: 64K x 12 bit dynamic ram array.

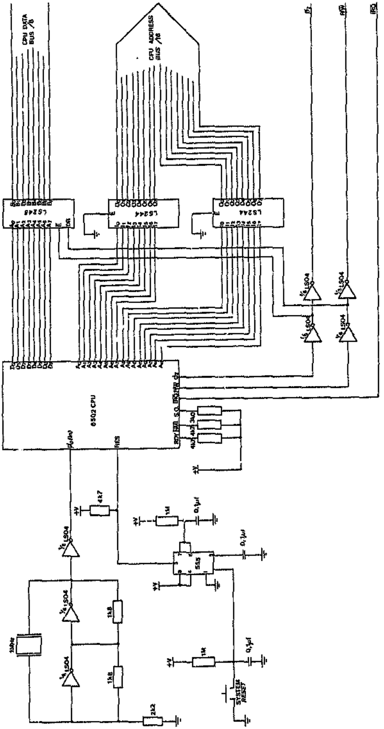


Figure F.8: 6502 based CPU circuit diagram.