TR 80 - 57

A REAL TIME FAST FOURIER TRANSFORM ANALYSER

A thesis submitted in fulfilment of the requirements for the degree of Master of Science at

Rhodes University, Grahamstown

by

JOHN STANLEY FISHER

Supervisor : Professor J.A. Gledhill

November 1979

ACKNOWLEDGEMENTS

This project, although largely practical, has required the full facilities available from the Rhodes Physics Department and the Rhodes Antarctic Research Program. I would firstly like to thank Professor J.A. Gledhill for being my supervisor on this project, and for his continual encouragement throughout the four years of work. I am also greatly indebted to Allon Poole for his technical and theoretical guidance, including the checking of the first draft of this thesis, and to whom the initial idea of using FFT in this context must be attributed. Most of my electronics advice was obtained from Clive Way-Jones whose method of indirect guidance by the suggestion of possible circuit solutions, was greatly appreciated. To all the other members of the Antarctic Research Group, and the Physics Department staff, I must pass on my thanks for the very pleasing environment they have created for research work.

Finally, I must thank my wife for the typing of the first draft of this thesis, and Maureen Jackson for her very efficient and professional typing of the final copy.

CONTENTS

ABSTRACT

From the requirements of the Ionosonde digitisation project, undertaken by Rhodes University Antarctic Research Group, it was decided to use the Fast Fourier Transform to compute the spectrum analysis. Several FFT algorithms are reviewed and properties discussed, and the Cooley Tukey algorithm chosen for utilization. The hardware implementation of this algorithm, and the microprogram control of the whole system are discussed in detail, and such design aspects that required computer simulation are also treated in detail. The final testing of the analyser is shown, and includes a test using data from an ionosonde sounding. The conclusions contain details of extensions to the analysers present operation, required by plans to place the whole Chirpsounder under microprocessor control.

1. INTRODUCTION

The Rhodes University Antarctic Research Group operate two Chirpsounder Ionosondes on a radio link between the South African Antarctic base, SANAE and Grahamstown. The object of this radio link is to measure the hourly, daily and seasonal changes of the ionised layers above the Southern Ocean, for solar-terrestrial research, and radio propagation predictions.

The prime parameter studied is the virtual height of each layer as a function of frequency, and this is obtained in a simple ionosonde by transmitting a pulse of fixed frequency radio waves, and timing the echo return. The radio waves are reflected by the ionised layers selectively, dependant upon their frequency, and thus return back to earth as a signal able to be detected by a radio receiver. By incrementing the frequency progressively, a plot of virtual height against frequency can be obtained, this plot being termed an 'ionogram'.

Research work, using ionograms as basic data, has been active for many years and has solved many problems associated with radio propagation, but as our knowledge has increased, the requirement for another dimension to the data has become necessary to explain problems and clarify theories. The angle of arrival of a received signal is a parameter that would be of immense value to researchers in ray tracing, and general oblique incidence work, but which, as yet, is not readily available from present equipment. This, together with Doppler shift measurements, would according to Rash 1978 (13) greatly increase the progress of research into travelling ionospheric disturbances.

The difficulty with most present Ionosondes, in respect of angle of arrival and Doppler shift measurement, is that the phase and magnitude of the incoming echoes or signals is lost in the data processing, so that the ionogram can only display the delay of the signals. The present Chirpsounder Ionosonde also fails in this respect, but due to its unique technique of ionogram production, the phase and magnitude of the incoming signals is preserved until the final stages of processing and by modifications to the signal analysis method, the phase could be extracted simultaneously with the magnitude. The signal analysis modifications are the subject of this dissertation, and the FFT analyser will essentially replace the final output stage of the Chirpsounder Ionosonde. A requirement for this analyser, is that it must output the data in digital format to enable a computer to do further processing, and in the future, to enable the data to be transmitted from the Antarctic Base directly to Rhodes University via a telex link.

The principal difference between the Chirpsounder Ionosonde and the more simple pulse ionosondes, is that a continuous sweep of frequency is transmitted rather than a pulse. The incoming received echo is then mixed with the instantaneous transmitter frequency (which will have advanced during the time the echo travelled to the ionosphere and back), and the resultant difference frequency will be proportional to the virtual height of the reflecting ionised layer. Since the mixing is coherent, the phase of the incoming echoes will be a meaningful quantity and will be present at the output of the receiver.

Thus, the information to be retrieved from the receiver signal consists of varying frequencies or tones whose frequency and amplitude indicate the virtual height and amplitude of the echo respectively. A spectrum analysis technique is therefore required to separate the tones, and the present chirpsounder employs an analogue spectrum analyser which outputs the data for recording on photographic film. In order to measure the angle of arrival of an incoming signal, two antennas must be used followed by two phase matched receivers, in order that the relative phase between the two antennas may be determined. The analyser must, therefore, be able to transform two sets of time data and complete the transforms in real time, leaving time available for further computation by a digital computer using the output transforms.

The Fast Fourier Transform (FFT) algorithm for spectral analysis, is the method chosen to replace the present analogue technique, and it fulfills all the requirements previously stated providing that it is utilized correctly. The final stipulation implies that there are several ways of implementing the algorithm, and the relative speed, for example, will differ between methods. Chapter 2 deals with the different FFT algorithms, but Chapter 3 decides how to implement the algorithm for our particular case. The expansion of Chapter 2 to include details of algorithms other than the one implemented by Rhodes, was intended to be of use to an FFT system designer whose overall requirements dictate a different approach. The hardware approach introduced in Chapter 3, is expanded in Chapters 4, 5 and 6 and particular problems, for instance, associated with the arithmetic, are discussed in detail.

The FFT is not a straight forward operation and since this unit will eventually be installed at both the Grahamstown and the SANAE chirpsounder stations, it will have to be operated and serviced by personnel possibly unfamiliar with the theory. Therefore, every effort must be made to make it simple but efficient, and to include built-in test procedures to facilitate fault finding. Documentation is obviously of paramount importance, and this thesis may be used as a manual to the unit. The Appendices contain circuit diagrams and layouts, which together with the detailed test and set-up procedures found in Chapter 7, will provide the information necessary for the operation of the analyser.

The computer programs used in the development of the analyser are included in appendix 1 so that future workers, possibly extending the capabilities of the present unit, may make use of them. The design work for the hardware was started 3 years ago and was designed around integrated circuits that were available at that time, and due to advances in technology over this period, there are no doubt, quicker and cheaper methods of producing the same results. In my conclusions, I mention possible areas of future study which might improve or extend the usefulness of the unit, but its present working form completely fulfills the design specification.

2. THE FFT ALGORITHMS

The object of this chapter is to show the development of the FFT from the continuous Fourier Transform (FT) pointing out the limitations in the equivalence between the two, and concluding with the properties of the FFT that are of use to the system or software designer.

The FFT does not refer to a single algorithm, but to a complete family, and it is the different properties exhibited by the separate members which gives the FFT its wide use in both hardware and software implementations. Optimisation of computation time against system complexity and cost is achieved by careful choice between algorithms.

The differing properties stem from the Discrete Fourier Transform (DFT) which is derived from the FT to deal with sampled data of the form required for digital computation. The limitations of the FFT compared with the FT are caused by sampling in both the time and frequency domains and truncation in the time domain as defined by the DFT, and therefore the development from DFT to FFT is an exact process.

Using the diagrammatic method of Brigham 1974 (1) the DFT will be defined, and sampling and truncation examined in respect of their effect in the frequency domain. The method of factorising the DFT to form the Cooley Tukey 1965 (2) algorithm, will also follow the notation of Brigham, and from this the other algorithms will be defined and compared as "signal flow graphs".

2 - 1 Discrete Fourier Transform

The transformation between the time domain signal h(t) and the frequency domain signal H(f) is defined by the FFT :

$$H(f) = h(t)e^{-j2\pi ft}\,dt \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \; 1$$

and the inverse transformation by :

$$h(t) = H(f)e^{j2\pi ft}\,df \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \; 2$$

noting that both h(t) and H(f) are defined between $+\infty$ and $-\infty$.

For the transformation to be undertaken in a digital computer, both h(t) and H(f) will have to be truncated to finite limits, and be represented by their respective sampled waveforms h(kT) and H(n/NT) where NT is the duration of the time domain signal and T the time between each of the N samples. Both k and n are integers in the range 0 to N - 1.

The first limitation of the equivalence of FT and FFT is known as aliasing, and is due to the incorrect spacing of the samples in the time domain with respect to the maximum frequency contained in the signal. The sampling theorem states that if a signal contains no energy at a frequency greater than a certain frequency f_c, then the continuous function h(t) can be exactly represented by its sampled values obtained by sampling at a rate of $2f_c$.

Thus if the sample rate is lower than that defined above, the sampled waveform cannot be held to represent the original h(t) and if an FFT is carried out on this sampled waveform, the resulting transform will only represent the frequency content of the modified h(t).

Aliasing can best be seen by examining the sampling function which is defined as :

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT)$$

and its FT as :

$$S(f) = \frac{1}{T} \sum_{n=-\infty}^{\infty} \delta(f - n/T)$$

Diagrammatically this is shown in Fig. 2 - 1.

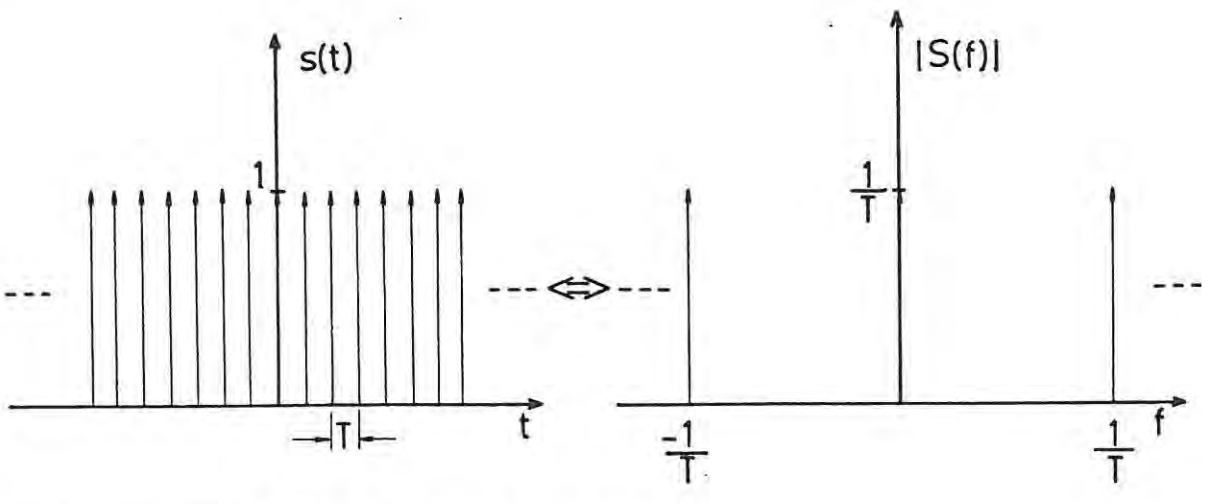


Fig. 2 - 1. Time domain sampling function.

It can be seem that as the spacing of the samples in the time domain increases, then the frequency domain impulse functions become closer together, also noticing that both time and frequency domain series, extend to $\pm\infty$. Therefore, any time signal h(t) sampled by s(t) (i.e. multiplied by s(t)) will produce a transform which is the result of convolution of H(f) and S(f) (since multiplication in the time domain is analagous to convolution in the frequency domain), and will thus appear as the transform H(f) repeated at each of the impulse functions of S(f). Fig. 2-2 shows a general signal h(t) and its transform, followed by h(t) x s(t) and its transform, clearly detailing the repetition in the frequency domain.

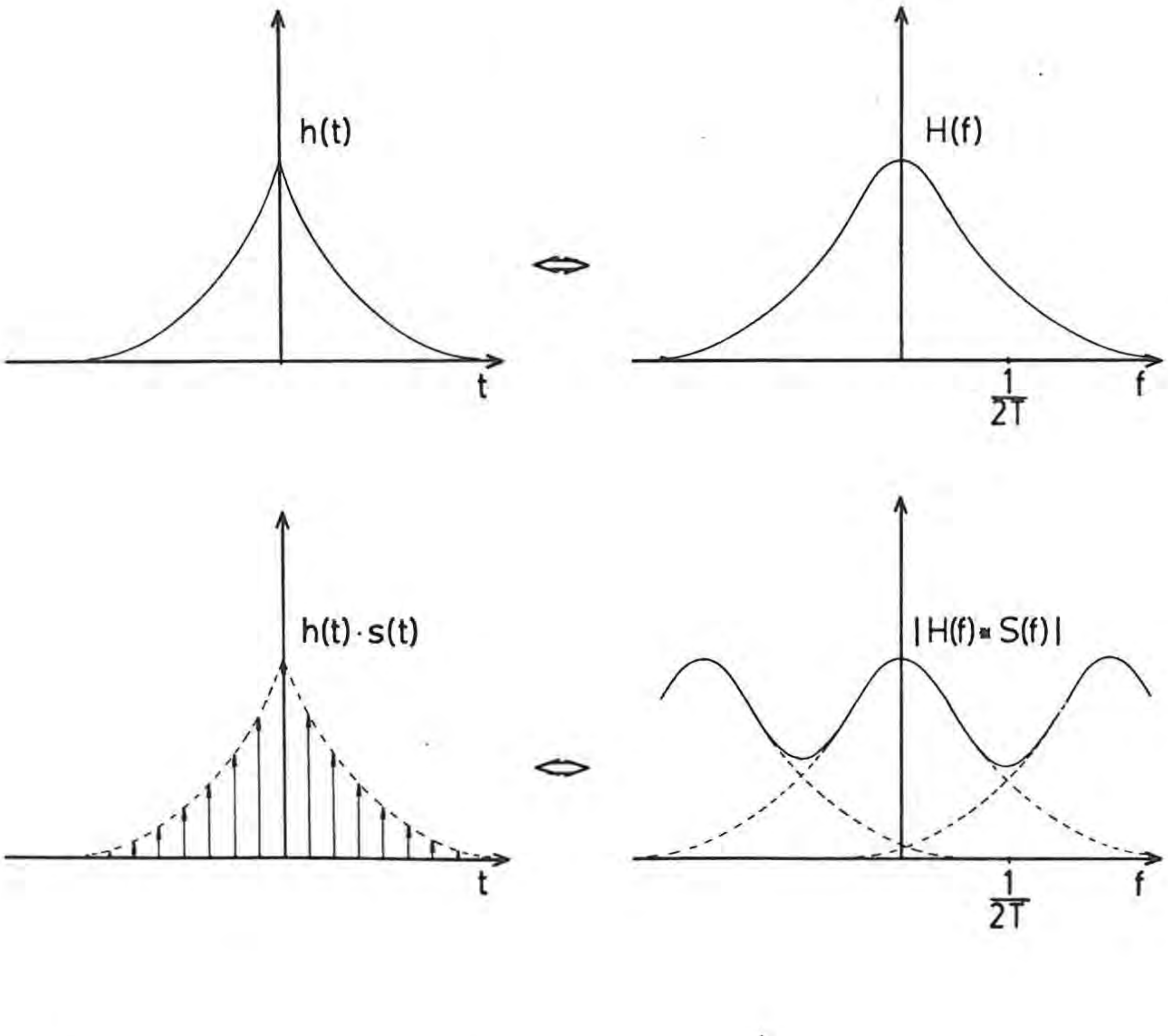


Fig. 2 - 2. Incorrectly sampled time series.

Since h(t) has frequencies above $\frac{1}{2T}$, the sampled transform contains those frequencies reflected back on top of the lower frequencies.

Therefore, before sampling, the time domain signal must be filtered (band limited) to contain only frequencies up to half the sample frequency, or conversely, the signal must be sampled at a rate of twice its highest frequency.

Proceeding then with this incorrectly sampled example, the next step towards the DFT is that the infinite number of samples be truncated to a finite size. The "tophat" function is used for this purpose to take N samples of the time signal and is therefore of length NT. Fig. 2-3 shows the "tophat" function, and the sin x/x waveform of its transform.

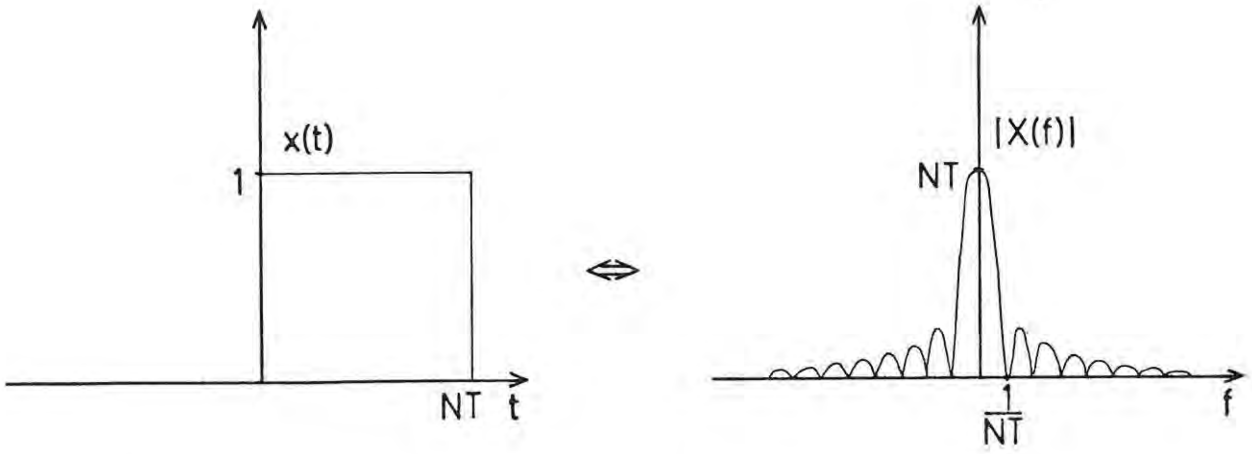


Fig. 2 - 3. "Tophat" truncation waveform.

By convolution of the "tophat" transform X(f) with the sampled signal transform of Fig. 2-2, the effect of the sin $x/_x$ waveform can be seen as "broadening and rippling". Fig. 2-4 shows this convolution.

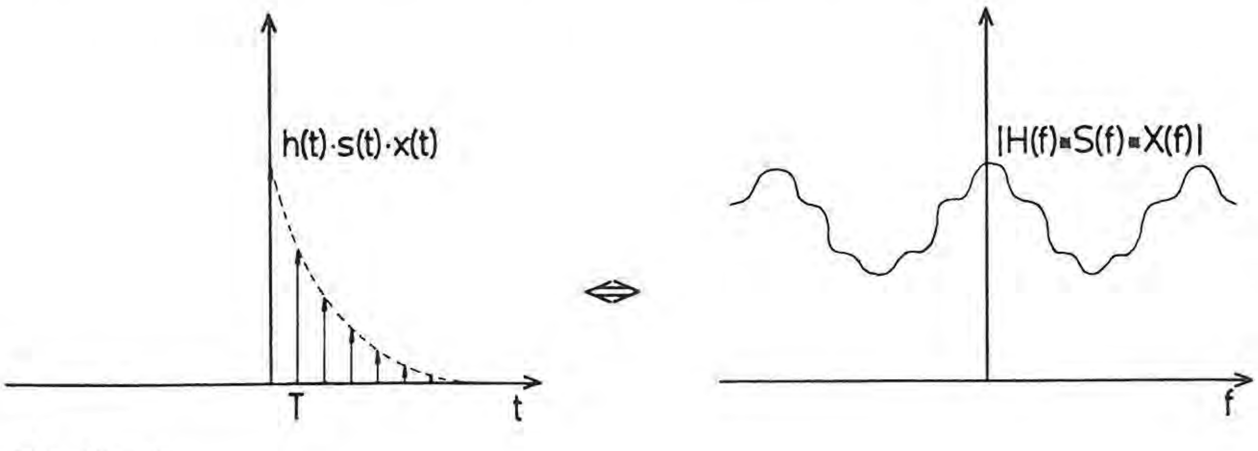


Fig. 2 - 4.

The final step in the progression to the DFT is to sample the continuous waveform in the frequency domain. Fig. 2-5 shows the time domain signal that results in the sampling waveform in the frequency domain.

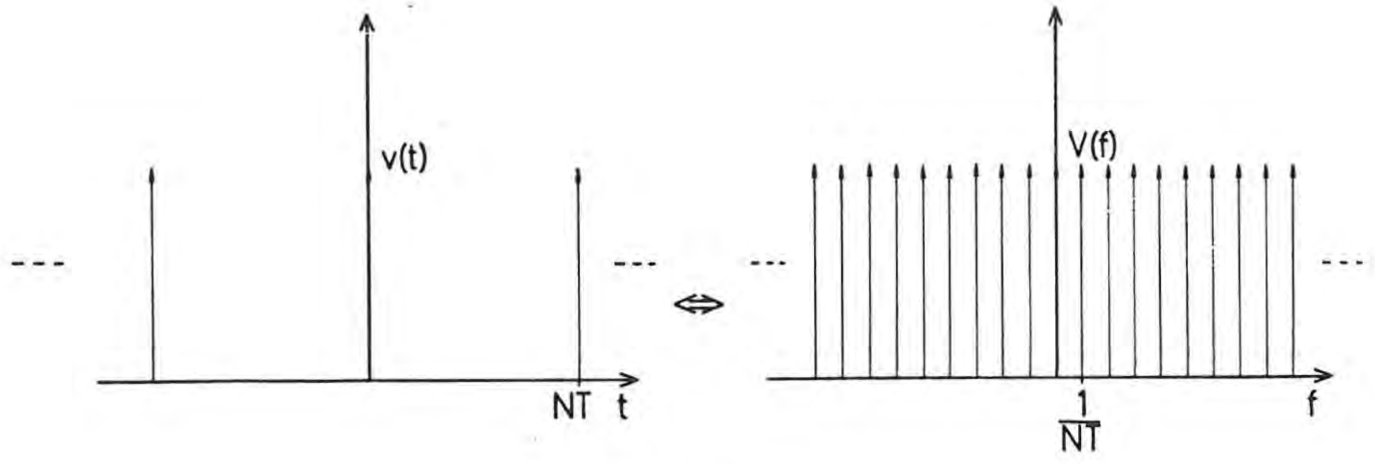


Fig. 2 - 5. Frequency domain sampling function.

The result of this frequency domain sampling is to form a periodic repetition of the truncated time domain signal. The combination of Fig. 2-4 and Fig. 2-5 to form the DFT of our example signal h(t) is shown in Fig. 2-6.

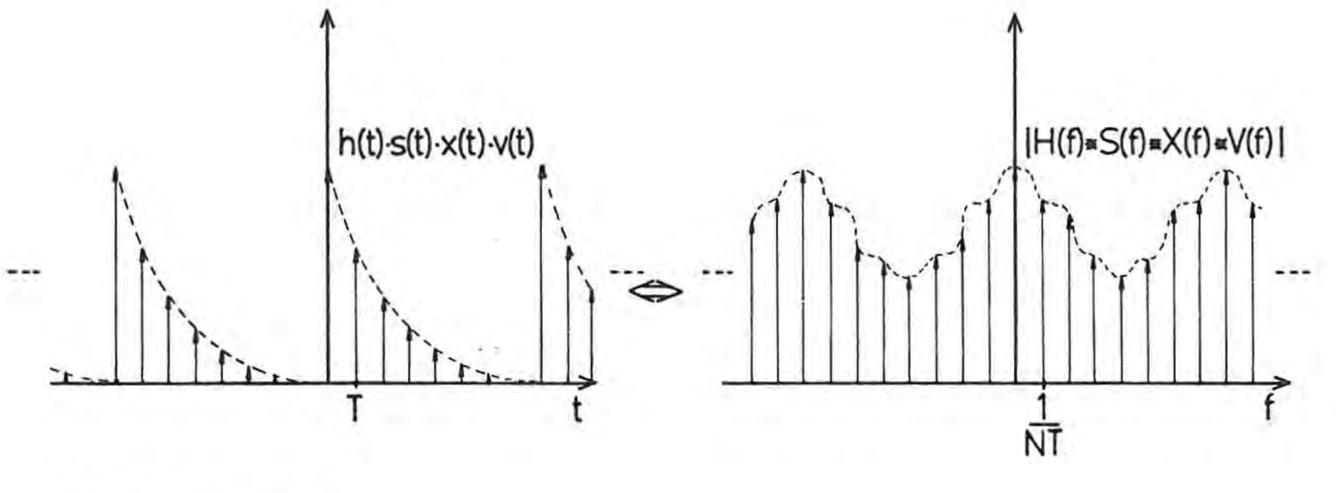


Fig. 2 - 6. DFT.

Comparison of the first transform in Fig. 2-2 and Fig. 2-6 indicates the degeneration of the ideal FT to the DFT, remembering that in this example we have violated the sampling theorem.

If we remove the aliasing caused by incorrect sample rate, we have only the sin x/x waveform degenerating the transform, and since the frequency domain is also sampled, the presence of this degeneration is not always apparent.

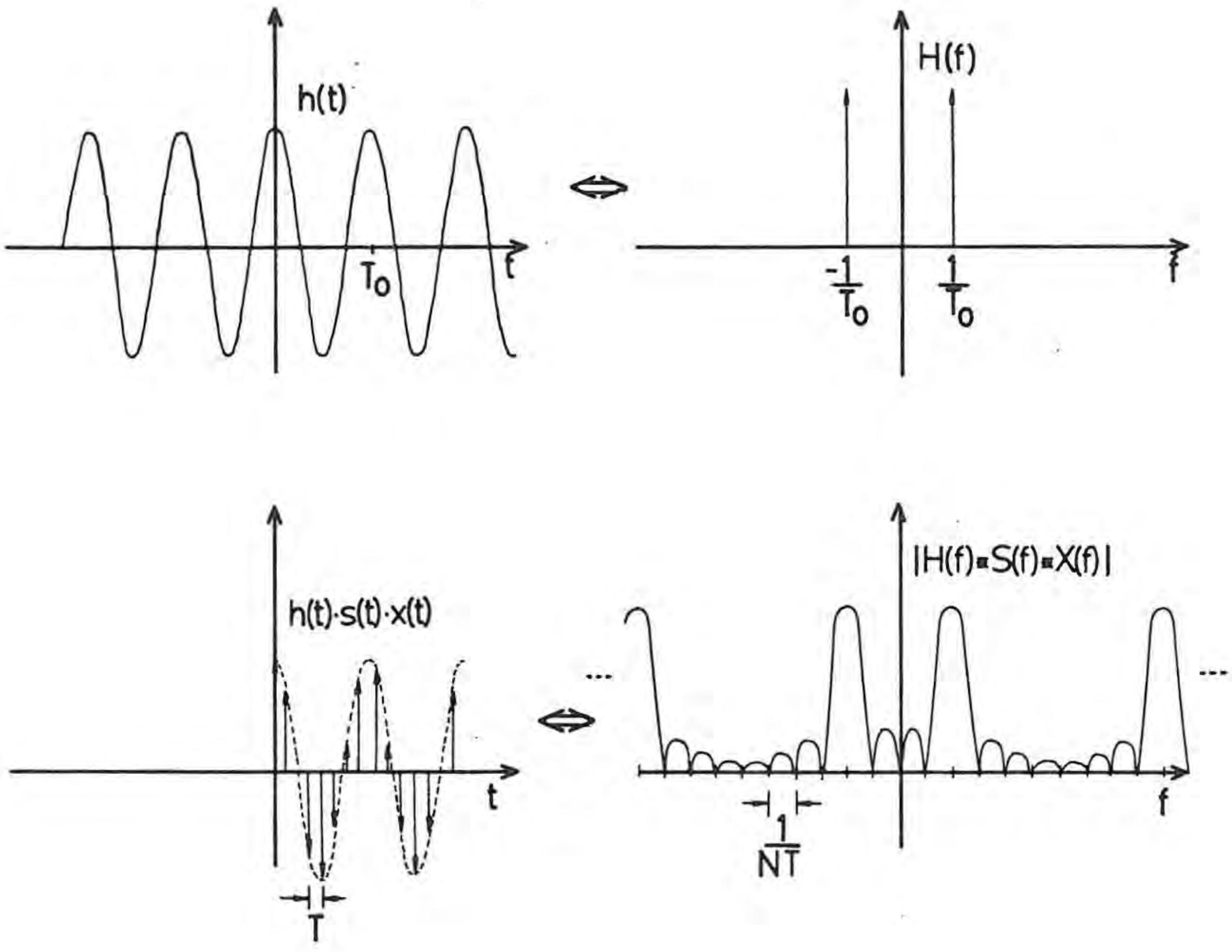


Fig. 2 - 7. Correctly sampled time series.

For example, if we take a cosine wave h(t) which we have truncated and sampled as described in the previous example, we obtain the transform sequence shown in Fig. 2-7. Notice that the cosine wave chosen has exactly two cycles within the "tophat" truncation function, and that the zeros in the final transform are regularly spaced $\frac{1}{NT}$ apart. Recalling from Fig. 2-5 that the sampling waveform for the frequency domain **V**(f) also has spacing of $\frac{1}{NT}$, the final DFT of the cosine wave appears as Fig. 2-8 since all the samples except two have fallen on the zeros of the sin x/x function.

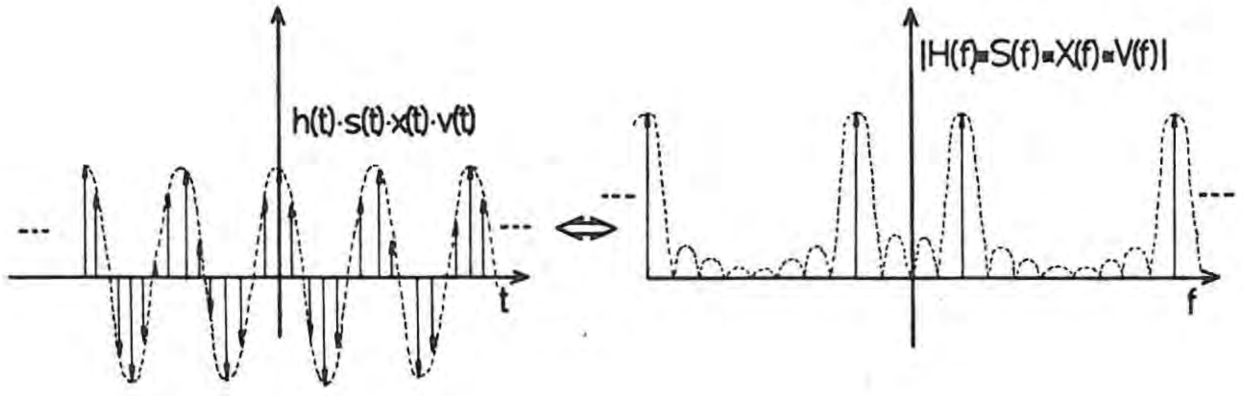


Fig. 2 - 8.

This is a very special case since in general the signal does not have an integral number of cycles within the NT samples, and therefore, in the general case, the V(f) samples will not fall on the zeros of the sin x/x waveform. The side lobes that thus appear are termed "leakage", and many authors have developed "low leakage windows", (in other words, truncation waveforms other than the "tophat" used in the above examples) to minimise the size of the side lobes.

Harris 1978 (8) sums together all the properties of these windows with respect to their effect on the detection of harmonic signals in the presence of broad-band noise. However, in the case of this study it was not felt necessary to explore the use of "windows", and the only further mention of them will be in Chapter 8 where possible future research will be discussed.

2 - 2 Factorising the DFT

The DFT is defined as :

$$H\left(\frac{n}{NT}\right) = \sum_{k=0}^{N-1} h(kT)\, e^{-j2\pi nk/N} \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad 6$$

Where, n = 0,1, ... N-1.

For convenience of notation we let $\frac{n}{NT} \equiv n$, $kT \equiv k$ and let $W = e^{-j2\pi/N}$

therefore, $$H(n) = \sum_{k=0}^{N-1} h(k)\, W^{nk} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 7$$

This calculation requires N^2 complex multiplications and $N(N-1)$ complex additions, which is a large task even for a fast computer, and so the FFT aims at reducing the number of multiplications and additions required.

If we assume that $N = 2^{\ell}$ where ℓ is an integer, then for the transform to be carried out in a binary computer, the indices n and k are best expressed in binary notation.

i.e. $$n = 2^{\ell-1} n_{\ell-1} \ldots +2^2 n_2 + 2n_1 + n_0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 8$$

$$k = 2^{\ell-1} k_{\ell-1} \ldots +2^2 k_2 + 2k_1 + k_0 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 9$$

where $n_i = 0$ or 1

and $k_i = 0$ or 1

Eq. 7 then becomes :

$$H(n_{\ell-1} \ldots n_1, n_0) = \sum_{k_0=0}^{1} \sum_{k_1=0}^{1} \sum_{k_{\ell-1}=0}^{1} h(k_{\ell-1} \ldots k_1, k_0) \quad \ldots\ldots \quad 10$$

$$W^{(2^{\ell-1}k_{\ell-1} \ldots +2k_1+k_0)\,(2^{\ell-1}n_{\ell-1} \ldots +2n_1+n_0)}$$

splitting the W term, results in the two forms of this basic algorithm.

By separating the values of k_{ℓ} the "Decimation in time" or Cooley-Tukey 1965 (2) algorithm is developed.

Also, by separating the values of n_{ℓ} the "Decimation in frequency" algorithm is developed.

As mentioned at the beginning of this section, we will develop the "Decimation in time" algorithm, therefore Eq. 10 becomes :

$$H(n_{\ell-1},\ldots n_1,n_0) = \sum_{k_0=0}^{1}\sum_{k_1=0}^{1}\;\sum_{k_{\ell-1}=0}^{1} h(k_{\ell-1},\ldots k_1,k_0)$$

$$W^{(2^{\ell-1}n_{\ell-1}\ldots+2n_1+n_0)\;2^{\ell-1}k_{\ell-1}}$$

$$\ldots\; W^{(2^{\ell-1}n_{\ell-1}\ldots+2n_1+n_0)2k_1}$$

$$W^{(2^{\ell-1}n_{\ell-1}\ldots+2n_1+n_0)k_0} \quad \ldots\ldots\ldots\ldots\ldots 11$$

By multiplying out the exponent of W in each case, terms appear of the form $W^{2^{\ell}(2^{\ell-2}n_{\ell-1}k_{\ell-1})} = 1$ since $2^{\ell} = N$ and $W^{N} = e^{\frac{-j2\pi N}{N}} = 1$

This reduces the factors involved in the exponents of W considerably and enables the separation of the total sum into ℓ individual summations. Thus, if the input data is defined as $h_0(K_{\ell-1},\ldots k_1,k_0)$ then the result of summation over $k_{\ell-1}$ will be :

$$h_1(n_0,k_{\ell-2},\ldots k_1,k_0) = \sum_{k_{\ell-1}=0}^{1} h_0(k_{\ell-1},\ldots k_1,k_0)\; W^{2^{\ell-1}n_0k_{\ell-1}} \quad \ldots\ldots 12$$

noticing that $k_{\ell-1}$ has been replaced in h_1 by n_0,

h_2 can similarly be defined as :

$$h_2(n_0,n_1,k_{\ell-3},\ldots k_1,k_0) = \sum_{k_{\ell-2}=0}^{1} h_1(n_0,k_{\ell-2},\ldots k_1,k_0)W^{(2n_1+n_0)2^{\ell-2}k_{\ell-2}} \quad \ldots\ldots 13$$

and finally :

$$H(n_0,n_1,\ldots n_{\ell-1}) = h_{\ell}(n_0,n_1,\ldots n_{\ell-1})$$

$$= \sum_{k_0=0}^{1} h_{\ell-1}(n_0,n_1,\ldots k_0)W^{(2^{\ell-1}n_{\ell-1}\ldots+n_1+n_0)k_0} \quad \ldots\ldots 14$$

By expanding the W term in $h_1(n_0,\ldots k_1,k_0)$,

$$W^{2^{\ell-1}n_0k_{\ell-1}} = e^{\frac{-j2\pi N}{N2}} \begin{array}{cc} n_0 & k_{\ell-1} \\ \left[\begin{matrix} 0 \\ 0 \\ 1 \\ 1 \end{matrix}\right. & \left.\begin{matrix} 0 \\ 1 \\ 0 \\ 1 \end{matrix}\right] \end{array} = e^{-j\pi\begin{bmatrix}0\\ \\1\end{bmatrix}} \quad \ldots\ldots\ldots\ldots 15$$

$$= \pm 1$$

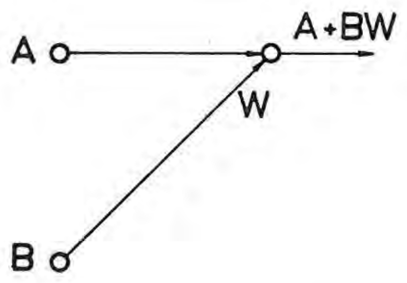
Writing out each term for h_1 ( )

$$h_1(n_0,\ldots k_1,k_0) = h_0(0,\ldots k_1,k_0)W^{2^{\ell-1}(n_0,0)} + h_0(1,\ldots k_1k_0)W^{2^{\ell-1}(n_0,1)} \quad \ldots\ldots 16$$

Each of the individual summations can be written similarly, split into two blocks of data, the W term for the first block always being 1 and that for the second, generally being complex. Therefore, the arithmetic operation carried out at each stage can be expressed as $h = A + BW$ where A is the data addressed by $k = 0$ and B by $k = 1$ and W by the requisite multiplying factor.

A method of graphically portraying the arithmetic procedure of the FFT, is the signal flow graph. Each elemental operation of $A + BW$ is termed a node and the group of operations which convert h_r into h_{r+1} are termed an array.

A node is shown thus :



The signal flow graph is best seen with specific values rather than the general case.

Consider then, the case where $N = 8 = 2^3$ $\ell = 3$ which indicates that there will be 3 arrays. This signal flow graph is shown in Fig. 2-9.

A property of this algorithm, although not evident from the signal flow graph, is that the output data H(n) has had its sequence changed. From Eq. 14, it can be seen that the significance of the binary digits has been reversed i.e. n_0 is the most significant bit and $n_{\ell-1}$ is the least significant. This bit reversal of the address can also be seen in the powers of the W terms in the last array, and the same W term sequence is used in each array, although the last array is the only one that proceeds completely through the sequence.

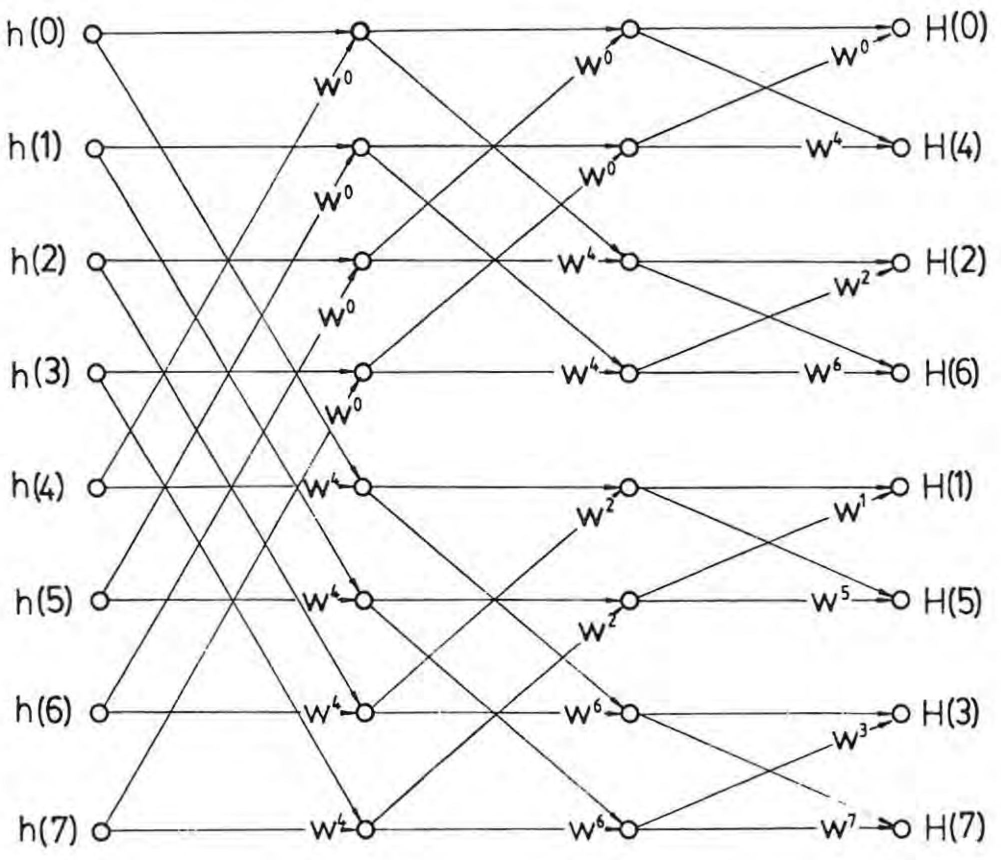


Fig. 2 - 9. Cooley Tukey Algorithm.

The number of complex multiplications in the FFT flow graph of Fig. 2-9 can be expressed as N x ℓ, whereas the original DFT equation contained N^2. Similarly Fig. 2-9 contains N x ℓ complex additions, whereas the DFT contained N(N-1). This is a vast improvement on the DFT, but further examination of the arithmetic of each array enables extra reduction of complex multiplication to Nℓ/2.

This is achieved due to the equal magnitude but differing sign of W terms in the flow diagram, which operate on the same data. In Fig. 2-9, for example, h(0) and h(4) are used twice to calculate the next array values : in the first case h(4) is multiplied by W^0 and in the second case by W^4.

However, since N = 8 in this case,

$$W^4 = W^{(0 + \frac{N}{2})}$$
$$= W^0 \times W^{\frac{N}{2}}$$
$$= W^0 \times (\text{Cos } \frac{2\pi N}{2N} - j \text{ Sin } \frac{2\pi N}{2N})$$
$$= W^0 \times (-1)$$

Therefore $\quad W^4 = -W^0$

Similarly, $W^6 = -W^2$, $W^5 = -W^1$ and $W^7 = -W^3$. This means that the product BW need only be calculated once, and then this product added and subtracted from A. Thus the final arithmetic can be expressed as $A \pm BW$.

Summarizing the properties of the Cooley Tukey algorithm portrayed in Fig. 2-9, we have :

(a) Ordered input data sequence, but bit reversed output data sequence.

(b) Uniform arithmetic operation throughout.

(c) W term sequence is bit reversed but identical in each array.

(d) Data separation for each node follows a strict modulo-two sequence.

(e) Results of calculations can replace the original operands in the data sequence, since data points are only used once in calculating the corresponding points in the new array. (In place arithmetic).

2 - 3 Other FFT algorithms

By bit reversing the data input sequence of the Cooley Tukey FFT, a different set of properties appears. Fig. 2-10 shows this form of the algorithm, and as can be seen, the data output sequence is ordered, the arithmetic operation is uniform throughout, but the W-term sequence has become ordered and follows a different sequence in each array. The W-terms are related in the same way as for Fig. 2-7 i.e. $W^4 = -W^0$,

$W^6 = -W^2$, $W^5 = -W^1$ and $W^7 = -W^3$ and therefore, the arithmetic is still $A \pm BW$ throughout.

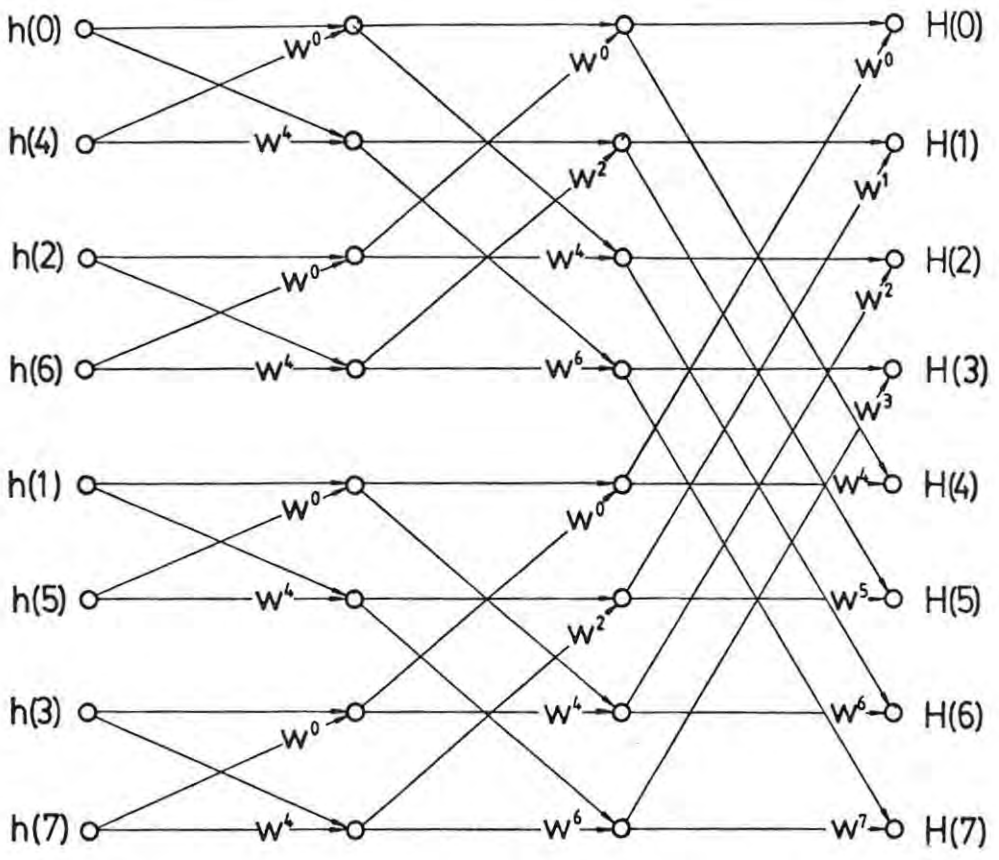


Fig. 2 - 10. Bit reversed input Cooley Tukey Algorithm.

The Sande Tukey or 'Decimation in Frequency' algorithm, as mentioned earlier, is developed in an identical manner to the Cooley Tukey algorithm, except that the n_ℓ term of the W power in equation 10 is split up to form the separate summations. Fig. 2-11 shows the signal flow graph for this algorithm, and indicates that it has ordered input data sequence, but bit reversed data output. The W-terms are related in the same manner as for the Cooley Tukey algorithm, and are seen to be ordered and follow a different sequence in each array. However, the main change is that the arithmetic operation is no longer $A \pm BW$, but has the form of $A + B$ and $W(A - B)$ for the two respective new values.

There is a form of the Sande Tukey algorithm that has bit reversed input data sequence and ordered output, formed in the same way as its Cooley Tukey counterpart. It exhibits the arithmetic of $A + B$ and

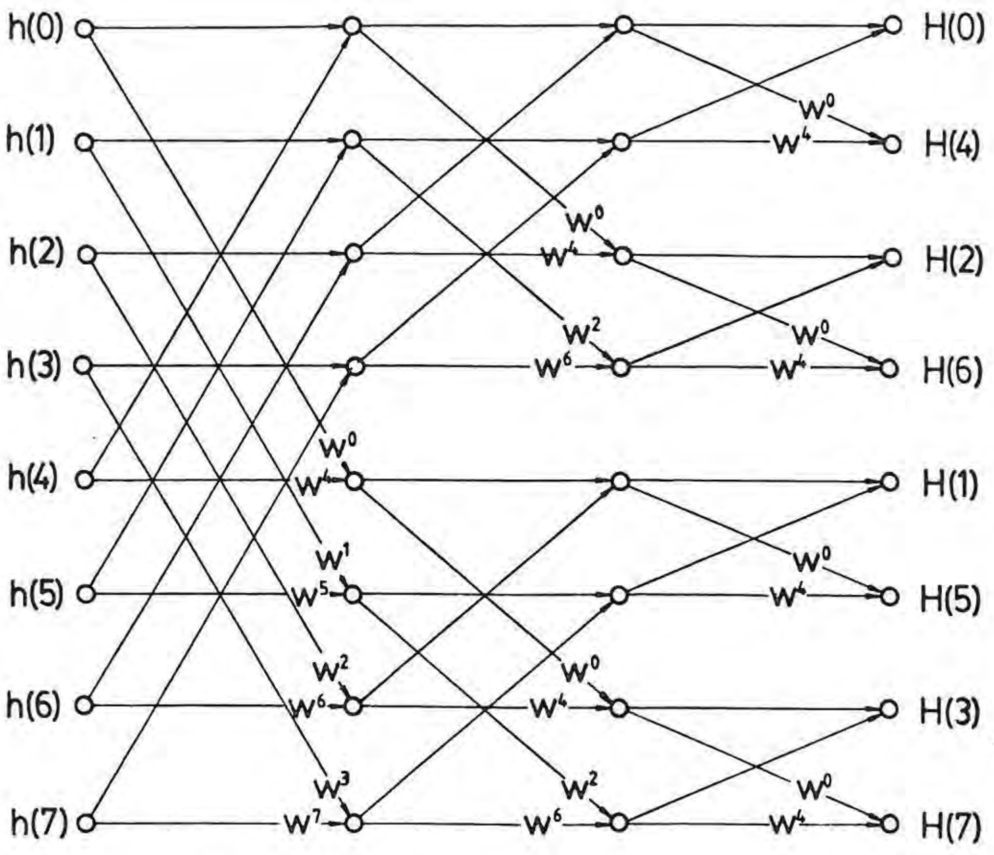


Fig. 2 - 11. Sande Tukey Algorithm.

W(A - B) but has a bit reversed W-term sequence that is the same for each array. This algorithm is shown in Fig. 2-12.

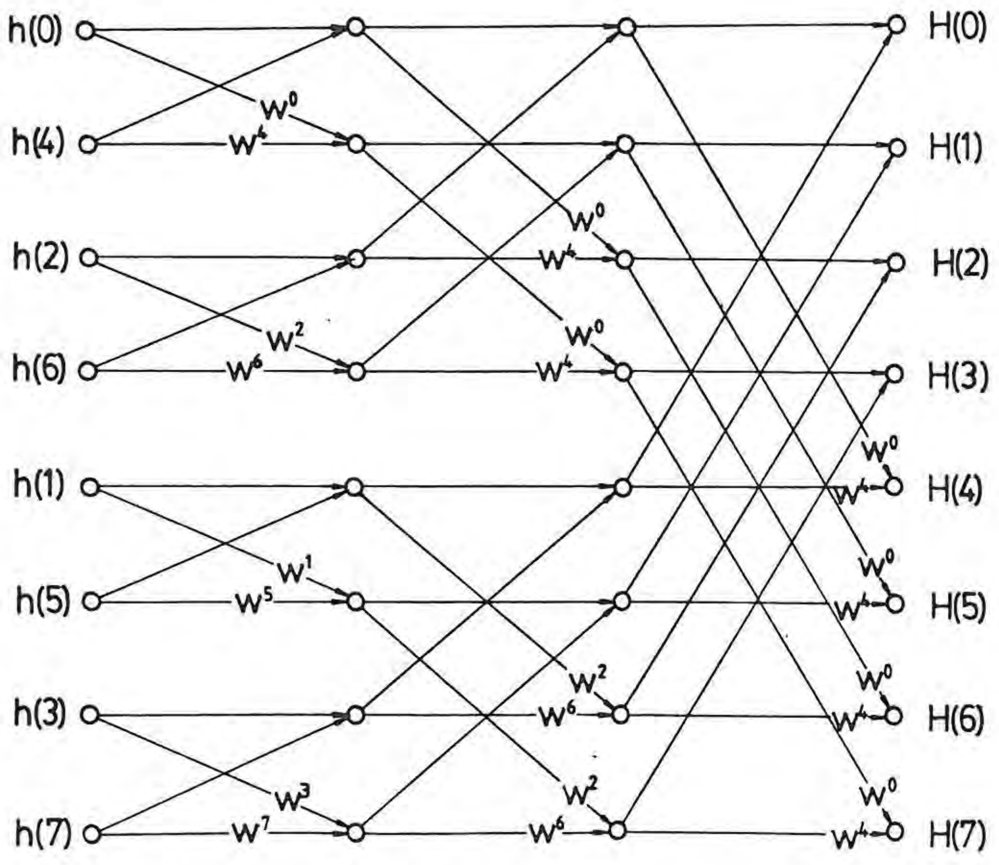


Fig. 2 - 12. Bit reversed input Sande Tukey Algorithm.

By manipulation of the Sande Tukey signal flow graph, shown in Fig. 2-11, a form of the algorithm with ordered input data sequence and ordered output, is obtainable. This is shown in Fig. 2-13 as a signal flow graph using the present notation, and it was modified from a paper by M.J. Corinthios 1971 (3) where this basic algorithm was used for a high speed signal processor. The W-term sequence is changed as a consequence, and the arithmetic can no longer be carried out 'in place', i.e. the results of the calculations can not be stored in the same operand locations as has been the case for all the previously mentioned algorithms.

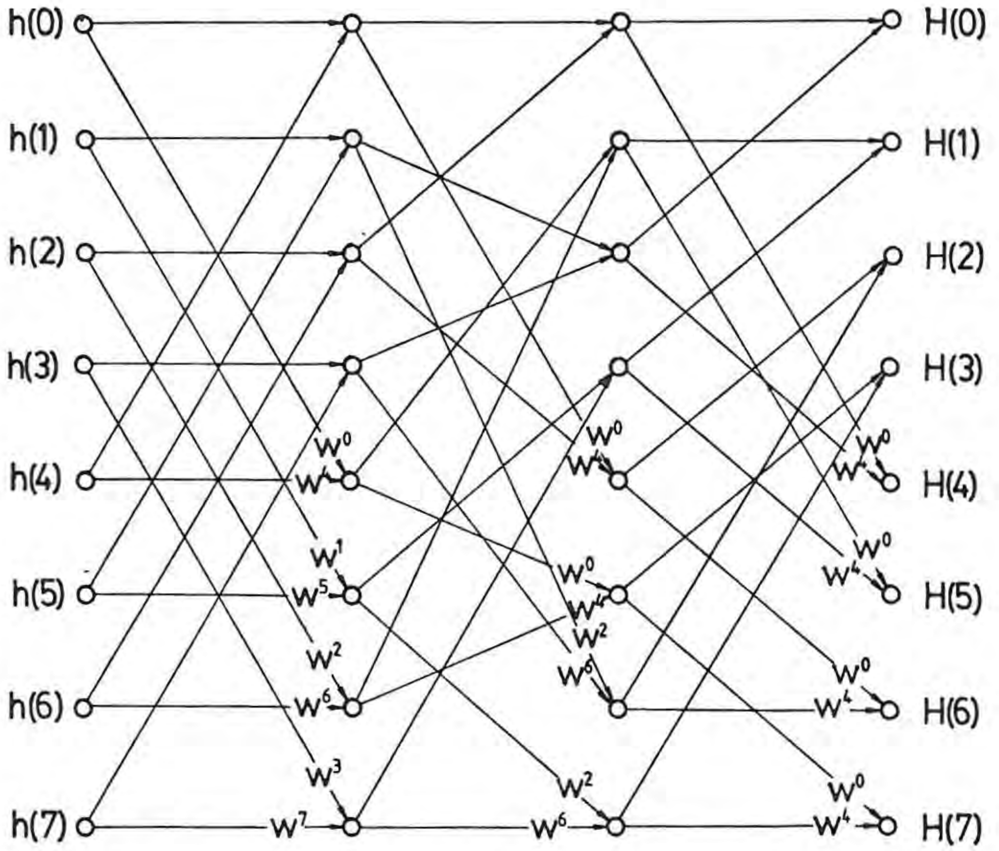


Fig. 2 - 13. Ordered input, ordered output Algorithm.

Finally, Fig. 2-14 shows the flow graph for a radix 3 algorithm i.e. $N = 3^2$, where there are 2 arrays. It is taken from a recent paper by E. Dubois and A.N. Venetsanopoulos 1978 (5), where it is the starting point for the development of a new algorithm requiring no multiplication.

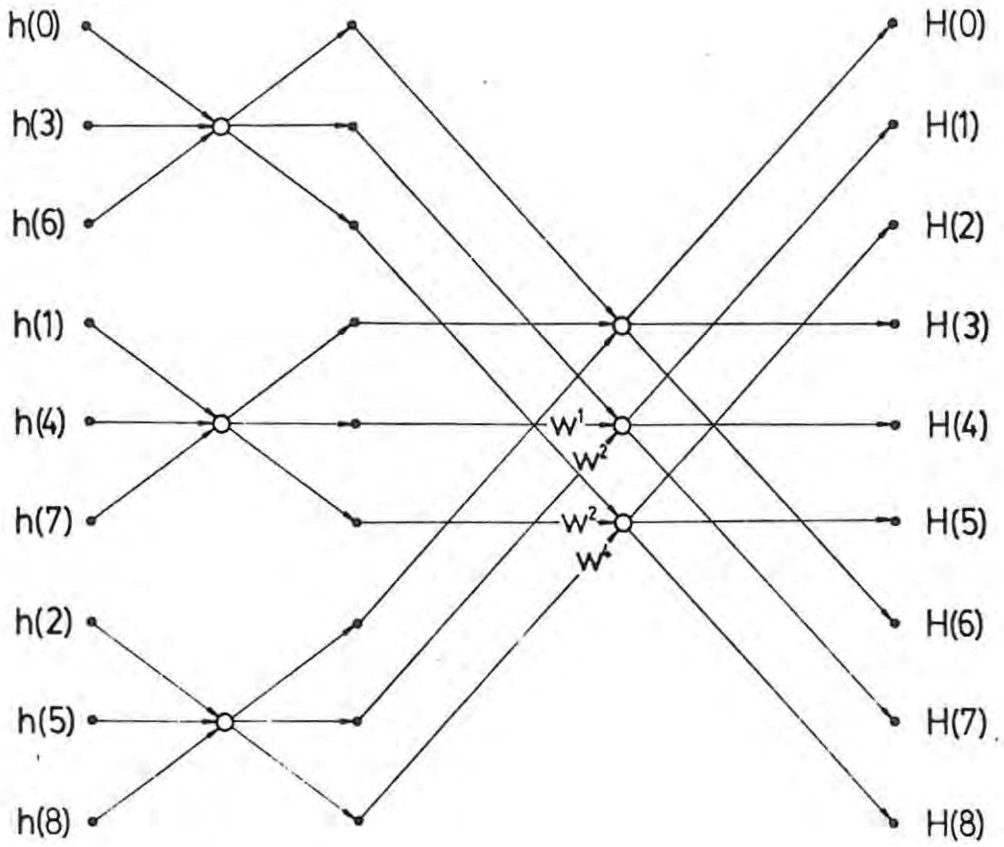


Fig. 2 - 14. Radix 3 Algorithm.

2 - 4 Simultaneous FFT on two Real Time Series

As mentioned in Chapter 1 this system must be capable of transforming two real time series which represent the outputs of two receivers with spacially separated antennas. The relative phase of the two signals received from these antennas is the parameter of interest here, for purposes of angle of arrival measurement and discrimination between the ordinary and extraordinary rays, and the only point that this condition stipulates, is that the two received signals must be sampled at the same instants and rates to preserve the phase information. Provided that each sampled time series can be stored in memory, the respective transforms could be carried out consecutively. All the algorithms, so far mentioned, compute a transform from one set of time data, where both the time data and its transform may be complex.

The time data does not usually have an imaginary part, and thus the imaginary part of the input has to be equated to zero. However, due to the odd and even properties of the DFT and hence the FFT, a method has been developed by which two real time series can be transformed simultaneously be treating one series as the imaginary part of the input.

If u(k) is made up of t(k) + js(k) where t(k) and s(k) are two real time series, then after transformation, U(n) will have, in general, the form R(n) + j I(n),

$$\text{where, } R(n) = \sum_{k=0}^{N-1} t(k) \cos \frac{2\pi nk}{N} + s(k) \sin \frac{2\pi nk}{N} \quad \ldots\ldots\ldots\ldots \quad 18$$

$$\text{and, } \quad I(n) = -\sum_{k=0}^{N-1} t(k) \sin \frac{2\pi nk}{N} - s(k) \cos \frac{2\pi nk}{N} \quad \ldots\ldots\ldots\ldots \quad 19$$

It can be seen from Eq. 18 and 19 that if u(k) was entirely real, (i.e. s(k) = 0) then,

$$U(n) = T(n) = \sum_{k=0}^{N-1} t(k) \cos \frac{2\pi nk}{N} - j \sum_{k=0}^{N-1} t(k) \sin \frac{2\pi nk}{N}$$

$$= R_e(n) + j\, I_o(n). \quad \ldots\ldots\ldots\ldots \quad 20$$

Where $R_e(n)$ is an even function and $I_o(n)$ is an odd function and similarly for the case of U(k) being entirely imaginary.

$$U(n) = j\, S(n) = R_o(n) + j\, I_e(n)$$

$$S(n) = I_e(n) - j\, R_o(n) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad 21$$

Therefore, by decomposing R(n) and I(n) into odd and even parts, the separate transforms T(n) and S(n) can be obtained from equations 20 and 21.

By decomposition,

$$R(n) = \left[\frac{R(n)}{2} + \frac{R(-n)}{2}\right] + \left[\frac{R(n)}{2} - \frac{R(-n)}{2}\right]$$

and since R(n) is periodic with period N,

$$R(-n) = R(N-n)$$

$$R_e(n) = \frac{R(n)}{2} + \frac{R(N-n)}{2}$$

$$R_o(n) = \frac{R(n)}{2} - \frac{R(N-n)}{2}$$

similarly with I(n),

$$I_e(n) = \frac{I(n)}{2} + \frac{I(N-n)}{2}$$

$$I_o(n) = \frac{I(n)}{2} - \frac{I(N-n)}{2}$$

therefore from 20,

$$T(n) = \left[\frac{R(n)}{2} + \frac{R(N-n)}{2}\right] + j\left[\frac{I(n)}{2} - \frac{I(N-n)}{2}\right] \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad 22$$

and from 21,

$$S(n) = \left[\frac{I(n)}{2} + \frac{I(N-n)}{2}\right] - j\left[\frac{R(n)}{2} - \frac{R(N-n)}{2}\right] \quad \ldots\ldots\ldots\ldots\ldots\ldots \quad 23$$

This method then enables the transforms of t(k) and s(k) to be separated after the FFT by a simple addition and subtraction, as defined by equations 22 and 23, and this calculation can also be carried out 'in place' if T(n) is stored at R(n) and S(n) stored at R(N-n).

In summary, it can be seen from the few samples, that a particular algorithm can be developed to exhibit certain properties of use to the designer. Speed is usually the criterion for development of a new algorithm, and system complexity usually the limit to the development. A compromise has, therefore to be drawn, and for this particular situation, ultra high speed was not necessary, but a simple, compact, easy-to-control algorithm was a major requirement. The choice of algorithm is left to Chapter 3 where our system requirements are developed and the selection of software or hardware implementation is made.

3. HARDWARE FFT

All the algorithms detailed in Chapter 2 could be implemented either as software or hardware. Obviously, some are more efficient as hardware algorithms due, for instance, to certain parallel operations, whereas others are suited more to software, where, for instance, the number of multiplications has been reduced (since software multiplication is time consuming). The medium in both cases is digital but the differences are in the specialisation of certain digital functions to improve speed. The choice between hardware and software is dictated by the speed required to complete the algorithm in real time. Expanding quantitatively on the operation of the Chirpsounder Ionosonde given in Chapter 1, a realisation of the system requirements will be developed, from which a choice of algorithm will be made, leading to basic system block diagrams.

3 - 1 Chirpsounder Operation

The ionosonde's transmitter outputs a signal whose frequency is linearly increasing with time, and the receiver uses this same signal as a mixer to obtain a sweeping bandwidth of reception in step with the transmitter. Fig. 3-1 indicates an echo received at time t_1 which has the same frequency as that transmitted at t_o.

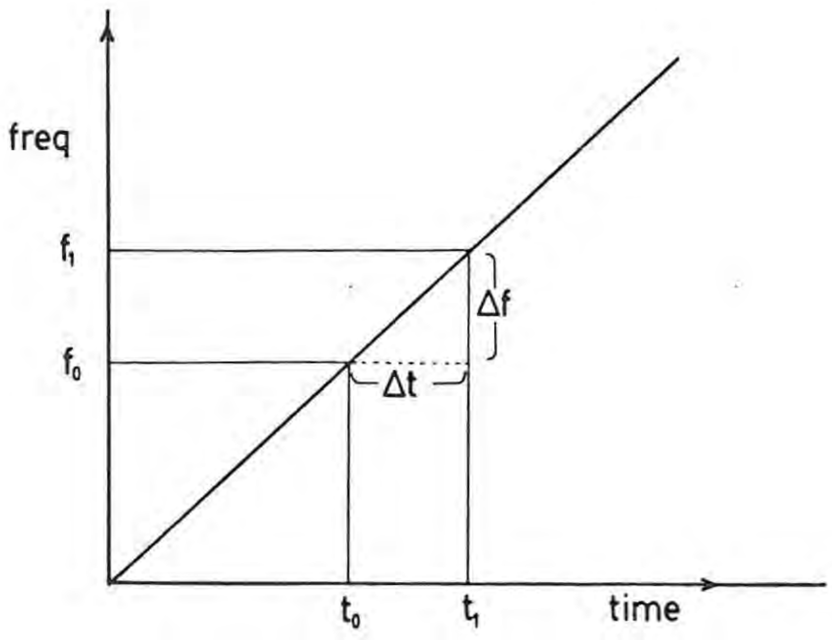


Fig. 3 - 1. Chirpsounder echo detection.

The delay Δt of the signal is the parameter of interest, being the time the radio wave took to reach the ionosphere and return, and hence giving the virtual height of reflection, and since $\frac{\Delta f}{\Delta t}$ = constant, a measurement of Δf will give Δt. At time t_1 the transmitter frequency, and hence receiver mixer frequency, has increased to f_1 and therefore, the output of the receiver will contain frequencies, $f_1 + f_0$ and $f_1 - f_0 = \Delta f$.

Since $f_1 + f_0$ is large compared to Δf, a filter can be used to separate these two signals and the receiver output frequency is hence directly proportional to the virtual height of the reflecting layer.

Assuming $\frac{\Delta f}{\Delta t}$ is 50 kHz/sec (the normal operation sweep rate for vertical incidence sounding) and if an echo had a frequency differing by 500 Hz from the present Tx frequency, then :

$$\Delta t = \frac{500}{50\text{x}10^3} = 10 \text{ msec}$$

Therefore the virtual height of reflection $= \frac{c \times \Delta t}{2} = \frac{3 \times 10^8 \times 10^{-2}}{2}$

$= 1500$ kilometres.

This is the maximum virtual height from which the chirpsounder could receive a reflection, (on account of the receiver output being bandwidth limited to 500Hz) and this height then represents the maximum frequency to be handled by the spectrum analysis. In practice, the height range up to 1000 kilometres i.e. 333Hz is analysed, which is sufficient for the present research.

To satisfy the sampling theorem detailed in Section 2-1, this signal must be sampled at a frequency of at least 1kHz, and to have a resolution of 1Hz in the frequency domain 1000 samples must be taken. (The FFT produces separate +ve and -ve frequency values). Therefore, samples are accumulated for 1 sec., spectral analysis is carried out on these samples, and to operate in real time, the next set of 1000 samples must be obtained simultaneously. The FFT must be completed well within the 1 sec. to allow further data analysis, such as the comparison of the phases of echoes that will be carried out on a microcomputer.

3 - 2 Hardware or Software

To satisfy the frequency resolution of 1Hz an FFT of $N = 1024 = 2^{10}$ was chosen and the sample frequency set at 1024 Hz. From this we can calculate the number of arithmetic operations that will be required to compute, for example, a Cooley Tukey algorithm. There will be 10 arrays, each containing 512 nodes, and at each node the operation $A \pm BW$ will be carried out, where in general A, B and W are all complex. Expanding this in complex form we have :

$$A \pm BW = A_R \pm (B_R W_R - B_I W_I) + j(A_I \pm (B_R W_I + B_I W_R)) \quad \ldots\ldots\ldots\ldots 24$$

This clearly would take 4 multiplications and six additions, making a total for the complete FFT of 20480 multiplications, and 30720 additions. This is a formidable task since multiplication is not a simple digital operation and requires several clock cycles and complicated control. Methods have been developed to reduce the number of multiplications, or the time spent in the operation, but in general circuit or control, complexity is increased. For example, Dubois and Venetsanopoulos 1978 (5), use a radix 3 algorithm to exchange all the multiplications for additions, but the control over selection of data for addition is more complex. In a "pipe line" FFT processor, a method, as detailed by Liu and Peled 1975 (11), would achieve a balance between speed and complexity. This method utilizes new technology in the form of "Read Only Memory" (ROM), to store partial product terms which could be added and shifted to achieve the desired multiplication. In a "pipe line" processor, for $N = 2^{10}$, there are ten arithmetic units which handle the different arithmetic sequences of each array individually, and hence produce a high circuit cost.

Finally, some applications require extra high speed and "parallel or cellular processors" are used. These replace every node with an arithmetic unit so that for $N = 2^{10}$, there would be 5120 such units. Cyre and Lipovski 1972 (4) describe such a system where, instead of feeding separate multiplication values to each node, they are calculated from the previous array value, and fed along with the data.

All of these examples are for specialised high speed processors, and are mentioned here to indicate the manipulation of the algorithms that is possible, to achieve certain specifications. Our system does not warrant such complexity or speed, so the decision to be made, is whether the algorithm could be best implemented in software or hardware.

Assume a software system is chosen, which is logical since the data analysis, following the FFT, would be a software program in a mini-computer. How long would it take to complete our FFT? A multiply routine in a mini-computer typically takes 100 clock cycles, which at the clock rate of 1MHz, would mean a total time, for just the multiplications, of 2,048 seconds, which is clearly far too slow. Fast multiply peripheral units are available, but even if these units only took 10 clock cycles, signifying a multiplication time of 200 msec, the time for addition and the complicated control sequence added to this, would leave little time available for the further computation within 1 second.

A hardware system is therefore required, which has the ability to be clocked, at up to 10 MHz, and that can utilize hardware control units which take advantage of the modulo-2 structure of some algorithms.

This approach, now depends upon the choice of algorithm to achieve the optimum balance between speed and circuit complexity.

3 - 3 Algorithm Choice

All algorithms have a basic structure, that is based around the arithmetic unit or units. Data has to be obtained in the correct sequence, together with the required multiplying factor, applied to the arithmetic unit, and finally, the results stored again in correct sequence. The control of the sequence of addressing operands and results, is a major time consumer, and if this could be automated in the hardware by choice of algorithm, then the FFT time would be almost reduced to the arithmetic time alone. The choice between algorithms, therefore, is reduced to selection of one that can incorporate such a control sequence, and that does not require too much hardware to achieve the desired control.

Referring back to the properties of algorithms detailed in section 2-2 and 2-3; from here one property appears as being of prime importance to a hardware FFT, that is, being able to carry out the arithmetic "in place". An algorithm without "in place" arithmetic would require a memory size twice that of an "in place" arithmetic system, since the results would need clear storage ready for the next array. The real time system which is envisaged, would require a large memory, since clearly, new samples could not be stored in the memory on which the arithmetic unit is operating. Therefore, by cutting down on memory size, a saving of power, parts count and control time is achieved.

In order for the algorithm to be automated, it must possess properties easily implemented by digital functions, such as binary counters and shift registers, and thus the radix 2 algorithms are favoured. Recalling the data selection sequence, required for Fig. 2-9, notice that for the first array, the points are selected 4 apart i.e. $h(0)$ and $h(4)$ are used to calculate $h_1(0)$ and $h_1(4)$. For the second array, the data is selected 2 apart i.e. $h_1(0)$ and $h_1(2)$, and for the final array, only 1 apart i.e. $h_2(0)$ and $h_2(1)$. This is highly modulo-2 arithmetic, achieved by dividing the displacement by 2 each array, and this form of addressing is common to the algorithms depicted by Figs. 2-9, 2-10, 2-11 and 2-12, i.e. both forms respectively of the Cooley Tukey and the Sande Tukey algorithms.

The W-term has to be addressed for each node and by examination of Fig. 2-9, it can be seen to be a complicated sequence. Recalling that $W^4 = -W^0$, then only W^0 is required for the whole of the first array, and similarly since $W^6 = -W^2$, W^0 is required for the first half of the array, and W^2 for the remainder. The final array, in which $W^5 = -W^1$ and $W^7 = -W^3$ selects all the W values required for this algorithm and their sequence is W^0, W^2, W^1, W^3. This is in fact a bit-reversed sequence, but the main property of this algorithm is that each array takes its values in the same sequence as the final array, i.e. the first array uses just the first W-term, the second array uses the first two W-terms, the third array uses the first four W-terms and if Fig. 2-9 was for $N>8$, the fourth array would take the first eight W-terms. This property is common for Fig. 2-9 and 2-12 and is clearly easily implemented in a digital system.

Figs. 2-10 and 2-11, however, have a different sequence which can be seen in the final array of Fig. 2-10, as being ordered i.e. W^0,

W^1, W^2, W^3. Assuming that this algorithm was for N>8 then the sequence for each array would be : The first array only uses the first value, the second array uses the first and the middle value, and the third array uses the first, the middle and the upper and lower quarter positions. This again is very modulo-2 and can therefore be implemented, but it is more complicated than the previous sequence.

The choice of algorithm has, therefore, been reduced to Fig. 2-9 or Fig. 2-12. They exhibit different arithmetic operations, and Fig. 2-9 accepts ordered data, whereas Fig. 2-12 requires bit reversed data. To address the data in bit reversed sequence is simple in hardware since it just requires the address lines to be swopped in order of significance i.e. the most significant becomes the least significant etc., so there is no time involved in this operation, and, therefore, no advantage gained by bit reversing the input as opposed to re-ordering the output.

Similarly, there is nothing to choose between the two types of arithmetic operation. The Cooley Tukey and the Sande Tukey arithmetic are shown in Fig. 3-2a and 3-2b respectively, noticing that the only difference is, that the multiplier changes position.

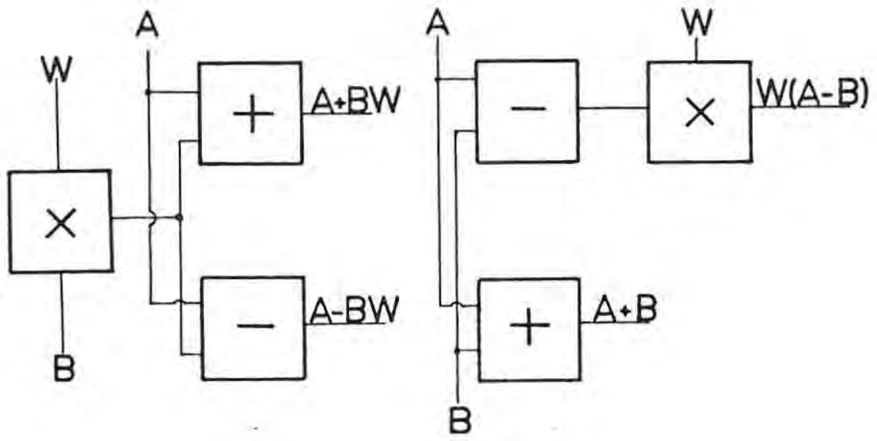


Fig. 3-2a Cooley Tukey Arithmetic. 3-2b Sande Tukey Arithmetic.

3 - 4 System Structure

The choice was made to implement the Cooley Tukey Algorithm of Fig. 2-9, and Fig. 3-3 shows a basic block diagram of the units required for the implementation.

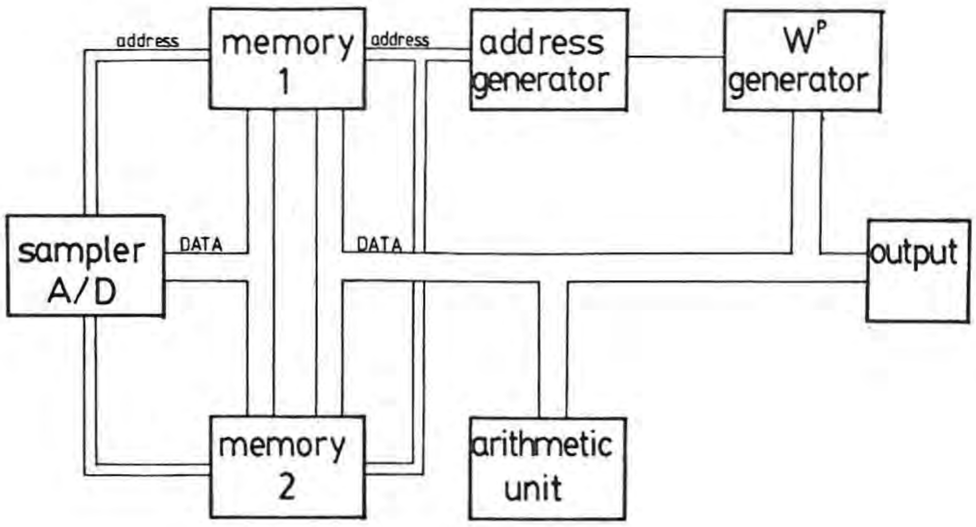


Fig. 3 - 3. Cooley Tukey algorithm implementation.

3 - 4 - 1 Memory

To enable the sampler to access its memory at any time while the FFT is being processed, a double data and address bus arrangment is required, and a multiplexer then selects which memory is connected. Since the sampler only writes to memory, a single direction bus is all that is required, whereas the system bus has to be bi-directional. For convenience, the two buses will be called the "sampler bus" and the "system bus", and similarly for the address. Diagrammatically this arrangement is shown in Fig. 3-4.

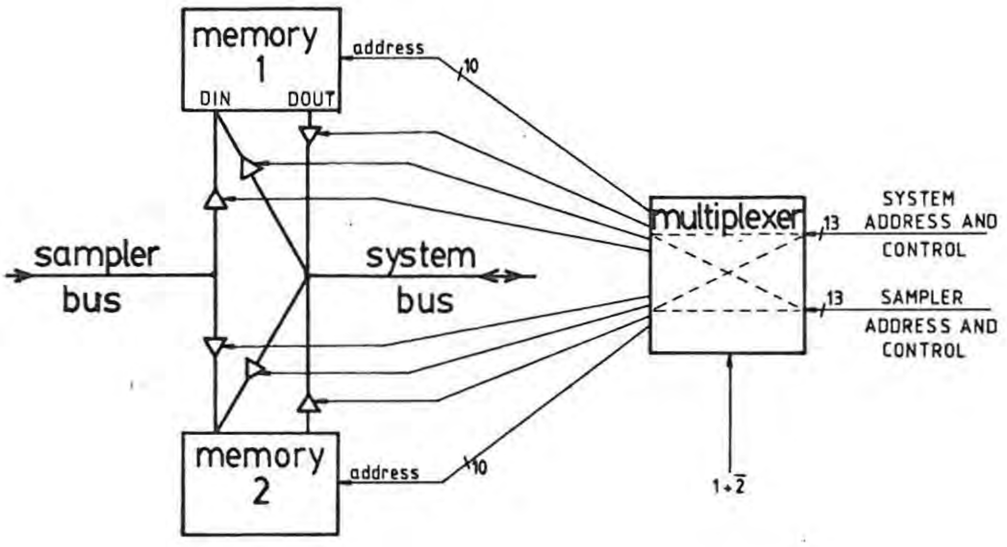


Fig. 3 - 4 Memory bus multiplexing.

Since the memory has an input port and an output port, the bus separation can be achieved by using tristate buffers, which either allow the data through or switch to a high impedence state to allow the other bus to pass data. The tristate buffer control signals also require multiplexing and so the multiplexer has to swop 13 lines each from the sampler and system each time the sampler has taken 1024 samples.

3 - 4 - 2 Arithmetic Unit

The structure of Fig. 3-2a has to be implemented, and a choice has to be made between serial or parallel units. Both adders and multipliers are available in serial and parallel form, but if the arithmetic is to be carried out in 2's complement binary, then a serial unit is the most economical. In 2's complement arithmetic, the sign is automatically determined if two numbers are added in a conventional manner. However, multiplication does not have this property, and usually, the sign of each number is tested, then the multiplication carried out on positive numbers and finally, the result adjusted in sign. This all takes extra time and makes extra complication for the arithmetic unit. Fortunately, Advanced Micro Devices released a serial multiplier that accepts 2's complement numbers and this unit is detailed in an article by J.R. Mick and J. Springer 1976 (12) where Booth's algorithm, the essence of the unit, is explained.

This algorithm makes use of the fact that a string of 1's in a binary number, running from bit weights 2^r to 2^s can be represented as, $2^{s+1} - 2^r$ (for s>r) e.g. 011100 (28_{10}) = $2^5 - 2^2$ = 28. Thus, in the long-hand method of multiplication, by shifting and adding, Booth's algorithm states that by examining the multiplier a bit at a time (serially), on the first 1 of a string, the multiplicand is subtracted from the partial product, and on the first 0 of a string it is added, and at each operation the partial product is shifted towards its least significant end. In a software environment, this algorithm achieves a time saving since it only requires shifts of the partial products at some stages, but in hardware it is much more useful since it accepts 2's complement numbers, e.g. the 2's complement number 111100 ($- 4_{10}$) = -2^2 = - 4 since the string of 1's represents the sign and a leading zero never appears to cause a final addition.

The integrated circuit designed around this algorithm is the Am25LS14, and it accepts the multiplicand as an 8 bit parallel input, the multiplier is serially applied least significant bit first, and the result is output serially, also least significant bit first. Since an 8 x 8 multiplication produces a 16 bit result, the multiplier must have its sign extended for the extra 8 clock cycles and so a special shift register is required. AMD thoughtfully produced the Am25LS22 which is an 8-bit sign extend shift register, at the same time as the Am25LS14, and with the FFT arithmetic in mind they also produced the Am25LS15 which is a quad serial adder/subtractor.

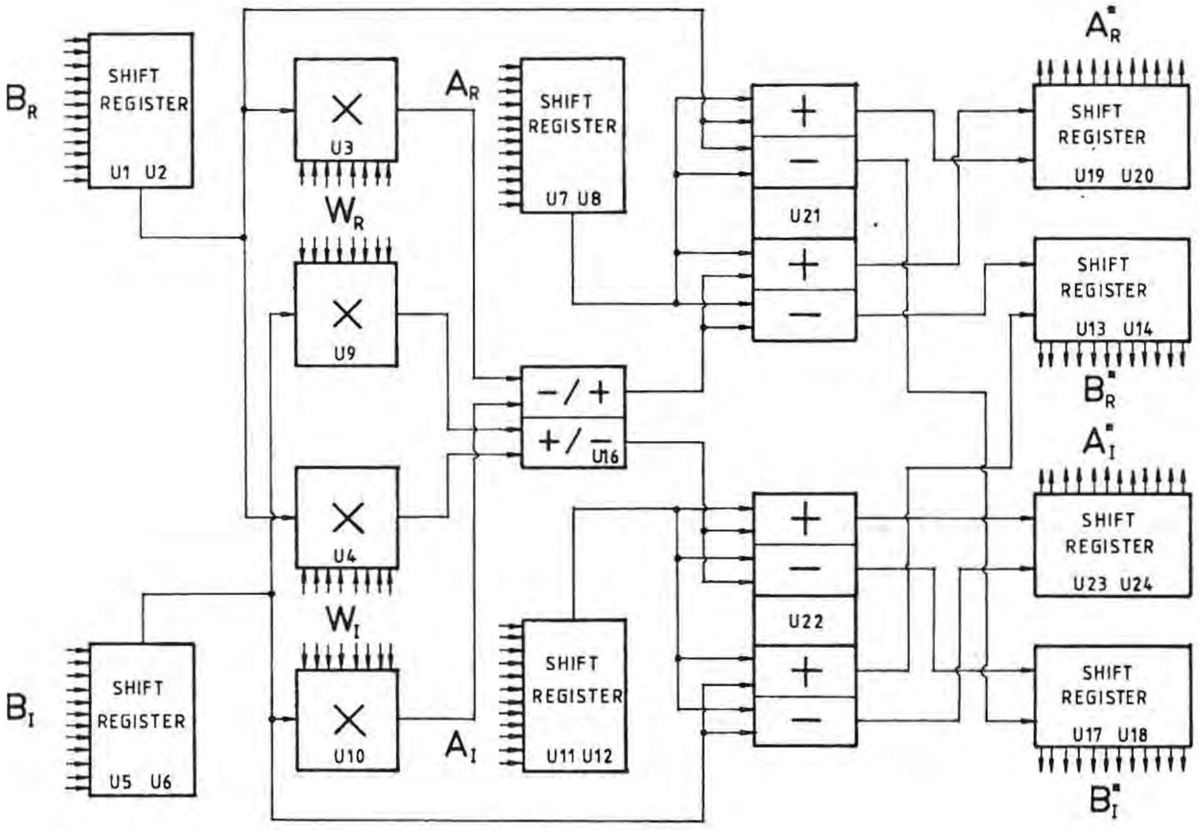


Fig. 3 - 5. Arithmetic unit.

Thus, a serial arithmetic unit was developed using the AMD integrated circuits and the block diagram is shown in Fig. 3-5.

The diagram in Fig. 3-5 is much more complicated than that of Fig. 3-2a since A, B and W are all complex, hence the arithmetic A $\pm$ BW becomes :

$$A \pm BW \equiv (A_R \pm B_R W_R \mp B_I W_I) + j(A_I \pm B_R W_I \pm B_I W_R)$$

In Fig. 3-5, A^* and B^* are the results of the calculation which for the FFT are $A^* = A + BW$ and $B^* = A - BW$. However, Fig. 3-5 is further complicated since the arithmetic unit has to be able to calculate the unscramble algorithm detailed in Section 2-4, and also to give a power spectrum output. These two extra functions have been built into Fig. 3-5 and involve extra adders and utilising the dual serial inputs to the output shift registers.

This is a convenient method of implementation since time is not wasted in transferring data between registers as in a parallel system, and also extra division by 2, required by the arithmetic, can easily be made by shifts in the output registers.

3 - 4 - 3 Address Generator

This unit had to create the address sequence to access the data required for each node of each array. Referring to Fig. 2-9 and extrapolating to the case of N samples, the sequence required is, that for the first array, the data is selected $\frac{N}{2}$ apart i.e. A is taken from n and B is taken from $\frac{N}{2} + n$. For the second array, the data is selected $\frac{N}{4}$ apart, for the third array, selected $\frac{N}{8}$ apart etc. After the first array, however, skips in the data sequence are required to jump the blocks of newly calculated values. For instance, in Fig. 2-9 for the second array, the data sequence for A is 0,1,4,5, and for B is 2,3,6,7. For A, the addresses 2 and 3 are skipped since they contain the new B^* values, calculated from A at 0 and 1, and B at 2 and 3.

Since $N = 2^\ell$, and at the rth array the separation of the values is $\frac{N}{2^r} = \frac{2^\ell}{2^r} = 2^{\ell-r}$, then the separation is always an integer power of 2. This means that in a binary number, the B address of, for example, the first array, is obtained by changing the most significant bit of the A address to a 1. Similarly, the second array B address, is obtained by changing the penultimate most significant bit. In each array, however, only $\frac{N}{2}$ addresses need be generated, (i.e. the A addresses) and from these, the B addresses are produced. Thus, for our system with $N = 2^{10}$, only a nine bit counter is required, which will count 0 - 511 for the A address of the first array, and the tenth bit will give the B addresses.

The sequence for the second array can be produced from this same nine bit counter by changing the ninth bit to the tenth bit position and putting the B address bit in the ninth bit position. Thus, the counter counts 0 - 255 (with the B addresses of 256 - 511) but then the next count will be 512 (with the B addresses of 768) since the carry skips the ninth bit position and produces a 1 in the tenth bit. Fig. 3-6 shows this use of the nine bit counter diagrammatically and extends to producing the count sequence for all the following arrays.

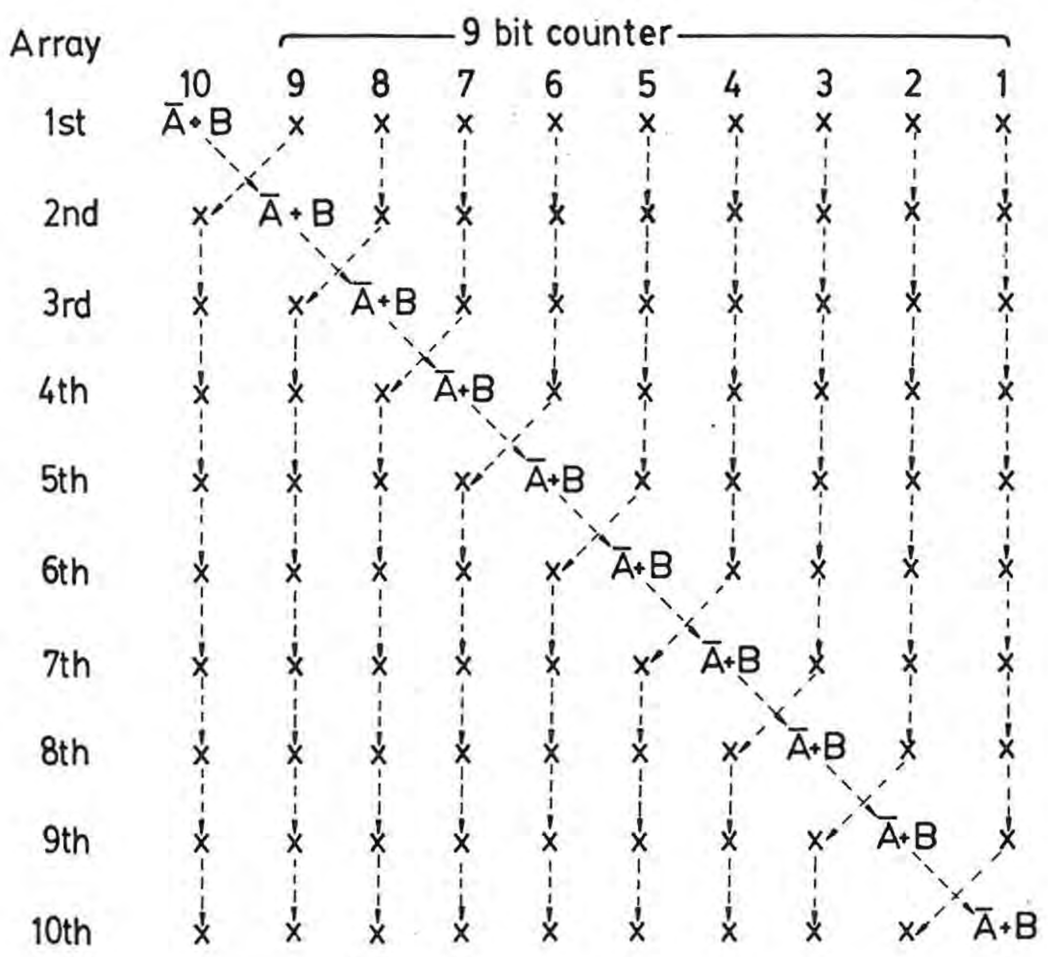


Fig. 3 - 6. Array address sequences.

In Fig. 3-6, the term $\bar{A} + B$ indicates that the bit is 0 for addressing A or 1 for addressing B, and the X's are the counter bits which can be 1 or 0.

This then can produce all the addressing required for each array of the FFT, which, after ten arrays, leaves the data in bit reversed order. A bit reversed address sequence is therefore, required to output the transformed data in the correct sequence, and this is simply $\bar{A} + B$ and the nine bit counter reversed in bit significance. Another bit reversed address sequence is required to address the operands for the separation of the two receiver signals as detailed in Section 2-4. It requires two addresses, one termed n (i.e. straight counting) and the other N-n (i.e. reverse counting) where n = 0, 1 .... N-1. The n sequence is the same as that for the output of the transform, whereas the N-n effectively addressed from the top of data down. By inverting the n sequence address, that is, making every 0 a 1 and vice versa, and adding 1, the N-n sequence can be created, so that for n = 1, N-n = 1023 which is the top of memory. For the case n = 0, N-n = 1024 which is outside the memory space, however, since the transform is periodic in the frequency domain N-n = -n = 0 which addressed the DC term, and therefore there is only one DC term for both receivers. The N-n sequence requires that the n sequence be decremented by 1 and then inverted, and this can be carried out using an adder to add -1 to the n address, and then driving the address bus through an inverter buffer. To implement these three address sequences, and to drive the address bus with only one at a time, an arrangement of tristate buffers is envisaged. Each sequence is isolated from the address bus by a tristate buffer and only one set of buffers is enabled at any one time.

The W-term address sequence is closely associated with the data address and it would be advantageous to produce it from the data address counter rather than have a separate counter and sequence unit. Referring to Fig. 2-9 again, it can be seen that the W-term in the first array is the same for all the $\frac{N}{2}$ arithmetic operations, and in the second array, it changes at the $\frac{N}{4}$th operation. In other words, each time the data address skips a block, the W-term changes. This can be seen in Fig. 3-6, that whenever the next most significant bit after the $\bar{A} + B$ bit changes, so does the W-term. This bit change is difficult to use to clock the W-term, since the carry is not available externally on a counter, but the previous state of all 1's below the $\bar{A} + B$ bit is easily detected and simple "Anding" used to generate a clock for the W-term. This method requires the W-term to be clocked after its use, to enable the next value to be available before the data address changes, thus putting the W-term change ahead of the data address change. Combining all these functions into one diagram produces Fig. 3-7.

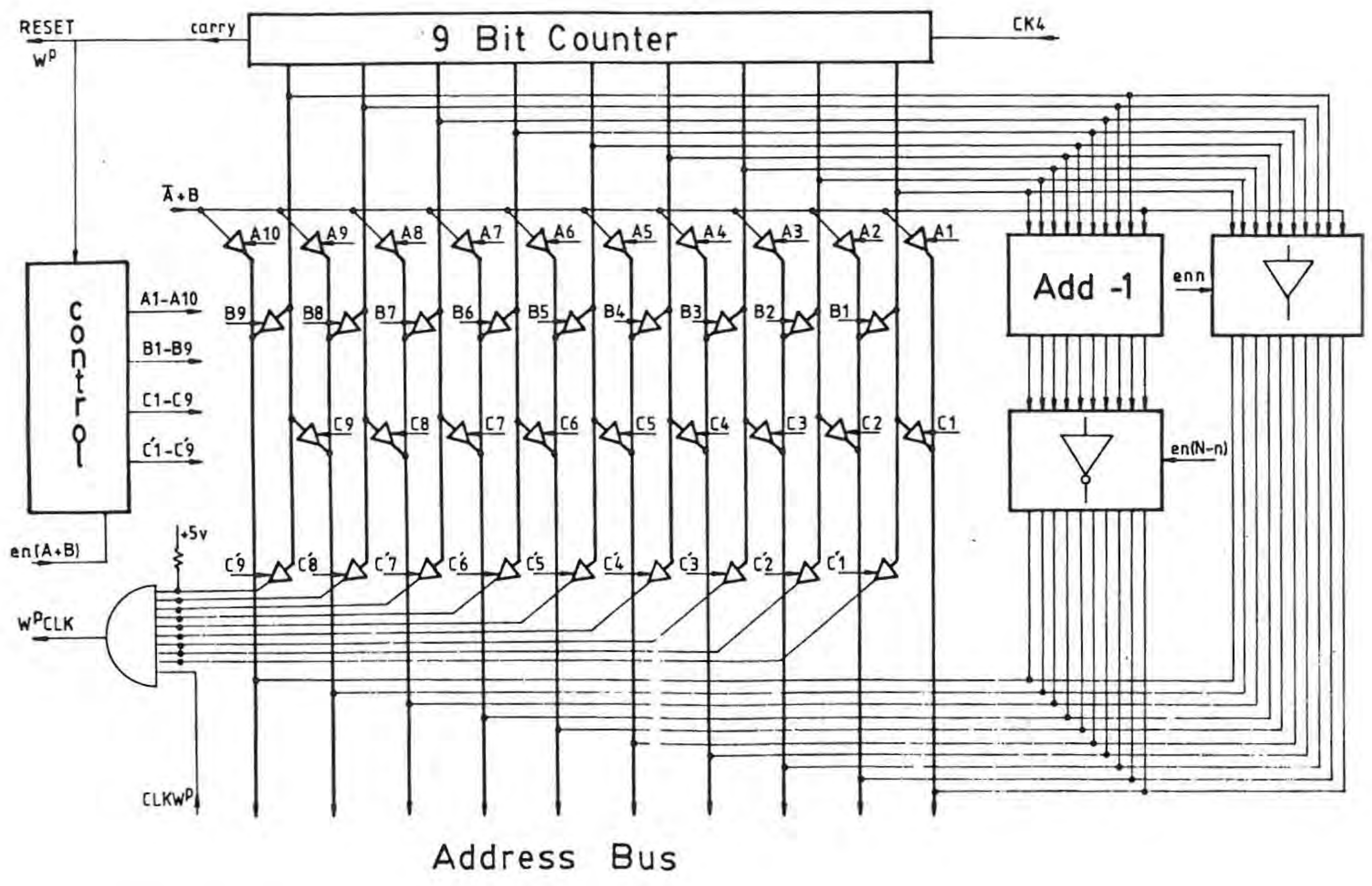


Fig. 3 - 7. Address Generator.

The "control" block in Fig. 3-7 is incremented each time the counter produces a carry (after a count of 512), and this changes the A, B, C and C' control sequence to that required for the next array. The bit reversed addresses are enabled by "en n" and "en (N-n)", with the FFT address sequence disabled through "en(A + B)". Sequence C' is enabled continuously, and the pull up resistors ensure that, as the lines are deselected by the tristate buffers, they stay at a logical 1. The control sequences for the tristate buffers are tabulated in Fig. 3-8 where a 1 indicates disable or high impedence and 0, enable.

3 - 4 - 4 W^P Terms

Recalling from Chapter 2 that $W^P = e^{\frac{-j2\pi P}{N}} = \text{Cos}\,\frac{2\pi P}{N} -j\,\text{Sin}\,\frac{2\pi P}{N}$ where $P = 0 \rightarrow \frac{N}{2} -1$

The real part of W^P can then be seen to be the first half cycle of a Cosine wave split into $\frac{N}{2}$ samples, and that the imaginary part is -1

array	A	B	C and C′
	10 9 8 7 6 5 4 3 2 1	9 8 7 6 5 4 3 2 1	9 8 7 6 5 4 3 2 1
1	0111111111	111111111	000000000
2	1011111111	011111111	100000000
3	1101111111	001111111	110000000
4	1110111111	000111111	111000000
5	1111011111	000011111	111100000
6	1111101111	000001111	111110000
7	1111110111	000000111	111111000
8	1111111011	000000011	111111100
9	1111111101	000000001	111111110
10	1111111110	000000000	111111111

Fig. 3 - 8. Address control sequences

times the first half cycle of a Sine wave also split into $\frac{N}{2}$ samples. Therefore, there are 512 W^P values each having a real and imaginary part, and rather than generate these values each time they are required (which would take time and special hardware), it was decided to have a look-up table. This takes the form of a Read Only Memory (ROM) addressed by a counter which is clocked from the address generator. Recalling that the power P is a bit reversed sequence through each array, the W^P terms were calculated and stored permanently in the ROM in this sequence. Fig. 3-9 shows the arrangement with a nine bit counter addressing the 512 W^P values, and the values being enabled onto the common data bus via a tristate drive.

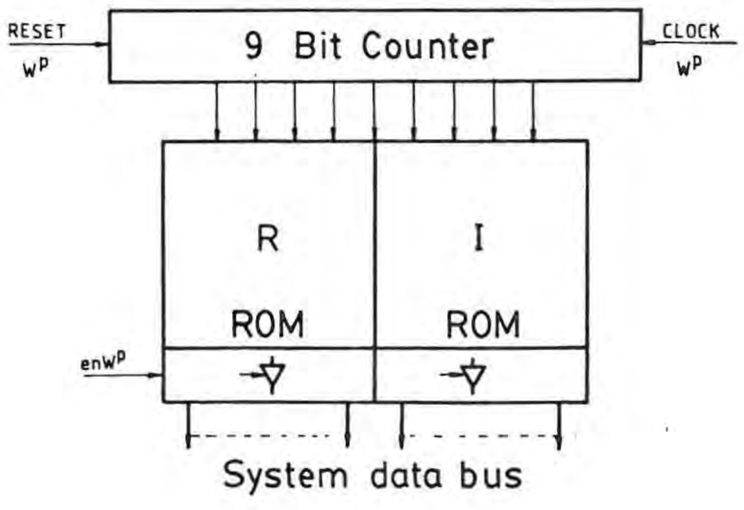


Fig. 3 - 9. W^P look up table.

3 - 4 - 5 Sampler and A/D Converter

The output of the ionosonde receiver which has been band limited to 500 Hz is sampled at a rate of 1024 Hz, and each of these samples converted into a digital word and stored in memory. This operation proceeds independently of the FFT and the only interface is a signal from the sampler to indicate that it has taken and stored 1024 samples of data. The sample rate is supplied by an external clock, which is accurately derived from the Chirp ionosonde time standard. The instant of sampling defines the relative phase of the spectral components, and hence to be able to compare sequential transforms, the sample time must be well defined.

At the instant of sampling, the data on the real and imaginary inputs are held and the Analogue to Digital (A/D) converter triggered to start conversion. The "end of conversion signal" then latches the real digital data into the "real" latch and then starts conversion on the imaginary input. The end of conversion then latches the imaginary data, writes both real and imaginary to memory and finally increments the address for the next sample. Fig. 3-10 shows a block diagram of this function.

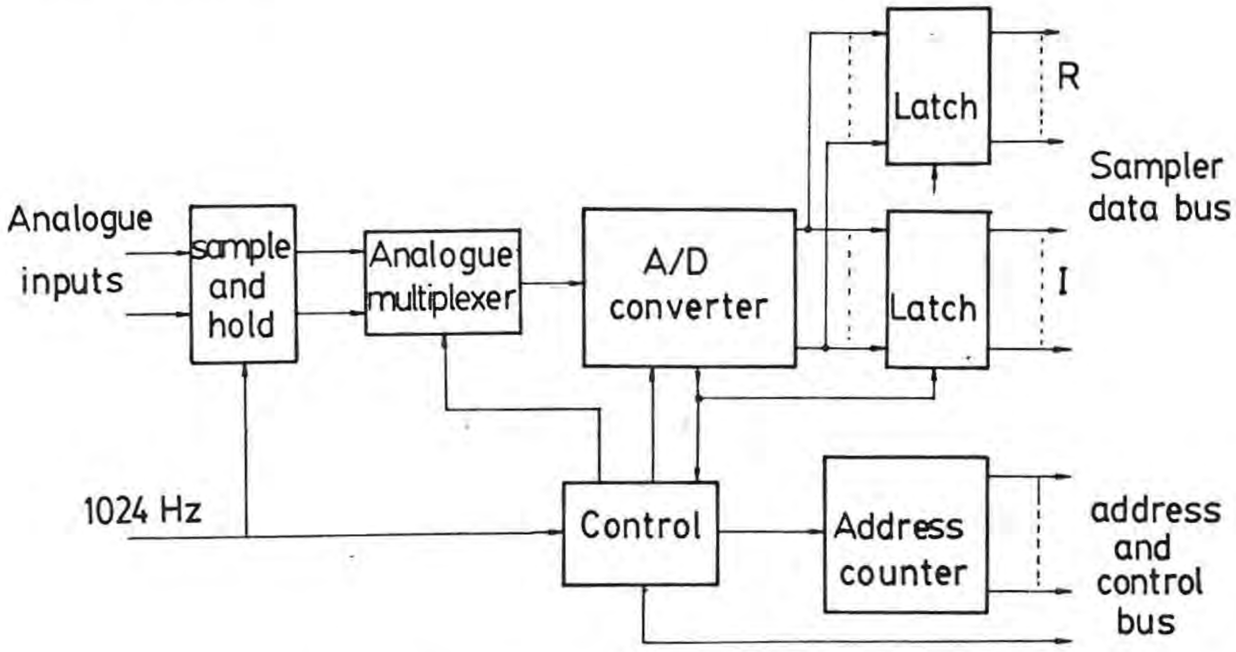


Fig. 3 - 10. Analogue Input.

3 - 4 - 6 Output

The transform, when complete, has to be transferred to a computer for further analysis, and so the output primarily consists of a latch from which the data is read. To facilitate testing of the unit, a Digital to Analogue (D/A) converter was included after the latch so that an oscilloscope trace can be obtained of the output. Fig. 3-11 shows the block diagram of this unit.

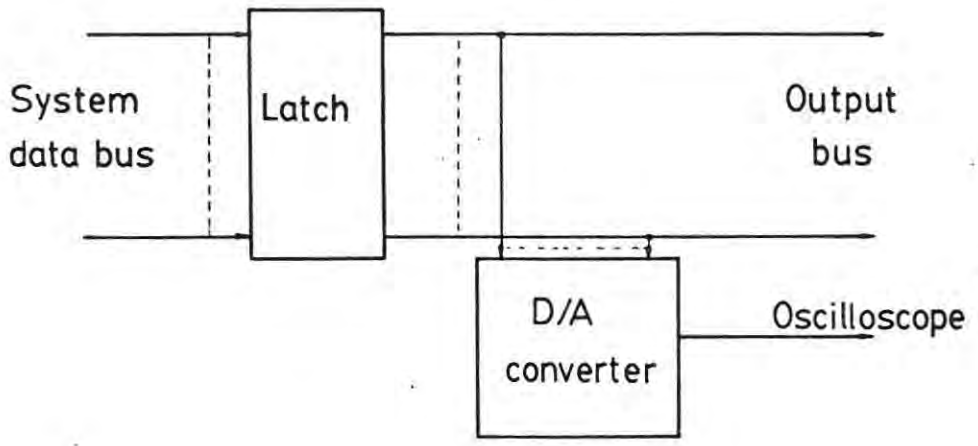


Fig. 3 - 11. Output and Display.

This Chapter has shown the development of a basic hardware system from the selected algorithm and has detailed the simple operation of the hardware. Chapter 4 will discuss the finer details of the arithmetic unit such as word size and truncation problems, and comprehensive circuit diagrams of the whole system can be found in Appendix 3.

4. ARITHMETIC REQUIREMENTS

The previous chapters have developed a basic system as outlined in the block diagram Fig. 3-3, and a few of the finer details were examined in section 3-4. The choice of size for the data word, (i.e. the number of bits used to represent the operands) was expressly not mentioned since it was dependent upon several factors, including accuracy, number system (2's complement, fixed point, or floating point) and the hardware available. This will be discussed here and an examination made of the errors involved in an iterative calculation such as the FFT.

The design of the arithmetic unit and its requirements was not completely open to choice, since the hardware to be used imposed limitations such as word size and number system. Also, the accuracy of the output data was limited to 8 bits since it was to be analysed further by an 8 bit South West Technical Products Corporation 6800 microcomputer, and this accuracy was deemed sufficient for the system requirements. Therefore, the minimum word size for the system, was 8 bits, but due to the hardware limitations of the AMD arithmetic chips to be used, the expansion above 8 bits was in blocks of 4 i.e. to 12, 16 or 20, bits. For purposes of speed and simplification of control, fixed point arithmetic was decided upon for the arithmetic unit. This was a considerable decision, and the consequences had to be carefully examined to understand the effect on the accuracy of the data. Also, due to speed, it was decided that the arithmetic was to be carried out in 2's complement notation. This was partly due to the fact that the serial multipliers chosen could handle 2's complement numbers, whereas, if this had not been so, special control would have had to be included for the multipliers, thus losing the advantages gained by 2's complement.

4 - 1 Fixed Point Binary Arithmetic

Let us consider what happens to the numbers through our repetitive arithmetic. The numbers are formed by an Analogue to Digital converter which has sampled the input time series, and produced a digital word which has a limited number of bits. Let the word have t bits to represent the magnitude of the sample, and an extra bit for the sign, disregarding, for the present, how the sign and magnitude are handled. If we add two such samples together, then the absolute maximum answer we can obtain, will have t + 1 bits for the magnitude, and 1 for the sign, i.e. the number of bits required to represent the number, has increased by 1. Therefore, if we were to allow this to proceed through the FFT arithmetic, we would have to ensure that there were enough bits in the arithmetic word, to take account of the maximum possible growth of the data. Thus, an analysis of the specific arithmetic to be carried out, must be made to ascertain whether the numbers are such that the results will increase or decrease as the calculations progress.

P.D. Welch 1969 (14) carried out such an analysis and concluded that "... in the root-mean-square (rms) sense, the numbers (both real and complex) are increased by $\sqrt{2}$ at each stage". and ".... the maximum modulus of the array of complex numbers is nondecreasing". This indicates that as a minimum, the values double after every other array (stage), so that if we start the calculations with a t bit significance, then after 10 arrays, we will require t + 5 bits minimum to represent the data. The peak value of a calculation, however, can exceed $\sqrt{2}$ times the input after each stage, and it is very data dependent, so an upper bound of 2 times each array was fixed as a trial condition to ensure no overflow of data. Our data word length, under this condition, would then be t + 10 bits.

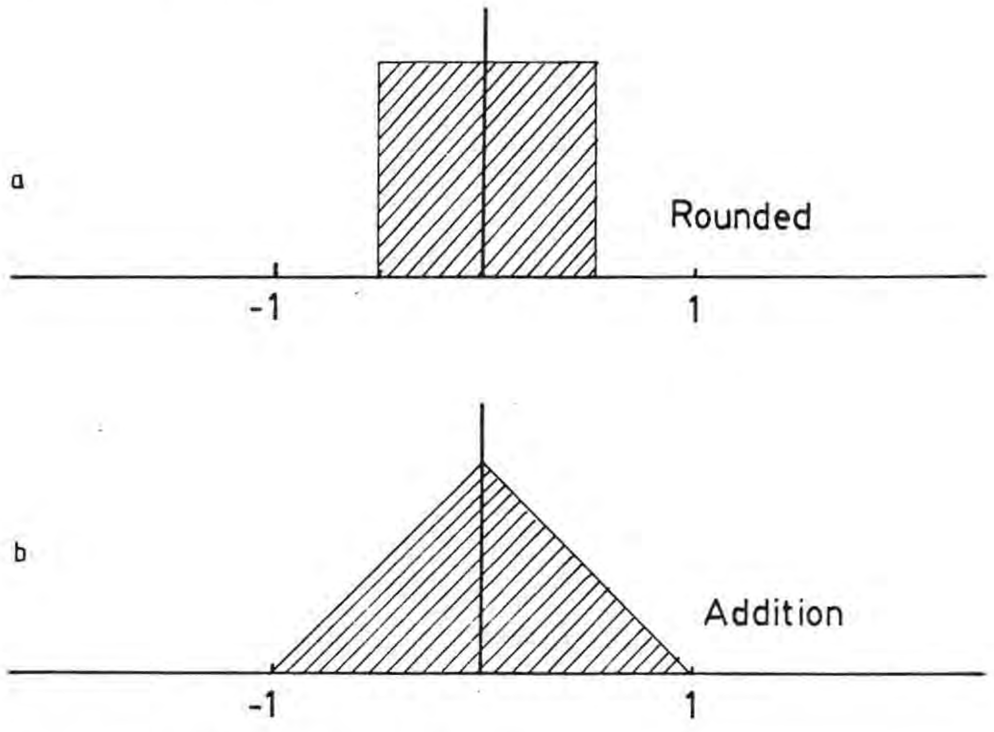


Fig. 4 - 1. Arithmetic error distributions.

Another approach to analysing this problem was to consider the accuracy of the numbers before and after a calculation. Fig. 4-1a, indicates a binary number from the output of an A/D converter, where the shaded area indicates the analogue value distribution which is represented by the binary number. This area is $\pm$ 1/2 the least significant bit value and has an equal or level probability within this range. Fig. 4-1b, shows the result of adding two A/D values, indicating the spread of the error, and is obtained by convolution of the two A/D error distributions. Hamming 1962 (7) p. 33, develops this distribution, and extends the idea to the addition of several values and using the "Central limit theorem" concludes that the final distribution will approach the normal distribution. Therefore, as the FFT calculations are increasing in word size, the error in the least significant bits is also increasing, and so the question can be asked, "can these least significant bits, that are in error, be discarded"? In other words, does the arithmetic unit and memory, have to cater for t + 10 bits, or could the data be "rounded" to t bits after each calculation without greatly affecting the data accuracy. Several authors deal with this and develop upper and lower bounds to the ratio of rms error to rms answer, e.g. Welch 1969 (14) or mean square error (floating point) e.g. Kaneko, Liu 1970 (10), Glisson, Black, Sage, 1969 (6), but all assume that the arithmetic operations are separate and parallel in nature so that each operation adds error to the analysis.

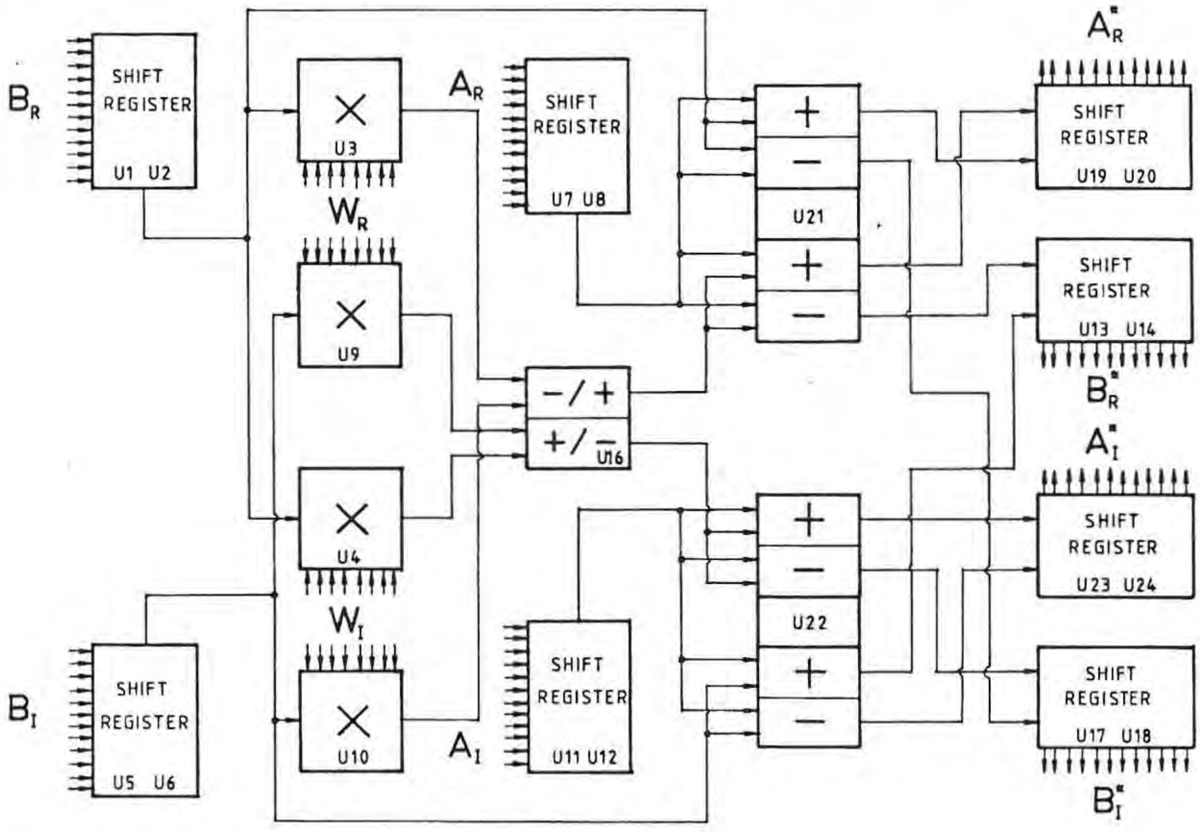


Fig. 4 - 2. Arithmetic unit block diagram.

Our particular case is different due to the serial form of the arithmetic operation and the 2's complement notation. Referring to Fig. 4-2 (a duplicate of Fig. 3-5), and assuming that all input operands are t bits wide, then the first operation is multiplication. W is applied in parallel to the multipliers and B is shifted in LS bits first. Out of each multiplier, a 2t bit product appears, LS bit first, of which the MS t bits are the data of interest, since the W term is a fractional multiplier, $\leqslant 1$. In a parallel processor, the LS t bits would be rounded or truncated, hence adding error into the data, but in our case, the LS bits are passed on to the addition stage. The central addition stage, therefore, operates on two 2t bit words to produce a possible 2t + 1 bit sum, which is then passed on to the final adders. The MS t + 1 bits are the data of interest, and are to be added to A which is a t bit word. Since there are no LS bits of A to correspond in binary weight to the t LS bits of the previous sum, truncation must occur, and this is carried out by not clocking the final adders or A until the t bits have passed. In 2's complement notation, the loss of these bits causes +ve numbers to be less in magnitude and -ve numbers to be greater. This is unfortunate, since in most theory associated with rounding, it is assumed that due to the equal probability of obtaining +ve or -ve numbers, the error converges on a zero mean value. With truncation on sign and magnitude notation data, the mean error does not converge but becomes more -ve after each truncation, but here, with 2's complement notation, the error becomes -ve at twice the sign and magnitude rate.

The distribution of this error in the final data was of interest here, to determine how much of the data to keep after each array, and hence, the storage word size. Fig. 4-3, details an analytical approach to predict the shape of the distribution after a typical arithmetic operation of addition followed by truncation. Each of the 4 stages represents the distribution of numbers around the 2's complement number, given in binary above the distribution (and decimal, below). The first stage, is the equal probability distribution of an ideal A/D converter with $\pm$ 1/2 bit accuracy. The second stage is the result of addition of two, stage one type distribution numbers, noticing how the distributions now overlap. Dividing the result by 2, but keeping the fractional part, results in the third stage, and only affects the number representation. The operation of truncating this number, is

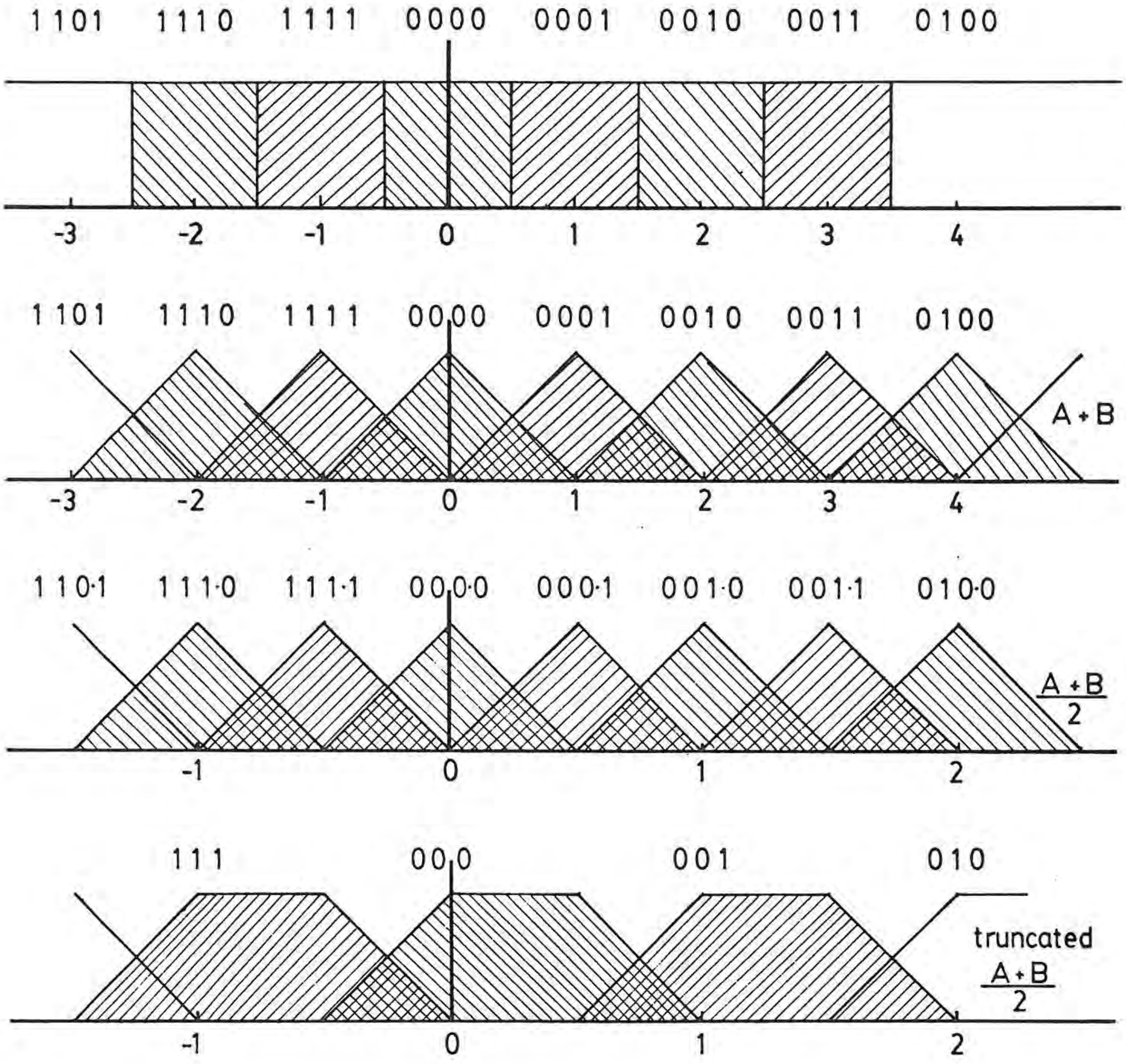


Fig. 4 - 3. Truncation error Distribution.

effectively the same as removing the fractional distributions, and lumping them together with the non-fractional distributions. Thus, the fourth stage shows this and it can now be seen that the mean of each distribution is 1/4 LS bit higher than the number representing it.

This analysis unfortunately, cannot proceed much further, since the convolution of the distributions, after adding, becomes less easy to determine analytically. The "central limit theorem" would usually come to the rescue here and define a normal or Gaussian distribution as result after several convolutions, but, in our case, there is no central limit, since the mean shifts -ve after each truncation. An empirical examination was therefore carried out, using FFTHIST in which one set of data was simultaneously transformed by 2 FFT routines. The first routine had a full accuracy arithmetic unit and the second had arithmetic to simulate the typical worst case truncation, that could occur in the FFT. That is, our serial arithmetic was simulated, but it was assumed that scaling was necessary after each array. The difference between these two sets of data was calculated after each array and printed out in histogram form with 8 bins representing 1 LS bit. To smooth the histograms, 5 different types of input data were used, these being mixtures of sine-waves with or without noise, and then the 5 sets of values averaged after each array.

Fig. 4-4 is the result of this program, run for N = 512. (Due to the fact that two arrays of data had to be kept in memory simultaneously, together with a bit reversed sequence for the W^P calculation, memory space became limited, and an N = 1024 transform was not physically possible).

The histogram after the second array confirms, nicely, the trapezium shaped distribution of Fig. 4-3, and it can be seen to degenerate slowly into what could be termed a "skew Gaussian" distribution after the final array. The fact that this trapezium distribution was not present after the first array is due to the computer program not taking into account the original $\pm$ 1/2 LS bit distribution of the data, and if all the histograms of Fig. 4-4 were convolved with this distribution, a truer likeness to Fig. 4-3 would appear. What is of interest in Fig. 4-4 is that, although the standard deviation of the error increases after each array, the mean value does not, and in fact it appears to remain constant after the third array.

This was a very useful point and it enabled a decision on word size to be made. As was stated earlier, the minimum possible word size was 8 bits and the next highest, easily achieved in hardware, was 12 bits, and so it was decided to split the 4 extra bits into 2 for overflow at the MS end and 2 for truncation errors at the LS end. From

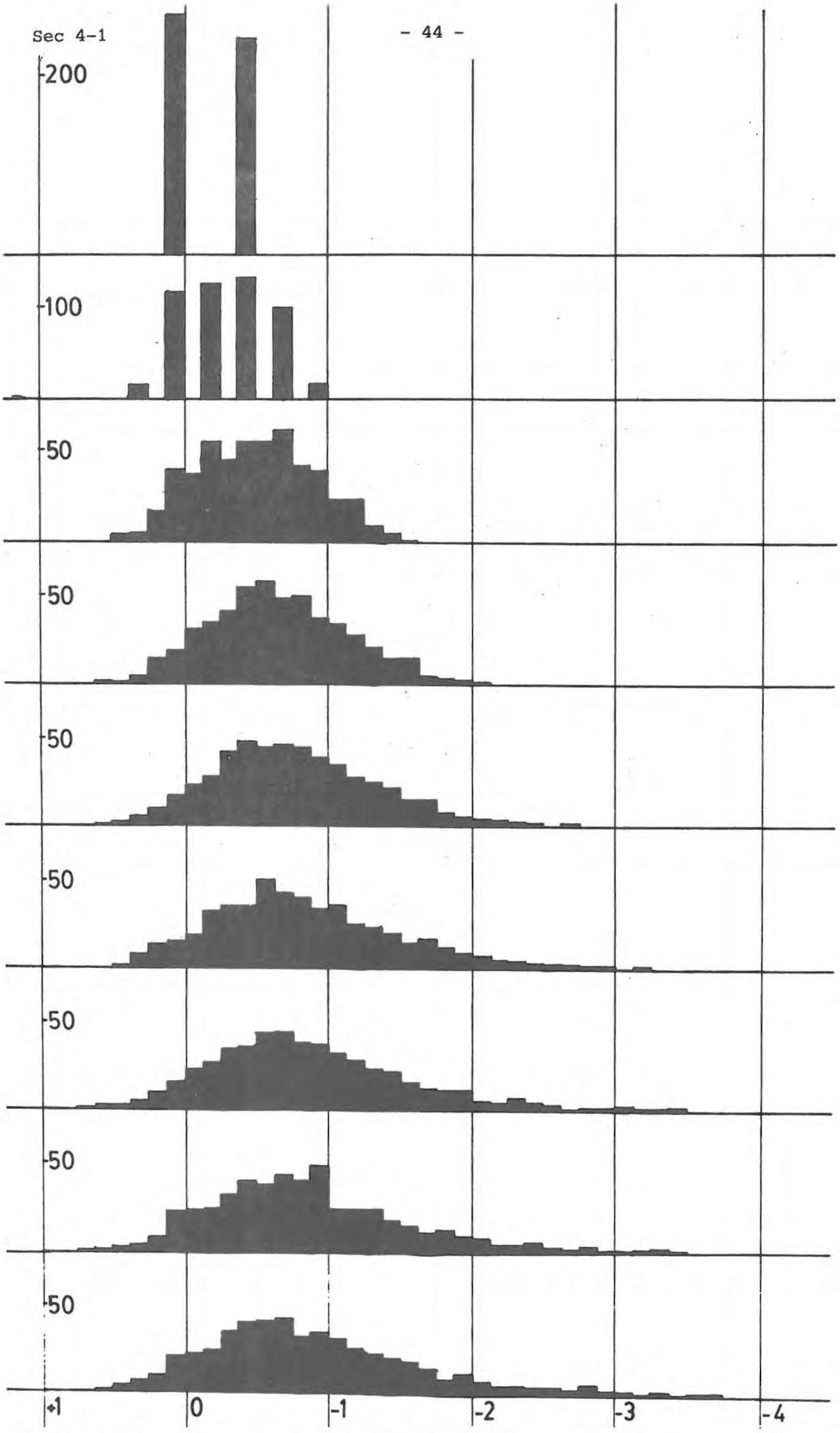


Fig. 4 - 4. FFT truncation error distribution.

the final histogram of Fig. 4-4, and extrapolating to include a 10th array, only the error shown above -2 would appear in the output data since the 2 bits (or 4 times the LS bit value) would encompass the $\pm$ 2 bits of error shown.

Thus, the system would now have a 12 bit data bus of which the middle 8 constitute the data to be finally output, and the arithmetic unit would have to cater for 12 bit operands. The memory would also have to be 12 bits wide, but this could easily be extended since the unit of memory was configured 1024 x 1 bit. One problem was encountered, however, with the multipliers of the arithmetic unit, since they accept an 8 bit parallel operand as the multiplier (the W term in this case) and would have to be cascaded to enable a 12 bit word to be loaded. In the cascaded configuration, however, extra clock pulses would be required just to shift the data out, and hence, would increase the arithmetic time.

In a very thorough paper by Glisson, Black and Sage, 1970 (6), the whole problem of arithmetic word size and "Kernel" or W^P size is dealt with. By fixing two of the variables, such as data word and arithmetic word size, the W^P word size was varied and the mean square error in the result calculated. From input dynamic range and hardware considerations, they choose the optimum data word size as 10 bits, and with the arithmetic carried out at 16 bit precision they conclude that there is nothing to be gained by having W^P greater than 10 bits, since the quantization error due to 10 bit data size is of the same order as the W^P introduced error.

It would appear from this that in our case, with an 8 bit data word and 12 bit arithmetic, that having W^P as 8 bits would be a fair choice. It was decided to test this, and FFTHIST was modified to include two sets of W^P data, one set with full accuracy and the other with variable limits. The accuracy between the two sets of data was only compared after the final array and printed out in histogram format. This program was called FFTHIST2 and is detailed in Chapter 5. A random number generator was used to produce psuedo-white noise as input, and the transform was again computed for N = 512. The results of this test are shown in Fig. 4-5 where W^P size is varied from 6 bit (32-32) to 11 bit (1024-1024) accuracy.

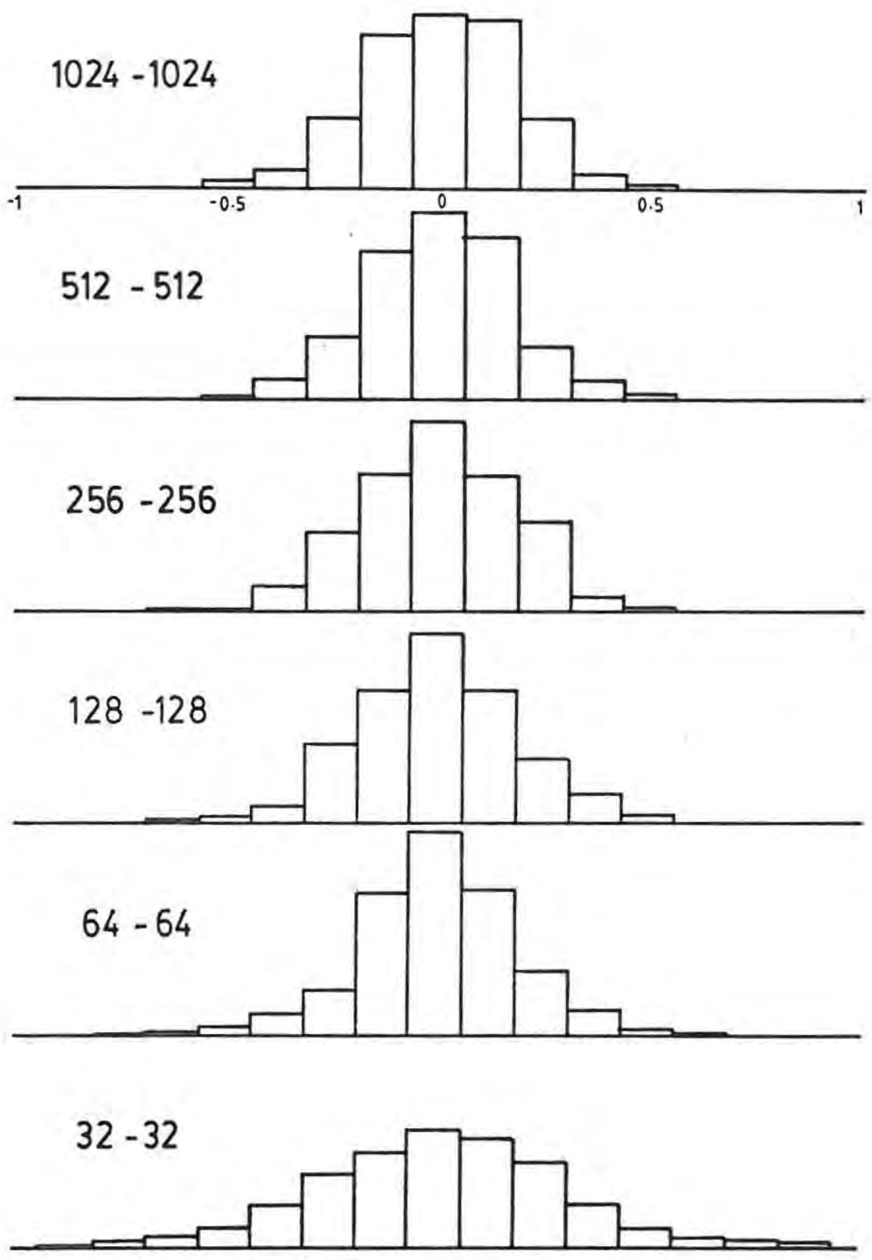


Fig. 4 - 5. FFTHIST2 results.

The x - axis is measured in LS bit values and it can be seen that the distributions above 7 bit accuracy (64-64) remain within $\pm$ 0,5 LS bit value (which is the minimum accuracy of the output data). This, therefore, confirms the findings of Glisson, Black and Sage, and indicates that a W^P table to 8 bit accuracy (128-128) is ample for our requirements.

When the W^P table was being prepared to 8 bit accuracy, a problem was encountered due to the 2's complement notation. The table could not be fit into the range +128 to -128 since in 8 bit 2's complement +128 does not exist. This meant that the W^P value +1 could not be defined, and this is the most frequently used value. To make +128 equivalent to the W^P value +1, the arithmetic unit has to divide by 128, which is an integer power of 2 and thus easily achieved, but if the range was +127 to -127, then the data would have to be divided by 127 and this is not easily carried out.

Therefore, a more detailed examination was made of Fig. 4-5 to ascertain whether a 7 bit (64-64) accuracy W^P table would be acceptable. The standard deviation of each histogram was calculated, assuming that they are approximately normal in distribution with zero mean, and the results are shown in Fig. 4-6. As can be seen, there is only a very small spread in values up to the 64-64 case, and then after that, the standard deviation increases rapidly. This would indicate that 7 bit accuracy is the absolute minimum that can be used in this case and from Fig. 4-5 it can be seen that only a very small amount of error (less than 1%) is outside the $\pm$ 1/2 LS bit range which, as stated earlier, is the minimum value of error. Thus the look-up table was computed to 7 bit accuracy and programmed into ROM. The program WPDATA was used to compute the values and to store them in a bit reversed order at specific locations in computer memory, for access by the PROM programmer routine. The PROMs used were configured 256x4, so that to make up the look-up table of 512 values, each real and imaginary, 8 PROMs were required.

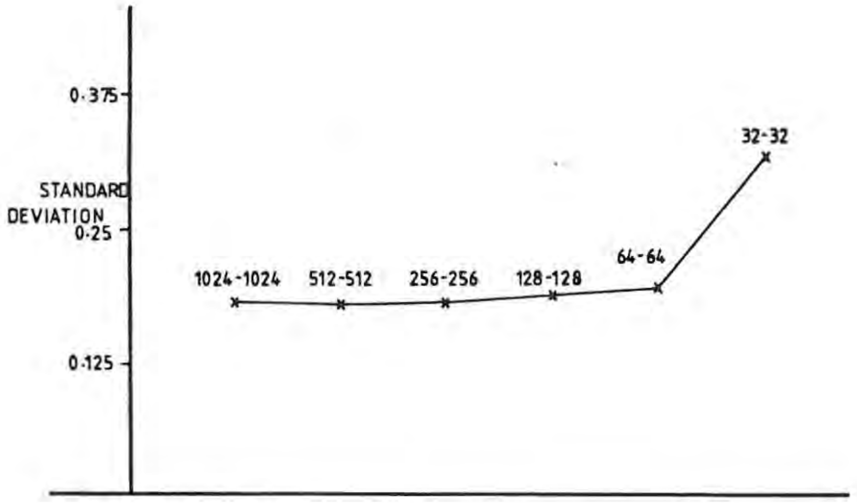


Fig. 4 - 6. W^P error standard deviation.

In conclusion, the design of the arithmetic was finalized as 8 bit output data accuracy, 12 bit arithmetic word size, and 7 bit W^P table size. The input data size need only have been 8 bits, but a 12 bit A/D converter was used for two reasons, firstly to place the quantization error below the middle 8 bits of the arithmetic word, and secondly to prevent overflow from large dynamic range input data.

5. COMPUTER SIMULATION

The object of this exercise was to test the algorithm completely and enable any hardware problems to manifest themselves. The simulation had to be done in such a way that the hardware constraints, such as word size and the truncation method, were reproduced in the software. To this end it was originally thought that a low level language such as assembler should be used since one could approach more closely to the data handling techniques used in the final system. After some effort, however, it became apparent that this approach was far too cumbersome, and in fact it was not simulating the hardware, since most of the functions, such as address control, were done in the hardware by specialised circuits.

Therefore the simulation was programmed in a higher level language (Basic) and as it turned out all the parameters that required simulation could be made available. The control program was made general in size; (that is N could be chosen to be 2, 4, 8, 16, 32, 128, 256, 512 or 1024), and all the early development of the program was carried out with N = 64. It became apparent at this stage that the form of Basic (MSI Basic), being used would be too slow to do any serious simulations with N = 1024 since the program would take 3 hours to complete. A faster form of Basic, SD Basic, was available on the computer, but it differed from MSI Basic in that it was a compiler language. This implies that once the program has been written into a text editor, it has to be compiled and assembled before it can be executed, whereas the MSI Basic is an interpreter language where the program is written into a program already running, which interprets the entered program and does the necessary computation. The MSI language is ideal for development since small changes can be made instantly, but due to the interpreter structure, a lot of time is wasted in transferring between the entered program and MSI itself. SD Basic, however inconvenient for development, has the extra speed to be able to do the N = 1024 transform in 15 minutes, and so the MSI program was transferred into this language almost directly.

The two forms of the program are detailed in this chapter and listings of both are found in Appendix 1. The results of the simulation however are discussed where relevant in other chapters.

5 - 1 Program development

Using the theory developed in Section 2-2, in particular the signal flow graph of Fig. 2-9, the control block diagram for the Cooley Tukey algorithm was drawn up, as shown in Fig. 5-1. A and B represent the two addresses of the operands required for the FFT butterfly, whereas X and Y are address limits, which cause the address sequence to change from array to array. This diagram does not include any provision for unscramble or power spectrum control.

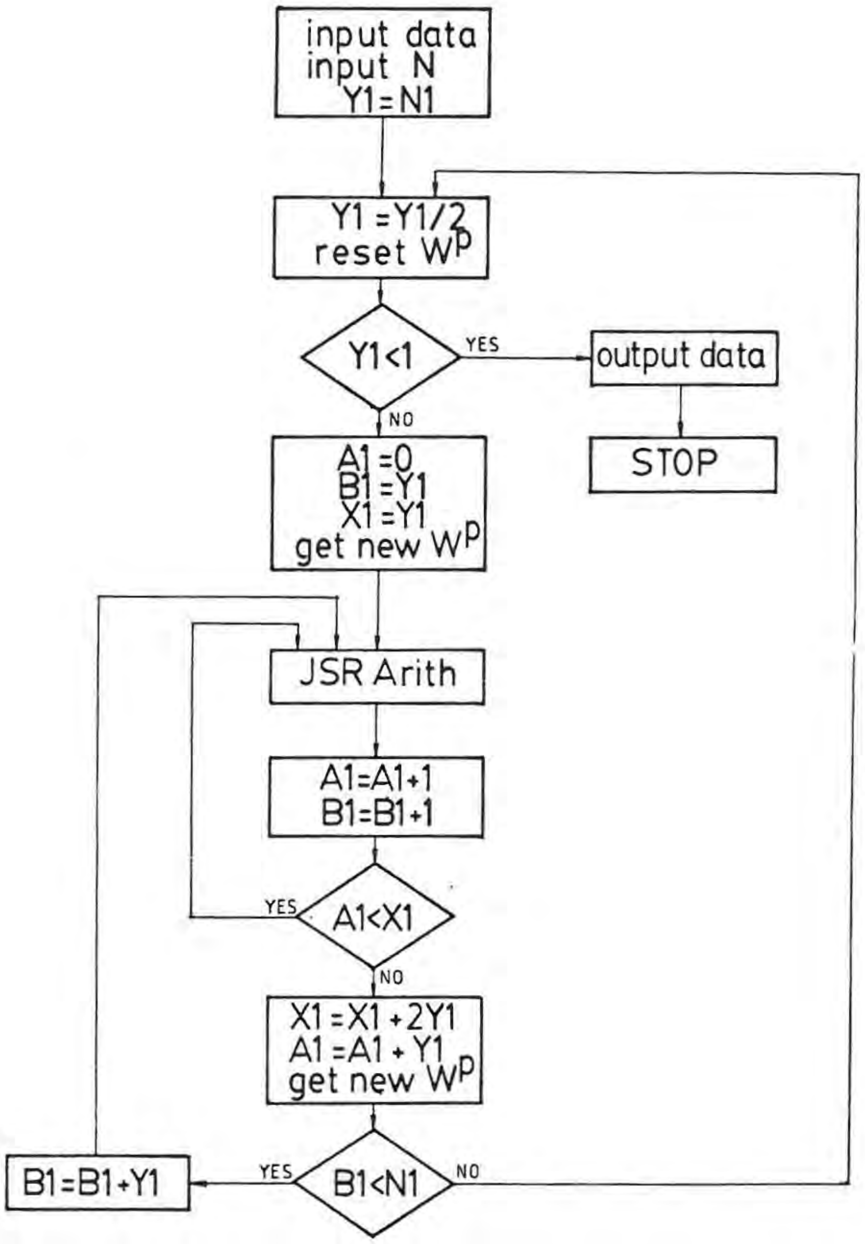


Fig. 5 - 1 Cooley Tukey algorithm control

As was mentioned earlier, most of the development of this part of the program was undertaken in MSI Basic and so the listing below is an MSI Basic listing of the program which represents the control of Fig. 5-1.

```
  5  Input N1                                          (Input data)
200  Y1 = N1
210  Y1 = Y1/2                                         (reset W^P)
220  IF Y1 < 1 THEN 300
230  A1 = O : B1 = Y1 : X1 = Y1                        (get new W^P)
240  GOSUB 500                                         (Arithmetic Routine)
250  A1 = A1 + 1 : B1 = B1 + 1
260  IF A1 <> X1 THEN 240
270  X1 = X1 + 2 x Y1 : A1 = A1 + Y1
280  IF B1 <> N1 THEN B1 = B1 + Y1 : GOTO 240          (get new W^P)
290  GOTO 210
300 (output data)
     STOP
```

The remarks in parenthesis refer to subroutines not listed, which will change depending on their implementation. For example, the W^P table, during the early development of this program, was calculated from a bit reversed sequence of addresses which was stored in a data file, but later the whole W^P table was stored in a data file in bit reversed sequence in an attempt to reduce the computation time.

This simple control program therefore became the heart of all the test programs developed, with the associated subroutines handling the simulation of the parameters of interest. Most of the simulation work was centered on the arithmetic routine with parameters such as operand word size being adjusted to test their effect on the accuracy of the calculations. The INT (integer) statement was used to keep the operands within a particular word size, and also to simulate the truncation process being carried out in the hardware system. The INT statement in both MSI and SD Basic takes the integer value less than or equal to the given number. Thus 3.35 would yield 3 but -3.35 would yield -4. This slight difference between the definition of the integer function for Basic and that, for instance for electronic calculators, is quite useful since the Basic definition simulates the operation of truncation of 2's complement numbers as detailed in Section 4-1.

A typical arithmetic routine to simulate the FFT butterfly is :

```
500     T1 = (A4(B1) * W1 - A5(B1) * W2)/128
        T2 = (A4(B1) * W2 + A5(B1) * W1)/128
        A4(B1) = INT((A4(A1) - T1)/Z1)
        A5(B1) = INT((A5(A1) - T2)/Z1)
        A4(A1) = INT((A4(A1) + T1)/Z1)
        A5(A1) = INT((A5(A1) + T2)/Z1)
        RETURN
```

In this example, the data is stored in arrays A4 and A5, being the real and imaginary parts respectively, and W1 and W2 represent the W^P values, real and imaginary respectively, which have been calculated prior to entering the arithmetic routine. The W^P values have been calculated within the range $\pm$ 128 and are integer, but the complex multiplication carried out at T1 and T2, is allowed a fractional part since as detailed in section 4-1 the serial arithmetic retains these LS bits. The 2's complement truncation is simulated in the final four equations, where the term Z1 scales the results by 2 when required.

This type of arithmetic is used in two simulation programs, FFTHIST and FFTHIST2, both written in SD Basic, and a listing of each can be found in Appendix 1.

5 - 2 FFTHIST

This program was written to simulate fully the arithmetic unit, in particular the truncation after each set of calculations. In order to compare the errors involved with this truncation, a standard or accurate arithmetic unit had to be simulated as well as the test arithmetic routine, and the differences between the results of the two arithmetic routines computed after each array. Input data had therefore to be split, for use by each arithmetic routine and thus A4 and A5 are the real and imaginary parts of the truncated arithmetic data, and A6 and A7 are the real and imaginary parts of the standard data. The W^P values W1 and W2 are identical for each arithmetic unit to prevent another variable from affecting the results. To calculate the W^P values, the program requires a bit reversed sequence, and this is computed by a program called BRPOKE which stores the sequence at memory location DOOO upwards. Therefore, BRPOKE must always be executed first, before FFTHIST. However, once the bit reversed sequence is in memory it will not be corrupted and thus need only be computed once per session.

The histogram type output is produced by sorting the calculated errors into bins, each bin representing an eighth of a LS bit of the output data. The number of times such an error is computed, is then printed out on the computer terminal or on an auxilliary printer.

The input data for this program is calculated internally from parameters that are input through the terminal at Run Time, so that data with mixtures of cosine waves and random noise can be produced. To smooth the resultant histograms, several transforms of different sets of data were computed and the histograms averaged. The results of this program are discussed in section 4-1.

5 - 3 FFTHIST2

This program retains the same input data generation and output histogram routines of FFTHIST, but the arithmetic being simulated is different. The object of this program is to ascertain the optimum size for the W^P term. In FFTHIST it was fixed at $\pm$ 128, but here a standard arithmetic routine uses fully accurate W^P values to compare with an arithmetic routine using variable W^P size selected through the input terminal. Two sets of data are computed as in FFTHIST, but both arithmetic routines are identical and correspond to the standard or accurate type used in FFTHIST. Again the bit reversed sequence is required, and is produced by BRPOKE and stores at DOOO upwards. The results of this program are discussed in Section 4-1.

5 - 4 Other programs

Once the word size for the W^P term was decided, a program was written to calculate the full table of values rounded to the required significance, and to store them in a data file. This program was called WPDATA, and was followed by another program, WPPOKE written to access the data file and store the table of values at memory location COOO upwards, to enable the PROM programmer to gain access to them. Listings of both of these programs can be found in Appendix 1.

In conclusion, the FFT simulation programs produced for this project were intended to be as general as possible, in order that other software users may have access to them. However, due to the features that had to be simulated, none of the programs are really efficient in computing a general FFT. An example of MSI Basic program FFT16, developed prior to changing to SD Basic, is given in Appendix 1, but a prospective FFT software user is advised to concentrate on the SD Basic program structure due to its speed and ability to handle the large arrays of data required in the FFT. The programs served their purpose well, manifesting several possible hardware problems and giving a general confidence that the theory behind the hardware was sound.

6. MICROPROGRAM CONTROL

The functional operation of the analyser has been fully discussed so far, but no mention has been made of the overall control. In section 3-2, the choice of a hardware system was made, but this still required software in some form to do the overall control. It was decided to use ROM as a storage medium for the software because of its fast access time, (typically 35 nanosecs) and also its nonvolatile nature, rather than having to load the system program after each power down. The sequencing of the address of this ROM was originally intended to be carried out by a specialized binary counter which could branch forward and backward under control of system parameters. However, Advanced Micro Devices had done similar design work previously and were marketing an integrated circuit called a microprogram sequencer Am 2909, which included the address counter and all the control for branching. Therefore, the microprogram control design became centered around the Am 2909.

The concept of controlling a system via microcode stored in memory is not new, but in the past it has been restricted to large systems only, where the hardwired logic alternative approach becomes too unwieldy. Its first large scale use came in minicomputer control where op codes from the main memory were decoded via a micro-instruction memory into the control signals required for a particular operation, and this has enabled many firms to emulate other systems just by changing the micro-instruction memory. This approach has become available by the advent of fast, low cost memory in the form of ROM which can be easily programmed to any specification.

6 - 1 Microprogram Sequencer

The micro-instruction memory in this application contains outputs for all the control lines of the system (e.g. arithmetic unit and memory) together with outputs which control its own addressing and sequencing. The start address and subsequent major branch addresses are obtained from outside the memory, enabling the overall sequence of operations to be laid out in correct order, with the micro-instruction memory requesting the next operation address upon completion of the present operation.

The major branch addresses are obtained from another ROM, termed the 'start address ROM', and this is also controlled by the micro-instruction memory. It has two parts however, which are selected from the front panel to switch the system from a fully automatic FFT routine, to a sequence of test routines; this being achieved by controlling the M.S. address bit of the ROM U12.

Due to the repetitive nature of the FFT algorithm, the operations defined under a single start address can encompass the whole algorithm. That is, a sequence of three start addresses for example, could define the operations : FFT, Power Spectrum, and output routines, and under each of these addresses, there might be as many as 100 micro-instructions which together, carry out the operation.

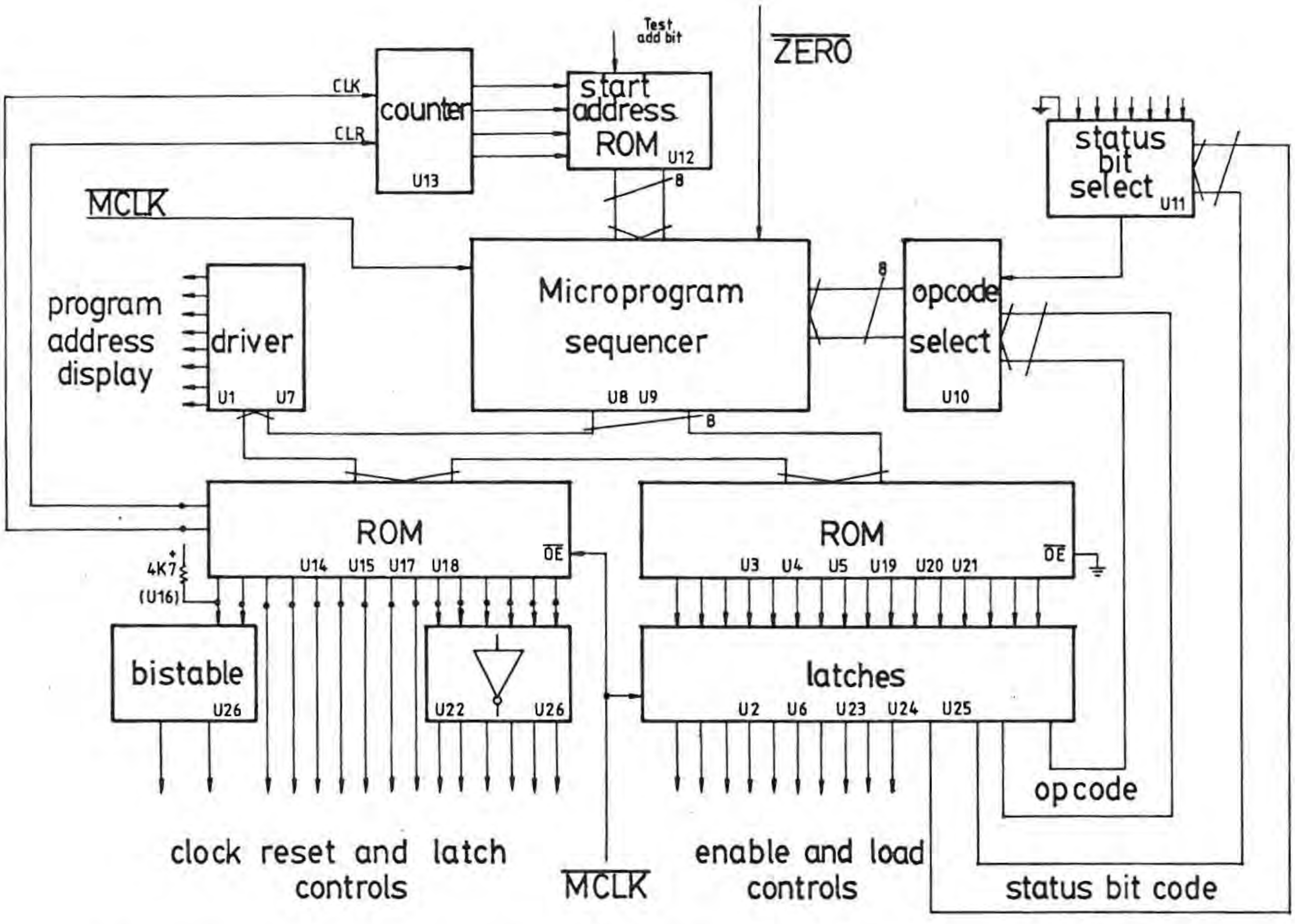


Fig. 6 - 1. Micro control board block diagram.

The micro program sequencer then controls the order of these micro-instructions. Fig. 6-1 shows the block diagram of the system micro program control with the Am 2909s (U8 & U9) as the central feature. The micro-instruction in this case is 40 bits wide and built up from 256x4 bit ROMs so that to address 256 words, an 8 bit address is required, and to handle this, 2 x Am2909 are cascaded. The control lines are split into two types; those that occupy only one clock cycle, such as resets and latch controls, and those that last for several cycles, such as enable the load controls. The latter type are latched, and these include the controls for the microprogram sequencer (status bit code and op code). Of the single cycle type, there are those that are active high, those active low, and two that are toggle types. To affect a branch, a status bit is selected for test, and a positive result causes the next op code to be changed, directing the microprogram sequencer to change to a different source for its next address. Fig. 6-2 details the structure of the Am 2909 and how the op codes change the address.

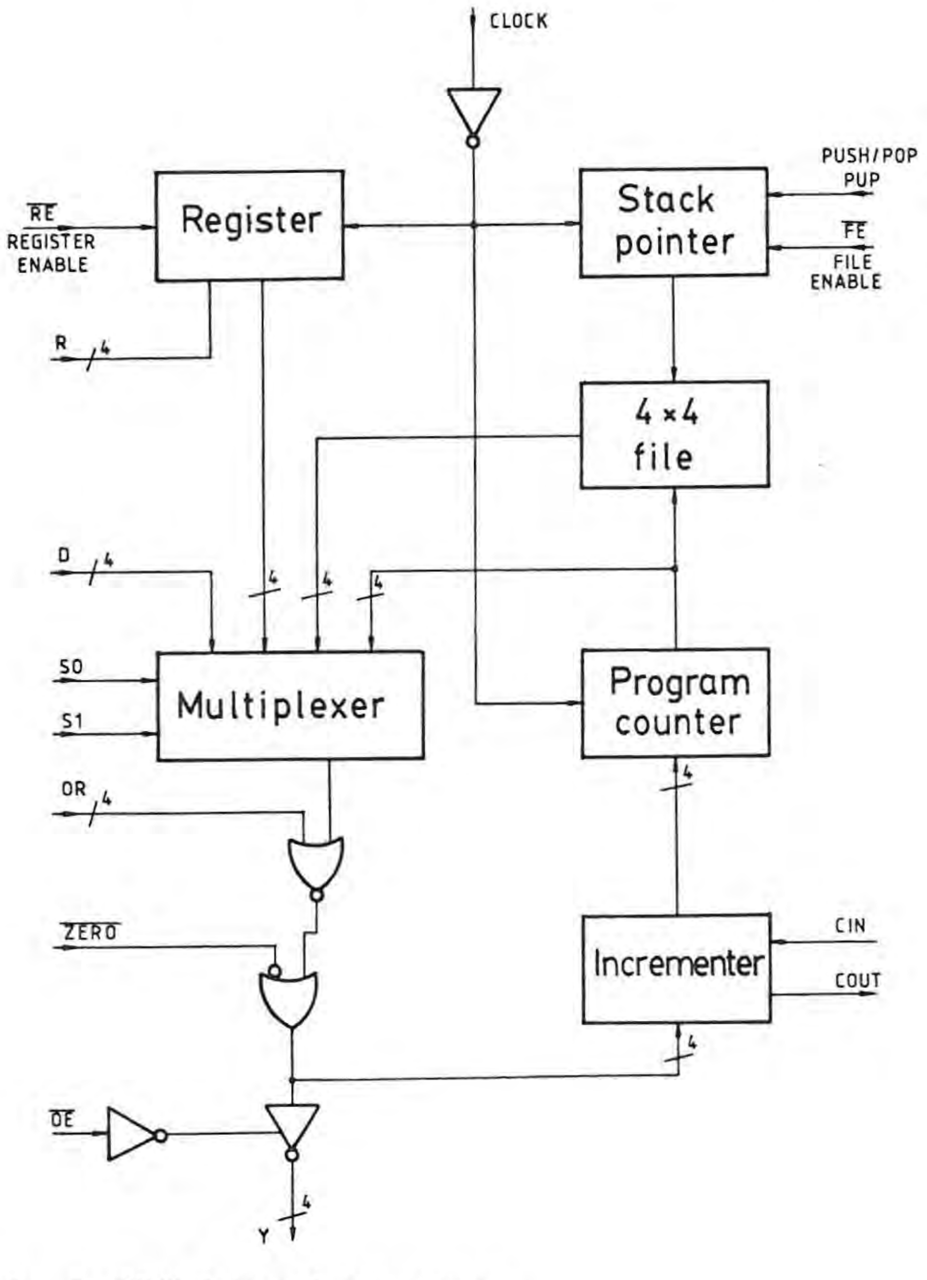


Fig. 6 - 2. Am 2909 internal structure.

S0 and S1 select the source for the next address from either direct, register, stack or PC (program counter), whereas PUP and $\overline{FE}$ control the loading of the stack and positioning the stack pointer, OR inputs mask out the address bits and $\overline{ZERO}$ takes all outputs to the low state. The direct D and register R inputs are paralleled in our system, $\overline{OE}$ is permanently tied low, Cout, of the LS Am 2909, is tied to Cin of the MS Am 2909, and only OR0 and OR1, of the LS Am 2909 are used. Combinations of the eight control lines, OR0, OR1, $\overline{RE}$, PUP, $\overline{FE}$, Cin, S1 and S0, are selected from ROM U10 to affect the particular microprogram sequencer functions, and $\overline{ZERO}$ is connected to a Master Reset on the front panel of the analyser to enable the whole system to be stopped.

6 - 2 Instruction Set and Programming

The instruction set referred to here, is that pertaining to the microprogram sequencer control, and is the set of combinations of the 8 control lines that are stored in ROM (U10). U10 is termed the OP CODE mapping PROM since it converts the 4 bit op code plus status bit, into the 8 control lines, and can thus contain $2^5 = 32$ different combinations or functions. During development, a large number of op codes were used, but for the final system, only 7 were selected. Before discussing the individual op codes, it would be best to analyse the programming required.

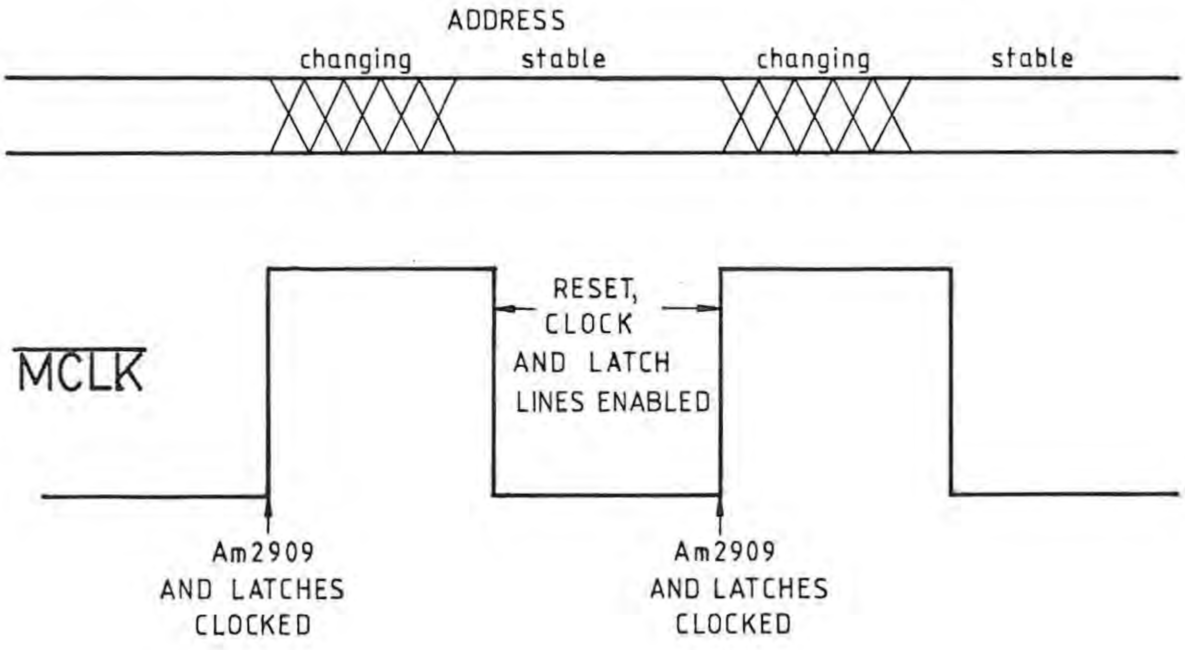

ADDRESS
changing
stable
changing
stable
$\overline{MCLK}$
RESET, CLOCK AND LATCH LINES ENABLED
Am2909 AND LATCHES CLOCKED
Am2909 AND LATCHES CLOCKED

Fig. 6 - 3. Micro-instruction clock cycle.

As mentioned earlier, several of the system control lines, including the op code select lines, are latched, and this greatly affects the analysis of the program. Referring to Fig. 6-3, which is the clock timing diagram for the whole microprogram control board, it can be seen that the latches are clocked at the same instant as the Am 2909s, and also, all the unlatched lines are enabled during the preceding half cycle. Therefore, the latched control lines are half a cycle behind their unlatched counterparts and must be considered as effective in the next clock cycle, when programming. Hedges 1978 (9) explains that, by the addition of the latch "the MCU becomes a next-address calculator instead of a current address calculator - in other words, the MCU is one instruction ahead of the data manipulation". In this system, the latched control lines can be considered as set up conditions for action that will take place in the next cycle. The status of the system is monitored by 7 status bits, which mostly change in response to a transition in an unlatched control line, such as a clock, but asynchronous status bits are latched prior to testing to ensure correct monitoring.

The programming, therefore, consisted of selecting the control line functions required, ensuring that latched lines were set up in the previous cycle and then deciding where to obtain the next instruction. Particular problems were encountered when programming that had to be overcome in special ways, and in one case, with special hardware. These were mainly to conserve ROM space, since several routines had to be written in the 256 words available. Firstly, during the FFT, it became necessary to scale the data after the arithmetic unit, but this scaling was not regular each array, only alternate arrays. Therefore, to write a general routine to handle this, the special case of scaling each array was programmed, and on each alternate array, a status bit caused one divide by 2 to be skipped. To skip an instruction, however, meant in effect, incrementing the PC by 2 and this was not a standard function; however by utilising the ORO control line, it could be achieved. The ORO line forces the LS bit of the ROM address to 1, so that if the divide by 2 to be skipped was positioned at an even address, then a true test of status bit would force that even address bit to a 1, hence missing out that instruction.

The second problem, also in the FFT routine, was where a particular function was repeated many times consecutively, hence being wasteful of memory space. To overcome this, a counter was used which was incremented by $\overline{\text{MCLK}}$, and a status bit set to detect the count of 7, this then causing the microprogram sequencer to jump out of a single instruction loop. The counter was cleared prior to entering the loop, and the instruction then carried out 7 times while in the loop.

The third problem encountered, was how to indicate to the program that a particular test routine had been requested on the front panel, by only utilising one status bit. The solution was to use a priority encoder on the front panel rotary switch to generate a count proportional to the number of the test routine and then to clock a counter from a program loop until the two counts were equal. This proved an efficient program, since as well as clocking the counter from the program, the start address ROM could be clocked simultaneously, thus automatically setting up the required test's start address.

	OPCODE		OR1	OR0	$\overline{RE}$	PUP	$\overline{FE}$	Cin	S1	S0
TEST CORRECT	0 0 0 0 0									
	0 0 0 0 1	PC	0	0	1	0	1	1	0	0
	0 0 0 1 0	PC	0	0	1	0	1	1	0	0
	0 0 0 1 1									
	0 0 1 0 0	PCOR0	0	1	1	0	1	1	0	0
	0 0 1 0 1									
TEST FALSE OR NO TEST	1 0 0 0 0	PC	0	0	1	0	1	1	0	0
	1 0 0 0 1	JD(TPC)	0	0	1	0	1	1	1	1
	1 0 0 1 0	JS(TPC)	0	0	1	0	1	1	1	0
	1 0 0 1 1	PCSTK	0	0	1	1	0	1	0	0
	1 0 1 0 0	PC(TOR0)	0	0	1	0	1	1	0	0
	1 0 1 0 1	PCPOP	0	0	1	0	0	1	0	0

Fig. 6 - 4. Opcode mapping ROM U10.

As mentioned earlier, only a subset of the possible op codes were used and these are tabulated in Fig. 6-4. In the lower half of the table are the normal 'no test selected' operations, and those with parenthesis in their psuedonym can be selected to test a status bit. Depending on the outcome of the test, the MS bit of the op code directs to the 'test correct' or 'test false' op code. For example, JD(TPC) would normally (with no test selected) jump to the address which appeared at the direct (D inputs) of the Am 2909 from the start address ROM U12. However, if a test was selected by defining a status bit number, and the test was true, then the sequencer would be instructed to obtain the next address from the program counter (PC).

The psuedonyms can be read as follows :

PC	Next address from the PC.
JD(TPC)	Jump direct, or upon test from PC.
JS(TPC)	Jump stack, or upon test from PC.
PCSTK	Next address from PC and load the stack.
PC(TORO)	Next address from PC, or upon test force the LS bit to a 1.
PCPOP	Next address from PC and pop the stack.
PCORO	Next address from PC and force the LS bit to a 1.

6 - 3 Program Structures

Using the instruction set defined by Fig. 6-4, normal program structures such as branching and looping could be configured, and those that were used in the present program are detailed below.

Looping, i.e. repeating one or more instructions under control of a status bit, was used throughout the present program and was carried out using the JS(TPC) operation :

```
enter routine                          PC
save present address + 1 for return    PCSTK
                                     ⎧ -
                     routine repeated⎨ -
repeat routine or jump out           ⎩ JS(TPC)       SB1
proceed to next routine                PC
```

Such loops can be nested 4 deep, and PCPOP used to roll down the stack.

Branching always uses the direct inputs, i.e. the start address ROM U12 and hence the instruction JD(TPC), and performs a simple jump operation.

leave previous routine	JS(TPC)	SB5
obtain next address	JD(TPC)	
enter next routine	PC	
increment start address ROM for next routine	PC	

Upon entering a routine in the Full FFT program, the start address ROM is incremented in preparation for proceeding to the next routine. However, it is not incremented immediately, since the first instruction is itself obtained from the start address ROM and would thus cause it to try and increment its own address. Therefore, a PC instruction is always the first instruction of a routine, and the start address ROM is incremented at a later instruction.

The system test routines, which are detailed in Chapter 7, break the Full FFT routine up into specific areas, such as Power Spectrum, or Unscramble, and follow each of these with an output routine to enable oscilloscope diagnosis of the results. These output routines, however, repeat indefinitely to give a steady oscilloscope trace and rather than write a special routine for outputting data, the Full FFT output routine is entered at a point after the start address ROM has been incremented so that the program always returns to its own start address, and hence repeats.

	ROUTINES	START ADDR.	
0 0	INITIALISE FFT	0 F	
0 1	FFT	2 8	
0 2	UNSCRAMBLE	8 2	
0 3	POWER SPECTRUM	5 E	FULL FFT
0 4	OUTPUT REAL	4 9	
0 5	OUTPUT IMAGINARY	4 7	
1 0	INITIALISE TEST	0 8	
1 1	TEST1 MEMORY	1 6	
1 2			
1 3	TEST2 FFT	2 8	TEST
1 4	OUTPUT BIT REVERSED	4 B	
1 5	TEST3 UNSCRAMBLE	8 2	
1 6	OUTPUT BIT REVERSED	4 B	
1 7	TEST4 POWER SPECTRUM	6 0	

Fig. 6 - 5. Start address sequence ROM U12.

The sequence of the routines, as stored in the start address ROM U12, is shown in Fig. 6-5 and a brief description of each routine is given below. The full programs are given in Appendix 2.

6 - 3 - 1 Initialise FFT

This routine is entered immediately the master reset is released if the Full FFT is selected. Its purpose is to reset address counters etc., and then wait for the sampler to indicate, via SB1, that a memory is full. It then passes control immediately to the FFT routine.

6 - 3 - 2 FFT

Into this routine is programmed the whole control to carry out the complex FFT butterfly arithmetic. It obtains the operands A, B, and W by controlling the addresses of each, latches them into the arithmetic unit, clocks them through the arithmetic unit, scaling the result as required, and finally stores A^* and B^* back in memory. It uses SB7 to count 7 clock cycles (and hence repeat an instruction), SB3 to indicate when to scale, and SB5 to indicate the end of the FFT algorithm.

6 - 3 - 3 Unscramble

This separates the two receiver signal transforms that were originally stored in the real and imaginary parts of the input memory, and requires a bit reversed address sequence since the data output of the FFT is in bit reversed order. It uses SB2 to indicate that it has completed the separation, and SB6 to wait until the outside computer is ready to accept the transformed data.

6 - 3 - 4 Power Spectrum

To indicate to the outside computer where the received echoes are, this routine computes the power spectrum $P = R^2 + I^2$ (but does not take the square root) and outputs the information to the computer, for both receiver 1 and 2. It also requires a bit reversed address sequence and uses SB2 to indicate the end of the computation.

6 - 3 - 5 Output Real

This outputs the whole of the real memory to the outside computer, receiver 1 transform, followed by receiver 2 transform, using a bit reversed sequence, and also using SB2 to indicate completion.

6 - 3 - 6 Output Imaginary

This is the same routine as above, but it is accessed earlier so that R/I output multiplexer can be toggled. The front panel R/I selector switch is disabled if the Full FFT is selected.

6 - 3 - 7 Initialise Test

This routine is the first entered if the test switch indicates a system test is to be carried out. It decides, using SB4, which test number is requested and passes control to that routine, having reset address counters etc. Each test routine is assumed to require two start addresses, one for the operation under test, and one for the output routine, and it can be seen from Fig. 6-5 that Test 2 and Test 3 conform to this. However, Test 1 and Test 4 have output routines integrated with the test operation, and therefore, only require the one start address, hence the gap in the ROM at location 12.

6 - 3 - 8 Test 1 Memory

This is a special test which outputs the contents of the memory to an oscilloscope in straight order to check for memory faults. The first operation, upon entering, is to change over the memories, and therefore control is only passed to this routine once the sampler has indicated, via SB1, that it has filled a memory. SB2 is used to control the address looping, and an oscilloscope trigger pulse is positioned in the return loop.

6 - 3 - 9 Test 2 FFT

This is the identical routine to that accessed in Full FFT operation, but it is followed immediately by a bit reversed output routine to enable a display of the transform before any further data arithmetic is carried out. Control of displaying either the real or imaginary part, is from the front panel. The first operation upon entering is to change over the memories, the same as Test 1.

6 - 3 - 10 Test 3 Unscramble

Again identical to the Full FFT operation, and again followed by the bit reversed output routine, but it does not change over memories.

6 - 3 - 11 Test 4 Power Spectrum

The identical routine to the Full FFT operation, which has the output routine built in, but for Test, the routine is accessed later in the program, and it does not change over memories.

6 - 4 Control Lines

There are a total of 41 control lines and each has been given a psuedonym appropriate to its action for ease of reference. Here is a list of the control lines with their operations :

$\overline{\text{VAL}}$	Output data valid
$\overline{\text{INT}}$	Interupt outside computer to request DMA
$\overline{\text{RE1}}$	Enable memory to read to the system
$\overline{\text{WEI}}$	Enable memory chips for writing
OC1, OC2 OC3, OC4	Op code select lines for the microprogram sequencer
Mux imp/CLK	Dual function. To the arithmetic unit it multiplexes the inputs to the output shift registers when computing the Unscramble Routine. To the test address counter it acts as a clock
SBA, SBB, SBC	Status bit select lines for the microprogram sequencer
$\overline{\text{Scope trigger}}$	Taken out to an external socket
$\overline{\text{TCLR}}$	Clears the $\overline{\text{MCLK}}$ counter which is used to generate SB7 on the test control board
$\overline{\text{R1}}$	A reset used in the address generator to return the sequence to that of the 1st FFT array
$\overline{\text{SR1}}$	A general reset used at the beginning of a routine to initialise the address generator
$\overline{\text{CL1}}$	A general reset for the arithmetic unit
CKRI	A clock that toggles the R/I line that controls the output bus multiplexer
$\overline{\text{LD2}}$	Loads the multiplier W^P into the arithmetic unit
CKW^P	A strobe to the address generator to synchronise the change in W^P

$\overline{\text{BR/S}}$	A reset to the start address ROM U12 counter
$\overline{\text{CK1}}$	A clock that shifts the operand B into the multipliers of the arithmetic unit
$\overline{\text{CKM}}$	A clock that toggles the 2 + $\overline{1}$ line that in turn swaps memories
$\overline{\text{CKDA}}$	Latches output data for access by the D/A or the output bus
$\overline{\text{CK4}}$	Clocks the address sequence counter
$\overline{\text{CK3}}$	Clocks the multipliers and adders in the arithmetic unit
$\overline{\text{BCLK}}$	Clocks the start address ROM U12 counter
$\overline{\text{CK2}}$	Clocks the operand A into the adders in the arithmetic unit
A+B	Used in the address generator to select either A or B from memory
$\overline{\text{en(N-n)}}$	Used in the address generator to enable a special bit reversed sequence for the unscramble routine
$\overline{\text{enn}}$	Enable for the normal bit reversed sequence from the address generator
$\overline{\text{en(A+B)}}$	Enable for the special FFT address sequence
$\overline{\text{en1}}$	Output enable of result B^* from the arithmetic unit
$\overline{\text{en4}}$	Output enable of operand W^P from the W^P ROM
$\overline{\text{WE2}}$	Bus driver enable for writing from the system to memory
$\overline{\text{WE3}}$	Bus driver enable for writing from the sampler to memory
$\overline{\text{LD3}}$	Loads the operand A into the arithmetic unit
$\pm/\overline{\text{CLR}}$	Dual function. To the arithmetic unit it changes an adder into a subtractor for computing the Power Spectrum. To the test address counter it acts as a clear
$\overline{\text{en3}}$	Output enable from the B shift registers of the arithmetic unit
$\overline{\text{LD1}}$	Loads the operand B into the arithmetic unit
$\overline{\text{en2}}$	Output enable of result A^* from the arithmetic unit.

7. TEST ROUTINES

Due to the complexity of the system hardware, it became necessary, during development, to allow for the testing of certain aspects of the algorithm without recourse to complicated diagnostic equipment. It was decided to include these tests in the final system as a means of periodic checking of its operation, and also to facilitate fault finding.

There is provision on the front panel switch and internally for 8 test routines, but at present only 4 are deemed necessary, and these are selected by the front panel switch. A description of the operation of these controls is given in section 7-1 to augment the test procedure detailed later. Section 7-2 details how the pre-programmed tests can be used to monitor the hardware performance, whereas section 7-3 describes the function of the analyser in respect of separating the echoes contained in an actual sounding from the ionosphere.

7 - 1 Front Panel Controls

The prime function of these controls is to switch the analyser from its full FFT microprogram to one of the microprogrammed tests. Other parameters such as system clock and sampler control, can be changed through the front panel and they affect both the tests and the full FFT and therefore must be correctly positioned at all times. Fig. 7-1 shows a facsimile of the front panel with its controls and labelling.

The system clock can be controlled by the top switch labelled RUN, TEST or MAN (manual) and the adjacent push button, thus enabling RUN speed of approximately 10MHz, TEST speed of 400KHz, or Manual single step. The slower speed is used for the Test routines to enable an accurate trace to be output from the D/A converter, which has a limited frequency response. The next pair of switches below the clock controls are to select the source for system start. Labelled MAN, AUTO, or ONE-SHOT, they select continuous control, external control, or single transform control respectively. This switch only affects the sampler (Analogue Input) and controls the 1024Hz sample clock, which in turn starts the microprogram by indicating that a memory is full. With the switch in the ONE-SHOT position, and by pressing the adjacent push button, the 1024Hz is enabled until a memory is full, and then disenabled until the push button is pressed again.

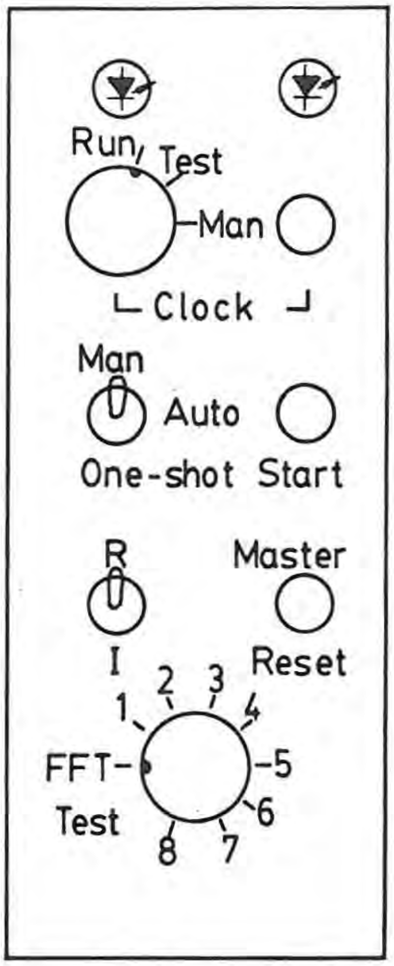


Fig. 7 - 1 Front Panel Controls.

MASTER RESET is a push button which forces the microprogram to its zero program address, and when this push button is released, the microprogram checks to see if a test has been selected, and if so, which one. With the rotary switch at the bottom of the panel in the FFT position, no tests are selected and the unit carries out the full FFT routine. The other positions 1 - 8 of this switch select the test routines : Test 1 Memory, Test 2 FFT, Test 3 unscramble and Test 4 Power Spectrum, and each of these tests is detailed at the program level in Chapter 6. The switch labelled R or I selects which bus, real or imaginary is to be output under test and is ineffective when the bottom rotary switch selects FFT. Finally, the two LEDs (light emitting diodes) at the top of the panel monitor the sampler real and imaginary data buses and indicate when the input signal is going over range. When the sampling ceases, the LEDs refer to the range of the last data word loaded, and therefore, for true monitoring of the input signal level, the sampler must be running. (The intermittant flashing of the LEDs indicates the optimum level).

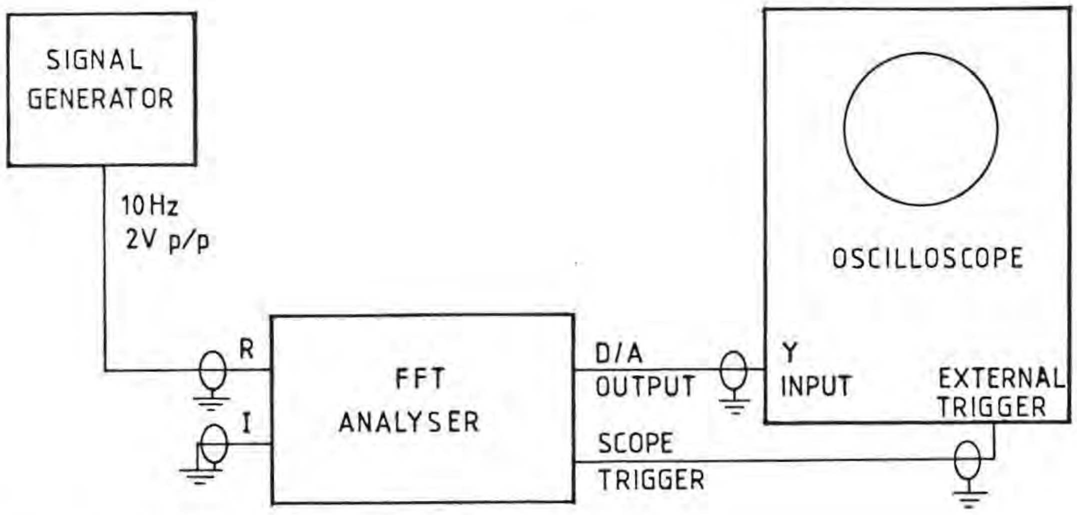


Fig. 7 - 2 Hardware Test Interconnections

7 - 2 Hardware Test

This test involves setting the analyser up as shown in Fig. 7-2 with a sine wave being fed into the "real (receiver 1) input" and the "imaginary (receiver 2) input" shorted to earth. The "D/A output" and "scope trigger" are special outputs specifically for use when testing the analyser, and they enable a steady oscilloscope trace of the output data to be obtained. With the system clock set to TEST, the system start set to MAN, and FFT/Test switch set to 1, pressing MASTER RESET will initiate the memory test and by selection of R or I the traces shown in Fig. 7-3 can be obtained. The real trace can be seen to begin repeating at the right hand edge, and it can also be seen that there are not exactly 10 cycles of the wave in the memory. If there had been exactly 10 cycles, then all the energy in the transform would be in one frequency channel, but since this is not the case, according to the Discrete Fourier Transform theory developed in Chapter 2, the transform energy will be present in the adjacent channels with amplitudes defined by the sin x/x function.

Switching the FFT/Test switch to 2 and pressing MASTER RESET initiates Test 2 FFT, and the transform of the test signal is obtained as Fig. 7-4 shows. The sin x/x side lobes are evident on either side of the main signal peak, and notice the even nature of the real trace compared to the odd nature of the imaginary trace, i.e. both real +ve and -ve frequency spikes are the same polarity, whereas the imaginary spikes have opposite polarity.

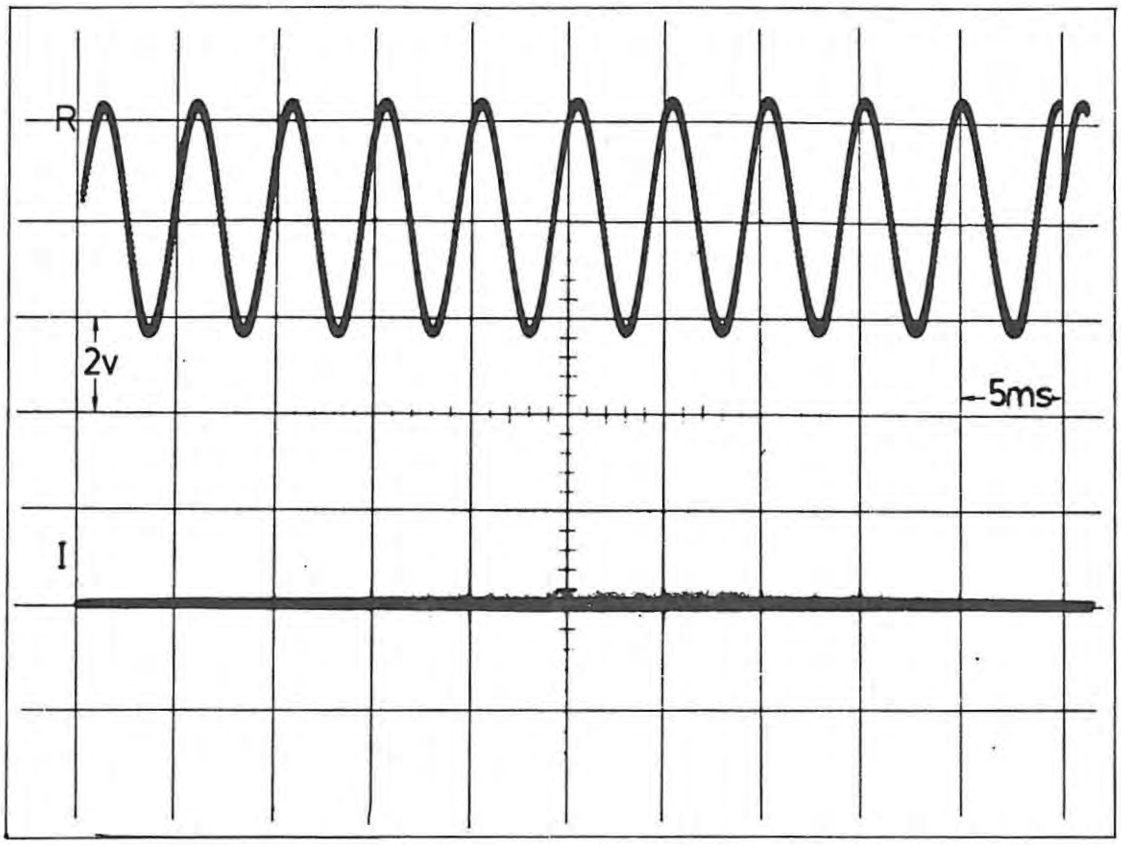


Fig. 7 - 3 Test 1 Memory

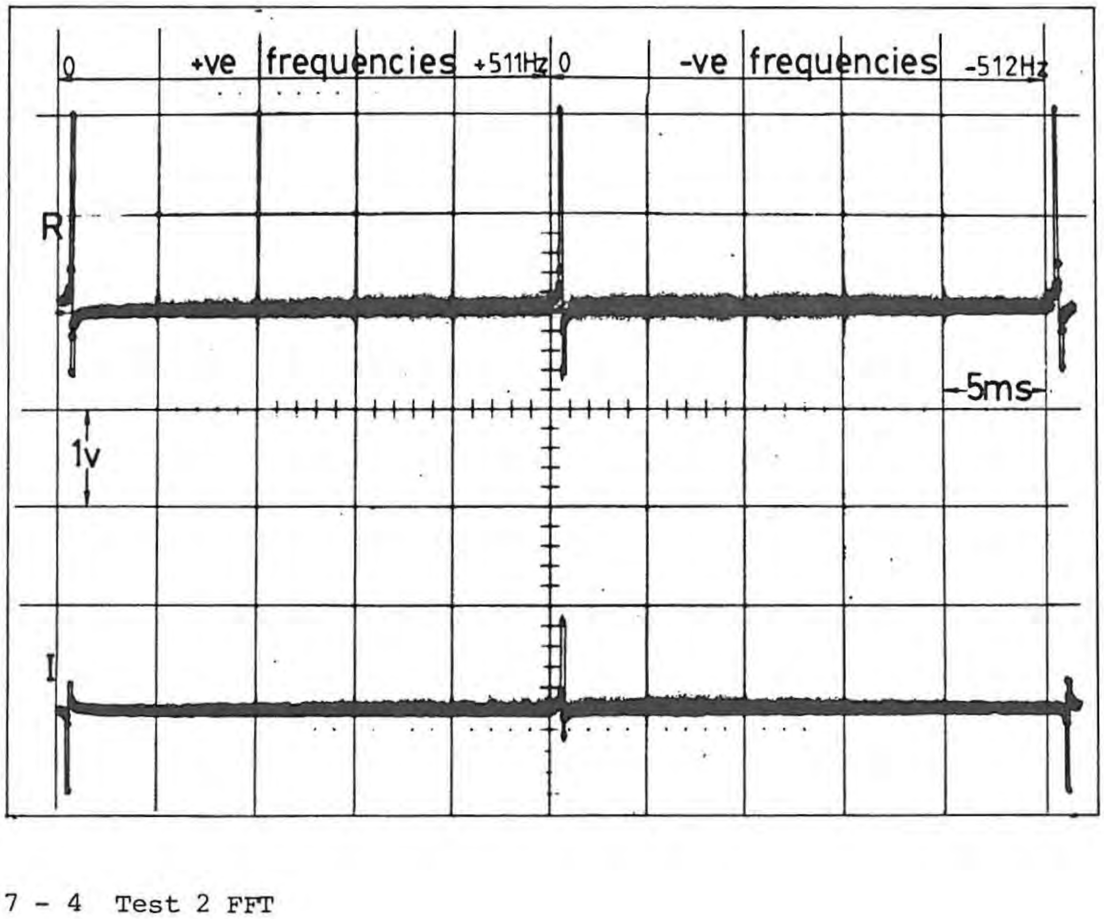


Fig. 7 - 4 Test 2 FFT

It is this odd and even symmetry that the unscramble routine, developed in section 2-4, employs to separate the two transforms of the real and imaginary inputs respectively. Since in this example the imaginary input is shorted to earth, then initiating Test 3 Unscramble will place the transform of the real input in the +ve frequency position (i.e. 0 - 511 memory locations) and will compute zero as the transform for the imaginary input and place this in the -ve frequency position (i.e. 512 - 1023). Fig. 7-5 shows this result.

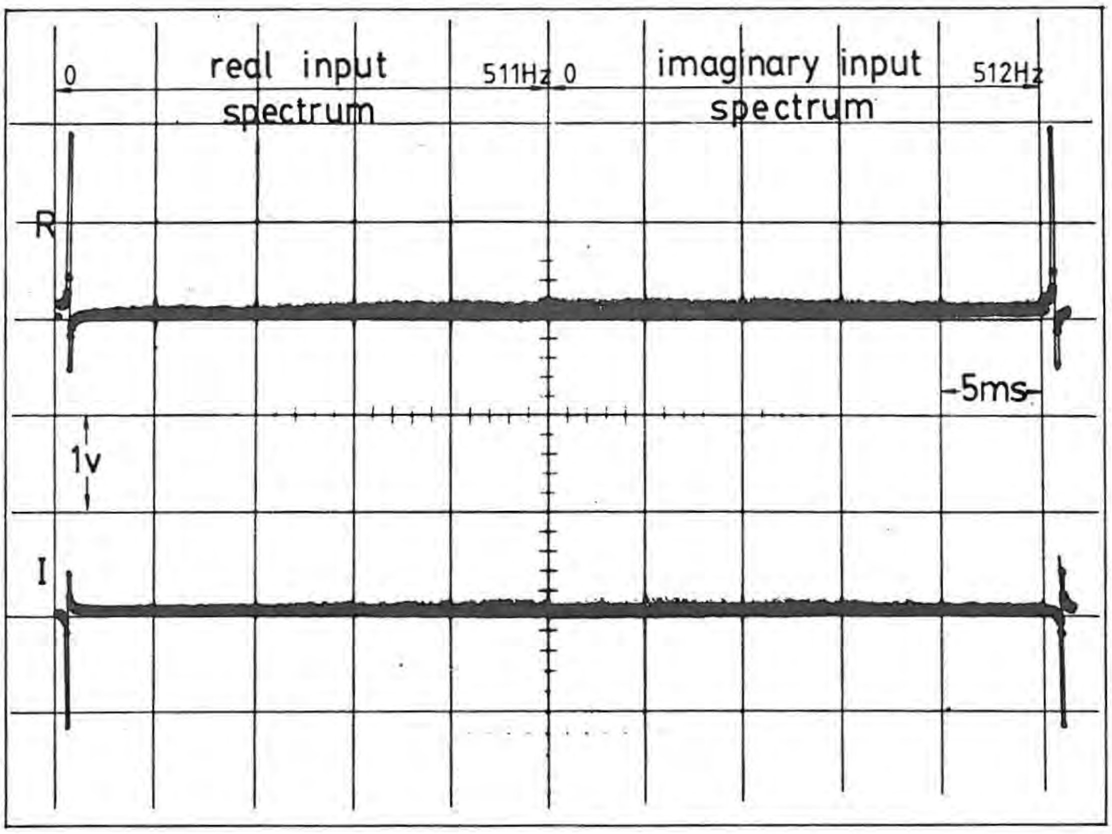


Fig. 7 - 5

Finally, initiating Test 4 Power Spectrum, takes the real input transform and the imaginary input transform of Fig. 7-5 and produces the power spectrum of each, but due to a signal inversion in the D/A converter, the +ve number in the computer becomes represented as a -ve voltage on the display, hence Fig. 7-6 shows the -ve spike in the real input transform power spectrum.

For diagnostic purposes, these tests will verify the correct operation of a particular stage in the computation. By altering the frequency of the input signal, its amplitude or its input port (real or imaginary), the required change in transform can be monitored.

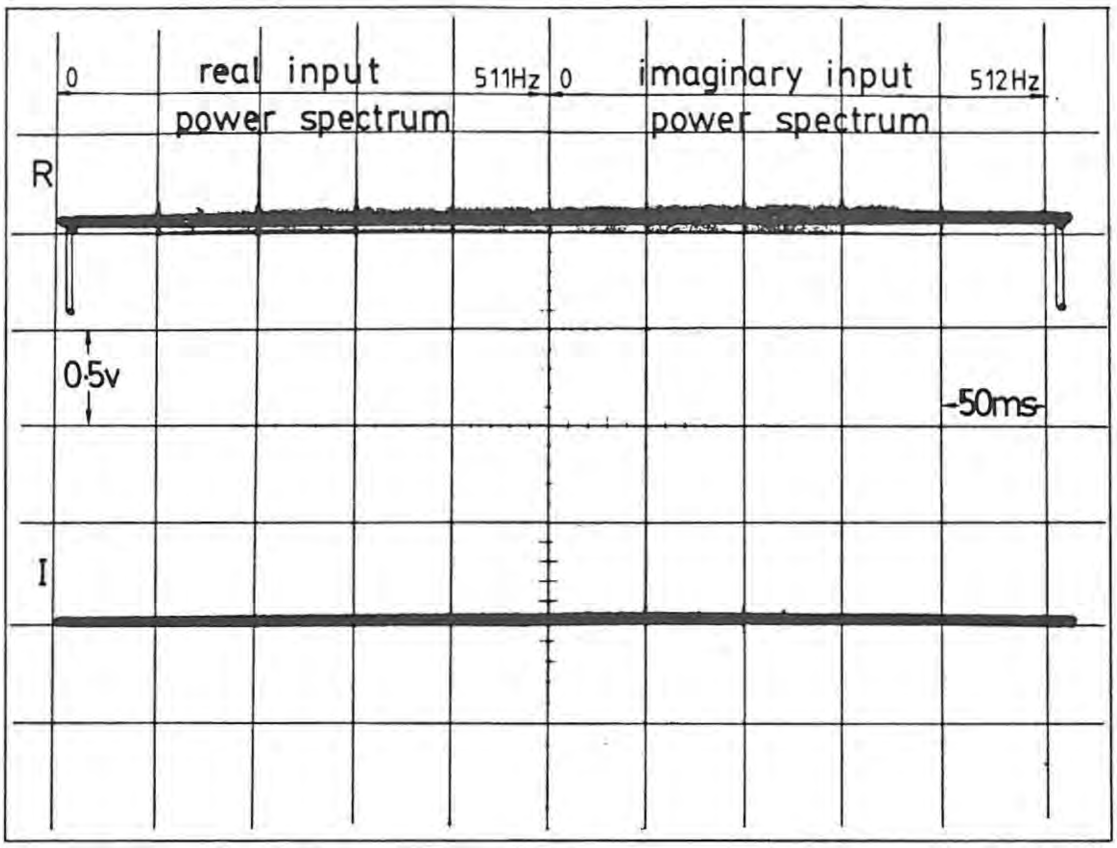


Fig. 7 - 6

7 - 3 Tests with Ionospheric data

This test is intended to show comprehensively the operation of the unscramble routine when handling ionospheric data. The analyser interconnections required are not quite the same as those used in the previous tests but the analyser front panel settings are identical. The output from the ionosonde receiver is now connected to the "real (receiver 1) input" and the signal generator is transferred to the "imaginary (receiver 2) input", whereas the oscilloscope connections remain the same as those depicted in Fig. 7-2.

The very scattered trace of Fig. 7-7 is the sampled time series taken from the output of the Chirpsounder receiver, and displayed using Test 1 Memory. The samples were taken at a point in a sounding where there were two closely spaced echoes corresponding to the ordinary and the extraordinary components of a reflection from the F_2 region of the ionosphere. These two components are two separate echoes, which from the theory of the Chirpsounder ionosonde developed in section 3-1, means that they appear at the output of the ionosonde receiver as two closely spaced tones or frequencies. The beat frequency resulting from the close spacing of the two echoes can be seen as modulation on the trace of Fig. 7-7, but the actual tones cannot be resolved.

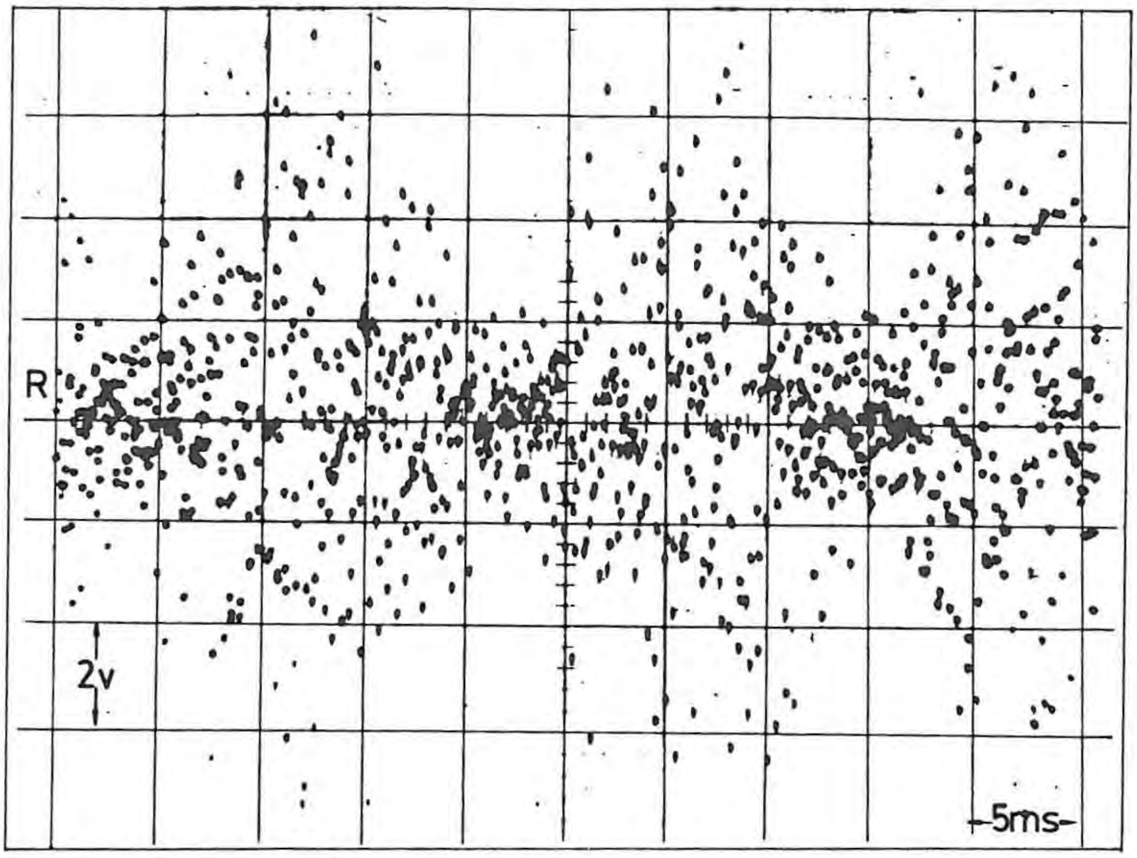


Fig. 7 - 7 Sampled Chirpsounder Receiver Output

To demonstrate the unscramble routine, a test signal of 300Hz 2V p/p was fed into the imaginary input and Fig. 7-8 shows the interesting result of sampling this high frequency signal, and outputting again via Test 1.

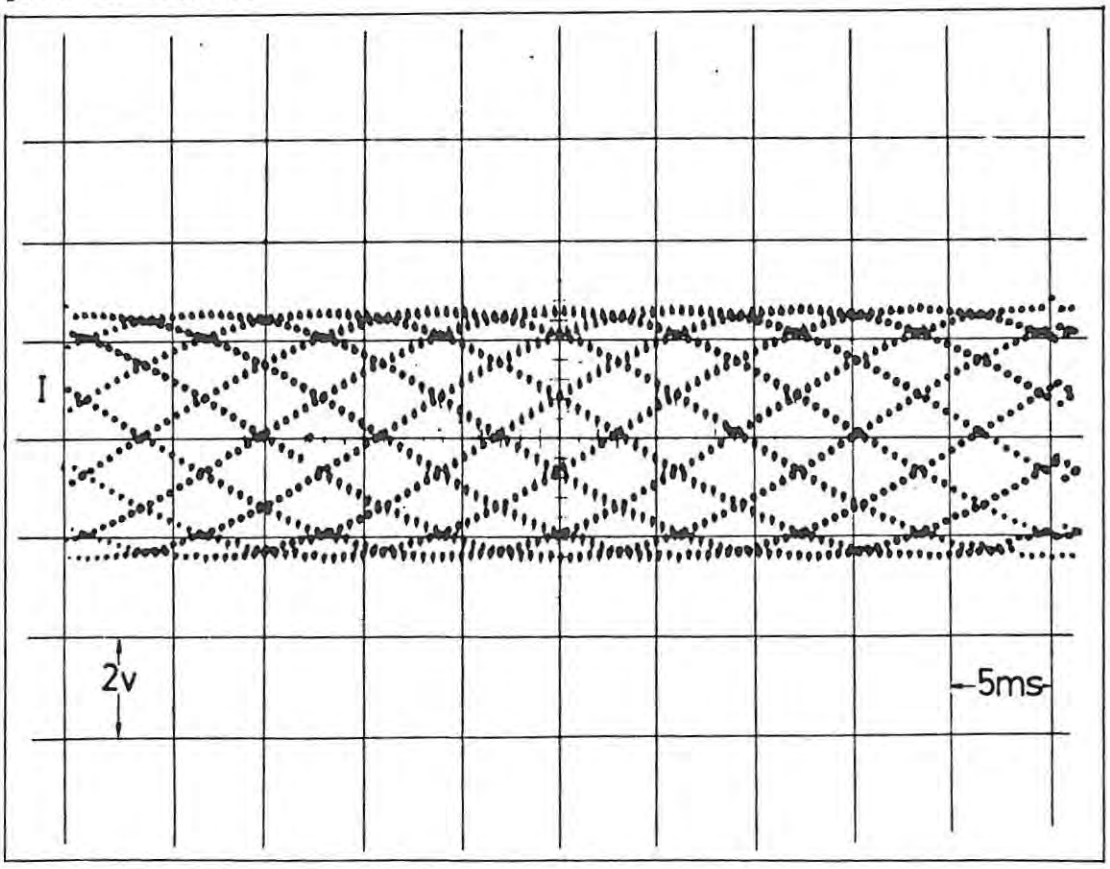


Fig. 7 - 8 300Hz Test Signal

The FFT of this combination of real and imaginary time series is shown in Fig. 7-9 where the two ionospheric echoes can be seen at about 100Hz (which corresponds to a virtual height of 300 kilometers, refer to section 3-1) and the test signal at 300Hz. Notice the odd and even properties of both sets of signals, and that the 300Hz signal produces an even imaginary transform since it was input as a time signal in the imaginary input.

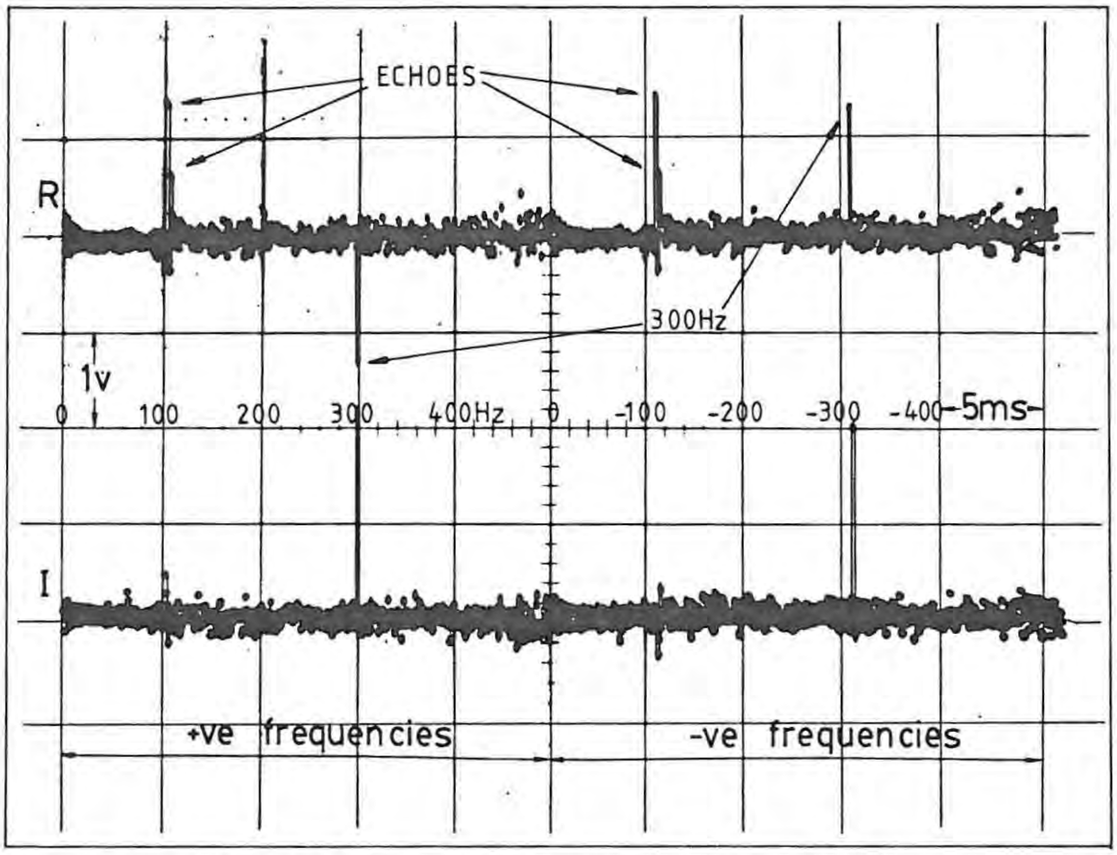


Fig. 7 - 9 Mixed Transform of Ionospheric Data and 300Hz.

The result of the unscramble routine Test 3 depicted in Fig. 7-10, shows very convincingly, the separation of the two transforms. When the FFT analyser is fitted into the Chirpsounder system, it will be handling two time series, separating their transforms, and to indicate to the following computer which are the principal echoes, it will compute the power spectrum of each transform.

The final Fig. 7-11 depicts the operation of Test 4 Power Spectrum, and shows again the two echoes and the 300Hz signal, but the noise in the live data transform has been lost. This is due to the fact that since the output has only 8 bit precision, the true power spectrum of

$R^2 + I^2$ which has a maximum size of 17 bits, cannot be calculated, so it is scaled by factors of 2 until it stays within the range.

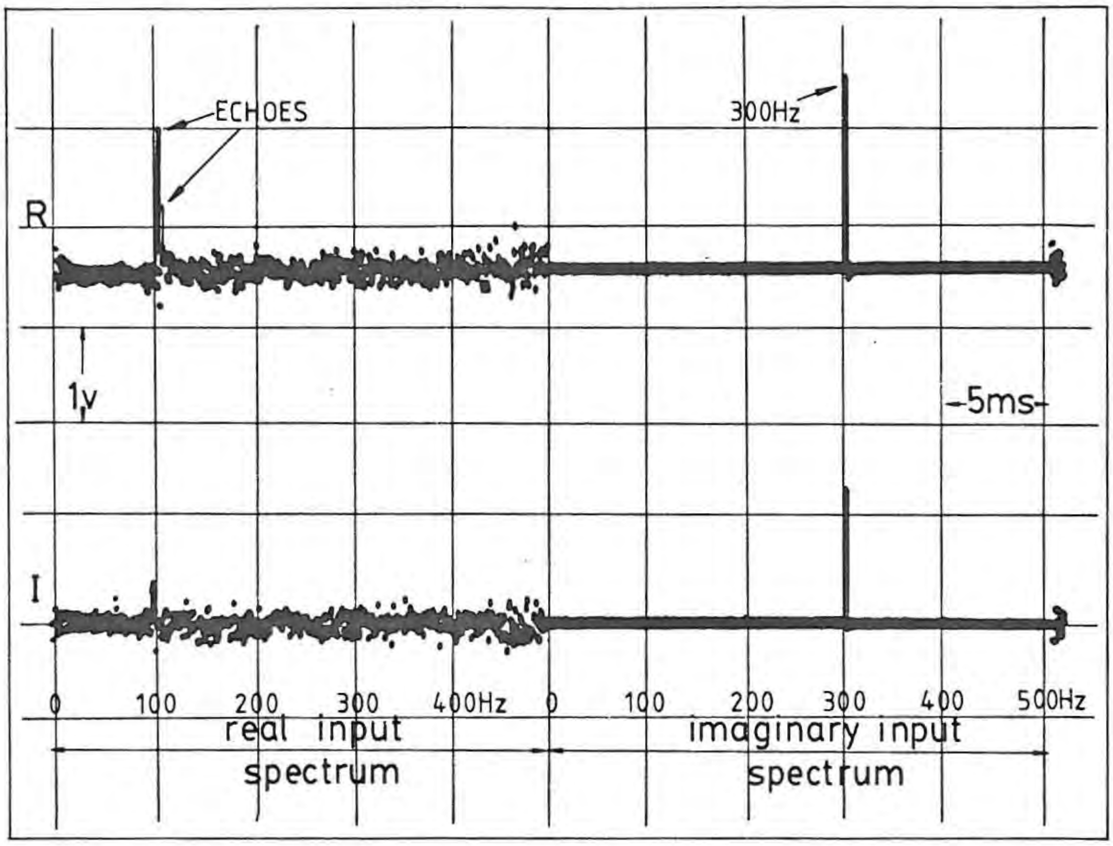


Fig. 7 - 10 Separated Transforms of Ionospheric Data and 300Hz.

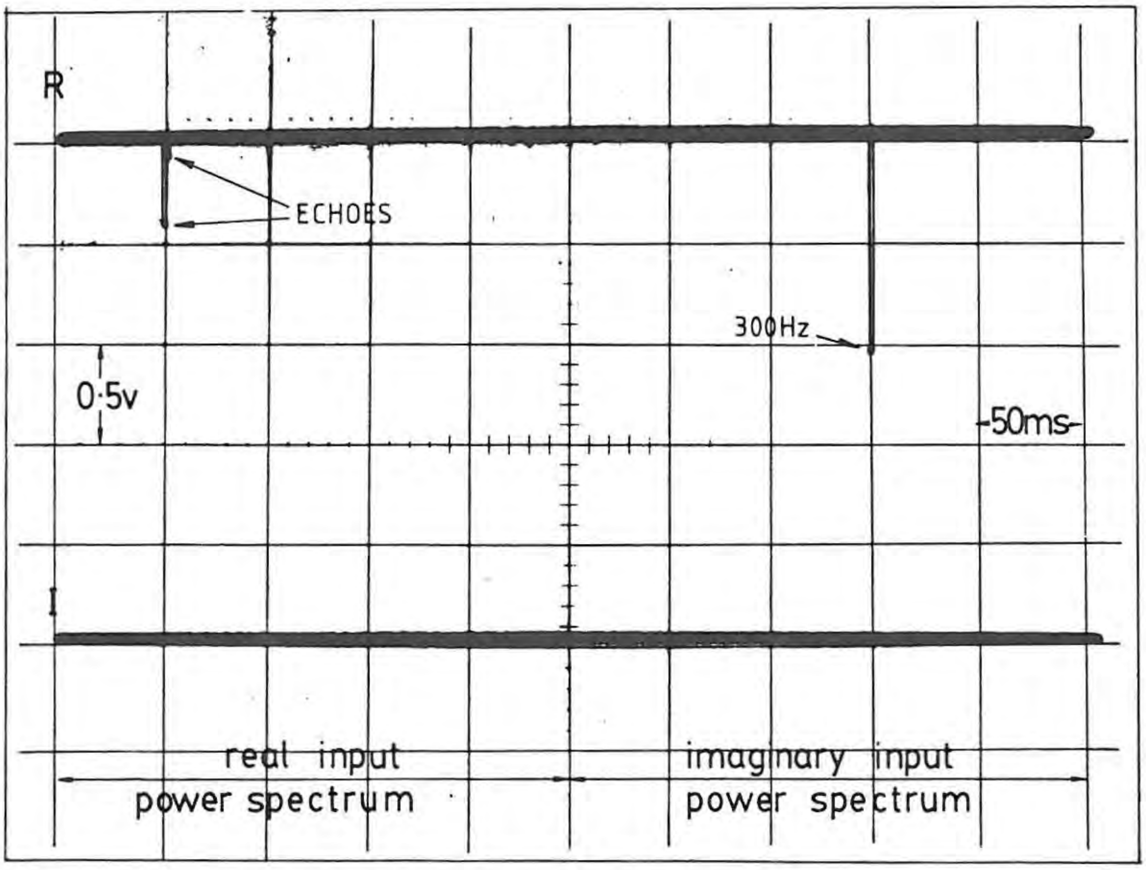


Fig. 7 - 11

All the tests detailed above, and the full FFT, may be single stepped through by switching the system clock to manual, and the address of the next micro-instruction can be seen displayed on the LEDs on the "micro-control" board, refer Fig. 6-1.

In conclusion, these test facilities were found very useful during development and it was felt that due to the complexity of the analyser's operation, and the lack of any visible end result to indicate correct operation, the tests were a necessary part of the analyser. All the oscilloscope traces shown can be easily obtained on any oscilloscope set up according to Fig. 7-2 and to the parameters shown on each trace. Slight changes in time base might be necessary to capture only one cycle of the output (as indicated in all the traces), and external triggering is recommended to be used throughout.

8. CONCLUSIONS

An FFT analyser with the properties detailed in Chapter 1 has been designed, tested and is now incorporated as part of a new system to be sent to the SANAE base in Antarctica during the summer of 1979/1980. The real time analyser performs the FFT in 50 msecs (including the separation of the two receiver transforms) and then outputs the data in 15 msecs. Since it takes 1 sec to accumulate 1024 samples, there are then 935 msecs left for further computation on the data. The Ionospheric data transforms detailed in Chapter 7 show the ability of the analyser to distinguish ionospheric echoes from background noise, and also its ability to separate the two receiver transforms.

Although at present the unit is functioning perfectly within its initial specification, future work has necessitated changes in the analyser's operation. A project to place the whole of the Chirpsounder under microprocessor control, at present being undertaken by G. Evans under the supervision of A.W.V. Poole, has extended the requirements to selectable N-point transforms, and variable analysis bandwidth.

Altering the analysis bandwidth is easily achieved simply by changing the sample rate, but extra work will be necessary to compute a transform for $N < 1024$. Initial discussions on this subject with Poole have indicated a possible approach whereby the sampler only takes the N values required, and the rest of memory is filled with zeros. The full 10 arrays of the 1024 point transform are then computed in the normal way and the output transform data will be found spread evenly within the normal 512 point output spectrum, e.g. for $N = 256$ the spectral points will be spaced 4 apart. The work involved in implementing this, will be to program the analyser, to zero its memory after it has output the last transform spectrum, so that when the memories are interchanged, the sampler has only to store its N values and the rest of memory remains zeros. For the analyser only to take 256 samples, for example, will require the external microprocessor to indicate to the analogue input circuitry, that it has to stop sampling, which in turn will initiate the FFT cycle. The control line to affect this interupt, has been included in anticipation and called "Stop Sampling".

As mentioned in Chapter 2, research into the use of "windows" or truncation functions, has not been carried out in this thesis, since the "top hat" function is regarded as sufficient for our present needs. From a close examination of the different windows and their transforms, as detailed by Harris 1978 (8), it would appear that as the peak-to-side-lobe ratio is increased, so the width of the central lobe increases, which impairs the ability to resolve closely spaced signals. The "tophat" function at present in use has the first side lobes of its transform 15dβ down on the main lobe, Harris (8), and since the output of the analysers is only 8 bits, which represents a dynamic range of 50dβ Glisson et al. (6), to place the side lobes below the resolution of the data would require 35dβ of suppression. The Hamming window depicted by Harris in Fig. 21, page 63, with "its peak-to-side-lobe ratio of 45 db would possibly be sufficient to do this.

LITERATURE CITED

(1) E. Oran Brigham, "The Fast Fourier Transform", Prentice Hall 1974.

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

(3) M.J. Corinthios, "A Fast Fourier Transform for High-Speed Signal Processing", IEEE Transactions on Computers, Vol. C-20, No. 8, pp. 843-846, August 1971.

(4) W.R. Cyre, and G.J. Lipovski, "On Generating Multipliers for a Cellular Fast Fourier Transform Processor", IEEE Transactions on Computers, Vol. C-21, pp. 83-87, January 1972.

(5) E. Dubois and A.N. Venetsanopoulos, "A New Algorithm for the Radix-3 FFT", IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-26, No. 3, pp. 222-225, June 1978.

(6) T.H. Glisson, C.I. Black and A.P. Sage, "The Digital Computation of Discrete Spectra Using the Fast Fourier Transform", IEEE Transactions on Audio and Electroacoustics, Vol. AU-18, No. 3, pp. 271-287, September 1970.

(7) R.W. Hamming, "Numerical Methods for Scientists and Engineers", McGraw-Hill, 1962.

(8) F.J. Harris, "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform", Proceedings of the IEEE, Vol. 66, No. 1, pp. 51-83, January 1978.

(9) T.M. Hedges, "Replacing hardwired logic with microcode", Electronics, Vol. 51, No. 23, pp. 125-129, November 1978.

(10) T. Kaneko and B. Liu, "Accumulation of Round-Off Error in Fast Fourier Transforms", Journal of the Association of Computing Machinary, Vol. 17, No. 4, pp. 637-654, October 1970.

(11) B. Liu and A. Peled, "A New Hardware Realization of Highspeed Fast Fourier Transformers". IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. ASSP-23, No. 6, pp. 543-547, December 1975.

(12) J.R. Mick and J. Springer, "Single chip multiplier expands digital role in signal processing", Electronics, Vol. 49, No. 10, pp. 103-108, May 1976.

(13) J.P.S. Rash, "Oblique Incidence Investigations of the Ionosphere Over the Southern Ocean". PhD thesis, Department of Physics, Rhodes University, Grahamstown, South Africa, December 1978.

(14) P.D. Welch, "A Fixed-Point Fast Fourier Transform Error Analysis", IEEE Transactions on Audio and Electroacoustics, Vol. AU-17, No. 2, pp. 151-157, June 1969.

GLOSSARY

A/D	Analogue to digital
AMD	Advanced Micro Devices
array	One of Log_2 N stages in the FFT where the whole data set is used in computation
bi-directional	When describing a data bus it indicates that data can be written to it or read from it by one device
bus	A parallel group of lines interconnecting devices
butterfly	A term used to describe the Cooley Tukey complex arithmetic of $A \pm BW$
chip	An integrated circuit
D/A	Digital to analogue
DFT	Discrete Fourier transform
DMA	Direct memory access
FFT	Fast Fourier transform
FT	Fourier transform
Latch	D type Flip-flop or bi-stable used as temporary storage
LS	Least Significant
25$\underline{\text{LS}}$14	Low power Schottky TTL
MCU	Microprogram control unit
MS	Most significant
node	A point in an array where the FFT arithmetic is carried out
Op-code	A binary number used to define a particular micro-program sequence
PROM	Programmable read only memory
R/I	Real or Imaginary
ROM	Read only memory
rounding	The operation of losing the fractional part of a number, by adding 0,5 and taking the integer value of the result
SANAE	The name of the South African Antarctic Base, South African National Antarctic Expedition

stack	List of operands or addresses held in memory with a pointer that indicates the next one to be used
Status bit	A signal line that indicates the status of a particular device
Toggle	Change to the opposite binary state
tristate	Type of output stage on TTL circuits, which possesses a high impedence state as well as high or low
truncating	The operation of losing the fractional part of a number or part of a waveform, by just chopping it off
TTL	Transistor Transistor Logic
Unscramble	The separation of two signal transforms from a single transform

```
DATA ORIGIN :8000
DIM A4(512),A5(512),A6(512),A7(512),D$(10),D(128),E(128),I
DIM N1,Z1,Y1,A1,A2,X1,X2,X3,W1,W2,B1,B2,T1,T2,S1,R1
INPUT "N S R D$"N1,S1,R1,D$\Z1=2\Y1=N1\X2=2*PI/1024
OPEN #2,D$
FOR X3=0 TO N1-1
 A4(X3)=INT(S1*512*COS(13*PI*X3/N1)+0.5)+INT(R1*1024*(RND-0.5)+0.5)
 A6(X3)=A4(X3)
 A5(X3)=0
 A7(X3)=0
NEXT X3
 210 Y1=Y1/2\B2=:D000\Z1=2\IFZ1=3 THEN Z1=1
PRINT Y1
IF Y1<1 THEN300
 A1=0\B1=Y1\X1=Y1
 230 A2=PEEK(B2)*256+PEEK(B2+1)
 W1=INT(128*COS(X2*A2)+0.5)
 W2=INT(-128*SIN(X2*A2)+0.5)
 B2=B2+4
 240 GOSUB 500
 A1=A1+1\B1=B1+1
IF A1<> X1 THEN 240
 X1=X1+2*Y1\A1=A1+Y1
IF B1<>N1 THEN B1=B1+Y1\GOTO 230
GOSUB 600\GOTO210
 300 B2=:D000\FORX3=0 TO N1-1
 A2=N1*(PEEK(B2+2*X3)*256+PEEK(B2+2*X3+1))/1024
PRINT #2, A4(A2),A5(A2),A6(A2),A7(A2)
NEXT X3
CLOSE #2\STOP
 500 T1=(A4(B1)*W1-A5(B1)*W2)/128
 T2=(A4(B1)*W2+A5(B1)*W1)/128
 A4(B1)=INT((A4(A1)-T1)/Z1)
 A5(B1)=INT((A5(A1)-T2)/Z1)
 A4(A1)=INT((A4(A1)+T1)/Z1)
 A5(A1)=INT((A5(A1)+T2)/Z1)
 T1=(A6(B1)*W1-A7(B1)*W2)/128
 T2=(A6(B1)*W2+A7(B1)*W1)/128
 A6(B1)=(A6(A1)-T1)/Z1
 A7(B1)=(A7(A1)-T2)/Z1
 A6(A1)=(A6(A1)+T1)/Z1
 A7(A1)=(A7(A1)+T2)/Z1
RETURN
 600 FORX3=0 TO 127
 D(X3)=0\E(X3)=0
NEXT X3
FOR X3=0 TO N1-1
 T1=A6(X3)-A4(X3)
 T2=A7(X3)-A5(X3)
 I=INT(T1*8+0.5)
 D(I+63)=D(I+63)+1
 I=INT(T2*8+0.5)
 E(I+63)=E(I+63)+1
NEXT X3
FOR X3=0 TO 127
PRINT #2, D(X3);
NEXT X3
PRINT #2
FOR X3=0 TO 127
PRINT #2, E(X3);
NEXT X3
PRINT #2
RETURN
END
```

FFTHIST

```
DATA ORIGIN :8000
DIMA4(512),A5(512),A6(512),A7(512),D$(10),D(128),E(128),I
DIMN1,Z1,Y1,A1,A2,X1,X2,X3,W1,W2,B1,B2,T1,T2,S1,R1,F,W3,W4,W5,W6
INPUT"N S F R W5 W6 D$"N1,S1,F,R1,W5,W6,D$\Z1=2\Y1=N1\X2=2*PI/1024
OPEN#2,D$
FORX3=0 TO N1-1
A4(X3)=INT(S1*512*COS(F*PI*X3/N1)+0.5)+INT(R1*1024*(RND-0.5)+0.5)
A6(X3)=A4(X3)
A5(X3)=0
A7(X3)=A5(X3)
NEXT X3
210  Y1=Y1/2\B2=:D000\Z1=2\IF Z1=3 THEN Z1=1
PRINT Y1
IFY1<1 THEN 300
A1=0\B1=Y1\X1=Y1
230 A2=PEEK(B2)*256+PEEK(B2+1)
W1=INT(W5*COS(X2*A2)+0.5)\W3=128*COS(X2*A2)+0.5
W2=INT(-W5*SIN(X2*A2)+0.5)\W4=-128*SIN(X2*A2)+0.5
B2=B2+4
240 GOSUB 500
A1=A1+1\B1=B1+1
IFA1<> X1 THEN 240
X1=X1+2*Y1\A1=A1+Y1
IFB1<>N1 THEN B1=B1+Y1\GOTO 230
GOTO 210
300 GOSUB 600\B2=:D000 \FOR X3=0 TO N1-1
A2=N1*(PEEK(B2+2*X3)*256+PEEK(B2+2*X3+1))/1024
PRINT#2, A4(A2),A5(A2),A6(A2),A7(A2)
NEXT X3
CLOSE#2\STOP
500 T1=(A4(B1)*W1-A5(B1)*W2)/W6
T2=(A4(B1)*W2+A5(B1)*W1)/W6
A4(B1)=(A4(A1)-T1)/Z1
A5(B1)=(A5(A1)-T2)/Z1
A4(A1)=(A4(A1)+T1)/Z1
A5(A1)=(A5(A1)+T2)/Z1
T1=(A6(B1)*W3-A7(B1)*W4)/128
T2=(A6(B1)*W4+A7(B1)*W3)/128
A6(B1)=(A6(A1)-T1)/Z1
A7(B1)=(A7(A1)-T2)/Z1
A6(A1)=(A6(A1)+T1)/Z1
A7(A1)=(A7(A1)+T2)/Z1
RETURN
600 FORX3=0TO127
D(X3)=0\E(X3)=0
NEXTX3
FORX3=0 TO N1-1
T1=A6(X3)-A4(X3)
T2=A7(X3)-A5(X3)
I=INT(T1*8+0.5)
D(I+63)=D(I+63)+1
I=INT(T2*8+0.5)
E(I+63)=E(I+63)+1
NEXTX3
FORX3=0TO127
PRINT#2, D(X3);
NEXTX3
PRINT#2
FORX3=0TO127
PRINT#2, E(X3);
NEXTX3
PRINT#2
RETURN
END
```

FFTHIST2

```
 1.00=DIMX,A2,A3,A4
 2.00=OPEN #2,"REVSEQ"
 3.00=FORX=0TO1023
 4.00=READ #2,A2
 5.00=A3=INT(A2/256)
 6.00=A4=A2-A3*256
 7.00=POKE :D000+2*X,A3
 8.00=POKE :D000+2*X+1,A4
 9.00=NEXT X
10.00=CLOSE #2
11.00=STOP
12.00=END
```

BRPOKE

```
0010  N1=1024:OPEN#20,"REVSEQ1"FORINPUT:FIELD#20,A2=4
0011 OPEN #30,"WPDATA"FOROUTPUT:FIELD#30,W4=4,W5=4
0012  X2=2*3.141592654/N1
0020 FOR W2=1TON1/2
0025 GET #20
0030  W4=INT(64*COS(X2*A2)+0.5)
0032  W5=INT(-64*SIN(X2*A2)+0.5)
0035 IF W4=0THEN42
0037 IF W4/ABS(W4)=-1THENW4=256+W4
0042 IF W5=0THEN45
0044 IF W5/ABS(W5)=-1THENW5=256+W5
0045 PUT #30
0046 ? W2
0060 GET #20:NEXTW2
0070 CLOSE #20,#30:STOP
```

WPDATA

```
0010 OPEN #20,"WPDATA"FORINPUT:FIELD#20,W4=4,W5=4
0015 SET #20=257
0020 FOR W2=0TO255
0030 GET #20:A=49152+W2
0040 POKE( A,W4):?W4;
0050 NEXT W2
0060 CLOSE #20:STOP
```

WPPOKE

```
0005  N1=1024:F=256
0010 DIM A4(255,4),A5(255,4),A6(255,4),A7(255,4)
0020 ? "D$,P";:INPUTD$,P
0040 OPEN #30,D$FOR INPUT :FIELD#30,A4=4,A5=4
0045  Z4=0:V4=0
0050 FOR X4=0 TO (-1+LOFF#30)
0055 IF (X4-Z4)=256 THEN Z4=Z4+256:V4=V4+1
0060 GET #30:A4(X4-Z4,V4)=A4:A5(X4-Z4,V4)=A5:NEXTX4
0070 CLOSE #30
0200  Y1=N1:OPEN#20,"WP1024"FOR INPUT:FIELD#20,W4=3,W5=3:Z1=1
0210  Y1=Y1/2:SET#20=1:?Y1:Z1=Z1+1:IFZ1=3 THENZ1=1
0220 IF Y1<1 THEN300
0230  A1=0:B1=Y1:X1=Y1:GET#20
0240  A=A1:B=B1
0241  V=0:IFA>=F THENA=A-F:V=1:IFA>=F THENA=A-F:V=2:IFA>=FTHENA=A-F:V=3
0242  U=0:IFB>=FTHENB=B-F:U=1:IFB>=FTHENB=B-F:U=2:IFB>=FTHENB=B-F:U=3
0245 GOSUB 500
0250  A1=A1+1:B1=B1+1
0260 IF A1<>X1THEN240
0270  X1=X1+2*Y1:A1=A1+Y1
0280 IF B1<>N1THENB1=B1+Y1:GET#20:GET#20:GOTO240
0290 GOTO 210
0295 CLOSE #20
0300 OPEN #20"REVSEQ1"FORINPUT:FIELD#20,A2=4
0310 FOR W2=1TON1:W=W2
0312  T=0:IFW>=FTHENW=W-F:T=1:IFW>=FTHENW=W-F:T=2:IFW>=FTHENW=W-F:T=3
0315 GET #20:Z=A2
0317  S=0:IFZ>=FTHENZ=Z-F:S=1:IFZ>=FTHENZ=Z-F:S=2:IFZ>=FTHENZ=Z-F:S=S
0320  A6(W,T)=A4(Z,S):A7(W,T)=A5(Z,S)
0330 NEXT W2
0340 CLOSE #20
0350 FOR W2=2 TO(N1/2)
0352  W=W2:Z=1026-W2:T=0:S=0
0355 IF W>=FTHENW=W-F:T=1:IFW>=FTHENW=W-F:T=2:IFW>=FTHENW=W-F:T=3
0357 IF Z>=FTHENZ=Z-F:S=1:IFZ>=FTHENZ=Z-F:S=2:IFZ>=FTHENZ=Z-F:S=3
0360  X7=-(A6(W,T)-A6(Z,S))/2:A6(W,T)=(A6(W,T)+A6(W,T)=A6(Z,S))/2
0370  A6(Z,S)=(A7(W,T)+A7(Z,S))/2:A7(W,T)=(A7(W,T)-A7(Z,S))/2
0380  A7(Z,S)=X7:NEXTW2
0390 GOTO 600
0500  R1=A4(B,U)*W4-A5(B,U)*W5
0510  I1=A5(B,U)*W4+A4(B,U)*W5
0520  A4(B,U)=INT((64*A4(A,V)-R1)/(64*Z1))
0530  A5(B,U)=INT((64*A5(A,V)-I1)/(64*Z1))
0540  A4(A,V)=INT((64*A4(A,V)+R1)/(64*Z1))
0550  A5(A,V)=INT((64*A5(A,V)+I1)/(64*Z1))
0580 RETURN
0600 FOR W2=1TON1
0602  W=W2:T=0
0605 IF W>=FTHENW=W-F:T=1:IFW>=FTHENW=W-F:T=2:IFW>=FTHENW=W-F:T=3
0610 ? #P,INT(A6(W,T)+0.5);:NEXTW2
0620 ? #P,
0630 FOR W2=1TON1
0632  W=W2:T=0
0635 IF W>=FTHENW=W-F:T=1:IFW>=FTHENW=W-F:T=2:IFW>=FTHENW=W-F:T=3
0640 ? #P,INT(A7(W,T)+0.5);:NEXTW2
0650 ? #P,:STOP
```

FFT16

REMARKS	PROM address Hex	$\boxed{U3}$					$\boxed{U4}$					$\boxed{U5}$					U14					U15					U17					U18					$\boxed{U19}$					$\boxed{U20}$					$\boxed{U21}$					STATUS BIT	opcode	$\boxed{\ }$ = LATCHED
			9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12			
			$\overline{VAL}$	$\overline{INT}$	$\overline{RET}$	$\overline{WE1}$		OC4	MUXINP/CLK	OC3	OC2		SBC	OC1	SBB	SBA		$\overline{SCOPE\ TRIG}$	$\overline{TCLR}$	$\overline{R1}$	$\overline{SR1}$		$\overline{CLT}$	$\overline{CKR1}$	$\overline{LD2}$	$\overline{CKWP}$		B R/$\overline{S}$	$\overline{CK1}$	$\overline{CKM}$	$\overline{CKDA}$		$\overline{CK4}$	$\overline{CK3}$	$\overline{BCLK}$	$\overline{CK2}$		A+B	$\overline{en(N-n)}$	$\overline{en\ n}$	$\overline{en(A+B)}$		$\overline{en1}$	$\overline{en4}$	$\overline{WE2}$	$\overline{LD3}$		+/$\overline{CLR}$	$\overline{en3}$	$\overline{LD1}$	$\overline{en2}$			
UNPROGRAMMED STATE			1	1	1	1		0	0	0	0		0	0	0	0		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1	0		
ENTER BY ZERO	00																																																				PC	
	01																										7	0																									PC	
	02						1				1	4		1																																							PCSTK	
	03						1				1	1				1																																				1	JS(TPC)	
	04											4		1																																							JD	
FIND TEST ROUTINE Reset	08						1				1	4		1			C			0	0																										7	0					PCSTK	Save return
	09						2			1		8	1																			D			0																	4	PC(TORO)	Skip if correct
Clock Test address counter	0A						5		1		1																					D			0																		JS	
	0B											4		1																																							JD	Go to routine
INITIALISE F.F.T	0F																																																				PC	
Reset	10						1				1	4		1			C			0	0											D			0																		PCSTK	
	11						1				1	1				1																																				1	JS(TPC)	
	12											4		1																																							JD	
TEST 1 MEMORY	16																																																				PC	
Change Memory	17						1				1	4		1													D			0																							PCSTK	Save return
en (A+B) A	18																																				6	0			0												PC	
"	19																																				6	0			0												PC	
" Output Data	1A	D			0																																6	0			0												PC	
" Latch at D/A	1B																										E				0																						PC	
Clock address	1C						1				1	2			1																	7	0																			2	JS(TPC)	
	1D						1				1	4		1																																							PCSTK	Save return
en (A + B) B	1E																D			0																	E				0												PC	
"	1F																																				E				0												PC	
" Output Data	20	D			0																																E				0												PC	
" Latch at D/A	21																										E				0																						PC	
Clock address	22						1				1	2			1																	7	0																			2	JS(TPC)	
	23						2			1		4		1																																							PCPOP	
Scope trigger Reset	24						1				1						5	0		0																																	JS	

☐ LATCHED

REMARKS	PROM address Hex	U3 (9 10 11 12: VAL, INT, RET, WET)	U4 (9 10 11 12: OC4, MUXINP/CLK, OC3, OC2)	U5 (9 10 11 12: SBC, OC1, SBB, SBA)	U14 (9 10 11 12: SCOPE TRIG, TCLR, RT, SRT)	U15 (9 10 11 12: CLT, CKRI, LD2, CKW^P)	U17 (9 10 11 12: B R/S, CK1, CKM, CKDA)	U18 (9 10 11 12: CK4, CK3, BCLK, CK2)	U19 (9 10 11 12: A+B, en(N-n), enn, en(A+B))	U20 (9 10 11 12: en1, en4, WE2, LD3)	U21 (9 10 11 12: ±/CLR, en3, LD1, en2)	STATUS BIT	opcode	
UNPROGRAMMED STATE		1 1 1 1	0 0 0 0	0 0 0 0	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	0		
F.F.T	28												PC	
	29		1 1	4 1			D O	D O					PCSTK	Save return)
en(A+B), B, enW^P clear arith.	2A					7 O			E O	B O			PC	
" Latch W^P at multiplier.	2B					D O			E O				PC	
" Output Data en ll^e load	2C	D O							E O		D O		PC	
" Latch B CK W^P	2D					E O	B O		E O				PC	
" Clock mult., adders A	2E				B O		B O	B O	6 0 O				PC	
8 " " "	2F						B O	B O	6 0 O				PC	
" " Output Data en ll^e load	30	D O					B O	B O	6 0 O	E O			PC	
" " Latch A	31		1 1	4 1			B O	A O O					PCSTK	
"	32		1 1	B 1 1 1			B O	B O				7	JS(TPC)	4 X
Clock A mult., adders, output	33		1 1	4 1			B O	A O O					PCSTK	
"	34		1 1	B 1 1 1			B O	A O O				7	JS(TPC)	7 X
"	35		2 1	4 1			B O	A O O					PCPOP	
"	36		2 1	4 1			B O	A O O					PCPOP	
"	37		2 1	3 1 1			B O	A O O				3	PC(TORO)	SKIP Next if odd array
"	38						B O	A O O					PC	
" en (A+B) A	39						B O	A O O	6 0 O				PC	
" " "	3A						B O	A O O	6 0 O				PC	
" " Output A*	3B								6 0 O	D O	E O		PC	
" " Write	3C	E O							6 0 O	D O	E O		PC	
en (A+B) B	3D								E O				PC	
" "	3E								E O				PC	
" " Output B*	3F								E O	5 O O			PC	
" " Write	40	E O							E O	5 O O			PC	
	41												PC	
Clock address	42		1 1	9 1 1				7 O				5	JS(TPC)	
	43			4 1									JD	
BIT REVERSED OUTPUT	47	B O											PC	← START FOR F.F.T.
CKRI	48	B O				B O							PC	2ND PASS IMAGINARY
	49	B O											PC	← START FOR F.F.T
BCLK	4A	B O						D O					PC	1ST PASS REAL
	4B	B O		E 1 1 1								6	JD(TPC)	← START FOR TEST
Save return) Scope trigger	4C	B O	1 1	4 1	5 O O								PCSTK	
enn	4D	B O							D O				PC	
"	4E	B O							D O				PC	
" Output Data	4F	9 O O							D O				PC	
" Latch at D/A	50	3 O O					E O						PC	
Clock address	51	B O	1 1	2 1				7 O				2	JS(TPC)	
	52	B O											PC	

[ ] = LATCHED

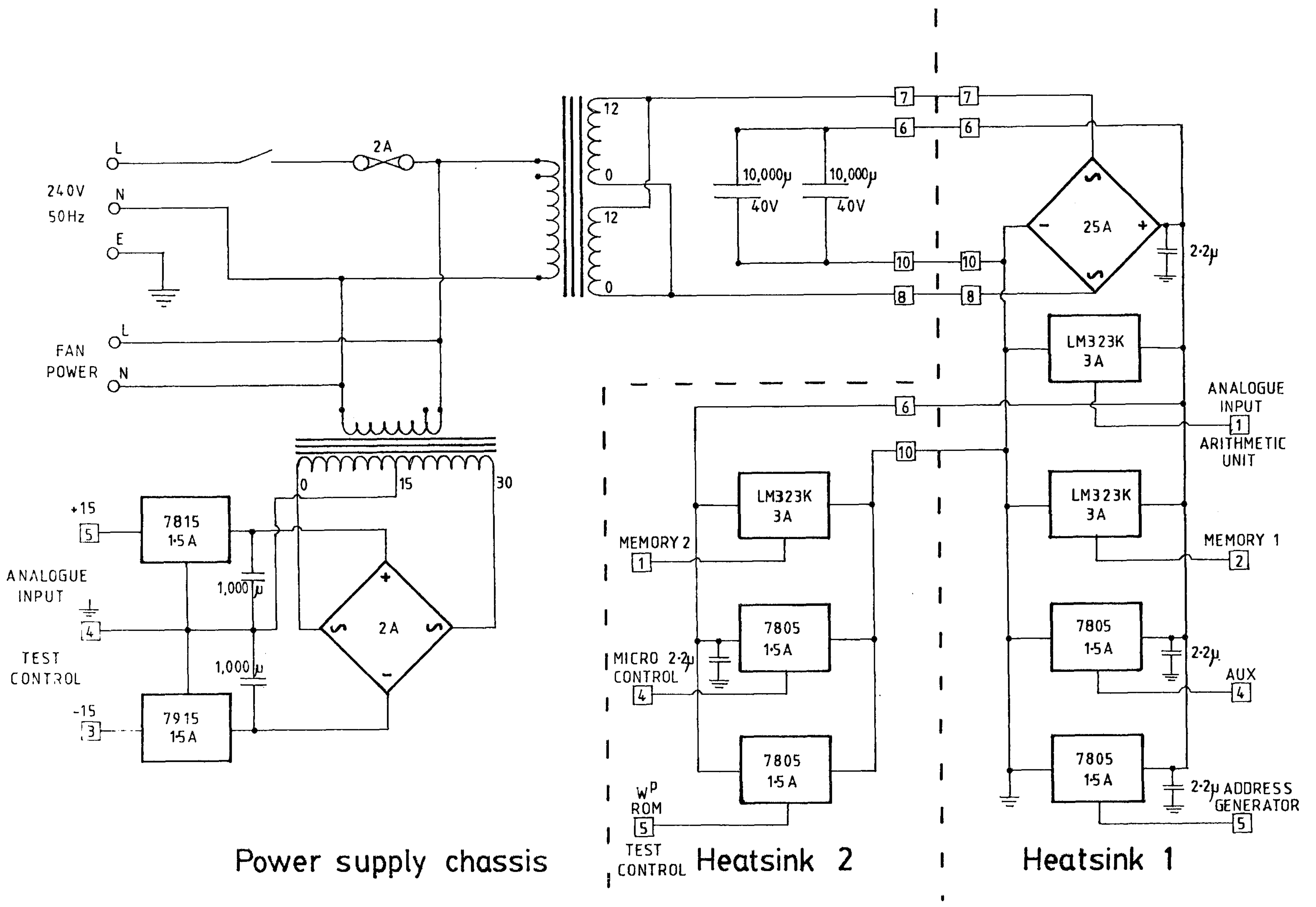
REMARKS	PROM address Hex	U3					U4					U5					U14					U15					U17					U18					U19					U20					U21					STATUS BIT	opcode	
			9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12			
			$\overline{VAL}$	$\overline{INT}$	$\overline{RET}$	$\overline{WET}$		OC4	MUXINP/CLK	OC3	OC2		SBC	OC1	SBB	SBA		SCOPE TRIG	$\overline{TCLR}$	$\overline{RT}$	$\overline{SRT}$		$\overline{CLT}$	$\overline{CKRT}$	$\overline{LD2}$	$\overline{CKW}^{P}$		$\overline{B}$ R/S	$\overline{CK1}$	$\overline{CKM}$	$\overline{CKDA}$		$\overline{CK4}$	$\overline{CK3}$	$\overline{BCLK}$	$\overline{CK2}$		A+B	$\overline{en(N-n)}$	$\overline{enn}$	$\overline{en(A+B)}$		$\overline{en1}$	$\overline{en4}$	$\overline{WE2}$	$\overline{LD3}$		±/$\overline{CLR}$	$\overline{en3}$	$\overline{LD1}$	$\overline{en2}$			
UNPROGRAMMED STATE			1	1	1	1		0	0	0	0		0	0	0	0		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1	0		
Save return) ↙	53	B		O			1				1	4		1			D			O																																		PCSTK	
en (N-n)	54	B		O																																		B		O														PC	
"	55	B		O																																		B		O														PC	
" Output Data	56	9		O	O																																	B		O														PC	
" Latch at D/A	57	3	O	O																							E				O																							PC	
Clock address	58	B		O			1				1	2			1																	7	O																			2	JS(TPC)	
	59	B		O								4		1																																							JD	
POWER SPECTRUM	5E	B		O																																																		PC	← START F.F.T
	5F	B		O																												D			O																			PC	
	60	B		O								E	1	1	1																																					6	JD(TPC)	← START TEST
Scope trigger Reset	61	B		O			1				1	4		1			5	O		O																																	PCSTK	SAVE RETURN) ↙
enn Clear Arithmetic	62	B		O																		7	O														D			O							7	O					PC	
"	63	B		O																																	D			O							7	O					PC	
" Output Data en ll^{e} load	64	9		O	O																																D			O							5	O		O			PC	
" Latch at Band Mult.	65	B		O			1				1	4		1			B		O			D			O		B		O																		7	O					PCSTK	7X
8{Clock mult. adders	66	B		O			1				1	B	1		1	1											B		O			B		O													7	O				7	JS(TPC)	
" " "	67	B		O			1				1	4		1													B		O			B		O													7	O					PCSTK	
15{" " " and output	68	B		O			1				1	B	1		1	1											B		O			A		O		O											7	O				7	JS(TPC)	7X
" " " "	69	B		O			2			1		4		1													B		O			A		O		O											7	O					PCPOP	
" " " "	6A	B		O			1				1	4		1													B		O			A		O		O											7	O					PCSTK	
" " " "	6B	B		O			1				1	B	1		1	1											B		O			A		O		O											7	O				7	JS(TPC)	6X
Clock address	6C	B		O			2			1		4		1																		7	O														7	O					PCPOP	
Output A *	6D	B		O			2			1		4		1																																	6	O			O		Pc PoP	
Latch at D/A	6E	3	O	O			1				1	2			1												E				O																7	O				2	JS(TPC)	
	6F	B		O																																											7	O					Pc	
Reset	70	B		O			1				1	4		1			D			O																											7	O					PCSTK	SAVE RETURN) ↙
en(N-n) Clear Arithmetic	71	B		O																		7	O														B		O								7	O					PC	
"	72	B		O																																	B		O								7	O					PC	
" Output Data en ll^{e} load.	73	9		O	O																																B		O								5	O		O			PC	
" Latch at Band Mult.	74	B		O			1				1	4		1			B		O			D			O		B		O																		7	O					PCSTK	
8{Clock mult. adders	75	B		O			1				1	B	1		1	1											B		O			B		O													7	O				7	JS(TPC)	7X
" " "	76	B		O			1				1	4		1													B		O			B		O													7	O					PCSTK	
15{" " " output	77	B		O			1				1	B	1		1	1											B		O			A		O		O											7	O				7	JS(TPC)	7X
" " " "	78	B		O			2			1		4		1													B		O			A		O		O											7	O					PCPOP	
" " " "	79	B		O			1				1	4		1													B		O			A		O		O											7	O					PCSTK	
" " " "	7A	B		O			1				1	B	1		1	1											B		O			A		O		O											7	O				7	JS(TPC)	6X
Clock address	7B	B		O			2			1		4		1																		7	O														7	O					PCPOP	
Output A *	7C	B		O			2			1		4		1																																	6	O			O		PCPOP	
Latch at D/A	7D	3	O	O			1				1	2			1												E				O																7	O				2	JS(TPC)	
	7E	B		O								4		1																																							JD	

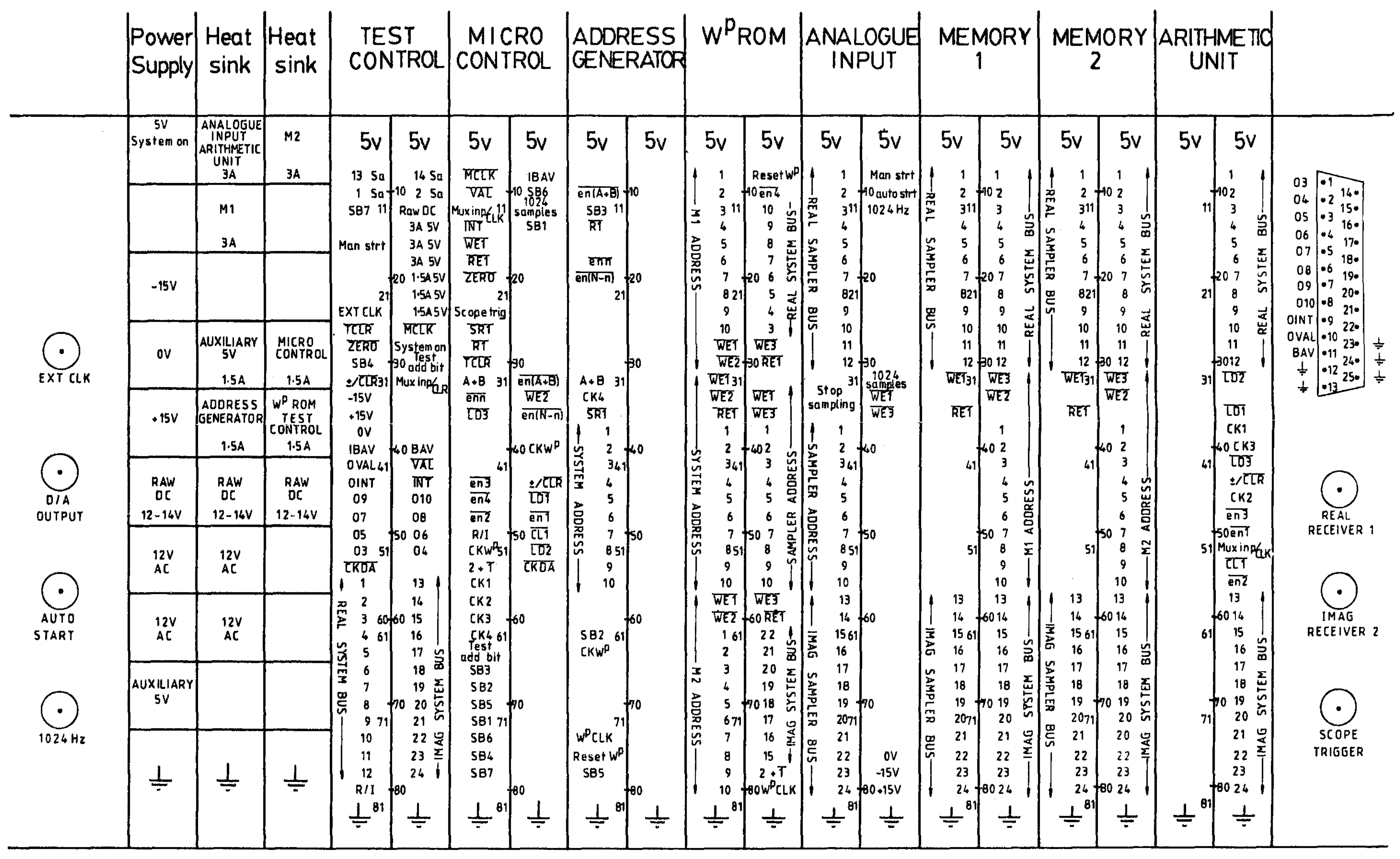
☐ ≡ LATCHED

REMARKS	PROM address Hex	U3					U4					U5					U14					U15					U17					U18					U19					U20					U21						opcode	
			9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12		9	10	11	12			
			VAL	INT	RET	WET		OC4	MUXINP/CLK	OC3	OC2		SBC	OC1	SBB	SBA		SCOPE TRIG	TCLR	RT	SRT		CLT	CKR1	LD2	CKWP		B R/S	CK1	CKM	CKDA		CK4	CK3	BCLK	CK2		A + B	en(N-n)	en n	en(A+B)		en1	en4	WE2	LD3		±/CLR	en3	LD1	en2	STATUS BIT		
UNPROGRAMMED STATE			1	1	1	1		0	0	0	0		0	0	0	0		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1		1	1	1	1	0		
UNSCRAMBLE	82																																																					PC	
Reset	83						1				1	4		1			D			0												D			0																			PCSTK	Save return
enn Clear Arithmetic	84						4		1													7	0														D			0														PC	
"	85						4		1																												D			0														PC	
" Output R(n) en ll^{e} load	86	D			0		4		1																												D			0		E				0								PC	
Latch R(n) at A	87						4		1																							E				0																		PC	
en (N-n)	88						4		1																												B		0															PC	
"	89						4		1																												B		0															PC	
" Output R(N-n)	8A	D			0		4		1																												B		0								D			0			PC	
Latch R(N-n) at B	8B						5		1		1	4		1			B		0								B		0																								PCSTK	
15{ Clock A, B adder + output	8C						5		1		1	B	1		1	1											B		0			A		0		0																7	JS(TPC)	7X
	8D						6		1	1		4		1																																							PCPOP	
	8E						5		1		1	4		1													B		0			A		0		0																	PCSTK	
" " " en(N-n)	8F						5		1		1	B	1		1	1											B		0			A		0		0	B		0													7	JS(TPC)	6X
Output B *	90						2			1		4		1																							B		0			5	0		0								PCPOP	
Write	91	E				0																															B		0			5	0		0								PC	
enn	92																																				D			0													PC	
"	93																																				D			0													PC	
Output A *	94																																				D			0		D			0		E				0		PC	
Write	95	E				0																															D			0		D			0		E				0		PC	
	96																																																				PC	
Clock address	97						1				1	2			1																	7	0																			2	JS(TPC)	
	98	B		0			1				1	4		1																																							PCSTK	
	99	B		0			1				1	A	1		1																																					6	JS(TPC)	
	9A	B		0								4		1																																							JD	

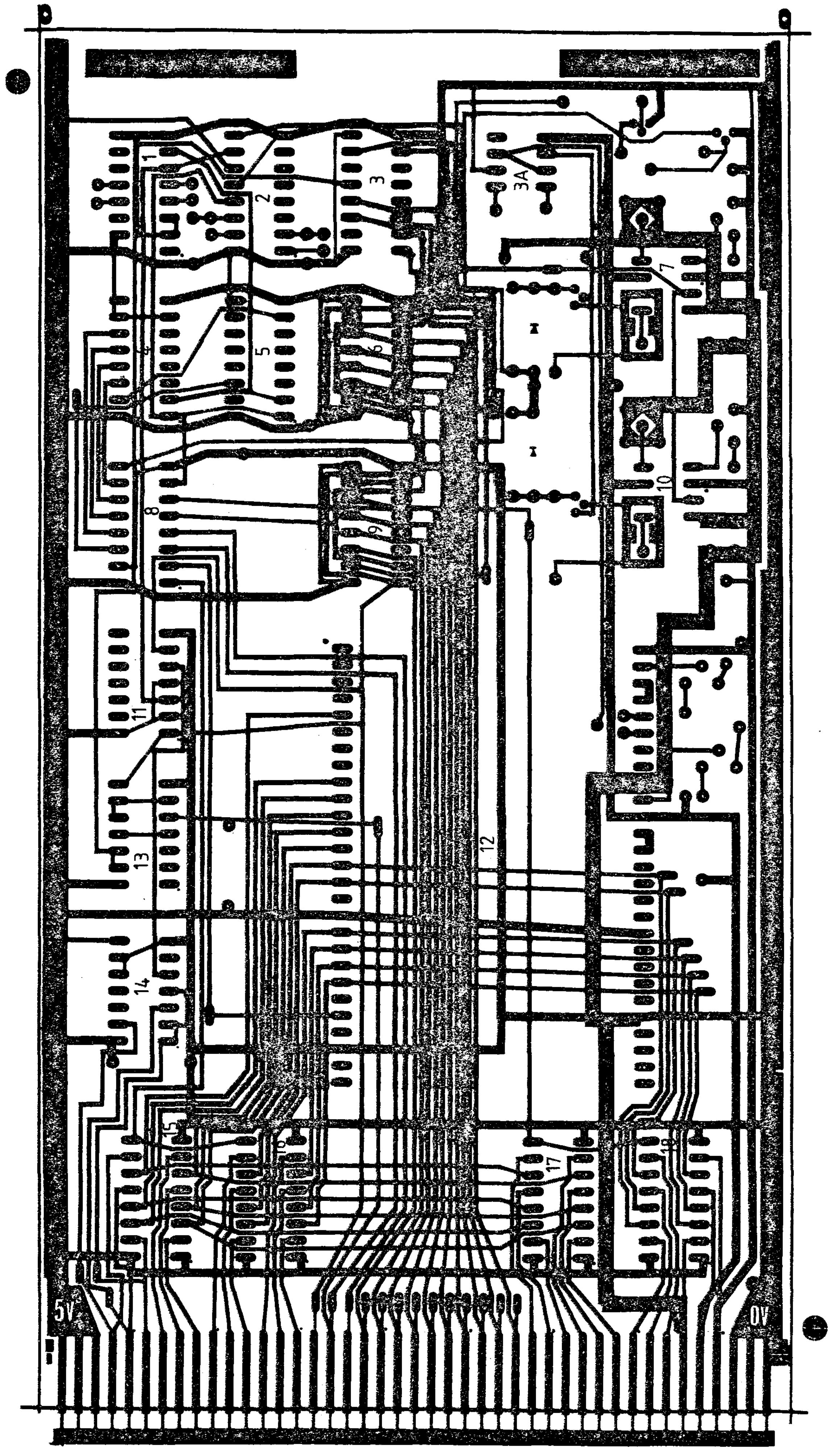
▭ ≡ LATCHED

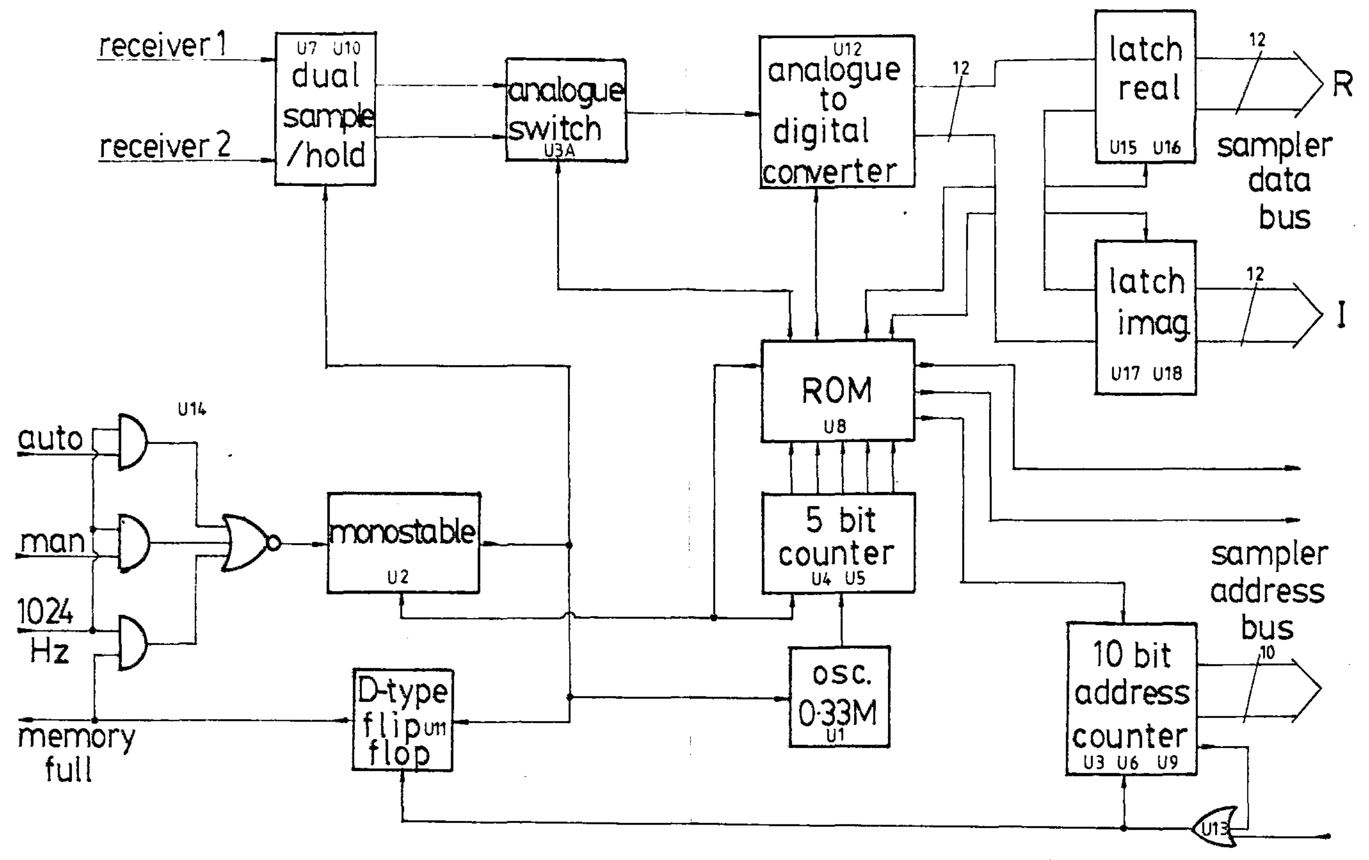
REMARKS	PROM address Hex	$\boxed{U3}$				$\boxed{U4}$				$\boxed{U5}$				U14				U15				U17				U18				$\boxed{U19}$				$\boxed{U20}$				$\boxed{U21}$						
		9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12	9	10	11	12			
		$\overline{VAL}$	$\overline{INT}$	$\overline{RET}$	$\overline{WET}$	OC4	MUXINP/CLK	OC3	OC2	SBC	OC1	SBB	SBA	SCOPE TRIG	$\overline{TCLR}$	$\overline{RT}$	$\overline{SR1}$	$\overline{CLT}$	$\overline{CKR1}$	$\overline{LD2}$	$\overline{CKW^P}$	$\overline{B}$ R/S	$\overline{CK1}$	$\overline{CKM}$	$\overline{CKDA}$	$\overline{CK4}$	$\overline{CK3}$	$\overline{BCLK}$	$\overline{CK2}$	A + B	$\overline{en(N-n)}$	$\overline{en\ n}$	$\overline{en(A+B)}$	$\overline{en1}$	$\overline{en4}$	$\overline{WE2}$	$\overline{LD3}$	+/$\overline{CLR}$	$\overline{en3}$	$\overline{LD1}$	$\overline{en2}$	STATUS BIT	opcode	
UNPROGRAMMED STATE		1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0		

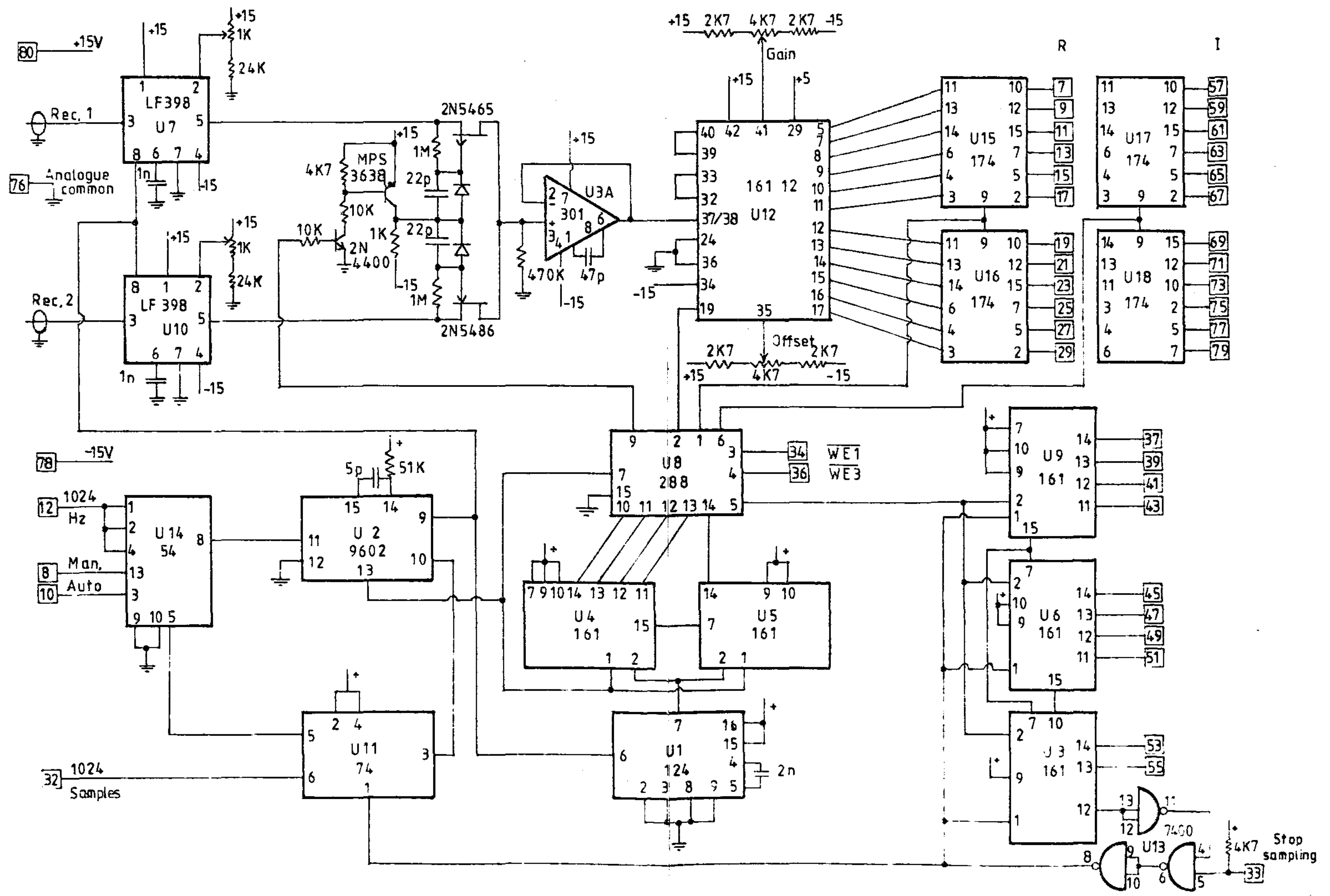




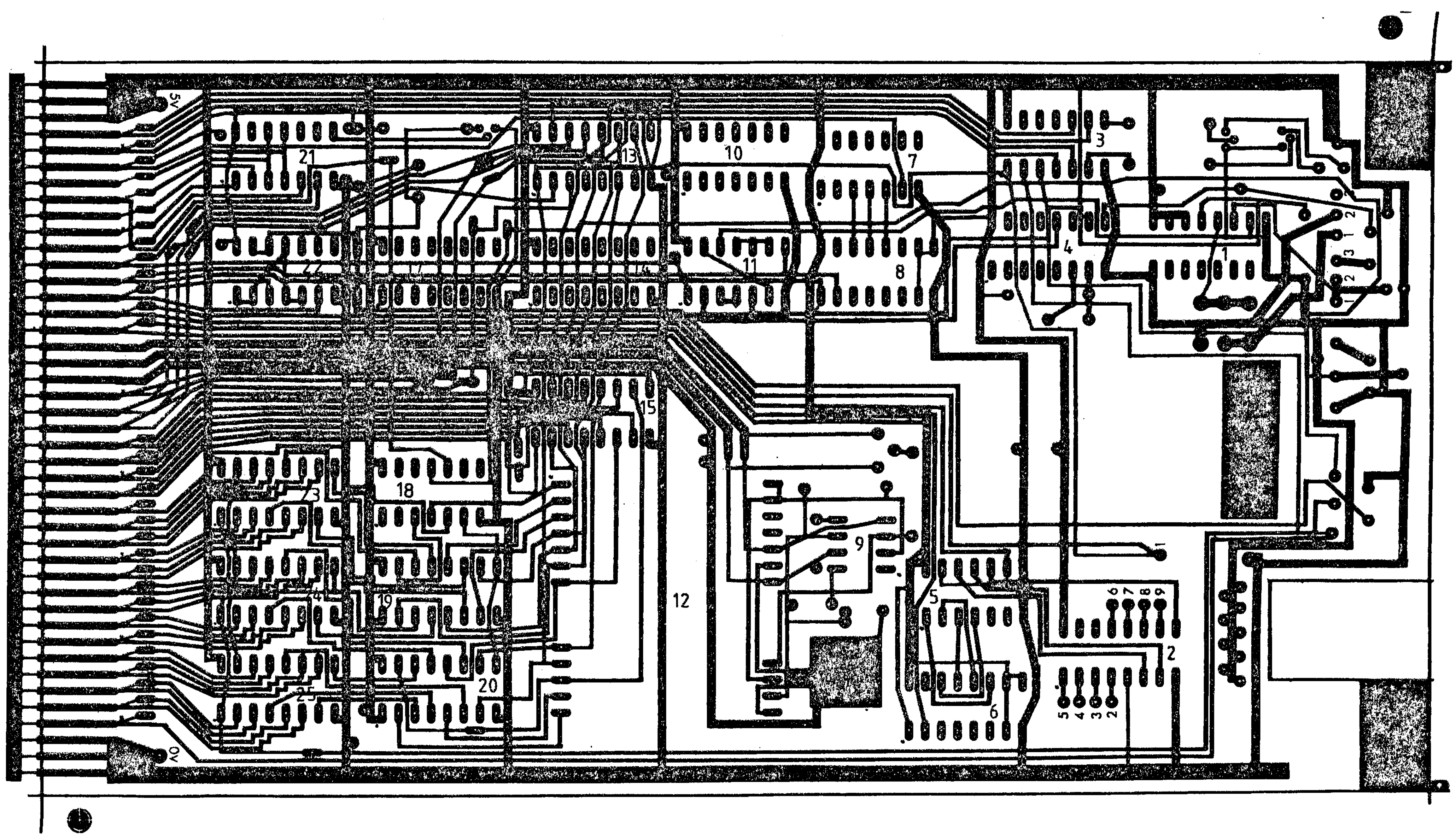
Back Plane Wiring and Power Supply

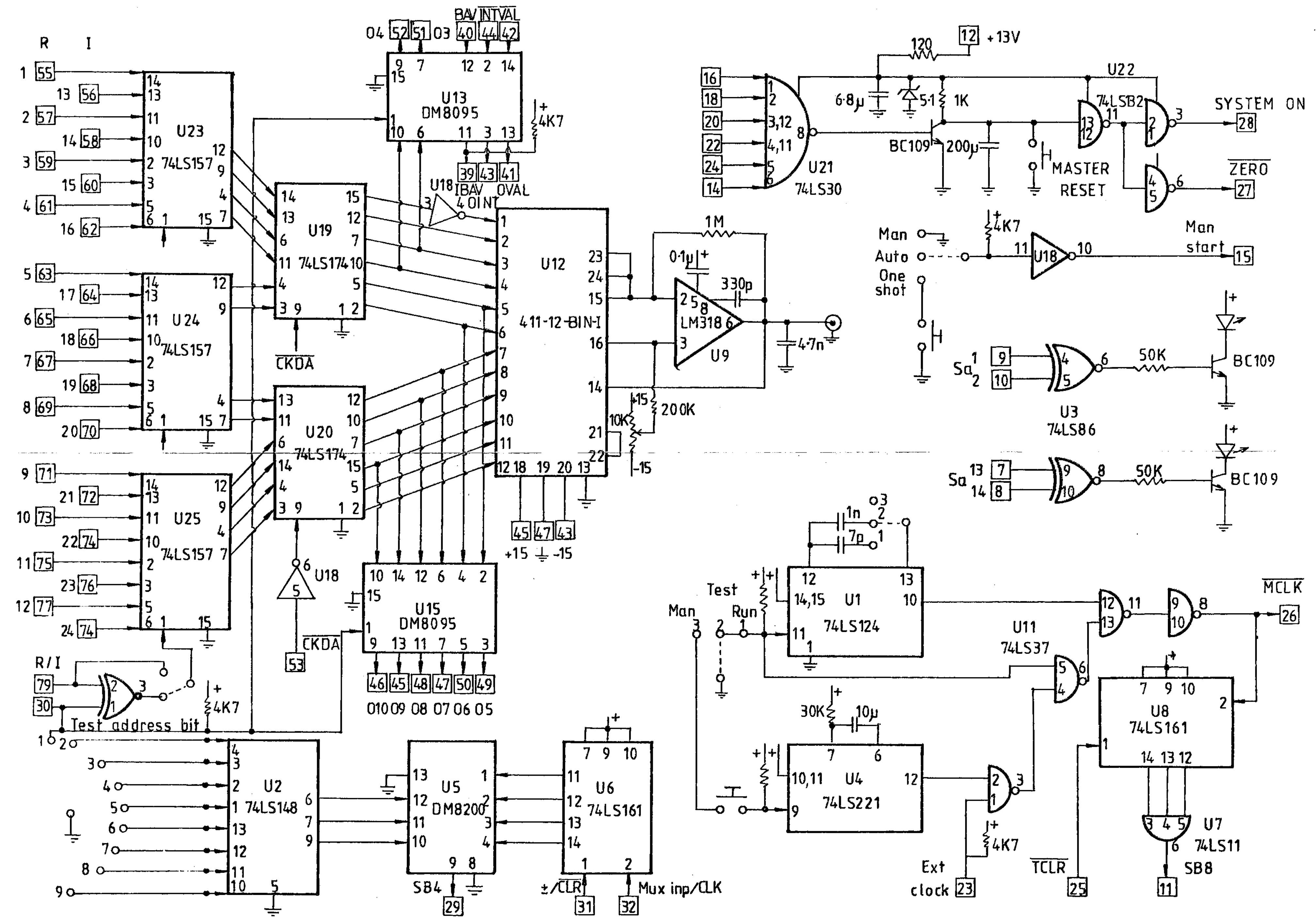






Analogue Input Board

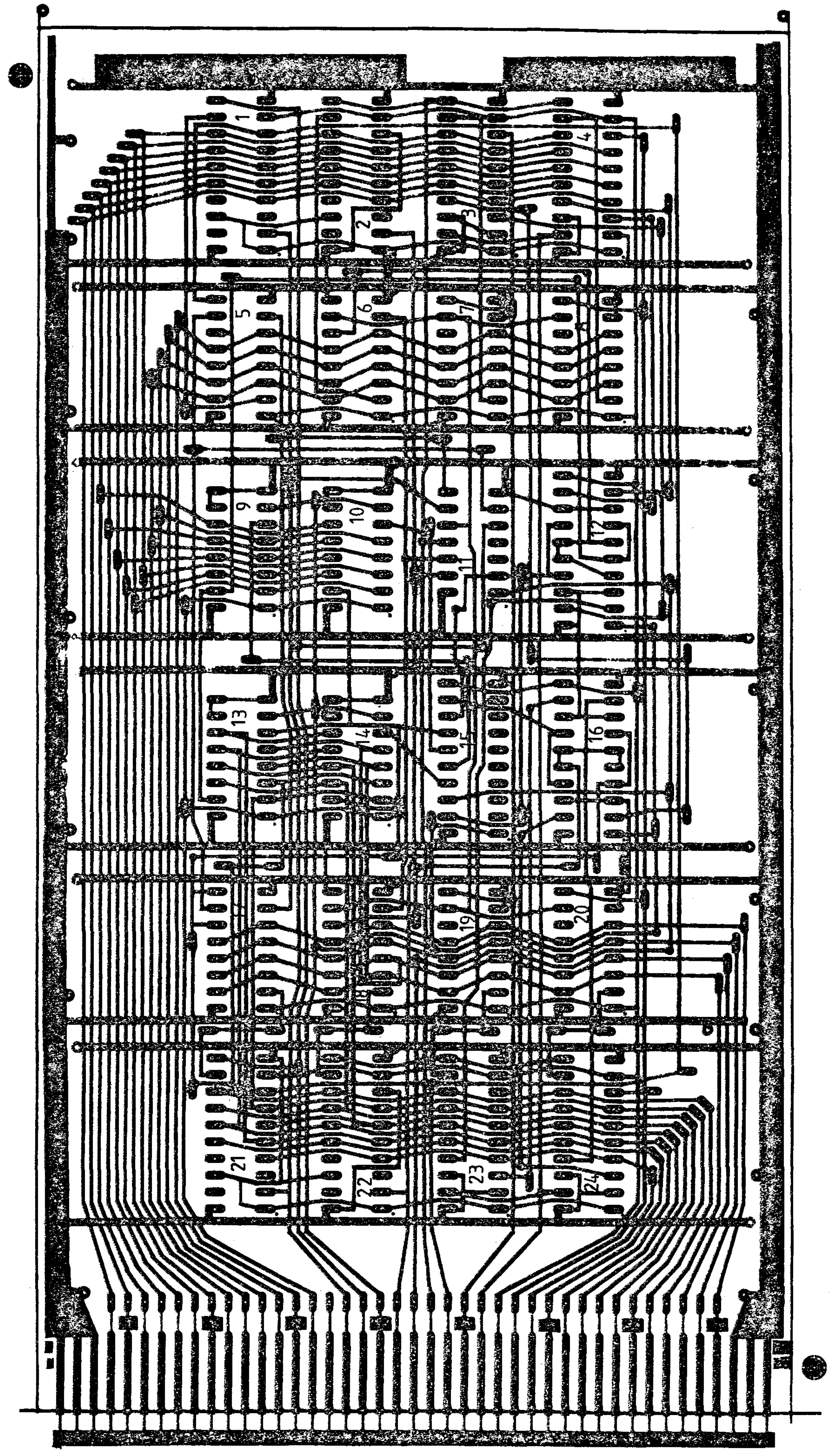


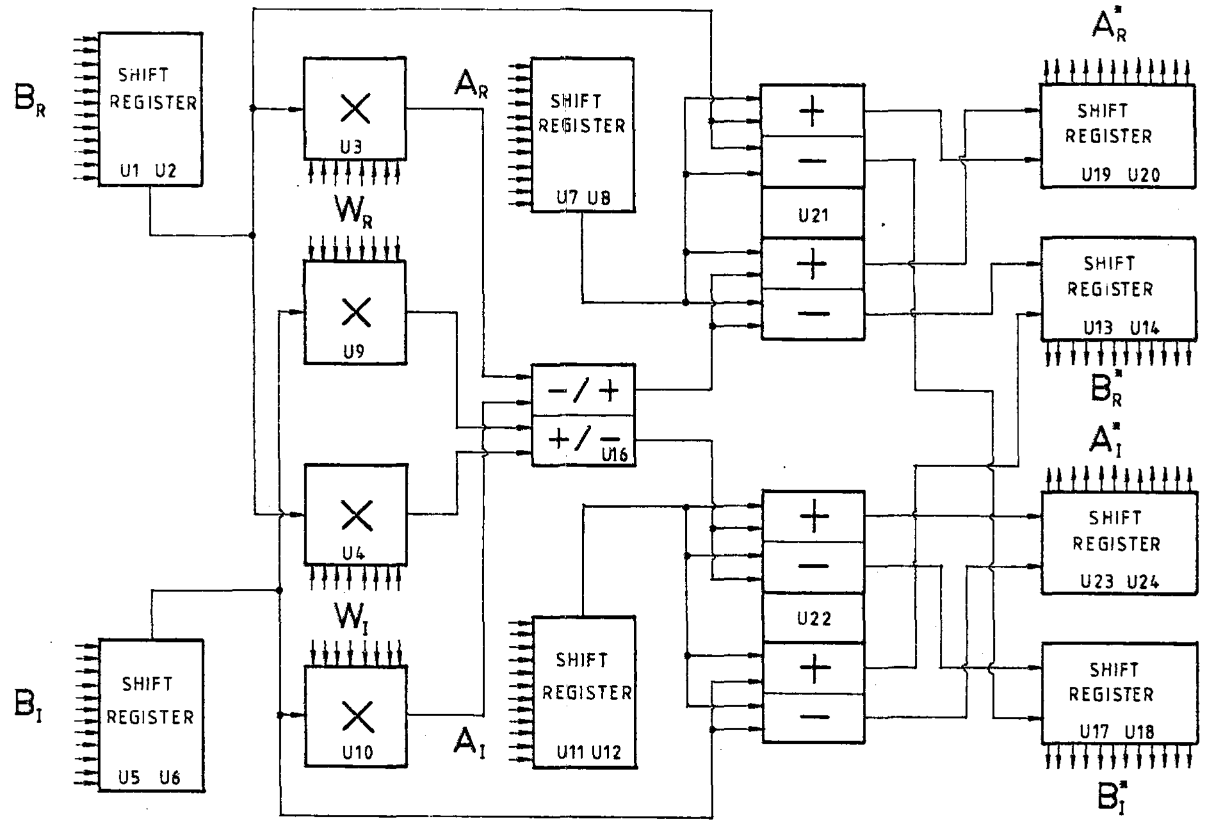


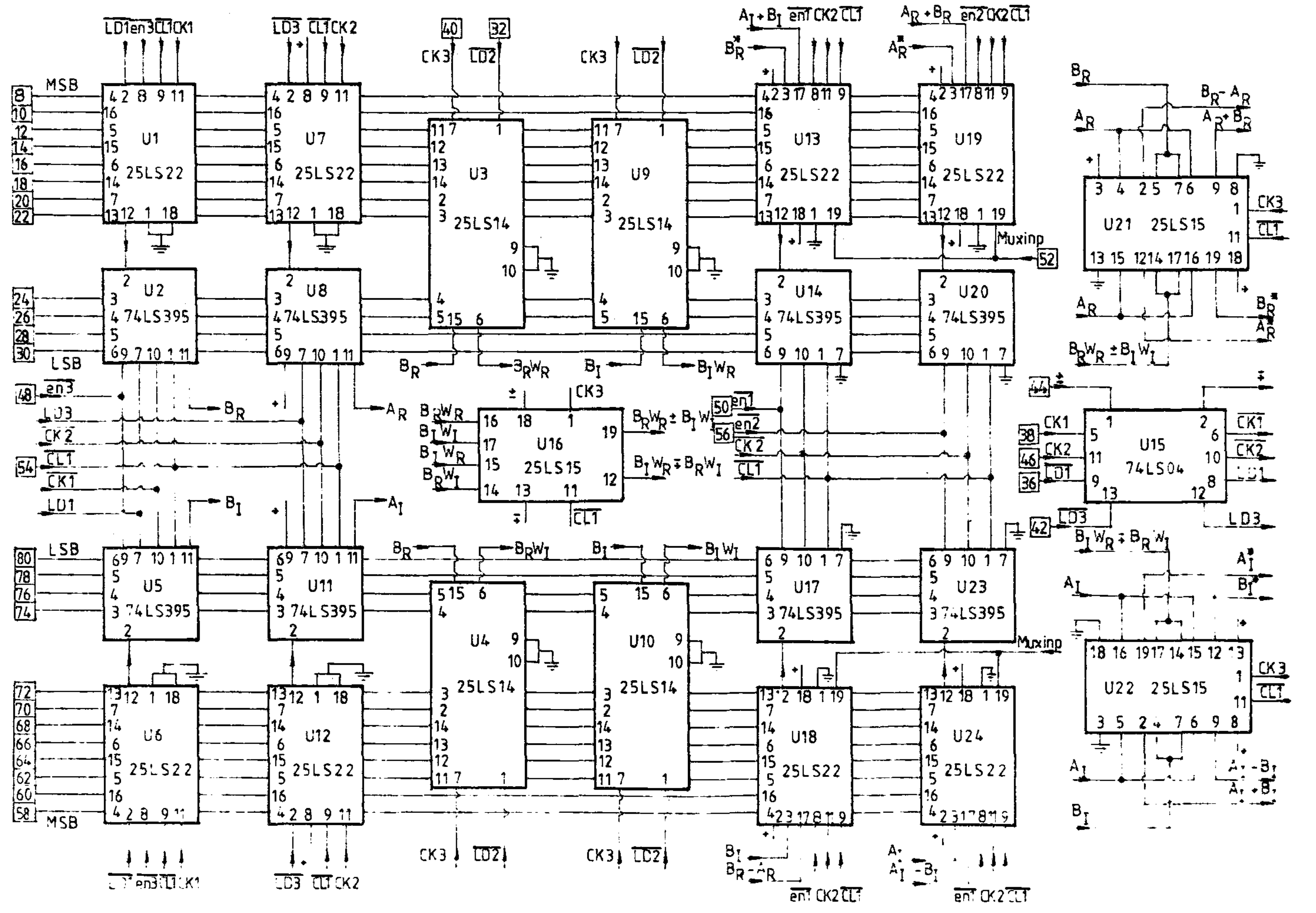
Appendix 3

- 93 -

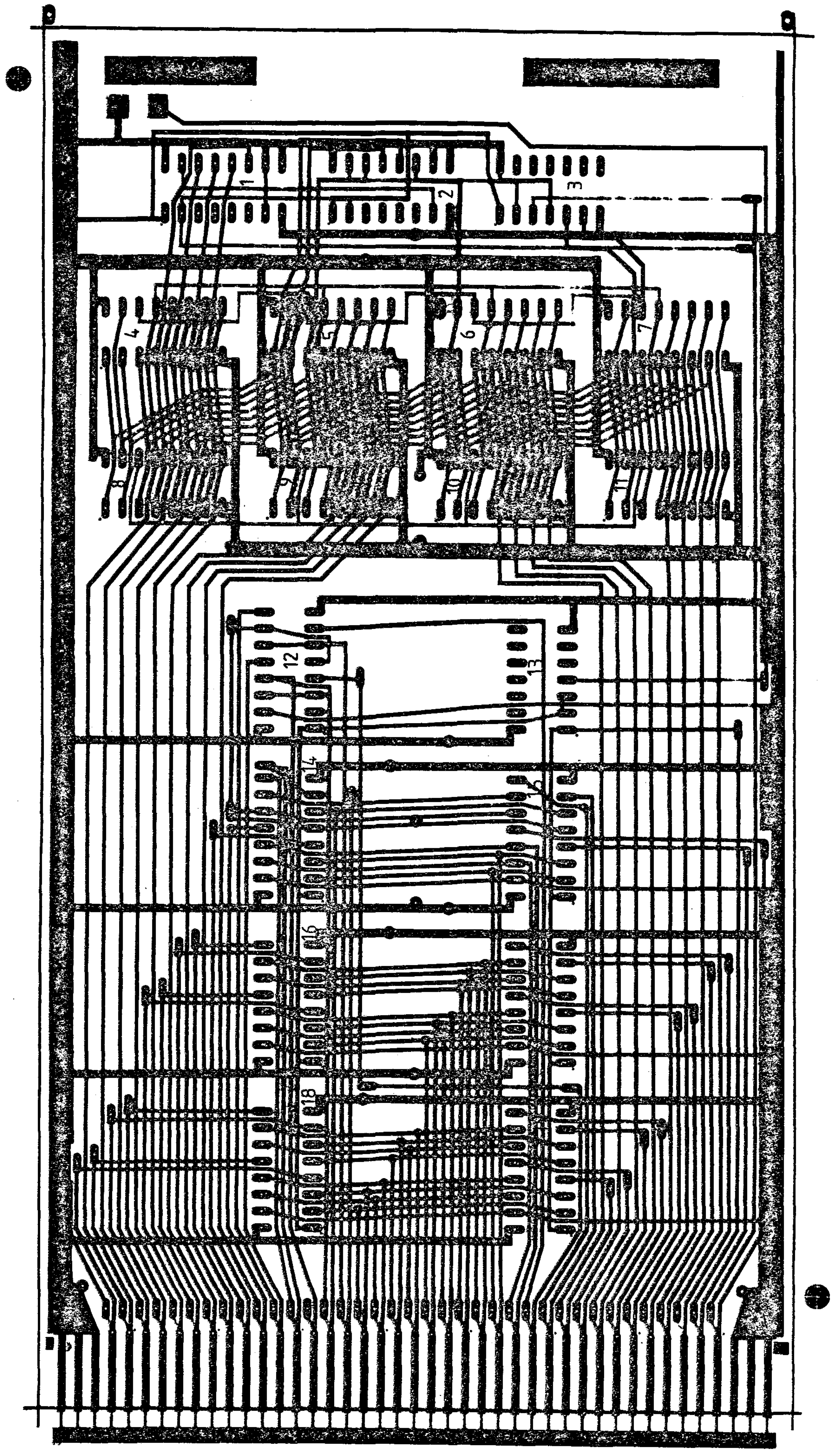
Test Control Board

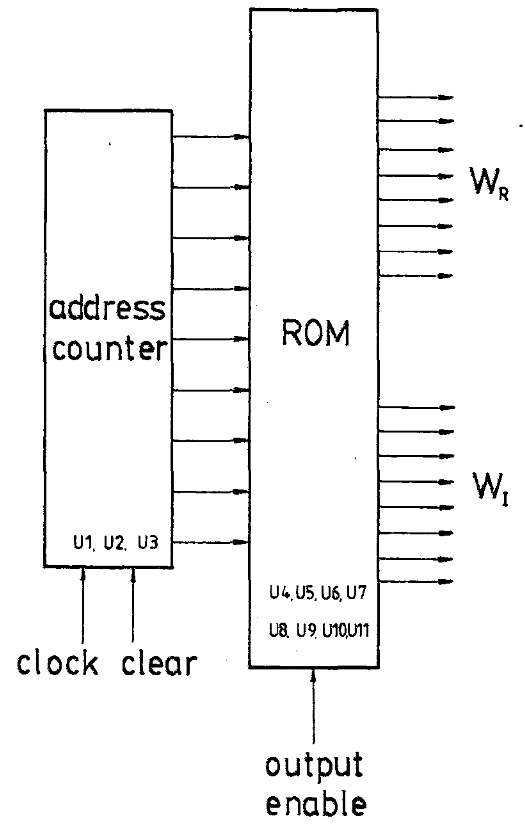


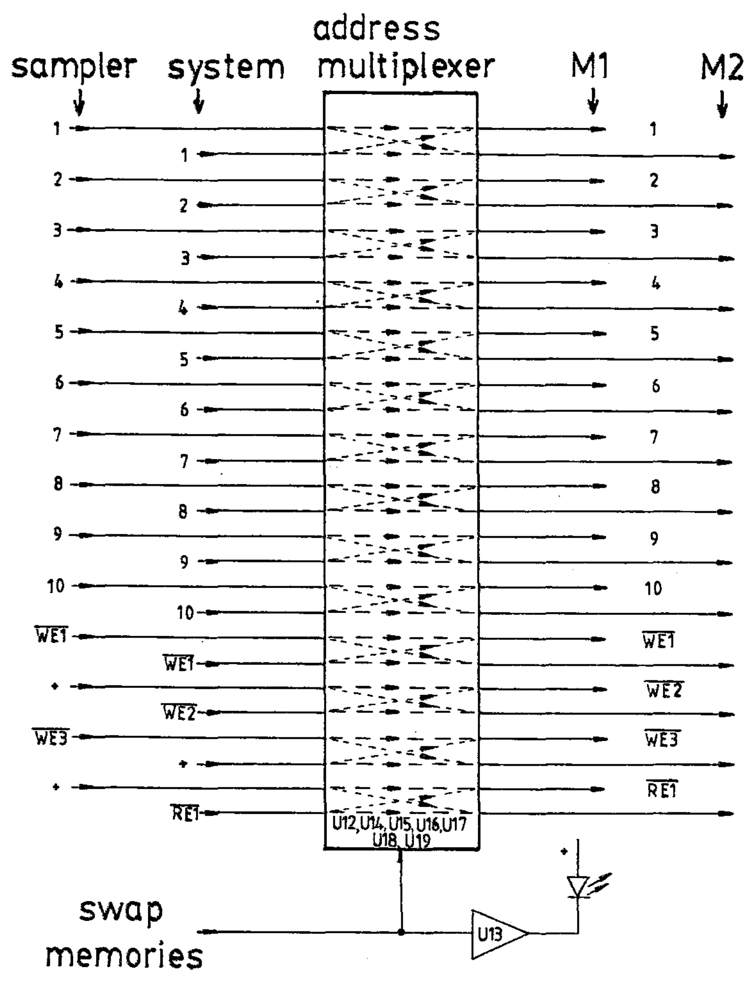


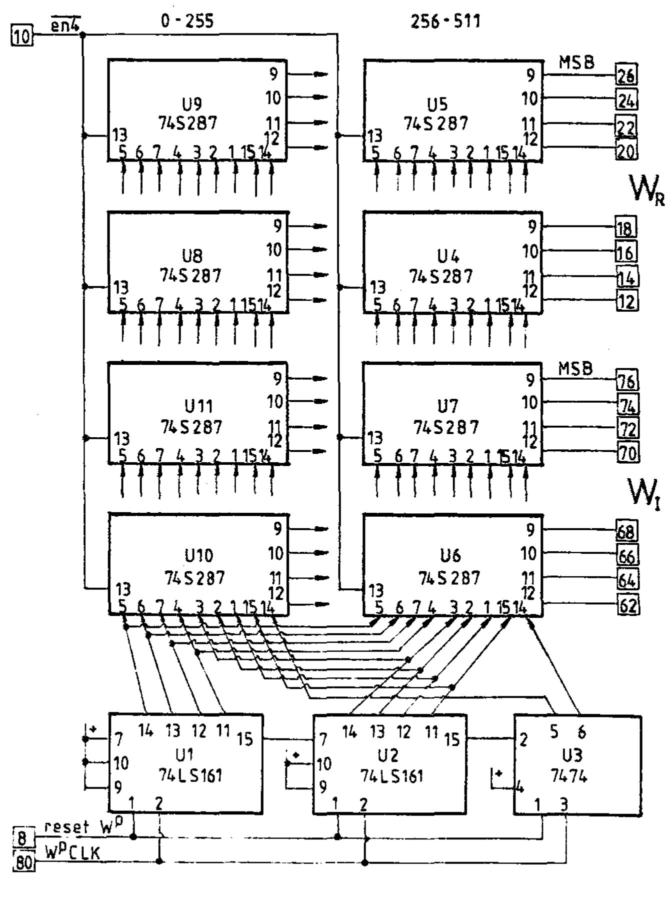


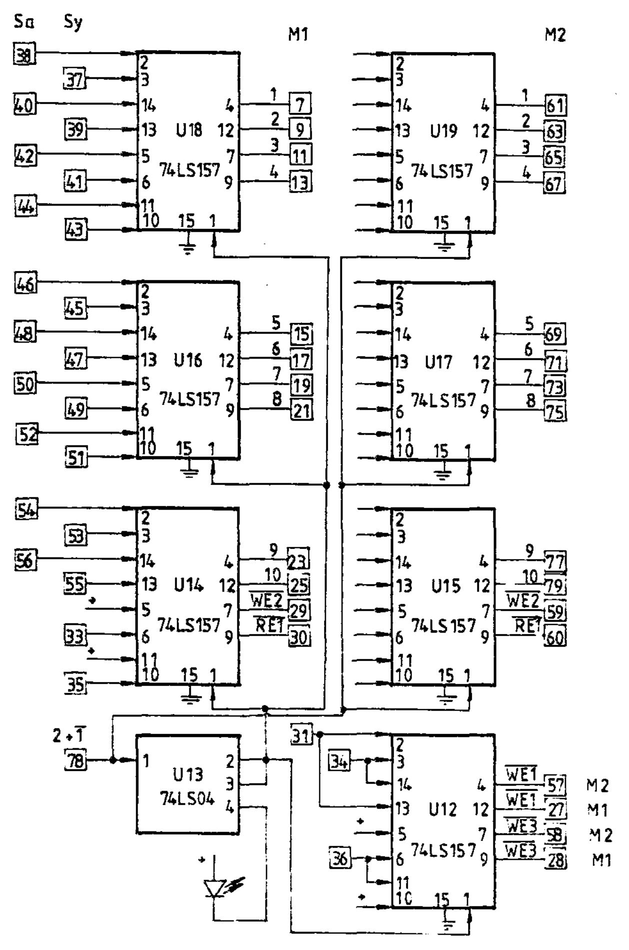
Arithmetic Unit Board


1
2
3
4
5
6
7
8
9
10
11
12
13
14
16
18

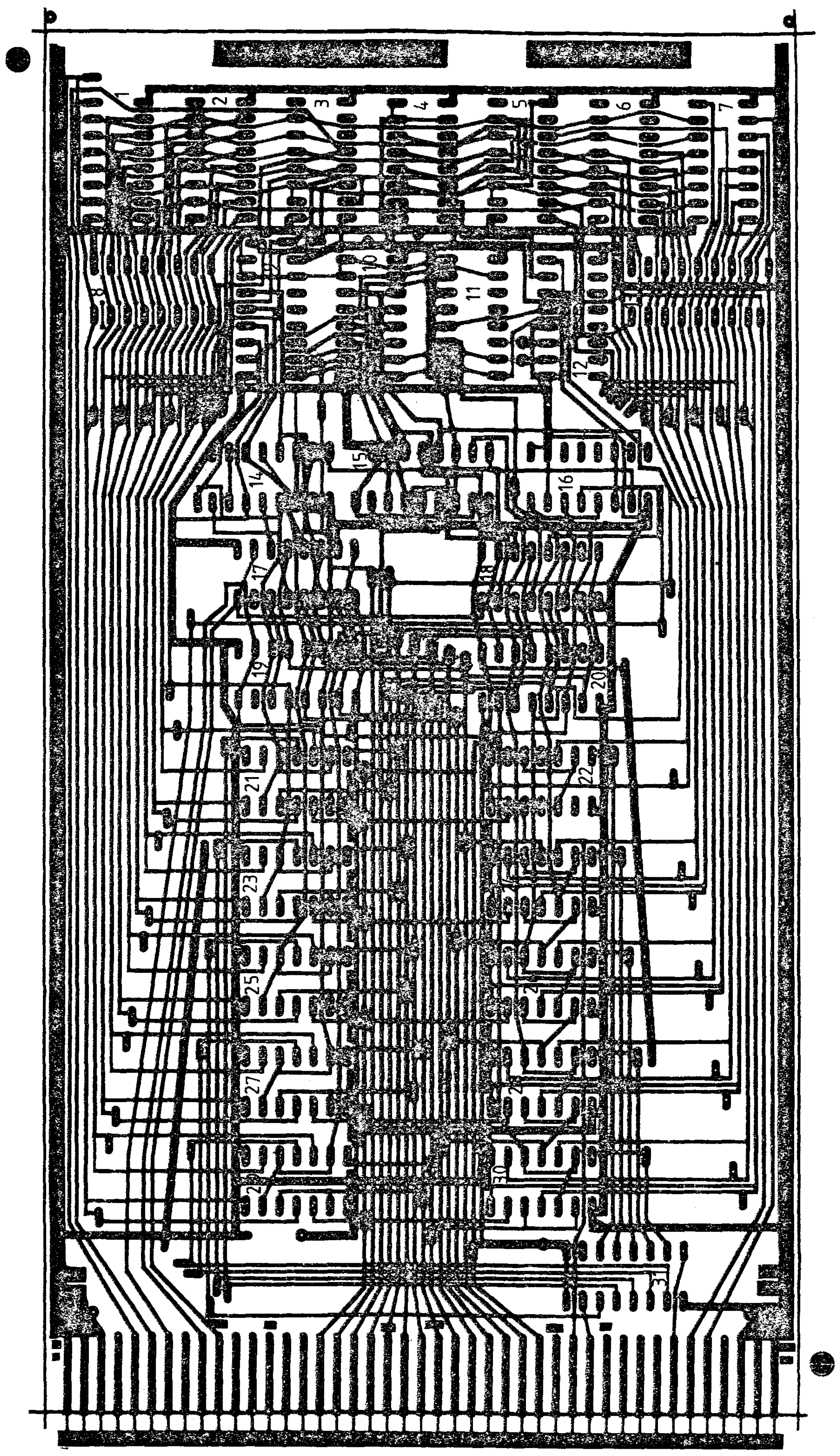

address counter
U1, U2, U3
ROM
U4, U5, U6, U7
U8, U9, U10, U11
W_R
W_I
clock
clear
output enable

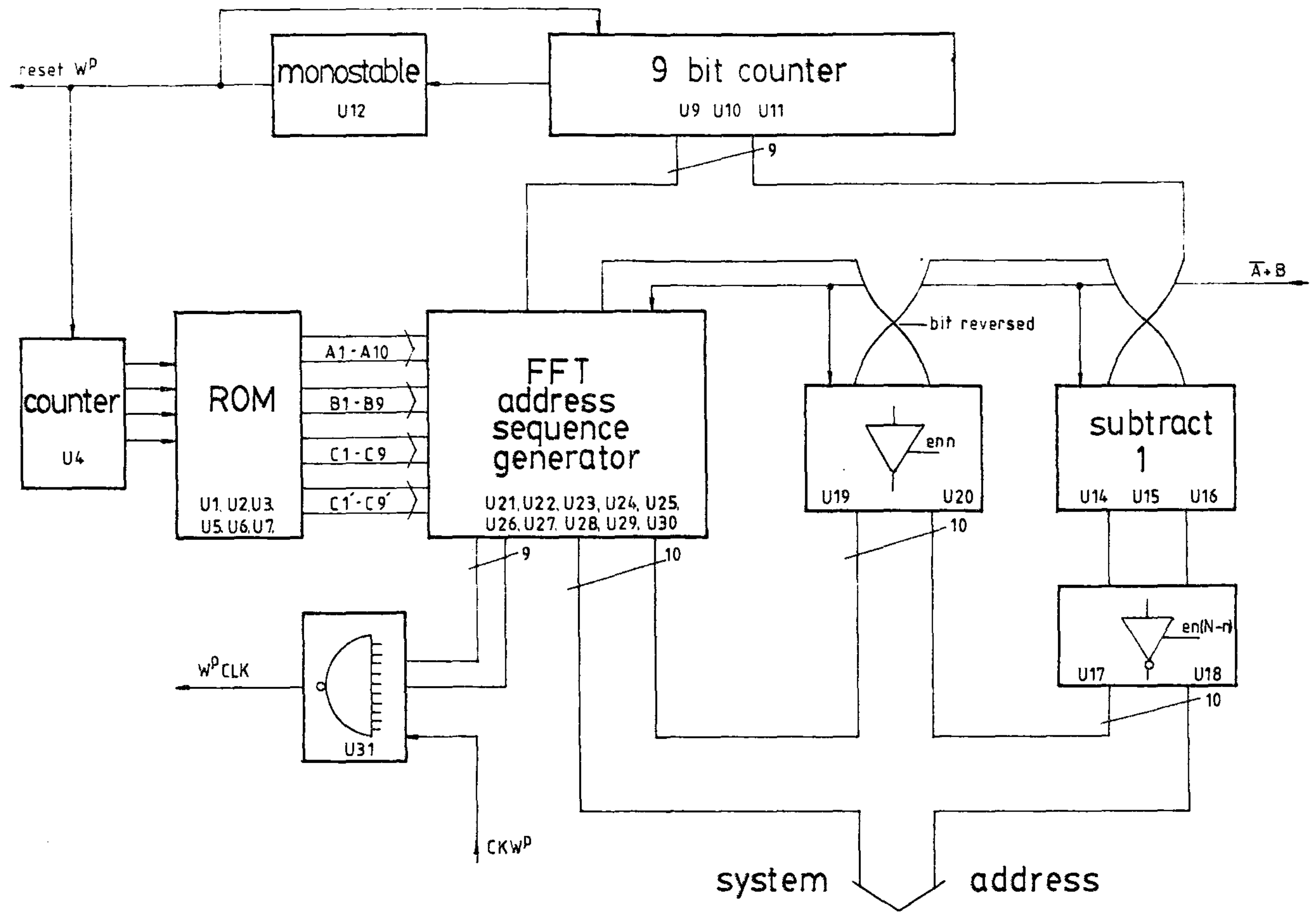

sampler
system
address multiplexer
M1
M2
WE1
WE2
WE3
RE1
U12, U14, U15, U16, U17
U18, U19
U13
swap memories

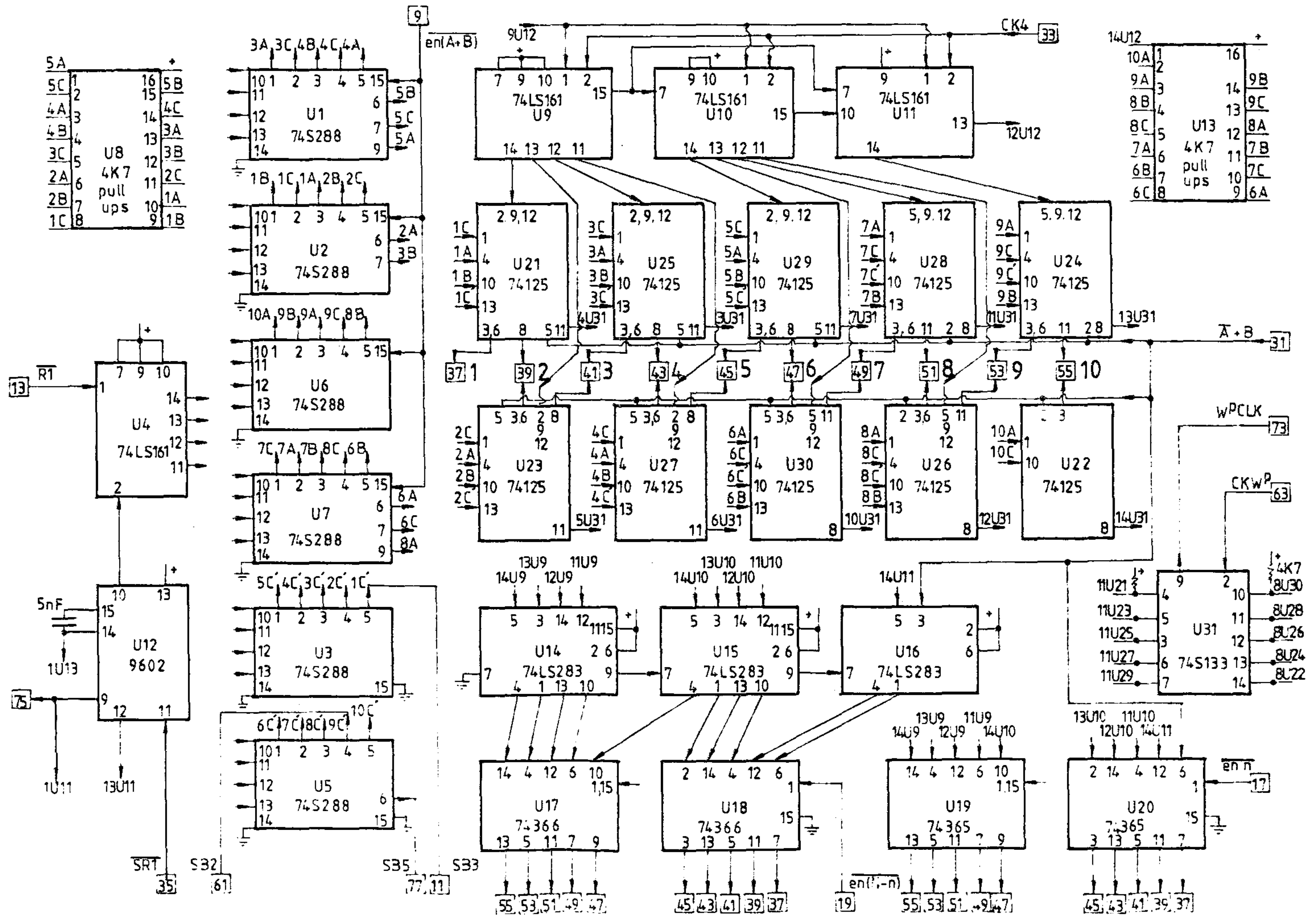

en4
0 - 255
256 - 511
U9 74S287
U5 74S287
U8 74S287
U4 74S287
U11 74S287
U7 74S287
U10 74S287
U6 74S287
MSB
W_R
W_I
U1 74LS161
U2 74LS161
U3 7474
reset W^P
W^PCLK


Sa
Sy
M1
M2
U18 74LS157
U19 74LS157
U16 74LS157
U17 74LS157
U14 74LS157
U15 74LS157
U13 74LS04
U12 74LS157
2+T

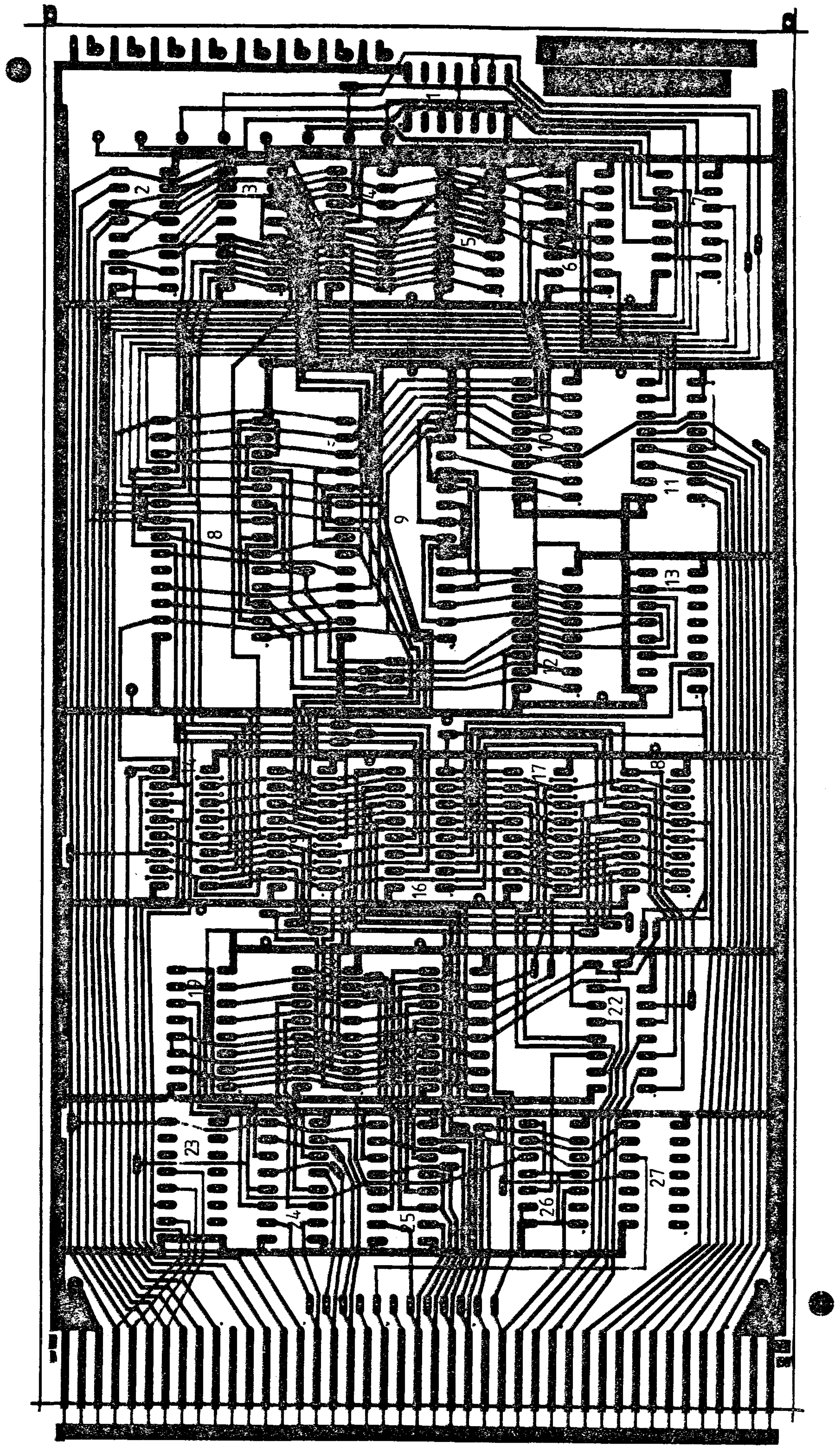
W^P ROM Board

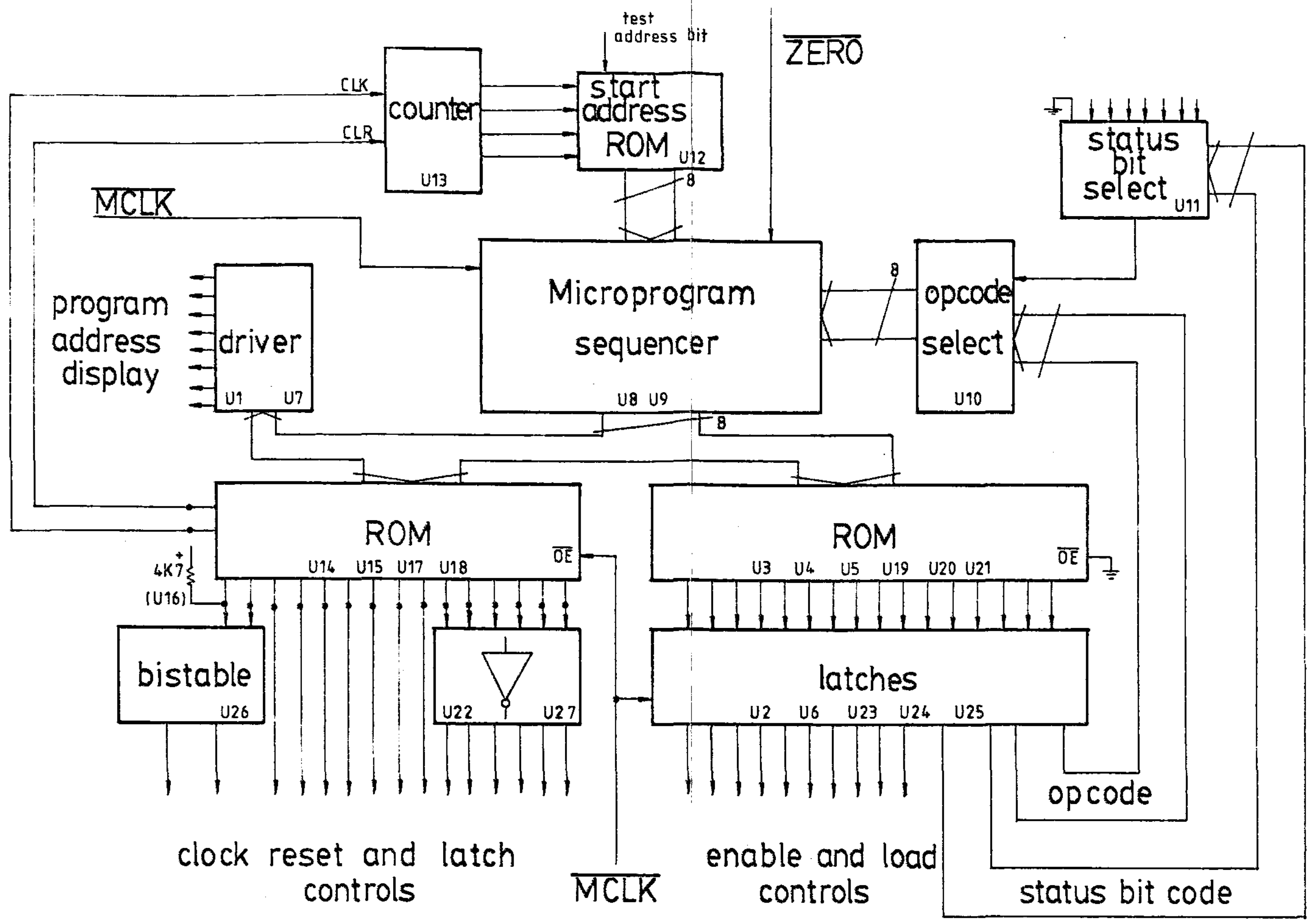


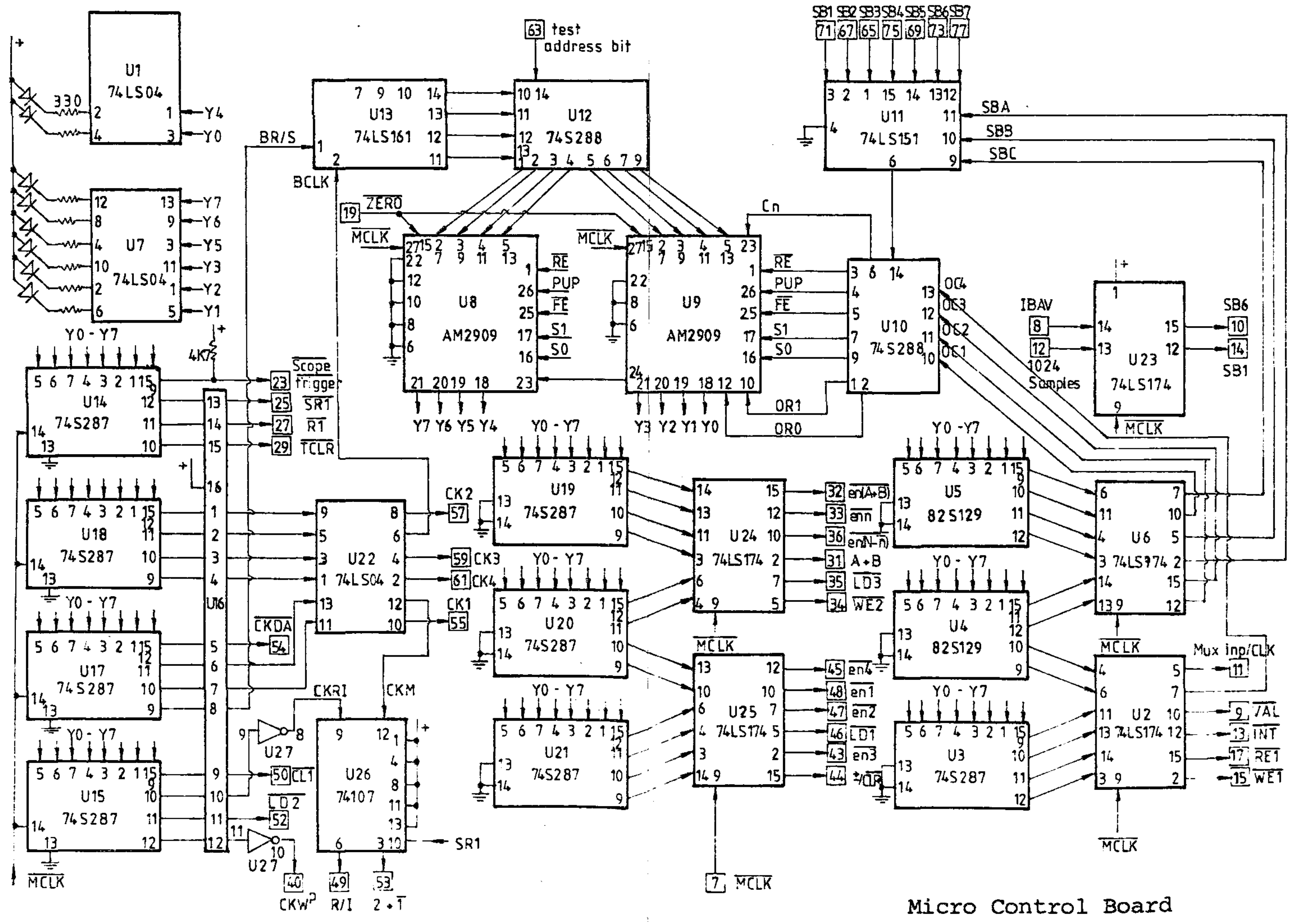




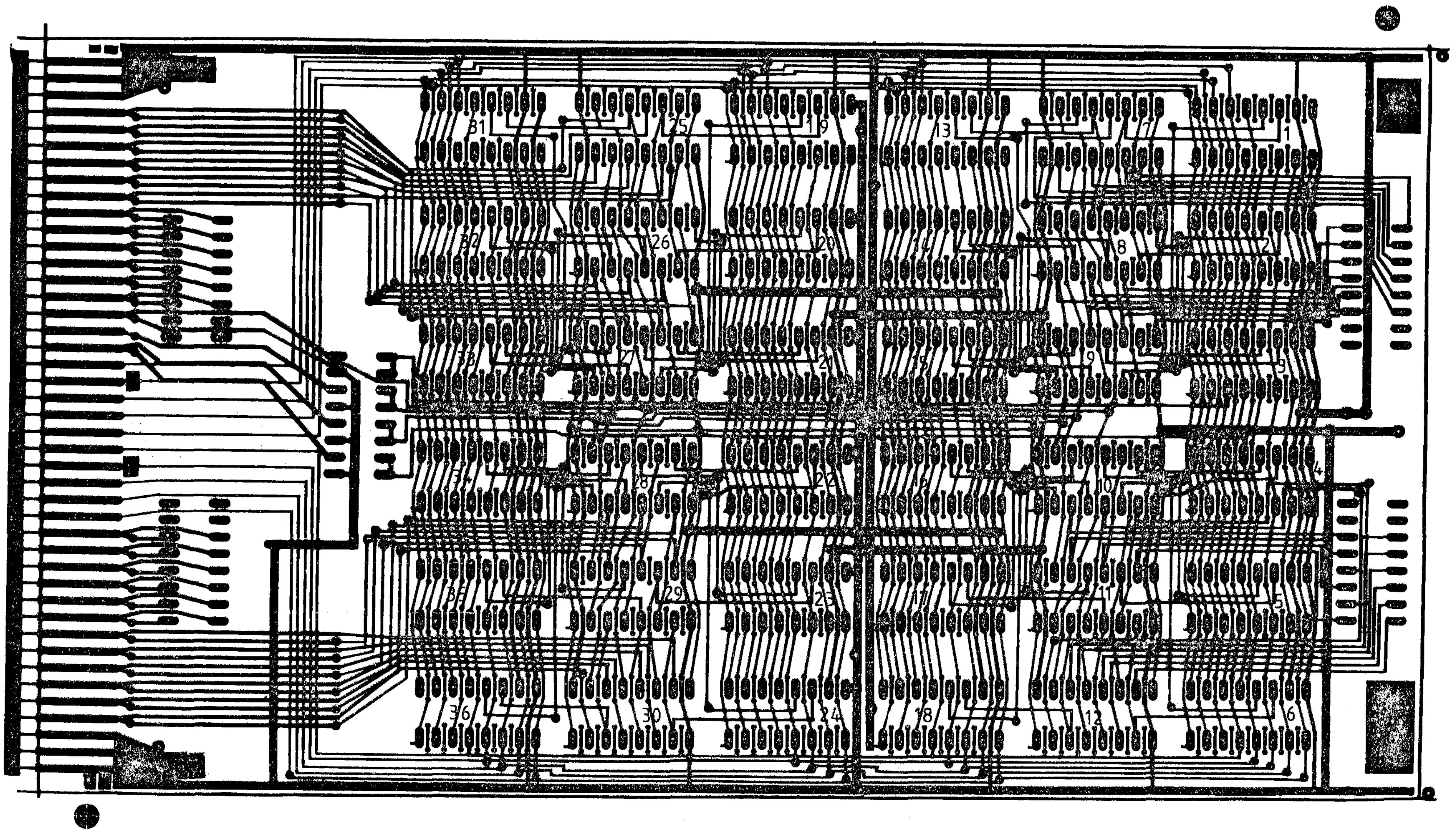
Address Generator Board

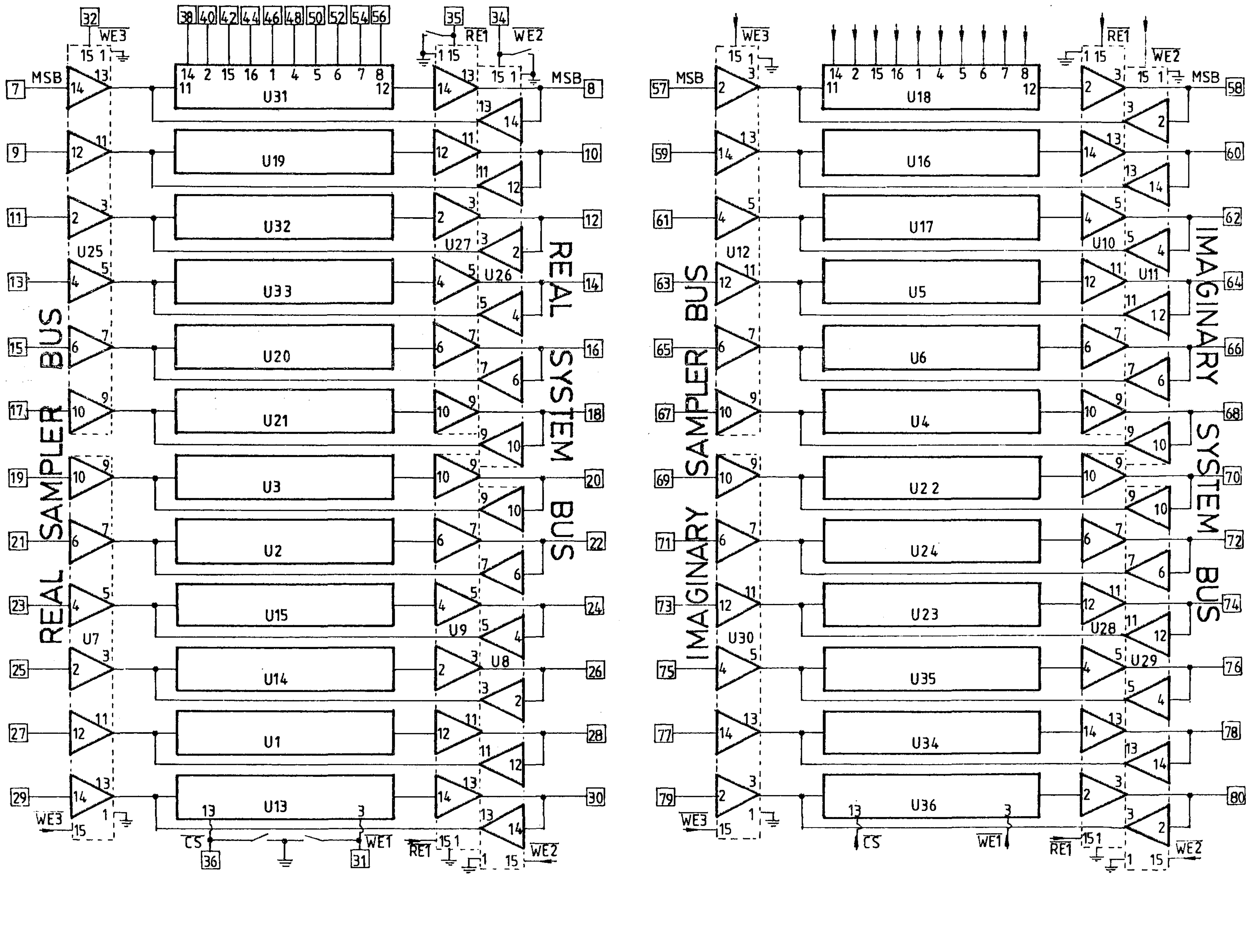






Micro Control Board





Memory Board