My copy

University of Sheffield

Final year project report

Department of Electronic and Electrical Engineering

50kHz Synchronous Receiver

For Medical Use

3rd Year Electronics Project

by J.S Bladen

SUMMARY

This thesis describes the design and construction of a two channel synchronous receiver giving amplitude and phase information at the fixed frequency of 50kHz. The instrument is intended for use in an induced current impedance imaging system. The receiver is controlled by a microprocessor which allows both considerable flexibility in data acquisition strategies and supports an HPIB interface. Images reconstructed from data measured by the receiver are presented.

ACKNOWLEDGEMENTS

I would like to thank Dr R.C. Tozer for his expertise and guidance throughout the project - especially concerning the analogue design. Also Frid and Joanne for providing me with copies of the reconstruction algorithm and for being so patient in explaining how it works!

DIS-ACKNOWLEDGEMENTS

No thanks whatsoever go to Colin (the computer) for the many extremely late nights it caused me. It's the only computer I know that utterly refuses to run the same programme the same way twice. Computers should not be allowed to have a personality!

CONTENTS:

CHAPTER 1: INTRODUCTION

1.1: SUMMARY OF IMPEDANCE IMAGING (1)

Producing images of internal parts of human beings has become a useful diagnostic tool for the medical doctor. Despite the inherent health risks in using X-rays they are still commonly used to diagnose broken bones and malfunctions of certain bodily organs. Ultrasonic scanning is used in the inspection of unborn babies and although this is generally labelled as being 'non-invasive', questions are now being asked as to the safety of this technique. In view of the cost and safety hazards of current scanning techniques, research continues into the development of alternative systems that are safe and capable of operating in real time.

'Induced Current Impedance Imaging' is a technique developed for safely scanning body tissue. High frequency, low magnitude currents are induced in the tissue and these produce voltages around the tissue circumference which are used to construct the required image. Two images can be constructed. The first is a conductivity map. Some tissue such as muscle and bone has a low conductivity and thus shows up well against the high conductivity of the body fluids. The second image available is a permittivity map. This has recently been developed in this department (2) in

an attempt to increase the information available from induced current impedance imaging.

The impedance imaging system lends itself well to the production of real time images.

1.2: SUMMARY OF IMPEDANCE IMAGING SYSTEM

The instrument to be developed is to form part of the signal processing section of an induced current impedance imaging system. To put this instrument into context a block diagram of the system is shown in Fig 1. Here is brief description of the system:

Currents are induced in the body tissue by one of three surrounding coils. These coils are offset from the centre of the tissue and are spaced around the centre by 120 degrees. For medical reasons, an energising frequency of 50 kHz is currently used. Electrodes are placed around the circumference of the tissue and the signals received by these electrodes are used to construct the conductivity and permittivity maps of the tissue cross section.

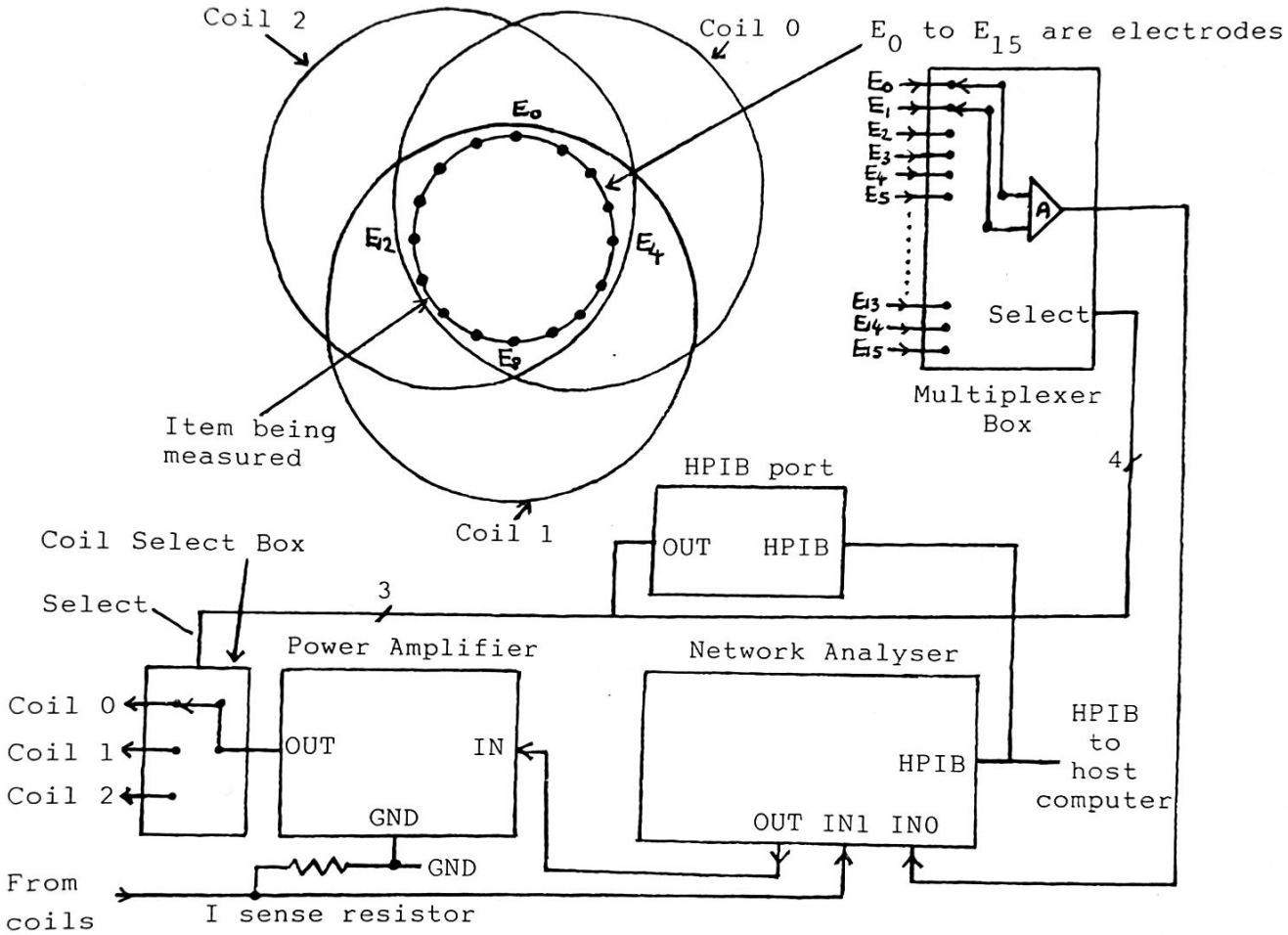
A 50kHz sinewave voltage is fed to a power amplifier which drives current into the required energising coil. The signal is routed to the required coil using a relay based switching system which is controlled by the host computer. Only one

coil is energised at a time. A resistor is placed in series with the coil to enable the coil current to be monitored.

Wires from the electrodes pass to a multiplexer unit which allows the host computer to select which electrode pair is being monitored. The unit also contains a bandpass filter to reduce the sensitivity to unwanted noise.

To construct the images, the amplitude and phase of the electrode voltages must be known relative to the coil drive currents and a network analyser is used to make these measurements. Two signals pass to the network analyser, one from the coil drive current sensing resistor and the other from the output of the multiplexer box. The analyser then calculates the relative gain and phase of the two signals and sends the result to the host computer. When the required information has been collected, the host computer can use the results to construct the required images.

Fig 1: BLOCK DIAGRAM OF THE IMAGING SYSTEM:



There are some limitations to the current system. Firstly the resolution is quite low at about 30 by 30 pixels. Secondly there are problems in obtaining an image of a 3D object. Currents circulating in the 3rd dimension are picked up by the electrodes around the 2D slice of interest causing errors in the readings which are difficult to eliminate. Current measurements are made on a thin 'slice' of conductive liquid into which materials of varying conductivity and permittivity are placed.

However, the system does have a great deal of promise both for medical use and in other areas of electronics. The purpose of this project is to reduce the cost of the hardware required for the imaging system by replacing the expensive wide band network analyser by a cheap single frequency version that has considerable flexibilty in the way that the data is sampled.

1.3: DEFINITION OF PROBLEM

The network analyser must conform to the following specification:

1) The instrument must have two signal inputs and from these be able to produce information which may be used, directly or indirectly, to determine the relative gain and phase between the two signals, at the required frequency.

2) The instrument must provide its own frequency source to which the receivers are automatically tuned.

3) All gain and phase measurements made for the construction of one particular image must be made to a relative accuracy of 1%. Absolute accuracy is not important.

4) The instrument should be not be sensitive to non-synchronous interference and noise.

5) The signal inputs must be able to cope with a range of input voltages from 0.05 volts rms to 1 volt rms, to allow for varying body conductivity, electrode amplifier gain and coil drive power.

6) The instrument should tolerate the input levels varying by up to 20 dB in the course of one set of measurements.

7) The instrument must be able to communicate the results down a standard speed HPIB interface to a host computer.

8) The cost of the instrument must be a substantial saving on the cost of the current wide band network analyser.

9) The instrument should me made portable, consistant with good mechanical and electrical design.

10) The instrument should have an output to control the electrode and coil select switches.

CHAPTER 2: DEVELOPMENT OF PROPOSED SYSTEM

The instrument required is a two input network analyser operating at the fixed frequency of 50kHz. To measure the relative amplitude, the amplitude of each input may be measured and the results divided. To measure the relative phase, the phase of each signal can be measured relative to the same arbitary point and the results subtracted. Thus some means of measuring the absolute amplitude and phase is required.

2.1: BUILDING BLOCKS

The following few sections briefly explore areas of interest in the search of a suitable system:

2.1.1: ENVELOPE DETECTION

Envelope detection is a non-synchronous method of detection which produces amplitude information. It is possible to get a narrow bandwidth and thus good noise immunity by using narrow band filters. However as the receiver bandpass filter frequency and the carrier frequency may drift apart, the bandwidth must not be too narrow. Narrow bandpass filters also suffer from long response times. It is not possible to produce phase information using this technique and so it is of no use in this application.

2.1.2: SYNCHRONOUS DETECTION

Synchronous receiving methods use the carrier frequency, or frequencies directly related to the carrier frequency, to process incoming signals. Given a long enough measurement time these methods will be immune to all frequencies other than the frequency of interest. In practice, complete immunity is not possible due to limited measurement time but the synchronous receiver still offers excellent results. Often synchronous receivers use a phase locked loop to reconstruct the carrier frequency. In the case of this network analyser, it is simpler and more accurate to derive all timing signals, including the carrier frequency, from one frequency source. This guarantees that the receiver is synchronous and saves having an external frequency source for the 50kHz carrier signal.

2.1.3: DOWN CONVERSION AND BANDLIMITING

Converting the frequency of the incoming signal to a much lower intermediate frequency (IF) has several advantages. Firstly it enables the network analyser frequency to be changed without re-designing the filter. All that is required is to change the frequency of the signal which is mixed with the incoming signal. Secondly it enables a narrow input bandwidth to be achieved without having to use complex filters. Thirdly it reduces the demands on the sample and hold amplifier.

To ensure that the information required is preserved, consider the following. Let the incoming signal be multiplied by a constant amplitude signal whose frequency differs from the carrier frequency by a small known amount. The output from the multiplier will be the sum and difference frequencies of the input signals and the low frequency component may be selected by a band pass filter. The amplitude of this signal will be proportional to the amplitude of the incoming signal and the phase will be equal to the phase of the incoming signal plus the phase offset of the multiplier signal.

ie: Let the incoming signal be: $y_1 = A_1 \cos(w_1 t - p_1)$

Let the multiplier signal be: $y_2 = A_2 \cos(w_2 t - p_2)$

A=amplitude; w=angular frequency; t=time; p=phase.

Then the low frequency component from the multiplier is:

$$m_{low} = 0.5 A_1 A_2 \cos((w_1 - w_2)t - (p_1 - p_2))$$

Both A_2 and p_2 will be cancelled when the relative amplitude and phase between the two inputs is calculated. Thus the required information is preserved.

2.1.4: FINDING AMPLITUDE AND PHASE

If a periodic sinewave is sampled at two points which are offset by ninety degrees, then these samples may be used to determine the amplitude and phase of the sinewave. Pythagoras' theorem is used to determine the amplitude and simple trigonometry to determine the angle.

ie: Let the signal be: $y = A \cos(wt-p)$

Let the sample time be: t_{sample}

Then the first sample will be: $s_1 = A \cos(wt_{sample}-p)$

and the second sample will be: $s_2 = A \cos(wt_{sample}-90-p)$

or: $s_2 = A \sin(wt_{sample}-p)$

Therefore: $\text{Amplitude} = (s_1^2+s_2^2)^{0.5} = A$

$\text{Phase} = \arctan(s_2/s_1) = wt_{sample}-p$

Note that in the above example the phase is relative to a cosinusoidal wave starting at t=0. The phase offset wt_{sample} is due to the sampling time being non zero. If both incoming samples are sampled at the same time then the offset will be the same for both and will cancel out in the calculation of their relative phase.

2.2: PROPOSED SYSTEM

The design of the proposed system was based on the synchronous detection technique due to its good noise immunity and ability to resolve phase information. The configuration used is described below.

2.2.1: SYNCHRONOUS RECEIVER CONFIGURATION

The incoming signals can either be converted to a low IF frequency or directly to dc. The advantage in converting directly to dc is that a simple low pass filter can be used instead of a bandpass filter. However this system does require two complete measurement circuits - one for the in phase component and one for the quadrature component. Converting to a low IF frequency requires a bandpass filter, but as the result is still ac, the in phase and quadrature components can be selected by the sampling circuit after the IF filter.

More complicated combinations exist which use both of the above techniques. Some of these lend themselves to digital filtering which enables the implementation of high order filters with bandwidths which are easily alterable. As these facilities are unlikely to be required by this network analyser, this approach was not taken.

The proposal drawn up for the analogue design was as follows: The incoming signals of 50kHz are down converted to the IF frequency of 1.282kHz. These are then filtered and passed to sample and hold amplifiers followed by analogue to digital converters. The 50kHz output frequency, the 51.282kHz local oscillator frequency and the sampling frequency are all derived from an 8MHz master clock. The frequencies are chosen for integer division ratios.

2.2.2: CONTROL BOARD

In order to control the operation of the analogue circuit, a micro-processor was used. The reasons for using a CPU rather than specialised hardware are as follows:

1) The data from the A to D converters must be read in a guaranteed time. The host computer to be used with the instrument is a multi-tasking network computer, which is not at all suitable for high speed dedicated tasks.
2) The only link with the host computer is via an HPIB interface. It is difficult properly to implement a bi-directional HPIB interface without using a CPU.
3) A CPU allows easy modification of the sampling methods and clock frequencies etc without modifying the hardware.

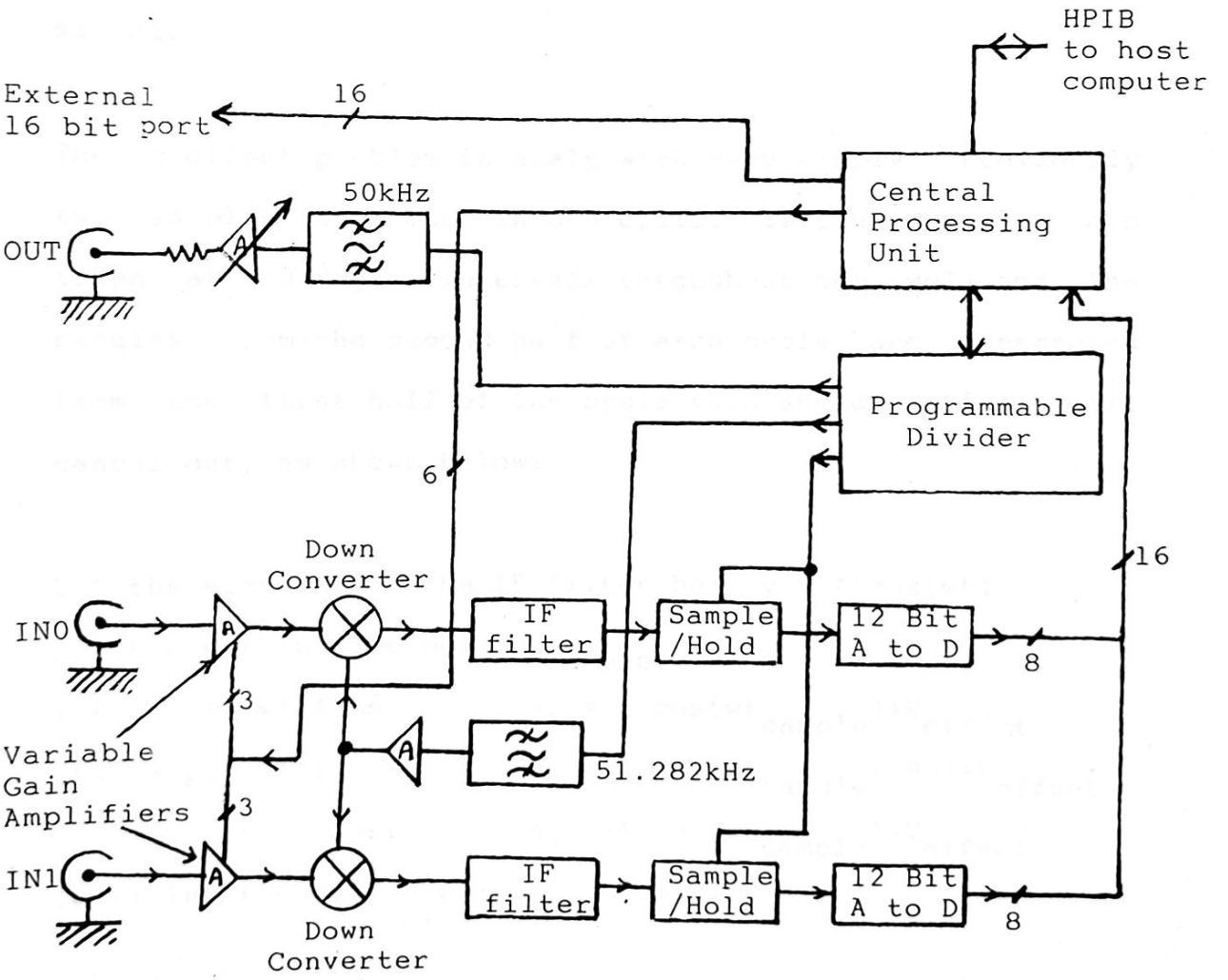
For reasons of speed, actual calculation of the relative gain and phase was not carried out by the CPU. This would have involved floating point arithmetic and would be substantially slower than leaving this to the 68020 based

host computer with maths co-processors! However the onboard CPU does carry out the arithmetic required to reduce the two tables of sample data to four hexadecimal numbers as well as performing overflow checks on the data.

One of the dangers of using a CPU is the injection of CPU generated interference into the analogue circuitry. With the inherent noise immunity of the synchronous receiver based design, this was not expected to be a problem. However to reduce interference from the CPU double buffering was used in the path from the A to D converters to the CPU data bus.

A block diagram of the proposed overall system is shown in Fig 2. The diagram shows the 50kHz output circuit and both channels of the input circuit, together with the digital control circuit. A variable input attenuator and a 16 bit external port are also shown and were added to conform to the system specification.

Fig 2: BLOCK DIAGRAM OF THE PROPOSED SYSTEM:



2.3: METHODS OF SAMPLING

2.3.1: OVERVIEW OF THE SAMPLING USED

So far it has been stated that by taking one in-phase sample and one quadrature sample the required amplitude and phase may be calculated. However two factors have not been mentioned yet. The first is dc offset errors in the sample and hold amplifiers and A to D converters. The second is noise, amplitude and phase variations in the incoming signal.

The dc offset problem is dealt with very simply. Previously two samples were made in one cycle. If four samples are taken at 90 degree intervals throughout the cycle and the results from the second half of each cycle are subtracted from the first half of the cycle then any dc offset will cancel out, as shown below:

Let the signal from the IF filter be: $y = A \cos(wt)$

Let the sample time be: t_{sample}

1st sample will be: $s_1 = A \cos(wt_{sample})+V_{offset}$

3rd sample will be: $s_3 = A \cos(wt_{sample}+180)+V_{offset}$

or: $s_3 = -A \cos(wt_{sample})+V_{offset}$

Combining these: $s_1-s_3 = 2A \cos(wt_{sample})$

Therefore the dc offset V_{offset} is cancelled.

This is equivalent to using a non-recursive digital bandpass filter with square wave coefficients (+1,+1,-1,-1). The problem of noise can be reduced by sampling a large number of cycles and averaging the results. It is possible to extend the principle of digital filtering by increasing the number of samples used. The above example uses four samples per cycle. Providing the hardware is capable of taking the samples fast enough and can produce the required sampling frequency, this can be increased. In this system different sampling modes are offered to evaluate the effectiveness of each.

2.3.2: SAMPLING MODES INSTALLED IN SYSTEM

Three modes of sampling are included in the software. If the user wishes to add further algorithms then it is a relatively easy task to add new 'mode' programmes (see the system software section).

The three in-built modes are as follows:

Mode 1 16 samples per cycle; 16 cycles sampled

Mode 2 8 samples per cycle; 32 cycles sampled

Mode 3 4 samples per cycle; 64 cycles sampled

In all cases 256 samples are taken per measurement. This is limited by the onboard RAM. The required mode can be selected by the host computer using a simple HPIB command.

CHAPTER 3: SYSTEM DESIGN

3.1: DESIGN AND DEVELOPMENT OF ANALOGUE BOARD

The analogue board has the following tasks to perform:

1) To amplify each input signal as required to ensure that each channel operates at the correct level.
2) To convert each input signal from 50kHz to the IF frequency.
3) To filter the IF signals with the required bandpass characteristics to reject noise present.
4) To sample the filtered IF signals and convert them into a digital form ready for the digital board.
5) To filter and amplify the 50kHz square wave to produce a sinusoidal source signal.

3.1.1: THE INPUT STAGES

The voltage dynamic range of the input signals is only expected to vary by about 20dB in the course of one set of measurements. However, to allow for different front end gain and different coil drive power, some means of input gain control is necessary. Use of a precision programmable amplifier here (eg 4 bit range, 16 bit accuracy) would mean that the gain could be altered for each measurement, increasing the resolution of the system. However, with varying gain, a variation of phase would be expected. This would require tables of calibration data to be stored,

together with the added software and hardware required for 'self calibration'. This would significantly add to the complexity of the system and would require careful layout of the hardware to minimise crosstalk and to ensure the best performance. Note also that the time taken to 'auto-scale' the input would increase the measurement time.

The opposite extreme is to have a variable gain control on the front panel. Provided the device is not used remotely this would be adequate, but the wiring to the front panel would cause considerable crosstalk problems. The solution used here is to have a low precision, software programmable gain control which is set to just prevent overload, and then left for the complete set of measurements. A constant gain and phase error will be incurred but this is manageable. The input signal is buffered and amplified by the first stage and an 8 way electronic switch is used to set the gain of the second stage. The gain is spread between two stages to reduce the gain bandwidth requirement of the op-amps to a level suited to the TL081/2 devices (5MHz).

Eg: For a maximum gain of 200:

<u>For a single stage:</u>

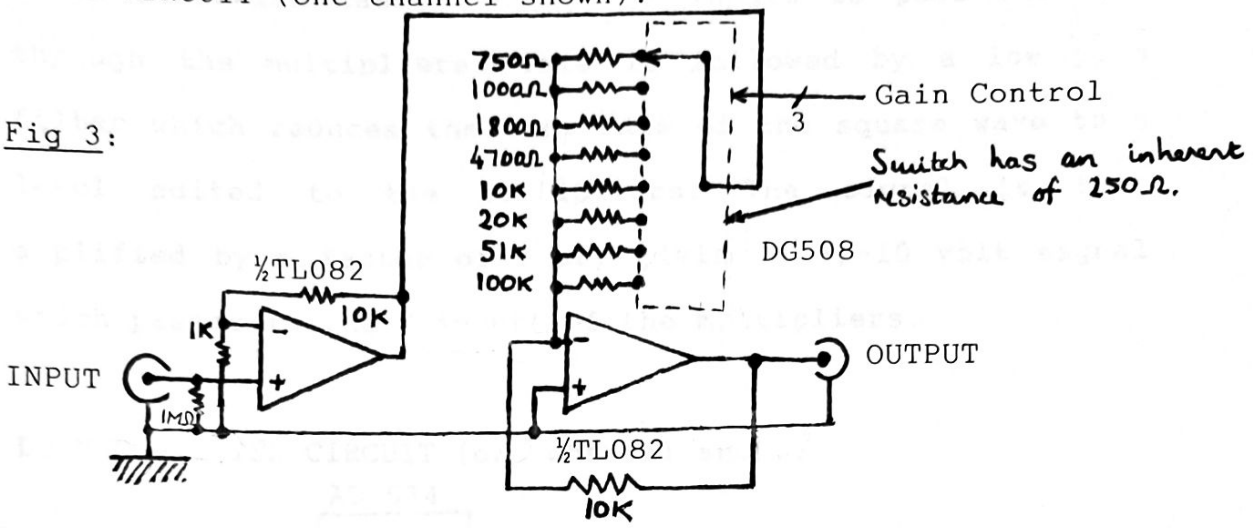
Op-amp gain bandwidth required=200*50kHz=10Mhz

<u>For two stages:</u>

Op-amp gain bandwidth required=$200^{0.5}$*50kHz=0.7Mhz

INPUT CIRCUIT (one channel shown):

Fig 3:



3.1.2: THE DOWN CONVERTERS

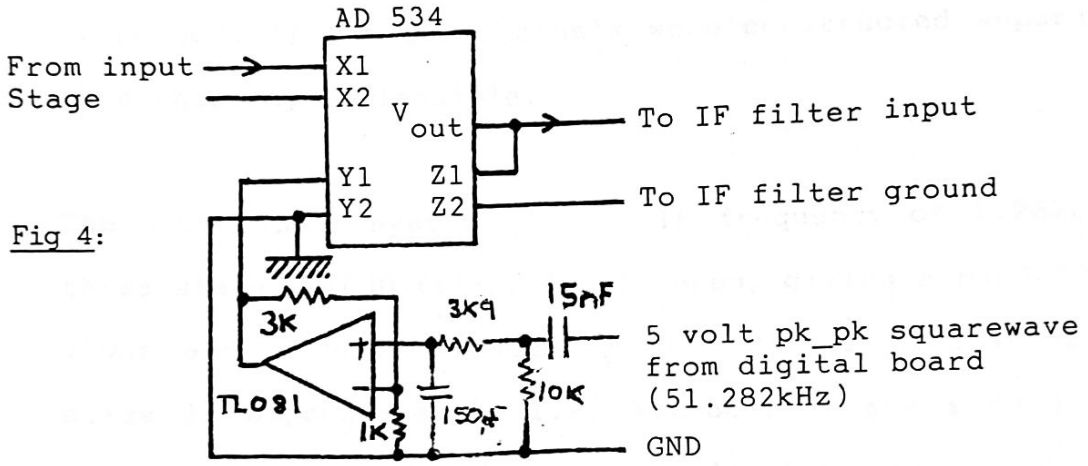
Each down converter receives a signal at 50kHz and converts it to the IF frequency of 1.282kHz. Multiplier chips are available for this purpose, and the one chosen was the AD534J by Analog Devices. This enhibits a maximum total multiplication error of 1.0% over a small temperature range and 1.5% over a large temperature range. The short term non-linearity is quoted at 0.01% which is approxiamately 13.5 bit accuracy. It should be noted that driving the Y input hard with the local oscillator signal, and feeding a small input signal to the X input, gives greater linearity and reduced breakthrough compared to using the inputs the opposite way around. Also, driving the multiplier at near maximum levels increases the stage signal to noise ratio.

The local oscillator signal is derived from CLK1, a signal produced by the digital board. The 5 volt square wave passes through a high pass filter to remove the dc component, which

would otherwise cause the 50 kHz inputs to pass straight through the multipliers. This is followed by a low pass filter which reduces the slew rate of the square wave to a level suited to the multipliers. The signal is then amplified by a factor of four, giving a +/-10 volt signal which passes to the Y inputs of the multipliers.

DOWN CONVERTER CIRCUIT (one channel shown):

Fig 4:



3.1.3: THE IF FILTERS

A multiplier produces the sum and difference frequencies of its input signal frequency spectrums. In our case, one input is a sinewave and the other a slew rate limited square wave. Thus the frequency spectrum of the multiplier output is quite complicated. The IF filter selects the required frequency component of 1.282kHz. The bandwidth of the IF filter directly determines the bandwidth over which the network analyser receives and thus the overall immunity to noise. A narrow IF bandwidth results in higher noise immunity but a longer settling time when the input changes.

One of the aims of the design was to make the circuit as flexible as possible. It would therefore be advantageous if the IF bandwidth and frequency could be made variable. (Normally the frequency would remain fixed, but here the local oscillator frequency is not continuously variable). However this requires electronic switching of components. In view of this, and the co-channel crosstalk that was experienced in the construction of the board, this idea was abandoned. If the two channels were constructed separately, then this may be feasible.

The 50kHz input system uses an IF frequency of 1.282kHz. A three stage FRIEND filter (3) is used, giving a bandwidth of 450Hz and a centre frequency of 1.287kHz. The Q of each stage is approxiamately 1.4. Attenuators are used between each stage to maintain an overall gain of unity. This enables the multiplier devices to operate at full scale and thus the highest signal to noise ratio.

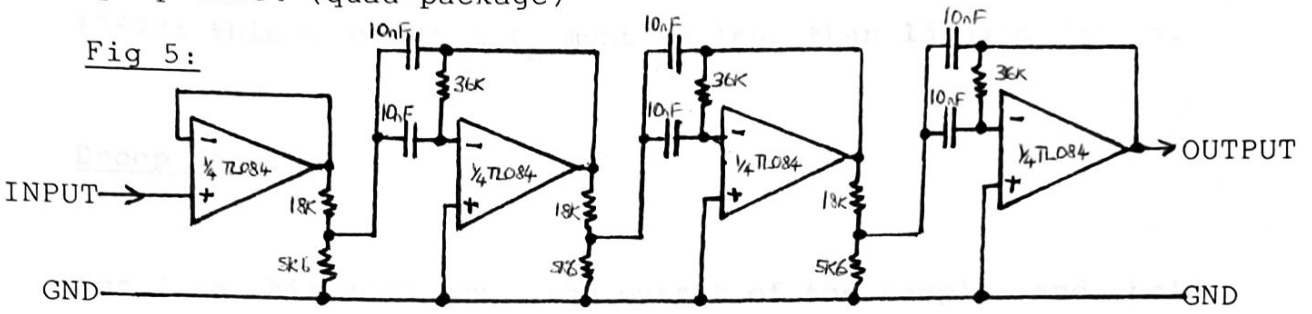
Note that the earth reference for this filter is connected to the Z2 connection of the multipliers, not to the signal input earth. The Z2 input is effectively the ground reference point for the multiplier output. This enables the both the A to D converters and the input circuitry to be earthed directly to the ground plane without creating an earth loop.

The theory for the FRIEND filter is contained in the appendices, together with plots of the frequency response obtained from the practical implementation of the filter.

IF FILTER CIRCUIT (one channel shown):

Opamp=TL084 (quad package)

Fig 5:



3.1.4: SAMPLING AND ANALOGUE TO DIGITAL CONVERSION (5)

The output of the IF filter stage is a sinewave of 1.282kHz. This must be sampled and converted into a form suitable for the digital board. The conventional system of a sample and hold amplifier, followed by an A to D converter was used. The requirements for the sample and hold amplifier were calculated as follows:

Aperture Uncertainty:

Let the output from the IF stage be $y = A\sin wt$

The rate of change of this is $\frac{dy}{dx} = Aw \cos wt$

Therefore the maximum rate of change is Aw volts/sec.

Let t_a be the aperture jitter (max deviation from the correct sampling time.)

For one bit accuracy, t_a must not allow the input to change by more than half a bit. As a worse case estimate, consider the max rate of change of input to be constant over t_a. Then Awt_a must be less than $A/2^N$ where N is the number of bits used for conversion. Therefore t_a must be less than $1/(2*w*2^N)$. For a 12 bit converter, at an IF frequency of 1282Hz this means that t_a must be less than 15 nanoseconds.

Droop Rate:

For one bit accuracy, the output of the sample and hold amplifier must not droop by more than half a bit between samples. The sampling frequency f_s depends on the sampling method employed, but is unlikely to be less than the IF frequency. Thus this is used as a worse case estimate. This gives a maximum time between samples of $1/f_s$ or $780*10^{-6}$ microseconds. For an output in the range 0 to 5 volts, and a twelve bit converter this gives a maximum acceptable droop rate of $(5/(2*2^N*780*10^{-6}))$ or 0.8 volts per second.

The Harris HA5320-5 fulfils both these requirements, with an aperture jitter of 0.3 nanoseconds and a typical droop rate of 0.08 volts per second. The worst case droop figure of 100 volts per second only occurs at extreme temperatures, but this does limit the useful temperature range of the complete unit. If this is important then an external hold capacitor may be added to reduce the droop at the expense of increased

acquisition time. In practice, the minimum sampling frequency is 4*1.282 kHz, which reduces the hold demands of the sample and hold amplifier. Other sample and hold products on the market were not significantly better in any respects and the excellent aperture jitter performance of the HA5320-5 means that it would be suitable for use with a 16 bit A to D converter if required.

The A to D converter used is the AD7572-5. This is a 12 bit converter with a conversion time of 5 microseconds. This enables a maximum sampling rate of 200,000 samples/second provided the sample and hold amplifier and the microprocessor can cope with this. This conversion rate is excessive for this application, but the devices were readily available at the time of construction.

An important point to note, is that the inputs to these devices are very sensitive to overload. They are quoted at being able to withstand inputs of +/- 15 volts, but in the absence of the power rails such levels seem to spell their destruction! This can arise in two ways:

a) The 5 volt rail fails causing the sample and hold stage to remain in hold mode. Then the output represents the arbitary voltage on the internal storage capacitor.

b) The analyser is overloaded.

The first situation is problematic. It is difficult to limit the A to D input voltage without degrading the performance

of the circuit. The protection system used was to diode clamp the A to D inputs to the 0 volt and 5 volt rails, with a 5 volt zener from the 5 volt rail to 0 volts. This prevents damage to the A to D converter, but results in the sample and hold amplifier being overloaded. Attempts to use current limiting resistors spoilt the performance of the sample and hold stage. The sample and hold amplifier can survive short term overloads, but it does mean that it gets hot! The result of this is that after the analyser is overloaded, the device should be allowed to cool (for say 20 seconds) before making further measurements. This problem could be solved by clamping the input of the sample and hold amplifier to +/- 2.5 volts, but this was not done in the prototype.

The A to D converters require a clock at 2MHz. This is derived from the microprocessor 4MHz clock using a D-type flip-flop configured as a divide by two stage.

Sampling is controlled by CLK2, a signal produced on the digital board, which is derived from the same frequency source as the 50kHz and local oscillator signals. On the rising edge of this pulse the D-type flip-flop is set, the sample and hold amplifiers change from sample to hold mode and the A to D converter starts conversion. After approxiamately 6.5 microseconds, the digital board reads in the lower 8 bits from each A to D converter. It then resets

the D type flip-flop which has the effect of telling the A to D converters to send the top 4 bits as well as setting the sample and hold amplifiers to sample mode, ready for the next conversion. The digital board then reads the top 4 bits from each A to D converter.

SAMPLING AND A TO D CIRCUIT (one channel shown):

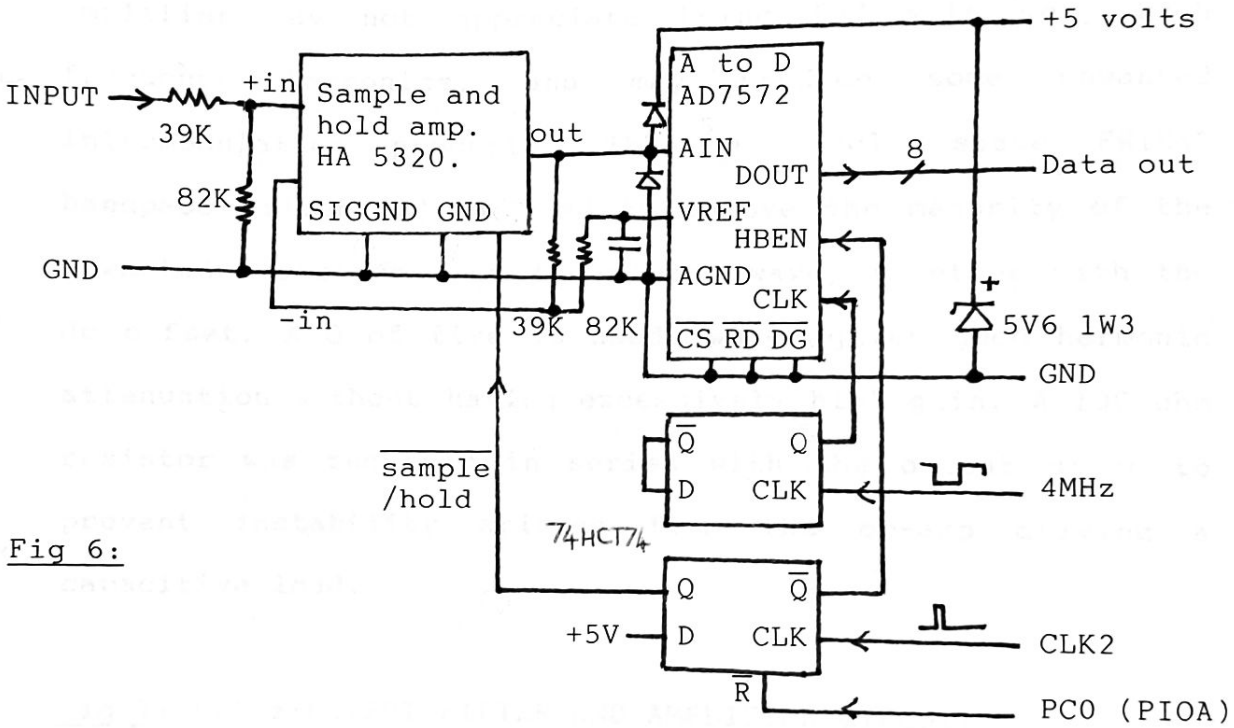


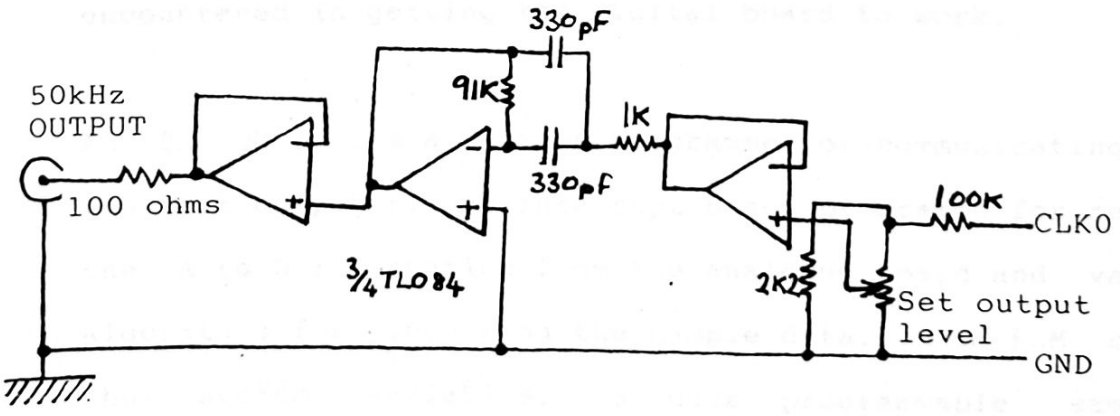
Fig 6:

The resistors associated with the sample and hold amplifier cause the +/- 2.5 volts input to be converted to 0 to 5 volts for the A to D converter. The values are as recommended by the AD7572-5 data sheet. The input attenuator is not strictly necessary for this application as the precise voltage range covered is not important.

3.1.5: OUTPUT FILTER AND AMPLIFIER

This circuit receives CLK0, a 50 kHz 5 V pk_pk square wave from the digital board and converts it into a sine wave. This signal is then buffered and sent to the front panel, where it is passes to the power amplifier. As the analyser is only sensitive to 50kHz, it is not important that the harmonics should be particularly low. However, the power amplifier may not appreciate being fed with very high frequency harmonics, and may produce some unwanted intermodulation products. Thus a single stage FRIEND bandpass filter (3) is used to remove the majority of the unwanted harmonics from the square wave, together with the dc offset. A Q of five is used, which gives good harmonic attenuation without having excessively high gain. A 100 ohm resistor was required in series with the output stage to prevent instability arising from the op-amp driving a capacitive load.

Fig 7: 50kHz OUTPUT FILTER AND AMPLIFIER CIRCUIT:



3.2: DESIGN AND DEVELOPMENT OF DIGITAL BOARD

The digital board has the following tasks to perform:

1) To produce the three clock signals required by the analogue board.
2) To read the A to D converter information and carry out the sampling algorithm on the data.
3) To produce a seven bit control word for the analogue board.
4) To communicate the sampled data to the host computer via an HPIB interface.
5) To produce an external 16 bit latched output for selecting the required electrode and drive coil.

3.2.1: THE MICRO-PROCESSOR

The digital board is based around the Z80C micro-processor running at 4MHz. The Z80 was chosen for reasons of limited design and test time. The device is easy to use, and a low level Z80A based CP/M based machine was available to simulate the Z80C and/or digital board if any problems were encountered in getting the digital board to work.

An 8K ROM stores a monitor programme for communicating with the host computer, an interrupt based programme for reading the A to D information from the analogue board and various algorithms for processing the sample data. A 2K RAM stores the system variables, a user programmable sampling

algorithm, and the sample data. In retrospect, it would have been better to use a larger RAM to enable more sample data to be stored. This would only require a larger IC socket and the additional address lines to be connected. For RAM's below 16K, no further address decoding would be required.

3.2.2: PERIPHERAL DEVICES AND ADDRESSES

The 27C64 ROM exists between addresses 0000h and 1FFFh and a duplicate copy exists between 2000h and 3FFFh due to incomplete address decoding. The 61C16 RAM exists between 4000h and 47FF, and copies exist up to 7FFFh. One eight bit external latch occupies address 8000h, with copies to BFFFh, and another occupies address C000h, with copies to FFFFh. These latches produce the 16 control external control lines required. Two parallel input/output chips (PIO's) are used. The devices used are the 82C55 by Intel, giving 24 I/O lines each. PIOA is associated with the analogue board and occupies port addresses 00h to 03h. PIOB is associated with the HPIB interface and occupies port addresses 40h to 43h. The programmable counter timer chip (CTC) used is the Intel 82C54 and occupies port addresses 80h to 83h.

3.2.3: PRODUCING CLOCK SIGNALS

The three clock signals CLK0, CLK1 and CLK2 are produced by a single programmable divider chip - the Intel 82C54. This contains three 16 bit programmable dividers which are each

fed with the 8MHz clock frequency. CLK0 and CLK1 are both used to produce near sine waves and so are generated as square waves. CLK2 is used to create interrupts and is therefore generated as a very narrow pulse (1 clock cycle or 0.125 microseconds). The chip is entirely programmable by the micro-processor and occupies port addresses 80h to 83h.

3.2.4: READING THE A TO D INFORMATION

The start of conversion occurs at the rising edge of CLK2. This signal passes to the analogue board to control the sample and hold and A to D devices. The signal also passes via an enabling gate, to the Non Maskable Interrupt (NMI) input of the micro-processor, where a rising edge forces an interrupt. The enabling signal is line PC0 of PIOA, and enables the programmer to determine when the interrupt line is to be enabled. The reason for using the Non Maskable Interrupt rather than the Maskable Interrupt which has built-in enabling and disabling, is because the NMI input is edge sensitive, whereas the Maskable Interrupt input is not. Using the Maskable interrupt would avoid the use of the enabling gate but would necessitate an external flip-flop to hold the interrupt line active until the micro-processor had responded.

After a short delay for the A to D conversion (6.5 microseconds) the microprocessor reads the least significant byte from the A to D converters via ports A and B of PIOA.

PC0 of PIOA then goes low, selecting the high byte from the A to D converters and temporarily disabling the interrupts. When the processor has read the high bytes from the A to D converters, PC0 is set high again in time for the next rising edge of CLK2. It should be noted that if CLK2 is too high in frequency, then the NMI routine will call itself causing the stack to run wild and the processor to crash. The minimum time between samples can be determined from the time taken to execute the NMI programme plus the time for the longest instruction that may be being executed when the interrupt occurs. This is because the Z80 always finishes its current instruction before servicing the interrupt.

Lines PC1 to PC6 of PIOA pass to the analogue board to set the gain of the input amplifiers. PC7 is used to gate the CTC divider chip to allow CLK0, CLK1 and CLK2 signals to be switched off if desired.

3.2.5: HPIB INTERFACE

The HPIB is Hewlett Packard's implementation of the IEEE 488 interface bus. In its simplest form, it is an eight bit parallel bus, with three wire handshaking and five system control lines. It is intended for computer communication with instrumentation. All lines must be properly terminated and all protocols correctly handled to avoid crashing the bus or causing reduced reliability. Crashing the bus will prevent the computer accessing any of the devices on the

bus, including disc drives etc. Fortunately, the HPIB controller does have a reset capability, should this occur.

Various custom chips are available to completely implement the HPIB standard. However this approach was not taken, for the following reasons:

1) The Intel chip set: 8291A and 8293 (two off) form a very compact and easy to use interface. However, the 8293 bus driver chips are extraordinarily expensive, and very difficult to come by. At the time of construction, not enough was known about the HPIB to use alternative bus drivers with confidence - there were too many unknowns in the system such as the software at both ends of the communication link.
2) The Texas Instruments and Motorola chips both had indecipherable data sheets, and it was unclear whether they would be easily interfaceable to the Z80C.
3) One alternative solution is a PIO based solution. The extra hardware involved in this solution compared to using a custom chip was minimal, and most of the gates required were available on the board as spares from other parts of the circuit.

The PIO solution was chosen. The disadvantage with this is that the software becomes more complicated. Also, the watchdog nature of the software means that it tends to become less structured and more difficult to alter for

alternative interfaces. For this reason, it may be decided later to implement the custom chip solution instead.

The HPIB handshaking is dealt with in more detail in the software design section. It is sufficient here to define the following signals:

ATN = Attention :Signal from the bus controller to indicate that it is issuing a command.

DAV = Data Available :Signal from the talking device to indicate that it has data available.

NRFD = Not Ready For Data :Signal from the listening device to indicate that it is not ready for data.

NDAC = Not Data Acknowledged :Signal from the listening device to indicate that it has not received data from talker.

EOI = End Or Indentify :If ATN is active then EOI active indicates a parallel poll is being carried out. If ATN is in-active then EOI active indicates that the last character of a sequence is being transmitted.

IFC = Interface Clear :Issued by HPIB controller to reset all connected devices.

ALL LINES ARE ACTIVE LOW - Most are open collector.

The following specification was drawn up for the HPIB circuit design:

1) The interface must terminate the bus correctly, and must not load the bus when the interface is switched off.
2) If the interface power is switched on whilst the bus is being used elsewhere it should not crash the bus by performing incorrect handshaking.
3) When the ATN signal becomes active, the interface must enter listen mode within 200ns. This involves tri-stating the data lines, making NRFD and NDAC active and DAV in-active. The interface must not wait for the micro-processor before doing this.
4) When the IFC line becomes active the micro-processor should be reset. This is in case the micro-processor crashes for any reason.
5) Talk and Listen LED's should fitted for testing and fault finding.

The solution arrived at uses the minimum hardware without compromising the specification above. The connection to the bus uses a pair of chips by National Semiconductor. These chips contain the bus termination circuits (resistors and diodes) together with electronic switches for disconnecting these circuits when power is removed from the chips. They also contain the bus receive and transmit components. A D-

type latch is used to achieve the 200ns response time when the ATN signal becomes active. When the ATN line goes low, the latch is reset, causing the NRFD and NDAC lines to go low. Also the bus driver chips are set to receive mode. In this mode, the DAV driver circuit is switched off, so it is not necessary to set the DAV line inactive on the interface side of the driver chip. The data bus driver is connected to port B of PIOB via resistors. These resistors are included to limit the current between the PIOB outputs and the driver chip outputs when the ATN line goes active whilst the device is talking. As both devices require only a very low input current, the addition of these resistors does not compromise the logic levels and the extra delay introduced is very much less than the settling delay of the bus.

All the handshaking lines pass to PIOB so that the microprocessor can monitor the bus and carry out the handshaking sequence.

NOTE: There are some problems in using the 82C55 PIO chip. When the mode of this chip is changed (eg changing a port from an output to an input) all the outputs go low until reprogrammed. According to the Intel data sheet, when the chip is hardware reset, all the pins are temporarily forced high. This would not be acceptable, since in one or other of these states it is bound to crash the HPIB bus. However in practice the chips only have a very small pull-up current,

and a 10K resistor to ground easily maintains a logic zero during reset. However, due to the outputs being set to logic zero after a mode change, which is required when changing from listen to talk and vice-versa, care must be taken when deciding on which polarity to have each output. For example in this design most outputs are pulled low during reset, but the line to the D-type flip-flop is pulled high. The important thing is to ensure that the reset state and the state during mode change are legal. As with the data lines, resistors are used between the PIOB handshaking line outputs and the bus driver chips. This saves complicated routing logic and completely ensures that outputs can never drive outputs. This used to be a problem with 74LS style logic, since resistors were not adequate with the greater input current required. Often 'dead time' circuits were required to avoid overlap.

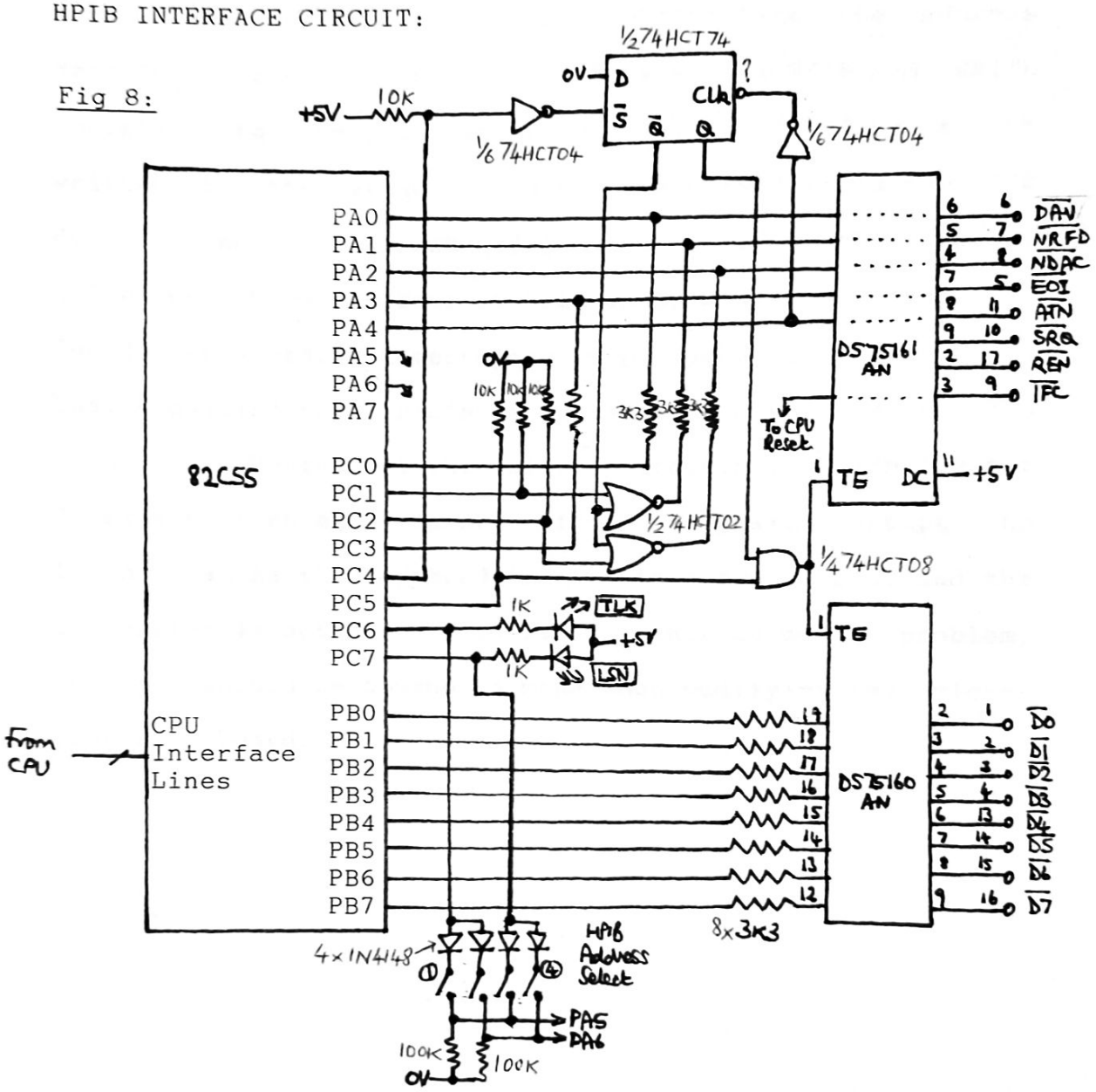
The remainder of the handshaking is carried out in software. It should be noted that when changing the direction of the circuit, careful sequencing of events is very important in order to produce glitch free outputs.

The talk and listen LED's are connected to two PIOB output lines. These outputs are also used to scan the HPIB address select switch matrix. A matrix was used for the address switches as only 3 input lines were left on PIOB and four or five would be required for non-multiplexed switches.

Otherwise, a non-multiplexed solution is neater in both hardware and software.

HPIB INTERFACE CIRCUIT:

Fig 8:



3.2.6: EXTERNAL 16 BIT LATCHED OUTPUT

IT SHOULD BE NOTED that these were a late addition and are not properly connected to the Z80C. Due to a lack of spare gates, the WRITE line is not combined with the address decoding for the latches. Therefore both a READ and WRITE operation to the latch addresses will causes data to be written to the latches. The problem with this is the Z80 dynamic memory refresh cycle also reads the memory addresses. When it does so the contents of the I register (used for vectored interrupts) is placed on the MSB address bus. Providing the I register contains a value less than 7Fh this is OK. However if the value is greater than 7Fh (ie bit 7 is set) then every memory refresh cycle will corrupt the latch data. As the vectored interrupts are not used, and the I register is set to zero on reset, this is not a problem, but it should be bourne in mind when modifying the micro-processor board.

3.3: DESIGN AND DEVELOPMENT OF SOFTWARE

The software has the following tasks to perform:

1) To carry out the HPIB protocol, and to provide high level commands to communicate with the HPIB interface.
2) To set up the ports and programmable divider chips associated with the analogue board.
3) To read in the sample data from the A to D converters in synchronisation with the CLK2 Non-Maskable interrupts.
4) To provide the user of the host computer with a variety of machine code monitor routines to enable debugging of the system (examine memory, change memory etc).
5) To allow data to be sent to the external ports and the programmable gain amplifiers of the analyser input stages.
6) To allow a variety of sampling algorithms to be included so that the merits of each can be evaluated.

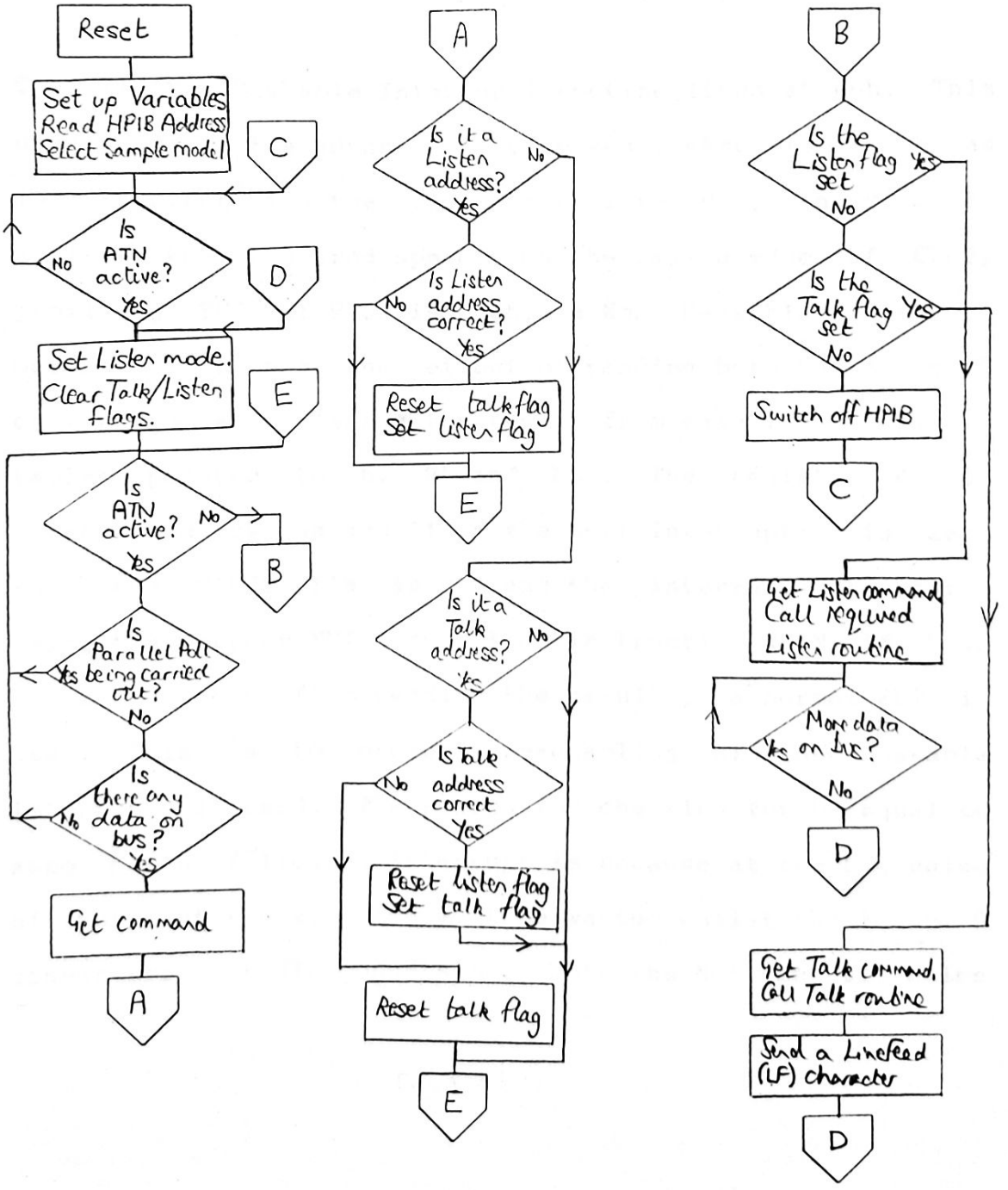
3.3.1: INTRODUCTION TO SOFTWARE

The processor used on the digital board is the Z80C running at 4MHz. In view of the simplicity of the software and the speed requirement, machine code was chosen for the software. An annotated copy of the software is included in the appendices. If the reader is not familiar with the Z80 then he or she will need to acquire a comprehensive description of the instruction set (see for example (4)) before attempting to understand or alter the software. The software

is prepared on another machine, assembled, and then programmed into an 8K EPROM from address 0000h onwards. The Z80 starts execution from 0000h after reset. Note that the Z80 stores all two byte words with the least significant byte first.

3.3.2: SOFTWARE FLOWCHART

Fig 9:



3.3.3: DESCRIPTION OF SOFTWARE

The programme starts with a short jump table giving easy access to the entry points of various important routines. These are not used in this programme, but should be used if external software requires routines from this programme. It is not satisfactory to call the routines at their true address, as they will move when the software is updated. On power up or reset, the CPU starts execution from 0000h.

The NMI (Non Maskable Interrupt) routine lives at 66h. This uses some of the advanced port access instructions such as INI together with the alternate register set, in order to achieve the required speed. On the rising edge of CLK2, providing PC0 of PIOA is high, a Non Maskable Interrupt occurs. This has the effect of reading both the A to D converters and placing the result from each into separate tables pointed to by HL and HL'. The register DE is decremented by one and if on the next interrupt DE is zero then the CARRY flag is set and the interrupt terminates using the correct NMI termination instruction RETN. If DE is not zero, then after reading the results, a normal RET is used. This is to prevent re-enabling of the Maskable interrupts if used. The reason for checking for DE equal to zero on the following interrupt is because at the beginning of the interrupt, time must be wasted whilst the A to D converters do their conversion. Note the NOP used for time

padding. The number of clock cycles (T states) are included after each instruction to assist working out the timing. For a 4MHz clock, each T state lasts 0.25 microseconds. The routine is run from a simple loop which checks for the carry to be set.

RESET sets up the stack and clears various variables stored in memory. It also reads the HPIB address select switches and stores the result in the variable ADDR. Mode 1 sampling is set up before jumping to ATN6 to wait for a command.

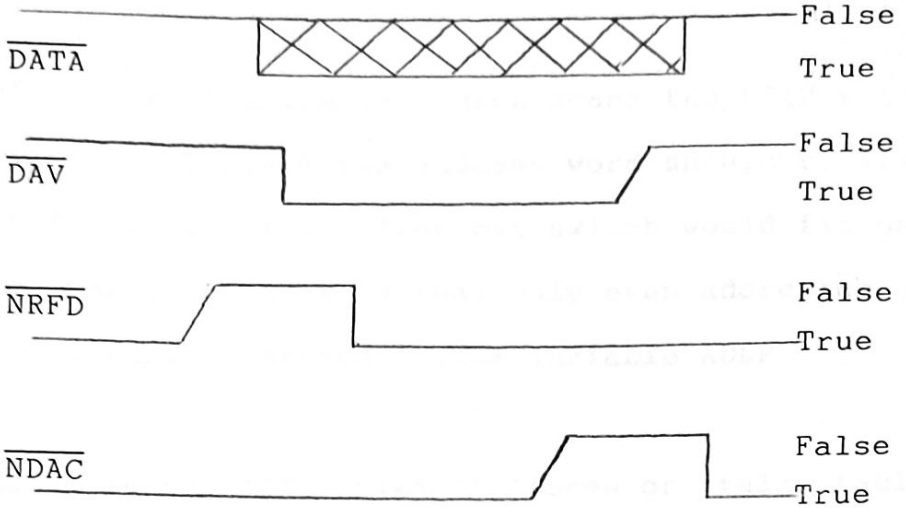
SETLSN and SETTLK are routines to set up the HPIB hardware. Note that due to a quirk in the PIO chip, the SETLSN may only be used (and only ever needs to be used) when ATN is active.

RXBYTE receives a single byte from the HPIB bus into register A. If ATN is or becomes active then receive is aborted and the CARRY flag is set. If EOI (End or Identify) is active during receiving, then the End of Text (EOT) flag is set. A diagram of the HPIB handshaking procedure is shown overleaf. Further details will be found in the appendices.

TXBYTE transmits a single byte from register A onto the HPIB bus, setting EOI active if the End Of Text flag is set. If there is an error, such as no one listening or ATN active, then transmit is aborted and the CARRY flag is set.

TIMING DIAGRAM OF HPIB HANDSHAKING:

Fig 10:



ATN is the procedure which receives all the HPIB command information on the bus. When the ATN line becomes active, the CPU should finish what it is doing and jump to this routine as quickly as possible. Failure to do so would hold up all the HPIB devices on the bus, as all HPIB commands must be received by all devices. The procedure receives a byte from the bus, removes the parity bit (whose presence was not documented in the HPIB manuals!) and checks whether it is the listen or talk address of this device. If it is, then various flags are set. As soon as a non command byte is transmitted the programme checks these flags, and if set calls the relevent listen or talk procedure. If the flags are not set then the interface is placed into idle mode until the next command is sent. This is also the state that the device is in after RESET.

RXREST reads any surplus bytes at the end of a command until the ATN line becomes active.

RDADDR is the routine which scans the HPIB select address switches. Ideally the address word should be five bits wide not four, but only a four bit switch would fit on the board! The result of this is that only even addresses may be set. The address is stored in the variable ADDR.

DOCMD scans the relevent listen or talk table for the received command and if it exists then that routine is executed. Otherwise the interface again becomes idle. Note that JP (HL) jumps to address HL not to the contents of HL. This is a bug in the Z80 assembly language.

GETHEX receives an ASCII hex byte (MSB first) from the bus into register A. Leading spaces are ignored but there must not be a space between each hex digit. Upper or lower case may be used and the CARRY flag is set if there is an error.

SNDHEX converts the contents of A into a two character ASCII hex word and transmits this onto the bus. The CARRY flag is set if there is an error.

After the command tables, there follow the command routines. Those prefixed with L are listen routines and those prefixed with T are talk routines. Many of these are simple machine

code editor facilities. The LS routine reads the sample data from the A to D converters. Note the loop which waits for the CARRY flag to be set by the NMI routine. The TS routine executes the routine pointed to by the variable MODPRG. This is the programme which calculates values from the sample data and sends them to the host computer.

There follow the sample mode programmes. Three modes (1,2,3) are installed in this version of the software. Apart from the parameters at the beginning, these are virtually identical and as space is plentiful, no attempt has been made to share similar routines. This is to achieve 'stand alone' mode modules. For clarity, most of the variables, constants and labels are suffixed with the mode number. The parameters used by each mode are stored at the beginning of that mode for quick reference. The algorithms performed by each of the modes are discussed in Chapter 2. For development purposes, one mode may be stored in RAM starting from 4100h. This is labelled mode 0, and the first eight bytes must conform to the standard used in the ROM based mode routines, as follows:

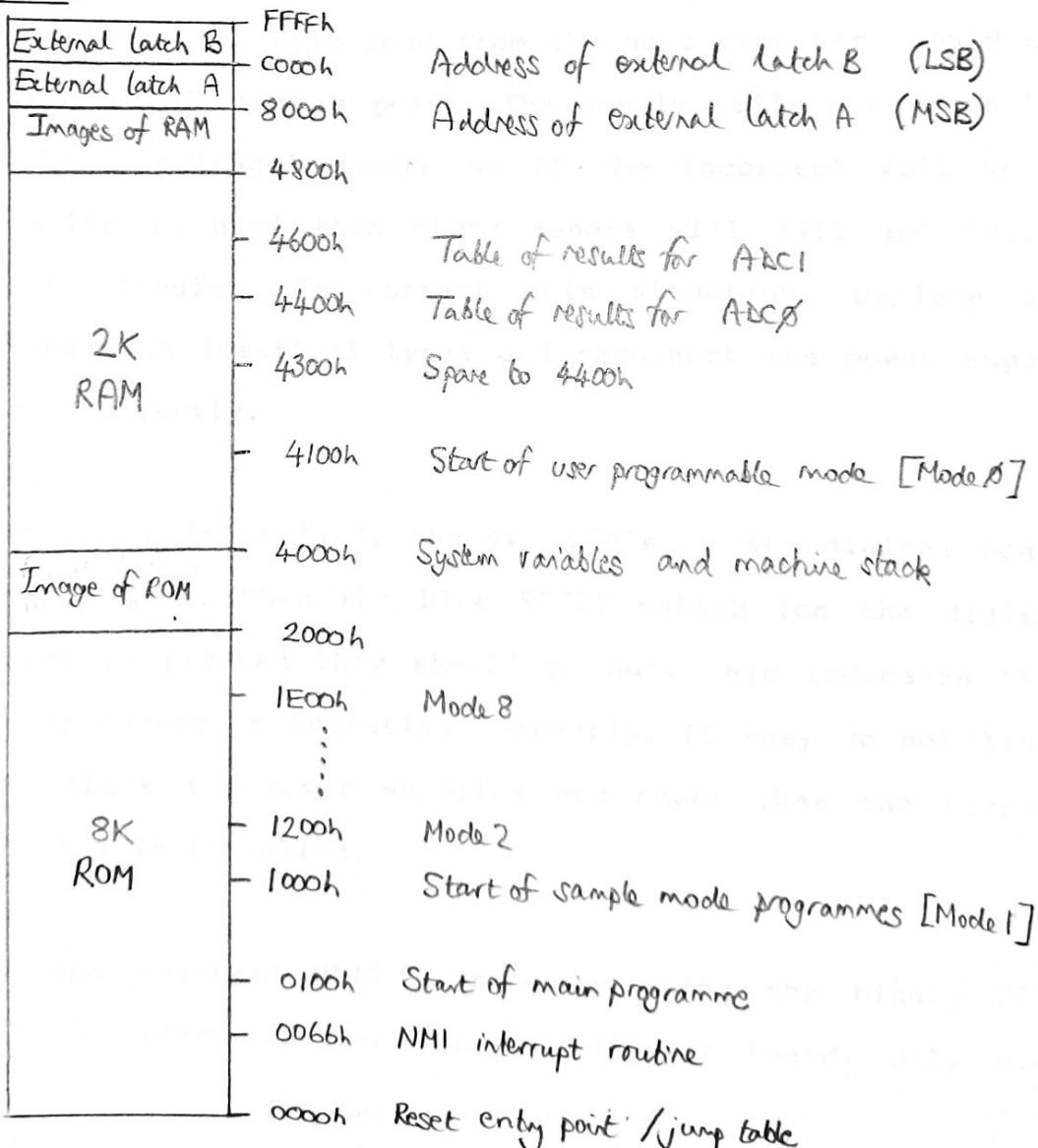
1) The first three bytes must contain an absolute jump to the mode initialisation routine.
2) The next three bytes must contain an absolute jump to the mode algorithm and transmit result routine.
3) The next two bytes must contain the the number of samples required for the sampling routine LS.

The presence of the RAM routine is indicated by a C3h (absolute jump instruction) in address 4100h.

As the host computer did not have a Z80 assembler at the time of design, it was found easier to assemble the modes on a Z80 machine and re-programme an EPROM, rather than down-loading a RAM mode from the host computer.

3.3.4: MEMORY MAP OF SYSTEM

Fig 11:



CHAPTER 4: USING THE NETWORK ANALYSER

4.1: POWERING UP THE UNIT

This section is included to help the first time user get the system 'up and running'.

The +15 volt, -15 volt and +5 volt lines should be connected to suitable power supplies. Allow 0.3 amps for the 15 volt supplies and 0.5 amps for the 5 volt supply. For the purpose of testing, the HPIB lead from the host computer should not be connected at this point. The supply rails are protected by 'over voltage' zeners so if the incorrect voltage or polarity is used then these zeners will fail and become short circuits. To correct this situation, replace the zeners with identical types and reconnect the power supply lines correctly.

When power is applied, the two LED's on the digital board should light. When the blue RESET switch (on the digital board) is pressed they should go out. This indicates that the processor is operating correctly. If they do not light then check the power supplies and check that the correct software is installed.

Set the required HPIB address by using the binary HPIB address select switches on the digital board. Only even

addresses may be selected. Switch one represents 2, and switch four represents 16. Thus even addresses from 0 to 30 may be set. Press the blue RESET switch to set this address.

The HPIB lead to the host computer should now be plugged into the Type 57 socket on the rear of the instrument. The pinout of this connector is shown in the HPIB section of the appendices.

The listen LED lights when the instrument is receiving data from the HPIB bus. This occurs when any bus HPIB commands are being sent to any bus devices, or when particular commands are being sent to this particular device. The talk LED only lights when the instrument has been requested to talk by the host computer. Talk mode may be aborted at any time by sending a new HPIB command to the bus by the HPIB controller.

The instrument may be reset at any time using the IFC (Interface Clear) facility of the HPIB interface. This facility will only be required if the instrument 'crashes' for some reason. The IFC facility has the same effect as pressing the blue RESET switch on the digital board.

4.2: HPIB COMMANDS

The HPIB commands available on this instrument do not conform to the standard HPIB command format. This is because the commands are mainly low level ones not supported by the HPIB standard.

It is important to distinguish between the two types of command. The HPIB commands are sent down the bus by the HPIB controller to configure the devices on the bus to carry out certain operations. This information is characterised by the ATN (attention) line of the bus being active. The instrument commands are sent from the host computer to a device on the bus to control its operation. These commands are characterised by the ATN line being in-active.

To configure a device on the bus to be a listener, the controller sends a single byte equal to the device address plus 32. To configure a device to be a talker the device address plus 64 is sent. It is possible to have several listeners on the bus but only one talker. The HPIB command 'unlisten' (63) removes all current listeners. The HPIB command 'untalk' (95) removes all current talkers.

The EOI (End Or Identify) line is sometimes used to indicate the end of data transfer. This facility is catered for by the instruments HPIB receive byte and transmit byte

routines, but is not used in any of the current command routines. In all cases it is replaced by the linefeed character (LF or ASCII 10).

There are two types of instrument command:

LISTEN COMMANDS: The listen commands are those which instruct the instrument to do something. All listen commands must terminate with the linefeed character. Note that spaces may be placed between different hexadecimal bytes but not between the characters forming one hexadecimal byte.

Listen Commands available:

Mdd	Selects the required sampling mode. 'dd' must be between 00h and 07h.
Waaaa dd dd....	Write to memory from address 'aaaa'. The data to be written follows the address. Any number of bytes may be written to consecutive locations starting at the start address.
Eaaaa	Execute subroutine from address 'aaaa'. This subroutine may be installed using the 'W' command. To return to the HPIB command checking loop use the RET (return) command. If the routine takes a long time then the ATN line should be monitored to ensure

that you are not holding the bus up. The routine is executed the moment the last character of the address is sent.

A dddd — Write data to the auxilliary ports. The first byte is written to the MSB port and the second byte to the LSB port.

G dd dd — Set gain of signal input stages. The first digit of each pair selects the channel and may be a 0 or 1. The second digit selects the gain and may be 0 to 7. 0 is the lowest gain and is default setting on power up.

TALK COMMANDS: The talk commands are those which request the instrument to send information to the host computer. After the command is issued the host computer listens for information from the instrument. The information is terminated either by a LF character or by issuing a new HPIB command. The LF character should be set up as the termination character in the host computer.

Talk commands available:

M — The current sampling mode (00h to 07h) is returned, followed by LF.

Raaaa — Reads the contents of memory starting at address 'aaaa'. Data bytes are available from consecutive memory

locations, starting from 'aaaa' until a new command is issued. No LF character is transmitted.

S When the command is issued the inputs are sampled and the results are put in RAM tables. When the instrument is requested to talk, the sampling algorithm processes the data and sends the results. The standard form for the results is: 'ya_0 yb_0 f_0 ya_1 yb_1 f_1'. The y values are six character ASCII hex numbers. Two numbers are given per channel: ya and yb (see below). The 0 and 1 subscripts indicate which channel the results are for. The flags f_0 and f_1 indicate whether there has been an overflow or not. A 'C' indicates that the data is correct and an 'E' indicates that there has been an error. If there is an error then the input gain for that channel should be reduced immediately to prevent damage to the sample and hold amplifiers.

The relative amplitude and phase results are calculated as follows:

$$\text{Relative Amplitude} = ((ya_0^2+yb_0^2)^{0.5}/(ya_1^2+yb_1^2)^{0.5})$$

$$\text{Relative Phase} = \arctan(yb_0/ya_0) - \arctan(yb_1/ya_1)$$

Note that a four quadrant arctan must be used, giving results between +pi and -pi. The standard arctan functions frequently give two quadrant results.

The following programmes have been written or adapted for use with this instrument. They all reside in the main directory of user 'john' of the Hewlett Packard computer (otherwise known as 'Colin') based in room E133. There is no password to this user name.

hpterm (source programme is hpterm.c)

Programme to talk directly to the instrument. The instrument is expected to be at HPIB address 30. Typing commands followed by 'enter' sends commands to the instrument. Typing 'enter' again requests the instrument to talk and prints the results. Typing 'S' followed by 'enter' initiates a sample. Pressing 'enter' again prints up the relative gain and phase of

the two input channels. Pressing 'enter' a third time prints up the results information as sent by the instrument.

meas (source programme is meas.c)

Programme to take measurements from the impedance imaging apparatus and store the results in a file. The results are stored in a form suitable for the back projection programme below. The external port facility has not been implemented yet and an existing HPIB port is used via the programme 'hpibcont'.

back (source programme is back.p)

Programme constructs conductivity and permittivity maps from the data stored by the 'meas' programme. Note that a phase offset is requested by the programme. This should be equal to the phase shift introduced by the front end multiplexer box at the frequency of interest (50kHz). With the prototype system, this was found to be -53.7 degrees. The 'minus' indicates a delay.

CHAPTER 5: RESULTS

5.1: PRELIMINARY RESULTS

Simple tests were carried out to indicate that the equipment operated as predicted. Part of the analogue board had to be re-built to reduce the crosstalk between the two channels. Even so, with one channel disconnected and set to its highest gain, there is noticeable pickup from the other channel. Considering that the input impedance is 1Mohm, this is not surprising. When driven from a low impedance source the problem is likely to disappear.

Potentiometer configurations were connected between the 50kHz output and each input. With the inputs driven to nearly full scale, the system produced results to better than 11 bits repeatability. As the input level was reduced, the repeatability was reduced. This is as expected for a system based on linear A to D converters. The full scale tests gave amplitude and phase results to within 1% of the results given by the HP3577 commercial network analyser for the same network.

It is not possible to make more accurate results without using rigidly constructed hardware. With an input driven to full scale, 1 bit in 12 corresponds to 0.088 degrees. Simply moving a BNC connector can produce phase shifts of the order of 1 degree!

The simplest way to test the system was to install it into the impedance imaging system and try it! This was the approach taken.

5.2: FRONT END PHASE SHIFT

In order to run the image construction programmes, the phase shift between the electrodes and the input to the network analyser must be known. This is measured as follows:

1) Select the first electrode pair (electrodes one and two).
2) Connect the network analyser output both to the reference input and to the electrode 2. Connect electrode 1 to ground.
3) Connect the output of the multiplexer box to the main input of the network analyser.
4) Increase the gain of each channel of the network analyser until each channel is just below overload.
5) Measure the phase angle, taking care of the sign.

For the prototype system this value was -53.7 degrees at 50kHz and -80 degrees at 48kHz.

Note that the phase shift of the analyser input stage depends on the gain selected. This phase shift varies from about 0.1 degrees for gain 1, to about 13 degrees for gain 7. Due to the difficulty in making these measurements, they

have not been accurately measured, but may need to be accounted for by the user.

5.3: EFFECT OF DIFFERENT SAMPLING MODES

As mentioned in the Chapter 2, there are three sampling modes available.

Mode 1	16 samples per cycle; 16 cycles sampled
Mode 2	8 samples per cycle; 32 cycles sampled
Mode 3	4 samples per cycle; 64 cycles sampled

The 'cycles sampled' refers to the number of cycles of IF signal sampled. The modes were compared by connecting a fixed network to the system and taking 100 samples per mode. The standard deviation of the results from each mode was then calculated.

The standard deviations for each mode were virtually identical, with Mode 2 being consistently marginally better. However when the mode programmes were temporarily altered so that each sampled 16 cycles, then Mode 1 clearly gave better results.

Thus it is largely the number of samples that are taken that determines the repeatability of the results, not the

sampling method used. Mode 1 should be the best mode to use as it takes the highest number of samples per cycle and therefore takes less time to make a measurement. However, sampling for less time means that the synchronous receiver is less immune to signals near the required frequency. On balance, Mode 2 seemed to give the most consistant results in the presence of the sort of noise present in the impedance imaging system.

Note that increasing the number of samples by a factor N, increases the repeatability of the results by the square root of N, at the expense of increased sampling time.

Testing out the performance of the sampling modes properly is difficult as it difficult to know what sort of noise will be present in normal use. Again, comparing the pictorial results given from the image construction programs is probably the best way.

5.4: IMAGES PRODUCED BY SYSTEM

A single perspex rod was placed in the 2D measurement tank and saline solution was poured in until the electrodes were just covered. Conductivity plots produced using both this instrument and a commercial network analyser (HP3577) are shown in Figs 12 and 13. The image construction algorithm was developed by William Pervis.

Fig 12: Synchronous receiver: (450Hz bandwidth)

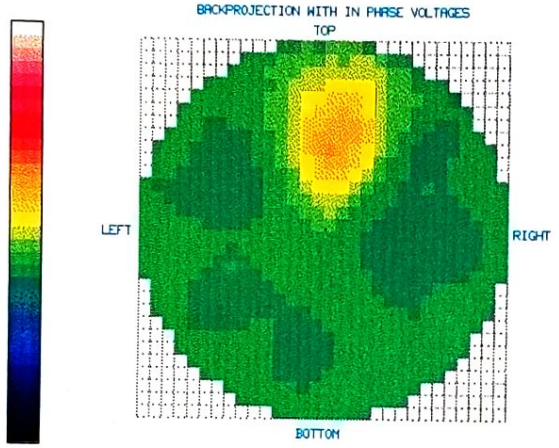
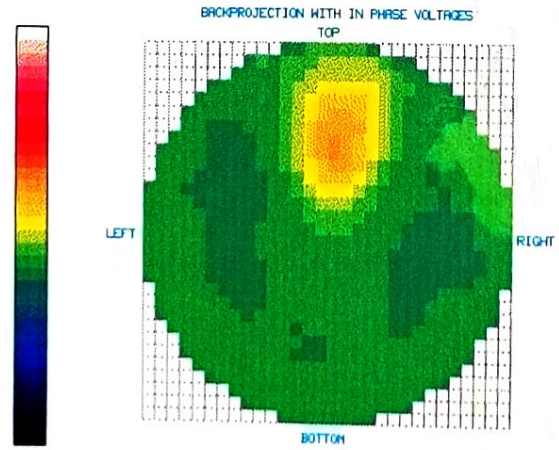


Fig 13: HP3577 Network Analyser: (100Hz bandwidth)



The results are good. In the vicinity of the perspex rod, both images show the same information. This clearly displays that the synchronous receiver based network analyser is correctly interfaced to the computer and the back projection software. The dark green patches around the edge of the images are an 'artifact' of the reconstruction process (2). The image produced in conjunction with the HP3577 contains slightly less 'artifact'.

This again points to the fact that the resolution of the system falls as the input level falls. The signals associated with the perspex rod are high level and so are measured accurately. The signals associated with rest of the tank are low level and so are measured relatively inaccurately.

Taking ten measurements per electrode pair and averaging the results reduced the artifact noticeably.

CHAPTER 6:

CONCLUSIONS AND SUGGESTIONS FOR SYSTEM IMPROVEMENT.

This prototype has served to show that the techniques used are an accurate and cost effective way to produce a system which replaces an expensive and bulky piece of commercial equipment. Images of a quality comparable to those using data from an HP3577 have been demonstrated using the equiptment and the component cost of the equipment is about £250, compared to £21,000 for the HP3577. No 'system flaws' have become evident and improving small sections of the circuit would produce an excellent piece of equipment. Due to its simplicity, the system could be built very compactly. A 16 channel network analyser could be constructed, enabling real time images to be produced (subject to the speed of the back projection algorithm).

The low level resolution could easily be improved by replacing the 12 bit A to D converters with 16 bit types. With the advent of digital audio equipment it is now possible to buy chips with two sample and hold gates and two 16 bit A to D converters in one package! However, in order to reap the full benefits from doing this, it may be necessary to reduce noise in the system and/or reduce crosstalk. The only successful way to reduce crosstalk is likely to be splitting the analogue board into three boards - putting each input channel on separate boards and putting the output circuitry on a separate board.

Research was not carried out into the effect of changing the bandwidth of the IF filter. Reducing the bandwidth would give better immunity to noise but would require a longer settling time.

REFERENCES:

1) Thomas H. Newton M.D. and D. Gordons Potts M.D., 'Radiology of the Skull and Brain. Technical Aspects of Computed Tomography', Vol 5, the C.V. Mosky Company.

2) Joanne Scaife - private communication.

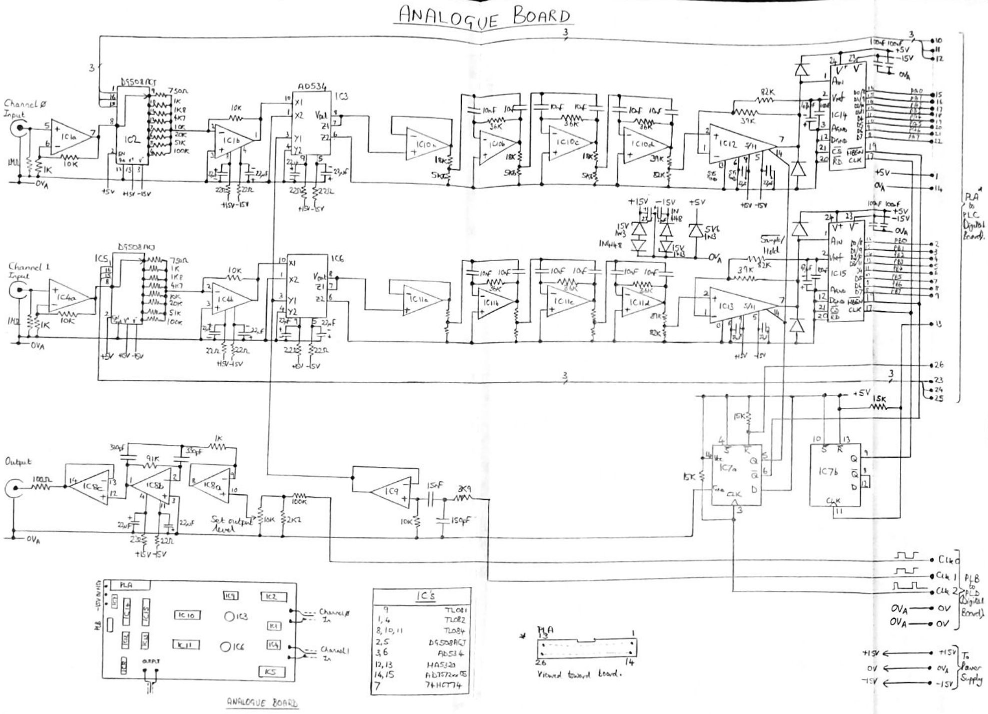
3) Me Van Valkenburg, 'Analogue Filter Design', Holt, Rineholt and Winston.

4) William Barden Jr., 'The Z80 microcomputer handbook', SAMS.

5) Analog Devices, 'Linear Design Seminar'.

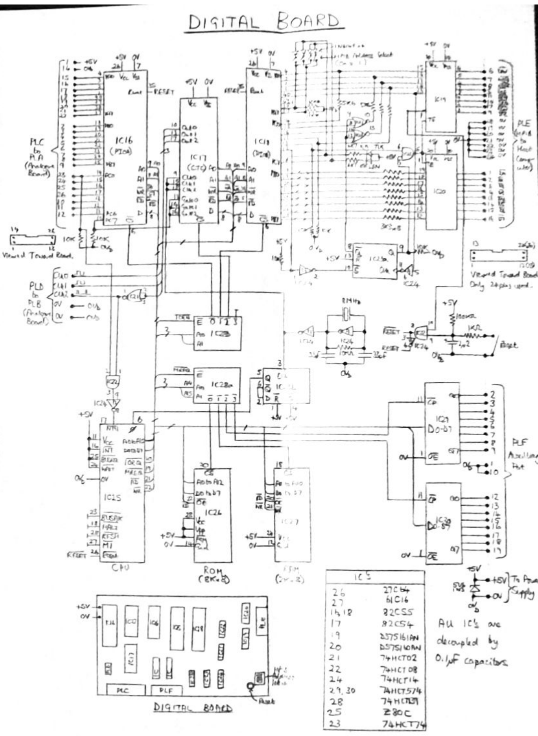
6) Hewlett Packard, 'Tutorial description of the HPIB'. Also 'Device I/O and User Interfacing. HP-UX Concepts and Tutorials.'

ANALOGUE BOARD



IC's	
9	TL081
1, 4	TL082
8, 10, 11	TL084
2, 5	DG508ACJ
3, 6	AD534
12, 13	HA5320
14, 15	AD7572 05
7	74HCT74

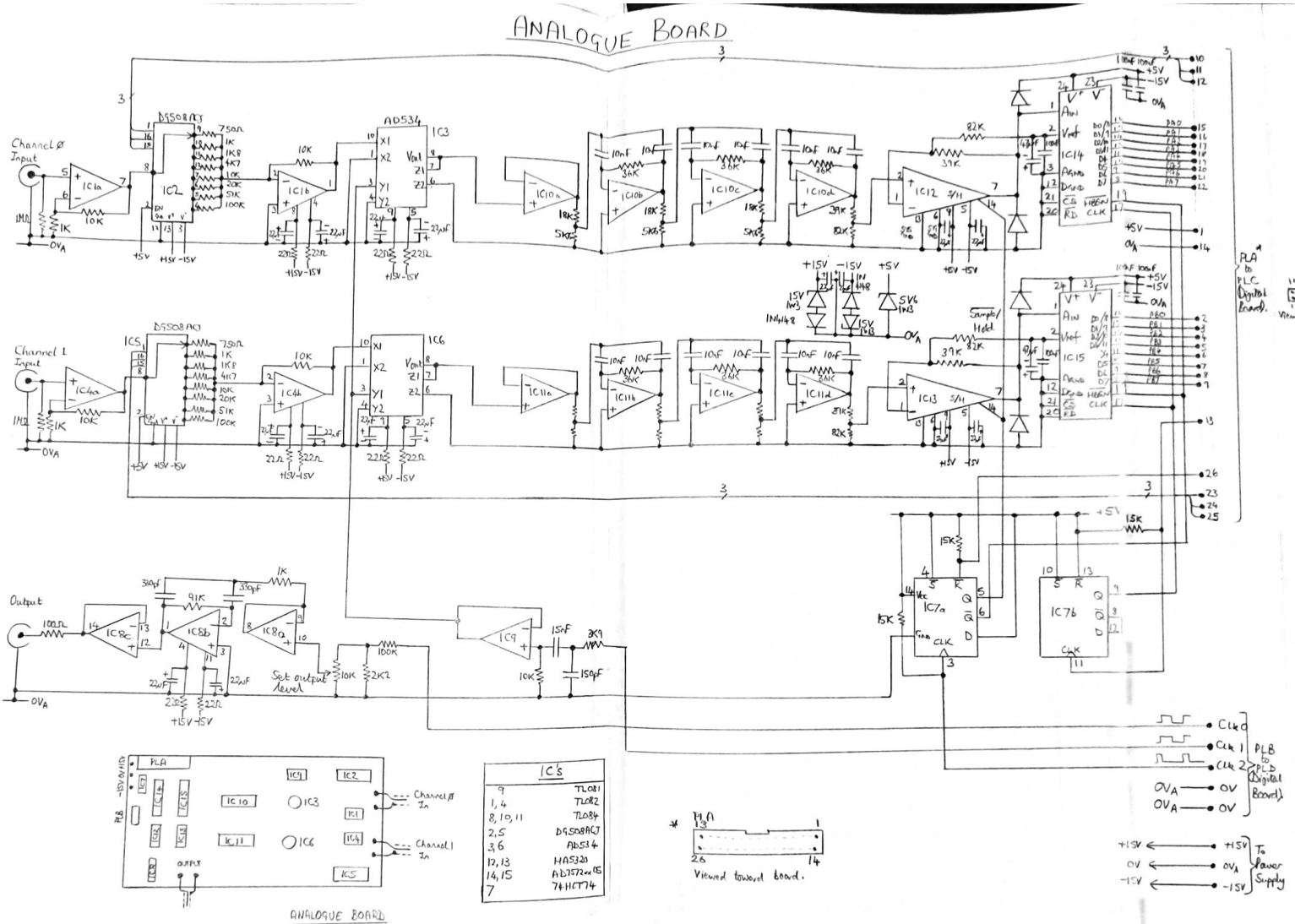
DIGITAL BOARD



IC's	
26	27C64
27	61C16
16, 18	82C55
17	82C54
19	DS75161AN
20	DS75160AN
21	74HCT02
22	74HCT08
24	74HCT14
29, 30	74HCT574
28	74HCT139
25	Z80C
23	74HCT74

All IC's are decoupled by 0.1µF capacitors.

ANALOGUE BOARD

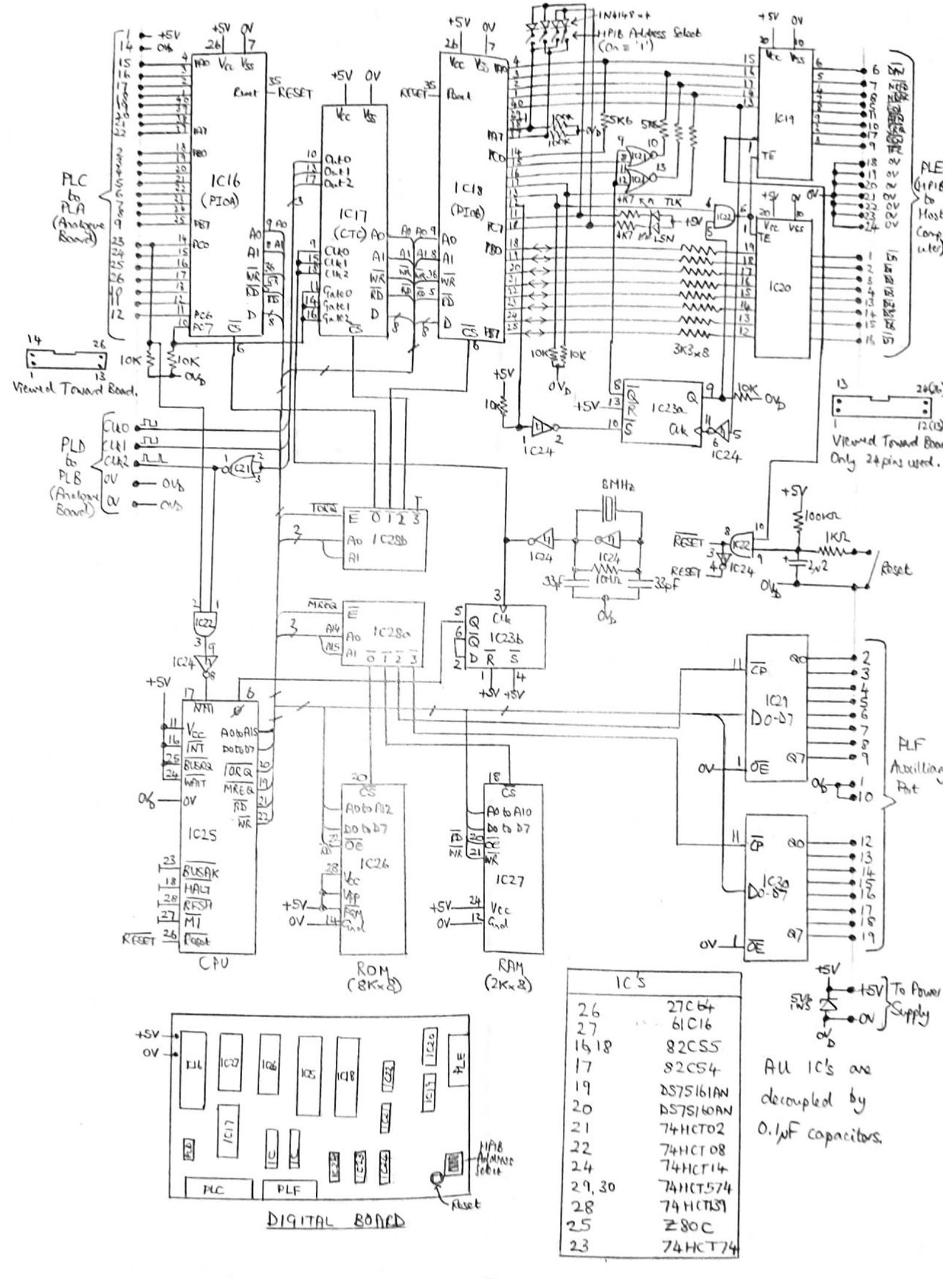


ANALOGUE BOARD

IC's	
9	TL081
1, 4	TL082
8, 10, 11	TL084
2, 5	DG508ACJ
3, 6	AD534
12, 13	HA5320
14, 15	AD7572xx05
7	74HCT74

Viewed toward board.

DIGITAL BOARD



IC's	
26	27C64
27	61C16
16, 18	82C55
17	82C54
19	DS75161AN
20	DS75160AN
21	74HCT02
22	74HCT08
24	74HCT14
29, 30	74HCT574
28	74HCT139
25	Z80C
23	74HCT74

All IC's are decoupled by 0.1μF capacitors.

DIGITAL BOARD

APPENDIX 2: Z80 SOFTWARE LISTING FOR DIGITAL BOARD

```
;software for Z80 based signal processing board
;ROM=0000h to 1FFFh  RAM=4000h to 47FFh
;PIOA communicates with analogue circuitry
;PIOB forms the basis of the HPIB interface
;NB if first byte of a mode block doesn't equal C3h then the mode
;is assumed not to exist

ramst   equ     04000h          ;start address of ram
ramlen  equ     0800h           ;length of ram
stkst   equ     4100h           ;stack pointer in ram
lindly  equ     0020h           ;settling delay time for hpib line

Idata0  equ     04400h          ;Initial value for data 0
Idata1  equ     04600h          ;Initial value for data 1

PIOA    equ     0h              ;port address of ADC PIO
PIOB    equ     040h            ;port address of HPIB PIO
AUXA    equ     08000h          ;memory address of auxilliary latch A
AUXB    equ     0C000h          ;memory address of auxilliary latch B
CTC     equ     080h            ;port address of 82C54 CTC

ABinCo  equ     092h            ;set ports A+B to input C to output
AinBCo  equ     090h            ;set ports A to input B+C to output

eot     equ     ramst+0         ;end of text flag                1 byte
curcmd  equ     ramst+1         ;current command store           1 byte
addr    equ     ramst+2         ;device hpib address             1 byte
cmdhex  equ     ramst+3         ;hex store for commands          2 bytes
numsmp  equ     ramst+5         ;number of samples to be taken   2 bytes
modprg  equ     ramst+7         ;current mode execution address  2 bytes
lsnsel  equ     ramst+9         ;1 if listen selected            1 byte
tlksel  equ     ramst+10        ;1 if talk selected              1 byte
modsel  equ     ramst+11        ;number of mode selected         1 byte
data0   equ     ramst+12        ;address of data from ADC 0      2 bytes
data1   equ     ramst+14        ;address of data from ADC 1      2 bytes
modmem  equ     ramst+16        ;scratchpad for mode routines    40 bytes

mode0   equ     (ramst/0100h)+1 ;MSB of user definable mode
modes   equ     01000h          ;start of rom based modes
nomods  equ     8               ;number of modes in rom

        org     0h

        jp      reset           ;reset device
        jp      setlsn          ;set listen mode
        jp      settlk          ;set talk mode
        jp      rxbyte          ;read ascii byte a from hpib
        jp      txbyte          ;send ascii byte a to hpib
        jp      gethex          ;get hex byte from hpib into a
        jp      sndhex          ;send hex byte in a to hpib
        jp      ATN             ;ATN entry address
        jp      NMI             ;non-maskable interrupt start address
```

```
        org     066h

NMI:    ld      a,d                     ;4T states
        or      e                       ;4T states
        jr      nz,doNMI                ;12T states (when jump exectuted)
        out     (PIOA+3),a              ;disable interrupts/conversions
        scf
        retn

doNMI:  nop                             ;4T states (time padding)
        dec     de                      ;6T states
        ini                             ;16T
        exx                             ;4T
        ini                             ;16T
        xor     a                       ;4T
        out     (PIOA+3),a      ;get high byte from ADC's       ;12T
        ini                             ;16T
        exx                             ;4T
        ini                             ;16T
        ld      a,1                     ;4T
        out     (PIOA+3),a              ;12T
        ret                             ;10T

        org     0100h           ;start of initialisation programme

reset:  ld      sp,stkst        ;set stack pointer to RAM

        ld      a,ABinCo        ;set ports A+B to input, C to output
        out     (PIOA+3),a
        out     (PIOB+3),a

        ld      h,mode0         ;MSB of user definable mode
        ld      l,0             ;hl=address of mode select byte
        ld      (hl),0h         ;set user def mode as not present

        ld      hl,Idata0       ;starting value for data0
        ld      (data0),hl
        ld      hl,Idata1       ;starting value for data1
        ld      (data1),hl

        call    rdaddr          ;store device address

        xor     a               ;(a=0)
        ld      (curcmd),a      ;no current command

        ld      a,0FFh
        ld      (modsel),a      ;mode not selected
        ld      (AUXA),a        ;set external port AUXA to FFh
        ld      (AUXB),a        ;set external port AUXB to FFh

        ld      a,1
        call    setmod          ;set ADC mode to 1 (default mode)
        jp      noATN6          ;wait for ATN active
```

```
;The following are the high level HPIB interface routines

setlsn: ld      a,ABinCo
        out     (PIOB+3),a      ;set ports A+B to input, C to output
        ld      a,001000110b
        out     (PIOB+2),a      ;set listen mode
        ret
        ;NB there is a period of a few micro seconds when both NRFD and
        ;NDAC lines go high. Any talker will assume that there is no listener
        ;during this time. Thus this routine should only be used when ATN has
        ;just been activated by the host and the software programmable hand-
        ;shaking lines are disabled.

settlk: ld      a,AinBCo
        out     (PIOB+3),a      ;set port A to input, B+C to output
        ld      a,010001001b
        out     (PIOB+2),a      ;set up talk handshaking
        ld      a,01011b
        out     (PIOB+3),a      ;switch hpib buffers to talk
        ret
        ;NB the handshaking lines are set before the hpib buffers are set to
        ;talk mode. This is to avoid the possibility of the hpib buffers being
        ;activated before the handshaking lines have settled (ie DAV may pulse
        ;low momentrally causing errors).

rxbyte: push    bc
        ld      a,010b
        out     (PIOB+3),a      ;set NRFD high

wtdav1: in      a,(PIOB+0)
        bit     4,a
        jr      z,ATactl        ;ATN active
        bit     0,a
        jr      nz,wtdav1       ;wait until DAV low

        ld      a,011b
        out     (PIOB+3),a      ;set NRFD low

        in      a,(PIOB+1)      ;read data byte
        cpl                     ;un-invert data
        ld      b,a             ;store data

        in      a,(PIOB+0)
        bit     3,a             ;check EOI line
        ld      a,0             ;reset eot flag
        jr      nz,noEOI        ;EOI high
        ld      a,1             ;set eot flag if EOI active
noEOI:  ld      (eot),a

        ld      a,0100b
        out     (PIOB+3),a      ;set NDAC high

wtdav2: in      a,(PIOB+0)
        bit     4,a
        jr      z,ATactl        ;return if ATN active
```

```
        bit     0,a
        jr      z,wtdav2        ;wait until DAV goes high

        ld      a,0101b
        out     (PIOB+3),a      ;set NDAC low
        ld      a,b
        pop     bc
        or      a
        ret

ATactl: pop     bc
        scf                     ;carry flag set if ATN active
        ret

txbyte: push    af
        in      a,(PIOB+0)
        bit     1,a
        jr      z,NRFDlo
        bit     2,a
        jr      nz,nolsns       ;no listeners (NRFD=NDAC=1)
NRFDlo: pop     af              ;get a off stack

        push    af
        cpl                     ;invert a
        out     (PIOB+1),a      ;output data to data port

        ld      a,(eot)
        and     a
        jr      z,noteot
stEOIa: ld      a,0110b         ;set EOI active if end of text flag set
        out     (PIOB+3),a

noteot: push    bc
        ld      bc,lindly       ;line delay
waitl:  dec     bc
        ld      a,b
        or      c
        jr      nz,waitl        ;wait for lines to settle
        pop     bc

wtnrdl: in      a,(PIOB+0)
        bit     4,a
        jr      z,ATact2        ;return if ATN active
        bit     1,a
        jr      z,wtnrdl        ;wait for NRFD to go high

        ld      a,00b
        out     (PIOB+3),a      ;set DAV low

wtndcl: in      a,(PIOB+0)
        bit     4,a             ;return if ATN active
        jr      z,ATact2
        bit     2,a
        jr      z,wtndcl        ;wait for NDAC high
```

```
        ld      a,01b
        out     (PIOB+3),a      ;set DAV high
        pop     af
        or      a               ;clear carry flag
        ret
nolsns:
ATact2: pop     af
        scf
        ret

;Start here when ATN line becomes active

ATN:    ld      sp,stkst
        call    setlsn
        ld      a,01001b
        out     (PIOB+3),a      ;clear ATN flipflop
        ld      a,01000b
        out     (PIOB+3),a      ;enable ATN flipflop
        xor     a
        ld      (lsnsel),a      ;clear listen select flag
        ld      (tlksel),a      ;clear talk select flag
        ld      (eot),a         ;clear end of text flag

nxtcom: ld      a,010b
        out     (PIOB+3),a      ;set NRFD high

wtdorc: in      a,(PIOB+0)
        bit     4,a
        jr      nz,noATN5       ;ATN line not active
        bit     3,a             ;check EOI line
        jr      z,wtdorc        ;parallel poll being carried out
        bit     0,a
        jr      nz,wtdorc       ;wait for data or command (DAV low)

        ld      a,011b
        out     (PIOB+3),a      ;set NRFD low

        in      a,(PIOB+1)      ;read data from lines
        cpl                     ;un-invert data
        and     07Fh            ;strip off parity bit
        ld      b,a             ;store received byte

        ld      a,0100b
        out     (PIOB+3),a      ;set NDAC high

wtdav3: in      a,(PIOB+0)
        bit     0,a
        jr      z,wtdav3        ;wait until DAV goes high

        ld      a,0101b
        out     (PIOB+3),a      ;set NDAC low
```

```
        ld      a,b             ;get received byte
        bit     5,a             ;see if listen address
        jr      z,notlsn        ;not listen address
        bit     6,a             ;see if not address
        jr      nz,nxtcom       ;not an address

        and     01Fh            ;process listen address
        ld      b,a
        ld      a,(addr)        ;get base address
        cp      b
        jr      nz,wrLadd       ;wrong listen address
        xor     a               ;(a=0)
        ld      (tlksel),a      ;cancel talk select
        inc     a               ;(a=1)
        ld      (lsnsel),a      ;select listen
wrLadd: jr      nxtcom          ;NB more than one listener allowed

notlsn: bit     6,a             ;see if talk address
        jr      z,nxtcom        ;not an address
        and     01Fh            ;process talk address
        ld      b,a
        ld      a,(addr)
        cp      b
        ld      a,0
        jr      nz,wrTadd       ;wrong talk address
        ld      (lsnsel),a      ;cancel listen select
        inc     a               ;select talk
wrTadd: ld      (tlksel),a      ;only one talker allowed
        jr      nxtcom

noATN5: ld      a,(lsnsel)
        and     a
        jr      nz,listen
        ld      a,(tlksel)
        and     a
        jr      nz,talk

        ld      a,011000000b
        out     (PIOB+2),a      ;switch handshaking off

noATN6: in      a,(PIOB+0)
        bit     4,a
        jr      nz,noATN6       ;wait for ATN active
        jp      ATN             ;this could be modified to check that bus
                                ;is idle before entering handshaking protocol
rxrest: call    rxbyte          ;receive any remaining bytes
        jr      nc,rxrest
        jp    * ATN

listen: call    rxbyte          ;NB already in listen mode
        jp      c,ATN
        ld      (curcmd),a
        ld      hl,Lcdtbl       ;get listen command table address
        call    docmd           ;do command
        jr      rxrest
```

* This should have been ATN6! Fortunately this never causes problems.

```
talk:   call    settlk          ;set hpib hardware to talk mode
        ld      a,(curcmd)      ;get current command
        ld      hl,Tcdtbl       ;get talk command table address
        call    docmd           ;do command
        jp      c,ATN
        ld      a,0Ah           ;send LF character-even if no command executed
        call    txbyte
        jp      ATN*
                                    See previous page
rdaddr: in      a,(PIOB+2)      ;read current port data
        ld      c,a             ;store this
        and     03Fh            ;TLK and LSN led's on
        or      040h            ;TLK led off
        out     (PIOB+2),a
        in      a,(PIOB+0)
        rlca
        rlca
        rlca
        and     00000110b
        ld      b,a
        ld      a,c
        and     03Fh            ;TLK and LSN led's on
        or      080h            ;LSN led off
        out     (PIOB+2),a
        in      a,(PIOB+0)
        rrca
        rrca
        rrca
        and     00011000b
        add     a,b
        ld      (addr),a
        ld      a,c
        out     (PIOB+2),a      ;restore port value
        ret

docmd:  ld      b,a
trynxt: ld      a,(hl)
        and     a
        ret     z
        cp      b               ;see if correct command
        jr      z,found
        inc     hl
        inc     hl
        inc     hl
        jr      trynxt

found:  inc     hl
        ld      a,(hl)
        ld      e,a
        inc     hl
        ld      a,(hl)
        ld      d,a
        ex      de,hl
        jp      (hl)            ;execute required command
```

```
gethex: call    rxbyte
        ret     c               ;return if ATN active
        cp      020h            ;see if space
        jr      z,gethex        ;if so get next byte
        call    chkhex
        ret     c               ;not hex digit
        rlca
        rlca
        rlca
        rlca
        ld      b,a
        call    rxbyte
        ret     c
        call    chkhex
        ret     c               ;not hex digit
        add     a,b
        or      a               ;clear carry flag
        ret

chkhex: cp      030h            ;check that it is hexadecimal
        ret     c               ;a=0h to 30h
        cp      03Ah
        jr      nc,hex1
        sub     030h
        or      a
        ret                     ;a=30h to 39h
hex1:   res     5,a             ;convert to upper case
        cp      041h
        ret     c               ;a=3Ah to 40h
        cp      047h
        ccf
        ret     c               ;a=47h to FFh
        sub     037h
        or      a
        ret                     ;a=41h to 46h

sndhex: push    bc
        ld      b,a
        rlca
        rlca
        rlca
        rlca
        call    hex2
        ld      a,b
        pop     bc
        ret     c               ;return if ATN active
hex2:   and     0Fh
        cp      0Ah
        jr      c,hex3
        add     a,07h
hex3:   add     a,030h
        call    txbyte
        ret
```

```
;This programme sets th required sampling mode

setmod: ld      d,a             ;store selected mode
        and     a               ;a=required mode
        jr      nz,modnz        ;not mode zero
        ld      h,mode0         ;MSB of user definable mode
        jr      modz            ;mode zero

modnz:  dec     a
        cp      nomods          ;compare with number of modes
        ret     nc              ;mode badly selected
        add     a,a             ;modes placed at 0200h intervals
        add     a,modes/0100h   ;MSB of built in modes
        ld      h,a

modz:   ld      l,0             ;hl=address of mode
        ld      a,(hl)          ;get first byte of mode
        cp      0C3h            ;see if first instruction is a jump
        ret     nz              ;return if mode not present

        ld      a,d
        ld      (modsel),a      ;mode selected

        push    hl
        ld      de,3
        add     hl,de
        ld      (modprg),hl

        add     hl,de
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ld      (numsmp),bc     ;number of samples

        pop     hl
        jp      (hl)            ;execute port initialisation routine

;These are the HPIB command tables

Lcdtbl: db      'M'     ;mode select
        dw      LMcom
        db      'W'     ;write hex
        dw      LWcom
        db      'R'     ;read hex
        dw      LRcom
        db      'E'     ;execute
        dw      LEcom
        db      'S'     ;start sample
        dw      LScom
        db      'A'     ;set auxilliary outputs
        dw      LAcom
        db      'G'
        dw      LGcom
        db      0       ;end of table
```

```
Tcdtbl: db      'M'     ;check mode selected
        dw      TMcom
        db      'R'     ;read hex
        dw      TRcom
        db      'S'     ;sample network
        dw      TScom
        db      0       ;end of table

LMcom:  call    gethex
        ret     c
        call    setmod          ;set up required mode
        ret

LWcom:  call    gethex
        ret     c
        ld      h,a
        call    gethex
        ret     c
        ld      l,a

nxbytl: call    gethex
        ret     c               ;return if not hex or ATN active
        ld      (hl),a
        inc     hl
        jr      nxbytl

LRcom:  ld      a,(curcmd)
        ld      c,a             ;store current command
        xor     a
        ld      (curcmd),a      ;clear current command
        call    gethex
        ret     c
        ld      h,a
        call    gethex
        ret     c
        ld      l,a
        ld      (cmdhex),hl
        ld      a,c
        ld      (curcmd),a      ;restore current command if syntax correct
        ret

LEcom:  call    gethex
        ret     c
        ld      h,a
        call    gethex
        ret     c
        ld      l,a
        jp      (hl)            ;execute from address hl

LScom:  ld      hl,(data1)      ;address to store data from ADC 1
        ld      c,1             ;port address of ADC 1
        exx
        ld      hl,(data0)      ;address to store data from ADC 0
        ld      c,0             ;port address of ADC 0
```

```
        ld      de,(numsmp)     ;number of samples

        and     a               ;clear carry flag
        ld      a,01b           ;enable NMI and ADC's
        out     (PIOA+3),a

nxtINT: jr      nc,nxtINT       ;wait until all data read (carry set)

        ret

LAcom:  call    gethex
        ret     c
        ld      c,a
        call    gethex
        ret     c
        ld      (AUXB),a        ;send data to latch
        ld      a,c
        ld      (AUXA),a        ;send data to latch
        ret

LGcom:  call    gethex
        ret     c
        ld      c,a
        and     0F0h            ;mask off upper nybble
        cp      020h
        ret     nc              ;only gain of channels 0 and 1 allowed

        and     a
        ld      b,1             ;shift left factor
        ld      d,011110001b    ;port mask
        jr      z,gain0
        ld      b,4
        ld      d,010001111b    ;port mask
gain0:  ld      a,c
        and     07h             ;get 3 bit gain value
sftlft: rlca
        djnz    sftlft
        ld      b,a

        in      a,(PIOA+2)
        and     d               ;mask off counter control bits
        or      b               ;insert new information
        out     (PIOA+2),a

        jr      LGcom

TMcom:  ld      a,(modsel)
        call    sndhex
        ret     c
        ld      a,10
        call    txbyte
        ret

TRcom:  ld      hl,(cmdhex)
        ld      b,16            ;number of bytes to send
```

```
nxbyt2: ld      a,(hl)
        inc     hl
        ld      (cmdhex),hl     ;update stored hex
        call    sndhex
        ret     c               ;return if ATN active
        ld      a,020h          ;space
        call    txbyte
        ret     c
        djnz    nxbyt2
        ld      a,10
        call    txbyte
        ret

TScom:  ld      hl,(modprg)     ;get mode programme start address
        jp      (hl)            ;execute current mode programme
```

```
;Start of sampling mode programmes
;Mode programmes are placed at 200h intervals starting at 'modes'

        org     modes           ;start of mode 1

nmbtsl  equ     3               ;number of bytes per result
nmresl  equ     2               ;number of results
nmcysl  equ     16              ;number of cycles sampled
nmspsl  equ     256             ;total number of samples
spcycl  equ     16              ;number of samples per cycle
smperl  equ     modmem+10       ;sample error flag
MSBmxl  equ     0Fh             ;MSB of max ADC output
LSBmxl  equ     0FFh            ;LSB of max ADC output

;50khz output frequency, 51.282 khz mixer frequency, 1.282khz filter frequency
;16 samples per cycle of downconverted signal
;256 samples per channel

model:  jp      mlinit                  ;model initialisation
        jp      mlexec                  ;model execute address
        dw      nmspsl                  ;number of samples required

mlinit: ld      a,01110b
        out     (PIOA+3),a      ;disable counters

        ld      a,000110110b    ;counter 0;set LSB then MSB;mode3;binary
        out     (CTC+3),a       ;50 kHz
        ld      a,160           ;LSB of count
        out     (CTC+0),a
        ld      a,0             ;MSB of count
        out     (CTC+0),a

        ld      a,001110110b    ;counter 1;set LSB then MSB;mode 3;binary
        out     (CTC+3),a       ;51.282 kHz
        ld      a,156           ;LSB of count
        out     (CTC+1),a
        ld      a,0             ;MSB of count
        out     (CTC+1),a

        ld      a,010110100b    ;counter 2;set LSB then MSB;mode2;binary
        out     (CTC+3),a       ;16*1.282 kHz
        ld      a,134           ;LSB of count
        out     (CTC+2),a
        ld      a,1             ;MSB of count
        out     (CTC+2),a

        ld      a,01111b
        out     (PIOA+3),a      ;enable counters
        ret

mlexec: ld      hl,(data0)      ;address of data from ADC0
        call    resltl          ;calculate result from table
        call    sendl           ;send result
        ret     c               ;return if ATN active
```

```
;Start of sampling mode programmes
;Mode programmes are placed at 200h intervals starting at 'modes'

        org     modes           ;start of mode 1

nmbtsl  equ     3               ;number of bytes per result
nmresl  equ     2               ;number of results
nmcysl  equ     16              ;number of cycles sampled
nmspsl  equ     256             ;total number of samples
spcycl  equ     16              ;number of samples per cycle
smperl  equ     modmem+10       ;sample error flag
MSBmxl  equ     0Fh             ;MSB of max ADC output
LSBmxl  equ     0FFh            ;LSB of max ADC output

;50khz output frequency, 51.282 khz mixer frequency, 1.282khz filter frequency
;16 samples per cycle of downconverted signal
;256 samples per channel

model:  jp      mlinit                  ;model initialisation
        jp      mlexec                  ;model execute address
        dw      nmspsl                  ;number of samples required

mlinit: ld      a,01110b
        out     (PIOA+3),a      ;disable counters

        ld      a,000110110b    ;counter 0;set LSB then MSB;mode3;binary
        out     (CTC+3),a       ;50 kHz
        ld      a,160           ;LSB of count
        out     (CTC+0),a
        ld      a,0             ;MSB of count
        out     (CTC+0),a

        ld      a,001110110b    ;counter 1;set LSB then MSB;mode 3;binary
        out     (CTC+3),a       ;51.282 kHz
        ld      a,156           ;LSB of count
        out     (CTC+1),a
        ld      a,0             ;MSB of count
        out     (CTC+1),a

        ld      a,010110100b    ;counter 2;set LSB then MSB;mode2;binary
        out     (CTC+3),a       ;16*1.282 kHz
        ld      a,134           ;LSB of count
        out     (CTC+2),a
        ld      a,1             ;MSB of count
        out     (CTC+2),a

        ld      a,01111b
        out     (PIOA+3),a      ;enable counters
        ret

mlexec: ld      hl,(data0)      ;address of data from ADC0
        call    resltl          ;calculate result from table
        call    sendl           ;send result
        ret     c               ;return if ATN active
```

```
        ld      hl,(data1)      ;address of data from ADC1
        call    resltl          ;calculate result from table
        call    send1           ;send result
        ret     c               ;return if ATN active

        ld      a,10
        call    txbyte          ;send line feed character
        ret

resltl: push    hl
        ld      hl,modmem       ;start of results in mode memory
        ld      b,nmbts1*nmres1 ;clear results buffer

clnxt1: ld      (hl),0
        inc     hl
        djnz    clnxt1
        pop     hl

        xor     a
        ld      (smper1),a

        ld      c,nmcys1        ;number of cycles sampled

nxcycl: ld      b,0
nxsmpl: ld      e,(hl)
        inc     hl
        ld      d,(hl)
        inc     hl

        ld      a,d
        or      e
        jr      nz,nounf1       ;no under flow
        jr      unf1

nounf1: ld      a,d
        cp      MSBmx1
        jr      nz,noovf1
        ld      a,e
        cp      LSBmx1
        jr      nz,noovf1

unf1:   ld      a,1
        ld      (smper1),a

noovf1: push    hl

        ld      hl,modmem       ;address of first result in memory
        ld      a,b
        cp      (spcyc1)/2
        jr      nc,jump50

        call    add1            ;add to total
        jr      jump51

jump50: call    sub1            ;subtract from total
```

```
jump51: ld      hl,modmem+nmbts1;address of second result in memory
        ld      a,b
        cp      (spcyc1)/4
        jr      c,jump52
        cp      (spcyc1)*3/4
        jr      nc,jump52

        call    add1
        jr      jump53

jump52: call    sub1

jump53: pop     hl
        inc     b
        ld      a,b
        cp      spcyc1
        jr      c,nxsmp1
        dec     c
        jr      nz,nxcyc1
        ret

add1:   push    de              ;de=new data, hl=result address
        push    bc
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ex      de,hl
        add     hl,bc
        ex      de,hl
        dec     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
        jr      nc,nocry1       ;no carry set
        inc     hl
        inc     (hl)            ;increment MSB
nocry1: pop     bc
        pop     de
        ret

sub1:   push    bc              ;de=new data, hl=result address
        push    de
        push    hl
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ld      l,c
        ld      h,b
        and     a
        sbc     hl,de
        ex      de,hl
        pop     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
```

```
        jr      nc,nobrr1       ;no borrow
        inc     hl
        dec     (hl)            ;decrement MSB
nobrr1: pop     de
        pop     bc
        ret

sendl:  ld      a,(smperl)
        and     a
        ld      a,'C'
        jr      z,noerr1
        ld      a,'E'
noerr1: call    txbyte
        ret     c
        ld      a,32
        call    txbyte
        ret     c

        ld      hl,modmem       ;start of results
        ld      de,nmbtsl       ;length of results
        ld      b,nmresl        ;number of results

jump55: call    sndhx1
        ret     c
        ld      a,32
        call    txbyte
        ret     c
        add     hl,de
        djnz    jump55
        ret

sndhx1: push    bc
        ld      b,e             ;bytes per sample
        add     hl,de

sndnx1: dec     hl              ;hl equals address of MSB
        ld      a,(hl)
        call    sndhex
        jr      c,sndnd1
        djnz    sndnx1
sndnd1: pop     bc
        ret
```

```
        org     modes+0200h     ;start of mode 2

nmbts2  equ     3               ;number of bytes per result
nmres2  equ     2               ;number of results
nmcys2  equ     32              ;number of cycles sampled
nmsps2  equ     256             ;total number of samples
spcyc2  equ     8               ;number of samples per cycle
smper2  equ     modmem+10       ;sample error flag
MSBmx2  equ     0Fh             ;MSB of max ADC output
LSBmx2  equ     0FFh            ;LSB of max ADC output

;50khz output frequency, 51.282 khz mixer frequency, 1.282khz filter frequency
;8 samples per cycle of downconverted signal
;256 samples per channel

mode2:  jp      m2init                  ;mode2 initialisation
        jp      m2exec                  ;mode2 execute address
        dw      nmsps2                  ;number of samples required

m2init: ld      a,01110b
        out     (PIOA+3),a      ;disable counters

        ld      a,000110110b    ;counter 0;set LSB then MSB;mode3;binary
        out     (CTC+3),a       ;50 kHz
        ld      a,160           ;LSB of count
        out     (CTC+0),a
        ld      a,0             ;MSB of count
        out     (CTC+0),a

        ld      a,001110110b    ;counter 1;set LSB then MSB;mode 3;binary
        out     (CTC+3),a       ;51.282 kHz
        ld      a,156           ;LSB of count
        out     (CTC+1),a
        ld      a,0             ;MSB of count
        out     (CTC+1),a

        ld      a,010110100b    ;counter 2;set LSB then MSB;mode2;binary
        out     (CTC+3),a       ;8*1.282 kHz
        ld      a,12            ;LSB of count
        out     (CTC+2),a
        ld      a,3             ;MSB of count
        out     (CTC+2),a

        ld      a,01111b
        out     (PIOA+3),a      ;enable counters
        ret

m2exec: ld      hl,(data0)      ;address of data from ADC0
        call    reslt2          ;calculate result from table
        call    send2           ;send result
        ret     c               ;return if ATN active

        ld      hl,(data1)      ;address of data from ADC1
        call    reslt2          ;calculate result from table
        call    send2           ;send result
```

```
        ret     c               ;return if ATN active

        ld      a,10
        call    txbyte          ;send line feed character
        ret

reslt2: push    hl
        ld      hl,modmem       ;start of results in mode memory
        ld      b,nmbts2*nmres2 ;clear results buffer

clnxt2: ld      (hl),0
        inc     hl
        djnz    clnxt2
        pop     hl

        xor     a
        ld      (smper2),a

        ld      c,nmcys2        ;number of cycles sampled

nxcyc2: ld      b,0
nxsmp2: ld      e,(hl)
        inc     hl
        ld      d,(hl)
        inc     hl

        ld      a,d
        or      e
        jr      nz,nounf2       ;no under flow
        jr      unf2

nounf2: ld      a,d
        cp      MSBmx2
        jr      nz,noovf2
        ld      a,e
        cp      LSBmx2
        jr      nz,noovf2

unf2:   ld      a,1
        ld      (smper2),a

noovf2: push    hl

        ld      hl,modmem       ;address of first result in memory
        ld      a,b
        cp      (spcyc2)/2
        jr      nc,jump60

        call    add2            ;add to total
        jr      jump61

jump60: call    sub2            ;subtract from total

jump61: ld      hl,modmem+nmbts2;address of second result in memory
        ld      a,b
```

```
        cp      (spcyc2)/4
        jr      c,jump62
        cp      (spcyc2)*3/4
        jr      nc,jump62

        call    add2
        jr      jump63

jump62: call    sub2

jump63: pop     hl
        inc     b
        ld      a,b
        cp      spcyc2
        jr      c,nxsmp2
        dec     c
        jr      nz,nxcyc2
        ret

add2:   push    de              ;de=new data, hl=result address
        push    bc
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ex      de,hl
        add     hl,bc
        ex      de,hl
        dec     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
        jr      nc,nocry2       ;no carry set
        inc     hl
        inc     (hl)            ;increment MSB
nocry2: pop     bc
        pop     de
        ret

sub2:   push    bc              ;de=new data, hl=result address
        push    de
        push    hl
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ld      l,c
        ld      h,b
        and     a
        sbc     hl,de
        ex      de,hl
        pop     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
        jr      nc,nobrr2       ;no borrow
        inc     hl
```

```
        dec     (hl)            ;decrement MSB
nobrr2: pop     de
        pop     bc
        ret

send2:  ld      a,(smper2)
        and     a
        ld      a,'C'
        jr      z,noerr2
        ld      a,'E'
noerr2: call    txbyte
        ret     c
        ld      a,32
        call    txbyte
        ret     c

        ld      hl,modmem       ;start of results
        ld      de,nmbts2       ;length of results
        ld      b,nmres2        ;number of results

jump65: call    sndhx2
        ret     c
        ld      a,32
        call    txbyte
        ret     c
        add     hl,de
        djnz    jump65
        ret

sndhx2: push    bc
        ld      b,e             ;bytes per sample
        add     hl,de

sndnx2: dec     hl              ;hl equals address of MSB
        ld      a,(hl)
        call    sndhex
        jr      c,sndnd2
        djnz    sndnx2
sndnd2: pop     bc
        ret
```

```
        org     modes+0400h     ;start of mode 3

nmbts3  equ     3               ;number of bytes per result
nmres3  equ     2               ;number of results
nmcys3  equ     64              ;number of cycles sampled
nmsps3  equ     256             ;total number of samples
spcyc3  equ     4               ;number of samples per cycle
smper3  equ     modmem+10       ;sample error flag
MSBmx3  equ     0Fh             ;MSB of max ADC output
LSBmx3  equ     0FFh            ;LSB of max ADC output

;50khz output frequency, 51.282 khz mixer frequency, 1.282khz filter frequency
;4 samples per cycle of downconverted signal
;256 samples per channel

mode3:  jp      m3init                  ;mode3 initialisation
        jp      m3exec                  ;mode3 execute address
        dw      nmsps3                  ;number of samples required

m3init: ld      a,01110b
        out     (PIOA+3),a      ;disable counters

        ld      a,000110110b    ;counter 0;set LSB then MSB;mode3;binary
        out     (CTC+3),a       ;50 kHz
        ld      a,160           ;LSB of count
        out     (CTC+0),a
        ld      a,0             ;MSB of count
        out     (CTC+0),a

        ld      a,001110110b    ;counter 1;set LSB then MSB;mode 3;binary
        out     (CTC+3),a       ;51.282 kHz
        ld      a,156           ;LSB of count
        out     (CTC+1),a
        ld      a,0             ;MSB of count
        out     (CTC+1),a

        ld      a,010110100b    ;counter 2;set LSB then MSB;mode2;binary
        out     (CTC+3),a       ;4*1.282 kHz
        ld      a,24            ;LSB of count
        out     (CTC+2),a
        ld      a,6             ;MSB of count
        out     (CTC+2),a

        ld      a,01111b
        out     (PIOA+3),a      ;enable counters
        ret

m3exec: ld      hl,(data0)      ;address of data from ADC0
        call    reslt3          ;calculate result from table
        call    send3           ;send result
        ret     c               ;return if ATN active

        ld      hl,(data1)      ;address of data from ADC1
        call    reslt3          ;calculate result from table
        call    send3           ;send result
```

```
        ret     c               ;return if ATN active

        ld      a,10
        call    txbyte          ;send line feed character
        ret

reslt3: push    hl
        ld      hl,modmem       ;start of results in mode memory
        ld      b,nmbts3*nmres3 ;clear results buffer

clnxt3: ld      (hl),0
        inc     hl
        djnz    clnxt3
        pop     hl

        xor     a
        ld      (smper3),a

        ld      c,nmcys3        ;number of cycles sampled

nxcyc3: ld      b,0
nxsmp3: ld      e,(hl)
        inc     hl
        ld      d,(hl)
        inc     hl

        ld      a,d
        or      e
        jr      nz,nounf3       ;no under flow
        jr      unf3

nounf3: ld      a,d
        cp      MSBmx3
        jr      nz,noovf3
        ld      a,e
        cp      LSBmx3
        jr      nz,noovf3

unf3:   ld      a,1
        ld      (smper3),a

noovf3: push    hl

        ld      hl,modmem       ;address of first result in memory
        ld      a,b
        cp      (spcyc3)/2
        jr      nc,jump70

        call    add3            ;add to total
        jr      jump71

jump70: call    sub3            ;subtract from total

jump71: ld      hl,modmem+nmbts2;address of second result in memory
        ld      a,b
```

```
        cp      (spcyc3)/4
        jr      c,jump72
        cp      (spcyc3)*3/4
        jr      nc,jump72

        call    add3
        jr      jump73

jump72: call    sub3

jump73: pop     hl
        inc     b
        ld      a,b
        cp      spcyc3
        jr      c,nxsmp3
        dec     c
        jr      nz,nxcyc3
        ret

add3:   push    de              ;de=new data, hl=result address
        push    bc
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ex      de,hl
        add     hl,bc
        ex      de,hl
        dec     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
        jr      nc,nocry3       ;no carry set
        inc     hl
        inc     (hl)            ;increment MSB
nocry3: pop     bc
        pop     de
        ret

sub3:   push    bc              ;de=new data, hl=result address
        push    de
        push    hl
        ld      c,(hl)
        inc     hl
        ld      b,(hl)
        ld      l,c
        ld      h,b
        and     a
        sbc     hl,de
        ex      de,hl
        pop     hl
        ld      (hl),e
        inc     hl
        ld      (hl),d
        jr      nc,nobrr3       ;no borrow
        inc     hl
```

```
        dec     (hl)            ;decrement MSB
nobrr3: pop     de
        pop     bc
        ret

send3:  ld      a,(smper3)
        and     a
        ld      a,'C'
        jr      z,noerr3
        ld      a,'E'
noerr3: call    txbyte
        ret     c
        ld      a,32
        call    txbyte
        ret     c

        ld      hl,modmem       ;start of results
        ld      de,nmbts3       ;length of results
        ld      b,nmres3        ;number of results

jump75: call    sndhx3
        ret     c
        ld      a,32
        call    txbyte
        ret     c
        add     hl,de
        djnz    jump75
        ret

sndhx3: push    bc
        ld      b,e             ;bytes per sample
        add     hl,de

sndnx3: dec     hl              ;hl equals address of MSB
        ld      a,(hl)
        call    sndhex
        jr      c,sndnd3
        djnz    sndnx3
sndnd3: pop     bc
        ret



        end
```

The HP-IB Interface

The Hewlett-Packard Interface Bus (HP-IB) was developed at HP as the solution to an expanding need for a universal interfacing technique that could be readily adapted to a wide variety of electronic instruments. It was later expanded to include high-speed disc drives and other high-performance computer peripherals. The HP-IB architecture was subsequently proposed to and accepted by the Institute of Electrical and Electronic Engineers (IEEE) and is now widely used throughout the electronic industry. HP-IB is compatible with IEEE standard 488-1978. The number of devices that can be connected to a given HP-IB interface depends on the loading factor of each device, but in general up to 15 devices (including the interface) can be connected together while still maintaining electrical, mechanical, and timing compatibility requirements on the bus.

General Structure

IEEE Standard 488-1978 defines a set of communication rules called "bus protocol" that governs data and control operations on the bus. The defined protocol is necessary in order to ensure orderly information traffic over the bus.

Each device (peripheral or computer interface) that is connected to the HP-IB can function in one or more of the following roles:

System Controller	Master controller of the HP-IB. The computer interface is usually the bus controller when all peripheral devices on the bus are slaves to the system computer. However, any other device can become the active controller if it is equipped to act as a controller and control is passed to it by the System Controller. The System Controller is always the active bus controller at power-up.
Active Controller	Current controller of the HP-IB. At power-up or whenever IFC (InterFace Clear) is asserted by the System Controller, the System Controller is the active controller. Under certain conditions, the System controller may pass control to another device that is capable of managing the bus in which case that device becomes the new active controller. The active controller can then pass control to another controller or back to the System Controller. If the System Controller asserts IFC, the active controller immediately relinquishes control of the bus.
Talker	A device that has been authorized by the current active controller to place data on the bus. Only one talker can be authorized at a time.

Listener — Any device that has been programmed by the active controller to accept data from the bus. Any number of devices on the bus can be programmed by the active controller to listen simultaneously at any given time.

In typical systems, an HP-IB interface in the computer can act as a **controller, talker**, and **listener**. If more than one computer is connected to the same bus, only one interface can be configured as System Controller to prevent conflicts at power-up (this is usually accomplished by a switch or wire jumper on the interface card). A device that can only accept data from the bus (such as a line printer) usually operates as a **listener**, while a device that can only supply data to the bus (such as a voltmeter) usually operates as a **talker**. However, before any device can talk or listen (after power-up initialization), it must be authorized to do so by the current active controller. Bus configuration varies, depending on the type of activity that is prevalent at the time. However, in any case, the bus can have only one Active Controller and only one talker at a given time, though it can have any number of listeners.

HP-IB is composed of 16 lines (plus ground) that are divided into 3 groups:

- Eight data lines form a bi-directional data path to carry data, commands, and device addresses.
- Three handshake lines control the transfer of data bytes.
- The five remaining lines control bus management.

Handshake Lines

The **handshake** lines used to synchronize data transfers are:

$\overline{\text{DAV}}$ — DAta Valid: Valid data has been placed on bus by talker.

$\overline{\text{NRFD}}$ — Not Ready For Data: One or more listeners not yet ready to accept data from the bus.

$\overline{\text{NDAC}}$ — Not Data ACcepted: One or more listeners has not yet accepted the data currently on the bus.

NOTE

The HP-IB interface uses negative (ground-true) logic for handshake, data, and bus management lines. This means that when the voltage on a line is at a logic LOW level, the line is **asserted** (true). When a logic HIGH voltage level is present on the line, the line is **not asserted** (false).

In general, software documentation refers to handshake and other lines by their name acronym such as DAV, NRFD, NDAC, etc. When discussing these same signal lines in hardware documents, it is customary to refer to ground-true (low-true) logic lines by their name acronym with a bar across the top such as $\overline{\text{DAV}}$, $\overline{\text{NRFD}}$, $\overline{\text{NDAC}}$, etc. In this document, both versions are used. The overbar is usually present when discussing hardware operation, but usually absent when software is being treated. In this tutorial, only the name is significant; signal names are synonymous with or without the overbar unless specifically noted otherwise – the overbar is used for the convenience of those readers whose experience is oriented more toward hardware than software.

The timing diagram in Figure 1−2 shows how handshake lines are used to complete a data item transfer. The discussion which follows is based on the contents of Figure 1−2.

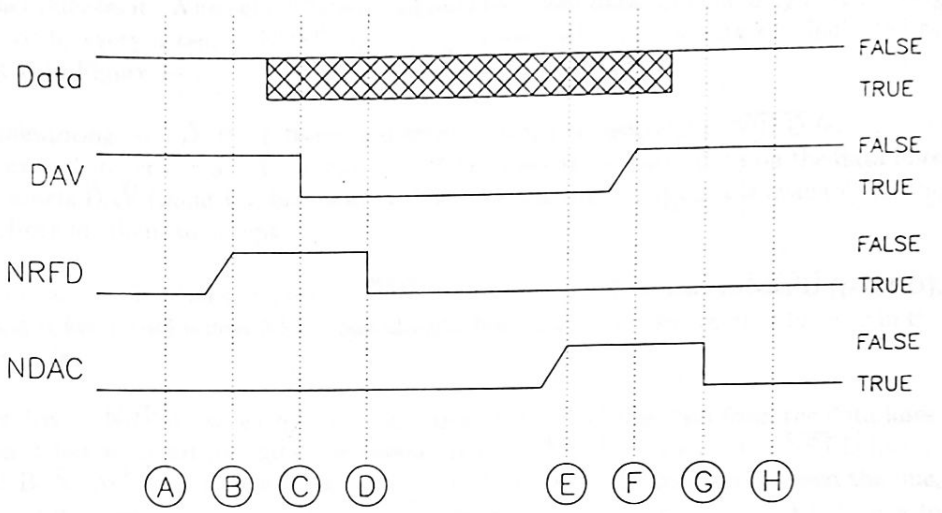


Figure 1−2. The HP-IB Handshake

All handshake lines are electrically connected in a "wired-OR" configuration which means that any device can pull the line low (active or asserted) at any time, and more than one device may pull the line low simultaneously or later in a given handshake cycle. The line then remains low until every device that was previously pulling the line low has released the line, allowing it to float to its high state. At the start of the handshake cycle (point A), the handshake lines are in the following states:

- $\overline{\text{DAV}}$ is false (high), meaning that the current talker has not yet placed valid data on the bus.
- $\overline{\text{NRFD}}$ is true (low), meaning that one or more listeners is not yet ready to accept data from the bus.
- $\overline{\text{NDAC}}$ is true (low), meaning that bus data has not yet been accepted by every listener on the bus.

When a listener is ready to accept data, it releases $\overline{\text{NRFD}}$, allowing it to go high provided no other listener is still holding the line low. However (due to the "wired-OR" interconnection scheme used by HP-IB), $\overline{\text{NRFD}}$ remains LOW (true) until **every** listener releases it. When every listener is ready to accept data (indicated by $\overline{\text{NRFD}}$ being released by every listener), $\overline{\text{NRFD}}$ changes to its logic HIGH (false) state as indicated by point B in Figure 1−2.

By monitoring $\overline{\text{NRFD}}$, the talker can determine when to send data: $\overline{\text{NRFD}}$ false means that every listener is ready to accept data. The talker then places data on the data lines and asserts $\overline{\text{DAV}}$ (point C), indicating to the listeners that valid data is available on the data lines for them to accept.

As soon as each listener detects that $\overline{\text{DAV}}$ has been asserted, it asserts $\overline{\text{NRFD}}$ (point D), driving it low (true) unless $\overline{\text{NRFD}}$ has already been driven low by another listener in the same cycle.

After driving $\overline{\text{NRFD}}$ low, each listener inputs and processes the data from the data lines. When it has accepted the data, the listener releases $\overline{\text{NDAC}}$. As with the $\overline{\text{NRFD}}$ line at point B, $\overline{\text{NDAC}}$ remains low (true) until every listener on the bus has released the line, allowing it to go high (false). When $\overline{\text{NDAC}}$ goes high, the false logic state indicates to the talker that every listener has accepted the data (point E).

When the talker determines that every listener has accepted the data, it releases the $\overline{\text{DAV}}$ line which rises to its high (false) state. At the same time, the talker disables its outputs to the data lines, allowing them to rise to their high (false) state (point F).

When $\overline{\text{DAV}}$ goes false, the listeners assert $\overline{\text{NDAC}}$ (point G), driving it low. This signifies the end of the handshake (point H), at which time all bus logic lines are again at the same state as they were before the handshake started (point A).

Bus Management Control Lines

There are five bus management control lines:

$\overline{\text{ATN}}$	ATtentioN: Treat data on data lines as commands, not data.
$\overline{\text{IFC}}$	InterFace Clear: Unconditionally terminate all current bus activity.
$\overline{\text{REN}}$	Remote ENable: Place all current listeners in Remote operating mode.
$\overline{\text{EOI}}$	End Or Identify: End of data message. If $\overline{\text{ATN}}$ is true (low), Active Controller is conducting a parallel poll (Identify) of devices on the bus.
$\overline{\text{SRQ}}$	Service ReQuest: Bus device is requesting service from current Active Controller.

$\overline{\text{ATN}}$: The Attention Line

Command messages are encoded on the data lines as 7-bit ASCII characters, and are distinguished from the normal data characters by the attention ($\overline{\text{ATN}}$) line's logic state. That is, when $\overline{\text{ATN}}$ is false, the states of the data lines are interpreted as data. When $\overline{\text{ATN}}$ is true, the data lines are interpreted as commands.

$\overline{\text{IFC}}$: The Interface Clear Line

Only the System Controller sets the $\overline{\text{IFC}}$ line true. By asserting $\overline{\text{IFC}}$, all bus activity is unconditionally terminated, the System Controller becomes the Active Controller, and any current talker and all listeners become unaddressed. Normally, this line is used to terminate all current operations, or to allow the System Controller to regain control of the bus. It overrides any other activity currently taking place on the bus.

$\overline{\text{REN}}$: The Remote Enable Line

This line allows instruments on the bus to be programmed remotely by the Active Controller. Any device addressed to listen while $\overline{\text{REN}}$ is true is placed in its remote mode of operation.

$\overline{\text{EOI}}$: The End or Identify Line

If $\overline{\text{ATN}}$ is false, $\overline{\text{EOI}}$ is used by the current talker to indicate the end of a data message. Normally, data messages sent over the HP-IB are sent using strings of standard ASCII code terminated by the ASCII line-feed character. However, certain devices must handle blocks of information containing data bytes within the data message that are identical to the line-feed character bit pattern, thus making it inappropriate to use a line-feed as the terminating character. For this reason, $\overline{\text{EOI}}$ is used to mark the end of the data message.

The Active Controller can use $\overline{\text{EOI}}$ with $\overline{\text{ATN}}$ true to conduct a parallel poll on the bus.

$\overline{\text{SRQ}}$: The Service Request Line
The Active Controller is always in charge of overall bus activity, performing such tasks as determining which devices are talkers and listeners, and so forth. If a device on the bus needs assistance from the Active Controller, it asserts $\overline{\text{SRQ}}$, driving the line low (true). $\overline{\text{SRQ}}$ is a request for service, not a demand, so the Active Controller has the option of choosing when and how the request is to be serviced. However, the device continues to assert $\overline{\text{SRQ}}$ until it has been satisfied (or until an interface clear command disables the request). Exactly what satisfies a service request depends on the requesting device, and is explained in the operating manual for the device.

The GPIO Interface

The **GPIO** (General Purpose Input/Output) interface is a very flexible parallel interface that can be used to communicate with a variety of devices. The GPIO interface utilizes data, handshake, and special-purpose lines to perform data transfers by means of various user-selectable handshaking methods.

While the GPIO interfaces used on various HP-UX computers are electrically very similar, they differ in certain important aspects. Refer to the appendices for Series 300 and 800 for information pertaining to your specific application.

Overview of HP-IB Commands

HP-IB commands consist of various data sequences that are sent over the eight HP-IB data lines while the ATN line is asserted (held LOW). The DIL subroutine *hpib_send_cmnd* provides a convenient means for sending bus commands by automatically handling the ATN line and the necessary handshaking operations between devices. However, *hpib_send_cmnd* can be used **only** when the computer interface to the bus is the active controller. Techniques for using *hpib_send_cmnd* are discussed later in this chapter.

Any device that is the intended recipient of an HP-IB command must have its remote enable line (REN) enabled by the System Controller (unless altered by the System Controller, REN is enabled, by default). Only the System Controller can alter the state of the REN line (see "System Controller's Duties" section later in this chapter).

HP-IB Data Bus Commands fall into four categories:

- **Universal commands** cause every properly equipped device on the bus to perform the specified interface operation, whether addressed to listen or not.
- **Addressed commands** are similar to universal commands, but are accepted only by bus devices that are currently addressed as listeners.
- **Talk and listen addresses** are commands that assign talkers and listeners on the bus.
- **Secondary commands** are commands that must always be used in conjunction with a command from one of the above groups.

The following table lists commands that can be sent with *hpib_send_cmnd*, along with the decimal and ASCII character equivalents of each command. This table is useful for reference when determining what values to use as parameters in *hpib_send_cmnd* subroutine calls.

Table 3.1 HP-IB Bus Commands

Command	Decimal Value	ASCII Character
Universal Commands:		
UNLISTEN	63	?
UNTALK	95	_
DEVICE CLEAR	20	DC4
LOCAL LOCKOUT	17	DC1
SERIAL POLL ENABLE	24	CAN
SERIAL POLL DISABLE	25	EM
PARALLEL POLL UNCONFIGURE	21	NAK
Addressed Commands:		
TRIGGER	8	BS
SELECTED DEVICE CLEAR	4	EOT
GO TO LOCAL	1	SOH
PARALLEL POLL CONFIGURE	5	ENQ
TAKE CONTROL	9	HT
Talk and Listen Addresses:		
Talk Addresses 0-30	64-94	@ thru ^ (uppercase ASCII)
Listen Addresses 0-30	32-62	space thru > (numbers and special characters)
Secondary Commands: (If a secondary command follows the PARALLEL POLL CONFIGURE command then it is interpreted as follows, otherwise its meaning is device dependent)		
PARALLEL POLL ENABLE	96-111	` thru o (lowercase ASCII)
PARALLEL POLL DISABLE	112	p

UNLISTEN
UNLISTEN **unaddresses** all current listeners on the bus. No means is available for unaddressing a given listener without unaddressing all listeners on the bus. This command ensures that the bus is cleared of all listeners before addressing a new listener or group of listeners.

UNTALK
UNTALK **unaddresses** any active talkers on the bus. Since no means is available for unaddressing a given talker, the UNTALK command is sent to all devices on the bus. This ensures that no conflict with a current talker can occur when addressing a new one.

DEVICE CLEAR
DEVICE CLEAR causes all devices that recognize this command to return to a predefined, device-dependent state, independent of any previous addressing. The reset state for any given device after accepting this command is documented in the operating manual for the device in question.

LOCAL LOCKOUT
LOCAL LOCKOUT disables local (front panel) control on all devices that recognize this command, whether the devices have been addressed or not.

SERIAL POLL ENABLE
SERIAL POLL ENABLE establishes serial poll mode for all devices that are capable of being bus talkers, provided they recognize and support the command. This command operates independent of whether the devices being polled have been addressed to talk. When a device is addressed to talk, it returns an 8-bit status byte message.

This command is handled through the DIL subroutine *hpib_spoll*, as discussed later in this chapter.

SERIAL POLL DISABLE
SERIAL POLL DISABLE terminates serial poll mode for all devices that support this command, whether or not the individual devices have been addressed.

The DIL subroutine *hpib_spoll* that performs this function is discussed at length later in this chapter.

TRIGGER (Group Execute Trigger)
TRIGGER causes devices currently addressed as listeners to initiate a preprogrammed, device-dependent action if they are capable of doing so. Use of this function and programming procedures are documented in operating manuals for devices that support it.

SELECTED DEVICE CLEAR
SELECTED DEVICE CLEAR resets devices currently addressed as listeners to a device-dependent state, provided they support the command. Refer to the device operating manual for more information about programming and the resulting state(s).

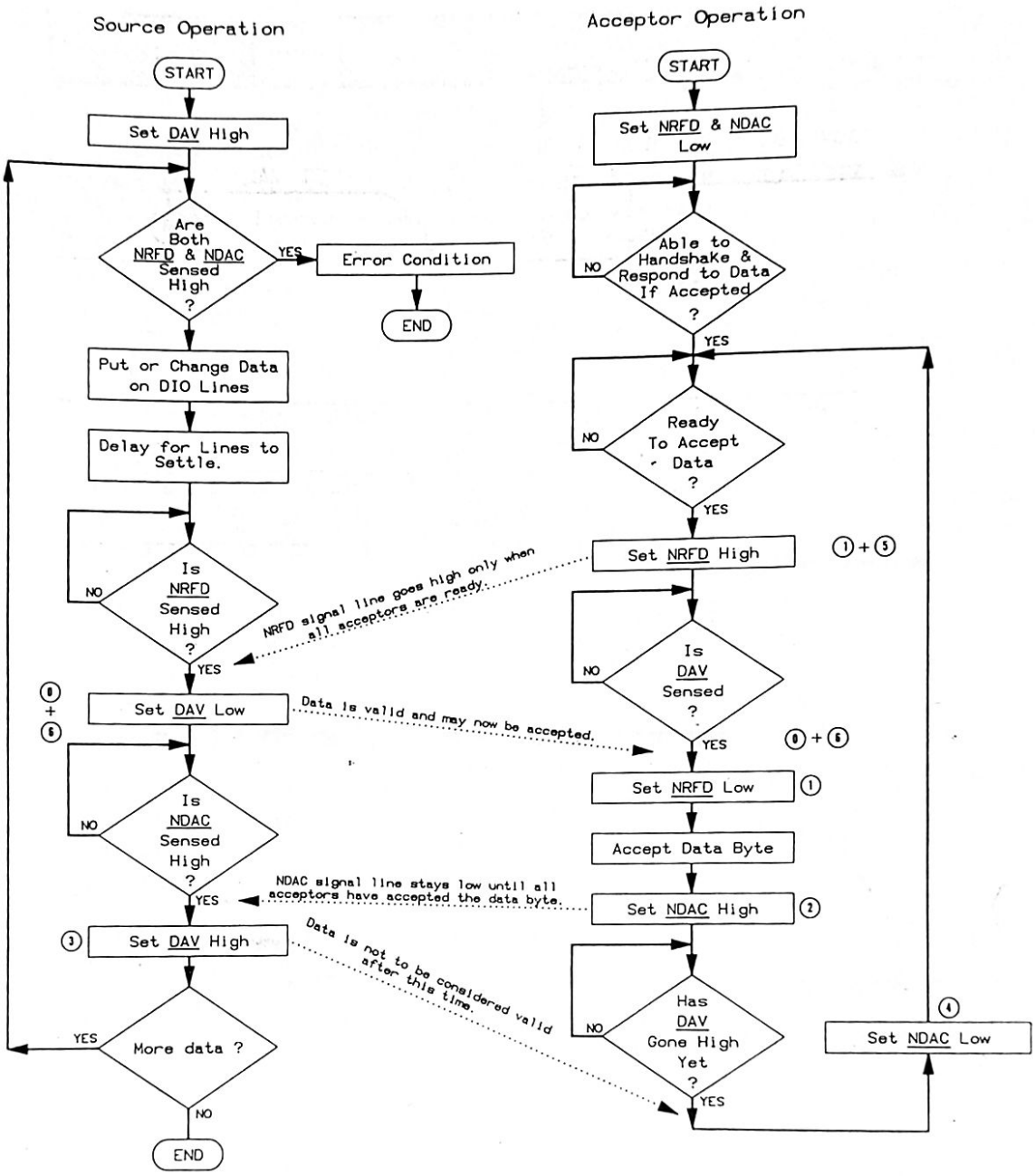
GO TO LOCAL
GO TO LOCAL causes devices currently addressed as listeners to return to the local-control state (exit from the remote state). Devices return to remote state next time they are addressed.

PARALLEL POLL CONFIGURE
PARALLEL POLL CONFIGURE tells devices currently addressed as listeners that a secondary command follows. This secondary command must be either PARALLEL POLL ENABLE or PARALLEL POLL DISABLE.

PARALLEL POLL ENABLE
PARALLEL POLL ENABLE configures devices addressed by PARALLEL POLL CONFIGURE to respond to parallel polls with a predefined logic level on a particular data line. On some devices, the response is implemented in a local form (such as by using hardware jumper wires) that cannot be changed.

Use of this command must be preceded by a PARALLEL POLL CONFIGURE command.

PARALLEL POLL DISABLE
The PARALLEL POLL DISABLE command prevents devices previously addressed by a PARALLEL POLL CONFIGURE command from responding to parallel polls. This command must be preceded by the PARALLEL POLL CONFIGURE command.



Logical flow of events for Source and Acceptor when transferring data using the handshake process.

Figure 2.4 Handshake Timing Sequence

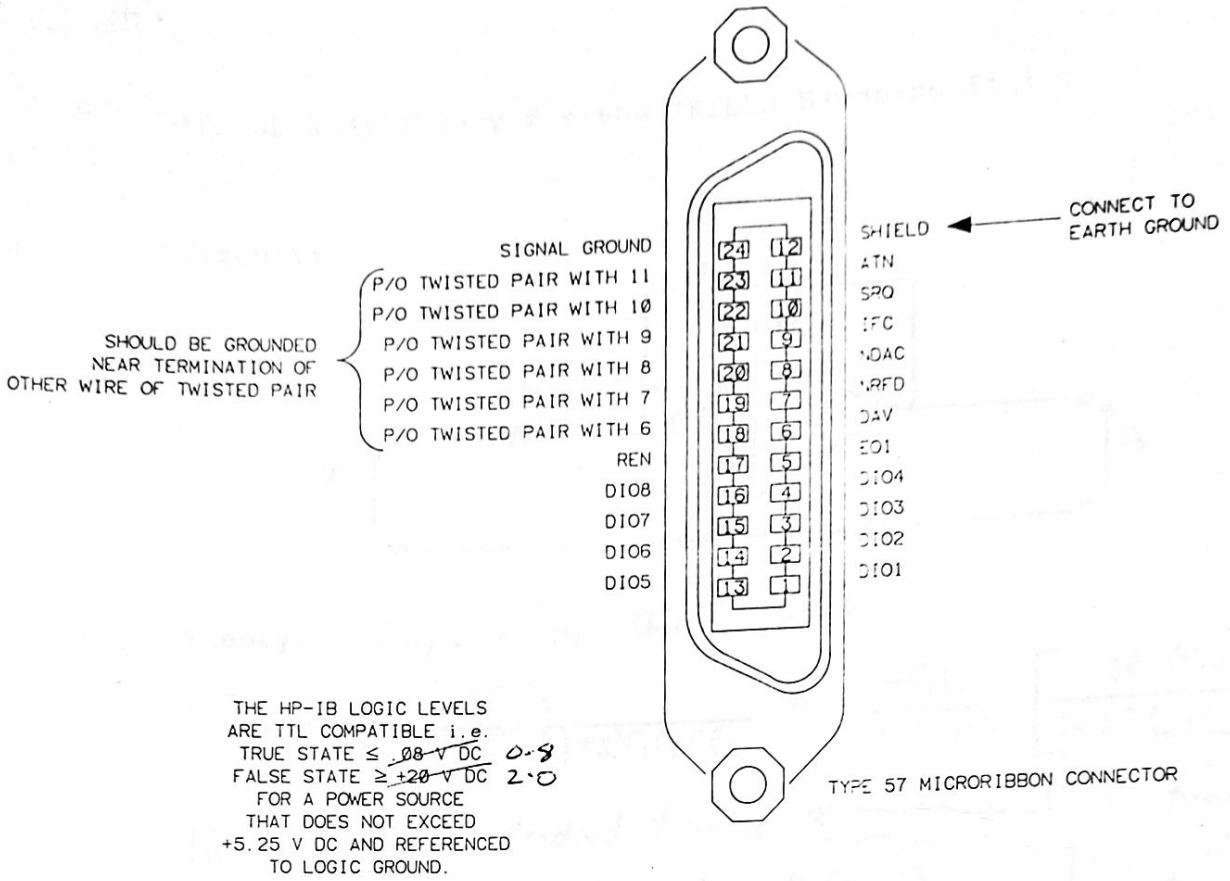
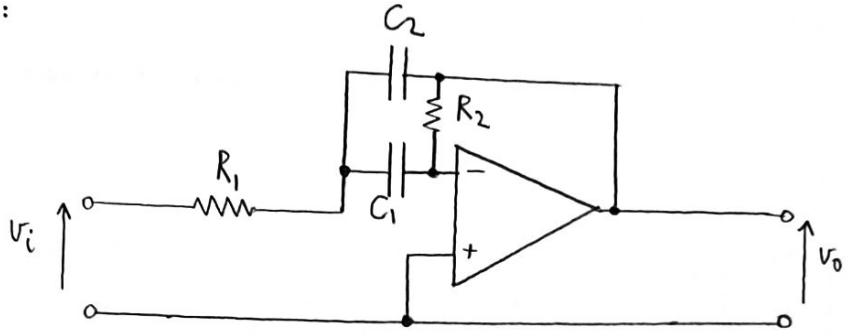


Figure 2.7 HP-IB Connector Pin Assignments

Viewed into SOCKET.

APPENDIX 4: Theory for the FRIEND Bandpass filter

Circuit:



Theory: Simple network theory gives:

$$\frac{V_o}{V_i} = \frac{-sC_1R_2}{1+sR_1(C_1+C_2)+s^2C_1C_2R_1R_2} = \frac{-C_1R_2}{R_1(C_1+C_2)}\left[\frac{sR_1(C_1+C_2)}{1+sR_1(C_1+C_2)+s^2C_1C_2R_1R_2}\right]$$

From inspection of standard form of second order transfer function:

$$T = \frac{1}{\omega_0} = \sqrt{C_1C_2R_1R_2} \; ; \; \frac{1}{Q} = \frac{R_1(C_1+C_2)}{T} = \sqrt{\frac{R_1}{R_2}}\left[\sqrt{\frac{C_1}{C_2}} + \sqrt{\frac{C_2}{C_1}}\right]$$

These are easiest to deal with if $C_1=C_2$; this also gives max Q.

Then $f_0 = \frac{1}{2\pi C\sqrt{R_1R_2}}$ and $Q = \frac{1}{2}\sqrt{\frac{R_2}{R_1}}$.

Gain at $f=f_0$ is $-\frac{1}{2}\frac{R_2}{R_1}$ as bracketed term becomes unity.

There should be no phase shift at $f=f_0$.

IF Filter: A unity gain filter was required, so R_1 was replaced by a potential divider. Three FRIEND filters were used in cascade. The calculated figures for one stage of the filter are as follows:

R_1=18k//5k6, R_2=36k and C_1=C_2=10nF giving:

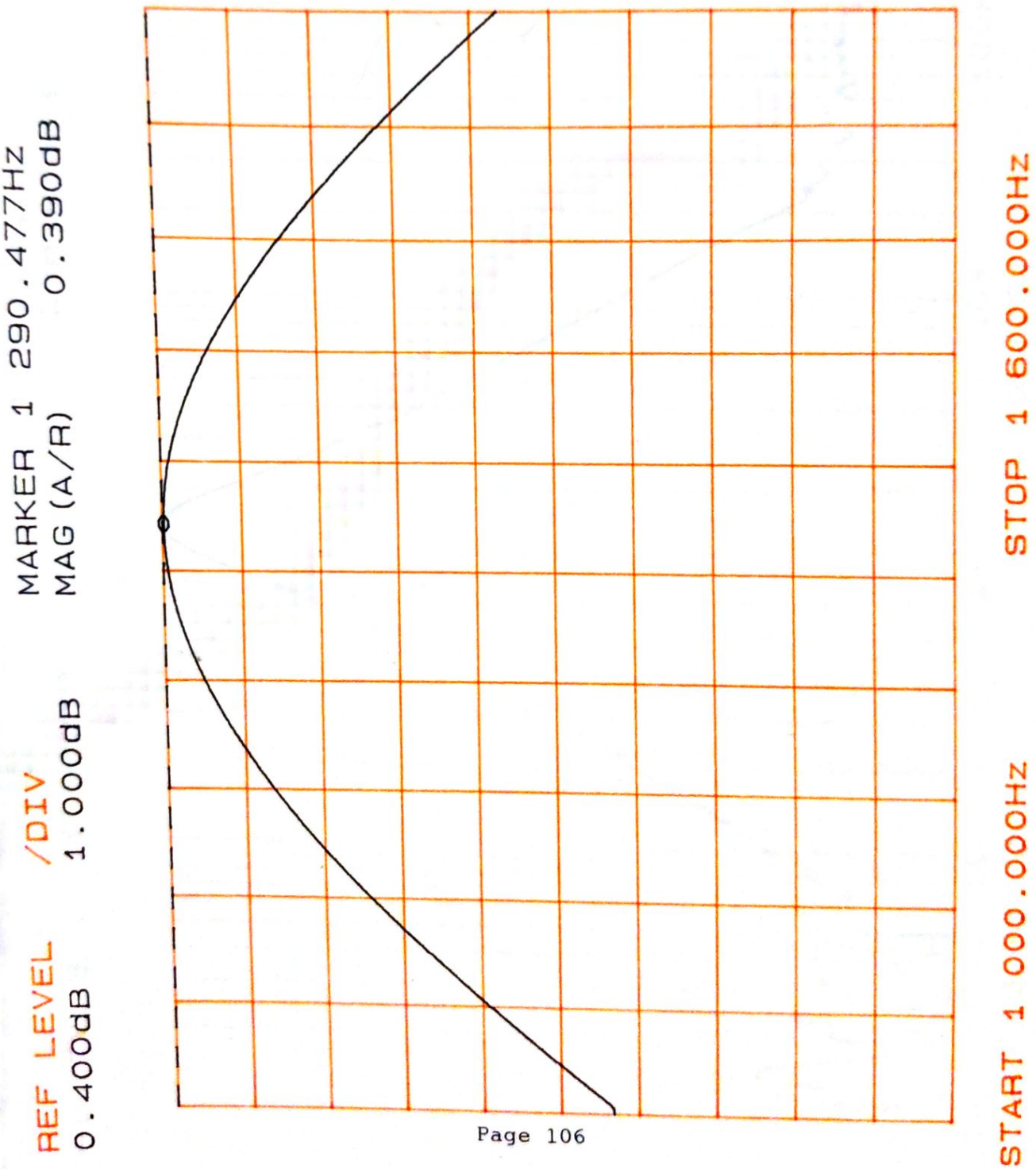
f_0 = 1284 Hz, Q = 1.45 and Resonant gain=-4.2

Frequency plots of the three stage FRIEND filter, as used in for the IF filter block, are shown overleaf.

IF filter response: Centre frequency=1290Hz

Bandwidth=450Hz

Diagram shows response over range 1000 to 1600 Hz



IF filter response: Centre frequency=1290Hz

Bandwidth=450Hz

Diagram shows response over range 10Hz to 100kHz