DEVELOPMENT OF A DIGITISING WORKSTATION FOR THE ELECTRONICS LABORATORY UTILISING THE PERSONAL COMPUTER.

By H.P. Janse van Rensburg

Thesis submitted in part fulfilment of the requirements for the Masters Diploma in Technology to the Department of Electrical Engineering (light current) at the Cape Technikon.

Research and Development Centre
Telkom S.A. LTD.
Cape Town
South Africa
July 1994

DECLARATION

I hereby declare that the contents of this thesis represents my own work and the opinions contained herein are my own. It has not been submitted before any examination at this or any other institution.

H.P. Janse van Rensburg

(Signature)

ACKNOWLEDGEMENTS

On completion of this thesis, I would like to extend my sincere thanks to the following individuals for their assistance and support:

- Mr P.H. Kleinhans for suggesting and commissioning the project concept and under whose guidance this research has been done. Furthermore, to the Cape Technikon for funding the project.

- Mr G.V. Williams and his staff at the Telkom S.A. Research and Development Centre, Cape Town, for the use of their facilities, their moral support and with whom many meaningful discussions were held.

- My fellow student technologists, engineers and fellow colleaques for their encouragement and assistance.

- My family and friends for their understanding, moral support and encouragement throughout the design of this project.

ABSTRACT

This thesis describes the design, development and implementation of a digitising workstation for the electronics laboratory that utilises the personal computer.

OPSOMMING

Hierdie tesis beskryf die ontwerp, ontwikkeling en implimentering van `n gerekenariseerde werkstasie vir die elektroniese laboratorium wat gebruik maak van digitaliserende tegnieke.

CONTENTS

1. INTRODUCTION.

The digitising workstation is a Personal-Computer-based test apparatus capable of digitising analogue signals at a maximum sampling rate of 20MHz. The features of the workstation are comparable to that of a dual channel Digital Storage Oscilloscope (DSO) but at a great reduction in cost. The controlling software allows the use of a mouse to control all operations of the digitising workstation. The output to the screen is in standard VGA resolution. The digitising workstation features manual as well as trigger control and all captured information can be archived to file.

The workstation consists of two main units:

- The analogue input unit
- The digital capturing and control unit

The analogue input unit also contains the analogue to digital converters used to digitise the input signals. The main function of the analogue input unit is to interface the input signal to the digital capturing and control unit. The digital capturing and control unit captures and stores the digitised analogue signal in fast static random access memory (RAM) from where it is available to the PC for processing.

The requirements for the PC are the following:

- IBM compatible PC with 80286 processor or higher
- 640K RAM
- VGA monitor
- Pointing device (mouse)

2. OBJECTIVE.

Sophisticated test equipment are very costly and cannot be afforded by most electronics enthusiasts. The situation in the education sphere is as such that funds are not always available to equip all workstations in the electronic laboratories with suitable test equipment. Sophisticated test equipment are also not always utilised at full potential by the students.

The availability of personal computers to the students is a necessity considering the design tools available to the electronics industry. Ideally all workstations in the electronics laboratories should be equipped with personal computers. It is thus a good idea to utilise these computers as both design platforms as well as for testing purposes.

A digitising workstation utilising the personal computer was suggested having the following features:

- The digitising workstation should have similar features to an oscilloscope
- Two channels
- All operations should be controlled by the keyboard and mouse
- 1 MHz signal resolution
- Storage facilities to file

- Calculations of signal frequency, voltage, and phase

One of the main advantages of this unit is the storage facilities to file. This allows the lecturer to capture and store signals and build up a library that could be used for future reference and demonstrations.

3. BASIC PRINCIPLES OF DIGITAL STORAGE OSCILLOSCOPES (DSOs)

As the digitising workstation should have similar features to a Digital Storage Oscilloscope the basic principle of DSO's needs to be considered.

The benefits of retaining signals long after they have disappeared from the input sockets have already been proved with analogue storage oscilloscopes. Using storage CRTs with special bistable state phosphors, and mesh storage layers (variable persistence and fast transfer), analogue storage scopes feature extremely high-writing speeds and the ability to display many waveforms with good vertical resolution - still at a good performance/price ratio. For example, fast transfer offers a high performance method of capturing single-shot events. Nevertheless, the signal is only briefly available to record with a camera.

Digital storage is forever! The accurate sampling of analogue signals at fixed intervals, for conversion to 'cold calculating' digital numbers, offers many extra advantages. For example, compared to the 3% accuracy of an analogue time-base, a digital time-base is accurate to better than 0.1%. [13] Signals stored in memory as digital words can be recalled time after time without deterioration; this permits manipulation and an excellent quality display on a low-cost

CRT in whatever form required. The storage and display flexibility not only allows such waveform calculations as risetime, peak-peak, but also enables many mathematical functions to be performed; e.g. multiplication of signals, fast Fourier transforms, etc. for signal analysis and comparison. In data system set-ups, signals can be sent via the IEEE 488-bus to a remote computer or X - Y plotter; e.g. for documentation or archiving purposes.

DSOs use a sampling technique to capture the fast analogue signals with high resolution, the maximum sampling frequency determining the frequency of signal that can be captured.

3.1. Sampling Techniques

Primarily, DSOs can be used to capture displayed waveforms and store them indefinitely, especially single events or pulse bursts, without the tedious problems of taking photographs at the precise moment of interest.

In early models, the benefits of DSOs came at a high price, and even then the performance was not particularly good. The problem was to find efficient circuit devices to capture very fast signals: previous high-speed data sampling techniques had limited picture resolution.[13]

Sampling, or taking parts of a signal to get a picture of the whole, is a technique used to piece together signals that cannot be displayed directly. The more samples taken, the more accurate the picture!

3.1.1. Real-Time Sampling

A one-time event obviously needs to be sampled at a single sweep. At mid-frequencies (up to about 50MHz), real-time sampling is used for single-sweep acquisition.

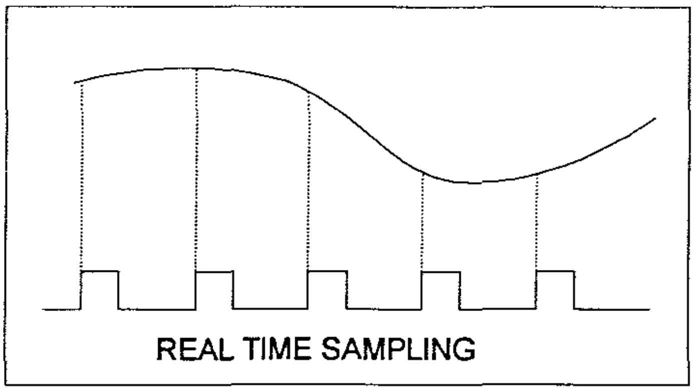


Figure 3.1

Here, the signal is sampled by successive data points taken at equal time intervals until the end of the sweep. The samples are then reconstructed on the display as a matrix formed by the desired time and voltage points. The limiting factor in real-time sampling is that the sample rate must be high compared with the incoming signal frequency.

3.1.2. Equivalent-Time Sampling

For fast repetitive signals, samples can be taken at various times over different sweeps to fill the waveform memory. It can be compared to a series of quick looks at various parts of a waveform because time doesn't allow one single look at the whole. The separate images are then put together to complete the picture as a continuous time trace. Two methods of equivalent-time sampling are generally used: sequential and random sampling.

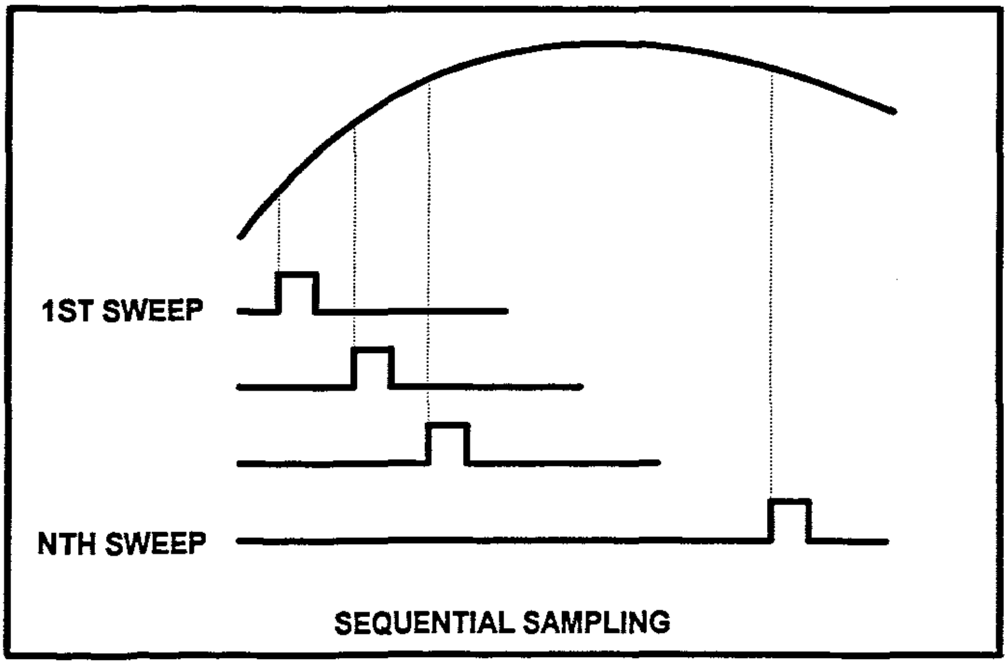


Figure 3.2

3.1.3 Sequential Sampling

Here, one point along the waveform is sampled per sweep until enough points are aquired by the memory. The number of sweeps needed to capture a signal will therefore depend on the number of points required to fill the memory.

Because of the fixed time relationship between sample and trigger point, sequential sampling is not suitable for showing the rising edge of short widely-spaced pulses.

3.1.4. Random Sampling

This is ideal for displaying fast rising edges, as the sample is taken on random cycles of the oscilloscope clock.

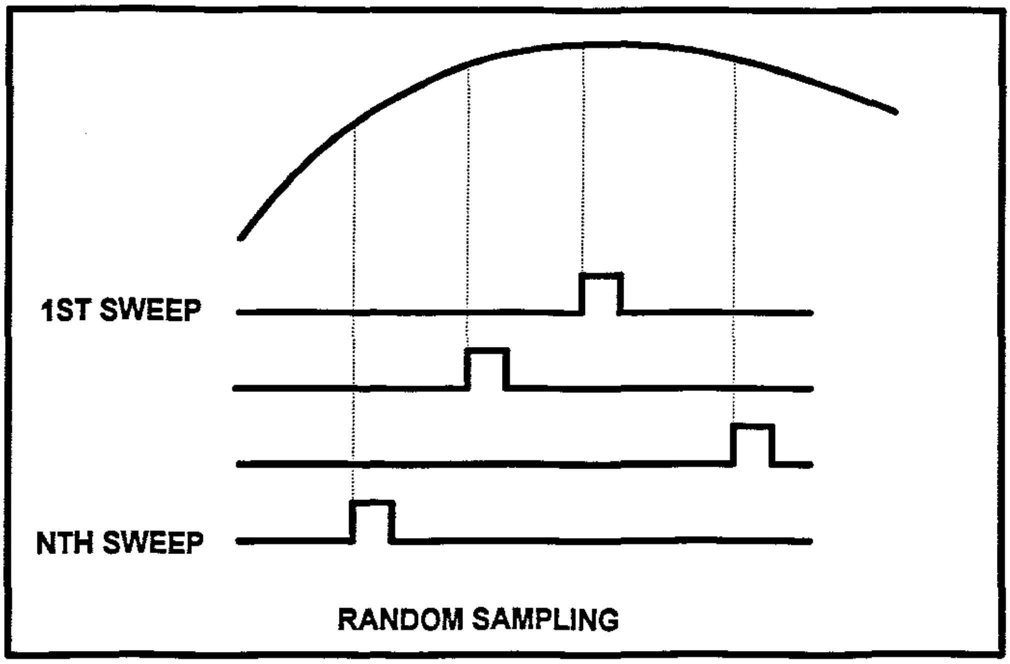


Figure 3.3

Random sampling occurs once per sweep, each point being stored in a memory location referenced to the trigger point. The advantages of this storage technique with fixed reference are pre- and post- triggering and jitter-free displays, discussed in section 3.3.4.

3.1.5. Slowing down the signal

As high frequency linear amplifiers are difficult and expensive to build, the DSO technique is to sample the high frequency signals and then reduce their frequency to enable them to be processed through reliable, linear amplifiers before accurate conversion to digital values using an analogue-to-digital converter. The resulting digital word equivalents are then reconstructed to form a continuous display.

3.1.6. Accuracy and resolution

Accuracy of signal reproduction depends on how well an instrument can resolve or distinguish signal differences within its specified accuracy. Although a DSO does not have the 'infinite' resolution of an analogue oscilloscope, its measuring power and storage capabilities are much more extensive. High-frequency performance is comparable because of the advantages in fast-sampling techniques already mentioned - some DSOs for example capture signals with frequencies as high as 2 GHz.

3.1.7. Roll Mode

In contrast to the fast signal techniques, lower frequency signals can also be captured and stored in memory. Because a DSO time-base is derived from a crystal source, it can be accurately divided to capture slow signals such as temperature drift over periods of several hours using time-base speeds of up to 1 hour/div. In this way, the DSO effectively acts as a line recorder. This capability is often called the ROLL mode as the waveform slowly rolls across the screen from right to left.

3.2. Analogue-To-Digital Conversion

To appreciate the advantages of a DSO it is useful to realise the importance of the ADC in determining the overall performance. The ADC specifies two key factors - sample rate and vertical resolution, both affecting accuracy of conversion.

3.2.1. Sample Rate

This determines the maximum frequency of the input waveform that can be sampled and reconstructed. The sampling clock frequency must be as high as possible, since it determines the time interval between points plotted for the matrix display.

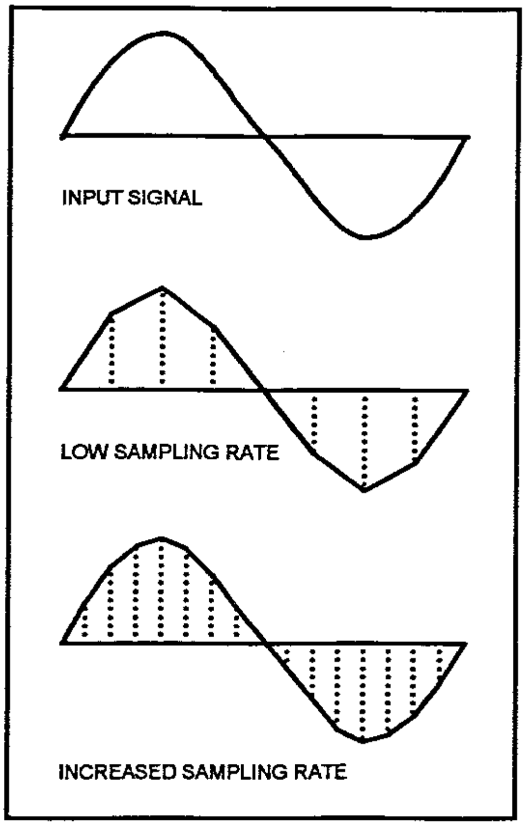


Figure 4.3

For good resolution of single-shot events, the clock frequency should be between four and ten times the signal speed, depending on wave shape. For instance, to look at a transient in some detail, ten samples may be needed. However, checking the presence of a repetitive signal, say a sine wave, will probably only require four samples per cycle.

The sampling theorem by Harry Nyquist places restrictions on the frequency content of the time function signal, f(t), and it can be simply stated as follows:

> **In order to recover the signal function f(t) exactly, it is necessary to sample f(t) at a rate greater than twice the highest frequency component. [12]**

This sampling rate of twice the highest frequency component, is refered to as the Nyquist rate.

The consequences of sampling a signal at a rate below twice its highest frequency component results in the phenomenon known as aliasing, overlapping or spectral folding.

3.2.1.1. Aliasing

A signal with a bandwidth, Fa, must be sampled at a rate, Fs > 2Fa in order to avoid loss of information. If the number of samples of a signal, F, is inadequate, i.e., Fs < 2F, a phenomenon called aliasing, inherent in the spectrum of the sampled signal, will cause a frequency equal to Fs - F, called an "alias" to appear in the signal band (of frequencies below Fs/2). For example, if Fs = 4kHz and F = 3kHz, a 1kHz alias will appear.**[11]**

Since noise is also aliased, it is essential to provide low-pass filtering prior to the sampling stage to prevent high frequency noise on the signal line from being aliased into the signal range.

The danger of aliasing is that a stable trace may be displayed at several time-base speeds, corresponding to different harmonics.

Aliasing errors can be reduced in a DSO by switching special filters into circuit at each time-base speed to remove input frequencies above twice the sampling frequency at that setting. However, these filters are expensive and will attenuate high-frequency transients that may need to be observed.

Often, a DSO incorporates a high-frequency detector with a visual warning indicator. To avoid aliasing errors, the rule-of-thumb is to start sampling at the highest sample rate or time-base speed and work down.

3.2.2. Vertical Resolution

This determines the detail that can be displayed. The higher the resolution, the nearer the reconstruction is to the original signal. The waveshape information improves as the vertical resolution increases.

3.2.3. Vertical levels

All the vertical levels of a signal must be represented as digital numbers, therefore the higher the resolution, the longer the binary word required. A single bit word length (2^1) offers two levels and would be suitable to represent signals of a rectangular nature where the levels are 0 and 1. Each extra bit doubles the resolution so a 2-bit word (2^2) could be used to represent four vertical levels, for example, 0, 0.25, 0.5, 0.75 and 1. A 3-bit converter would have 8 levels. For most applications, eight bits is seen to be an acceptable resolution, although for some specific areas of use this would be too coarse.

3.2.4. Vertical deflection sensitivity

An 8-bit converter gives 2^8 or 256 possible levels while a 10-bit converter gives 2^{10} or 1024 possible levels.

$$\%RESOLUTION = \frac{1}{2^N - 1} \times 100$$

$$\begin{aligned} \therefore \%RES(8-BIT) &= \frac{1}{2^8 - 1} \times 100 \\ &= 0.4\% \end{aligned}$$

$$\begin{aligned} \therefore \%RES(10-BIT) &= \frac{1}{2^{10} - 1} \times 100 \\ &= 0.1\% \end{aligned}$$

Besides sample rate and vertical resolution, digital sensitivity is also affected by the vertical deflection coefficients of a DSO, referred to in mV/div, and by its particular full scale reference. For instance, is full scale over 8 or 10 divisions or is it a specified voltage window? Remember however that a vertical deflection of a 2 mV/div on an 8-bit DSO is not as sensitive as a 5mV/div on a 10-bit DSO.

The 10-bit DSO has four times more resolution than the 8-bit DSO, so:

$$\textbf{Sensitivity increase} = \frac{2}{5} \times 4 = 1.6$$

3.3. Conversion Techniques

Two methods of sampling and converting analogue waveforms to digital are generally used in DSOs: semiconductor 'flash' converters or the charged-coupled devices (CCDs).

3.3.1. Flash Converters

Basically, a flash converter uses an array of comparators, one less than the number of vertical decision levers (2^n-1). So a 10-bit ADC will have (2^n -1) 1024-1 comparators.
Each comparator compares the input signal with a reference level set by a resistor chain to give a 0 or 1 output. The results are then decoded (256 for an 8-bit converter) to give a digital word. Conversion speeds can be extremely high, but high vertical resolution demands very accurate resistors in the divider chain.

Around 300 MS/s, flash converters not only become expensive but so do the memory devices. Even memories with access times of 25 NS, which relate to 40 MS/s, are too expensive for low-cost DSOs. Digital multiplexing and interlaced memories can overcome these limitations, but demand expensive circuit designs. Another problem is that the restricted input band-width of flash converters limits the upper frequency of capture. This can be avoided by using sample-and-hold gates to extend the input bandwidth of the ADC.

A sample-and-hold gate captures and holds an analogue signal at a specific point in time, controlled by a timing strobe. The signal is fed into a high input impedance amplifier. The resulting current is switched to charge up a capacitor. The stored voltage is held for analogue-to-digital conversion until the next sample is switched in.

3.3.2. Charge-Coupled Devices (CCDs)

A charge-couple device consists of micro semi-conductor capacitors, often referred to as 'wells', in which charges are stored and sequentially transferred from one well to the next. The device acts as an analogue shift register on a first-in, first-out basis. In effect, it is a chain of sample-and-hold gates similar to the ones described, along which the charge ripples, whilst maintaining separation from the next. Sampling is very fast. When all the cells are full, the charge is an analogue replica of the signal that can be read out slowly for conversion to digital form.

3.3.2.1. Multiplexing CCDs

Although a CCD gives high performances at a much lower cost than an equivalent flash converter, CCDs are generally limited to less than 1000 samples long because of cumulative signal degradation from cell to cell (droop effect).

As with flash converters, CCDs can be multiplexed to increase sample rate and memory depth. Some DSOs multiplex up to thirty-two CCDs to give effective sample rates of 1.3 GS/s and memory depths of over 10 k-words, but at a cost.

3.3.2.2. CCD parameters

The most important CCD parameters are the leakage current, transfer efficiency (percentage of electrons transported from one well to the next), speed and size of charge packet. The size of charge packet determines the signal-to-noise ratio. Several types of CCD have been developed to improve these parameters.

3.3.4. Post- and Pre-triggering

Analogue oscilloscopes are obviously limited to triggering and displaying one screen of the waveform. However, one of the great benefits of a DSO is its ability to acquire data on both sides of the trigger point.

Post-triggering enables it to delay acquisition until after the trigger point by many screens (1000 for some DSOs), thereby effectively increasing the memory size. This means that any part of the wave-train can be selected for viewing without the need (and cost) for large memories.

And DSOs have the powerful acquisition capability of displaying pre-trigger information since the input waveform is continuously recorded by the CCD. If acquisition is stopped on receipt of a trigger, the CCD is already holding pre-trigger data and this is selectable. For example, if the DSO triggered on a fault, it is possible to observe the conditions that led up to the fault.

3.3.5. Glitch-catching

As seen by a DSO, a glitch is the blind spot that falls between samples. The chances of capturing these fast transients are remote if the TIME/DIV setting for the displayed waveform is well below that required for the glitch frequency. Some DSOs incorporate a MIN/MAX mode using analogue peak detectors, which is useful for displaying fast glitches or indicating modulation envelopes. The TIME/DIV setting can be selected to display the whole of the signal you need, while the envelope-mode display is built-up at a much faster digitising rate.

4. ANALOGUE TO DIGITAL CONVERSION

There are many methods by which an analogue signal can be converted into a digital signal, with varying conversion rates, costs, and susceptibility to noise. In this section the major methods in use will be examined; ramp, dual-slope integration, successive approximation and parallel conversion.

4.1. Ramp Conversion

Ramp conversion is the least expensive and slowest method of converting analogue information to digital. It is an ideal method to use in a digital voltmeter where the number of conversions required per unit time is minimal.

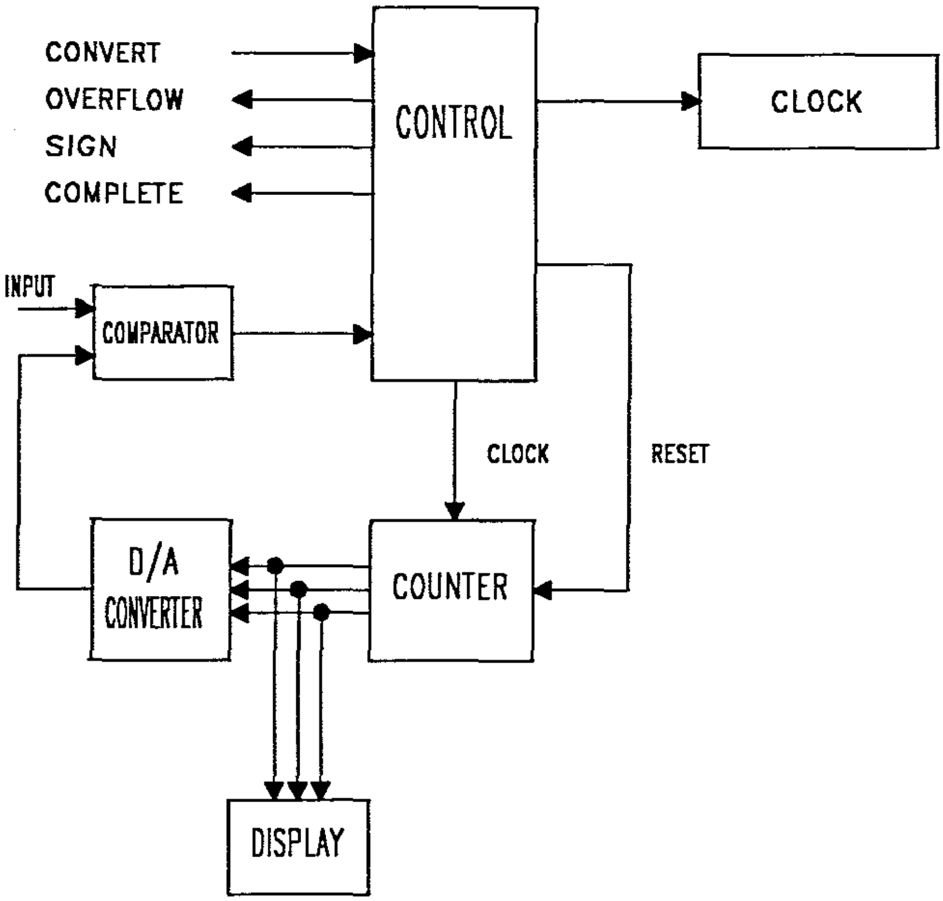


Figure 4.1

Figure 4.1 is a block diagram of a ramp conversion A/D converter. The input voltage to be measured is fed to the voltage comparator: Upon receipt of the convert signal, the control resets the counter to 0 and then supplies clock pulses to the counter. The binary output of the counter is fed to the D/A converter, which outputs an analogue voltage in response to its digital input. This analogue voltage is then fed to the voltage comparator that compares the output of the D/A converter with the analogue input. As soon as the

D/A input exceeds the input voltage, the comparator signals the control circuit, and it stops the counter. The binary number in the counter then represents the voltage of the input signal. The control circuit will also output a polarity signal, indicating whether positive or negative, and an overflow signal if the input signal exceeds the highest possible voltage of the D/A converter.

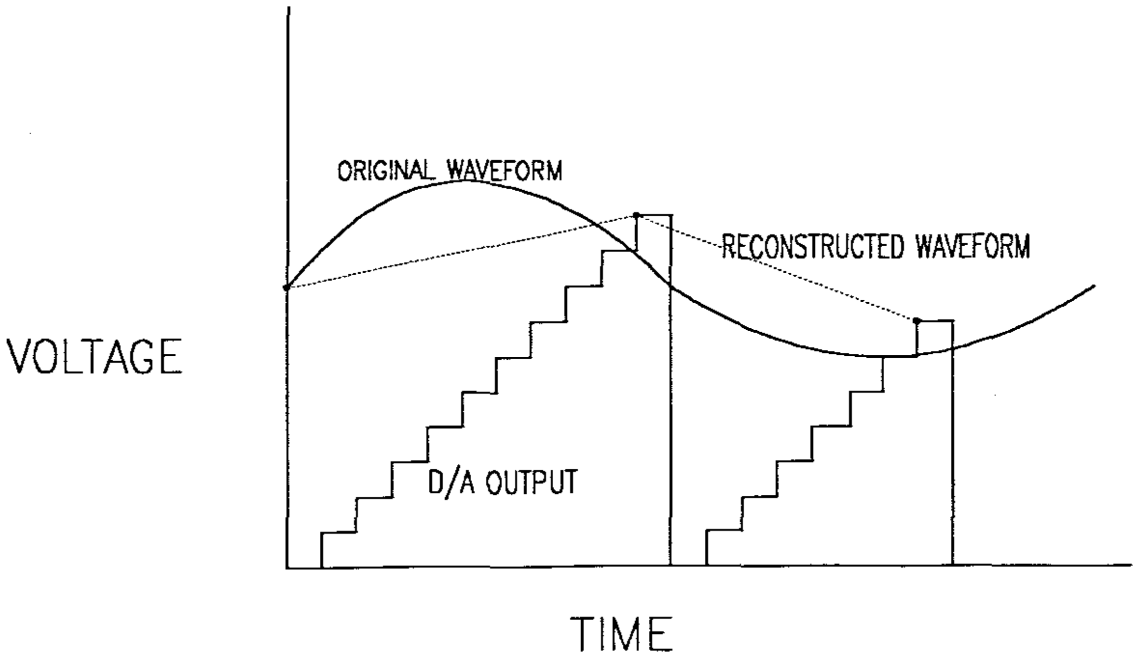


Figure 4.2

Figure 4.2 illustrates a waveform converted using this method. As can be seen, most of the time between samples is taken up by the counter. Because of this, only three samples can be taken over the entire waveform. If these samples were then converted back to analogue, the resultant waveform would be shown by the dashed lines.

A modification of the converter results in much improved operation. By making the counter an up/down counter, it can proceed from its previous analogue point to the next analogue point without having to reset. Figure 4.3 illustrates a waveform converted by this method. Note that this results in many more samples, better representing the original signal.

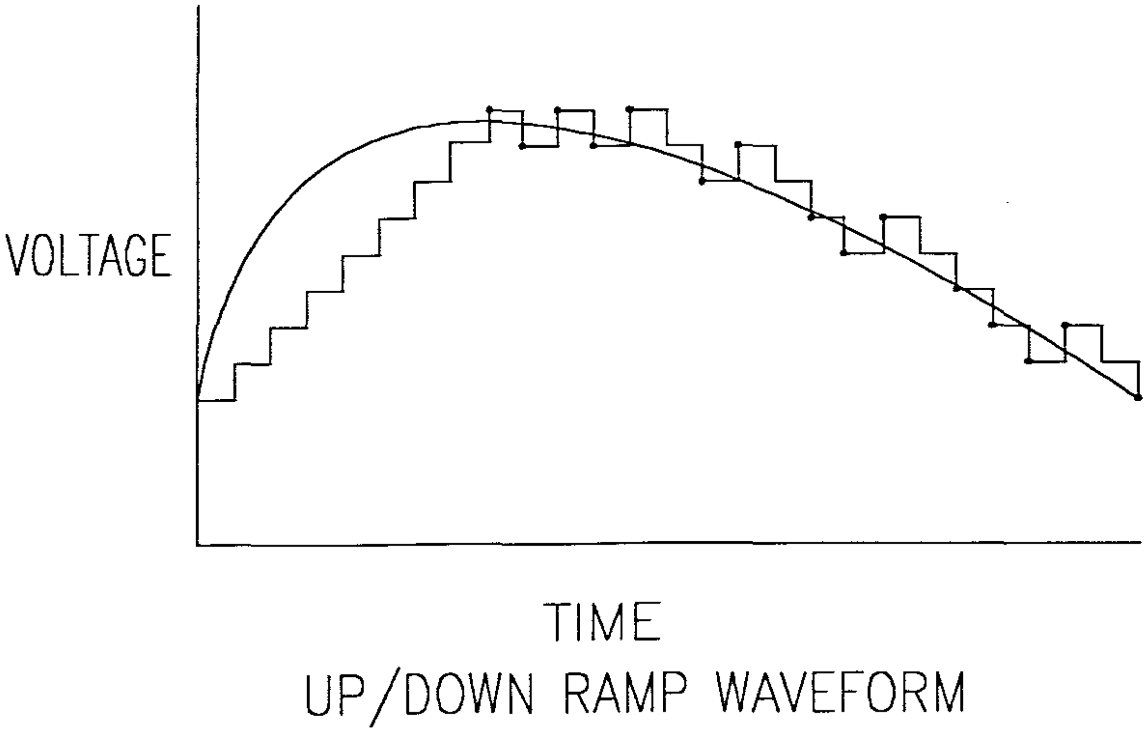


Figure 4.3

4.2. Dual-Slope Integration

One of the disadvantages of the ramp converter is its susceptibility to noise. This problem is overcome by the dual-slope method (Figure 4.4). The core of this system is the integrating amplifier, an operational amplifier connected as an integrator.

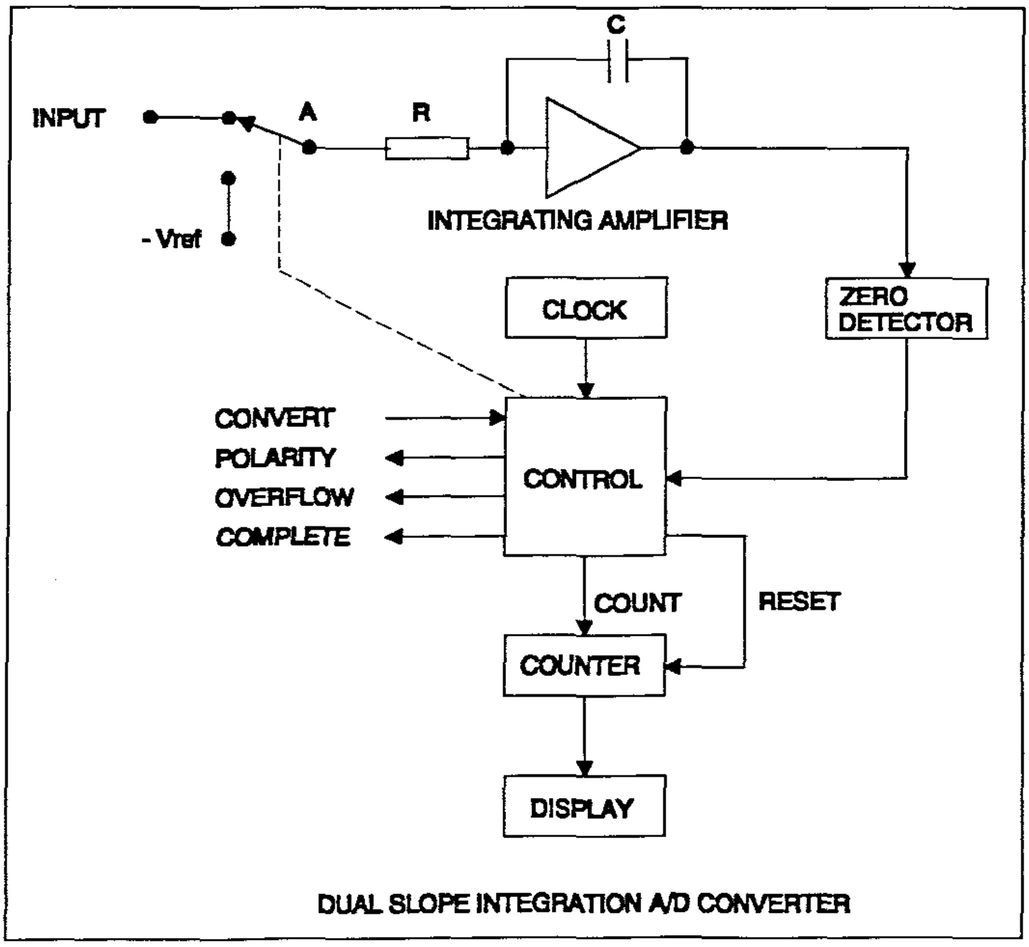


Figure 4.4

If point A of the amp was at +1v and R were 1kΩ, then the current through R would be 1mA. But this current can only come from the capacitor, since the input of the amp is assumed to have infinite impedance. However, the formula for the charge on a capacitor is

$$Q = CV$$

where Q is charge in coulombs (C), C is capacitance, and V is voltage. But observe that current is defined as charge per unit time (coulombs per second). If 1A is flowing through a circuit, this represents 1 C/s. If Q is increasing at 1C/s (thus, 1A is flowing), then the voltage must be increasing at some linear rate. Therefore, constant current through a capacitor will result in a ramp voltage across that capacitor.

From the foregoing discussion, it can be concluded that when the integrating amp is connected to a voltage source, it will produce a ramp voltage at its output. Upon receipt of a convert signal, the amp is connected to the input for a predetermined length of time. The counter is then turned on, and the amp is switched to the -ref voltage, resulting in current flow in the opposite direction through R, discharging C. This produces a negative ramp at the amp output, ultimately crossing through 0V. When 0V crossover is detected, the counter is stopped and its binary value represents the input voltage.

The dual-slope integration method has the advantage of low cost, relatively short conversion time (approximately 50ms), and good noise immunity, since the input is really an average of the waveform while it is connected to the op amp.

4.3. Successive Approximation

With the advent of computers, shorter conversion times were required - on the order of microseconds, rather than milliseconds. The successive approximation method was devised to meet the challenge.

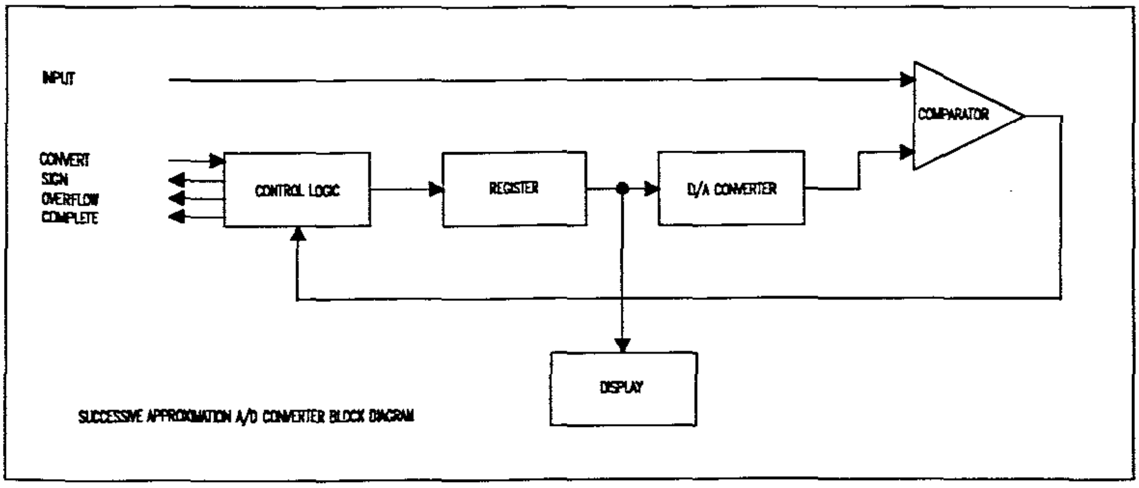


Figure 4.5

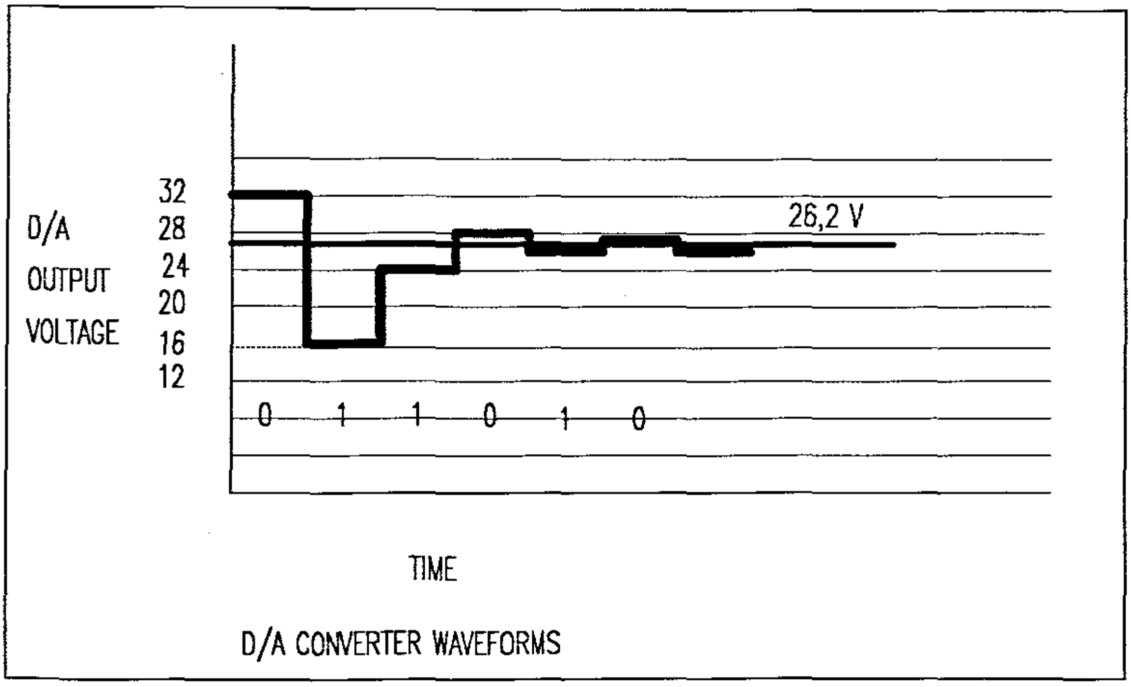


Figure 4.6

Figure 4.5 is a block diagram of the system, and Figure 4.6 illustrates the output of the D/A converter compared with the input voltage. Assume a five-bit conversion must be made from a 26.2V signal using a maximum reference voltage of 64V.

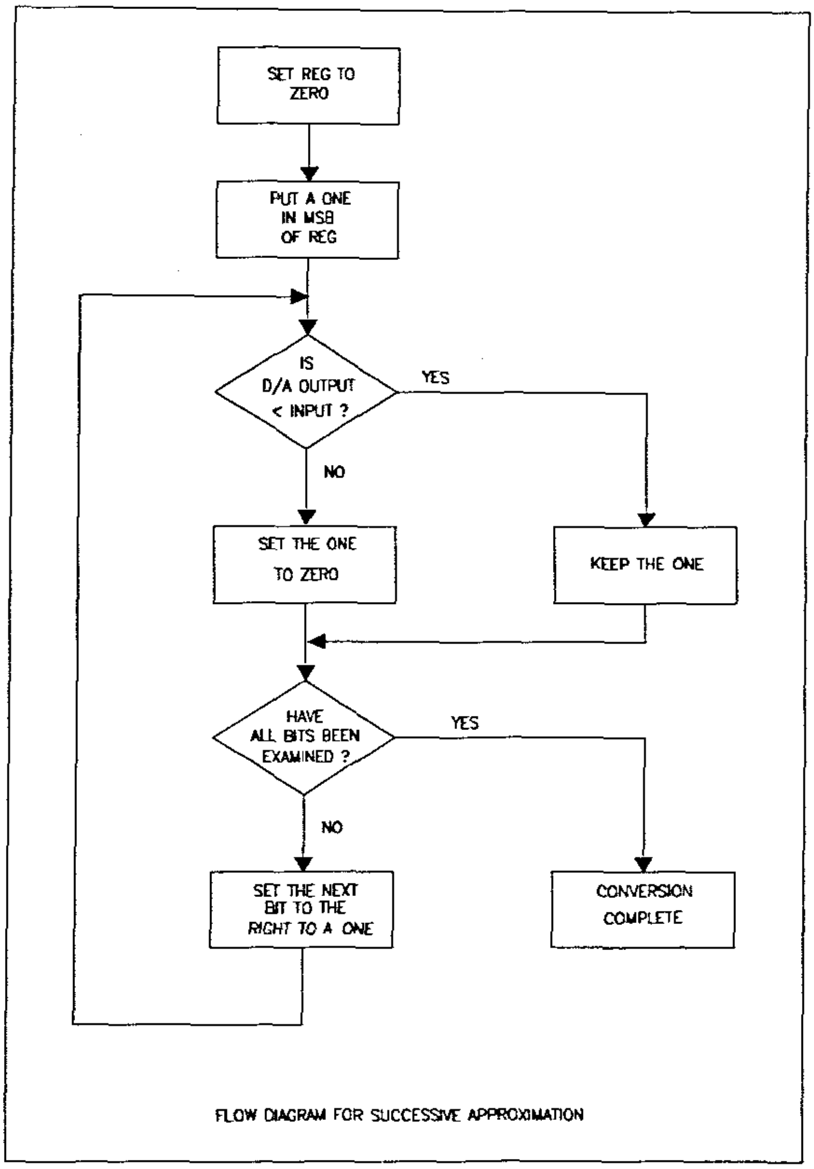


Figure 4.7

Referring to the flow diagram, the register is first set to 0, then a 1 is placed in its MSB. This is converted by the D/A and fed to the comparator where it is compared with the input voltage. If the D/A output exceeds the input, the bit is set to 0; if the D/A is less than the input, the 1 is retained in the register. In this case, the output of the D/A is 32V, resulting in this bit being set to 0.

Next, a 1 is placed in the next bit to the right, resulting in an output Of 16v from the D/A. Since this is less than the input, the 1 is retained in this bit position. Placing a 1 in the third position results in a 24V output, again less than the input. Therefore, the 1 is retained. Placing a1 in the fourth position results in an output of 28V, exceeding the input. Therefore, bit 4 is set to 0. A 1 in bit 5 results in 26V, less than the input; it is therefore retained. A 1 in bit 6 results in 27V output, exceeding the input; it is therefore set to 0. Thus, the resultant binary word is 011010, representing 26V.

The successive approximation method is very fast; many units convert in less than 250nS/bit. However, it is more expensive than the ramp methods.

4.4. Parallel Conversion

Parallel conversion is the easiest of all the conversion methods. It is also the fastest and most expensive. Figure 4.8 illustrates this method for a three-bit A/D converter.

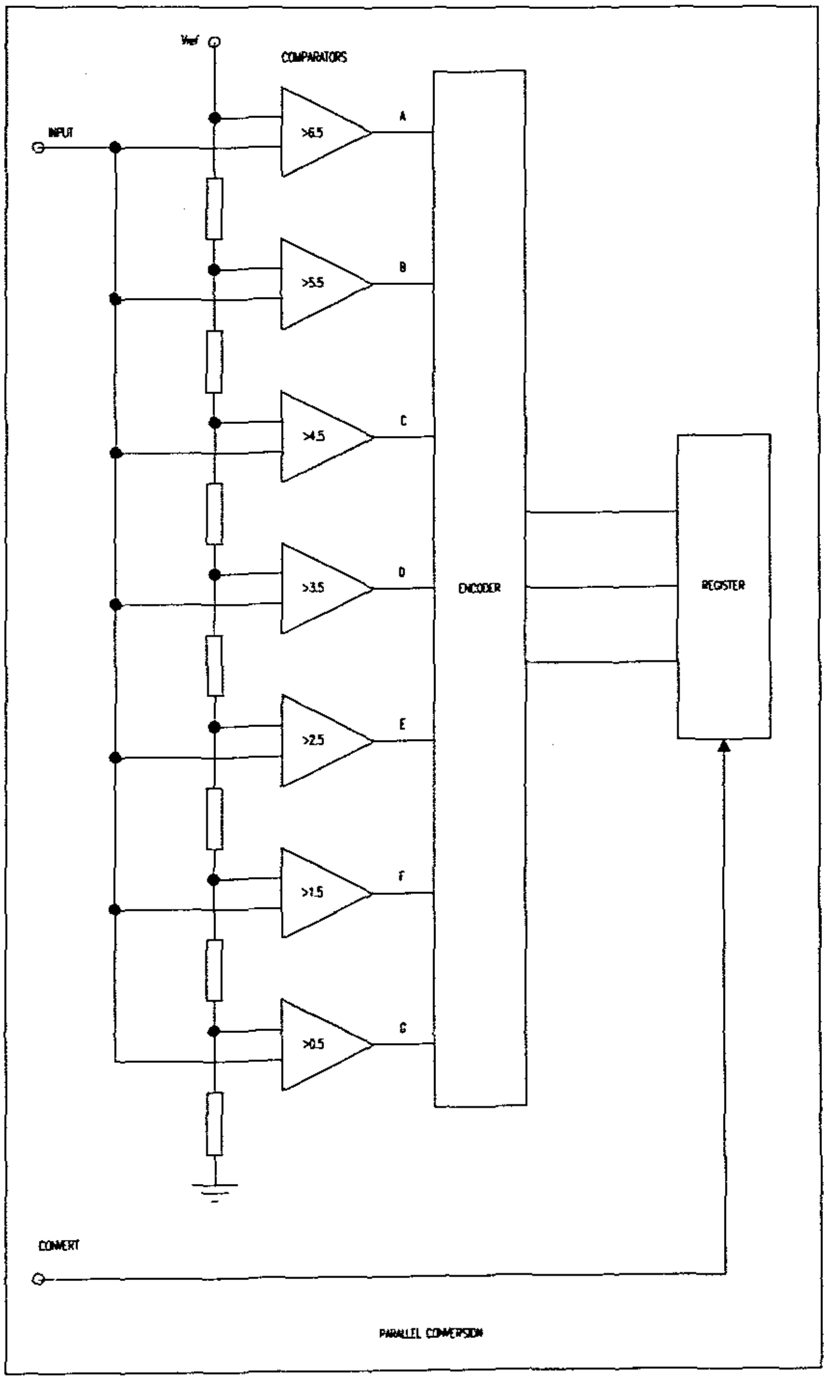

Vref
COMPARATORS
INPUT
>6.5
A
>5.5
B
>4.5
C
>3.5
D
>2.5
E
>1.5
F
>0.5
G
ENCODER
REGISTER
CONVERT
PARALLEL CONVERSION

Figure 4.8

The input is fed to all seven voltage level comparators. If the input were 3.23V, the outputs of comparators A, B, C, and D would all be 0's, and those from E, F, and G would be 1's. The encoder would then translate this into a three-bit code and clock it into the register.

The big disadvantage of this system is the cost, for an N-bit converter requires $2^n - 1$ comparators. Therefore a 10-bit converter would require $2^{10} - 1$ or 1023 comparators. However, in applications requiring little resolution and great speed, it is very useful.

5. HARDWARE IMPLEMENTATION

In order to obtain 1 MHz resolution a sampling rate of at least 10 MHz is necessary, giving 10 points from which to reconstruct the signal. Any increase above 10 MHz would obviously improve the signal clarity. Resolution of 8 bits or better is preferable, but 7 bits will yield 128 levels. Considering the resolution of standard VGA (640 x 480), 128 levels will produce a signal that will cover more than a quarter of the screen height. If the space occupied by the menu bar and the second channel is taken into account, 7 Bits resolution will be sufficient.

The device chosen for the A-D conversion is the ADC 207-MC (DATEL). These A-D converters operate at sampling rates of 20MHz and has seven bit resolution.

5.1. ADC-207 Video Flash Converter (DATEL)

5.1.1. General Description

The ADC-207 is the industry's first 7-bit flash converter using a high-speed 1.2 micron CMOS process. This process offers some very distinctive advantages over other processes, making the ADC-207 a unique device. The smaller geometrics of the process achieves high-speed, better linearity and better temperature performance. Since the ADC-207 is a CMOS device, it also has very low power consumption (250 mW). The device draws power from a single +5V supply, and is conservatively rated for 20 MHz operation. The ADC-207 allows using sampling apertures as small as 12nS, making it more closely approach an ideal sampler. The small sampling aperture also allows the device to operate at frequencies greater than 20 MHz.

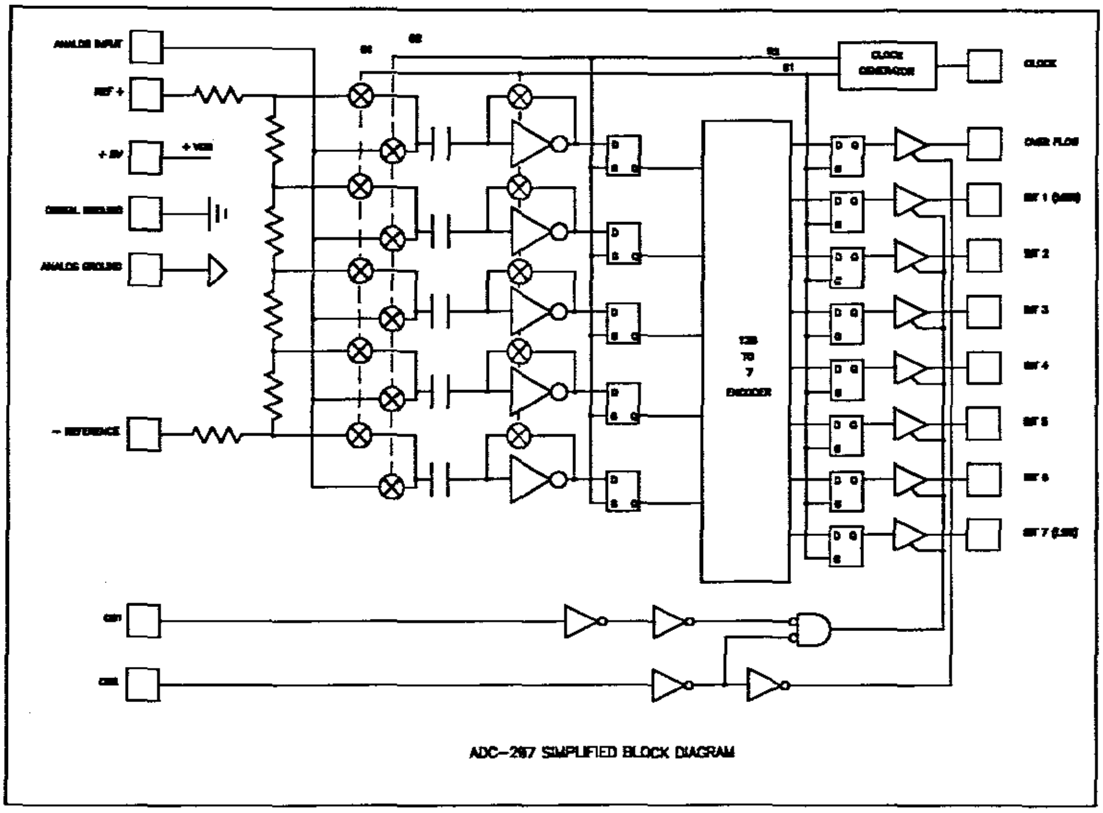
The ADC-207 has 128 comparators which are auto-balanced on every conversion so as to cancel out any offsets due to temperature and / or dynamic effects. The resistor ladder has a midpoint tap for use with an external voltage source to improve integral linearity beyond 7 bits. The ADC-207 also provides the user with 3-state outputs for easy interfacing to other components. There are two models of the ADC-207 covering the two operating temperature ranges, 0 to 70 degrees C and -55 to +125 degrees C.

5.1.2. Theory Of Operation

The ADC-207 uses switched capacitor scheme in which there is an auto-zero phase and a sampling phase. Figure 5.1 shows the simplified block diagram of the ADC-207. The ADC-207 uses a single clock input. When the clock is at a high state (logic 1), the ADC-207 is in the auto-zero phase (01). When the clock is at the low state (logic 0), the ADC-207 is in the sampling phase (02). During phase 1, the 128 comparator outputs are shorted to their inputs through CMOS switches. This serves the purpose of bringing the inputs and outputs to the transition levels of the respective comparators. The inputs of the comparators are also connected to 128 sampling capacitors. The other end of the 128 capacitors are also shorted to 128 taps of a resistor ladder, via CMOS switches. Therefore during phase 1 the sampling capacitors are charged to the differential voltage between a resistor tap and its respective comparator transition voltage. This eliminates offset differences between comparators and yields better temperature performance. During phase 2 (02) the input voltage is applied to the 128 capacitors, via CMOS switches. This forces the comparators to trip either high or low. Since the comparators during phase 1 were sitting at their transition point, they can trip very quickly to the correct state. Also during phase 2 the outputs of the comparators are loaded into internal latches which in turn feed 128 to 7 encoder. When going back into phase 1 the output of the encoder is loaded into an output latch. The latch then feeds the 3-state output buffer. This means that the ADC-207 is of pipeline design. To do a single conversion, the ADC-207 requires a positive pulse followed by a negative pulse followed by a positive pulse. Continuous conversion requires one cycle/sample (one positive pulse and one negative pulse). The 3-state buffer has two enable lines, CS1 and CS2. CS1 has the function of enabling/disabling bits 1 through 7. CS2 has the function of enabling/disabling bits 1 through 7 and the overflow bit. Also a full-scale input produces all ones, including the overflow bit at the output. The ADC-207 has an adjustable resistor ladder string. The top end, middle point, and bottom end are brought out for use with application circuits. These pins are called +Ref, MID POINT, and -Ref, respectively. In typical operation +Ref is tied to +5v, -Ref is tied to ground, and MID POINT is bypassed to ground. Such a configuration results in a 0 to 5V dc input voltage range. The MID POINT pin can also be tied to a 2.5V source to further improve integral linearity. This is usually not necessary unless better than 7 bit linearity is needed.

Table 1. Chip Select Truth Table

CS 1	CS 2	Bits 1 - 7	Overflow Bit
0	0	3 State mode	3 State Mode
1	0	3 State Mode	3 State Mode
0	1	DATA Output	DATA Output
1	1	3 State Mode	DATA Output



ADC-207 SIMPLIFIED BLOCK DIAGRAM

Figure 5.1

5.1.3. Technical Notes

1. Input Buffer Amplifier - Since the ADC-207 has a switched capacitor type input, the input impedance of the 207 is dependent on the clock frequency. At relatively slow conversion rates a general purpose type input buffer can be used: at high conversion rates DATEL recommends either the HA-5033 or the LH-0033.

2. Reference Ladder - Adjusting the voltage at +Ref adjusts the gain of the ADC-207. Adjusting the voltage at - Ref adjusts the offset or zero of the ADC-207. The midpoint pin is usually bypassed to ground through a 0.1uF capacitor, although it can be tied to a precision voltage halfway between +Ref and - Ref. This would improve integral linearity beyond 7 bits.

3. Clock Pulse Width - To improve performance above 20 MHz, the clock should be adjusted so that the negative portion of the clock signal is 15nS wide. This makes the ADC-207 sample over a shorter period of time thereby more closely approach an ideal sampler.

5.1.4. Application

Figure 5.2 shows typical connections for using the ADC-207. In this configuration the input voltage range is 0 to 5V dc. The input voltage range is determined by the reference voltage. For operating in lower input voltage ranges, the reference input must be tied to the corresponding lower voltage value. For example to operate the ADC-207 in 0 to 3V input range the +REF input must be tied to +3V dc. Further, for higher speed operation (above 20 MHz) users may modify the clock signal to facilitate duty cycle adjustment.

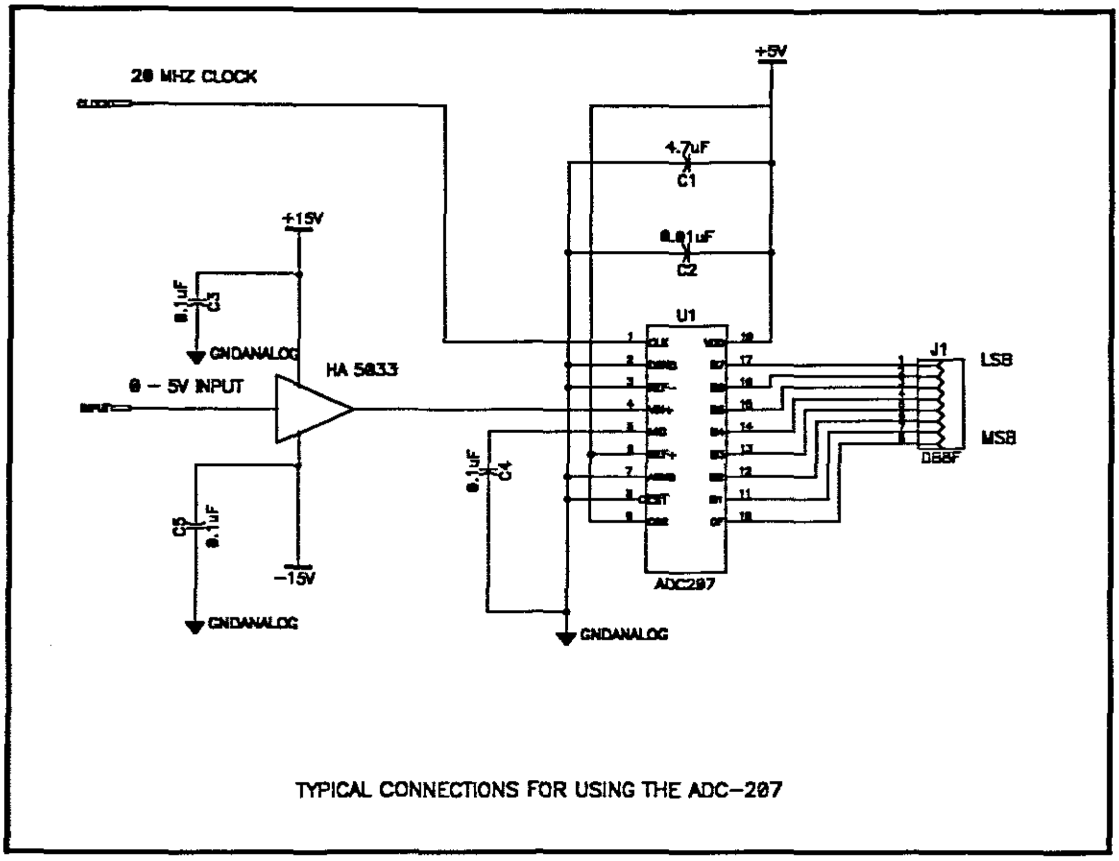


Figure 5.2

5.2. Buffer Memory

The Random Access Memory (RAM) used in most Personal Computers (PCs) have access speeds of about 70 nS. The maximum write speed can be calculated as follows:

$$\text{Speed} = \frac{1}{70\text{ns}} = 14.28\text{MHz}$$

The conversion rate of the A-D converter is 20MHz, thus exceeding the maximum rate of the RAM available in the PC.

In order to store samples at rates of 20MHz and higher, memory devices operating at 25 nS become a necessity. FIFO memory devices are not readily available operating at the required speeds so high-speed static RAM devices were investigated. Static RAM devices with a 20nS rating are available to the computer industry and is most suitable for this application.

The static RAM devices used are 5C6408-20 from MICRO TECHNOLOGY. They are organised as 8192 words of 8 bits each. Two static RAM devices are used - one for each A-D converter.

Usually, the rate at which samples are written into the RAM stays constant while the sampling rate of the A-D converter is changed to allow for different sampling rates. At low sampling rates successive address locations in RAM will contain the same data thus wasting the capacity of the RAM. In the case of the digitising workstation the sampling rate of the A-D stays constant at 20MHz. The addressing rate of the RAM is varied to allow for the sampling rates needed. At high sampling rates the result is the same as with conventional designs, but at low sampling rates only selected samples are written into memory. Multiple addresses containing the same data are thus eliminated thus improving on the usage of available storage space.

The improvement in the use of storage space reduces the RAM size requirements dramatically. The digitising workstation uses 8 Kilobytes of static RAM per channel as opposed to several Megabytes used in other designs. When the 8 Kilobytes of data is displayed, 13 screen lengths or pages, can be scrolled through before the end of the RAM is reached.

The data stored in RAM needs to be available to the PC for display. Several options exist for transferring the data to the PC, but the most suitable option was to memory map the external RAM as part of the internal RAM of the PC. The following section describes the different memory configurations found in the PC as well as the mapping option used.

There are three basic configurations of memory:

- Conventional
- Extended
- Expanded

In addition, most systems have an upper memory area.

5.2.1. Conventional Memory

Conventional memory is the basic type of memory found on all computers. Most computers have at least 640 kilobytes of conventional memory. Programs can use conventional memory without the special instructions needed to use other types of memory.

MS-DOS uses some conventional memory. The device drivers and commands listed in the CONFIG.SYS and AUTOEXEC.BAT files use additional conventional memory. The memory left over is available for other programs.

5.2.2. Extended Memory (XMS)

One way to add more memory to the system is to install extended memory. Extended memory is available only with 80286 or higher processors.

Most programs that use conventional memory cannot use extended memory because the number of addresses that identify locations in extended memory to programs are beyond the addresses most programs can recognise. Only the addresses in the 640K of conventional memory are recognised by all programs.

Programs need special instructions to recognise the higher addresses in extended memory. Extended memory is fast and efficient for programs that can use it. However, many programs are not designed to use extended memory.

To use extended memory efficiently, a program should be installed called an extended-memory manager. An extended memory manager prevents different programs from using the same part of extended memory at the same time. The extended-memory manager also makes it easier for programs to use extended memory.

5.2.3. Expanded Memory (EMS)

Another way to add memory in excess of 640K is to install expanded memory. Most computers can accommodate expanded memory, which consists of two parts: an expanded-memory board, which must be installed on the computer; and a program called an expanded-memory manager, which comes with the expanded-memory board.

A program designed to use expanded memory does not have direct access to the information in expanded memory. Instead, expanded memory is divided into 16K segments called pages. When a program requests information that is in expanded memory, the expanded-memory manager maps or copies the appropriate page to an area called a page frame. A program gets the information from the page frame.

Expanded-memory boards and managers conform to the Lotus/Intel/Microsoft Expanded Memory Specification (LIM EMS) version 3.2 or 4.0, which specifies how programs make use of expanded memory.

Some programs are unable to use expanded memory because they were not designed to interact with an expanded-memory manager. However, because expanded memory was introduced before extended memory, more programs are designed to use expanded memory rather than use extended memory.

Because an expanded-memory manager allows programs access to a limited amount of information at one time, expanded memory can be slower and more cumbersome for programs to use than extended memory.

5.2.4. Upper Memory Area

Most systems have 384K of free space called the upper memory area. This area is immediately adjacent to the 640K of conventional memory. The upper memory area is not considered part of the total memory of the computer because programs cannot store information in this area. This area is normally reserved for running the system's hardware, such as the monitor.

Information can be mapped from another type of memory to parts of the upper memory area left unused by the system. These unused parts are called upper memory blocks. (One use of this mapping is for running programs that use expanded memory.)

The RAM used in the Digitising Workstation is mapped at address D000h and falls into an unused slot in the upper memory area. A software function is used to fetch the data from specified addresses. The data is transferred to the conventional memory from where the software can present it on the screen.

5.3. Programmable Peripheral Interface (8255)

In order to control the operation of the Digitising Workstation an 8255 device is used. The 8255 eliminates the need for manual switches and enables the use of software for the selection of the various functions.

The Intel 8255 is a general purpose programmable I/O device designed for use with Intel microprocessors and micro controllers. It has 24 I/O pins which may be individually programmed in two groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bi-directional bus mode which uses 8 lines for bi-directional bus, and 5 lines, borrowing one from the other group, for handshaking.

The function of the 8255 is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255 is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

5.3.1. OPERATING MODES

MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.

- Outputs are latched.
- 16 different Input/Output configurations are possible in this Mode.

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In Mode 1, port A and port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port

MODE 2 (Strobed Bi-directional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bi-directional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

The 8255 is a very powerful tool for interfacing peripheral equipment to the microcomputer system. It represents the optimum use of available pins and is flexible enough to interface almost any I/O device without the need for additional external logic.

Each peripheral device in a microcomputer system usually has a "service routine" associated with it. The routine manages the software interface between the device and the CPU. The functional definition of the 8255 is programmed by the I/O service routine and becomes an extension of the system software. By examining the I/O devices interface characteristics for both data transfer and timing, a control word can easily be developed to initialise the 8255 to exactly fit the application.

It was decided to use the 8255 in MODE 0 as only basic latched outputs were required.

5.4. Circuit Description

Figure 5.3 shows the simplified block diagram of the Digitising Workstation. Only one channel is shown as the second channel is a duplication of the first.

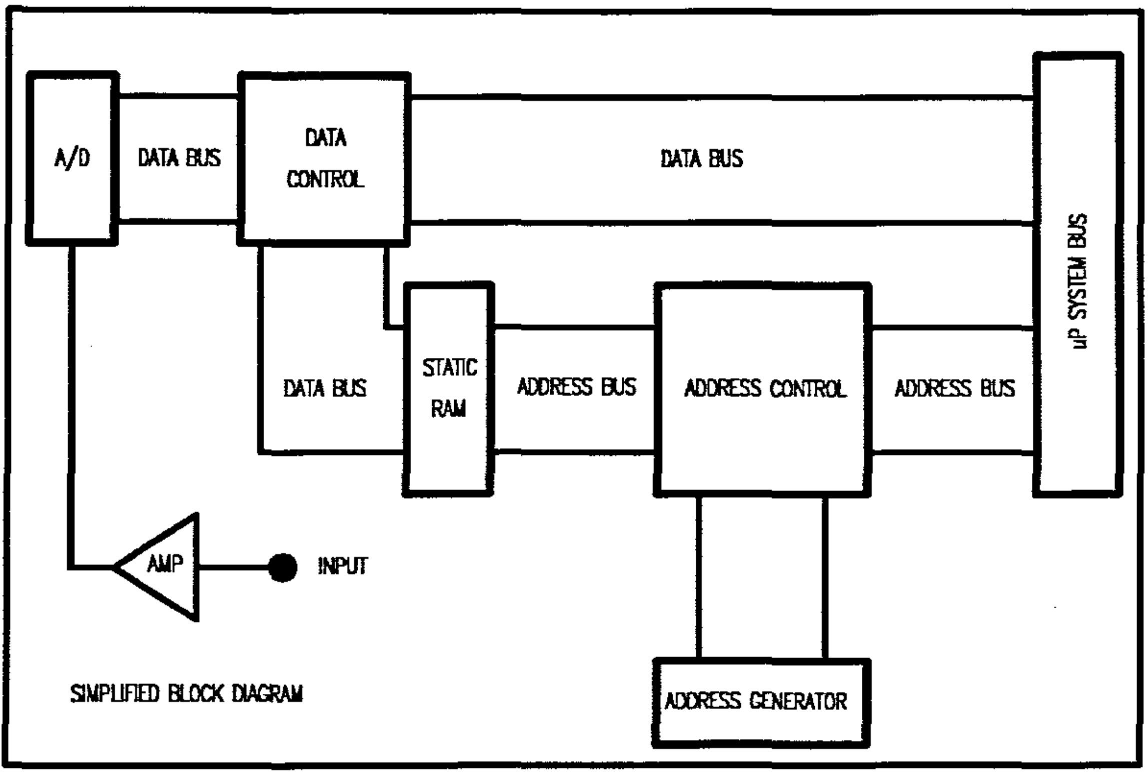


Figure 5.3

The circuit diagram can be divided into two sections:

- Analogue input stage
- Digital capturing stage

5.4.1. Analogue Input Stage

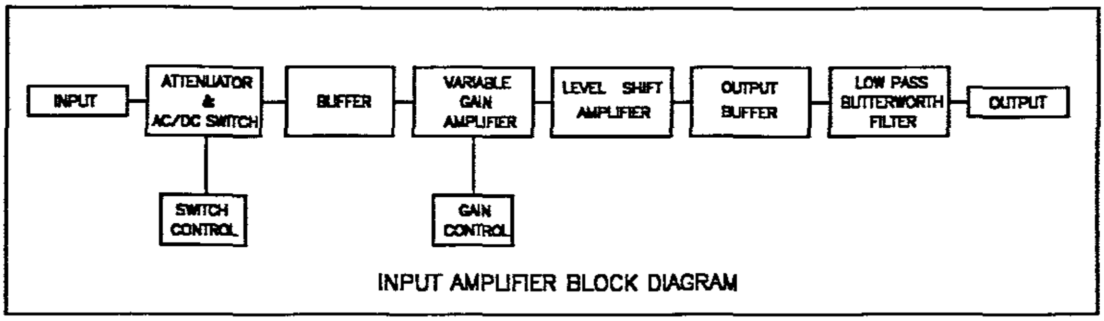


Figure 5.4

5.4.1.1. Description of the analogue circuit

In order to allow small signals to be sampled a programmable gain buffer amplifier is needed with a bandwidth of at least 10MHz. An AC/DC input is also required. Level shifting of the signal is also required to allow sampling of signals with negative polarity. Input impedance of at least 1MΩ is needed. Minimum input signal of 50 mV and maximum input signal of ±12V is necessary. The output needs to be offset by 2.5V and limited between +5V and 0V to prevent overload of the analogue to digital converter.

Figure 5.4 shows the simplified block diagram of the analogue circuit.

The input can be connected through a capacitor or directly via relay contacts. An input attenuator is used to raise the input impedance above 1MΩ. The signal is then fed via a wide-band buffer to the gain switching stage. Gain control is done by using a programmable gain amplifier. The output is then fed to an inverting amplifier with an output bias of +2.5V. The signal is then fed to a wide-band buffer which supplies the drive current for the switched capacitor input of the A-D converter. A fifth order passive Butterworth filter is used to limit the input bandwidth to the A-D converter to 10MHz.

5.4.1.2. Detailed Analogue Circuit Description

Refer to appendix B.

The input circuit is identical for both channels and only the operation of one channel will be discussed.

The input signal is applied to J5. K1 is a change-over relay and functions as the input AC/DC switch. The relay is operated by a transistor (Q1) under the control of the software. When the input setting is on AC, the DC component is blocked by C19. R27 and R30 forms the input attenuator with an impedance of more than 1MΩ. U8 is a wide-band unity-gain buffer (OPA633). The signal is then applied to the variable gain amplifier (U5). The AD600 (U5) has a bandwidth of 30 MHz at variable gains between 0 and 40 dB. The gain of the AD600 is set by applying a voltage of between 0 and 1 Volt on the C1HI pin. R9 to R12 are potentiometers used to preset the gain settings. The gain voltages are applied to an analogue multiplexer (U6) which selects the gain voltage under software control.

The output from the U5 is fed to an inverting amp (U2). U2 is a high frequency op-amp set at a fixed gain of two. R8 and R7 are used to level shift the output of the inverting amp to +2.5V. The output is applied to a buffer amplifier (U1) which supplies the drive current for the switched capacitor input of the A-D converter. L1, L2, C3, C4 and C5 forms a 5th order low-pass Butterworth filter which has the function of limiting the bandwidth to the A-D converter to 10MHz.

5.4.2. Digital Capturing Stage

The Digital stage can be subdivided into the following stages:

- Analogue to digital converter
- High speed static RAM

- Address counters
- Clock generator
- Address buffers
- Data buffers
- Address decoders
- Control circuits

5.4.2.1. Description of the Digital Circuit

The output from the input amplifier is connected to the input of the analogue to digital converter (A-D). The A-D is referenced at +5V and will therefore accept signals within the 0 to +5V *range. For the A-D to be able to digitise* AC signals the input amplifier is offset by 2.5V. The output code associated with +2.5V (64 Binary) will therefore represent a signal of 0V while an output code associated with +5V (128 Binary) will represent a signal of +2.5V and the code for 0V (0 Binary) will represent -2.5V. By scaling input signals accordingly it is possible to digitise positive and negative signals.

The digital signal is applied via a buffer to the static RAM. The static RAM is an 8K x 8 bit device and is rated at better then 20nS write time. The static ram is of a type commonly used as cache ram in computer systems. Four high-speed binary counters are used for addressing the static

RAM. The counters are configured in such a way that they form a 13 bit synchronys binary counter. Considering the operating speed of the project, National Semiconductor's "F" series of components proved to be vital and 74F161 devices were used.

The datum-data from the A-D are therefore written into memory under control of the address counter. When the counter overflows the pervious data will be overwritten. This enables the device to capture the most recent 8192 samples. When these samples are displayed on the computer screen, 8192 samples will represent almost 13 screens of information.

In order to allow the computer to read the data from the RAM, the address lines are connected to the address lines of the computer. This change-over is accomplished by using buffers as switches. The write operation is also stopped and the static RAM configured in the read mode using the 8255. The static RAM is address decoded at D000h and falls into an unused area in the upper memory area. Configuring the external memory as internal memory, the transfer of data is achieved at the same rate as that of the computer. The data lines are connected to the D0-D7 lines. This would mean that on an AT system only the even addresses will contain data from the static RAM. On odd addresses, data will be expected on the D8-D15 lines. To overcome this problem a circuit was designed that uses an inverted BHE signal connected to A0 of the static RAM. On even addresses the data byte will appear on the data bus and the computer will read it from the D0-D7 lines into the accumulator (AL). On odd addresses, data from the next position will also appear on the D0-D7 lines but the computer will now read this data into the High byte of the accumulator (AH) and the data on the D8-D15 lines (which contains no information) will be read into the AL. By swapping the alternative AL and AH bytes using software this problem is overcome and the need to have connections to D8-D15 eliminated. This also means a reduction of the component count.

The data is arranged as an array of data and displayed by the computer.

A trigger circuit is needed to facilitate an internal trigger function. This is accomplished by using two four-bit magnitude comparators that are configured to react to a digital code greater than a pre-set code selected by the 8255. The output signal is latched and then used to control a binary counter, which will allow a predetermined number of samples to be stored in RAM. When the counter reaches terminal count, it will stop the write operation of the RAM and control will be handed over to the PC to read out the stored information. By using the scroll function in the software, it is possible to view signals before and after the trigger point. The trigger point is user selectable and allows 128 trigger levels.

The external trigger interfaces to the internal trigger and shares the same circuitry. The external trigger is controlled by a +5V signal.

5.4.2.2. Detailed Digital Circuit Description

Refer to appendix B.

The circuit lay-out of the A-D converter is similar to the circuit *described in Figure* 5.2. *The recommended input* buffer (HA 5033) is replaced by the analogue input circuit described previously.

J1 is a crystal oscillator module operating at 20MHz. It is used to generate the master clock for the circuit. The clock signal is fed to the A-D converter via three inverters (U3-E, U3-F and U23-A). These inverters are used to time the data from the A-D converter according to the capturing circuit. The clock output also feeds the timebase generator formed by two binary counters (U16 and U20). The outputs provide the eight basic sampling rates used on the circuit. These eight frequencies are applied to a multiplexer (U25). The 8255 (U19) is used to control the multiplexer and only one frequency or sampling rate can be selected at a time. The selected sampling frequency is applied to the a 13 bit synchronous counter (U17, U21, U24 and U26). These counters are used as the address counter for the Static Ram devices (U8 and U13). The addressing is applied via buffers (U7 and U11). The logic circuit formed by U6-A, U4-A, U3-A and U3-B, controls the write and read operation of the Static Ram's. U9 and U12 are buffers used to control the addressing from the PC. The logic circuit will allow addressing either from the address counter or from the PC under control of software. The data lines (AD0 - AD6 and AC0 - AC7) are also controlled by means of buffers (U5 and U14). The data lines will either be connected to the A-D converters or to the PC bus, depending on the read or write cycle of the circuit.

The address decoding for the memory devices are done by U2, which is an eight bit magnitude comparator that is pre-set to decode addresses above D000h. The decoding signal is used to control the buffers used during the transfer of data to the PC bus.

U10 and U15 forms the circuit that determines the number of bytes stored under control of the trigger signals.

The data lines of the 8255 (U19) are buffered by U18 in order to prevent loading of the PC bus. The 8255 is memory mapped at address 300h and the decoding is done by U22. The outputs of the 8255 are used to set up the internal trigger level, control the read and write operation of the Static RAM devices, control the sampling rate and control the gain settings and input switches of the buffer amplifiers.

5.5. Power Supply

The Digital circuit draws its power from the PC and requires only +5V.

The analogue circuit requires ±12V and ±5V. These voltages are supplied by a separate power supply external to the PC. It was decided to use an external power supply for the analogue circuit, because the switch mode power supply of the PC induced noise into the circuit.

6. THE MSCOPE.EXE PROGRAM

6.1. Main Function

The main function calls the initialize_graphics function to auto detect the graphics driver used and will abort if an error is encountered.

The outport function is used to initialise the 8255 with value 80h. This places the 8255 in the latched output mode (mode 0).

The menubar function is used to construct the menu bar at the top as well as the bottom of the screen.

The mouse driver is then initialised for graphics mode. If the mouse is not detected the program will abort and a message displayed on the screen. The mouse is initialised to show an arrow cursor on the screen. The Mouse.Event function is used to return the screen co-ordinates where the event occurred.

A switch statement is used to determine whether a key was depressed. If a Q was pressed the program will abort. The user will be prompted to enter a filter value when the F key is pressed. The filter function will be explained at a later stage. The user will also be prompted for a clock frequency value when the C key is pressed. This allows the use of different clock frequencies.

The Mouse.Event is the main event in the program. A switch statement is used to determine whether the left mouse button was pressed. The screen co-ordinates are returned to the function and multiple if statements are used to determine whether the button was depressed in a defined area called a hot spot. Hot spots are defined for all the menu functions. The user calls a menu item by moving the mouse cursor over the menu item and then pressing the left mouse button. The program will then determine which part of the program to execute. The individual items of the mouse event will be described in detail.

6.1.1. Menu

The menu option is defined by hot spot 1. The menu1 function is used to draw a sub menu on the screen when the mouse button is pressed over the menu option. The menu items used are the FILE SAVE, FILE IMPORT and scroll value (<< >> VALUE) options.

6.1.1.1. File Save

The File_store function prompts the user to specify a data drive as well as a file name. A binary file is then opened with the specified filename. If an error condition exists, a message will be displayed on the screen indicating that the file cannot be opened. Fwrite commands are then used to write the sampled data for both channels as well as the sampling parameters to the file.

6.1.1.2. File Import

The file_dir function creates a list of all the files, containing sampled data, on the specified drive. The file is then opened and tested for an opening error. The fread function is used to read the file and to place the sampled data as well as the sampling parameters in the arrays and variables specified.

6.1.1.3. Scroll Value (<< >> value)

The user is prompted for a shift factor. This factor determines the number of addresses to move at a time when a forward or backward scroll of the signal is performed.

6.1.2. Quit

The QUIT option is defined by hot spot 2. This function is used to terminate the operation of the program and return to DOS.

6.1.3. Zoom X value (<<X>>)

This function, defined by hot spot 3, is used to expand or reduce the X value when the signal is displayed on the screen. The user is prompted for an X value. The X value is then enlarged or reduced accordingly. The factor is written to the EXP variable. The atoi function is used to convert the ASCII value entered on the keyboard to an integer value. The X co-ordinate is then multiplied with the factor when the signal is displayed.

6.1.4. Zoom Y value (<<Y>>)

This function, defined by hot spot 4, is used to expand or reduce the Y value when the signal is displayed on the screen. The user is prompted for an Y value. The Y value is then enlarged or reduced accordingly. The factor is written to the VERT variable. The atoi function is used to convert the ASCII value entered on the keyboard to an integer value.

The Y co-ordinate is then multiplied with the factor when the signal is displayed.

6.1.5. Shift Backward

The variable SHIFT is used as an index into the arrays VAL and VAL1. By decrementing the value of the index, it is possible to move backwards in the array. The variable SHIFTF defines the number of addresses to shift at a time. The shift backward option is controlled by hot spot 5.

6.1.6. Shift Forward

The variable SHIFT is used as an index into the arrays VAL and VAL1. By incrementing the value of the index, it is possible to move forward in the array. The variable SHIFTF defines the number of addresses to shift at a time. The shift forward option is defined by hot spot 6.

6.1.7. Dot Display

When the mouse button is depressed over hot spot 7, the signal will be displayed as individual pixels using the putpixel command. The variable DISP, having a zero value, is used by the display function to place it in the dot mode.

6.1.8. Line Display

Hot spot 8 is used to control the line display function. In this mode the signal is presented as a solid line connecting the sampling points using the lineto command. The variable DISP, having a value of 1, is used by the display function to place it in the line display mode.

6.1.9. Manual

Hot spot 9 controls the manual sampling operation of the digitising workstation. The manual function uses outport commands to write to the 8255 controlling the write operation of the RAM. After a delay of 500 ms, capturing is stopped and the chan1 function used to enable the read operation for channel 1. The read1 function is used to read the data from the RAM, which is memory mapped, and place it in an array called VAL. If the second channel was enabled, the chan2 and read2 functions are used to read from the second channel and place it in an array called VAL1.

The display function is then used to display the signals on the screen. The workstation is then prepared for the next capture.

6.1.10. Internal Trigger

Hot spot 10 controls the internal trigger operation of the digitising workstation. The inttrig function does the set-up for the internal trigger operation. When a signal greater than the trigger level is experienced, the hardware will react to the trigger pulse and stop the capturing of samples after a fixed count. The data from both channels are transferred to the PC in the same way as described under section 6.1.9.

The trigtest function is then used to determine the validity of the trigger pulse and the operation repeated until a valid trigger pulse is received. The operation will abort when a key is depressed.

6.1.11. External trigger

Hot spot 11 controls the external trigger operation of the digitising workstation. The external trigger operation is similar to the internal trigger operation described in section 6.1.10. The trigger pulse in this case is controlled from an outside source. Again the validity of the trigger pulse needs to be verified before the signal is displayed.

6.1.12. Reference

When the mouse is pressed over hot spot 12, the program is prepared for calculations. A reference position is created at the position of the next mouse event which is used for calculating voltage, frequency, period and phase angle.

6.1.13. Shift Up

The OFFSET and OFFSET2 variables define the offset from the top of the screen. The movement of the display is fixed at increments of 10 pixels. The offset decreases by 10 when the signal is shifted upwards.

6.1.14. Shift Down

The OFFSET and OFFSET2 variables define the offset from the top of the screen. The movement of the display is fixed at increments of 10 pixels. The offset increases by 10 when the signal is shifted downwards.

6.1.15. Filter

The digitising workstation uses two filtering techniques. The first one is a display filter that will ignore any stray samples that deviate from the previous sample with the parameter set up using the F key. This technique was described earlier.

The second filtering technique replaces samples that deviate from the previous sample, with more than a pre-set value, with an averaged value using interpolation. The pre-set value is stored in the variable filter and is entered by the user as the start value. The filter value is decremented and the operation repeated until the stop value is reached (also specified by the user). This technique removes all unwanted samples without effecting the signal negatively.

It must be stressed that the filters are only used to rid the signal from unwanted noise and spikes and not to introduce specific cut-off frequencies associated with analogue filters.

6.1.16. AC Ch1

The value 40h is written to port B of the 8255 to control the AC input of channel 1.

6.1.17. DC Ch1

The value of port B is ANDED with BFh to control the DC input of channel 1.

6.1.18. Voltage Scale CH1

The vscale function is used to select the input attenuator for channel 1 and to scale the signal accordingly. Three scales are available: 10V, 5V and 1V.

6.1.19. Trigger Level

The function trigger_level is used to set up the trigger level needed. Values between 0 and 128 will be accepted. When the internal trigger option is used, a signal exceeding the trigger level will start the counter which determines the number of samples to captured. Capturing will stop when the counter reaches terminal count and the signal will be displayed.

6.1.20. Sampling Rate

The sampling rate is set up using the sscale function. Eight sampling rates are available: 20 MHz, 10 MHz, 5 MHz, 2.5 MHz, 1.25 MHz, 625 KHz, 312.5 KHz and 156.2 KHz.

6.1.21. CH1 Up/Down Control

A value of 0 in the variable ch_1 enables channel 1 to be shifted vertically. When the mouse key is pressed over hot spot 21, ch_1 is reset to 0.

6.1.22. AC Ch2

The value 80h is written to port B of the 8255 to control the AC input of channel 2.

6.1.23. DC Ch1

The value of port B is anded with 7Fh to control the DC input of channel 2.

6.1.24. Ch2 Up/Down Control

A value of 1 in the variable ch_1 enables channel 2 to shifted vertically. When the mouse key is pressed over hot spot 24, ch_1 is set to 1.

6.1.25. Voltage Scale Ch2

The vscale function is used to select the input attenuator for channel 2 and to scale the signal accordingly. Three scales are available: 10V, 5V and 1V.

6.1.26. Ch2 On

The variable chan2on is set to 1, which is used to by the display function to enable channel 2.

6.1.27. Ch2 Off

The variable chan2on is reset to 0, which is used to by the display function to disable channel 2.

6.2. Functions

In this section the most important functions will be explained.

6.2.1. The draw function

The draw function is used to display the signal on the screen.

The draw function has a number of options:

- Line or dot display
- X scale factor
- Y scale factor
- Offset on screen
- Position in the data array
- Colour
- Channel 2 on/off
- 0V reference line
- Display filter

The background colour is set by the function setbkcolor to black. The variable PIXEL_COUNT is set to 640 which is equal to the number of pixels on the screen in the X direction. A for loop is used to control the position where the value should be displayed. If the LINE variable is reset, the signal will be displayed as dots. If the LINE variable is set, the signal will be displayed as a solid line. The putpixel command is used to display the signal as dots, while the lineto command is used for the line display.

The i variable defines the horizontal position on the screen. The EXP variable defines the X scale factor. By multiplying the i variable with the EXP variable, the signal can be horizontally expanded. The user is prompted for an X value when the mouse button is depressed over hot spot 3.

The OFFSET variable is used to determine the vertical position on the screen. The value of the OFFSET variable is incremented or decremented when the mouse button is depressed over hot spot 13 or hot spot 14.

The VAL variable defines an array of chars. This array contains the data to be displayed. The SHIFT variable together with the i variable is used as an index into the array. The signal exists only as seven bit data, so the value is anded with 7Fh to mask bit number eight. The signal value is also multiplied with the VERT variable, which determines the vertical scale. The user is prompted for a Y value when the mouse is depressed over hot spot 4.

The display filter mentioned earlier is used to ignore stray samples not associated with the signal. The absolute value of the first sample is compared with the absolute value of the second one. When the difference is greater than the value specified in the fil variable, the second sample will be ignored and the next sample considered. In this way samples not forming part of the signal can be ignored. The fil variable can be changed by pressing the F key and entering a new value.

The operation of the second channel is similar to the first channel. The only differences are the array called VAL1, containing the data for the second channel as well as the variable OFFSET2, which defines the vertical position.

A reference line is drawn for both channels showing the relationship of the signal to 0V. The reference line will follow the signal when it is shifted vertically. In this way, the signals for both channels could be displayed separately without overlapping each other and still maintaining a reference to 0V. The signals could also be overlapped by shifting them vertically.

6.2.2. Calculations

Calculations are done by using the mouse. After the signal is displayed, the left mouse button and the right mouse button are used to mark references on the wave from which calculations for voltage, frequency, period and phase angle are done. The pixels on the screen and the sampled values have a linear relationship. It is thus possible to use the mouse co-ordinates to calculate the required signal parameters.

When the left mouse button is pressed for the first time a reference point is created from which all measurements are calculated. The first reference point is stored in the variables xval1 and yval1. When the left mouse button is pressed again, the co-ordinates are stored in variables xval2 and yval2. When the right mouse button is pressed the co-ordinates are stored in the variables xval3 and yval3.

When the difference between yval1 and yval2 is taken, the potential difference between the two points can be calculated. The following statement is used for the calculation of the potential difference:

```
volt = ((yval1 - yval2)*0.039370078)/VERT;
```

By deviding with the vertical scale factor, VERT, voltage calculations stays true for all vertical scales.

Frequency and period between the reference and the second marker can be calculated by taking the difference between xval1 and xval2. The following statements are used to calculate the frequency and period:

```
frek = (1/(xval2 - xval1)*SAMPLE*1000)*EXP;
```

```
time = ((xval2 - xval1)*1/SAMPLE)/EXP;
```

The phase angle and period are calculated using the difference between xval1 and xval3 as well as the first period calculation. The following statements are used:

```
phase = (((xval3 - xval1)*1/SAMPLE)/EXP)/time*360;
```

```
time1 = ((xval3 - xval1)*1/SAMPLE)/EXP;
```

The SAMPLE variable represents the sampling frequency and the EXP variable, the horizontal scale.
The sampling parameters are also shown on the screen together with the calculations. The function gprint is used to display the results in graphics mode.

6.2.3. The read1 and read2 Functions

The read1 and read2 functions are written in assembly language and included as part of the main program.

These functions read the sampled data from the digitising workstation to the arrays VAL and VAL1.

7. PROBLEMS ENCOUNTERED

During the development of the digitising workstation several problems were encountered.

7.1. Bandwidth of Op-Amps

The requirement to be able to sample signals at 1MHz eliminated the use of normal operational amplifiers as their gain bandwidth products are limited to values less than 2MHz. If gain were to be introduced into the circuit, insufficient bandwidth would result. Several discrete designs were considered but failed because of physical size, noise and distortion. The best option proved to be video amplifiers. The AD829 (Analog Devices) video op-amp was used having a bandwidth greater than 50MHz at gains of up to 20 and slew rate of 230V/μs. The use of these amplifiers solved the bandwidth problem but introduced some other problems.

Oscillations in the region of 40MHz occurred when these op-amps were used. This was rectified by using decoupling capacitors on the power leads as well as using a capacitor towards earth on the compensation pin.

Another problem encountered with the op-amps was output drift when the input was high impedance. The output would not settle around 0V when the input was disconnected, but drifted to +VCC. The only way to overcome this was to terminate the input of the op-amp reducing the input impedance. To increase the input impedance high speed buffers (OPA633) were used having a slew rate of 2500V/μs.

7.2. Finding A Suitable A-D converter

When the project was commissioned, suitable high speed A-D converters were not available or were too costly. ADC207 (DATEL) devices were imported from the United States of America.

7.3. Operating Speed

The operating speed of the project caused normal logic components to fail or to cause excessive delays. The F series from National Semiconductor Corporation were the only type of component that would meet the speed requirements. The F series are not fully supported in South Africa and the design had to be changed several times because of component shortages.

The speed of the project caused breadboarding to fail in most cases. PC boards had to be designed each time the circuit was modified.

7.4. Component Sourcing

Component sourcing proved to be a difficult task. When stocks were not available, suppliers were not prepared to order small quantities. Frequently components specified in data books were not supported by the distributors. Sourcing was done throughout the country as well as abroad.

7.6. Working on the PC Bus

Interfacing the project with the PC bus also created problems. Any loading of the bus caused the operation of the PC to cease. Buffers were used throughout to rectify this.

7.7. Design of the PC Boards

The design of the pc board layout was also a difficult task as well as time consuming because of a complex circuit. Specialised Computer Aided Design (CAD) tools only became available at a mature stage of the project. Orcad was used for the initial schematic capture while the routing was done using Tango.

7.8. Direct Addressing of the RAM

When a Micro Processor is used in a design, the addressing of the RAM is handled by the processor. In the case of the digitising workstation, the rate of addressing is too high. The addressing had to be done manually using counters. The counters used needed to be of the F series (National Semiconductor), but were not readily available.

7.9. Software

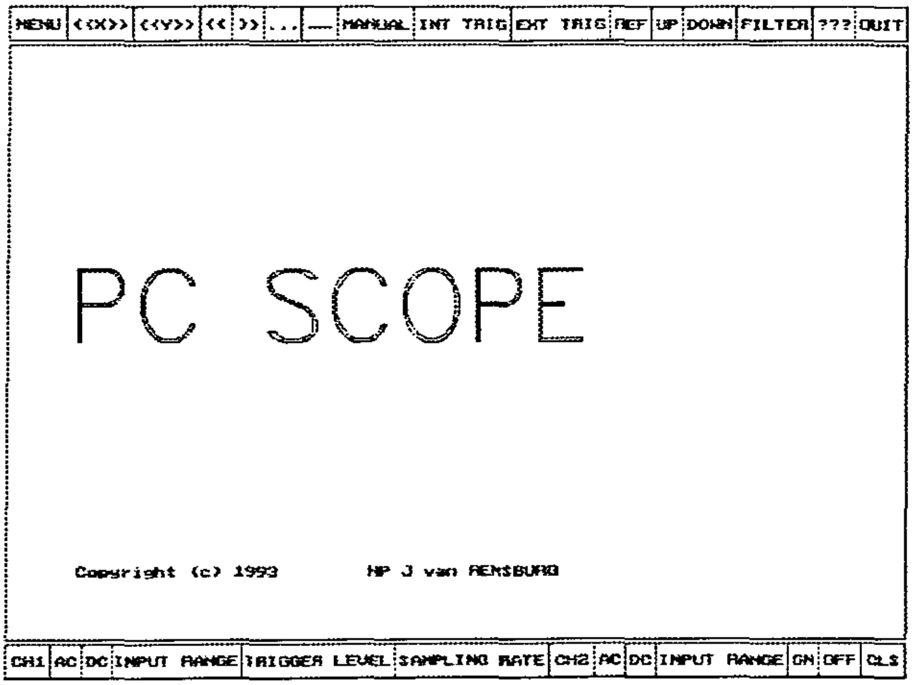
The complexity of the software required the use of a high level language.

8. TEST RESULTS

The digitising workstation is fully functional, operating at a clock frequency of 20MHz and captures samples of two channels simultaneously. Captured signals can be written to file. Calculations like voltage, frequency and phase angle can be done.

Extra features included are:

- mouse driver
- scroll functions
- zoom functions
- filtering of the signal
- trigger options



PC SCOPE

Copyright (c) 1993 HP J van RENSBURG

Figure 8.1

Figure 8.1 shows the main screen with the logo that will be displayed for two seconds. Different colours are used to differentiate between the menus.

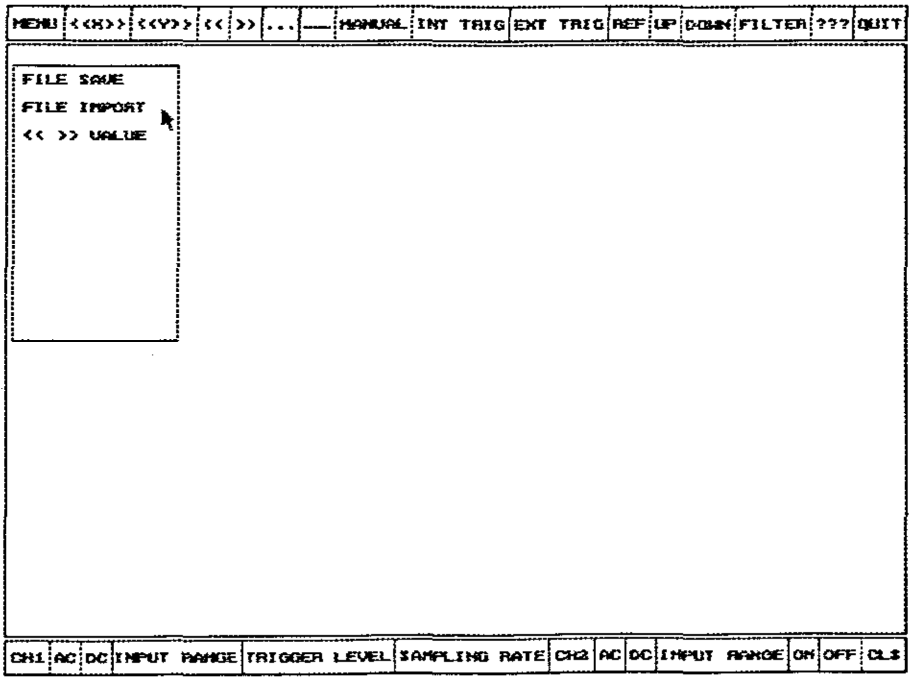
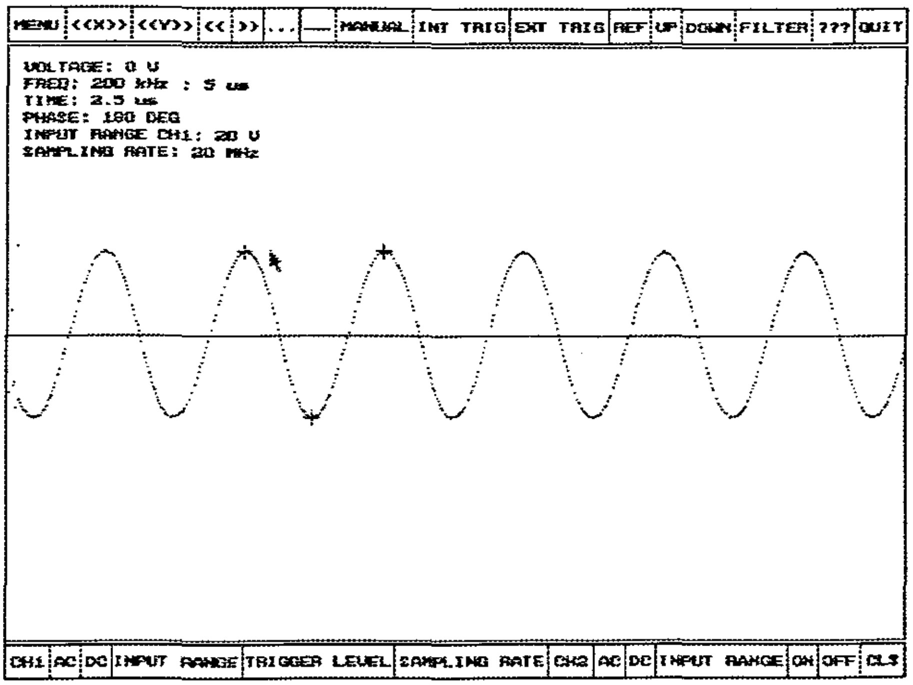


Figure 8.2

Figure 8.2 shows the main menu with the file import, file save and shift factor options.



```
VOLTAGE: 0 V
FREQ: 200 kHz : 5 us
TIME: 2.5 us
PHASE: 180 DEG
INPUT RANGE CH1: 20 V
SAMPLING RATE: 20 MHz
```

Figure 8.3

Figure 8.3 shows a sine wave displayed in the dot mode. The reference cursors are also visible. The calculations field shows the parameters of the signal.

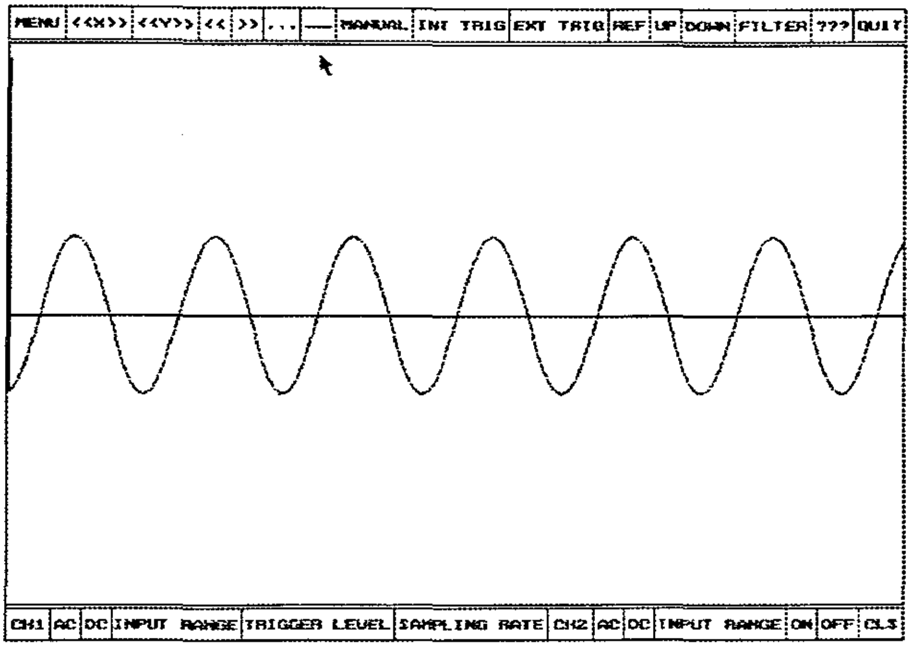
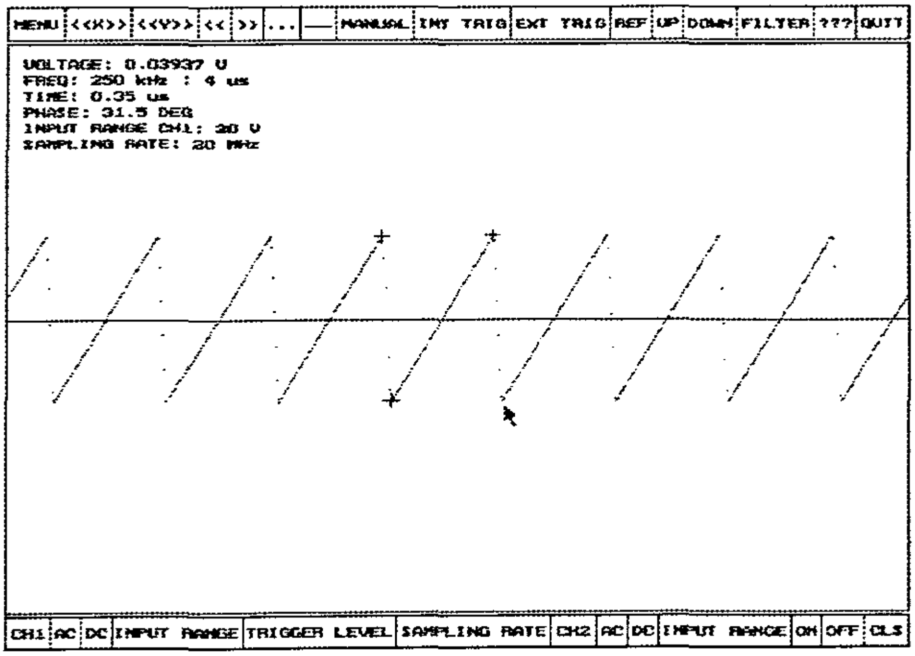


Figure 8.4

Figure 8.4 shows the same signal in the line display mode.



```
VOLTAGE: 0.03937 V
FREQ: 250 kHz : 4 us
TIME: 0.35 us
PHASE: 31.5 DEG
INPUT RANGE CH1: 20 V
SAMPLING RATE: 20 MHz
```

Figure 8.5

Figure 8.5 shows a more complex waveform in the dot mode. The individual samples can clearly be seen.

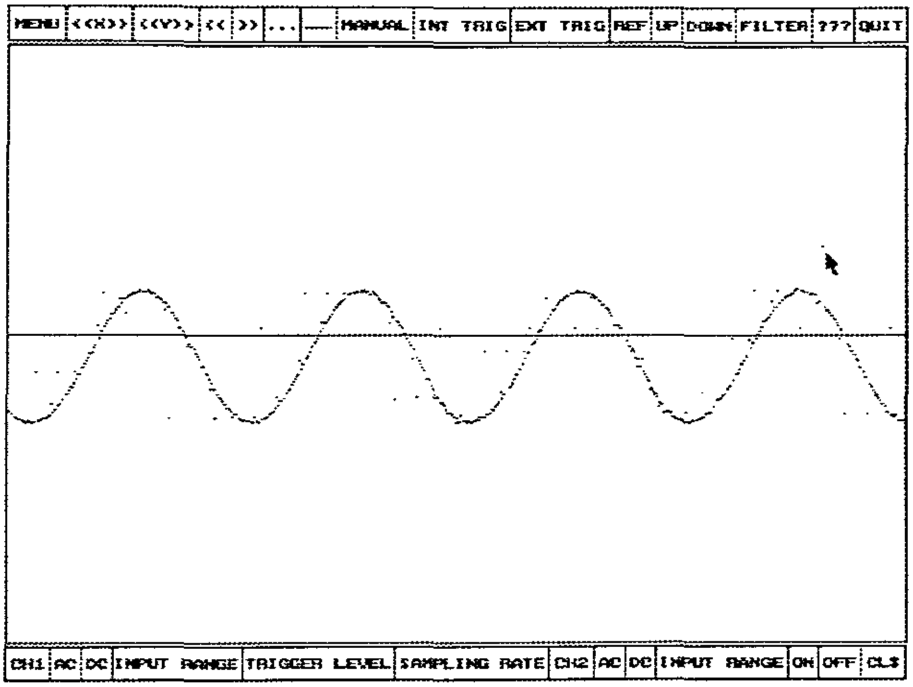


Figure 8.6

Figure 8.6 shows a noisy sine wave.

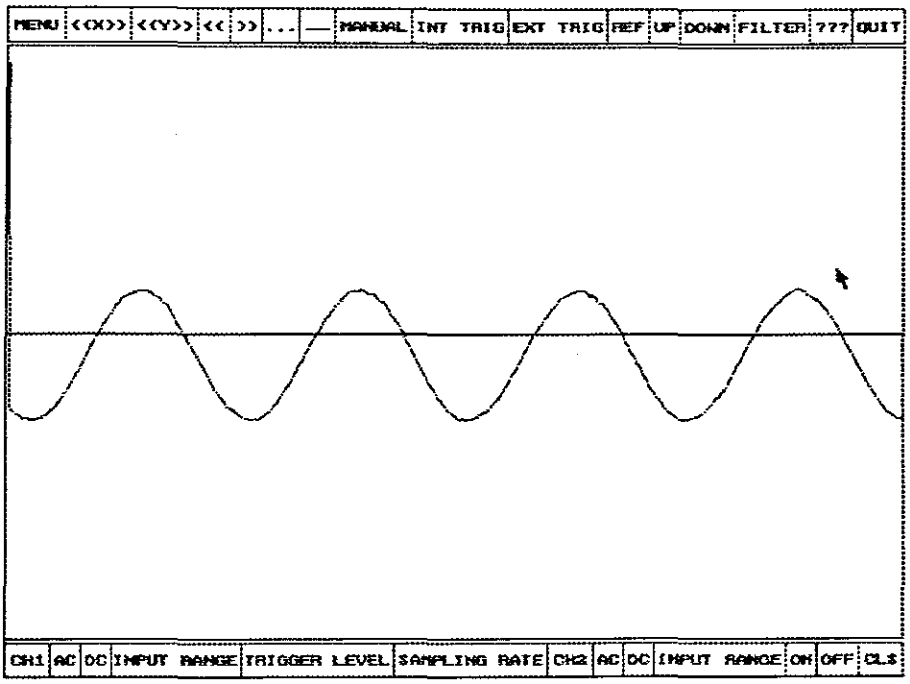


Figure 8.7

Figure 8.7 shows the same signal as in Figure 9.6 with the display filter activated. The line mode is used to connect the samples.

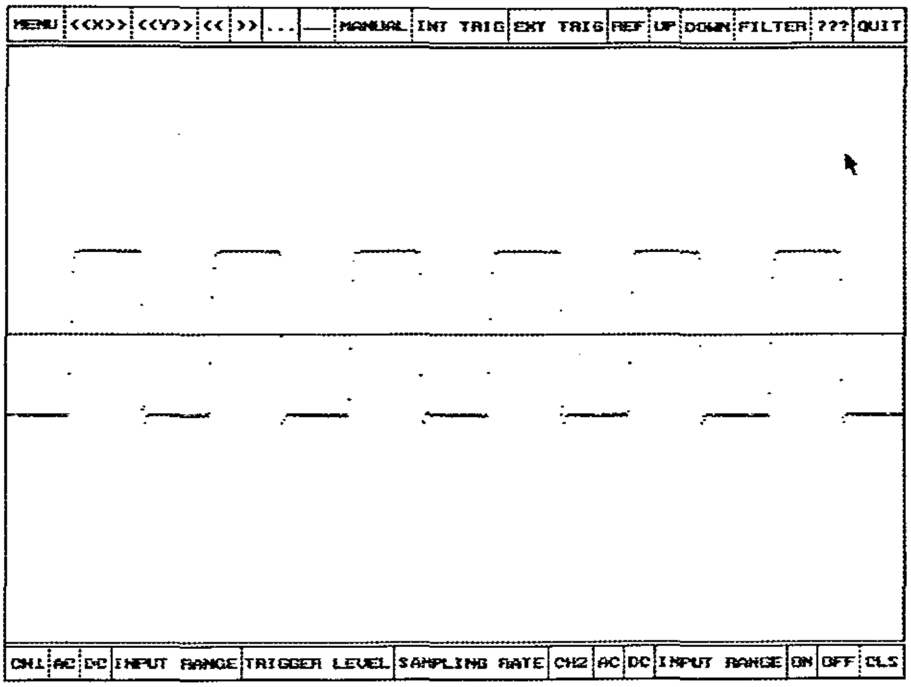


Figure 8.8

Figure 8.8 shows a square wave.

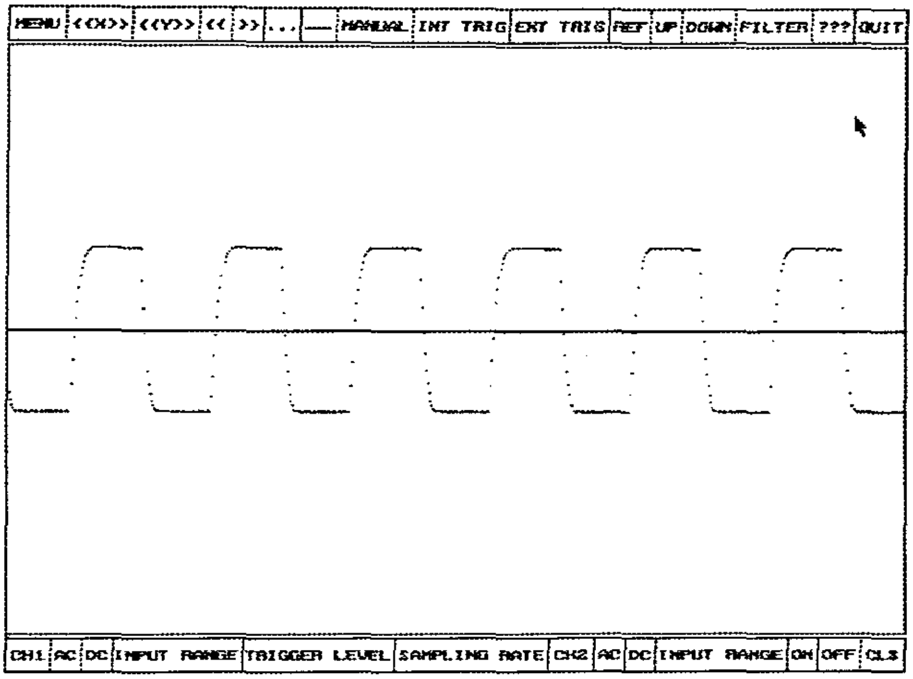
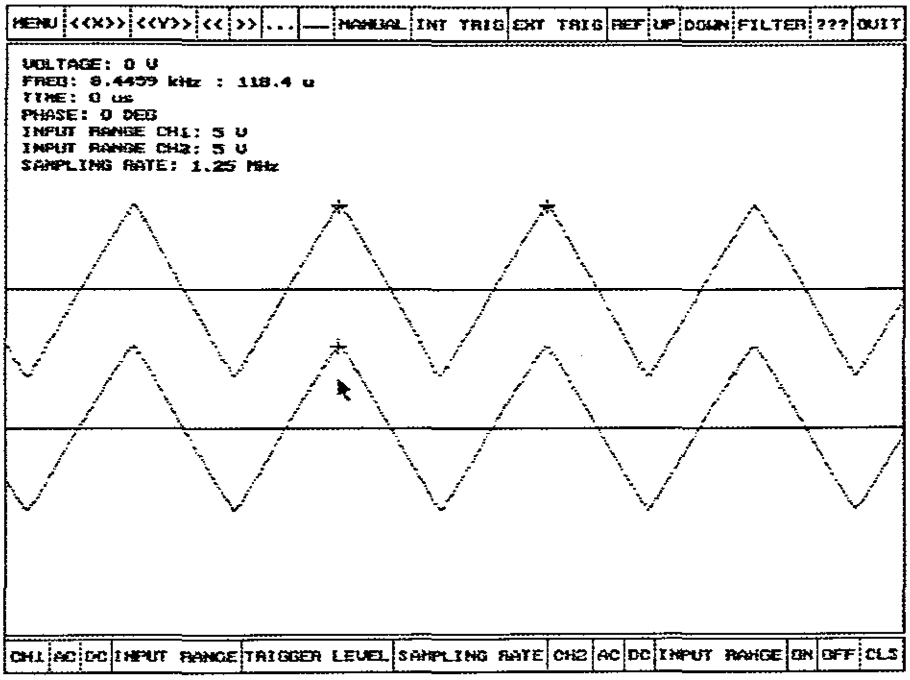


Figure 8.9

Figure 8.9 shows the same square wave as shown in figure 8.8, but after the filter was used. The interpolated samples are clearly visible.



```
VOLTAGE: 0 V
FREQ: 8.4459 kHz : 118.4 u
TIME: 0 us
PHASE: 0 DEG
INPUT RANGE CH1: 5 V
INPUT RANGE CH2: 5 V
SAMPLING RATE: 1.25 MHz
```

Figure 8.10

Figure 8.10 shows signals on both channels.

9. FUTURE ENHANCEMENTS

Software development is an on-going process that is susceptible to future enhancements. The following possible enhancements could be included in the next version:

- Writing a Windows based program
- Adding export facilities to other file formats
- Adding import facilities from other file formats
- Automating the calculation of the signal parameters
- Converting the unit to a Logic Analyser

9.1. Writing a Windows based program

The mscope.exe program is currently a DOS application. The interaction between various Windows packages which allows files to be transferred between them should appeal more to users preferring a multitasking environment.

9.2. Adding Export Facilities to other File Formats

The file format used for saving signal data does not correspond to other designs. It could be useful to be able to convert the captured images to other file formats.

9.3. Adding Import Facilities from other File Formats

Importing captured images from other packages could be useful when comparisons with signal, captured with other packages, needs to be done.

9.4. Automating the Calculation of the Signal Parameters

Currently a cursor scheme is implemented to calculate the signal parameters. Automating the calculations could improve the overall performance of the unit.

9.5. Converting the unit to a Logic Analyser

With the analogue buffer removed, the unit could be used as a Logic Analyser with 8 inputs. This would entail a change in the software.

10. CONCLUSION

The Digitising Workstation has wide ranging applications in the educational and general electronics market, not only as a laboratory instrument, but as a general-purpose digitising unit.

In the electronics maintenance sphere, the unit could be very useful. Standard waveforms at various test points could be captured and archived and then compared with test points on the circuit under test. In this way a library of standard waveforms can be created. The Digitising Workstation could be installed in a Notebook computer resulting in a fully portable test and measuring unit. A hardcopy output of the unit could be included in reports giving visual proof that a fault has been resolved.

In the education sphere the workstation could also be very useful. A library of captured signals could be used in class demonstrations by the lecturer. Students could also use hardcopy results in documentation of research projects.

To conclude, the final product surpasses the requirements initially requested and would prove to be advantageous in any electronics environment.

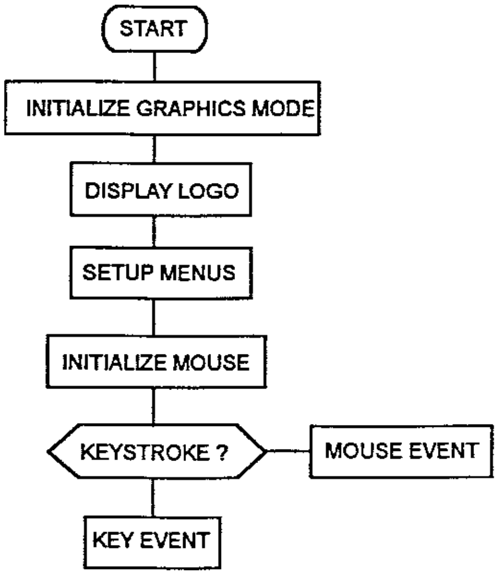
11. BIBLIOGRAPHY

1.	HOROWITTZ,P. HILL, W.	1987	The Art of Electronics
2.	INTEL CORPORATION	1988	Memory Components Handbook
3.	NATIONAL SEMICONDUCTOR COMPANY	1988	Linear Databook 1,2,3
4.	NATIONAL SEMICONDUCTOR COMPANY	1988	Cmos Logic Databook
5.	TEXAS INSTRUMENTS	1973	The Linear and Circuits Data Book
6.	BURR-BROWN	1986	Integrated Circuits Data Book
7.	McGRAW-HILL	1990	Teach Yourself C

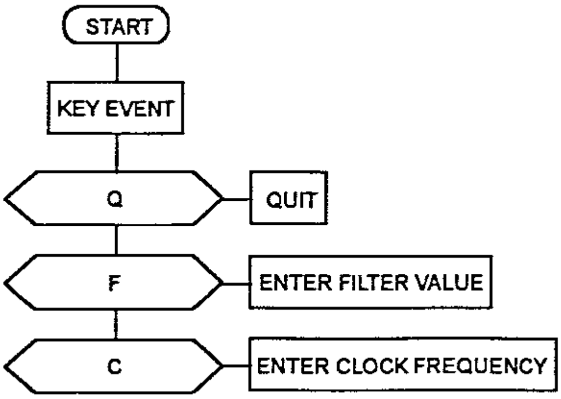
8.	BORLAND	1992	Turbo C++, Users Guide
9.	BORLAND	1991	Borland C, Library Reference
10.	ABEL, P.	1987	I.B.M. P.C. Assembler Language and Programming
11.	ANALOG DEVICES	1992	Data Converter Reference Manual Volume 2
12.	NATIONAL SEMICONDUCTOR COMPANY	1988	Application note 236. An Introduction to the Sampling Theorem
13.	PHILIPS		Basic Principles of DSOs using the Philips PM3350

12. APPENDIX A

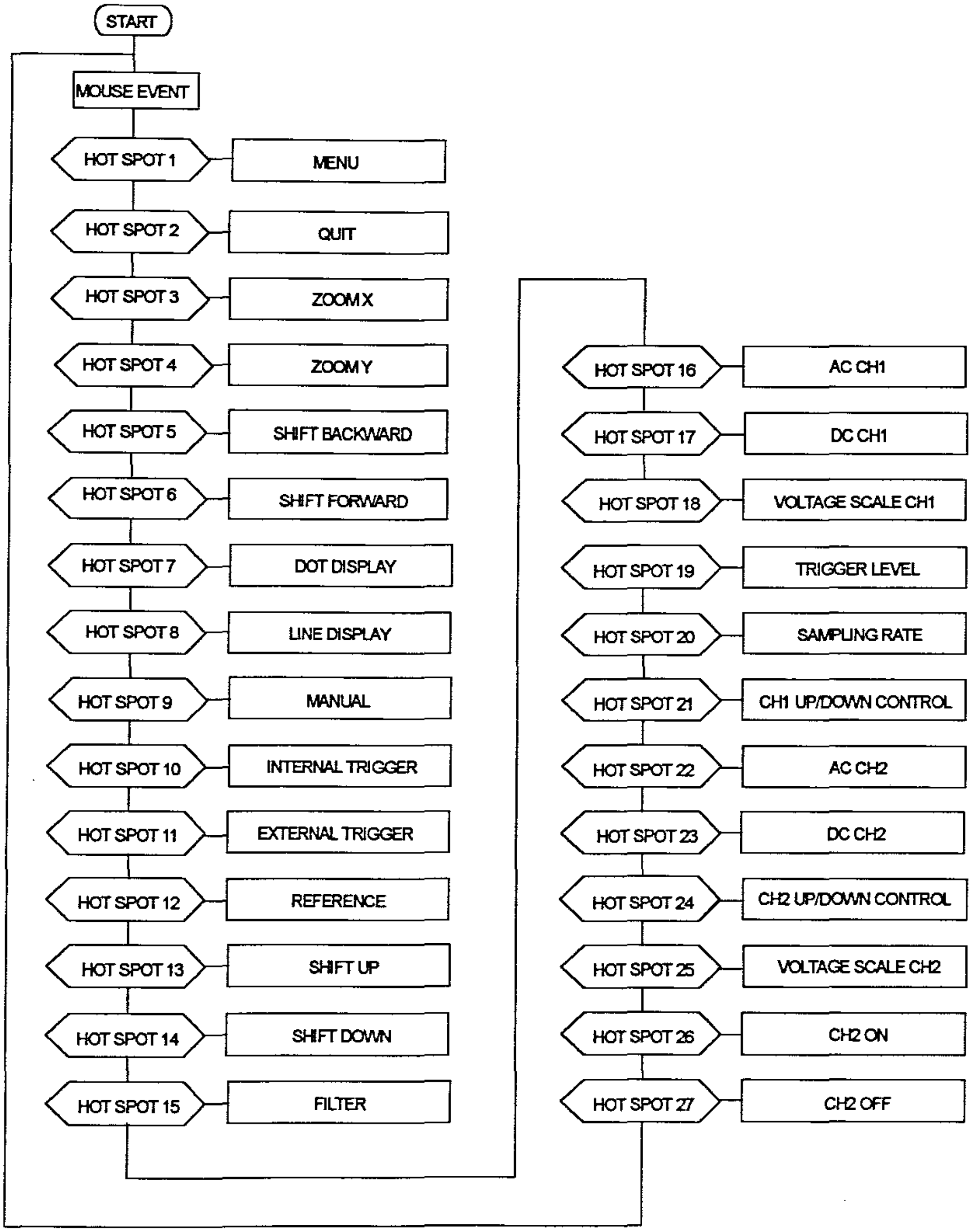
12.1. Flow Diagrams



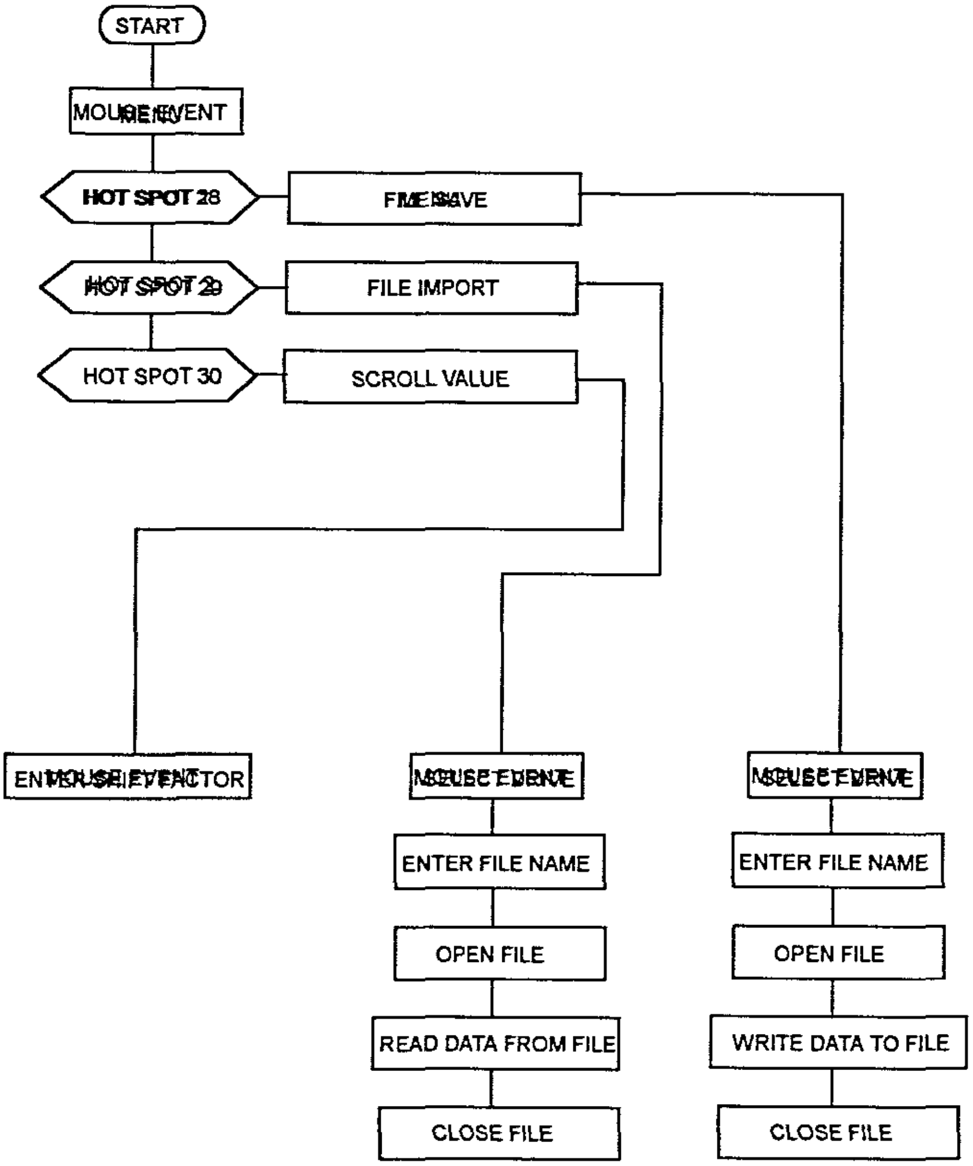
Initialisation



Key Event


START
MOUSE EVENT
HOT SPOT 1
MENU
HOT SPOT 2
QUIT
HOT SPOT 3
ZOOM X
HOT SPOT 4
ZOOM Y
HOT SPOT 5
SHIFT BACKWARD
HOT SPOT 6
SHIFT FORWARD
HOT SPOT 7
DOT DISPLAY
HOT SPOT 8
LINE DISPLAY
HOT SPOT 9
MANUAL
HOT SPOT 10
INTERNAL TRIGGER
HOT SPOT 11
EXTERNAL TRIGGER
HOT SPOT 12
REFERENCE
HOT SPOT 13
SHIFT UP
HOT SPOT 14
SHIFT DOWN
HOT SPOT 15
FILTER
HOT SPOT 16
AC CH1
HOT SPOT 17
DC CH1
HOT SPOT 18
VOLTAGE SCALE CH1
HOT SPOT 19
TRIGGER LEVEL
HOT SPOT 20
SAMPLING RATE
HOT SPOT 21
CH1 UP/DOWN CONTROL
HOT SPOT 22
AC CH2
HOT SPOT 23
DC CH2
HOT SPOT 24
CH2 UP/DOWN CONTROL
HOT SPOT 25
VOLTAGE SCALE CH2
HOT SPOT 26
CH2 ON
HOT SPOT 27
CH2 OFF

Mouse Event


START
MOUSE EVENT
MENU
HOT SPOT 28
FILE SAVE
HOT SPOT 29
FILE IMPORT
HOT SPOT 30
SCROLL VALUE
ENTER SCALE FACTOR
MOUSE EVENT
MOUSE EVENT
SELECT DRIVE
ENTER FILE NAME
OPEN FILE
READ DATA FROM FILE
CLOSE FILE
MOUSE EVENT
SELECT DRIVE
ENTER FILE NAME
OPEN FILE
WRITE DATA TO FILE
CLOSE FILE

Menu

12.2. SOFTWARE

12.2.1. Mscope.cpp

```
#include <graphics.h>
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <bios.h>
#include <dos.h>
#include <sys/stat.h>
#include <string.h>
#include <fcntl.h>
#include <io.h>
#include <dir.h>
#include "msmouse.h"
#include <math.h>
#include <ctype.h>


void clearviewport_ ( void )
{
     Mouse.Hide ( );
     clearviewport ( );

}

void gdisplay ( char gch)
{
switch (gch)
  {
  case 'a': outtext("a"); break;
  case 'b': outtext("b"); break;
  case 'c': outtext("c"); break;
  case 'd': outtext("d"); break;
  case 'e': outtext("e"); break;
  case 'f': outtext("f"); break;
  case 'g': outtext("g"); break;
  case 'h': outtext("h"); break;
  case 'i': outtext("i"); break;
  case 'j': outtext("j"); break;
  case 'k': outtext("k"); break;
  case 'l': outtext("l"); break;
  case 'm': outtext("m"); break;
  case 'n': outtext("n"); break;
  case 'o': outtext("o"); break;
  case 'p': outtext("p"); break;
  case 'q': outtext("q"); break;
case 'r': outtext("r"); break;
case 's': outtext("s"); break;
case 't': outtext("t"); break;
case 'u': outtext("u"); break;
case 'v': outtext("v"); break;
case 'w': outtext("w"); break;
case 'x': outtext("x"); break;
case 'y': outtext("y"); break;
case 'z': outtext("z"); break;
case '0': outtext("0"); break;
case '1': outtext("1"); break;
case '2': outtext("2"); break;
case '3': outtext("3"); break;
case '4': outtext("4"); break;
case '5': outtext("5"); break;
case '6': outtext("6"); break;
case '7': outtext("7"); break;
case '8': outtext("8"); break;
case '9': outtext("9"); break;
case 'A': outtext("A"); break;
case 'B': outtext("B"); break;
case 'C': outtext("C"); break;
case 'D': outtext("D"); break;
case 'E': outtext("E"); break;
case 'F': outtext("F"); break;
case 'G': outtext("G"); break;
case 'H': outtext("H"); break;
case 'I': outtext("I"); break;
case 'J': outtext("J"); break;
case 'K': outtext("K"); break;
case 'L': outtext("L"); break;
case 'M': outtext("M"); break;
case 'N': outtext("N"); break;
case 'O': outtext("O"); break;
case 'P': outtext("P"); break;
case 'Q': outtext("Q"); break;
case 'R': outtext("R"); break;
case 'S': outtext("S"); break;
case 'T': outtext("T"); break;
case 'U': outtext("U"); break;
case 'V': outtext("V"); break;
case 'W': outtext("W"); break;
case 'X': outtext("X"); break;
case 'Y': outtext("Y"); break;
case 'Z': outtext("Z"); break;
case '~': outtext("~"); break;
case '!': outtext("!"); break;
case '@': outtext("@"); break;
case '#': outtext("#"); break;
case '$': outtext("$"); break;
case '%': outtext("%"); break;
case '^': outtext("^"); break;
case '&': outtext("&"); break;
case '*': outtext("*"); break;
```

```
   case '(': outtext("("); break;
   case ')': outtext(")"); break;
   case '_': outtext("_"); break;
   case '+': outtext("+"); break;
   case '|': outtext("|"); break;
   case '`': outtext("`"); break;
   case '-': outtext("-"); break;
   case '=': outtext("="); break;
   case '[': outtext("["); break;
   case ']': outtext("]"); break;
   case ';': outtext(";"); break;
   case ',': outtext(","); break;
   case '.': outtext("."); break;
   case '/': outtext("/"); break;
   case '{': outtext("{"); break;
   case '}': outtext("}"); break;
   case ':': outtext(":"); break;
   case '<': outtext("<"); break;
   case '>': outtext(">"); break;
   case '?': outtext("?"); break;
   case ' ': outtext(" "); break;


   }
}


char* gscan ()
{
     char *test,*test1;
     test1 = test;
     while ((*test = getch()) != 13) gdisplay(*test++);
     return test1;
}

char* gscanxy (int x, int y)
{
     moveto(x,y);
     char *test,*test1;
     test1 = test;
     while ((*test = getch()) != 13) gdisplay(*test++);
     *test++=0;
     return test1;
}

int giscanxy (int x, int y,int retint)
{
     moveto(x,y);
     char *test,counter=0;
     retint=0;
     char upcounter = 0;
     while ((*test = getch()) != 13)
     {gdisplay(*test++);counter++;}
     while (counter-- > 0)
     {
          retint +=(*(--test) - 48) * pow10(upcounter++);
     }
     return retint;
}

int giscan (int retint)
{

     char *test,counter=0;
     retint=0;
     char upcounter = 0;
     while ((*test = getch()) != 13)
     {gdisplay(*test++);counter++;}

     while (counter-- > 0)
     {
          retint +=(*(--test) - 48) * pow10(upcounter++);
     }
     return retint;
}


void gprint (char *tet)
{
     while (*tet != 0) gdisplay(*tet++);
}

void gprintxy (char *tet, int x, int y)
{
     moveto(x,y);
     while (*tet != 0) gdisplay(*tet++);
}


void *save;

#define PIXEL_COUNT 640

int Mx, My;
unsigned E=Mouse.Event(Mx,My);

int  BCOLOR=0,DISP=0,OFFSET=100,EXP=1,GRID=50,PCOLOR=15;
int  VERT=1;
int  GR=0,SHIFT=1,SHIFTF=320,handle,porta,portb,portc;
int  filter=100,SKIP=1;
int  stopvalue=0;
int  trigval=0;
int  OFFSET2=200;
int  fil = 1000;
float clk = 20;
int  ch_1=1;
char filename;
```

```
FILE *fp;
char VAL[8192];
char VAL1[8192];
char TEMP[8192];
float     SAMPLE=20;
float     IN_SCALE;
float     IN_SC2;
void      *buf;
float     FREQ;
int       second = 0;
char      monster[4];
char      inset[4];
float     xval1=400,xval2=400,xval3=400,yval1=400,yval2=400;
float     yval3=400;
int       calc = 0,calcval = 0,man = 0,inter = 0,exter = 0;
int       AC1=1,DC1=4;
int       chan2on=1;
int       AREG;
#include "read1.inc"
#include "read2.inc"
#include "read_eprom.inc"

void      initialize_graphics ( void )
{
int       gdriver = DETECT, gmode, errorcode;

     initgraph(&gdriver, &gmode, "");
     errorcode = graphresult();
     if (errorcode != grOk)
     {
       printf("Graphics error: %s\n",
       grapherrormsg(errorcode));
       printf("press any key to halt:");
       getch();
       exit (1);
     }
}

void change_drive   ( void )
{
     clearviewport_ ( );
     setviewport(2,40,638,451,2);
     setfillstyle(1,1);
     int poly[8];
     poly[0]=10;
     poly[1]=10;
     poly[2]=200;
     poly[3]=10;
     poly[4]=200;
     poly[5]=35;
     poly[6]=10;
     poly[7]=35;
     fillpoly(4,poly);
     outtextxy (20,20,"ENTER DRIVE: ( A - F )");
     Mouse.Show ( );

          switch ( getche( ) )
          {
          case 'a':
          case 'A':setdisk ( 0 );
               break;
          case 'b':
          case 'B':setdisk ( 1 );
               break;
          case 'c':
          case 'C':setdisk ( 2 );
               break;
          case 'd':
          case 'D':setdisk ( 3 );
               break;
          case 'e':
          case 'E':setdisk ( 4 );
               break;
          case 'f':
          case 'F':setdisk ( 5 );
               break;

          }
}



void      file_dir  ( void )
{

struct         ffblk ffblk;

int  done,disk,counter=0;
     clearviewport_ ( );
     setviewport(2,40,638,451,2);
     change_drive ( );
     clearviewport_ ( );
     setviewport(0,40,638,451,0);
     done = findfirst ("*.dat",&ffblk,0);
     while (!done)
     {
          gprintxy(ffblk.ff_name,20,15*counter++);
          done = findnext ( &ffblk );

     }

          Mouse.Show ( );
          gotoxy (18,10);
          printf ("  ENTER FILENAME:  ");
          line (0,452,0,28);
          scanf ("%s", &filename);

          setviewport(2,40,638,451,0);
```

```
}
void file_store     ( void )
{
     clearviewport_ ( );
     setviewport(2,40,638,451,2);
     change_drive ( );
     Mouse.Show ( );
     clearviewport_ ( );
     gotoxy(18,10);
     printf(" ENTER FILENAME:  ");
     scanf ("%s", &filename);
     Mouse.Show ( );
}

void            FILTER      ( void )
{
     int j;
     for (j=0; j!=8192;j++)
     {
       if (abs(VAL[j+1]-VAL[j])>filter)

       VAL[j+1] = (VAL[j]+VAL[j+2])*0.5;

       else

       VAL[j+1] = VAL[j+1];

     }
     int k;
     if (chan2on == 1)
     {
     for (k=0; k!=8192;k++)
     {
       if (abs(VAL1[k+1]-VAL1[k])>filter)

       VAL1[k+1] = (VAL1[k]+VAL1[k+2])*0.5;

       else

       VAL1[k+1] = VAL1[k+1];

     }
     }
}

void FILTER_ (void )
{
     int k;
     if (filter>=stopvalue)
          {
               for (k=filter; k!=stopvalue;k--)
```

```
                    {
                         FILTER ( );
                         filter = filter - 1;
                    }
          }
}

void      draw      ( void )
{
int i,j,k;
int LINE;
     setbkcolor(BCOLOR);

     for (i=0; i<=PIXEL_COUNT;i++)

     {
          LINE = DISP;    /* "0" = dots, "1" = line */
          if (LINE != 1)
          {
               while (abs(VAL[i+1]-VAL[i])>fil)
               {
                if (i==639)
                 {
                  break;
                 }
                i++;
               }
               putpixel(i*EXP, OFFSET+(VAL[i+SHIFT] &
               0x7f)*VERT,PCOLOR );


          }
          else
          {
               while (abs(VAL[i+1]-VAL[i])>fil)
               {
               if (i==639)
               {
               break;
               }
               i=i+1;
               }
               lineto(i*EXP, OFFSET+(VAL[i+SHIFT] &
               0x7f)*VERT);


          }


     }
     if( chan2on == 1 )
     {

     moveto(0,0);
```

```
    for (j=0; j!=PIXEL_COUNT;j++)
    {
        LINE = DISP;    /* "0" = dots, "1" = line */
        if (LINE != 1)
        {
            while (abs(VAL[j+1]-VAL[j])>fil)
            {
            if (j==639)
            {
            break;
            }
            j++;
            }
            putpixel(j*EXP, OFFSET2+(VAL1[j+SHIFT] &
            0x7f)*VERT,PCOLOR );

        }
        else
        {

            while (abs(VAL[j+1]-VAL[j])>fil)
            {
            if (j==639)
            {
            break;
            }
            j++;
            }
            lineto(j*EXP, OFFSET2+(VAL1[j+SHIFT] &
            0x7f)*VERT);

        }

    }
    }
    for (k=0; k!=PIXEL_COUNT;k++)
          {
            putpixel(k*EXP, OFFSET+(64*VERT), 1 );
            putpixel(k*EXP, OFFSET2+(64*VERT), 1 );
          }
}

void chan1      ( void )
{
    portc = (portc & 0xf7);
    outport(0x302,portc);

}
void chan2      ( void )
{
    portc = (portc | 0x08);
    outport(0x302,portc);
}
```

```
void  calculations  ( void )
{
float volt,frek,time,phase=0,time1=0;
               if (calcval == 1)
               {

               setviewport(2,30,225,125,1);
               clearviewport_ ( );
               volt = (( yval1 - yval2)*0.039370078)/VERT;

               if ((xval2 - xval1)!=0)
               {
               frek =(1/( xval2 - xval1 )*SAMPLE*1000)*EXP;
                              //16000
               }


          time =(( xval2 - xval1 )*1/SAMPLE)/EXP;
          time1 =(( xval3 - xval1 )*1/SAMPLE)/EXP;
          phase = ((( xval3 -xval1)*1/SAMPLE)/EXP)/time*360;

                    char string[25];

                    int sig = 5;
                    char *VOLT = "VOLTAGE: ";
                    char *VO = " V";
                    char *FR = "FREQ: ";
                    char *FRE = " kHz : ";
                    char *TI = "TIME: ";
                    char *TIM = " us";
                    char *PH = "PHASE: ";
                    char *DEG = " DEG";
                    char *VS = "INPUT RANGE CH1: ";
                    char *VS2 = "INPUT RANGE CH2: ";
                    char *SR = "SAMPLING RATE: ";
                    char *ME = " MHz";

                    gcvt(volt,sig,string);
                    moveto(10,10);
                    gprint (VOLT);
                    gprint (string);
                    gprint (VO);
                    gcvt(frek,sig,string);
                    moveto(10,22);
                    gprint (FR);
                    gprint (string);
                    gprint (FRE);
                    sig=4;
                    gcvt(time,sig,string);
                    gprint (string);
                    gprint (TIM);
                    gcvt(time1,sig,string);
                    moveto(10,34);
                    gprint (TI);
                    gprint (string);
                    gprint (TIM);
                    moveto(10,46);
                    sig=3;
                    gcvt(phase,sig,string);
                    gprint (PH);
                    gprint (string);
                    gprint (DEG);
                    moveto(10,58);
                    sig=5;
                    gcvt(IN_SCALE,sig,string);
                    gprint (VS);
                    gprint (string);
                    gprint (VO);
                    moveto(10,58);
                    if (chan2on == 1)
                    {
                         moverel(0,12);
                         gcvt(IN_SC2,sig,string);
                         gprint (VS2);
                         gprint (string);
                         gprint (VO);
                         moveto(10,70);
                    }
                    moverel(0,12);
                    gcvt(SAMPLE,sig,string);
                    gprint (SR);
                    gprint (string);
                    gprint (ME);
                    setviewport(2,40,638,451,0);
                    Mouse.Show ( );

                    }
}


void display        ( void )
{
     draw( );
     calculations( );
     Mouse.Show ( );

}

int trigtest ( void )
{
     delay ( 200 );
     int t=0;
     while (bioskey(1)==0)
     {


               for (t=0; t != 8192;t++)
```

```
                     {

                          if ( ( VAL[t] & 0x80 ) != 0  )
                               {
                                    t=0;
                                    //chan1 ( );
                                    //read1 ( );
                                    //chan2 ( );
                                    //read2 ( );
                                    read_eprom ( );
                                    return 1;
                               }

                     }



               t=0;
               clearviewport_ ( );
               Mouse.Show ( );
               setfillstyle(1,1);
               int poly[8];
               poly[0]=10;
               poly[1]=10;
               poly[2]=240;
               poly[3]=10;
               poly[4]=240;
               poly[5]=35;
               poly[6]=10;
               poly[7]=35;
               fillpoly(4,poly);
               outtextxy (20,20,"WAITING FOR TRIGGER");
               delay (2000);
               t=0;
                //  chan1 ( );
                //  read1 ( );
                //  chan2 ( );
                //  read2 ( );
                read_eprom ( );
          }
               return 0;


}


void trigger_level ( void )
{
          gotoxy(10,10);
          printf("ENTER TRIGGER VALUE: ");
          scanf("%d",&trigval);
}
void inttrig   ( void )
{
     outport(0x300,trigval);
     portc = (portc | 0x20);  //set man
     outport(0x302,portc);
     portc = (portc | 0x80);  //set reset
     portc = (portc & 0x0ef); // reset inter
     portc = (portc | 0x40);  //set cont
     outport(0x302,portc);
     portc = (portc & 0x0df); //reset man
     portc = (portc & 0x7f);  // reset reset
     outport(0x302,portc);
     delay (200);

}

void external ( void )
{
     portc = (portc | 0x20);  //set man
     outport(0x302,portc);
     portc = (portc | 0x80);  //set reset
     portc = (portc | 0x40);  // set cont
     portc = (portc | 0x10);  // set inter
     outport(0x302,portc);
     portc = (portc & 0x7f);  //reset reset
     portc = (portc & 0xdf);  //reset man
     outport(0x302,portc);

     chan1 ( );
     read1 ( );
     if (chan2on == 1)
     {
     chan2 ( );
     read2 ( );
     }
     trigtest ( );
     clearviewport_ ( );
     display ( );
     Mouse.Show ( );
}


void manual ( void )
{
     portc = (portc & 0xbf);  //reset cont
     portc = (portc & 0xdf);  //reset man
     outport(0x302,portc);
     delay (500);
     portc = (portc | 0x20);  //set man
     outport(0x302,portc);

}


void vscale         ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     setviewport(0,0,638,451,0);
     setfillstyle(1,2);
     int poly[8];
     poly[0]=96;
     poly[1]=40;
     poly[2]=150;
     poly[3]=40;
     poly[4]=150;
     poly[5]=165;
     poly[6]=96;
     poly[7]=165;
     fillpoly(4,poly);
     outtextxy (100,50,"10.0 v");
     outtextxy (100,75,"5.00 v");
     outtextxy (100,100,"1.00 v");
     outtextxy (100,125,"      ");
     outtextxy (100,150,"      ");

     line (96,65,150,65);
     line (96,90,150,90);
     line (96,115,150,115);
     line (96,140,150,140);

     setfillstyle(1,1);

     poly[0]=216;
     poly[1]=210;
     poly[2]=400;
     poly[3]=210;
     poly[4]=400;
     poly[5]=237;
     poly[6]=216;
     poly[7]=237;
     fillpoly(4,poly);
     outtextxy (226,220,"SELECT VOLTAGE SCALE: ");
     setviewport(2,40,638,451,2);
     Mouse.Show ( );

}
void sscale         ( void )
{

     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     setviewport(0,0,638,451,0);
     setfillstyle(1,2);
     int poly[8];
     poly[0]=96;
     poly[1]=40;
     poly[2]=176;
     poly[3]=40;
     poly[4]=176;
     poly[5]=240;
     poly[6]=96;
     poly[7]=240;
     fillpoly(4,poly);
     outtextxy (100,50,"20.00 MHz");
     outtextxy (100,75,"10.00 MHz");
     outtextxy (100,100,"5.000 MHz");
     outtextxy (100,125,"2.500 MHz");
     outtextxy (100,150,"1.250 MHz");
     outtextxy (100,175,"625.0 MHz");
     outtextxy (100,200,"312.5 kHz");
     outtextxy (100,225,"156.2 kHz");


     line (96,65,176,65);
     line (96,90,176,90);
     line (96,115,176,115);
     line (96,140,176,140);
     line (96,165,176,165);
     line (96,190,176,190);
     line (96,215,176,215);
     setfillstyle(1,1);

     poly[0]=216;
     poly[1]=210;
     poly[2]=400;
     poly[3]=210;
     poly[4]=400;
     poly[5]=237;
     poly[6]=216;
     poly[7]=237;
     fillpoly(4,poly);
     outtextxy (226,220,"SELECT SAMPLING RATE: ");

     setviewport(2,40,638,451,2);
     Mouse.Show ( );


}
float vrange1 ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0f8) | 0x00);
     outport(0x301,portb);
     return (20.0);
}
float vrange2 ( void )
{

     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0f8) | 0x01);
     outport(0x301,portb);
     return (5.0);

}
float vrange3 ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0f8) | 0x02);
     outport(0x301,portb);
     return (1.0);
}
float vrange4 ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0f8) | 0x03);
     outport(0x301,portb);
     return (0.5);
}
float vrange5 ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0f8) | 0x04);
     outport(0x301,portb);
     return (0.1);
}
float vrange6 ( void )
{
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
     outport(0x303,0x80);
     portb = ((portb & 0x0c7) | 0x0);
     outport(0x301,portb);
     return (20.0);
}
float vrange7 ( void )
{

     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     Mouse.Show ( );
```

```
    outport(0x303,0x80);
    portb = ((portb & 0x0c7) | 0x8);
    outport(0x301,portb);
    return (5.0);

}
float vrange8 ( void )
{
    setviewport(2,40,638,451,2);
    clearviewport_ ( );
    Mouse.Show ( );
    outport(0x303,0x80);
    portb = ((portb & 0x0c7) | 0x10);
    outport(0x301,portb);
    return (1.0);
}
float vrange9 ( void )
{
    setviewport(2,40,638,451,2);
    clearviewport_ ( );
    Mouse.Show ( );
    outport(0x303,0x80);
    portb = ((portb & 0x0c7) | 0x18);
    outport(0x301,portb);
    return (0.5);
}
float vrange10 ( void )
{
    setviewport(2,40,638,451,2);
    clearviewport_ ( );
    Mouse.Show ( );
    outport(0x303,0x80);
    portb = ((portb & 0x0c7) | 0x20);
    outport(0x301,portb);
    return (0.1);
}


float range1   ( void )
{
    outport(0x303,0x80);
    portc = ((portc & 0xf8) | 0x0);
    outport(0x302,portc);
    return (clk);
}
float range2        ( void )
{
    outport(0x303,0x80);
    portc = ((portc & 0xf8) | 0x01);
    outport(0x302,portc);
    return (clk/2);
}
float range3        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x02);
     outport(0x302,portc);
     return (clk/4);
}
float range4        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x03);
     outport(0x302,portc);
     return (clk/8);
}
float range5        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x04);
     outport(0x302,portc);
     return (clk/16);
}
float range6        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x05);
     outport(0x302,portc);
     return (clk/32);
}
float range7        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x06);
     outport(0x302,portc);
     return (clk/64);
}
float range8        ( void )
{
     outport(0x303,0x80);
     portc = ((portc & 0xf8) | 0x07);
     outport(0x302,portc);
     return (clk/128);
}


void      screen ( void )
{
          setbkcolor (0);

}
void      menu1     ( void )
{
          setviewport (5,43,122,240,1);
          clearviewport ( );
          setviewport(2,40,638,451,2);
          setfillstyle(1,4);
          int poly[8];
          poly[0]=3;
          poly[1]=3;
          poly[2]=120;
          poly[3]=3;
          poly[4]=120;
          poly[5]=200;
          poly[6]=3;
          poly[7]=200;
          fillpoly(4,poly);
          outtextxy(10,10,"FILE SAVE");
          outtextxy(10,30,"FILE IMPORT");
          outtextxy(10,50,"<< >> VALUE");




}
void logo      ( void )
{
     setbkcolor(BCOLOR);
     settextstyle(SANS_SERIF_FONT,HORIZ_DIR,8);
     outtextxy (50,160,"PC SCOPE");
     settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
     outtextxy (50,400,"Copyright (c) 1993        HP J van
     RENSBURG");


}

void      menubar ( void )
{
          setfillstyle(1,4);
          int poly[8];
          poly[0]=0;
          poly[1]=0;
          poly[2]=639;
          poly[3]=0;
          poly[4]=639;
          poly[5]=25;
          poly[6]=0;
          poly[7]=25;
          fillpoly(4,poly);


          line (0,28,639,28);
          line (639,28,639,452);
          line (639,452,0,452);
          line (0,452,0,28);


          setfillstyle(1,1);
```

```
bar3d(0,455,32,479,0,0);
outtextxy (5,464,"CH1");
setfillstyle (1,4);
bar3d(32,455,54,479,0,0);
outtextxy (36,464,"AC");
bar3d(54,455,76,479,0,0);
outtextxy (58,464,"DC");
setfillstyle(1,4);
bar3d(76,455,168,479,0,0);
outtextxy (78,464,"INPUT RANGE");
bar3d(168,455,277,479,0,0);
outtextxy (171,464,"TRIGGER LEVEL");

setfillstyle(1,2);
bar3d(277,455,387,479,0,0);
outtextxy (281,464,"SAMPLING RATE");

setfillstyle(1,1);
bar3d(387,455,419,479,0,0);
outtextxy (392,464,"CH2");
setfillstyle ( 1,4 );
bar3d(419,455,441,479,0,0);
outtextxy (423,464,"AC");
bar3d(441,455,463,479,0,0);
outtextxy (445,464,"DC");
bar3d(463,455,556,479,0,0);
outtextxy (466,464,"INPUT RANGE");
setfillstyle ( 1,1 );
bar3d(556,455,578,479,0,0);
outtextxy (560,464,"ON");
setfillstyle ( 1,4 );
bar3d(578,455,608,479,0,0);
outtextxy (582,464,"OFF");
bar3d(608,455,639,479,0,0);
outtextxy (613,464,"CLS");

outtextxy (5,10,"MENU");
outtextxy (45,10,"<<X>>");

line (41,25,41,0);
line (88,25,88,0);
outtextxy (92,10,"<<Y>>");
line (135,25,135,0);
outtextxy (139,10,"<<");
line (158,25,158,0);
outtextxy (162,10,">>");
line (181,25,181,0);
outtextxy (183,10,"...");
line (208,25,208,0);
line (212,15,230,15);
line (234,25,234,0);
outtextxy ( 238,10,"MANUAL");
line (288,25,288,0);
outtextxy (292,10,"INT TRIG");
```

```
          line (358,25,358,0);
          outtextxy (362,10,"EXT TRIG");
          line (428,25,428,0);
          outtextxy (432,10,"REF");
          line (458,25,458,0);
          outtextxy (462,10,"UP");
          line (480,25,480,0);
          outtextxy (484,10,"DOWN");
          line (518,25,518,0);
          outtextxy (522,10,"FILTER");
          line (572,25,572,0);
          outtextxy (576,10,"???");
          line (602,25,602,0);
          outtextxy (606,10,"QUIT");


}



int       main  (void)
{
int  escape= 0,submenu= 0;
     initialize_graphics( );
     outport(0x303,0x80);
     screen ( );
     menubar ( );
     logo ( );
     delay ( 2000 );
     setviewport(2,40,638,451,2);
     clearviewport ( );
     setviewport (0,0,640,480,1);

     Mouse.Setup(Graphics);
     if (!Mouse.Operating()) {
          closegraph();
          cprintf("Mouse not detected.\n");
          exit(1);
     }
     Mouse.SetGCursor(ArrowCursor);

          while ( !escape )
      //  while ( !escape || E !=RMouseDown)
     {
          E=Mouse.Event(Mx,My);


          char ch=' ';
          if (bioskey(1)) ch = bioskey(0);

          switch    ( ch)
          {
               case 'q':
```

```
     case 'Q':escape = 1;
           cleardevice ( );
           break;
     case 'f':
     case 'F': gotoxy(20,8);
            printf ("ENTER FILTER VALUE:  ");
            scanf("%d",&fil);
            break;
     case 'c': gotoxy(20,8);
            printf ("ENTER CLOCK FREQ:  ");
            scanf("%f",&clk);
            break;


}
switch (E) {
     case LMouseDown:
          if (My>=0 && My<25)  //first line
               {
               if (Mx >=0 && Mx<40)
               {
                    sound(30);
                    delay(10);
                    nosound( );
                    calcval=0;
                    calc=2;

                    setviewport(2,40,638,451,2);
                    //menu
                    menu1();
                    submenu=1;
                    break;
               }


               if (Mx >=603 && Mx<639)
               {    sound(30);
                    delay(10);
                    nosound( );
                    setviewport(2,40,638,451,2);
                    clearviewport_();
                    Mouse.Show ( );
                    escape = 1;
                    break;
               }
               if (Mx >=42 && Mx<87)
               {    sound(30);
                    delay(10);
                    nosound( );
                    calcval=0;
                    calc=2;
                    char *test;
                    setviewport(2,40,638,451,2);
                              //x
     clearviewport_ ( );
     setfillstyle(1,1);
     int poly[8];
     poly[0]=10;
     poly[1]=10;
     poly[2]=250;
     poly[3]=10;
     poly[4]=250;
     poly[5]=35;
     poly[6]=10;
     poly[7]=35;
     fillpoly(4,poly);
     outtextxy (20,20,"SELECT
     HORIZONTAL SCALE: ");
     test=gscanxy(230,20 );
     EXP=atoi(test);
     clearviewport_ ( );
     setviewport(2,40,638,451,2);
     display ( );
     Mouse.Show ( );
     break;
}
if (Mx >=89 && Mx<134)
{    sound(30);
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     char *test;
     setviewport(2,40,638,451,2);
     //Y
     clearviewport_ ( );
     setfillstyle(1,1);
     int poly[8];
     poly[0]=10;
     poly[1]=10;
     poly[2]=240;
     poly[3]=10;
     poly[4]=240;
     poly[5]=35;
     poly[6]=10;
     poly[7]=35;
     fillpoly(4,poly);
     outtextxy (20,20,"SELECT
     VERTICAL SCALE: ");
     test=gscanxy(220,20);
     VERT=atoi(test);
     clearviewport_ ( );
     setviewport(2,40,638,451,2);
     display ( );
     Mouse.Show ( );
     break;
}
if (Mx >=136 &&Mx<157)
```

```
{    sound(30);
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     setviewport(2,40,638,451,2);
     //backward
     clearviewport_ ( );
     SHIFT =( SHIFT - SHIFTF );
     display ( );
     Mouse.Show ( );
     break;
}
if (Mx >=159 &&Mx<180)
{
     sound(30);
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     setviewport(2,40,638,451,2);
     //forward
     clearviewport_ ( );
     SHIFT =( SHIFT + SHIFTF );
     display ( );
     Mouse.Show ( );
     break;
}
if (Mx >=182 &&Mx<207)
{
     sound(30);
     delay(10);
     nosound( );
     setviewport(2,40,638,451,2);
     //dot
     clearviewport_ ( );
     DISP = 0;
     display ( );
     Mouse.Show ( );
     break;
}
if (Mx >=209 &&Mx<233)
{
     sound(30);
     delay(10);
     nosound( );
     setviewport(2,40,638,451,2);
     //line
     clearviewport_ ( );
     DISP = 1;
     display ( );
     Mouse.Show ( );
     break;
}
```

```
if (Mx >=235 &&Mx<287)
{
     sound(30);
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     setviewport(2,40,638,451,2);
     //manual
     clearviewport_ ( );
     manual ( );
     chan1 ( );
     read1 ( );
     if (chan2on == 1)
     {
       chan2 ( );
       read2 ( );
     }
     display ( );
     man=0;
     portc = (portc & 0xbf);
     //reset cont
     portc = (portc & 0xdf);
     //reset man
     outport(0x302,portc);
     Mouse.Show ( );
     break;
}
if (Mx >=289 &&Mx<357)
{
     sound(30);
     delay(10);
     nosound ( );
     calcval=0;
     calc=2;
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     inttrig ( );
     chan1 ( );
     read1 ( );
     if (chan2on == 1)
     {
        chan2 ( );
        read2 ( );
     }
     trigtest ( );
     clearviewport_ ( );
     display ( );
     Mouse.Show ( );
     inter = 0;
     Mouse.Show ( );
     break;                    //internal
}
if (Mx >=359 &&Mx<427)
```

```
{
    sound(30);
    delay(10);
    nosound( );
    calcval=0;
    calc=2;
    setviewport(2,40,638,451,2);
    clearviewport_ ( );
    external ( );
    exter=0;
    Mouse.Show ( );
    break;              //external
}
if (Mx >=429 &&Mx<457)
{
    sound(30);
    delay(10);
    nosound( );
    calcval=0;
    calc=2;
    setviewport (xval1,yval1-
    5,xval1,yval1+5,1);
    clearviewport ( );
    setviewport (xval1-
    5,yval1,xval1+5,yval1,1);
    clearviewport ( );
    setviewport(2,40,638,451,2);
    display ( );
    calc = 1;     //ref
    Mouse.Show ( );
    break;
}

if (Mx >=459 &&Mx<479)
{
    sound(30);      //up
    delay(10);
    nosound( );
    calcval=0;
    calc=2;
    if (ch_1 == 1)
    {
       OFFSET = (OFFSET - 10);
    }
    else
    {
       OFFSET2 = (OFFSET2 - 10);
    }
    setviewport(2,40,638,451,2);
    clearviewport_ ( );
    display ( );
    Mouse.Show ( );
    setviewport(2,40,638,451,2);
    break;
```

```
}
if (Mx >=481 &&Mx<517)
{
     sound(30);    //down
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     if (ch_1 == 1)
     {
        OFFSET = (OFFSET + 10);
     }
     else
     {
        OFFSET2 = (OFFSET2 + 10);
     }
     setviewport(2,40,638,451,2);
     clearviewport_ ( );
     display ( );
     Mouse.Show ( );
     setviewport(2,40,638,451,2);
     break;
}
if (Mx >=519 &&Mx<571)
{
     sound(30);
     delay(10);
     nosound( );
     calcval=0;
     calc=2;
     char *test;
     setfillstyle(1,1);
     int poly[8];
     poly[0]=10;
     poly[1]=10;
     poly[2]=240;
     poly[3]=10;
     poly[4]=240;
     poly[5]=35;
     poly[6]=10;
     poly[7]=35;
     fillpoly(4,poly);
     outtextxy (20,20,"ENTER START
     VALUE: ");
     test=gscanxy(200,20);
     filter=atoi(test);
     fillpoly(4,poly);
     outtextxy (20,20,"ENTER STOP
     VALUE: ");
     test=gscanxy(200,20);
     stopvalue=atoi(test);
     fillpoly(4,poly);
     outtextxy (20,20,"PLEASE
     WAIT");
```

```
          FILTER_ ( );
          setviewport(2,40,638,451,2);
          clearviewport_ ( );
          display ( );
          setviewport(2,40,638,451,2);
          Mouse.Show ( );
          break;
     }

}
if (My>=455 && My<480)
{
     if (Mx >=1 && Mx<32)
     {
          ch_1 = 1;
          break;
     }
     if (Mx >=33 &&Mx<53)
     {

          setviewport(32,455,54,479,2);
          clearviewport_( );
          setviewport(0,0,639,479,2);
          setfillstyle(1,1);
          bar3d(32,455,54,479,0,0);
          outtextxy(36,464,"AC");
          setviewport(54,455,76,479,2);
          Mouse.Show( );
          clearviewport_( );
          setviewport(0,0,639,479,2);
          setfillstyle(1,4);
          bar3d(54,455,76,479,0,0);
          outtextxy(58,464,"DC");
          setviewport(2,40,638,451,2);
          Mouse.Show( );
          portb = (portb | 0x40);
          //set ac ch1
          outport(0x301,portb);
          break;
      }
     if (Mx >=55 &&Mx<75)
      {


          setviewport(54,455,76,479,2);
          clearviewport_( );
          setviewport(0,0,639,479,2);
          setfillstyle(1,1);
          bar3d(54,455,76,479,0,0);
          outtextxy(58,464,"DC");
          setviewport(32,455,54,479,2);
          Mouse.Show( );
          clearviewport_( );
          setviewport(0,0,639,479,2);
     setfillstyle(1,4);
     bar3d(32,455,54,479,0,0);
     outtextxy(36,464,"AC");
     setviewport(2,40,638,451,2);
     Mouse.Show( );
     portb = (portb & 0xbf);
     //reset ac ch1
     outport(0x301,portb);
     break;
 }
if (Mx >=77 &&Mx<167)
 {
     man=2;
     vscale ( );
     break;
 }
if (Mx >=169 &&Mx<276)
     //triggerlevel
 {
     setviewport (2,40,638,451,2);
     clearviewport_( );
     trigger_level( );
     setviewport (2,40,638,451,2);
     clearviewport( );
     Mouse.Show ( );
     break;

 }
if (Mx >=278 &&Mx<386)  //sampling
 {
     man = 1;
     sscale ( );
     break;
 }
if (Mx >=387 &&Mx<420)
 {
     ch_1 = 0;
     break;
 }
if (Mx >=420 &&Mx<440)
{

setviewport(419,455,441,479,2);
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,1);
bar3d(419,455,441,479,0,0);
outtextxy(423,464,"AC");
setviewport(441,455,463,479,2);
Mouse.Show( );
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,4);
bar3d(441,455,463,479,0,0);
```

```
outtextxy(445,464,"DC");
setviewport(2,40,638,451,2);
Mouse.Show( );
portb = (portb | 0x80);
//set ac ch2
outport(0x301,portb);
break;

}
if (Mx >=442 &&Mx<462)
{
setviewport(441,455,463,479,2);
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,1);
bar3d(441,455,463,479,0,0);
outtextxy(445,464,"DC");
setviewport(419,455,441,479,2);
Mouse.Show( );
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,4);
bar3d(419,455,441,479,0,0);
outtextxy(423,464,"AC");
setviewport(2,40,638,451,2);
Mouse.Show( );
portb = (portb & 0x7f);
//reset ac ch2
outport(0x301,portb);
break;

}
if (Mx >=464 &&Mx<555)
{
man=3;
vscale ( );
break;
}
if (Mx >=557 &&Mx<577)
{
setviewport(556,455,578,479,2);
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,1);
bar3d(556,455,578,479,0,0);
outtextxy(560,464,"ON");
setviewport(578,455,608,479,2);
Mouse.Show( );
clearviewport_( );
setviewport(0,0,639,479,2);
setfillstyle(1,4);
bar3d(578,455,608,479,0,0);
outtextxy(582,464,"OFF");
setviewport(2,40,638,451,2);
```

```
     chan2on = 1;
     display ( );
     Mouse.Show( );
     break;
     }
     if (Mx >=579 &&Mx<607)
     {
     setviewport(556,455,578,479,2);
     clearviewport_( );
     setviewport(0,0,639,479,2);
     setfillstyle(1,4);
     bar3d(556,455,578,479,0,0);
     outtextxy(560,464,"ON");
     setviewport(578,455,608,479,2);
     Mouse.Show( );
     clearviewport_( );
     setviewport(0,0,639,479,2);
     setfillstyle(1,1);
     bar3d(578,455,608,479,0,0);
     outtextxy(582,464,"OFF");
     setviewport(2,40,638,451,2);
     Mouse.Show ( );
     clearviewport_ ( );
     chan2on = 0;
     display ( );
     Mouse.Show ( );
     break;
     }
     if (Mx >=609 &&Mx<638)
     {
     setviewport(2,40,638,451,2);
     clearviewport_( );
     Mouse.Show( );
     break;
     }
}

if (Mx>=0 && Mx<100)  //file save
{
if (My >=42 && My<63)
{
if (submenu == 1)
{
sound(30);
delay(10);
nosound( );
file_store ( );
if ((fp = fopen(&filename,"wb"))==NULL)
{
printf("cannot open file");
exit(1);
}
fwrite(&SAMPLE,sizeof(float),1,fp);
fwrite(&IN_SCALE,sizeof(float),1,fp);
```

```
fwrite (&IN_SC2,sizeof(float),1,fp);
fwrite (VAL,sizeof VAL,1,fp);
fwrite (VAL1,sizeof VAL1,1,fp);
fclose(fp);
clearviewport_ ( );
setviewport(2,40,638,451,2);
submenu = 0;
Mouse.Show ( );
break;
}
}
if (My >=65 && My<85)   //file import
{
if (submenu == 1)
{
sound(30);
delay(10);
nosound( );
file_dir( );
if((fp = fopen(&filename, "rb"))==NULL)
{
printf("cannot open file");
exit (1);
}

fread(&SAMPLE,sizeof(float),1,fp);
fread(&IN_SCALE,sizeof(float),1,fp);
fread(&IN_SC2,sizeof(float),1,fp);
fread(VAL,sizeof VAL,1,fp);
fread(VAL1,sizeof VAL1,1,fp);
fclose (fp);
clearviewport_ ( );
setviewport(2,40,638,451,2);
display ( );
submenu = 0;
Mouse.Show ( );
break;
}
}

if (My >=87 && My<97)
{
if (submenu == 1)
{
sound(30);
delay(10);
nosound( );
char *test;
setviewport(2,40,638,451,2);
clearviewport_ ( );
setfillstyle(1,1);
int poly[8];
poly[0]=10;
poly[1]=10;
```

```
    poly[2]=300;
    poly[3]=10;
    poly[4]=300;
    poly[5]=35;
    poly[6]=10;
poly[7]=35;
fillpoly(4,poly);
outtextxy(20,20,"SELECT SHIFT FACTOR: ");
test=gscanxy(200,20 );
SHIFTF=atoi(test);
setviewport(2,40,638,451,2);
clearviewport_ ( );
display ( );
Mouse.Show ( );
break;
}
}

if (Mx >=361 && Mx<440)
{
cleardevice ( );
break;
}
if (Mx >=441 && Mx<540)
{
cleardevice ( );
break;
}
if (Mx >=541 && Mx<640)
{
cleardevice ( );
break;
}
}

if (My>=40 && My<451)  //calculations
{
if (Mx >=2 && Mx<638)
{
if (calc == 1)
{
sound(30);
delay(10);
nosound( );
setviewport(0,0,638,451,2);
xval1 = Mx;
yval1 = My;
calc = 0;
Mouse.Hide ( );
setcolor (14);
line (xval1,yval1-5,xval1,yval1+5);
line (xval1-5,yval1,xval1+5,yval1);
setcolor (15);
Mouse.Show ( );
```

```
setviewport(2,40,638,451,2);
break;
}
}
if (Mx >=2 && Mx<638)
{
if (calc==0&& man==0 && inter==0 && exter==0)
{
sound(30);
delay(10);
nosound( );
setviewport (xval2,yval2-5,xval2,yval2+5,1);
clearviewport_ ( );
setviewport (xval2-5,yval2,xval2+5,yval2,1);
clearviewport_ ( );
Mouse.Show ( );
setviewport (0,0,638,451,1);
xval2 = Mx;
yval2 = My;
calcval = 1;
Mouse.Hide ( );
setcolor (14);
line (xval2,yval2-5,xval2,yval2+5);
line (xval2-5,yval2,xval2+5,yval2);
setcolor (15);
setviewport(2,40,638,451,2);
display ( );
Mouse.Show ( );
break;
}
}
}
if (Mx >=96 && Mx<176)
{
if (My >=40 && My<65)
{
if ( man == 1 )
{
SAMPLE = range1( );
outport(0x301,portb);
man = 0;
setviewport(2,40,638,451,2);
clearviewport_( );
Mouse.Show( );
break;
}

}
if (My >=65 && My<90)
{
if ( man == 1 )
{
SAMPLE = range2 ( );
outport(0x301,portb);
```

```
man = 0;
setviewport(2,40,638,451,2);
clearviewport_( );
Mouse.Show( );
break;
}
}
if (My >=90 && My<115)
{
if ( man == 1 )
{
SAMPLE = range3 ( );
outport(0x301,portb);
man = 0;
setviewport(2,40,638,451,2);
clearviewport_( );
Mouse.Show( );
break;
}
}
if (My >=115 && My<140)
{
if ( man == 1 )
{
SAMPLE = range4 ( );
outport(0x301,portb);
man = 0;
setviewport(2,40,638,451,2);
clearviewport_( );
Mouse.Show( );
break;
}
}
if (My >=140 && My<165)
{
if ( man == 1 )
{
SAMPLE = range5 ( );

outport(0x301,portb);
man = 0;
setviewport(2,40,638,451,2);

clearviewport_( );
Mouse.Show( );
break;
}

}
if (My >=165 && My<190)
{
if ( man == 1 )
{
SAMPLE = range6 ( );
```

```
outport(0x301,portb);
man = 0;
setviewport (2,40,638,451,2);

clearviewport_( );
Mouse.Show( );
break;
}
}
if (My >=190 && My<215)
{
if ( man == 1 )
{
SAMPLE = range7 ( );

outport(0x301,portb);
man = 0;
setviewport (2,40,638,451,2);

clearviewport_( );
Mouse.Show( );
break;
}
}
if (My >=215 && My<240)
{
if ( man == 1 )
{
SAMPLE = range8 ( );

outport(0x301,portb);
man = 0;
setviewport (2,40,638,451,2);

clearviewport_( );
Mouse.Show( );
break;
}

}
                    }
if (Mx >=96 && Mx<150)
{
if (My >=40 && My<65)
{
if ( man == 2 )
{

IN_SCALE = vrange1 ( );
man = 0;
break;
}
if ( man == 3 )
```

```
{

IN_SC2 = vrange6 ( );
man = 0;
break;
}
}
if (My >=65 && My<90)
{
if ( man == 2 )
{

IN_SCALE = vrange2 ( );
man = 0;
break;
}
if ( man == 3 )
{

IN_SC2 = vrange7 ( );
man = 0;
break;
}
}
if (My >=90 && My<115)
{
if ( man == 2 )
{

IN_SCALE = vrange3 ( );
man = 0;
break;
}
if ( man == 3 )
{

IN_SC2 = vrange8 ( );
man = 0;
break;
}

}
if (My >=115 && My<140)
{
if ( man == 2 )
{
IN_SCALE = vrange4 ( );
man = 0;
break;
}
if ( man == 3 )
{

IN_SC2 = vrange9 ( );
```

```
     man = 0;
     break;
     }

     }

     if (My >=140 && My<165)
     {
     if ( man == 2 )
     {
     IN_SCALE = vrange5 ( );
     man = 0;
     break;
     }
     if ( man == 3 )
     {

     IN_SC2 = vrange10 ( );
     man = 0;
     break;
     }
     }
     }
     break;


case RMouseDown:  if (My>=40 && My<451)
//phase angle
{
if (Mx >=2 && Mx<638)
{

{
if (calc==0 && man==0 && inter == 0&& exter == 0)
{
sound(30);
delay(10);
nosound( );
setviewport (xval3,yval3-5,xval3,yval3+5,1);
clearviewport_ ( );
setviewport (xval3-5,yval3,xval3+5,yval3,1);
clearviewport_ ( );
Mouse.Show ( );
setviewport (0,0,638,451,1);
xval3 = Mx;
yval3 = My;
calcval = 1;
Mouse.Hide ( );
setcolor (14);
line (xval3,yval3-5,xval3,yval3+5);
line (xval3-5,yval3,xval3+5,yval3);
setcolor (15);
setviewport(2,40,638,451,2);
display ( );
```

```
          Mouse.Show ( );
          break;
          }
          }
          }
          }

          }
     }
     Mouse.TurnOff();
     closegraph( );
     return 0;
}
```

13.2.2. Include files

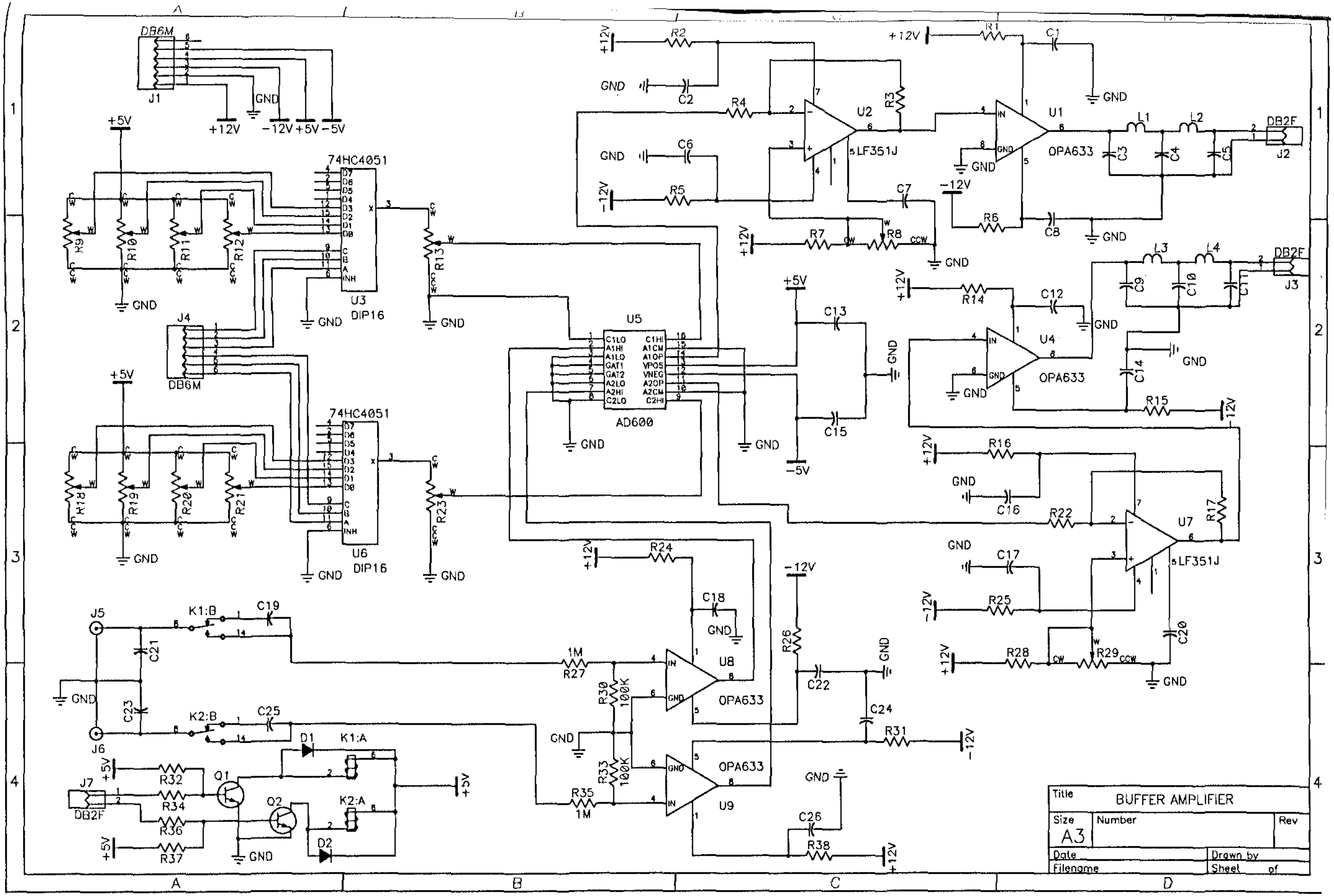
12.2.2 read1.inc

```
void       read1   ( void )
{
     asm{
          PUSH AX
          PUSH BX
          PUSH CX
          PUSH DX
          PUSHF
          PUSH SI
          PUSH DI
          PUSH DS
          MOV  CX,4096
          MOV  SI,00H
         }
JUMPBACK:
     asm{
          MOV  AX,0D000H
          MOV  DS,AX
          MOV  AX,[SI]
          AND  AX,0FFH
          MOV  DX,AX
          POP  DS
          MOV  AREG,DX
          MOV  [VAL+SI],DL
          INC  SI
          PUSH DS
          MOV  AX,0D000H
          MOV  DS,AX
          MOV  AX,[SI]
          AND  AX,0FF00H
          XCHG AL,AH
          MOV  DX,AX
          POP  DS
          MOV  AREG,DX
          MOV  [VAL+SI],DL
          INC  SI
          DEC  CX
          PUSH DS
          JNE  JUMPBACK
          POP  DS
          POP  DI
          POP  SI
          POPF
          POP  DX
          POP  CX
          POP  BX
          POP  AX


     }
}
```

12.2.3. read2.inc

```
void      read2   ( void )
{
     asm{
          PUSH AX
          PUSH BX
          PUSH CX
          PUSH DX
          PUSHF
          PUSH SI
          PUSH DI
          PUSH DS
          MOV  CX,4096
          MOV  SI,00H
         }
JUMPBACK:
     asm{
          MOV  AX,0D000H
          MOV  DS,AX
          MOV  AX,[SI]
          AND  AX,0FFH
          MOV  DX,AX
          POP  DS
          MOV  [VAL1+SI],DL
          INC  SI
          PUSH DS
          MOV  AX,0D000H
          MOV  DS,AX
          MOV  AX,[SI]
          AND  AX,0FF00H
          XCHG AL,AH
          MOV  DX,AX
          POP  DS
          MOV  [VAL1+SI],DL
          INC  SI
          DEC  CX
          PUSH DS
          JNE  JUMPBACK
          POP  DS
          POP  DI
         POP  SI
         POPF
         POP  DX
         POP  CX
         POP  BX
         POP  AX

     }
}
```

13. APPENDIX B

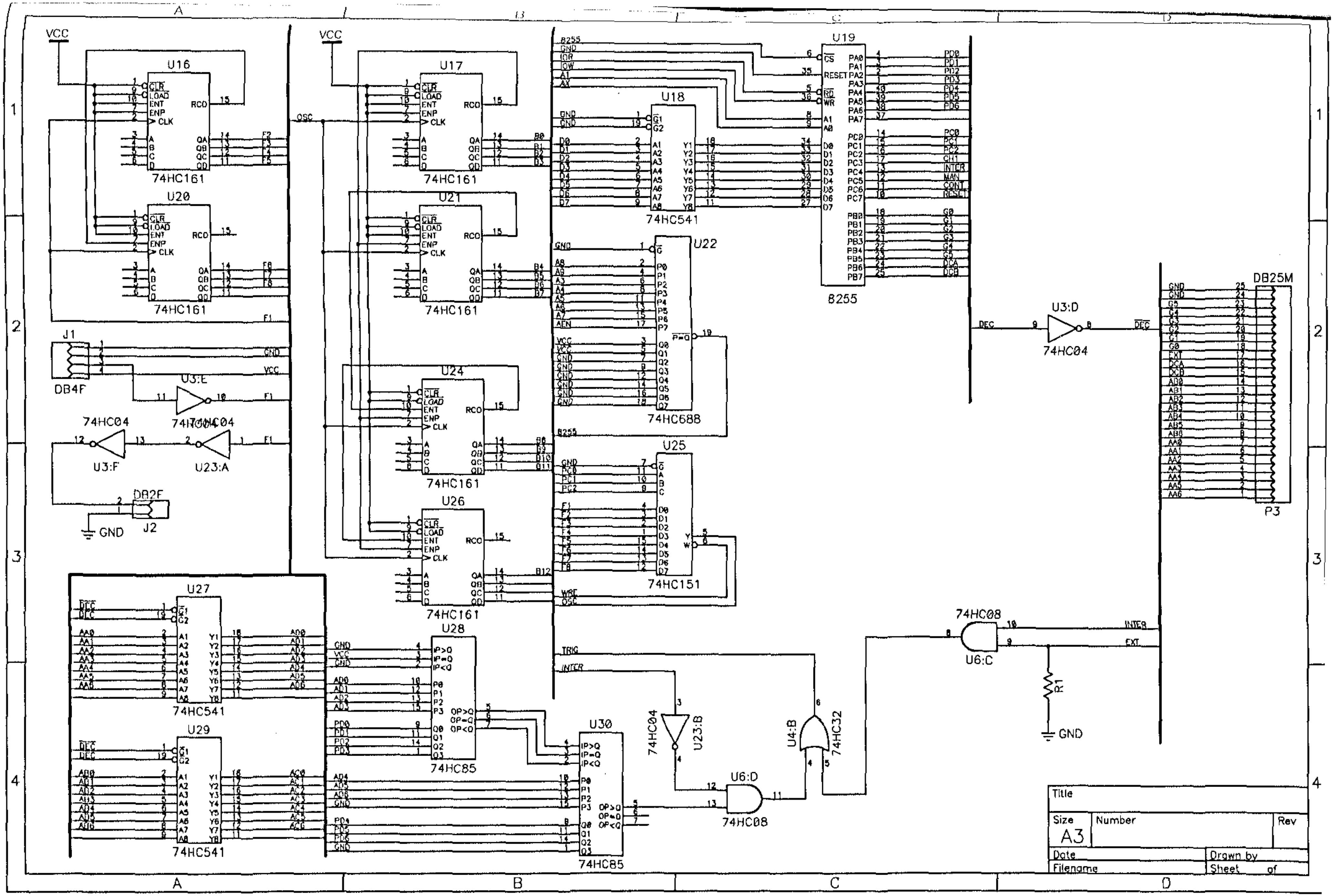


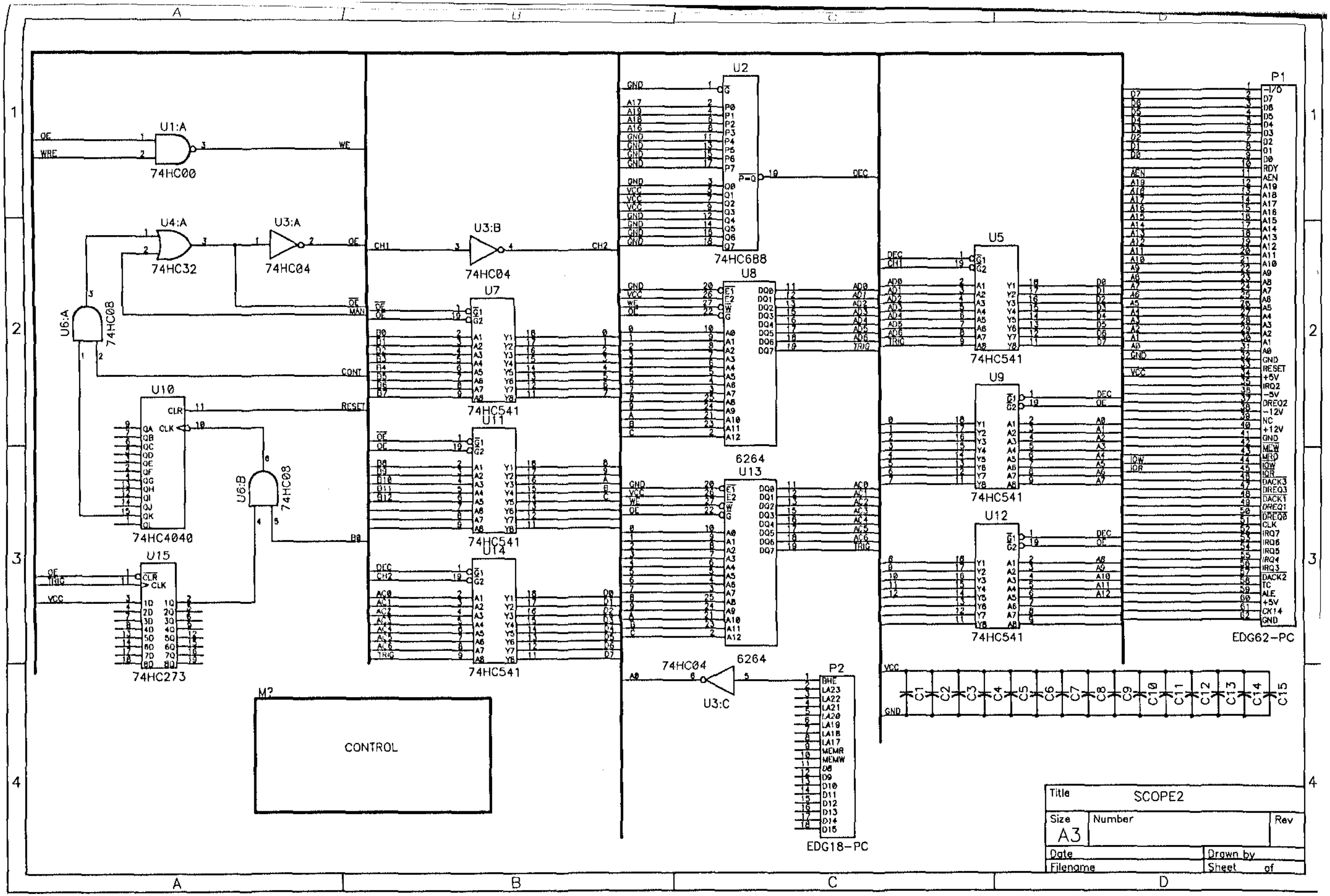
Title BUFFER AMPLIFIER

Size A3 | Number | Rev

Date | Drawn by

Filename | Sheet of


A
B
C
D
1
2
3
4
VCC
U16
74HC161
U20
74HC161
U17
74HC161
U21
74HC161
U24
74HC161
U26
74HC161
CLR
LOAD
ENT
ENP
CLK
RCO
QA
QB
QC
QD
OSC
U19
8255
CS
RESET
RD
WR
A1
A0
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
PD0
PD1
PD2
PD3
PD4
PD5
PD6
G0
G1
G2
G3
G4
G5
DCA
DCB
CH1
INTER
MAN
CONT
RESET
IOR
IOW
AX
AEN
U18
74HC541
U22
74HC688
P=Q
U25
74HC151
WRE
J1
DB4F
GND
VCC
F1
U3:E
74HC04
U3:F
U23:A
DB2F
J2
U3:D
DEC
DB25M
P3
EXT
AD0
AB0
AA0
U27
74HC541
U29
74HC541
U28
74HC85
U30
74HC85
TRIG
INTER
74HC04
U23:B
U4:B
74HC32
U6:D
74HC08
U6:C
R1
Title
Size
A3
Number
Rev
Date
Drawn by
Filename
Sheet
of


U1:A
74HC00
U4:A
74HC32
U3:A
74HC04
U3:B
74HC04
U6:A
74HC08
U6:B
74HC08
U10
74HC4040
U15
74HC273
U7
74HC541
U11
74HC541
U14
74HC541
U2
74HC688
U8
6264
U13
6264
U3:C
74HC04
U5
74HC541
U9
74HC541
U12
74HC541
P1
EDG62-PC
P2
EDG18-PC
C1
C2
C3
C4
C5
C6
C7
C8
C9
C10
C11
C12
C13
C14
C15
M?
CONTROL
OE
WRE
WE
CH1
CH2
CONT
RESET
B0
DEC
TRIG
VCC
GND
MAN
Title SCOPE2
Size A3
Number
Rev
Date
Drawn by
Filename
Sheet of
A
B
C
D
1
2
3
4